IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE CS · 2026 · SET 2

IAE Academy free study material · Updated 27 October 2026

GATE CS 2026 Set 2 Solved Paper

INSIDE

- All 65 questions, in the order of the paper
- The official answer to every question
- A step-by-step solution to every question
- The answer key on one page

Question paper and official answer key: IIT Guwahati

Online, with the questions linked to their topic notes: **iaeacademy.in/study/papers/cs-2026-2**

FREE COUNSELLING

Preparing for GATE CS? Talk to IAE Academy.

Or call **+91 84204 36524**

Every day, 10:30 am to 7:30 pm

Chat on WhatsApp

Free. Not for sale. Pass it on.

About this paper

65 questions, 100 marks. Question types: 35 MCQ · 11 MSQ · 19 NAT.

Official answer key: IIT Guwahati · https://gate2026.iitg.ac.in/doc/download/2026/Keys/CS2_Keys.pdf

We placed each question under a subject and topic ourselves; the exam paper does not label them.

This paper online, with the questions linked to their topic notes:
iaeacademy.in/study/papers/cs-2026-2

Answer key

Q	TYPE	MARKS	ANSWER
Q.1	MCQ	1	**A**
Q.2	MCQ	1	**C**
Q.3	MCQ	1	**A**
Q.4	MCQ	1	**A**
Q.5	MCQ	1	**C**
Q.6	MCQ	2	**A**
Q.7	MCQ	2	**A**
Q.8	MCQ	2	**A**
Q.9	MCQ	2	**B**
Q.10	MCQ	2	**C**
Q.11	MCQ	1	**A**
Q.12	MCQ	1	**A**
Q.13	MCQ	1	**D**
Q.14	MCQ	1	**B**
Q.15	MCQ	1	**C**
Q.16	MCQ	1	**B**
Q.17	MCQ	1	**C**
Q.18	MCQ	1	**C**
Q.19	MCQ	1	**A**
Q.20	MCQ	1	**A**
Q.21	MCQ	1	**B**
Q.22	MCQ	1	**C**
Q.23	MCQ	1	**B**
Q.24	MCQ	1	**A**
Q.25	MSQ	1	**A, B**
Q.26	MSQ	1	**A, B, C**
Q.27	MSQ	1	**A, C**
Q.28	MSQ	1	**B, D**
Q.29	MSQ	1	**B, C**
Q.30	NAT	1	**3**
Q.31	NAT	1	**24**
Q.32	NAT	1	**3**
Q.33	NAT	1	**4094**
Q.34	NAT	1	**-60.25**
Q.35	NAT	1	**13**
Q.36	MCQ	2	**C**
Q.37	MCQ	2	**A**
Q.38	MCQ	2	**A**
Q.39	MCQ	2	**A**
Q.40	MCQ	2	**B**
Q.41	MCQ	2	**D**
Q.42	MCQ	2	**D**
Q.43	MCQ	2	**D**
Q.44	MCQ	2	**B**
Q.45	MCQ	2	**A**
Q.46	MCQ	2	**A**
Q.47	MSQ	2	**C, D**
Q.48	MSQ	2	**A, C, D**
Q.49	MSQ	2	**A, C**
Q.50	MSQ	2	**A, B**
Q.51	MSQ	2	**A, B, C**
Q.52	MSQ	2	**A, B**
Q.53	NAT	2	**16417**
Q.54	NAT	2	**1536**
Q.55	NAT	2	**3**
Q.56	NAT	2	**128**
Q.57	NAT	2	**2.30 to 2.40**
Q.58	NAT	2	**4**
Q.59	NAT	2	**6**
Q.60	NAT	2	**9**
Q.61	NAT	2	**1**
Q.62	NAT	2	**48**
Q.63	NAT	2	**0.5**
Q.64	NAT	2	**40**
Q.65	NAT	2	**30**

General Aptitude Q.1 to Q.10

Q.1 MCQ · 1 MARK

General Aptitude › Grammar & vocabulary

Expedite, Hasten, Hurry, _____

Fill the blank by choosing a word with a meaning similar to that of the words given above.

(A) Accelerate

(B) Retard

(C) Provide

(D) Disable

Official answer: **A**

SOLUTION

1. The given words Expedite, Hasten and Hurry all mean to make something happen faster or to speed it up.
2. We need the option closest in meaning to this group, that is, a synonym of the three words.
3. (A) Accelerate means to increase speed or to make something happen sooner. It matches the group.
4. (B) Retard means to slow down or delay. It is the opposite of the given words, an antonym.
5. (C) Provide means to supply or give. It has no link with speed.
6. (D) Disable means to make unable to function. It has no link with speed either.
7. Only (A) shares the meaning of the given words, so the answer is (A).

KEY IDEA

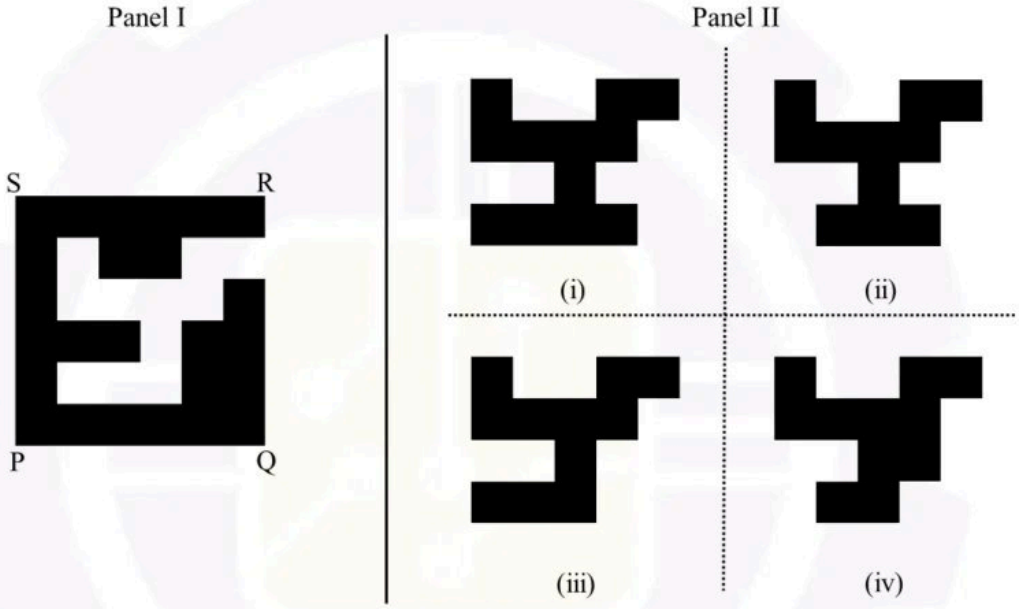
Find the shared meaning of the given words (to speed up), then pick the option that is a synonym of that meaning and not an antonym or an unrelated word.

WATCH OUT FOR

Choosing (B) Retard because it is also about speed. It is the opposite of the given words, and the question asks for a similar meaning.

Q.2 MCQ · 1 MARK

General Aptitude › Spatial

Q.2	A black square PQRS has been cut into two parts. One part of it is shown in Panel I. Which one of the shapes in Panel II is the other part? 
(A)	(i)
(B)	(ii)
(C)	(iii)
(D)	(iv)

Official answer: **C**

SOLUTION

1. The two parts together make the whole square PQRS. So the other part is exactly the white region inside the outline of Panel I, in the same position and orientation.
2. Read the white region on the grid, with rows and columns counted from corner S. Row 2: column 2, and columns 5 and 6. Row 3: columns 2 to 5. Row 4: column 4 only, because the tab fills columns 2-3 and the right-hand black part fills columns 5-6. Row 5: columns 2 to 4. The piece is 5 cells wide (columns 2-6), 4 cells tall (rows 2-5), and has $1 + 2 + 4 + 1 + 3 = 11$ cells.

3. Describe its outline. The top row has a single cell at the left end and a two-cell bar at the right end. The second row is a run of four cells from the left edge that stops one cell short of the right edge, so the top-right bar sticks out over a step. Next comes a one-cell stem in the middle column (column 4). The bottom row is a three-cell foot that runs from the left edge and ends directly under the stem.
4. Check the bottom row. Rows 4 and 5 are black at columns 5 and 6, so the missing piece has no cell to the right of its stem at the bottom. Options (i) and (ii) both have a bottom bar that runs out on both sides of the stem. That bar would overlap the black part, so (i) and (ii) are ruled out.
5. Check the right edge. Column 6 is white only in row 2, because the black part fills column 6 from row 3 down. So the piece is one cell tall at its right edge, a top-right bar over a step. Option (iv) goes two rows deep at its right edge, which would overlap the black cell at row 3, column 6. So (iv) is ruled out.
6. Option (iii) matches row by row. It has a single cell at the top left, a top-right bar over a step, a four-cell second row, a one-cell stem and a foot running to the lower left that ends under the stem. So (iii) is the other part, which is option (C).

KEY IDEA

The missing piece is the white region left inside the square's outline, in the same orientation. Read it row by row on the grid. The black part's steps on the right side (column 6 from row 3, columns 5-6 from rows 4-5) fix the piece's right edge and its bottom row, and only (iii) fits both.

WATCH OUT FOR

Matching only the top of the gap, the short left arm and the long right bar, and picking (i). The bottom row of the gap is a foot running left from the stem, with nothing to its right because the black part fills that corner. A piece with an I-beam bottom bar cannot fit. Another slip is to miss the step under the top-right bar, which is what rules out (iv).

Q.3 MCQ · 1 MARK

General Aptitude › Statistics & probability

A day can only be cloudy or sunny. The probability of a day being cloudy is 0.5, independent of the condition on other days. What is the probability that in any given four days, there will be three cloudy days and one sunny day?

(A) 1/4

(B) 3/4

(C) 2/3

(D) 3/8

Official answer: **A**

SOLUTION

1. Each day is cloudy or sunny with probability 0.5 each, independently, so the number of cloudy days X in four days is binomial with $n = 4$, $p = 0.5$.
2. Binomial rule: $P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$.
3. Three cloudy days and one sunny day means $k = 3$: $P(X = 3) = \binom{4}{3} (0.5)^3 (0.5)^1$.
4. $\binom{4}{3} = 4$ and $(0.5)^3 (0.5) = (0.5)^4 = \frac{1}{16}$.
5. So $P = 4 \times \frac{1}{16} = \frac{1}{4}$.
6. Check by listing: there are $2^4 = 16$ equally likely sequences of cloudy/sunny days. Exactly 4 of them have one sunny day (the sunny day can fall on any of the 4 days), giving $\frac{4}{16} = \frac{1}{4}$. This is option (A).

KEY IDEA
With independent two-outcome trials of equal probability, count the favourable arrangements and divide by the 2^n equally likely sequences (or use the binomial formula).

WATCH OUT FOR
Forgetting the factor $\binom{4}{3} = 4$ for the position of the sunny day and answering $(0.5)^4 = 1/16$, or confusing the result with 3/4 or 3/8.

Q.4 MCQ · 1 MARK

General Aptitude › Numerical

The values of Stock A and Stock B on a particular day are Rs. 50 and Rs. 80, respectively. An investor invests Rs. 100 in Stock A and Rs. 80 in Stock B. He sells all the stocks the next day when the value of Stock A is Rs. 55 and Stock B is Rs. 70. The profit made by the investor is Rs. _____

(A) 0

(B) 5

(C) 10

(D) 20

Official answer: **A**

SOLUTION

1. The amounts invested are given, not the number of shares, so first find the number of units bought: $\text{units} = \dfrac{\text{amount invested}}{\text{price per unit}}$.
2. Stock A: $\dfrac{100}{50} = 2$ units.
3. Stock B: $\dfrac{80}{80} = 1$ unit.
4. Next day, A is worth $2 \times 55 = 110$, so the gain on A is $110 - 100 = 10$.
5. Next day, B is worth $1 \times 70 = 70$, so the loss on B is $80 - 70 = 10$.
6. Net profit $= 10 - 10 = 0$. Check: total invested $= 100 + 80 = 180$ and total received $= 110 + 70 = 180$, so the profit is 0, option (A).

KEY IDEA
When money invested is given rather than the number of shares, convert it to units (amount divided by price), then value the units at the new price and compare the total received with the total invested.

WATCH OUT FOR
Using the price rise of Stock A (5) and fall of Stock B (10) directly as the gain and loss, ignoring that Rs. 100 buys 2 units of A while Rs. 80 buys only 1 unit of B. This gives a profit of 5 - 10 = -5 or a similar wrong figure instead of 0.

Q.5 MCQ · 1 MARK

General Aptitude › Reasoning

'When it is raining, peacocks dance.'

Based only on this sentence, which one of the following options is necessarily true?

(A) Peacocks dance only when it is raining.

(B) When peacocks dance, it is raining.

(C) When peacocks are not dancing, it is not raining.

(D) When it is not raining, peacocks do not dance.

Official answer: **C**

SOLUTION

1. Let R = "it is raining" and D = "peacocks dance". The sentence 'When it is raining, peacocks dance.' is the conditional $R \Rightarrow D$.
2. A conditional is logically equivalent to its contrapositive: $R \Rightarrow D \equiv \neg D \Rightarrow \neg R$. Its converse $D \Rightarrow R$ and its inverse $\neg R \Rightarrow \neg D$ are not implied by it.
3. (A) 'Peacocks dance only when it is raining' means $D \Rightarrow R$. This is the converse, so it is not necessarily true.
4. (B) 'When peacocks dance, it is raining' is also $D \Rightarrow R$, the converse. Not necessarily true.
5. (C) 'When peacocks are not dancing, it is not raining' is $\neg D \Rightarrow \neg R$, the contrapositive. It is necessarily true: if it were raining, the peacocks would dance.
6. (D) 'When it is not raining, peacocks do not dance' is $\neg R \Rightarrow \neg D$, the inverse. It is not implied, because peacocks may also dance on dry days.
7. Only (C) must be true, so the answer is (C).

KEY IDEA
A statement 'If P then Q' is equivalent only to its contrapositive 'If not Q then not P'. Its converse and inverse do not follow from it.

WATCH OUT FOR
Reading 'When it is raining, peacocks dance' as 'peacocks dance only when it is raining'. That confuses the statement with its converse, and the inverse in (D) is the same slip.

Q.6 MCQ · 2 MARKS

General Aptitude › Reasoning

Water : P :: Food : Q

Choose the P and Q combination from the options below to form a meaningful analogy.

(A) P = Thirst; Q = Hunger

(B) P = Drink; Q = Hunger

(C) P = Thirst; Q = Satiated

(D) P = Wet; Q = Critic

Official answer: **A**

SOLUTION

1. An analogy $A : B :: C : D$ needs the same relationship in both pairs, so first find the relation in Water : P.
2. Water is what a person takes to relieve thirst. The relation is: the thing taken (water) satisfies a bodily need (thirst).
3. Apply the same relation to Food : Q. Food is what a person takes to relieve hunger, so Q should be the need that food relieves: Hunger.
4. Check (A): Water : Thirst :: Food : Hunger. Each item satisfies the need named beside it, so the relation holds on both sides.
5. Check (B): Water : Drink pairs a thing with an action or category (water is something you drink), while Food : Hunger pairs a thing with the need it relieves. The two relations differ, so it fails.
6. Check (C): Water : Thirst pairs a thing with the need it relieves, but Food : Satiated pairs a thing with the state it produces. The relations differ, so it fails.
7. Check (D): Water : Wet and Food : Critic have no common relation, so it fails.

8. Only (A) keeps one consistent relation (item to the need it satisfies) on both sides.

KEY IDEA

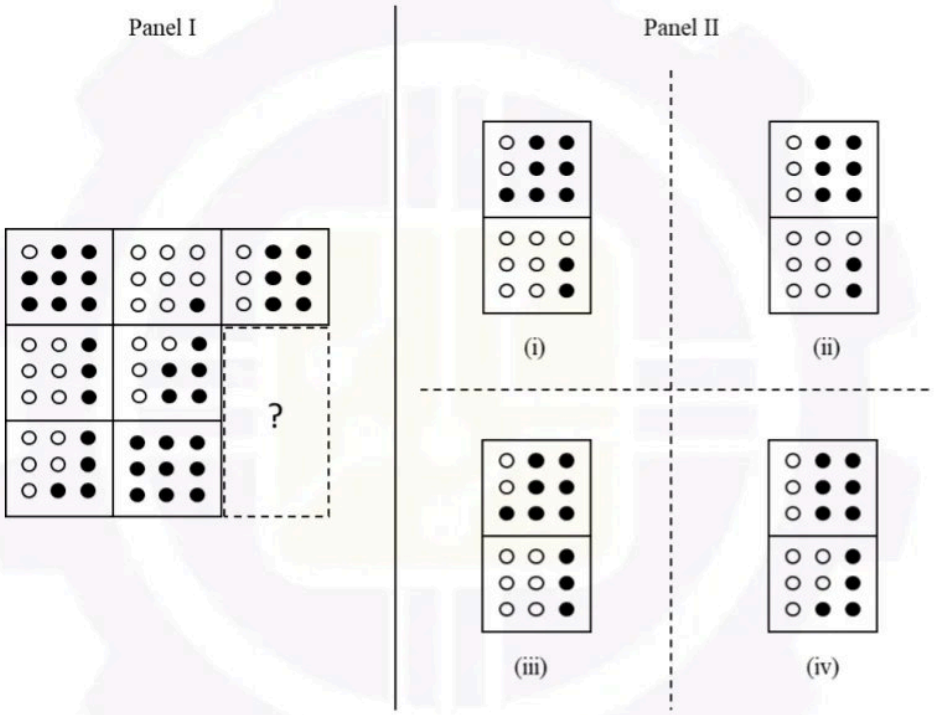
In an analogy, name the relation in the complete pair first (water satisfies thirst), then pick the option that gives the same relation for the second pair (food satisfies hunger).

WATCH OUT FOR

Choosing (B) because Water : Drink looks natural. It breaks the pattern, because Food : Hunger relates an item to a need, not to an action or category. Choosing (C) because Satiated sounds like a result of eating overlooks that Thirst is a need, not a result.

Q.7 MCQ · 2 MARKS

General Aptitude › Spatial

Q.7	Two tiles are missing in Panel I. Which one of the options in Panel II is the appropriate choice for the missing tiles? 
(A)	(i)
(B)	(ii)
(C)	(iii)
(D)	(iv)

Official answer: **A**

SOLUTION

1. Count the filled dots in each tile of Panel I. Row 1: 8, 1, 6. Row 2: 3, 5, missing. Row 3: 4, 9, missing.
2. The counts 8, 1, 6, 3, 5, 4, 9 are seven of the numbers 1 to 9, each used once. The only unused numbers are 2 and 7, so the two missing tiles hold 2 and 7 dots in some order.
3. The known lines suggest a magic square, all sums equal to 15: row 1 is $8 + 1 + 6 = 15$, column 1 is $8 + 3 + 4 = 15$, and column 2 is $1 + 5 + 9 = 15$.

4. Column 3 must also sum to 15, so $6 + x + y = 15$ and $x + y = 9$. Row 2 gives $3 + 5 + x = 15$, so $x = 7$. Row 3 gives $4 + 9 + y = 15$, so $y = 2$. The top missing tile has 7 filled dots and the bottom one has 2.
5. Count the filled dots in Panel II. (i): top 7, bottom 2. (ii): top 6, bottom 2. (iii): top 7, bottom 3. (iv): top 6, bottom 4. Only (i) gives 7 on top and 2 below.
6. Cross-check with the filling order of the tiles. Each tile is filled column by column from the right, and each column is filled from the bottom upward. So 1 is the bottom-right dot, 3 is the whole right column, 6 is the right and middle columns, and 7 adds the bottom-left dot. The 2-dot tile is the bottom two dots of the right column. Option (i) shows exactly these two tiles.
7. The answer is (i), option A.

KEY IDEA

Count the filled dots in each tile. The counts form a magic square of 1 to 9 with every row, column and diagonal summing to 15, so the missing tiles hold 7 and 2 dots.

WATCH OUT FOR

Comparing how the dots look instead of counting them, or taking the missing tiles in the wrong order (2 above 7). Row 2 needs 7 and row 3 needs 2, so the 7-dot tile goes on top.

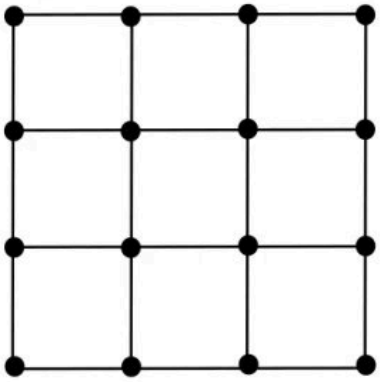
Q.8 MCQ · 2 MARKS

General Aptitude › Reasoning

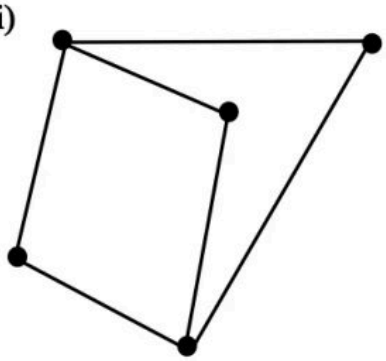
Q.8 Figures (i) and (ii) represent intercity highway systems. The black dots represent cities and the line segments between them represent intercity highways.
A salesperson needs to make a trip. She needs to start from a city, visit each of the remaining cities exactly once, and finally return to the same city from which she started.

Which one of the following options is then true?

(i)



(ii)



(A)	Such a trip is possible for (i), but not for (ii).
(B)	Such a trip is possible for (ii), but not for (i).
(C)	Such a trip is possible for both (i) and (ii).
(D)	Such a trip is possible neither for (i) nor for (ii).

Official answer: **A**

SOLUTION

1. The trip described starts at a city, visits every other city exactly once and returns to the start. In graph terms it is a closed route through all the dots, a Hamiltonian cycle. Check each figure.
2. Figure (i) is a 4×4 grid of 16 cities. Label the cities by (row, column). A valid trip is: (1,1), (1,2), (1,3), (1,4), then down to (2,4), left to (2,3), (2,2), down to (3,2), right to (3,3), (3,4), down to (4,4), left to (4,3), (4,2), (4,1), then up to (3,1), (2,1), and back to (1,1). Each step moves along a highway. This covers $4 + 3 + 3 + 4 + 2 = 16$ cities, each once, and returns to the start. So the trip is possible for (i).

3. Figure (ii) has 5 cities. Call them A (top-left), B (top-right), C (inner right), D (left) and E (bottom). The highways are A-B, A-C, A-D, B-E, C-E and D-E. So A and E are each joined only to B, C and D, and B, C and D are joined only to A and E.
4. In any closed route, each city is entered once and left once, so it uses exactly 2 of its highways. A trip through all 5 cities uses exactly 5 highways.
5. B, C and D each have only 2 highways, so the trip must use both highways at each of them. That is 6 highways, and no highway is counted twice, since none joins two of B, C, D. But the trip has only 5 highways. This is a contradiction.
6. Equivalently, every highway joins one of $\{A, E\}$ to one of $\{B, C, D\}$, so the cities alternate between the two groups along a closed route. That needs the groups to be equal in size, but they have 2 and 3 cities. So no such trip exists for (ii).
7. Hence the trip is possible for (i) but not for (ii): option (A).

KEY IDEA

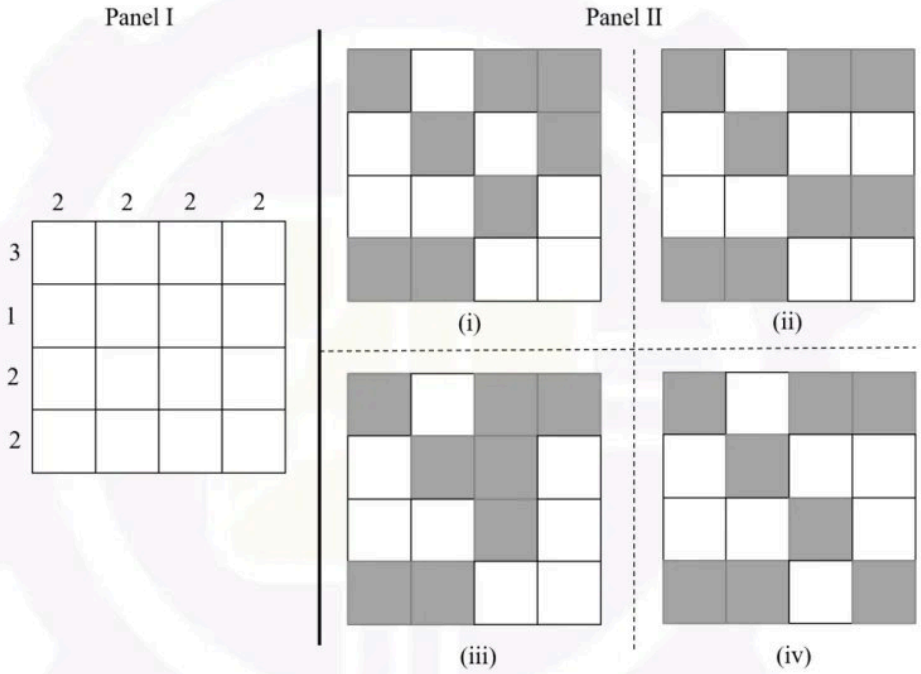
The trip is a closed route through every city once, a Hamiltonian cycle. A grid with an even number of rows or columns has one. A graph where all cities lie on two sides and the sides are unequal (here 2 and 3) cannot have one, because the route must alternate sides.

WATCH OUT FOR

Assuming a route can pass through a city twice, or that it need not return to the start. Another is to assume that having many highways in (i) guarantees a trip, or that few highways in (ii) is enough to rule one out, without actually finding the route or the contradiction.

Q.9 MCQ · 2 MARKS

General Aptitude › Reasoning

Q. 9	The figure in Panel I below is a grid of cells with four rows and four columns. The numbers on the top and on the left represent the number of cells that are to be shaded in that column and row, respectively. Which one of the options shown in Panel II below represents the grid shaded correctly? 
(A)	(i)
(B)	(ii)
(C)	(iii)
(D)	(iv)

Official answer: **B**

SOLUTION

1. The row clues are $3, 1, 2, 2$ (total 8) and the column clues are $2, 2, 2, 2$ (total 8), so a correct grid has exactly 8 shaded cells.
2. Check the row counts first, because one wrong row rejects an option at once.
3. Option (i): row 2 is white, shaded, white, shaded. That is 2 shaded cells, but the clue is 1. Rejected.
4. Option (iii): row 2 is white, shaded, shaded, white. That is 2 shaded cells, but the clue is 1. Rejected.

5. Option (iv): row 4 is shaded, shaded, white, shaded. That is 3 shaded cells, but the clue is 2. Rejected.
6. Option (ii): the rows have $3, 1, 2, 2$ shaded cells, which matches the row clues.
7. Option (ii), columns: column 1 is shaded, white, white, shaded, so 2. Column 2 is white, shaded, white, shaded, so 2. Column 3 is shaded, white, shaded, white, so 2. Column 4 is shaded, white, shaded, white, so 2. All match the column clues, so option (ii) is correct.

KEY IDEA

Test each candidate grid against the clues. Count the shaded cells in each row and column. A single row or column that disagrees with its clue rules the grid out.

WATCH OUT FOR

Checking only the column clues, or only some of the rows. Grids (i) and (iii) have correct columns in places but a wrong row 2, and grid (iv) has a wrong row 4. Check every row and every column before choosing.

Q.10 MCQ · 2 MARKS

General Aptitude › Statistics & probability

An unbiased six-faced dice whose faces are marked with numbers 1, 2, 3, 4, 5, and 6 is rolled twice in succession and the number on the top face is recorded each time. The probability that the sum of the two recorded numbers is a prime number is _____

(A) $\frac{3}{36}$
(B) $\frac{13}{36}$
(C) $\frac{15}{36}$
(D) $\frac{19}{36}$

Official answer: **C**

SOLUTION

1. Two rolls of the dice give $6 \times 6 = 36$ equally likely ordered outcomes.
2. The sum lies between 2 and 12. The primes in that range are 2, 3, 5, 7 and 11.
3. Count the ordered pairs for each prime sum. Sum 2: $(1, 1)$, which is 1 way. Sum 3: $(1, 2), (2, 1)$, which is 2 ways. Sum 5: $(1, 4), (2, 3), (3, 2), (4, 1)$, which is 4 ways.
4. Sum 7: $(1, 6), (2, 5), (3, 4), (4, 3), (5, 2), (6, 1)$, which is 6 ways. Sum 11: $(5, 6), (6, 5)$, which is 2 ways.
5. Favourable outcomes: $1 + 2 + 4 + 6 + 2 = 15$.
6. Probability $= \frac{\text{favourable}}{\text{total}} = \frac{15}{36}$, which is option (C).

KEY IDEA

List the prime values a sum of two dice can take (2, 3, 5, 7, 11). Count the ordered pairs for each and divide the total by 36.

WATCH OUT FOR

Counting unordered pairs, so that (1,2) and (2,1) count once. Another slip is leaving out 2 (the sum from (1,1)) or 11, which gives 12/36 or 13/36.

Computer Science and Information Technology Q.11 to Q.65

Q.11 MCQ · 1 MARK

Discrete Mathematics › Mathematical logic

For two different persons x and y, the predicate $M(x, y)$ denotes that x knows y. Consider the following statement.

There is a person who does not know anyone else, but that person is known by everyone else.

Which one of the following expressions represents the above statement?

(A) $(\exists y)(\forall x)\ ((x \neq y) \rightarrow (M(x, y) \wedge \neg M(y, x)))$

(B) $(\forall y)(\exists x)\ ((x \neq y) \rightarrow (M(x, y) \wedge \neg M(y, x)))$

(C) $(\exists y)(\exists x)\ ((x \neq y) \rightarrow (M(x, y) \wedge \neg M(y, x)))$

(D) $(\forall y)(\forall x)\ ((x \neq y) \rightarrow (M(x, y) \wedge \neg M(y, x)))$

Official answer: **A**

SOLUTION

1. The statement names one particular person who is known by everyone else and who knows no one else. So the outer quantifier is $\exists$ on that person. Call the person y: $(\exists y)$.
2. "Everyone else" means every person x with $x \neq y$. A universal claim about such persons is written $(\forall x)\ ((x \neq y) \rightarrow \dots)$, because the guard $x \neq y$ restricts the claim to the others.
3. "Known by everyone else": each such x knows y. Since $M(a, b)$ means a knows b, this is $M(x, y)$.
4. "Does not know anyone else": y does not know any such x. This is $\neg M(y, x)$.
5. Both conditions must hold for every $x \neq y$, so join them with $\wedge$ under the same guard:
$$(\exists y)(\forall x)\ \big((x \neq y) \rightarrow (M(x, y) \wedge \neg M(y, x))\big)$$
This is option A.
6. Option D, $(\forall y)(\forall x)$, claims the condition for every ordered pair of distinct persons. Take two different people a and b. Using $(x, y) = (a, b)$ gives $M(a, b) \wedge \neg M(b, a)$. Using $(x, y) = (b, a)$ gives $M(b, a) \wedge \neg M(a, b)$. These contradict each other, so D is false whenever there are at least two people. It does not express the statement.
7. Options B, $(\forall y)(\exists x)$, and C, $(\exists y)(\exists x)$, use an existential x with an implication. Choosing $x = y$ makes the premise $x \neq y$ false, so the implication is true. Hence $(\exists x)(\dots)$ is true for every y. B is therefore true of every y, and C is true as soon as some y exists. Neither says anything about the persons other than y, so neither expresses the statement.
8. Only A puts $\forall$ on x under the guard $x \neq y$, so only A forces every other person to know y and y to know none of them. The answer is A.

KEY IDEA

"There is a person" is $\exists$. "Everyone else" is $\forall$ with the guard $x \neq y$, joined by an implication. Both conditions, $M(x, y)$ and $\neg M(y, x)$, sit under that one guard.

WATCH OUT FOR

Using $\exists x$ with an implication. Picking $x = y$ makes the premise false, so the implication holds trivially and the formula says nothing. A guarded "for everyone else" needs $\forall x$.

Q.12 MCQ · 1 MARK

Data Structures and Algorithms › Trees & heaps

Q.12 The set **T** represents various traversals over binary tree. The set **S** represents the order of visiting nodes during a traversal.

T	S
I: Inorder	L: left subtree, node, right subtree
II: Preorder	M: node, left subtree, right subtree
III: Postorder	N: left subtree, right subtree, node

Which one of the following is the correct match from **T** to **S** ?

(A) I – L, II – M, III – N

(B) I – M, II – L, III – N

(C) I – N, II – M, III – L

(D) I – L, II – N, III – M

Official answer: **A**

SOLUTION

1. Inorder traversal visits the left subtree, then the node, then the right subtree. This is exactly description L.
2. Preorder traversal visits the node first, then the left subtree, then the right subtree. This is exactly description M.

3. Postorder traversal visits the left subtree, then the right subtree, then the node last. This is exactly description N.
4. Therefore I (Inorder) – L, II (Preorder) – M, III (Postorder) – N.
5. This exact mapping I–L, II–M, III–N is option (A).

KEY IDEA
Each binary tree traversal is defined purely by the relative order in which the node is visited with respect to its left and right subtrees: Inorder = Left, Node, Right; Preorder = Node, Left, Right; Postorder = Left, Right, Node.

WATCH OUT FOR
Confusing preorder and postorder: remembering "node visited first" or "node visited last" without anchoring it to the mnemonic (Pre = node before children, Post = node after children), which leads to swapping M and N.

Q.13 MCQ · 1 MARK

Theory of Computation › Turing machines

Which one of the following statements is equivalent to the following assertion?

Turing machine M decides the language $L \subseteq \{0,1\}^*$

(A) Turing machine M halts on all input strings in $\{0,1\}^*$

(B) Turing machine M accepts all input strings in L

(C) Turing machine M rejects all input strings in $\{0,1\}^* - L$

(D) Turing machine M accepts all input strings in L and rejects all input strings in $\{0,1\}^* - L$

Official answer: **D**

SOLUTION

1. A Turing machine M **decides** $L \subseteq \{0,1\}^*$ if it halts on every input $w \in \{0,1\}^*$, accepting when $w \in L$ and rejecting when $w \notin L$.
2. Check (A): halting on all inputs is necessary but not sufficient. A machine that halts everywhere but accepts the wrong strings decides some other language, not L.
3. Check (B): accepting every string of L only gives $L \subseteq L(M)$. M may also accept strings outside L, or loop on them. This is at best a recognising condition.
4. Check (C): rejecting every string of $\{0,1\}^* - L$ says nothing about strings in L. M could reject them or loop on them.
5. Check (D): M accepts every string of L and rejects every string of $\{0,1\}^* - L$. Every input in $\{0,1\}^*$ lies in L or in its complement, so every input gets a definite accept or reject. Hence M halts on all inputs and $L(M) = L$, which is exactly the definition of deciding L.
6. Rejecting a string means halting in the reject state. So (D) already contains both halting on all inputs and correct classification of every string. Only (D) is equivalent to the assertion.

KEY IDEA
A decider for L halts on every input and gives the right verdict on each: accept inside L, reject outside L. Only a condition that covers both L and its complement, with a definite verdict on each, is equivalent to deciding.

WATCH OUT FOR
Choosing (A) because a decider halts on all inputs. Halting everywhere does not make the machine decide L, because it may accept or reject the wrong strings. Choosing (B) confuses deciding with recognising, where the machine may loop on strings outside L.

Q.14 MCQ · 1 MARK

Engineering Mathematics › Probability & statistics

The probability density function $f(x)$ of a random variable X which takes real values is

$$f(x) = \frac{1}{3\sqrt{2\pi}} \exp\left(-\frac{x^2}{18}\right), \quad x \in (-\infty, +\infty)$$

Which one of the following statements is correct about the random variable X ?

(A) X is an exponential random variable

(B) X is a normal random variable

(C) X is a Poisson random variable

(D) X is a uniform random variable

Official answer: **B**

SOLUTION

1. A normal density with mean μ and variance σ^2 is $f(x) = \dfrac{1}{\sigma\sqrt{2\pi}} \exp\left(-\dfrac{(x-\mu)^2}{2\sigma^2}\right)$ for $x \in (-\infty, +\infty)$.
2. Compare the exponents: $\dfrac{x^2}{18} = \dfrac{(x-0)^2}{2\sigma^2}$. This gives $\mu = 0$ and $2\sigma^2 = 18$, so $\sigma^2 = 9$ and $\sigma = 3$.
3. Check the constant: $\dfrac{1}{\sigma\sqrt{2\pi}} = \dfrac{1}{3\sqrt{2\pi}}$. This matches the given density, so $X \sim N(0, 9)$.
4. The other options fail. An exponential density is zero for $x < 0$, but this $f(x) > 0$ for every real x. A Poisson variable is discrete, with a probability mass function and no density. A uniform density is constant on a finite interval, but this $f(x)$ changes with x.
5. So X is a normal random variable, option (B).

KEY IDEA

Match the density to the standard normal form. The exponent gives μ and σ^2, and the constant $\dfrac{1}{\sigma\sqrt{2\pi}}$ confirms it.

WATCH OUT FOR

Reading $\exp(-x^2/18)$ as an exponential random variable because of the word "exp". An exponential distribution has density $\lambda e^{-\lambda x}$ for $x \geq 0$ only. A quadratic in the exponent signals a normal distribution.

Q.15 MCQ · 1 MARK

Database Management and Warehousing › Relational algebra

Q.15 In the context of DBMS, consider the two sets **T** and **S** given below.

T	S
I: Logical schema	L: Views
II: Physical schema	M: File organization and indexes
III: External schema	N: Relations

Which one of the following is the correct match from **T** to **S** ?

(A) I – L, II – M, III – N

(B) I – M, II – L, III – N

(C) I – N, II – M, III – L

(D) I – N, II – L, III – M

Official answer: **C**

SOLUTION

1. The three-schema architecture has three levels: external (the user views), conceptual (logical) and internal (physical).
2. The logical (conceptual) schema describes the whole database in terms of the data model. In the relational model this is the set of **relations** (tables), so I matches N.

3. The physical (internal) schema describes how the data is stored: **file organization and indexes**. So II matches M.
4. The external schema describes what each user group sees, which is defined through **views**. So III matches L.
5. The match is I – N, II – M, III – L. Check against the options: (A) pairs I with L and (B) pairs I with M, both wrong; (D) pairs II with L, wrong. Only option (C) fits.

KEY IDEA
The three-schema architecture maps the logical (conceptual) schema to relations, the physical (internal) schema to file organization and indexes, and the external schema to views.

WATCH OUT FOR
Matching the logical schema with views, or the external schema with relations. Views are what users see (external). Relations make up the logical design of the whole database.

Q.16 MCQ · 1 MARK

Digital Electronics › Boolean logic

Which one of the following options is **not** a property of Boolean Algebra?

Note: $+$ is OR operation, $\cdot$ is AND operation, and $'$ is NOT operation

(A) $a + b = b + a$

(B) $a \cdot a' = 1$

(C) $a + a' = 1$

(D) $a \cdot b = b \cdot a$

Official answer: **B**

SOLUTION

1. Check each option against the postulates of Boolean algebra.
2. (A) $a + b = b + a$ is the commutative law for OR. It holds.
3. (D) $a \cdot b = b \cdot a$ is the commutative law for AND. It holds.
4. (C) $a + \overline{a} = 1$ is the complement law for OR. It holds: if $a = 0$ then $0 + 1 = 1$, and if $a = 1$ then $1 + 0 = 1$.
5. (B) The complement law for AND is $a \cdot \overline{a} = 0$. Test it: if $a = 0$ then $0 \cdot 1 = 0$, and if $a = 1$ then $1 \cdot 0 = 0$. The result is never 1, so $a \cdot \overline{a} = 1$ is false.
6. So the only statement that is not a property of Boolean algebra is option B.

KEY IDEA
The complement laws pair OR with 1 and AND with 0: $a + \overline{a} = 1$ and $a \cdot \overline{a} = 0$.

WATCH OUT FOR
Swapping the two complement laws and writing $a \cdot \overline{a} = 1$, which is the dual of the OR law with the constant left unchanged. Duality swaps the constants as well as the operators.

Q.17 MCQ · 1 MARK

Programming in C › Functions & scope

In C runtime environment, which one of the following is stored in heap?

(A) A static variable declared inside a function

(B) An array of integers declared inside a function

(C) A dynamically allocated array of integers created using `malloc()` function call

(D) Return address of a function

Official answer: **C**

SOLUTION

1. A `static` local variable is stored in the data (or BSS) segment, not the heap. It retains its value between calls because it has static storage duration, not because it resides on the heap.
2. A plain array of integers declared inside a function (without `static`) is a local/automatic variable, allocated in the function's stack frame; it is destroyed when the function returns.
3. A dynamically allocated array created using `malloc()` explicitly requests memory from the heap at runtime; this memory persists until explicitly freed with `free()`. This is the defining example of heap storage.
4. The return address of a function is pushed as part of the function call's activation record on the stack, used to resume execution after the call returns. It is stack-resident, not heap-resident.
5. Only option (C) describes memory obtained via dynamic allocation, so it is the one stored in the heap.

KEY IDEA

A C program's runtime memory layout has four regions: code/text, static/global (data+BSS) segment, stack (activation records: locals, parameters, return addresses), and heap (memory obtained via `malloc` / `calloc` / `realloc`, released only by `free` or process exit).

WATCH OUT FOR

Assuming any "array" or anything that "lives beyond a single line" must be on the heap, or confusing `static` storage duration with heap allocation: a `static` local variable is in the data/BSS segment, not the heap, even though it also outlives the function call.

Q.18 MCQ · 1 MARK

Computer Organization and Architecture › I/O, interrupts & DMA

Consider the following two statements about interrupt handling mechanisms in a CPU.

S1: In non-vectored interrupt mechanism, it usually takes more time to start the Interrupt Service Routine (ISR) when compared to that in a vectored interrupt mechanism.

S2: In daisy-chain interrupt mechanism, the CPU polls all the input devices individually to determine the source of the interrupt.

Which one of the following options is correct with respect to S1 and S2 ?

(A) Both S1 and S2 are true

(B) Both S1 and S2 are false

(C) S1 is true and S2 is false

(D) S1 is false and S2 is true

Official answer: **C**

SOLUTION

1. S1: A vectored interrupt mechanism has the interrupting device supply the address (vector) of its own ISR directly to the CPU during the interrupt-acknowledge cycle, so the CPU can jump almost immediately to the correct ISR.
2. In a non-vectored interrupt mechanism, the CPU always transfers control to one fixed/predetermined address first, and additional steps (software polling of device status registers, or a priority-encoding step) are needed to identify which device caused the interrupt before the correct ISR can begin.
3. These extra identification steps mean starting the ISR usually takes more time in the non-vectored scheme than in the vectored scheme, so S1 is TRUE.
4. S2: Daisy-chaining is a hardware-based interrupt-priority and identification scheme: devices are connected in series, and a single interrupt-acknowledge signal from the CPU propagates serially

through the chain.

5. The first device in the chain that is requesting service intercepts (blocks) the propagating signal and identifies itself (e.g., by gating its vector onto the bus); devices further down the chain never see the signal that cycle.
6. This hardware propagation is different from software polling, where the CPU itself sequentially queries each device's status register to find the interrupting source; daisy-chaining exists specifically to avoid that CPU overhead.
7. Since the CPU does not poll all devices individually in a daisy-chain scheme, S2's claim is FALSE.
8. Combining the two results: S1 is true and S2 is false, which is option (C).

KEY IDEA

Vectored interrupts let the interrupting device supply its ISR address/vector directly, so dispatch is fast; non-vectored interrupts force the CPU into extra identification steps before it can jump to the right ISR. Daisy-chaining resolves priority and identifies the interrupting device entirely in hardware (the acknowledge signal is passed serially and intercepted by the requesting device). It is the alternative to CPU software polling, not an instance of it.

WATCH OUT FOR

Confusing daisy-chaining (a hardware priority/identification scheme) with software polling (where the CPU itself queries each device in sequence); the two are often taught back to back and are easy to conflate because both eventually "check devices one by one", but in daisy-chaining that checking is done by hardware propagation of a signal, not by the CPU executing a polling loop.

Q.19 MCQ · 1 MARK

Programming in C › Functions & scope

Q.19 Consider the following three ANSI-C programs, P1, P2, and P3.

P1	P2	P3

`#include <stdio.h>`
 	
`int a=5;`
`int main(){`
`  int a=7;`
`  return(0);`
`}`
	`#include <stdio.h>`
	
	`int main(){`
	`  int a=5;`
	`  int a=7;`
	`  return(0);`
	`}`
		`#include <stdio.h>`
		
		`int main(){`
		`  int a=5;`
		`  float a=7;`
		`  return(0);`
		`}`

Which one of the following statements is true?

(A) Only P1 will compile without any error

(B) Only P2 will compile without any error

(C) Only P3 will compile without any error

(D) All three programs P1, P2, and P3 will compile without any error

Official answer: **A**

SOLUTION

1. In P1, `int a=5;` is declared at file (global) scope, while `int a=7;` is declared inside `main()` 's block scope. A block-scope declaration is allowed to have the same name as a file-scope one; the inner `a` simply shadows the outer `a` for the duration of the block. This is legal ANSI-C: P1 compiles without any error.
2. In P2, both `int a=5;` and `int a=7;` are declared directly inside the same block (the body of `main()`) with no intervening nested block. Declaring the same identifier twice with the same type in the same scope

is an illegal redeclaration in ANSI C (a compiler such as gcc reports `redefinition of 'a'`). So P2 fails to compile.

3. In P3, `int a=5;` and `float a=7;` are again both declared directly in the same block (main's body). This is the same same-scope redeclaration violation as P2, compounded by a conflicting type for `a` (`int` vs `float`). So P3 also fails to compile.
4. Since P1 is the only one of the three programs that compiles cleanly, and both P2 and P3 fail due to illegal redeclaration of `a` within the same scope, the true statement is option (A): Only P1 will compile without any error.

KEY IDEA

In C, an identifier may be redeclared in a nested (inner) scope to shadow an outer declaration (that is legal), but declaring the same identifier twice within the exact same scope (the same block) is always a compile-time redeclaration error, regardless of whether the types match.

WATCH OUT FOR

Treating all three programs as "declaring `a` twice" as if that phrase alone decides compilability, without checking whether the two declarations sit in the SAME scope (P2, P3: illegal) or in DIFFERENT, nested scopes (P1: legal shadowing). Some also mistakenly believe C requires the types to differ (as in P3) before a redeclaration becomes an error, missing that P2's same-type, same-scope redeclaration is equally illegal.

Q.20 MCQ · 1 MARK

Database Management and Warehousing › Transactions

Consider concurrent execution of two transactions $T1$ and $T2$ in a DBMS, both of which access a data object A. For these two transactions to not conflict on A, which one of the following statements must be true?

(A) Both $T1$ and $T2$ only read A

(B) $T1$ reads A and $T2$ writes A

(C) $T1$ writes A and $T2$ reads A

(D) Both $T1$ and $T2$ write A

Official answer: **A**

SOLUTION

1. Two operations of different transactions on the same data item **conflict** if at least one of them is a write.
2. This gives three conflicting pairs on A: read-write $(R_1(A), W_2(A))$, write-read $(W_1(A), R_2(A))$ and write-write $(W_1(A), W_2(A))$.
3. Check (B): $T1$ reads and $T2$ writes, a read-write pair, so they conflict.
4. Check (C): $T1$ writes and $T2$ reads, a write-read pair, so they conflict.
5. Check (D): both write, a write-write pair, so they conflict.
6. Check (A): both only read. A read-read pair is never a conflict, because swapping the two reads changes neither the values read nor the final state of A.
7. So the only way the transactions do not conflict on A is that both only read A.

KEY IDEA
Two operations conflict when they belong to different transactions, touch the same data item, and at least one is a write. If neither transaction writes, there is no conflict.

WATCH OUT FOR
Treating a read-write or write-read pair as harmless because only one transaction writes. Any single write is enough to make the pair conflict, because swapping the two operations can change what the reader sees or the final value of A.

Q.21 MCQ · 1 MARK

Computer Networks › Performance

Consider a file of size 4 million bytes being transferred between two hosts connected via a path consisting of three consecutive links of bandwidth 2 Mbps, 500 kbps, and 1 Mbps, respectively. All processing delays and propagation delays are negligible. Assume that there is no other background traffic over the path and no other additional overhead to transfer the file.

Which one of the following is the total time (in seconds) to transfer the file?

Note: 1M=10^6, 1k=10^3

(A) 731

(B) 64

(C) 8

(D) 16

Official answer: **B**

SOLUTION

1. File size in bits: $4 \times 10^6 \text{ bytes} \times 8 = 32 \times 10^6 \text{ bits}$.
2. Link rates (with $1\text{M} = 10^6$, $1\text{k} = 10^3$): $2 \text{ Mbps} = 2 \times 10^6$, $500 \text{ kbps} = 5 \times 10^5$ and $1 \text{ Mbps} = 10^6$ bits per second.
3. Processing and propagation delays are negligible, and there is no other traffic or overhead. The file therefore flows through the three links as one continuous stream, and the slowest link sets the end-to-end throughput.
4. The slowest link is the 500 kbps link, so the end-to-end throughput is $5 \times 10^5 \text{ bits/s}$.
5. $$T = \frac{32 \times 10^6}{5 \times 10^5} = 64 \text{ s}$$
6. Check: the same data would take $\frac{32 \times 10^6}{2 \times 10^6} = 16 \text{ s}$ on the first link and $\frac{32 \times 10^6}{10^6} = 32 \text{ s}$ on the third. Both links are faster than the bottleneck and work at the same time as it, so they add nothing. The answer is option (B).

KEY IDEA
With negligible delays, no overhead and no other traffic, a file sent over several links in series is limited by the slowest link. The time is the file size in bits divided by the bottleneck rate. Do not add up the time on each link.

WATCH OUT FOR
Adding the transfer times of the three links (16 + 64 + 32 = 112 s), which treats each link as if it waited for the whole file. Other slips are forgetting to convert bytes to bits, which gives 8 s, and reading 500 kbps as 500 Mbps.

Q.22 MCQ · 1 MARK

Computer Networks › IPv4

Which one of the following protocols may need to broadcast some of its messages?

(A) SMTP

(B) FTP

(C) DHCP

(D) HTTP

Official answer: **C**

SOLUTION

1. SMTP, FTP and HTTP run over TCP. TCP is connection-oriented and point-to-point, so each of their messages goes to one known unicast address; none of them needs to be broadcast.
2. DHCP runs over UDP, which can carry broadcast datagrams. A client that is booting has no IP address of its own and does not know any DHCP server's address.
3. So the client sends DHCPDISCOVER with source `0.0.0.0` and destination `255.255.255.255` (the limited broadcast address), so that any DHCP server on the local network can receive it.
4. The server's DHCPOFFER and DHCPACK may also be broadcast, because the client cannot yet receive unicast packets addressed to an IP address it does not hold.
5. DHCP is therefore the only listed protocol that may need to broadcast some of its messages. The answer is (C).

KEY IDEA

Broadcast needs a connectionless transport (UDP) and a sender that does not yet know its peer's address. Only DHCP fits, because a booting host has no IP address and must discover a server by broadcast.

WATCH OUT FOR

Thinking that any application protocol can broadcast. SMTP, FTP and HTTP use TCP connections between two known hosts, so broadcast is impossible for them.

Q.23 MCQ · 1 MARK

Operating Systems › CPU scheduling

Which one of the following CPU scheduling algorithms **cannot** be preemptive?

(A) Shortest Remaining Time First (SRTF) Scheduling

(B) First Come First Serve (FCFS) Scheduling

(C) Round Robin Scheduling

(D) Priority Scheduling

Official answer: **B**

SOLUTION

1. A scheduling algorithm is preemptive if the CPU can be taken from a running process before its burst ends, either on a timer interrupt or when a better candidate arrives. Check each option against this.
2. SRTF is the preemptive form of shortest job first. A newly arrived process whose burst is shorter than the running process's remaining time takes the CPU. So SRTF is preemptive by definition.
3. Round Robin gives each process a time quantum. When the quantum expires the running process is moved to the back of the ready queue. This is preemption by the timer, so Round Robin is preemptive.
4. Priority Scheduling has both a preemptive and a non-preemptive version. In the preemptive version, a higher-priority arrival displaces the running process. So it can be preemptive.
5. FCFS always runs the process that arrived earliest, and runs it until it finishes or blocks. A later arrival always has a later arrival time, so it can never displace the running process. No timer or priority rule exists to take the CPU away, so FCFS cannot be preemptive. The answer is (B).

KEY IDEA
Preemption needs a rule that takes the CPU away from a running process, such as a timer or a better candidate arriving. FCFS has no such rule. Its order is fixed by arrival time, so it is non-preemptive by definition.

WATCH OUT FOR
Thinking Priority Scheduling is only non-preemptive. It has both versions, so it can be preemptive. Only FCFS has no preemptive form.

Q.24 MCQ · 1 MARK

Data Structures and Algorithms › Complexity

Consider the following functions, where n is a positive integer.

$$n^{1/3},\ \log(n),\ \log(n!),\ 2^{\log(n)}$$

Which one of the following options lists the functions in increasing order of asymptotic growth rate?

Note: Assume the base of log to be 2.

(A) $\log(n)$, $n^{1/3}$, $2^{\log(n)}$, $\log(n!)$
(B) $n^{1/3}$, $\log(n)$, $\log(n!)$, $2^{\log(n)}$
(C) $\log(n)$, $n^{1/3}$, $\log(n!)$, $2^{\log(n)}$
(D) $2^{\log(n)}$, $n^{1/3}$, $\log(n)$, $\log(n!)$

Official answer: **A**

SOLUTION

1. Since the log base is given as 2 (per the note), $2^{\log(n)} = 2^{\log_2(n)} = n$ by the definition of logarithm as the inverse of exponentiation.
2. $\log(n)$ grows slower than any positive power of n (in particular slower than $n^{1/3}$), so it is the smallest of the four functions.
3. Since the exponent $1/3 < 1$, $n^{1/3} = o(n)$. So $n^{1/3}$ grows more slowly than $n = 2^{\log(n)}$.
4. By Stirling's approximation, $\log(n!) = \Theta(n \log n)$, which grows strictly faster than n itself (since $\log n \to \infty$ as $n \to \infty$).
5. Combining all comparisons: $\log(n) < n^{1/3} < n\ (= 2^{\log(n)}) < n \log n\ (= \log(n!))$.
6. This increasing-growth-rate order, $\log(n)$, $n^{1/3}$, $2^{\log(n)}$, $\log(n!)$, is exactly option (A).

KEY IDEA
Simplify each function to a standard growth class using $2^{\log_2(n)} = n$ and $\log(n!) = \Theta(n \log n)$ (Stirling), then order the resulting classes: $\log n \ll n^{1/3} \ll n \ll n \log n$.

WATCH OUT FOR
Forgetting that "log" here means log base 2 (as stated in the note) and mis-simplifying $2^{\log(n)}$; a common error is to treat $2^{\log(n)}$ as growing faster than n (e.g., confusing it with $n^{\log(n)}$ or 2^n), or to assume $\log(n!)$ grows only like $\log(n)$ rather than $\Theta(n \log n)$ by Stirling's approximation.

Q.25 MSQ · 1 MARK

Data Structures and Algorithms › Complexity

Which of the following can be recurrence relation(s) corresponding to an algorithm with time complexity $\Theta(n)$?

(A) $T(n) = T(n-1) + 1, \quad T(1) = 1$
(B) $T(n) = 2T\left(\frac{n}{2}\right) + 1, \quad T(1) = 1$
(C) $T(n) = 2T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$

(D) $T(n) = T(n-1) + n, \quad T(1) = 1$

Official answer: **A, B**

SOLUTION

1. Option A: $T(n) = T(n-1) + 1$, $T(1) = 1$. Unrolling: $T(n) = T(n-1) + 1 = T(n-2) + 2 = \cdots = T(1) + (n-1) = n$. So $T(n) = \Theta(n)$, which qualifies.
2. Option B: $T(n) = 2T(n/2) + 1$. Master theorem with $a = 2,\ b = 2,\ f(n) = 1$: $n^{\log_b a} = n^{\log_2 2} = n$. Since $f(n) = 1 = O(n^{1-\varepsilon})$ for $\varepsilon = 1$, this is Case 1, so $T(n) = \Theta(n^{\log_b a}) = \Theta(n)$, which qualifies.
3. Option C: $T(n) = 2T(n/2) + n$. Master theorem with $a = 2,\ b = 2,\ f(n) = n$: $n^{\log_b a} = n$, and $f(n) = n = \Theta(n^{\log_b a})$, so this is Case 2, giving $T(n) = \Theta(n \log n)$; this is not $\Theta(n)$, so it does not qualify.
4. Option D: $T(n) = T(n-1) + n$. Unrolling: $T(n) = T(n-1) + n = T(n-2) + (n-1) + n = \cdots = T(1) + (2 + 3 + \cdots + n) = \Theta(n^2)$ (sum of an arithmetic series); this is not $\Theta(n)$, so it does not qualify.
5. Only A and B evaluate to $\Theta(n)$; C is $\Theta(n \log n)$ and D is $\Theta(n^2)$.

KEY IDEA

Solve each recurrence (by unrolling or the Master theorem) and check which reduce to $\Theta(n)$.

WATCH OUT FOR

Applying the Master theorem's Case 1 condition loosely and forgetting that a constant $f(n) = 1$ is $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$ (e.g. $\varepsilon = 1$), or confusing Case 1 (branching term dominates) with Case 2 (equal growth, giving an extra $\log n$ factor). This is exactly what makes option C a distractor for $\Theta(n)$.

Q.26 MSQ · 1 MARK

Discrete Mathematics › Sets & relations

Let R be a binary relation on the set $\{1, 2, \ldots, 10\}$, where $(x, y) \in R$ if the product of x and y is square of an integer. Which of the following properties is/are satisfied by R?

(A) Reflexive

(B) Symmetric

(C) Transitive

(D) Antisymmetric

Official answer: **A, B, C**

SOLUTION

1. $(x, y) \in R$ exactly when xy is the square of an integer.
2. Reflexive: $x \cdot x = x^2$ is a perfect square for every x, so $(x, x) \in R$ for all $x \in \{1, \ldots, 10\}$. True.
3. Symmetric: $xy = yx$, so if xy is a perfect square then so is yx. True.
4. Transitive: let $xy = a^2$ and $yz = b^2$ with a, b integers. Then $(xy)(yz) = y^2 \cdot xz = (ab)^2$, so $xz = \left(\frac{ab}{y}\right)^2$. Thus xz is an integer and the square of a rational number. A rational whose square is an integer must itself be an integer, so xz is the square of an integer and $(x, z) \in R$. True.
5. Equivalently, xy is a square exactly when x and y have the same square-free part, so R is an equivalence relation. For example, on $\{1, \ldots, 10\}$ the classes are $\{1, 4, 9\}$, $\{2, 8\}$, $\{3\}$, $\{5\}$, $\{6\}$, $\{7\}$, $\{10\}$.
6. Antisymmetric: $(1, 4) \in R$ since $1 \cdot 4 = 2^2$, and $(4, 1) \in R$, but $1 \neq 4$. So R is not antisymmetric. False.
7. Therefore R is reflexive, symmetric and transitive: options A, B and C.

KEY IDEA

xy is a perfect square exactly when x and y have the same square-free part, so R is an equivalence relation. It is therefore reflexive, symmetric and transitive, and it fails antisymmetry as soon as two distinct numbers share a class.

WATCH OUT FOR

Doubting transitivity because $xz = (ab/y)^2$ looks like a fraction. Since xz is an integer and a rational square, it is an integer square. Another slip is to treat a symmetric relation as antisymmetric because the pairs look 'ordered'. Here $(1, 4)$ and $(4, 1)$ are both in R.

Q.27 MSQ · 1 MARK

Engineering Mathematics › Calculus

For a real number a, let $I(a) = \int_{-1}^{1} (3x^2 - ax + 1)\, dx$. Which of the following statements is/are true?

(A) The value of $I(a)$ is independent of the value of a

(B) The value of $I(a)$ can vary with the value of a

(C) There exists $a \in (-\infty, +\infty)$ such that $I(a)$ is a positive real number

(D) There exists $a \in (-\infty, +\infty)$ such that $I(a)$ is a negative real number

Official answer: **A, C**

SOLUTION

1. Split the integral by linearity: $I(a) = \int_{-1}^{1} (3x^2 + 1)\, dx - a \int_{-1}^{1} x\, dx$.
2. The function x is odd and the interval $[-1, 1]$ is symmetric about 0, so $\int_{-1}^{1} x\, dx = 0$ and the term containing a vanishes.
3. Evaluate the remaining integral: $\int_{-1}^{1} (3x^2 + 1)\, dx = \left[x^3 + x\right]_{-1}^{1} = (1 + 1) - (-1 - 1) = 4$.
4. Hence $I(a) = 4$ for every real a, so $I(a)$ is a constant function of a.
5. (A) is true, because $I(a)$ does not depend on a. (B) is false, because $I(a)$ never varies with a.
6. (C) is true, because every a gives $I(a) = 4 > 0$, so such an a exists. (D) is false, because $I(a) = 4$ is never negative.

KEY IDEA

The integral of an odd function over a symmetric interval is zero, so the $-ax$ term contributes nothing and $I(a)$ is the constant 4.

WATCH OUT FOR

Treating $-ax$ as if it makes the integral depend on a. The integrand depends on a, but its integral over $[-1, 1]$ does not, because the odd part cancels. Another slip is forgetting that "there exists" in (C) is satisfied by any a once the value is the constant 4.

Q.28 MSQ · 1 MARK

Digital Electronics › Number systems

In a system, numbers are represented using 4-bit two's complement form. Consider four numbers N1 = 1011, N2 = 1101, N3 = 1010 and N4 = 1001 in the system. Which of the following operations will result in arithmetic overflow?

(A) N1 + N2

(B) N2 + N3

(C) N3 - N4

(D) N1 + N4

Official answer: **B, D**

SOLUTION

1. A 4-bit two's complement number lies in $[-2^3, 2^3 - 1] = [-8, 7]$. Overflow occurs exactly when the true result falls outside this range.

2. Decode each number, the sign bit having weight -8: $N1 = 1011 = -8 + 3 = -5$, $N2 = 1101 = -8 + 5 = -3$, $N3 = 1010 = -8 + 2 = -6$, $N4 = 1001 = -8 + 1 = -7$.
3. (A) $N1 + N2 = -5 + (-3) = -8$. This is inside $[-8, 7]$, so there is no overflow. Check: $1011 + 1101 = 1\,1000$, and dropping the carry-out gives $1000 = -8$, which is correct.
4. (B) $N2 + N3 = -3 + (-6) = -9 < -8$. Overflow. Check: $1101 + 1010 = 1\,0111$, which gives $0111 = +7$. Two negatives gave a positive, so the sign is wrong.
5. (C) $N3 - N4 = -6 - (-7) = +1$. This is inside the range, so there is no overflow. Check: $N3 + \overline{N4} + 1 = 1010 + 0110 + 1 = 1010 + 0111 = 1\,0001$, which gives $0001 = +1$, which is correct. Subtracting numbers of the same sign can never overflow, because the result has a smaller magnitude than the larger operand.
6. (D) $N1 + N4 = -5 + (-7) = -12 < -8$. Overflow. Check: $1011 + 1001 = 1\,0100$, which gives $0100 = +4$. Two negatives gave a positive, so the sign is wrong.
7. Overflow happens in (B) and (D).

KEY IDEA

Overflow in two's complement happens when the true sum lies outside $[-8, 7]$ for 4 bits. Adding two numbers of the same sign can overflow. Adding numbers of opposite sign, or subtracting numbers of the same sign, never can.

WATCH OUT FOR

Treating the carry-out of the sign bit as the overflow signal. In (A) a carry-out occurs but the result -8 is valid. The real test is whether the true value fits in the range, or equivalently whether two operands of the same sign give a result of the opposite sign. A second slip is treating the subtraction in (C) like an addition of two negatives.

Q.29 MSQ · 1 MARK

Theory of Computation › CFG & PDA

Which of the following grammars is/are ambiguous?

(A) $S \to aSb \mid \epsilon$

(B) $E \to E + E \mid E * E \mid id$

(C) $S \to aS \mid Sa \mid \epsilon$

(D) $S \to aS \mid \epsilon$

Official answer: **B, C**

SOLUTION

1. A grammar is ambiguous if some string in its language has two distinct parse trees (equivalently, two distinct leftmost derivations).
2. (A) $S \to aSb \mid \varepsilon$ generates $\{a^n b^n \mid n \geq 0\}$. In a string $a^n b^n$ with $n \geq 1$, the first symbol a and the last symbol b must come from the same use of $S \to aSb$, and the string is ε only when $n = 0$. So every step is forced, and each string has exactly one parse tree. Unambiguous.
3. (B) $E \to E + E \mid E * E \mid id$ fixes no precedence or associativity. For $id + id * id$, the root can be $E \to E + E$ with $id * id$ as the right child. It can also be $E \to E * E$ with $id + id$ as the left child. These are two different parse trees for the same string. Ambiguous.
4. (C) $S \to aS \mid Sa \mid \varepsilon$ generates a^*. For the string a, one derivation is $S \Rightarrow aS \Rightarrow a$ and another is $S \Rightarrow Sa \Rightarrow a$. In the first, the root has children a, S. In the second, the root has children S, a. These are two different parse trees. Ambiguous.
5. (D) $S \to aS \mid \varepsilon$ is right-linear and generates a^*. The string a^n can only be derived by applying $S \to aS$ exactly n times and then $S \to \varepsilon$. There is one parse tree per string. Unambiguous.
6. The ambiguous grammars are (B) and (C).

KEY IDEA

A grammar is ambiguous if one string has two parse trees. A production that can grow the string from either end, such as $aS \mid Sa$, or an operator rule with no precedence or associativity, such as $E \to E + E \mid E * E$, gives such a string. A grammar that is forced one way, such as aSb or right-linear aS, cannot be ambiguous.

WATCH OUT FOR

Judging ambiguity from the language rather than the grammar. a^* is a simple regular language, yet grammar (C) for it is ambiguous, while grammar (D) for the same language is not. Another slip is thinking $S \to aSb \mid \varepsilon$ is ambiguous because it has two productions. Having two productions does not make a grammar ambiguous. What matters is whether one string can be derived in two different ways.

Q.30 NAT · 1 MARK

Data Structures and Algorithms › Sorting & hashing

The keys 5, 28, 19, 15, 26, 33, 12, 17, 10 are inserted into a hash table using the hash function $h(k) = k \bmod 9$. The collisions are resolved by chaining. After all the keys are inserted, the length of the longest chain is _____. (answer in integer)

Official answer: **3**

SOLUTION

1. Compute h(k) = k mod 9 for each key in insertion order: 5→5, 28→1, 19→1, 15→6, 26→8, 33→6, 12→3, 17→8, 10→1.
2. Group the keys by their bucket index (chaining): bucket 1 = {28, 19, 10}; bucket 3 = {12}; bucket 5 = {5}; bucket 6 = {15, 33}; bucket 8 = {26, 17}; buckets 0, 2, 4, 7 are empty.
3. Compare chain lengths: bucket 1 has 3 keys, buckets 6 and 8 have 2 keys each, the rest have 1 or 0.
4. The longest chain has length 3 (bucket 1, containing 28, 19, 10).

KEY IDEA

Compute h(k) = k mod 9 for every key in insertion order, group keys by resulting bucket index (chaining), and report the size of the largest bucket: that size is the "longest chain," not the key values themselves.

WATCH OUT FOR

Miscomputing a modulus (e.g., treating 28 mod 9 or 19 mod 9 incorrectly) or forgetting that bucket indices only range 0 to 8 (table size 9), causing keys to be misplaced into the wrong bucket and the chain count to come out wrong.

Q.31 NAT · 1 MARK

Engineering Mathematics › Linear algebra

Consider the system of linear equations given below.

$$ax + y = b$$

$$16x + ay = 24$$

Suppose the values of a and b are chosen such that the system of linear equations produce multiple solutions. Then the product of a and b is _____. (answer in integer)

Official answer: **24**

SOLUTION

1. Multiple solutions require $\operatorname{rank}(A) = \operatorname{rank}([A \mid B]) < 2$. The two equations must therefore be proportional, coefficients and constants alike.
2. Write the system as $\begin{bmatrix} a & 1 \\ 16 & a \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 24 \end{bmatrix}$. Here $a \neq 0$, because $a = 0$ gives $\det A = -16 \neq 0$, which is a unique solution.
3. Proportional rows require $\frac{a}{16} = \frac{1}{a} = \frac{b}{24}$.

4. From $\frac{a}{16} = \frac{1}{a}$ we get $a^2 = 16$, so $a = 4$ or $a = -4$. Equivalently, $\det A = a^2 - 16 = 0$.
5. For $a = 4$: $\frac{b}{24} = \frac{1}{4}$, so $b = 6$. The second equation is then 4 times the first, and $ab = 24$.
6. For $a = -4$: $\frac{b}{24} = -\frac{1}{4}$, so $b = -6$. The second equation is then -4 times the first, and $ab = 24$.
7. Both cases give the same product, so $ab = 24$.

KEY IDEA

A 2x2 system has multiple solutions only when $\det A = 0$ and the augmented column is consistent, which means the two equations are proportional. Set $\det A = a^2 - 16 = 0$, then fix b from the constants.

WATCH OUT FOR

Setting only $\det A = 0$ and stopping at $a = \pm 4$, without choosing b to make the constants proportional. For any other b the system has no solution. Another slip is to take only $a = 4$ and miss the $a = -4$ case, though here both give the same product.

Q.32 NAT · 1 MARK

Data Structures and Algorithms › Sorting & hashing

Consider an array $A = [10, 7, 8, 19, 41, 35, 25, 31]$. Suppose the merge sort algorithm is executed on array A to sort it in increasing order. The merge sort algorithm will carry out a total of 7 merge operations.

A merge operation on sorted left array L and sorted right array R is said to be void if the output of the merge operation is the elements of array L followed by the elements of array R.

The number of void merge operations among these 7 merge operations is _____. (answer in integer)

Official answer: **3**

SOLUTION

1. Merge sort recursively halves A=[10,7,8,19,41,35,25,31] down to 8 singleton leaves in original order: [10],[7],[8],[19],[41],[35],[25],[31], then merges back up in 3 levels totaling 4+2+1=7 merges, matching the stem.
2. Level 1 merges (adjacent singleton pairs): merge([10],[7])→[7,10], but L followed by R=[10,7]≠[7,10] → NOT void. merge([8],[19])→[8,19]=L+R=[8,19] → VOID. merge([41],[35])→[35,41], L+R=[41,35]≠ → NOT void. merge([25],[31])→[25,31]=L+R → VOID.
3. Level 2 merges: merge([7,10],[8,19])→[7,8,10,19], L+R=[7,10,8,19]≠ → NOT void. merge([35,41],[25,31])→[25,31,35,41], L+R=[35,41,25,31]≠ → NOT void.
4. Final merge (level 3): merge([7,8,10,19],[25,31,35,41]): since max(L)=19<min(R)=25, the merged sorted output equals L followed by R exactly → VOID.
5. Total void merges = 2 (level 1) + 0 (level 2) + 1 (final) = 3.

KEY IDEA

In standard top-down merge sort, the array is recursively halved until singletons, then merged back up level by level. A merge of L and R is "void" exactly when every element of L is already $\leq$ every element of R (i.e. $\max(L) \leq \min(R)$), so the merged output equals L followed by R unchanged.

WATCH OUT FOR

Assuming a merge is void whenever L and R are each individually already sorted (true for every leaf-pair merge by definition) rather than checking whether L's elements are ALL less than R's elements; e.g. merge([10],[7]) has two "sorted" singletons but is NOT void since 10 > 7, so the output [7,10] differs from L followed by R = [10,7].

Q.33 NAT · 1 MARK

Computer Networks › IPv4

If an IP network uses a subnet mask of 255.255.240.0, the maximum number of IP addresses that can be

assigned to network interfaces is _____. (answer in integer)

Official answer: **4094**

SOLUTION

1. The mask is 255.255.240.0. In binary the third octet is $240 = 11110000_2$, and the first two octets are all ones.
2. Count the 1-bits in the mask: $8 + 8 + 4 = 20$, so the prefix length is /20.
3. The number of host bits is $32 - 20 = 12$.
4. The network has $2^{12} = 4096$ addresses in total.
5. Two of these cannot be assigned to interfaces: the all-zeros host address (the network address) and the all-ones host address (the directed broadcast address).
6. Assignable addresses $= 2^{12} - 2 = 4094$.

KEY IDEA

Count the 1-bits in the mask to get the prefix length, then take the host bits as 32 minus the prefix. The number of assignable addresses is $2^h - 2$, because the network and broadcast addresses are reserved.

WATCH OUT FOR

Answering 4096 (all 2^{12} addresses), which forgets that the network address and the broadcast address cannot be given to interfaces. Another slip is reading 240 as 4 ones in the wrong octet and getting a prefix other than /20.

Q.34 NAT · 1 MARK

Digital Electronics › Number systems

The 32-bit IEEE 754 single precision representation of a number is 0xC2710000. The number in decimal representation is _____. (rounded off to two decimal places)

Official answer: **-60.25**

SOLUTION

1. Write the hex word in binary: $\mathrm{C} = 1100,\ 2 = 0010,\ 7 = 0111,\ 1 = 0001$, and the other four hex digits are 0000. So 0xC2710000 is $1100\,0010\,0111\,0001\,0000\,0000\,0000\,0000$.
2. Split the 32 bits as 1 sign bit, 8 exponent bits and 23 fraction bits: $1 \mid 10000100 \mid 11100010000000000000000$.
3. IEEE 754 single precision gives the value $(-1)^s \times 1.f \times 2^{e-127}$, where e is the exponent field read as an unsigned number.
4. The sign bit is $s = 1$, so the number is negative.
5. The exponent field is $10000100_2 = 128 + 4 = 132$, so the true exponent is $132 - 127 = 5$.
6. The fraction begins 1110001 followed by zeros, so $1.f = 1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{128} = 1.8828125$.
7. Value $= -1.8828125 \times 2^5 = -60.25$, which is already exact to two decimal places.

KEY IDEA

Split the 32 bits as 1 sign, 8 exponent and 23 fraction bits. The value is $(-1)^s \times 1.f \times 2^{e-127}$, with the leading 1 of the mantissa implied.

WATCH OUT FOR

Regrouping the hex bits wrongly. The exponent field starts at the second bit, so it straddles the first two hex digits. Forgetting the implied leading 1, or the bias of 127, also gives a wrong value.

Q.35 NAT · 1 MARK

Compiler Design › Lexical analysis

A lexical analyzer uses the following token definitions

- $letter \rightarrow [A - Za - z]$
- $digit \rightarrow [0 - 9]$

- $id \rightarrow letter\ (letter \mid digit)^*$
- $number \rightarrow digit^+$
- $ws \rightarrow (blank \mid tab \mid newline)^+$

For the string given below,

`x1 23mm 78 y 7z zz5 14A 8H AaYcD`

the number of tokens (excluding ws) that will be produced by the lexical analyzer is _____. (answer in integer)

Official answer: **13**

SOLUTION

1. The analyzer always takes the **longest match** from the current position. An id must start with a letter and may then continue with letters or digits. A $number$ is a run of digits only.
2. White space is skipped, because ws is excluded from the count. Split the string at the blanks: `x1` , `23mm` , `78` , `y` , `7z` , `zz5` , `14A` , `8H` , `AaYcD` .
3. `x1` begins with a letter, so the whole of it is one id. Tokens: 1.
4. `23mm` begins with a digit, so it cannot be an id. The longest $number$ is `23` , because `m` is not a digit. Then `mm` is one id. Tokens: 2.
5. `78` is one $number$. Tokens: 1.
6. `y` is one id. Tokens: 1.
7. `7z` : the longest $number$ is `7` , then `z` is an id. Tokens: 2.
8. `zz5` begins with a letter, and a digit may follow inside an id, so it is one id. Tokens: 1.
9. `14A` : `14` is a $number$, then `A` is an id. Tokens: 2.
10. `8H` : `8` is a $number$, then `H` is an id. Tokens: 2.
11. `AaYcD` is all letters, so it is one id. Tokens: 1.
12. Total: $1+2+1+1+2+1+2+2+1=13$.

KEY IDEA

A lexer takes the longest match at each position. An identifier must start with a letter, while a number is digits only, so a lexeme that starts with a digit splits where the digits end.

WATCH OUT FOR

Treating a lexeme such as `23mm` or `7z` as a single token. An id cannot start with a digit, so each of these splits into a $number$ followed by an id. Another slip is splitting `zz5` into two tokens, when a digit is allowed inside an id after its first letter.

Q.36 MCQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

Consider a complete graph K_n with n vertices ($n > 4$). Note that multiple spanning trees can be constructed over K_n. Each of these spanning trees is represented as a set of edges. The Jaccard coefficient between any two sets is defined as the ratio of the size of the intersection of the two sets to the size of the union of the two sets.

Which one of the following options gives the lowest possible value for the Jaccard coefficient between any two spanning trees of K_n ?

(A) $\frac{1}{n}$

(B) $\frac{1}{2n-3}$

(C) 0

(D) $\dfrac{1}{n-1}$

Official answer: **C**

SOLUTION

1. Every spanning tree of K_n has exactly $n-1$ edges (a spanning tree on n vertices always has $n-1$ edges), so each spanning tree corresponds to an $(n-1)$-element subset of K_n's edge set.
2. The Jaccard coefficient of two such edge sets T_1, T_2 is $\dfrac{|E(T_1) \cap E(T_2)|}{|E(T_1) \cup E(T_2)|}$, which is always ≥ 0; it is minimized to exactly 0 precisely when T_1 and T_2 share no edge at all, i.e. when the two spanning trees are edge-disjoint.
3. So the question reduces to: can K_n (for $n > 4$) always supply two pairwise edge-disjoint spanning trees? K_n has $\binom{n}{2} = \dfrac{n(n-1)}{2}$ edges in total. Since each spanning tree needs $n-1$ of them, the number of pairwise edge-disjoint spanning trees that can possibly be packed into K_n is at most $\left\lfloor \dfrac{n(n-1)/2}{n-1} \right\rfloor = \left\lfloor \dfrac{n}{2} \right\rfloor$.
4. This bound is a classical tight result for complete graphs: K_n can always be decomposed into (or shown to contain) $\left\lfloor \dfrac{n}{2} \right\rfloor$ pairwise edge-disjoint spanning trees; the bound is attained, not merely an upper limit. This also follows from the Nash-Williams/Tutte spanning-tree-packing theorem, since in K_n the trivial vertex partition (no merging) gives the binding constraint.
5. For $n > 4$ (i.e. $n \geq 5$), $\left\lfloor \dfrac{n}{2} \right\rfloor \geq 2$, so at least 2 pairwise edge-disjoint spanning trees of K_n always exist. Pick any such pair T_1, T_2.
6. For this pair, $|E(T_1) \cap E(T_2)| = 0$, so their Jaccard coefficient equals $\dfrac{0}{2(n-1)} = 0$.
7. Since the Jaccard coefficient can never be negative, and the value 0 is shown to be attainable for every $n > 4$, the lowest possible value of the Jaccard coefficient between any two spanning trees of K_n is 0: option (C).

KEY IDEA

The Jaccard coefficient of two spanning-tree edge sets is minimized (to 0) exactly when two edge-disjoint spanning trees of K_n can be found; the maximum number of pairwise edge-disjoint spanning trees packable into K_n is the classical result $\left\lfloor \dfrac{n}{2} \right\rfloor$, which is at least 2 for every $n > 4$.

WATCH OUT FOR

Assuming any two spanning trees of K_n must share at least one edge (confusing "distinct" with "edge-disjoint"), or forgetting that the classical spanning-tree-packing bound $\lfloor n/2 \rfloor$ for K_n is tight (achievable), not just an upper bound, leading test-takers to pick a small nonzero value like $\dfrac{1}{2n-3}$ instead of 0.

Q.37 MCQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

Let G be a weighted directed acyclic graph with m edges and n vertices. Given G and a source vertex s in G, which one of the following options gives the worst case time complexity of the fastest algorithm to find the lengths of shortest paths from s to all vertices that are reachable from s in G?

(A) $\Theta(m+n)$

(B) $\Theta(m+n\log(n))$

(C) $\Theta(nm)$

(D) $\Theta(n^3)$

Official answer: **A**

SOLUTION

1. G is a weighted directed acyclic graph (DAG), so a topological ordering of its n vertices always exists and can be computed in Θ(m+n) time using DFS with post-order reversal, or Kahn's algorithm (repeatedly removing in-degree-0 vertices).
2. Initialize dist[s] = 0 and dist[v] = ∞ for every other vertex v.
3. Process the vertices in topological order. When a vertex u is dequeued, dist[u] is already finalized to its true shortest-path length, because every edge into u comes from a vertex earlier in the topological order, and all such vertices have already been processed and relaxed.
4. For each vertex u processed, relax every outgoing edge (u, v): if dist[u] + w(u, v) < dist[v], set dist[v] = dist[u] + w(u, v).
5. Each of the n vertices is processed exactly once, and each of the m edges is relaxed exactly once (when its tail vertex is processed); there is no repeated relaxation, unlike in Bellman-Ford. So the relaxation phase costs Θ(n) (vertex visits) + Θ(m) (edge relaxations) = Θ(m+n).
6. Adding the Θ(m+n) cost of computing the topological order gives a total worst-case running time of Θ(m+n) + Θ(m+n) = Θ(m+n).
7. This is asymptotically optimal: any algorithm must read the entire input graph (all m edges and n vertices), which already takes Ω(m+n) time, so no algorithm can beat Θ(m+n) in the worst case.
8. Comparing with the other options confirms why they are wrong here: Θ(m + n log n) (B) is Dijkstra's complexity with a binary heap, needed only for general graphs with non-negative weights (no topological order available); Θ(nm) (C) is Bellman-Ford's complexity, needed for general graphs that may contain cycles; Θ(n^3) (D) is the complexity of all-pairs algorithms like Floyd-Warshall, which solve a different (harder) problem. Since G is guaranteed acyclic, the topological-order relaxation method is strictly faster than all of these, giving Θ(m+n) as the correct worst-case complexity of the fastest algorithm.
9. Therefore the correct option is (A) Θ(m+n).

KEY IDEA
For a weighted DAG, computing a topological order (Θ(m+n) via DFS or Kahn's algorithm) and then relaxing edges in that order gives correct single-source shortest paths in one linear pass: no priority queue or repeated relaxation is needed, unlike in general graphs.

WATCH OUT FOR
Assuming the graph is a general weighted graph (with cycles/negative or arbitrary edges) and reaching for Dijkstra's Θ(m + n log n) (option B) or Bellman-Ford's Θ(nm) (option C), forgetting that acyclicity permits a much faster topological-order relaxation algorithm.

Q.38 MCQ · 2 MARKS

Data Structures and Algorithms › Complexity

Consider an array A of integers of size n. The indices of A run from 1 to n. An algorithm is to be designed to check whether A satisfies the condition given below.

$$\forall i, j \in \{1, \ldots, n-1\} \text{ such that } i > j,\ (A[i+1] - A[i]) > (A[j+1] - A[j])$$

Which one of the following gives the worst case time complexity of the fastest algorithm that can be designed for the problem?

(A) $\Theta(n)$

(B) $\Theta(\log(n))$

(C) $\Theta(n \log(n))$

(D) $\Theta(n^2)$

Official answer: **A**

SOLUTION

1. Let $d[k] = A[k+1] - A[k]$ for $k = 1, \ldots, n-1$ (the sequence of consecutive differences of A).
2. The given condition says: for every pair $i > j$ in $\{1, \ldots, n-1\}$, $d[i] > d[j]$. This is exactly the statement that $d[1] < d[2] < \cdots < d[n-1]$, i.e. d is strictly increasing.
3. Strict inequality ($<$) is transitive, so it suffices to check only the $n-2$ consecutive pairs $d[1] < d[2],\ d[2] < d[3],\ \ldots,\ d[n-2] < d[n-1]$: if every adjacent pair satisfies $<$, then by transitivity every non-adjacent pair with $i > j$ also satisfies $d[i] > d[j]$.
4. Algorithm (upper bound): scan the array once left to right, computing each $A[k+1] - A[k]$ in $O(1)$ time and comparing it with the previous difference. This does $O(n)$ arithmetic operations and $O(n)$ comparisons in total, so the condition can be checked in $O(n)$ time.
5. Lower bound: whether A satisfies the condition depends on every entry of A: changing any single unread entry can change some $d[k]$ and flip the answer. So any correct algorithm must inspect all n entries in the worst case, giving $\Omega(n)$.
6. Since $O(n)$ (achievable) meets $\Omega(n)$ (necessary), the worst-case time complexity of the fastest possible algorithm is $\Theta(n)$: option (A).

KEY IDEA
A condition holding for every pair of indices $i > j$ (not just adjacent ones) is a disguised way of saying a derived sequence is strictly monotonic; strict monotonicity only ever needs to be checked between consecutive terms, because strict inequality is transitive.

WATCH OUT FOR
Assuming that a condition quantified over ALL pairs $i > j$ requires checking all $\binom{n-1}{2}$ pairs directly, leading to a wrongly assumed $O(n^2)$ (or, worse, mistaking the transitivity shortcut for something that needs sorting, leading to a wrongly assumed $O(n \log n)$); both miss that strict inequality between all pairs collapses to strict monotonicity, which is checkable with one linear pass over consecutive terms.

Q.39 MCQ · 2 MARKS

Data Structures and Algorithms › Design techniques

Consider a table T, where the elements $T[i][j]$, $0 \le i, j \le n$, represent the cost of the optimal solutions of different subproblems of a problem that is being solved using a dynamic programming algorithm. The recursive formulation to compute the table entries is as follows:

$$T[0][k] = T[k][0] = 1 \quad \text{for } k = 0, 1, 2, \ldots, n$$

$$T[i][j] = 2T[i-1][j] + 3T[i][j-1] \quad \text{for } 1 \le i, j \le n$$

Consider the following two algorithms to compute entries of T. Assume that for both the algorithms, for all $0 \le i, j \le n$, $T[i][j]$ has been initialized to 1.

Algorithm B_1:

```
For i = 1, 2, ..., n
  For j = 1, 2, ..., n
    T[i][j] = 2T[i-1][j] + 3T[i][j-1]
```

Algorithm B_2:

```
For s = 2, 3, ..., 2n
  For i = 1, 2, ..., n
    For j = 1, 2, ..., n
      If (i + j == s)
        T[i][j] = 2T[i-1][j] + 3T[i][j-1]
```

Algorithm B_k, $k \in \{1, 2\}$ is said to be correct if and only if it calculates the correct values of $T[i][j]$, for all $0 \leq i, j \leq n$, (as per the recursive formulation) at the end of the execution of the algorithm B_k.

Which one of the following statements is true?

(A) Both algorithms B_1 and B_2 are correct

(B) Algorithm B_1 is correct, but algorithm B_2 is incorrect

(C) Algorithm B_2 is correct, but algorithm B_1 is incorrect

(D) Both algorithms B_1 and B_2 are incorrect

Official answer: **A**

SOLUTION

1. The recurrence for interior cells is $T[i][j] = 2T[i-1][j] + 3T[i][j-1]$ for $1 \leq i, j \leq n$, and every cell with $i = 0$ or $j = 0$ is a boundary value fixed at 1 by $T[0][k] = T[k][0] = 1$. Neither algorithm's loop body ever touches a boundary cell (both inner loops range over i, j from 1 to n), so the boundary values stay correct throughout.
2. **Algorithm** B_1 (row-major: $i = 1, \ldots, n$ outer, $j = 1, \ldots, n$ inner). Fix a row i. By induction on j: $T[i][0] = 1$ is the correct boundary value and is never overwritten before $T[i][1]$ is computed. When $T[i][j]$ is computed, $T[i][j-1]$ was computed earlier in the same row's j-loop (since $j - 1 < j$) and is therefore already correct by the induction hypothesis, and $T[i-1][j]$ was computed and finalized during the previous row's iteration (row $i - 1$ is fully completed before row i starts) and is also correct. So every $T[i][j]$ is computed from two already-correct values. By induction over i (rows) and j (columns within a row), B_1 computes every entry correctly.
3. **Algorithm** B_2 (anti-diagonal: outer loop $s = 2, \ldots, 2n$, inner loops scan all (i, j) and update those with $i + j = s$). A cell (i, j) with $i + j = s$ depends only on $(i - 1, j)$, whose index sum is $s - 1$, and $(i, j - 1)$, whose index sum is also $s - 1$. Every cell with index sum $s - 1$ is updated during a strictly earlier outer iteration ($s - 1 < s$) and is therefore already correct by induction on s (base case $s = 2$: dependencies have sum 1, i.e. $T[0][1]$ or $T[1][0]$, both untouched boundary values equal to 1).
4. Within one fixed s, the cells satisfying $i + j = s$ (e.g., for $s = 4$: $(1, 3), (2, 2), (3, 1)$) never depend on each other: each one's two dependencies always lie on diagonal $s - 1$, never on diagonal s itself. So the order in which i, j are scanned inside a fixed s is irrelevant to correctness; every cell on diagonal s is computed only from already-finalized diagonal-$(s - 1)$ (or boundary) values. By induction over s, B_2 computes every entry correctly.
5. Both B_1's row-major order and B_2's anti-diagonal order are valid topological orders of the dependency DAG in which $T[i][j]$ depends only on $T[i-1][j]$ and $T[i][j-1]$ (each strictly decreasing $i + j$ by 1). Neither algorithm ever reads a cell before the cell it depends on has been finalized, so both algorithms are correct.
6. This matches option (A): both algorithms B_1 and B_2 are correct.

KEY IDEA

Whenever a DP recurrence's dependencies always come from strictly "smaller" cells in some measure (here, $i + j$ strictly decreases by 1 at each dependency), any loop order that is a valid topological order of that dependency structure computes the table correctly: row-major and anti-diagonal ("wavefront") order are both such valid orders here, since each cell's two dependencies fall on the immediately preceding row/column or the immediately preceding diagonal, never on the same diagonal.

WATCH OUT FOR

Assuming that scanning order only matters when it visits cells "out of the usual row/column order," and therefore either (a) wrongly flagging B_2 as incorrect on the intuition that its triple-nested loop with an `if` condition looks unusual or wasteful, without checking that its dependencies are actually satisfied, or (b) failing to verify that cells sharing the same diagonal ($i + j = s$) never depend on each other, which is the fact that makes the inner scan order inside a fixed s harmless.

Q.40 MCQ · 2 MARKS

Digital Electronics › Boolean logic

Consider the following 4-variable Boolean function

$$F(A, B, C, D) = \sum m(0, 1, 2, 3, 8, 9, 10, 11)$$

Consider A as MSB, D as LSB. Which one of the following options represents the minimal sum of products form for the above function?

Note: $+$ is OR operation, $\cdot$ is AND operation, $'$ is NOT operation

(A) $\overline{A} + \overline{B} + \overline{C} + \overline{D}$

(B) $\overline{B}$

(C) $\overline{A} \cdot \overline{B} + A \cdot B$

(D) $\overline{A}$

Official answer: **B**

SOLUTION

1. Write each minterm as $ABCD$: $0 = 0000,\ 1 = 0001,\ 2 = 0010,\ 3 = 0011,\ 8 = 1000,\ 9 = 1001,\ 10 = 1010,\ 11 = 1011$.
2. In all eight minterms $B = 0$, while A, C and D take every combination of values. The function is therefore 1 exactly when $B = 0$: the eight listed minterms are precisely the eight input combinations with $B = 0$.
3. K-map (rows AB, columns CD in Gray order 00, 01, 11, 10):

$AB \backslash CD$	00	01	11	10
00	1	1	1	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

4. Rows 00 and 10 are adjacent on the map (the top and bottom edges wrap around), so all eight 1s form one group of 8, which cannot be enlarged because every other cell is 0.
5. In that group A, C and D each take both values, and only $B = 0$ stays fixed, so the group is $\overline{B}$.
6. Algebraic check: $F = \overline{A}\,\overline{B}(\overline{C}\,\overline{D} + \overline{C}D + CD + C\overline{D}) + A\overline{B}(\overline{C}\,\overline{D} + \overline{C}D + CD + C\overline{D})$. The bracket equals 1, so $F = \overline{A}\,\overline{B} + A\overline{B} = \overline{B}(\overline{A} + A) = \overline{B}$.
7. Truth-table check of the other options: (A) is 1 for the input $A = 1, B = 1, C = 1, D = 0$ where $F = 0$; (C) gives 0 at minterm 8 ($A = 1, B = 0$) where $F = 1$; (D) is 0 at minterm 8 where $F = 1$. Only $\overline{B}$ matches F on all 16 inputs.
8. The minimal sum of products is $F = \overline{B}$, a single literal, which is option (B).

KEY IDEA

When the minterm list is exactly every combination of some variables with the remaining variable fixed, the function collapses to that single literal. Here the eight minterms are all inputs with $B = 0$, so $F = \overline{B}$.

WATCH OUT FOR

Picking (D) $\overline{A}$ because the first four minterms (0 to 3) have $A = 0$, and forgetting that minterms 8 to 11 have $A = 1$. Another slip is missing that the top and bottom K-map rows are adjacent, so the two blocks of four merge into one group of eight.

Q.41 MCQ · 2 MARKS

Compiler Design › Parsing

Consider the canonical $LR(0)$ parsing of the grammar below using terminals $\{a, b, c\}$ and non-terminals $\{A, B, C, S\}$ with S as the start symbol.

$$S \to ACB$$

$$A \to aA \mid \epsilon$$

$$C \to cC \mid \epsilon$$

$$B \to bB \mid b$$

Which one of the following options gives the number of shift-reduce conflicts that will occur in the $LR(0)$ ACTION table?

(A) 2

(B) 3

(C) 4

(D) 5

Official answer: **D**

SOLUTION

1. Augment the grammar with $S' \to S$. In an $LR(0)$ automaton a complete item $A \to \alpha\cdot$ puts a reduce entry in every terminal column. A shift-reduce conflict therefore occurs in a state for each terminal t that has an item $\cdot\, t$ in that state alongside a complete item.
2. $I_0 = \text{closure}(S' \to \cdot S) = \{S' \to \cdot S,\ S \to \cdot ACB,\ A \to \cdot aA,\ A \to \cdot\}$. The state shifts on a and also reduces $A \to \varepsilon$ on a. This is 1 conflict, on a.
3. $\text{goto}(I_0, a) = I_a = \{A \to a \cdot A,\ A \to \cdot aA,\ A \to \cdot\}$. It shifts on a and reduces $A \to \varepsilon$ on a. This is 1 conflict, on a. Also $\text{goto}(I_a, a) = I_a$, and $\text{goto}(I_a, A) = \{A \to aA\cdot\}$, which has no shift item.
4. $\text{goto}(I_0, A) = I_A = \{S \to A \cdot CB,\ C \to \cdot cC,\ C \to \cdot\}$. It shifts on c and reduces $C \to \varepsilon$ on c. This is 1 conflict, on c.
5. $\text{goto}(I_A, C) = \{S \to AC \cdot B,\ B \to \cdot bB,\ B \to \cdot b\}$. It has no complete item, so no conflict.
6. $\text{goto}(I_A, c) = I_c = \{C \to c \cdot C,\ C \to \cdot cC,\ C \to \cdot\}$. It shifts on c and reduces $C \to \varepsilon$ on c. This is 1 conflict, on c. Also $\text{goto}(I_c, c) = I_c$, and $\text{goto}(I_c, C) = \{C \to cC\cdot\}$, which has no shift item.
7. From $\{S \to AC \cdot B, \ldots\}$, $\text{goto}(\cdot, b) = I_b = \{B \to b \cdot B,\ B \to b\cdot,\ B \to \cdot bB,\ B \to \cdot b\}$. It shifts on b and reduces $B \to b$ on b. This is 1 conflict, on b. Also $\text{goto}(I_b, b) = I_b$.
8. The remaining states are $\{B \to bB\cdot\}$, $\{S \to ACB\cdot\}$ and $\{S' \to S\cdot\}$. Each has a complete item and no shift item, so none has a conflict.
9. Total: $1 + 1 + 1 + 1 + 1 = 5$ shift-reduce conflicts, which is option (D).

KEY IDEA

In $LR(0)$, a complete item reduces on every terminal. So a state with a complete item and an item $\cdot t$ conflicts on t. Here the ε-productions $A \to \varepsilon$ and $C \to \varepsilon$ and the production $B \to b$ (beside $B \to bB$) each create such a state.

WATCH OUT FOR

Counting only states, or only the first state of each loop, and missing that the looping states I_a and I_c conflict again. Another slip is to count a state that has several reduce entries as one conflict per entry rather than one per terminal.

Q.42 MCQ · 2 MARKS

Database Management and Warehousing › Normal forms

In the context of schema normalization in relational DBMS, consider a set F of functional dependencies. The set of all functional dependencies implied by F is called the closure of F. To compute the closure of F, Armstrong's Axioms can be applied. Consider X, Y, and Z as sets of attributes over a relational schema. The three rules of Armstrong's Axioms are described as follows.

Reflexivity: If Y $\subseteq$ X, then X → Y
Augmentation: If X → Y, then XZ → YZ for any Z
Transitivity: If X → Y and Y → Z, then X → Z

The additional rule of Union is defined as follows.
Union: If X → Y and X → Z, then X → YZ

It can be proved that the additional rule of Union is also implied by the three rules of Armstrong's Axioms. Listed below are four combinations of these three rules. Which one of these combinations is both necessary and sufficient for the proof?

(A) Reflexivity, Augmentation, and Transitivity

(B) Reflexivity and Augmentation

(C) Transitivity

(D) Augmentation and Transitivity

Official answer: **D**

SOLUTION

1. Given $X \to Y$ and $X \to Z$, we must derive $X \to YZ$.
2. Augment $X \to Z$ with X: $XX \to XZ$, that is $X \to XZ$. (Augmentation)
3. Augment $X \to Y$ with Z: $XZ \to YZ$. (Augmentation)
4. Apply transitivity to $X \to XZ$ and $XZ \to YZ$: $X \to YZ$. (Transitivity)
5. This derivation uses only Augmentation and Transitivity, so these two are sufficient and Reflexivity is not needed. Options A and B are therefore not 'necessary' combinations, since each includes Reflexivity or lacks Transitivity.
6. Augmentation is necessary. Without it, the only rules left are Reflexivity (which gives trivial dependencies) and Transitivity (which only chains dependencies already present). Starting from $X \to Y$ and $X \to Z$, with Y, Z and X arbitrary disjoint attribute sets, these can never create a dependency with YZ on the right.
7. Transitivity is necessary. With only Reflexivity and Augmentation, every dependency obtained from $X \to Y$ and $X \to Z$ has the form $XW \to YW$ or $XW \to ZW$ (or a trivial one). To have the left side exactly X, W must lie inside X, so the right side is Y or Z together with part of X. It never contains both Y and Z, so $X \to YZ$ is not reached. The step that joins $X \to XZ$ to $XZ \to YZ$ needs Transitivity.
8. So the combination that is both necessary and sufficient is Augmentation and Transitivity: option (D).

KEY IDEA
Union is derived by augmenting $X \to Z$ with X to get $X \to XZ$, augmenting $X \to Y$ with Z to get $XZ \to YZ$, then chaining the two by transitivity. Reflexivity never appears.

WATCH OUT FOR
Choosing (A) because the textbook calls the three axioms together 'complete'. The question asks which rules the proof of Union needs, and Reflexivity is not one of them.

Q.43 MCQ · 2 MARKS

Computer Networks › Data link & MAC

Consider the transmission of data bits 110001011 over a link that uses Cyclic Redundancy Check (CRC) code for error detection. If the generator bit pattern is given to be 1001, which one of the following options shows the remainder bit pattern appended to the data bits before transmission?

(A) 011

(B) 101

(C) 000

(D) 100

Official answer: **D**

SOLUTION

1. The generator 1001 is $x^3 + 1$, of degree 3, so append 3 zeros to the data: $110001011\,000$ (12 bits).
2. Divide 110001011000 by 1001 using modulo-2 (XOR) division. Take the first four bits: $1100 \oplus 1001 = 0101$, partial remainder 101.
3. Bring down bit 5 (0): $1010 \oplus 1001 = 0011$, partial remainder 011.
4. Bring down bit 6 (1): 0111 has leading bit 0, so no XOR; partial remainder 111. Bring down bit 7 (0): $1110 \oplus 1001 = 0111$, partial remainder 111.
5. Bring down bit 8 (1): $1111 \oplus 1001 = 0110$, partial remainder 110. Bring down bit 9 (1): $1101 \oplus 1001 = 0100$, partial remainder 100.
6. Bring down bit 10 (0): $1000 \oplus 1001 = 0001$, partial remainder 001. Bring down bit 11 (0): 0010 has leading bit 0, partial remainder 010.
7. Bring down bit 12 (0): 0100 has leading bit 0, so the final remainder is 100.
8. Cross-check with polynomials: the shifted data is $x^{11} + x^{10} + x^6 + x^4 + x^3$. Since $x^3 \equiv 1 \pmod{x^3 + 1}$, this reduces to $x^2 + x + 1 + x + 1 = x^2$, which is 100.
9. The remainder 100 is appended, so the transmitted frame is 110001011100. This is option (D).

KEY IDEA
A generator of degree r gives an r-bit remainder. Append r zeros to the data, divide by the generator using modulo-2 (XOR) division, and the remainder is the CRC. With $1001 = x^3 + 1$ you can also use $x^3 \equiv 1$ to reduce the shifted data polynomial directly.

WATCH OUT FOR
Forgetting to append 3 zeros before dividing, or skipping a bit while bringing bits down. Another slip is XOR-ing when the leading bit of the partial remainder is 0, when it should be shifted with no XOR.

Q.44 MCQ · 2 MARKS

Computer Organization and Architecture › Instructions & addressing

Consider a processor that has 16 general purpose registers and it uses 2-byte instruction format for all its instructions. Variable-sized opcodes are permitted. There are three different types of instructions; M-type, R-type, and C-type. Each M-type instruction has 2 register operands and a 6-bit immediate operand. Each R-type instruction has 3 register operands. Each C-type instruction has a register operand and a 6-bit offset value. If there are 2 unique M-type opcodes and 7 unique R-type opcodes, which one of the following options gives the maximum number of unique opcodes possible for C-type instructions?

(A) 8

(B) 4

(C) 64

(D) 16

Official answer: **B**

SOLUTION

1. Registers: 16 GPRs ⇒ each register operand needs $\log_2 16 = 4$ bits.
2. Instruction length is fixed at 2 bytes = 16 bits for every type.
3. M-type: operand bits $= 2 \times 4 + 6 = 14$ ⇒ opcode length $= 16 - 14 = 2$ bits.
4. R-type: operand bits $= 3 \times 4 = 12$ ⇒ opcode length $= 16 - 12 = 4$ bits.
5. C-type: operand bits $= 4 + 6 = 10$ ⇒ opcode length $= 16 - 10 = 6$ bits.
6. Because the decoder must identify the instruction type from the opcode bits alone (before knowing how many operand bits follow), the opcodes of all three types together must form a prefix-free set: no opcode may be a prefix of another.
7. Model this with a binary tree of depth 6 (the longest opcode length), giving $2^6 = 64$ leaves in total.
8. Each 2-bit M-type opcode, extended to depth 6, blocks $2^{6-2} = 16$ leaves; 2 such opcodes block $2 \times 16 = 32$ leaves.
9. Each 4-bit R-type opcode blocks $2^{6-4} = 4$ leaves; 7 such opcodes block $7 \times 4 = 28$ leaves.
10. Leaves consumed by M-type and R-type together: $32 + 28 = 60$.
11. Leaves remaining at depth 6: $64 - 60 = 4$.
12. Each C-type opcode is already 6 bits (full depth), so it occupies exactly 1 leaf ⇒ maximum unique C-type opcodes $= 4$, matching option (B).

KEY IDEA

Variable-length opcodes that must be uniquely decodable from a fixed 16-bit instruction behave exactly like a prefix-free (Huffman) code: the set of opcodes, extended to a common comparison depth, must occupy disjoint blocks of a binary code tree (Kraft's inequality). Count how many leaves of that tree the already-fixed M-type and R-type opcodes use up, and the remainder bounds the C-type opcode count.

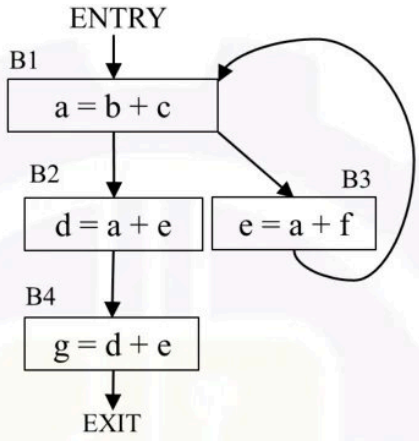
WATCH OUT FOR

Treating each instruction type's leftover opcode bits independently (e.g., concluding C-type has $2^6 = 64$ possible opcodes) without imposing the prefix-free condition that lets a single opcode-decoder distinguish M-, R-, and C-type instructions from the same 16-bit word; this ignores that shorter opcodes (M-type, R-type) consume large blocks of the shared code space that longer C-type opcodes cannot then use.

Q.45 MCQ · 2 MARKS

Compiler Design › Optimisation

Q.45 Consider the control flow graph given below.



Which one of the following options is the set of live variables at the exit point of each basic block?

(A) B1:{a, b, c, e, f}, B2:{d, e}, B3:{b, c, e, f}, B4:∅

(B) B1:∅, B2:{d, e}, B3:{a, c, f}, B4:∅

(C) B1:{a, b, c, e, f}, B2:{d, e}, B3:{c, e, f}, B4:∅

(D) B1:∅, B2:{d, e, f}, B3:{a, b, c, e, f}, B4:∅

Official answer: **A**

SOLUTION

1. A variable is live at a point if its current value may be used on some path before it is redefined. Liveness runs backward: $\mathrm{OUT}(B) = \bigcup_{S \in \mathrm{succ}(B)} \mathrm{IN}(S)$ and $\mathrm{IN}(B) = \mathrm{use}(B) \cup (\mathrm{OUT}(B) - \mathrm{def}(B))$.
2. Block use and def sets: B1 use $\{b, c\}$, def $\{a\}$. B2 use $\{a, e\}$, def $\{d\}$. B3 use $\{a, f\}$, def $\{e\}$. B4 use $\{d, e\}$, def $\{g\}$. Successors: B1 to B2 and B3, B2 to B4, B3 to B1, B4 to EXIT.
3. B4: $\mathrm{OUT}(B4) = \varnothing$ because nothing is used after EXIT. So $\mathrm{IN}(B4) = \{d, e\}$.

4. B2: $\mathrm{OUT}(B2) = \mathrm{IN}(B4) = \{d, e\}$, and $\mathrm{IN}(B2) = \{a, e\} \cup (\{d, e\} - \{d\}) = \{a, e\}$.
5. B1 and B3 form a loop, so iterate from empty sets. Start with $\mathrm{IN}(B1) = \varnothing$. Then $\mathrm{OUT}(B3) = \varnothing$ and $\mathrm{IN}(B3) = \{a, f\}$.
6. Then $\mathrm{OUT}(B1) = \mathrm{IN}(B2) \cup \mathrm{IN}(B3) = \{a, e, f\}$ and $\mathrm{IN}(B1) = \{b, c\} \cup (\{a, e, f\} - \{a\}) = \{b, c, e, f\}$.
7. Second pass: $\mathrm{OUT}(B3) = \mathrm{IN}(B1) = \{b, c, e, f\}$, so $\mathrm{IN}(B3) = \{a, f\} \cup (\{b, c, e, f\} - \{e\}) = \{a, b, c, f\}$.
8. Then $\mathrm{OUT}(B1) = \{a, e\} \cup \{a, b, c, f\} = \{a, b, c, e, f\}$ and $\mathrm{IN}(B1) = \{b, c\} \cup \{b, c, e, f\} = \{b, c, e, f\}$. This is unchanged, so a fixed point is reached.
9. Live at block exits: B1 $\{a, b, c, e, f\}$, B2 $\{d, e\}$, B3 $\{b, c, e, f\}$, B4 $\varnothing$. This is option A.

KEY IDEA

Live-variable analysis is a backward data-flow problem. Compute OUT as the union of the successors' IN sets, and IN = use ∪ (OUT − def). Where blocks form a loop, iterate from empty sets until nothing changes.

WATCH OUT FOR

Reporting the live-in sets instead of the live-out sets. Another slip is stopping after one pass over the B1–B3 loop, which leaves B3's exit set short of {b, c, e, f}. A third is forgetting that B3 redefines e, so e is removed from IN(B3) and enters OUT(B1) only through B2's use of e.

Q.46 MCQ · 2 MARKS

Database Management and Warehousing › Indexing

An index in a DBMS is said to be dense if an index entry appears for every search-key value in the indexed file. Otherwise it is called a sparse index. Consider the following two statements.

S1: A hash index must be a dense index

S2: A B^+ tree index can be a sparse index

Which one of the following options is correct?

(A) Both S1 and S2 are true

(B) Both S1 and S2 are false

(C) S1 is true and S2 is false

(D) S1 is false and S2 is true

Official answer: **A**

SOLUTION

1. A sparse index keeps entries for only some search-key values, usually one per data block of a file stored in search-key order. To search, find the largest index entry not exceeding the key, then scan forward in the sorted file from there.
2. S1: A hash index maps a key to a bucket through the hash function. Hashing keeps no order, so a key value with no entry of its own cannot be reached by scanning forward from a neighbouring entry. Every search-key value therefore needs its own entry, so a hash index must be dense. S1 is true.
3. S2: A B^+ tree index can be built as a sparse (clustering) index over a file stored in search-key order. Its leaf level then holds one entry per data block instead of one per key value. A search follows the tree to the right block and scans within it. Such an index is sparse, so S2 is true.
4. Both statements are true, so the answer is option (A).

KEY IDEA

A sparse index needs the data file sorted on the search key, so that it can scan forward from the nearest smaller entry. A hash index has no such order, so it must be dense. A B+ tree index over a sorted file can be sparse, with one leaf entry per block.

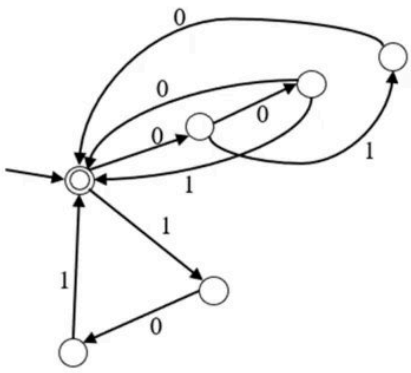
WATCH OUT FOR

Taking "B+ tree index" to mean "always dense", which forgets that a clustering B+ tree over a sorted file can be sparse. The opposite slip is believing a hash index can be sparse.

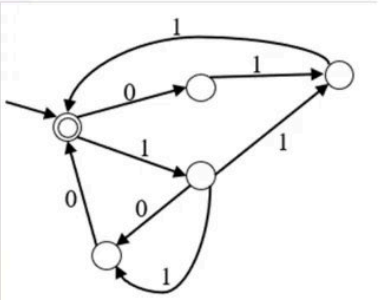
Q.47 MSQ · 2 MARKS

Theory of Computation › Finite automata

Q.47 Consider the following two finite automata D_1 and D_2.



D_1



D_2

Which of the following statements is/are true?

(A)	$L(D_1) = L(D_2)$
(B)	$L(D_1)$ is a proper subset of $L(D_2)$
(C)	$L(D_1) \cap L(D_2) = \{\epsilon\}$
(D)	$\left(L(D_1) \cup L(D_2)\right)^*$ consists of all strings in $\{0,1\}^*$ whose length is divisible by 3

Official answer: **C, D**

SOLUTION

1. In each automaton the start state is the only accepting state, and every walk that leaves it returns to it after exactly 3 transitions (the other states have no edges among themselves except along these 3-step loops). So $L(D_i) = S_i^*$, where S_i is the set of labels of the length-3 loops through the start state.
2. D_1 loops: $q_0 \xrightarrow{0} b \xrightarrow{0} a \xrightarrow{0} q_0$ gives 000; the same path ending with the edge $a \xrightarrow{1} q_0$ gives 001; $q_0 \xrightarrow{0} b \xrightarrow{1} c \xrightarrow{0} q_0$ gives 010; $q_0 \xrightarrow{1} d \xrightarrow{0} e \xrightarrow{1} q_0$ gives 101. So $S_1 = \{000, 001, 010, 101\}$.

3. D_2 loops: $p_0 \xrightarrow{0} f \xrightarrow{1} g \xrightarrow{1} p_0$ gives 011; $p_0 \xrightarrow{1} h \xrightarrow{1} g \xrightarrow{1} p_0$ gives 111; $p_0 \xrightarrow{1} h \xrightarrow{0} i \xrightarrow{0} p_0$ gives 100; $p_0 \xrightarrow{1} h \xrightarrow{1} i \xrightarrow{0} p_0$ gives 110. So $S_2 = \{011, 111, 100, 110\}$.
4. $S_1 \cap S_2 = \varnothing$ and $|S_1| + |S_2| = 8$, so $S_1 \cup S_2 = \{0,1\}^3$.
5. (A) and (B) are false: $000 \in L(D_1)$ but $000 \notin L(D_2)$, and $011 \in L(D_2)$ but $011 \notin L(D_1)$. The languages are incomparable, so neither equality nor proper inclusion holds.
6. (C) is true: a nonempty string in both languages has length $3k$ with $k \geq 1$. Its first 3 symbols form a loop label, so they lie in S_1 and also in S_2, which is impossible because $S_1 \cap S_2 = \varnothing$. Only ε is common to both.
7. (D) is true: $(L(D_1) \cup L(D_2))^* = (S_1^* \cup S_2^*)^* = (S_1 \cup S_2)^* = (\{0,1\}^3)^*$, which is exactly the set of strings over $\{0,1\}$ whose length is divisible by 3.

KEY IDEA

When the start state is the only accepting state and every return loop has the same length, the language is the star of the set of loop labels. Here the two loop sets split $\{0,1\}^3$ into two disjoint halves.

WATCH OUT FOR

Assuming the two machines accept the same language because both have a single accepting start state and 3-step loops. The loop label sets differ, so the languages are incomparable. Another slip is to miss a loop label such as 001 or 110, which come from the two parallel edges into the last state.

Q.48 MSQ · 2 MARKS

Theory of Computation › Context-free languages

Let $\Sigma = \{a, b, c, d\}$ and let $L = \{a^i b^j c^k d^\ell \mid i, j, k, \ell \geq 0\}$.

Which of the following constraints ensure(s) that the language L is context-free?

(A) $i + k = j + \ell$

(B) $i = k$ and $j = \ell$

(C) $i = \ell$ and $j = k$

(D) $i + j = k + \ell$

Official answer: **A, C, D**

SOLUTION

1. Each option adds one constraint to $L = \{a^i b^j c^k d^\ell \mid i, j, k, \ell \geq 0\}$. Check each restricted language for context-freeness separately.
2. **(A)** $i + k = j + \ell$. Split on whether $i \geq j$. If $i = j + m$ with $m \geq 0$, then $j + m + k = j + \ell$ gives $\ell = k + m$, so the word is $a^m\,(a^j b^j)\,(c^k d^k)\,d^m$. If $j = i + m$ with $m > 0$, then $i + k = i + m + \ell$ gives $k = \ell + m$, so the word is $(a^i b^i)\,(b^m c^m)\,(c^\ell d^\ell)$.
3. Both cases have context-free grammars. First case: $S_1 \to aS_1d \mid XY$, $X \to aXb \mid \varepsilon$, $Y \to cYd \mid \varepsilon$, which gives exactly $a^m a^j b^j c^k d^k d^m$. Second case: $S_2 \to XZY$ with the same X and Y and $Z \to bZc \mid \varepsilon$. With $S \to S_1 \mid S_2$ the grammar generates the whole language (check: ad, bc, $abcd$ are generated; ac has $i + k = 2 \neq 0 = j + \ell$ and is not). So (A) is context-free.
4. **(B)** $i = k$ **and** $j = \ell$, i.e. $\{a^i b^j c^i d^j\}$: the a's are matched with the c's and the b's with the d's, so the two dependencies cross. Pumping lemma: take $w = a^n b^n c^n d^n$, where n is the pumping length. Any split $w = uvxyz$ with $|vxy| \leq n$ and $|vy| \geq 1$ puts vxy inside at most two neighbouring blocks. If v or y holds two different letters, pumping up breaks the order $a^*b^*c^*d^*$. Otherwise pumping changes the count of a or b (but not c or d), or of b or c, or of c or d. In every case some pair (a, c) or (b, d) loses its balance, so $uv^2xy^2z \notin L$. (B) is not context-free.
5. **(C)** $i = \ell$ **and** $j = k$, i.e. $\{a^i b^j c^j d^i\}$: the dependencies are nested. $S \to aSd \mid T$, $T \to bTc \mid \varepsilon$ generates exactly this language, so (C) is context-free.

6. **(D)** $i + j = k + \ell$. Split on whether $i \geq \ell$. If $i = \ell + m$ with $m \geq 0$, then $\ell + m + j = k + \ell$ gives $k = j + m$, so the word is $a^{\ell}\, a^{m}\, (b^{j} c^{j})\, c^{m}\, d^{\ell}$. If $\ell = i + m$ with $m > 0$, then $i + j = k + i + m$ gives $j = k + m$, so the word is $a^{i}\, b^{m}\, (b^{k} c^{k})\, d^{m}\, d^{i}$.
7. Both cases are nested, so one grammar covers them:

$$S \to aSd \mid T \mid V$$
$$T \to aTc \mid U$$
$$U \to bUc \mid \varepsilon$$
$$V \to bVd \mid W$$
$$W \to bWc \mid \varepsilon$$

   $S \to aSd \mid T$ gives $a^{\ell} T d^{\ell}$, then T gives $a^{m} b^{j} c^{j} c^{m}$, the first case. $S \to aSd \mid V$ gives $a^{i} V d^{i}$, then V gives $b^{m} b^{k} c^{k} d^{m}$, the second case. Check on short strings: ac ($S \Rightarrow T \Rightarrow aTc \Rightarrow ac$), bd, ad and $abcd$ are generated; ab has $i + j = 2 \neq 0$ and is not. A PDA sees it too: push one symbol for every a and b, pop one for every c and d, let the finite control enforce the order $a^{*} b^{*} c^{*} d^{*}$, and accept on an empty stack. So (D) is context-free.
8. Correct options: (A), (C) and (D).

KEY IDEA

A one-stack machine can check a constraint when the matched counts are nested: the symbols counted first are matched in reverse order by the symbols counted last. In (D), every a and b is pushed and every c and d pops, so the constraint is a single stack balance. In (A) and (C) the constraint splits into a union of nested patterns. (B) needs two crossing matches (a with c and b with d), and a single stack cannot do both.

WATCH OUT FOR

Calling (D) non-context-free because it involves all four letters. $i + j = k + \ell$ is one count balanced against another, the shape of $\{x^{n} y^{n}\}$ with $x \in \{a, b\}$ and $y \in \{c, d\}$, and a PDA checks it with one stack. An Ogden or pumping argument against (D) breaks down because the pumped window can be a block of a's together with a block of d's, with all the b's and c's between them, and pumping both blocks keeps the sums equal. Another slip is putting the inner nonterminal in the wrong place in a grammar for (A), for example $X \to aXb \mid Y$ with $Y \to cYd$, which produces $a^{j} c^{k} d^{k} b^{j}$ instead of $a^{j} b^{j} c^{k} d^{k}$.

Q.49 MSQ · 2 MARKS

Data Structures and Algorithms › Trees & heaps

Consider a binary search tree (BST) with n leaf nodes ($n > 0$). Given any node V, the key present in the node is denoted as $Val(V)$. All the keys present in the given BST are distinct. The keys belong to the set of real numbers.

For a node V, let $Suc(V)$ denote the node that is its inorder successor. If a node V does not have an inorder successor, then $Suc(V)$ is $NULL$. As there are no duplicates, if $Suc(V)$ is not $NULL$, then $Val(V) < Val(Suc(V))$.

Corresponding to every leaf node L_i that has a non-NULL $Suc(L_i)$, a new key k_i with the following property is to be inserted into the BST.

$$Val(L_i) < k_i < Val(Suc(L_i))$$

Let K represent the list of all such new keys to be inserted into the BST.

Which of the following statements is/are true?

(A) K cannot have any duplicates

(B) K will have at least one element

(C) After inserting all keys from K, the height of the BST can increase at most by one

(D) Number of nodes in the BST will double after inserting all keys from K

Official answer: **A, C**

SOLUTION

1. Key fact: Suc(V) is defined with respect to the inorder traversal of the WHOLE tree, i.e. the full sorted sequence of all n' node values (not just leaves). So for any leaf L_i with Suc(L_i) ≠ NULL, the pair (Val(L_i), Val(Suc(L_i))) is a pair of values that are consecutive/adjacent in the tree's overall sorted order.
2. Since the values are distinct reals, the open interval (Val(L_i), Val(Suc(L_i))) between two values that are adjacent in sorted order contains no existing key, and any such 'gap' intervals for different leaves are pairwise disjoint (they are the gaps between different consecutive pairs of one fixed sorted sequence, and the successor map is injective: each node is the immediate successor of at most one node).
3. Hence whichever value k_i a leaf L_i contributes must lie strictly inside its own disjoint interval, so no two chosen keys k_i, k_j (i ≠ j) can be equal. Statement (A), K cannot have duplicates, is TRUE.
4. Check (B): take n = 1 (a single node, which is simultaneously the root and the only leaf). This node holds the maximum (and only) key, so it has no inorder successor, Suc = NULL, and no key qualifies for insertion, so K = ⌀. Statement (B), K will have at least one element, is FALSE in general.
5. Check (C): L_i is a leaf, so it has no right child. For a value strictly between a node's value and its inorder successor's value with nothing in between, the standard BST-insertion rule is: it becomes the right child of the predecessor if the predecessor has no right child (which always holds here since L_i is a leaf), otherwise the left child of the successor. Since L_i always satisfies 'no right child', k_i is always inserted as the right child of L_i, i.e. at depth(L_i) + 1.
6. Because each k_i attaches only below its own leaf L_i (a leaf cannot be an ancestor of any other original node), inserting the keys for other leaves does not change L_i's depth or the search path down to it, so this holds regardless of insertion order. Since every leaf's depth is at most the tree's height h, every new node lands at depth ≤ h + 1, so the height after all insertions can increase by at most one. Statement (C) is TRUE.
7. Check (D): $|K|$ = (number of leaves) − (1 if the global maximum node is itself a leaf, else 0), and the new total node count is (original nodes) + $|K|$. For a full/complete binary tree with n leaves, total original nodes = 2n − 1 and $|K|$ = n − 1, giving a new total of 3n − 2, which is far less than doubling (2(2n − 1) = 4n − 2) for n > 0. Statement (D), nodes will double, is FALSE in general.
8. So the true statements are (A) and (C).

KEY IDEA

Every leaf with a non-NULL successor sits at the boundary of a unique, pairwise-disjoint 'gap' in the tree's overall sorted key sequence, and, because a leaf has no right child, the new key for that gap is always forced to attach as that leaf's right child at depth exactly one more than the leaf's own depth.

WATCH OUT FOR

Treating Suc(V) as the inorder successor within the leaf sequence only (rather than the successor in the full tree's inorder traversal), or assuming the new key could land anywhere in the tree rather than realizing the 'no right child ⇒ becomes right child of the predecessor' insertion rule forces it to attach directly below its own leaf, both of which make it easy to wrongly reject (A) or (C), or to wrongly accept (B)/(D) without checking the n = 1 / full-tree counterexamples.

Q.50 MSQ · 2 MARKS

Data Structures and Algorithms › Linear structures

Consider a stack S and a queue Q. Both of them are initially empty and have the capacity to store ten elements each. The elements 1, 2, 3, 4, and 5 arrive one by one, in that order. When an element arrives, it is

assigned either to S (pushed on S) or to Q (enqueued to Q). Once all the five elements are stored, the output is generated in two steps. First, stack S is emptied by popping all elements. Then queue Q is emptied by dequeueing all elements. The output obtained by following this process is 4 3 1 2 5.

Given the output, the objective is to predict whether an element was assigned to S or Q.

Which of the following options is/are possible valid assignment(s) of the elements?

Note: In the options, the notation xS denotes that element x was assigned to S and yQ denotes that element y was assigned to Q.

(A) 1S, 2Q, 3S, 4S, 5Q

(B) 1Q, 2Q, 3S, 4S, 5Q

(C) 1Q, 2Q, 3Q, 4S, 5S

(D) 1S, 2S, 3S, 4Q, 5Q

Official answer: **A, B**

SOLUTION

1. Since elements 1,2,3,4,5 arrive in strictly increasing order, whichever subset is pushed onto S is pushed in increasing order, so popping S produces that subset in strictly decreasing order.
2. Whichever subset is enqueued into Q is dequeued in the same (arrival) order, so it comes out in strictly increasing order.
3. The final output is (decreasing sequence popped from S) followed by (increasing sequence dequeued from Q), and together these two parts must partition {1,2,3,4,5} exactly, each element appearing once. So a valid assignment corresponds exactly to a way of splitting the printed output 4,3,1,2,5 into a decreasing prefix followed by an increasing suffix.
4. Test split after position 2: prefix [4,3] is decreasing (4>3); suffix [1,2,5] is increasing (1<2<5): valid. This means S={3,4} (pushed 3 then 4, popped 4,3) and Q={1,2,5} (enqueued/dequeued 1,2,5 in that order), giving 1Q,2Q,3S,4S,5Q, option B.
5. Test split after position 3: prefix [4,3,1] is decreasing (4>3>1); suffix [2,5] is increasing (2<5): valid. This means S={1,3,4} (pushed in arrival order 1,3,4, popped 4,3,1) and Q={2,5} (dequeued 2,5 in arrival order), giving 1S,2Q,3S,4S,5Q, option A.
6. Check the remaining splits: after position 1, suffix [3,1,2,5] is not increasing (3>1): fails. After position 4, prefix [4,3,1,2] is not decreasing (1<2): fails. The whole sequence as an all-S split (k=5) is not decreasing throughout, and the all-Q split (k=0) is not increasing throughout. So k=2 and k=3 are the only valid splits.
7. Directly verify option C (1Q,2Q,3Q,4S,5S): S={4,5} pushed 4 then 5, popped in order 5,4; Q={1,2,3} dequeued 1,2,3; combined output = 5,4,1,2,3, which does not match the given output 4,3,1,2,5: option C is invalid.
8. Directly verify option D (1S,2S,3S,4Q,5Q): S={1,2,3} pushed in order, popped 3,2,1; Q={4,5} dequeued 4,5; combined output = 3,2,1,4,5, which does not match 4,3,1,2,5: option D is invalid.
9. Therefore the possible valid assignments are exactly options A and B.

KEY IDEA

A push-then-pop-all sequence of the increasing arrival order 1..5 must come out strictly decreasing (LIFO reverses order), while an enqueue-then-dequeue-all sequence of the same increasing arrivals must come out strictly increasing (FIFO preserves order); so the full output is always a decreasing block (from S) immediately followed by an increasing block (from Q), and checking validity reduces to finding every way to split the printed output into such a decreasing-prefix / increasing-suffix pair.

WATCH OUT FOR

Trying to verify each option only by re-simulating it forward (push/pop, enqueue/dequeue) one at a time, which is correct but slow and error-prone under time pressure; the faster and more reliable route is to first characterize the output's required shape (decreasing-then-increasing split of {1,...,5} in arrival order) and then just check where the given output 4 3 1 2 5 can be split that way, which immediately rules out any option whose S-subset, if popped, would not sit at the front in decreasing order matching the given prefix.

Q.51 MSQ · 2 MARKS

Operating Systems › Synchronisation

Consider three processes P1, P2, and P3 running identical code, as shown in the pseudocode below. A and B are two binary semaphores initialized to 1 and 0, respectively. X is a shared variable initialized to 0. Each line in the pseudocode is executed atomically.

Pseudocode of P1, P2, and P3

```
 Wait(A);
Print(*);
X = X+1;
If (X == 2)
  {
    Print($);
    Signal(B);
  }
Signal(A);
Wait(B);
Print(#);
Signal(B);
```

Assume that any of the three processes can start to execute first and context switching can happen between these processes at any arbitrary time and in any arbitrary order.

Which of the following patterns is/are possible to be generated as an outcome of the execution of these three processes?

(A) `**$*###`

(B) `**$#*##`

(C) `**$##*#`

(D) `***$###`

Official answer: **A, B, C**

SOLUTION

1. Call the processes in the order in which they get through `Wait(A)` : first, second, third. `A` starts at 1 and `Signal(A)` comes only after `Print(*)` , `X = X+1` and the `If` test. So that part is a critical section. The three `*` prints and the increments of `X` happen one process at a time.
2. `X` becomes 2 in the second process to enter. That process prints `$` straight after its own `*` , then does `Signal(B)` . So the output always starts with `*` , `*` , `$` . The third `*` cannot appear before `$` , because the third process enters only after the second process does `Signal(A)` , and that comes after `$` . Option D (`***$###`) has three `*` before `$` and is impossible.

3. `B` starts at 0 and the only `Signal(B)` before any `#` is the one in the second process, after `$`. Until then every process that has left its critical section is blocked on `Wait(B)`. So no `#` can appear before `$`.
4. After `Signal(B)`, `B = 1`. A process that passes `Wait(B)` sets `B = 0`, prints `#`, and `Signal(B)` sets it back to 1. The waiting processes therefore get through one at a time, and all three `#` are eventually printed. Nothing deadlocks.
5. After the prefix `**$` the remaining output is the third process's `*` and three `#`. The first and second processes have left their critical sections, so their `#` may be printed before or after the third `*`. The third process's own `#` comes after its own `*`.
6. So the third `*` cannot come after all three `#`. The possible tails are `*###` (A), `#*##` (B) and `##*#` (C). Each is reachable. For example, for C, let the first and second processes print their `#`, and only then let the third process enter and print `*` and `#`.
7. The pattern `***$###` is impossible, so only A, B and C are possible.

KEY IDEA

Semaphore `A` makes the first half a critical section, so `$` is always printed right after the second `*`. The semaphore `B` (initially 0) holds back every `#` until `$`. After that the three `#` print one at a time. The third `*` may fall anywhere in the tail except after all three `#`.

WATCH OUT FOR

Allowing a `#` before `$`, or a third `*` before `$` (option D). Another slip is forgetting that the third process's `*` has to wait for `Signal(A)`, which comes after `$`.

Q.52 MSQ · 2 MARKS

Computer Organization and Architecture › Memory & cache

Consider a system with a processor and a 4 KB direct mapped cache with block size of 16 bytes. The system has a 16 MB physical memory. Four words P, Q, R, and S are accessed by the processor in the same order 10 times. That is, there are a total of 40 memory references in the sequence P, Q, R, S, P, Q, R, S, ...

Assume that the cache memory is initially empty. The physical addresses of the words are given below (1 word = 1 byte).

P: 0x845B32, Q: 0x845B26, R: 0x845B36, S: 0x846B32

Which of the following statements is/are true?

Note: 1K=2^{10} and 1M=2^{20}

(A) Every access to P results in a cache miss

(B) Every access to R results in a cache hit

(C) Every access to Q results in a cache miss

(D) Except the first access to S, all subsequent accesses to S result in cache hits

Official answer: **A, B**

SOLUTION

1. Cache size 4 KB, block size 16 B ⇒ number of lines = $4096/16 = 256 = 2^8$, so offset = 4 bits, index = 8 bits. Physical memory 16 MB = 2^{24} ⇒ address width 24 bits, so tag = $24 - 8 - 4 = 12$ bits.
2. offset + index = 12 bits = 3 hex digits, and tag = 12 bits = 3 hex digits, so each 6-hex-digit physical address splits exactly as: tag = first 3 hex digits, index = next 2 hex digits, offset = last hex digit.
3. P = 845B32 → tag 845, index B3, offset 2. Q = 845B26 → tag 845, index B2, offset 6. R = 845B36 → tag 845, index B3, offset 6. S = 846B32 → tag 846, index B3, offset 2.
4. P and R share tag 845 and index B3, so they lie in the SAME 16-byte cache block (their offsets 2 and 6 both fall inside one block, whose offsets run 0-15). S has the same index B3 but a different tag (846), so S maps to the same cache line as P/R and evicts whatever currently occupies it. Q has index B2, a line never touched by P, R or S.

5. Round 1 (cache initially empty): access P → miss, loads block (tag 845, index B3) into line B3. Access Q → miss, loads block (tag 845, index B2) into line B2. Access R → line B3 already holds tag 845, matching R's tag → hit. Access S → line B3 holds tag 845 ≠ 846 → miss, loads block (tag 846, index B3), evicting the P/R block.
6. Round 2 onward: access P → line B3 now holds tag 846 (from S) ≠ 845 → miss, reloads block (845, B3). Access Q → line B2 still holds tag 845, untouched by anything else → hit. Access R → line B3 was just reloaded by P with tag 845 → hit. Access S → line B3 now holds tag 845 (from P) ≠ 846 → miss, reloads block (846, B3).
7. This P-miss / Q-hit / R-hit / S-miss pattern (with Q's very first access also being a miss) repeats identically for rounds 2 through 10, since P and S perpetually evict each other's block from line B3 while line B2 (Q) is never disturbed by anyone else.
8. Evaluate each option: (A) P misses on all 10 of its accesses: TRUE. (B) R hits on all 10 of its accesses, including round 1: TRUE. (C) Q misses only on its first access and hits on the remaining 9: FALSE, since "every access" is wrong. (D) S misses on every single access (not just the first), because P re-evicts S's block each round before S is next accessed: FALSE.

KEY IDEA

In a direct-mapped cache, split the 24-bit physical address into tag/index/offset. Here offset+index together equal exactly 3 hex digits and tag also equals exactly 3 hex digits, so a 6-hex-digit address splits cleanly as tag = first 3 digits, index = next 2 digits, offset = last digit. Two addresses share a cache block only if BOTH tag and index match; they conflict (evict each other) if only the index matches.

WATCH OUT FOR

Assuming S, having a different tag from P/R, must land in a different cache line and never interferes with them, missing that S shares the SAME index (B3) with P and R, which is exactly what causes S to evict, and be evicted by, the P/R block every round. A second common slip is assuming R's very first access must be a miss like P's, overlooking that R is accessed right after P has just loaded that same block within round 1.

Q.53 NAT · 2 MARKS

Operating Systems › File systems & disks

To keep track of free blocks in a file system, one of the two approaches is generally used – using bitmaps (bit vectors) or using linked lists. Consider that the linked list approach is used to keep track of free blocks in a file system. Assume that the disk size is 16 GB, block size is 2 KB, and block numbers used are 32-bit long. A single pointer of size 4 bytes is used in each block of the list to point to the next block of the list. The number of blocks required to hold the free disk block numbers is _____. (answer in integer)
Note: 1K=2^10 and 1G=2^30

Official answer: **16417**

SOLUTION

1. Number of disk blocks: $\frac{16\,\text{GB}}{2\,\text{KB}} = \frac{2^{34}}{2^{11}} = 2^{23} = 8388608$. The free list must be able to hold the numbers of all these blocks, so 2^{23} block numbers are stored.
2. Each block number is 32 bits $= 4$ bytes.
3. Each list block is $2\,\text{KB} = 2048$ bytes, and 4 bytes of it hold the pointer to the next list block. Space for block numbers $= 2048 - 4 = 2044$ bytes.
4. Block numbers per list block $= \frac{2044}{4} = 511$.
5. List blocks needed $= \left\lceil \frac{2^{23}}{511} \right\rceil = \left\lceil \frac{8388608}{511} \right\rceil$.
6. $511 \times 16416 = 8388576$, so $8388608 - 8388576 = 32$ block numbers are left over and need one more block.
7. Number of blocks $= 16416 + 1 = 16417$.

KEY IDEA
Count the blocks on the disk, then see how many 4-byte block numbers fit in one list block after setting aside 4 bytes for the next-block pointer. Divide and round up.

WATCH OUT FOR
Using all 2048 bytes for block numbers gives 512 per block and $2^{23}/512 = 16384$. This forgets that 4 bytes of each list block hold the pointer to the next list block.

Q.54 NAT · 2 MARKS

Operating Systems › Memory management

A system has a Translation Lookaside Buffer (TLB) that has a reach of 1 MB. TLB reach is defined as the total amount of physical memory that can be accessed through the TLB entries. The paging system uses pages of size 4 KB. The virtual address space is 64 GB and physical address space is 1 GB. If each TLB entry stores a 4-bit process id, page number, frame number, and a 2-bit control field, then the size of the TLB (in bytes) is _____. (answer in integer)

Note: 1K=2^10, 1M=2^20, 1G=2^30

Official answer: **1536**

SOLUTION

1. Each TLB entry maps one page, so the number of entries is $\dfrac{\text{TLB reach}}{\text{page size}} = \dfrac{2^{20}}{2^{12}} = 2^8 = 256.$
2. The page size is $4\,\text{KB} = 2^{12}$ bytes, so the page offset is 12 bits.
3. The virtual address space is $64\,\text{GB} = 2^{36}$ bytes, so a virtual address has 36 bits. The page number has $36 - 12 = 24$ bits.
4. The physical address space is $1\,\text{GB} = 2^{30}$ bytes, so a physical address has 30 bits. The frame number has $30 - 12 = 18$ bits.
5. Bits per TLB entry $= 4\ (\text{process id}) + 24\ (\text{page number}) + 18\ (\text{frame number}) + 2\ (\text{control}) = 48$ bits $= 6$ bytes.
6. TLB size $= 256 \times 6 = 1536$ bytes.

KEY IDEA
The number of TLB entries is the TLB reach divided by the page size. Each entry holds the process id, the page number, the frame number and the control bits. The page number and frame number sizes come from the address widths minus the page offset.

WATCH OUT FOR
Using the full 36-bit virtual address or the 30-bit physical address instead of subtracting the 12-bit offset. Another slip is leaving out the process id or control bits, or giving the size in bits (48 per entry) instead of bytes.

Q.55 NAT · 2 MARKS

Operating Systems › Memory management

Consider contiguous allocation of physical memory to processes using variable partitioning scheme. Suppose there are 8 holes in the memory of sizes 20 KB, 4 KB, 25 KB, 18 KB, 7 KB, 9 KB, 15 KB, and 12 KB. Assume that no two holes are adjacent. Two processes P1 of size 16 KB and P2 of size 9 KB arrive in that order, and they are allocated memory using the best-fit technique. After allocating space to P1 and P2, the number of holes of size less than 8 KB is _____. (answer in integer)

Note: 1K=2^10

Official answer: **3**

SOLUTION

1. Best fit allocates each process to the smallest hole that is large enough for it.
2. P1 needs $16\ \text{KB}$. The holes of at least $16\ \text{KB}$ are $20, 25, 18$. The smallest is $18\ \text{KB}$, so P1 goes there and leaves a hole of $18 - 16 = 2\ \text{KB}$.

3. The holes are now $20, 4, 25, 2, 7, 9, 15, 12$ (in KB).
4. P2 needs $9\ \mathrm{KB}$. The holes of at least $9\ \mathrm{KB}$ are $20, 25, 9, 15, 12$. The smallest is exactly $9\ \mathrm{KB}$, so P2 fills it completely and no leftover hole remains.
5. The holes are now $20, 4, 25, 2, 7, 15, 12$ (7 holes).
6. The holes smaller than $8\ \mathrm{KB}$ are $4, 2, 7$. The count is 3.

KEY IDEA

Best fit takes the smallest hole that is big enough. An exact fit leaves no hole behind, while a larger hole leaves a remainder that becomes a new hole.

WATCH OUT FOR

Using first fit or worst fit would put P1 in the 20 KB or 25 KB hole. Another slip is leaving a zero-size hole after the exact fit of P2 and counting it. A third is forgetting that the 2 KB remainder from P1 is a new hole smaller than 8 KB.

Q.56 NAT · 2 MARKS

Computer Organization and Architecture › Memory & cache

Consider a system with 1 MB physical memory and a word length of 1 byte. The system uses a direct mapped cache, with block numbers starting from 0. The word with physical address 0xA2C28 is mapped to the cache block number 176_{10}. The maximum possible size of the cache (in KB) for this configuration is _____. (answer in integer)

Note: $1K = 2^{10}$ and $1M = 2^{20}$.

Official answer: **128**

SOLUTION

1. Physical memory $= 1\ \mathrm{MB} = 2^{20}$ bytes and the word length is 1 byte, so a physical address is 20 bits wide (bit19 to bit0), with no separate word-offset-within-word bits needed.
2. Write $\mathrm{0xA2C28}$ in 20-bit binary: $1010\ 0010\ 1100\ 0010\ 1000$ (bit19…bit0). (Check: this reproduces $\mathrm{0xA2C28} = 666664_{10}$.)
3. A direct-mapped address splits as $\mathrm{Tag} \mid \mathrm{Block\ number}\ (k\ \mathrm{bits}) \mid \mathrm{Block\ offset}\ (b\ \mathrm{bits})$, with block size $= 2^b$ bytes, number of blocks $= 2^k$, and cache size $= 2^{k+b}$ bytes.
4. $176_{10} = 10110000_2$ (8 bits). Scanning every possible 8-bit window of the 20-bit address (all 13 offsets $b = 0, \ldots, 12$) for the value 10110000 shows exactly one match: bits$[13:6] = 1, 0, 1, 1, 0, 0, 0, 0 = 176$. So the block-offset width must be $b = 6$.
5. With $b = 6$ fixed, grow the block-number width k upward from 8 one bit at a time; the field still reads 176 only while each newly absorbed higher bit is 0. bit14$= 0$ ($k = 9$, field still 176), bit15$= 0$ ($k = 10$, still 176), bit16$= 0$ ($k = 11$, still 176).
6. bit17$= 1$, so absorbing it changes the field's value away from 176 (it becomes $2048 + 176 = 2224$). Hence $k = 11$ is the largest block-number width consistent with the given mapping, i.e. at most $2^{11} = 2048$ cache blocks.
7. Maximum cache size $= 2^{k+b} = 2^{11+6} = 2^{17}$ bytes $= 2^{17}/2^{10} = 2^7 = 128\ \mathrm{KB}$. (Tag width $= 20 - 17 = 3$ bits, nonzero, so the split is self-consistent.)

KEY IDEA

In a direct-mapped cache, the address splits into Tag | Block number (k bits) | Block offset (b bits). Given only the numeric value of the block number (not k or b individually), find every bit-offset b at which an 8-bit window of the physical address equals that block number in binary; here only $b = 6$ works. Then, holding b fixed, the block-number field can be widened (k increased) for free as long as every newly included higher-order bit is 0, since a leading 0 does not change the field's decimal value; this is what lets the cache be "maximum possible" size rather than a single fixed size.

WATCH OUT FOR

Assuming the number of bits in the binary form of the block number (here $176_{10} = 10110000_2$, 8 bits) directly gives k, and stopping there, i.e. computing cache size as $2^{8+6} = 16\ \text{KB}$ instead of recognizing that higher-order zero bits can be folded into a wider block-number field without changing its decimal value, which is what the word "maximum possible" in the question is asking you to exploit.

Q.57 NAT · 2 MARKS

Computer Organization and Architecture › Pipelining

A non-pipelined instruction execution unit that operates at 1.6 GHz clock takes an average of 5 clock cycles to complete the execution of an instruction. To improve the performance, the system was pipelined with a goal of achieving an average throughput of one instruction per clock cycle. However, it could operate only at 1.2 GHz due to pipeline overheads. While executing a program in the pipelined design, 30% of instructions encountered a stall of 2 cycles due to pipeline hazards. The speed-up obtained by the pipelined design over the non-pipelined one for this program is _____. (rounded off to two decimal places)

Note: 1G=10^9

Official answer: **2.30 to 2.40**

SOLUTION

1. Non-pipelined execution time per instruction: with CPI = 5 and clock frequency f = 1.6 GHz,
$$T_{np} = \frac{CPI}{f} = \frac{5}{1.6\,\text{GHz}} = 3.125\,\text{ns}$$
2. Pipelined design's ideal CPI is 1 (target of one instruction per clock cycle). However, 30% of instructions each incur an extra stall of 2 cycles, adding an average of $0.30 \times 2 = 0.6$ cycles per instruction to the CPI.
3. Average CPI for the pipelined design:
$$CPI_p = 1 + 0.6 = 1.6$$
(equivalently, $0.7 \times 1 + 0.3 \times 3 = 1.6$, since 70% of instructions take 1 cycle and 30% take $1 + 2 = 3$ cycles).
4. Pipelined execution time per instruction, with clock frequency $f_p = 1.2\,\text{GHz}$:
$$T_p = \frac{CPI_p}{f_p} = \frac{1.6}{1.2\,\text{GHz}} = 1.3333\,\text{ns}$$
5. Speed-up is the ratio of execution times:
$$\text{Speed-up} = \frac{T_{np}}{T_p} = \frac{3.125}{1.3333} = 2.34375$$
6. Rounded to two decimal places, the speed-up is 2.34.

KEY IDEA

Speed-up from pipelining = (execution time per instruction, non-pipelined) / (execution time per instruction, pipelined), where execution time per instruction = CPI / clock frequency. The pipelined design's effective CPI must include the extra stall cycles caused by hazards, not just the ideal CPI = 1.

WATCH OUT FOR

Forgetting to add the hazard-stall penalty to the pipelined CPI (using CPI = 1 for the pipelined design instead of 1 + 0.3×2 = 1.6), which gives a wrong speed-up of 3.125/(1/1.2) = 2.4 instead of the correct 2.34; or comparing raw clock frequencies (1.6/1.2) instead of execution times per instruction.

Q.58 NAT · 2 MARKS

Computer Networks › TCP

Consider a new TCP connection between a sender and a receiver. The receiver advertised window is constant at 48 KB, the maximum segment size (MSS) is 2 KB, and the slow start threshold for TCP congestion control is 16 KB. Assume that there are no timeouts or duplicate acknowledgements. The number of rounds of transmission required for the congestion control algorithm of the TCP connection to reach the congestion avoidance phase is _____. (answer in integer)

Note: 1K=2^10

Official answer: **4**

SOLUTION

1. Work in units of MSS. $\text{MSS} = 2\ \text{KB}$, so the slow start threshold is $16/2 = 8$ MSS and the receiver window is $48/2 = 24$ MSS. The receiver window is larger than the threshold, so it never limits the sender before the threshold is reached.
2. A new connection starts in slow start with $\text{cwnd} = 1\ \text{MSS} = 2\ \text{KB}$. With no timeouts or duplicate ACKs, every ACK adds one MSS, so the window doubles every round.
3. Take the window used in each round (the congestion window at the start of that round, which is also the amount sent in it):

Round	cwnd (MSS)	cwnd (KB)
1	1	2
2	2	4
3	4	8
4	8	16

4. In rounds 1 to 3 the window is $2, 4, 8\ \text{KB}$, all below the threshold of $16\ \text{KB}$, so the sender is still in slow start.
5. In round 4 the window is $16\ \text{KB}$, which equals the threshold. Slow start ends here (the window is not allowed to grow past the threshold in slow start), and from this point the window grows by one MSS per round, which is congestion avoidance.
6. The connection therefore reaches the congestion avoidance phase in round 4, so the number of rounds of transmission needed is 4.

KEY IDEA

In slow start the window doubles every round from 1 MSS. Count the rounds until the window used in a round equals the slow start threshold: 2, 4, 8, 16 KB, which is round 4.

WATCH OUT FOR

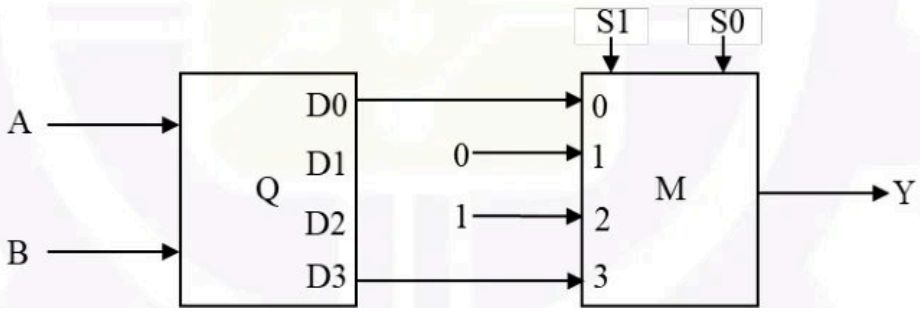
Mixing two conventions for the window, such as listing the window after the ACKs in one row and before them in another, or letting slow start run on to 32 KB. Slow start stops at the threshold. Another slip is forgetting that the starting window is 1 MSS = 2 KB, not 1 KB, which gives the wrong number of doublings.

Q.59 NAT · 2 MARKS

Digital Electronics › Combinational

Q.59 Consider the digital circuit shown below with two input lines A and B, two select lines S0 and S1, and an output line Y. The blocks Q and M represent active high 2:4 decoder and 4-to-1 multiplexer, respectively. Out of 16 possible input combinations, the number of combinations that produce Y=1 is _____. (*answer in integer*)

Note: One input combination is an instance of [A B S1 S0].



Official answer: **6**

SOLUTION

1. Read the wiring from the figure. The multiplexer data inputs are $I_0 = D_0$, $I_1 = 0$ (tied low), $I_2 = 1$ (tied high) and $I_3 = D_3$. Decoder outputs D_1 and D_2 are not connected.
2. The decoder Q is active high with inputs A, B, so $D_0 = \overline{A}\,\overline{B}$ and $D_3 = AB$.
3. A 4-to-1 multiplexer with select lines S_1S_0 gives $Y = \overline{S_1}\,\overline{S_0}\,I_0 + \overline{S_1}S_0\,I_1 + S_1\overline{S_0}\,I_2 + S_1S_0\,I_3$.
4. Case $S_1S_0 = 00$: $Y = D_0$, which is 1 only for $AB = 00$. That gives 1 combination.
5. Case $S_1S_0 = 01$: $Y = I_1 = 0$ for every AB. That gives 0 combinations.
6. Case $S_1S_0 = 10$: $Y = I_2 = 1$ for every AB. That gives 4 combinations.
7. Case $S_1S_0 = 11$: $Y = D_3$, which is 1 only for $AB = 11$. That gives 1 combination.
8. The cases are disjoint and together cover all 16 combinations $[A\,B\,S_1\,S_0]$, so the total is $1 + 0 + 4 + 1 = 6$.

KEY IDEA

Split the 16 combinations by the select value S_1S_0. Each select value picks one mux input, and you count the AB values that make that input 1. A constant input gives 0 or 4, and a decoder output gives 1.

WATCH OUT FOR

Counting only the 4 combinations with $S_1S_0 = 10$ and forgetting that D_0 and D_3 are each 1 for exactly one AB value. Another slip is treating the unused outputs D_1 and D_2 as if they were connected to the mux.

Q.60 NAT · 2 MARKS

Programming in C › Pointers & arrays

Consider the following ANSI-C program.

```
#include <stdio.h>

int main(){
    int *ptr, a, b, c;
    a=5; b=11; c=20;
    ptr=&a; *ptr=c; ptr=&c;
    a=*(&b); c=*ptr-a;
    printf("%d",c);
    return(0);
}
```

The output of this program is _____. (answer in integer)

Note: Assume that the program compiles and runs successfully.

Official answer: **9**

SOLUTION

1. Initialize: a=5, b=11, c=20.
2. ptr=&a; sets ptr to point to a.
3. *ptr=c; writes c's current value (20) into a, so a=20; c is unchanged at 20.
4. ptr=&c; retargets ptr to point to c (it no longer points to a).
5. a=*(&b); : &b dereferenced by * is simply b, so a=11 (overwriting the earlier a=20).
6. c=*ptr-a; : since ptr points to c, *ptr yields c's current value, 20; so c = 20 - 11 = 9.
7. printf("%d",c); outputs 9.

KEY IDEA
Track pointer targets and values separately: an assignment through `*ptr` changes whatever object `ptr` currently points to, not necessarily the same variable it pointed to earlier or later.

WATCH OUT FOR
Assuming `*ptr` always refers to `a` throughout the program (since `ptr` was first set to `&a`), and missing that `ptr=&c;` re-targets it to `c` before the final `c=*ptr-a;` line. This leads students to wrongly read `*ptr` as `a`'s value there instead of `c`'s.

Q.61 NAT · 2 MARKS

Programming in C › Recursion

Consider the following ANSI-C function.

```
int func(int start, int end){
    int length=end+1-start;
    if((length<1)||(start<0)||(end<0)){ return(0); }
    if(length%3==0){
        return(func(start+1, end));
    } else if(length%3==1){
        return(1+func(start, end-1));
    } else {
        return(func(start+2, end));
    }
}
```

The maximum possible value that can be returned from this function is _____. (answer in integer)

Note: Ignore syntax errors (if any) in the function.

Official answer: **1**

SOLUTION

1. Let $L = \text{end} - \text{start} + 1$ be the value of `length` at a call. Track only how $L \bmod 3$ evolves and how many `+1`s the recursion adds.
2. Also note `start` never decreases (it is passed as `start`, `start+1`, or `start+2`) and `end` never increases (it is passed as `end` or `end-1`). So once `start >= 0` and `end >= 0` hold initially, the recursion cannot terminate for a reason other than `length < 1`, except possibly through the one `end - 1` step described below.
3. Branch $L \bmod 3 = 0$: calls `func(start+1, end)`, giving new length $L - 1$. Since $L \equiv 0$, the new residue is $(L-1) \bmod 3 = 2$. This branch adds 0 to the return value.
4. Branch $L \bmod 3 = 1$: calls `1+func(start, end-1)`, giving new length $L - 1$. Since $L \equiv 1$, the new residue is $(L-1) \bmod 3 = 0$. This branch adds exactly 1 to the return value.
5. Branch $L \bmod 3 = 2$: calls `func(start+2, end)`, giving new length $L - 2$. Since $L \equiv 2$, the new residue is $(L-2) \bmod 3 = 0$. This branch adds 0 to the return value.
6. So the residue-transition map is $0 \to 2$, $1 \to 0$, $2 \to 0$, and only the transition out of residue 1 ever adds a point.
7. Residue 1 never appears as the TARGET of any transition: the only targets produced are 2 (from 0) and 0 (from 1 or 2). So residue 1 can occur only at the very first call (if the initial length is $\equiv 1 \pmod 3$) and can never recur later in the recursion.
8. Hence at most one `+1` can ever be earned in any run, and it is earned only when the initial length satisfies $L_0 \equiv 1 \pmod 3$; every subsequent step (cycling through residues 0 and 2) contributes 0 until `length < 1` triggers the base case returning 0.
9. Trace check with $L_0 = 1$ (e.g. `start = end = 0`): $1 \bmod 3 = 1$, so it returns `1+func(start, end-1)`. The inner call now has length $0 + 1 - 0 = 0 < 1$. Wait, recompute: `func(0,-1)` has length $= (-1) + 1 - 0 = 0$, so `length<1` (and `end<0`) is true and it returns 0. Total $= 1 + 0 = 1$.
10. Trace check with $L_0 = 4$: residues go $4 \bmod 3 = 1$ (+1) → length 3, residue 0 (+0) → length 2, residue 2 (+0) → length 0, which hits `length<1` and returns 0. Total $= 1$.
11. Since no sequence of lengths can ever revisit residue 1, no run can earn a second `+1`. The maximum possible return value is therefore 1, achieved by any starting pair with $(\text{end} - \text{start} + 1) \equiv 1 \pmod 3$.

KEY IDEA

Since `start` never decreases and `end` never increases, the recursion is driven entirely by `length` $\bmod 3$: the transition map is $0 \to 2$, $1 \to 0$, $2 \to 0$, and only the $1 \to 0$ step ever adds `+1`. Because residue 1 is never a *target* of any transition, it can occur only once (at the very start, if at all), so at most one point can ever be earned in the whole recursion.

WATCH OUT FOR

Assuming the `+1` can be earned repeatedly (e.g. once per "cycle" of 3 in the original length) without checking which residues are actually reachable after the first step; tracing only one example instead of proving that residue 1 can never recur rules out any larger value.

Q.62 NAT · 2 MARKS

Engineering Mathematics › Linear algebra

The determinant of a 4×4 matrix A is 3. The value of the determinant of $2A$ is _____. (answer in integer)

Official answer: **48**

SOLUTION

1. Multiplying every entry of an $n \times n$ matrix A by k multiplies each of its n rows by k.

2. The determinant is linear in each row, so scaling one row by k multiplies the determinant by k. Scaling all n rows gives $\det(kA) = k^n \det A$.
3. Here $n = 4$, $k = 2$ and $\det A = 3$.
4. $\det(2A) = 2^4 \times 3 = 16 \times 3 = 48$.

KEY IDEA
For an $n \times n$ matrix, $\det(kA) = k^n \det A$, because each of the n rows picks up a factor of k.

WATCH OUT FOR
Writing $\det(2A) = 2 \det A = 6$ by multiplying the determinant by k only once instead of k^n (here $2^4 = 16$).

Q.63 NAT · 2 MARKS

Engineering Mathematics › Probability & statistics

Suppose an unbiased coin is tossed 6 times. Each coin toss is independent of all previous coin tosses. Let E_1 be the event that among the second, fourth, and sixth coin tosses, there are at least two heads. Let E_2 be the event that among the first, second, third, and fifth coin tosses, there are equal number of heads and tails.

The conditional probability $P(E_1 \mid E_2)$ is equal to _____. (rounded off to one decimal place)

Official answer: **0.5**

SOLUTION

1. E_2 concerns tosses 1, 2, 3, 5. Equal heads and tails among 4 tosses means exactly 2 heads. There are $\binom{4}{2} = 6$ such outcomes, all equally likely.
2. Toss 2 is the only toss common to E_1 and E_2. Tosses 4 and 6 do not appear in E_2, so they are independent of it and each is a fair coin.
3. Given E_2, toss 2 is a head in $\binom{3}{1} = 3$ of the 6 outcomes, because the other head must be among tosses 1, 3, 5. So $P(\text{toss 2 is H} \mid E_2) = \frac{3}{6} = \frac{1}{2}$, and $P(\text{toss 2 is T} \mid E_2) = \frac{1}{2}$.
4. If toss 2 is a head, E_1 needs at least one head among tosses 4 and 6. This has probability $1 - \frac{1}{4} = \frac{3}{4}$.
5. If toss 2 is a tail, E_1 needs both tosses 4 and 6 to be heads. This has probability $\frac{1}{4}$.
6. By total probability over toss 2: $P(E_1 \mid E_2) = \frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{1}{4} = \frac{1}{2} = 0.5$.

KEY IDEA
Condition on the one toss the two events share (toss 2). The remaining tosses of E_1 are independent of E_2, so split on toss 2 and use total probability.

WATCH OUT FOR
Treating E_1 and E_2 as independent, or forgetting that toss 2 appears in both events. Another slip is counting the outcomes of E_2 as $\binom{4}{2}$ out of 2^6 instead of conditioning on the four tosses only.

Q.64 NAT · 2 MARKS

Engineering Mathematics › Calculus

Consider a function $f : (0, 1) \to \{0, 1\}$ defined as follows.

For a real number $r \in (0, 1)$, $f(r) = 1$ if the second digit after the decimal point in r is one of the four digits 2, 3, 6 and 7. Otherwise, $f(r)$ is equal to 0.

The number of points in $(0, 1)$ at which f is discontinuous is _____. (answer in integer)

Official answer: **40**

SOLUTION

1. $f(r)$ depends only on the second decimal digit of r. So f is constant on each interval $\left[\frac{k}{100}, \frac{k+1}{100}\right)$ for $k = 0, 1, \ldots, 99$, and it is continuous at every point inside such an interval. A discontinuity can occur only at the boundary points $\frac{k}{100}$, $k = 1, \ldots, 99$.

2. At $r = \frac{k}{100}$ use the terminating expansion, so its second digit is $d = k \bmod 10$. Then $f(r)$ equals the constant value on the interval to the right of r, so the right-hand limit equals $f(r)$. Only the left-hand limit can differ.
3. The value of f for each second digit d is

d	0	1	2	3	4	5	6	7	8	9
f	0	0	1	1	0	0	1	1	0	0

4. f is discontinuous at $\frac{k}{100}$ exactly when the value on the interval to its left (second digit $d-1$) differs from the value on the interval to its right (second digit d). Within one decade this happens at $d = 2$ ($0 \to 1$), $d = 4$ ($1 \to 0$), $d = 6$ ($0 \to 1$) and $d = 8$ ($1 \to 0$): 4 jump points. At $d = 1, 3, 5, 7, 9$ the value does not change, so f is continuous there.
5. Between decades the digit goes from 9 to 0 and f goes from 0 to 0, so there is no jump at $0.1, 0.2, \ldots, 0.9$.
6. There are 10 decades (first digit $0, 1, \ldots, 9$), each with 4 jump points, all inside $(0, 1)$. The number of points of discontinuity is $10 \times 4 = 40$.

KEY IDEA

f is a step function that is constant on each interval $[k/100, (k+1)/100)$. It can jump only at the multiples of 0.01, and it does so exactly where the second digit changes between a value with $f = 0$ and a value with $f = 1$.

WATCH OUT FOR

Counting the digits $2, 3, 6, 7$ (4 digits) or the 8 changes of value per decade instead of the 4 genuine jumps per decade. Another mistake is to count the change from digit 9 to digit 0 at $0.1, 0.2, \ldots$ as a jump, when both digits give $f = 0$.

Q.65 NAT · 2 MARKS

Computer Networks › Performance

It is necessary to design a link-layer protocol between two hosts that are directly connected over a lossless link of length 3000 kilometers. Assume that the link bandwidth is 10^8 bits per second and that the propagation delay in the link is 5 nanoseconds per meter. Every transmitted data byte is assigned a unique sequence number.

Let N be the minimum number of bits needed for the sequence number field in the protocol header such that

i. the sequence numbers do not wrap around before 60 seconds, and
ii. the maximum utilization of the link is achieved.

The value of N is _____. (answer in integer)

Official answer: **30**

SOLUTION

1. Propagation delay: $3000\ \text{km} = 3 \times 10^6\ \text{m}$, so $T_p = 3 \times 10^6 \times 5\ \text{ns} = 15\ \text{ms}$ and $\text{RTT} = 2T_p = 30\ \text{ms}$.
2. Condition ii (maximum utilization): the sender must keep the pipe full for one RTT, so the window is the bandwidth-delay product: $10^8 \times 0.03 = 3 \times 10^6\ \text{bits} = 375000\ \text{bytes}$.
3. Every byte has its own sequence number, so the sequence space must cover this window. Even with selective repeat (which needs $2W$ numbers): $2 \times 375000 = 750000 \le 2^{20} = 1048576$, so condition ii needs at most 20 bits. (Go-Back-N needs $W + 1 = 375001 \le 2^{19}$, i.e. 19 bits.)
4. Condition i (no wrap-around in 60 s): at full rate the link carries $\dfrac{10^8}{8} = 1.25 \times 10^7$ bytes per second, so in 60 s the sender uses up to $1.25 \times 10^7 \times 60 = 7.5 \times 10^8$ sequence numbers.
5. We need $2^N \ge 7.5 \times 10^8$. Since $2^{29} = 536870912 < 7.5 \times 10^8$ and $2^{30} = 1073741824 \ge 7.5 \times 10^8$, we get $N \ge 30$.

6. Condition i (30 bits) is stricter than condition ii (at most 20 bits), and 30 bits satisfies both. Hence $N = 30$.

KEY IDEA

Two conditions give two lower bounds on the sequence-number size: the window (bandwidth-delay product) fixes one, and the number of bytes sent at full rate in 60 s fixes the other. Take the larger.

WATCH OUT FOR

Using only the bandwidth-delay product (about 19 or 20 bits) and forgetting that, with one sequence number per byte, 60 s of full-rate sending uses 7.5×10^8 numbers. Another slip is counting bits instead of bytes when sizing the sequence space.

FROM IAE ACADEMY

Preparing for GATE CS?

IAE Academy teaches GATE CS in Ballygunge, Kolkata, and online. Counselling is free: ask us about batches, fees or your preparation.

FREE COUNSELLING

Chat on WhatsApp

Call +91 84204 36524

Open every day, 10:30 am to 7:30 pm.

Scan to chat on WhatsApp

THE GATE CS COURSE

Classroom batches in Ballygunge and online batches. Fees and details are on the course page.

Scan: course page

See the course

FREE ON IAEACADEMY.IN

More GATE CS papers, question by question with the official answers, and free notes by subject.

Scan: more papers

More solved papers

Found this useful? Send it to a friend preparing for GATE.

Send to a friend on WhatsApp

IAE Academy free study material. The questions are those of the GATE CS 2026 Set 2 question paper, organised by IIT Guwahati; the answers are from its official answer key. Prepared and published by IAE Academy, Kolkata. Free to read and free to share, unaltered and with attribution. Not affiliated with, endorsed by or produced by IIT Guwahati or the GATE committee.

IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE coaching in Kolkata since 2012 · EC, EE, IN, CS and IT, DA, BT and CE · in the classroom and online

227A Rash Behari Ave, Ballygunge, Kolkata 700019

+91 84204 36524 · iaeacademy.in · Updated 27 October 2026

Get directions