IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE CS · 2026 · SET 1

IAE Academy free study material · Updated 27 October 2026

GATE CS 2026 Set 1 Solved Paper

INSIDE

- All 65 questions, in the order of the paper
- The official answer to every question
- Step-by-step worked solutions
- 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-2026-1**

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: 28 MCQ · 24 MSQ · 13 NAT.

Official answer key: IIT Guwahati · https://gate2026.iitg.ac.in/doc/download/2026/Keys/CS1_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-2026-1

Answer key

Q	TYPE	MARKS	ANSWER
Q.1	MCQ	1	**B**
Q.2	MCQ	1	**C**
Q.3	MCQ	1	**C**
Q.4	MCQ	1	**D**
Q.5	MCQ	1	**C**
Q.6	MCQ	2	**B**
Q.7	MCQ	2	**D**
Q.8	MCQ	2	**A**
Q.9	MCQ	2	**A**
Q.10	MCQ	2	**C**
Q.11	MCQ	1	**B**
Q.12	MCQ	1	**A**
Q.13	MCQ	1	**A**
Q.14	MCQ	1	**B**
Q.15	MCQ	1	**D**
Q.16	MCQ	1	**B**
Q.17	MCQ	1	**A**
Q.18	MCQ	1	**D**
Q.19	MSQ	1	**A, C, D**
Q.20	MSQ	1	**B, D**
Q.21	MSQ	1	**A, C**
Q.22	MSQ	1	**B, C**
Q.23	MSQ	1	**B, C**
Q.24	MSQ	1	**C**
Q.25	MSQ	1	**A, B, C**
Q.26	MSQ	1	**B, D**
Q.27	MSQ	1	**C**
Q.28	MSQ	1	**C**
Q.29	MSQ	1	**A, C**
Q.30	MSQ	1	**C, D**
Q.31	MSQ	1	**A, B, C**
Q.32	NAT	1	**3**
Q.33	NAT	1	**11**
Q.34	NAT	1	**9**
Q.35	NAT	1	**6**
Q.36	MCQ	2	**C**
Q.37	MCQ	2	**B**
Q.38	MCQ	2	**A**
Q.39	MCQ	2	**B**
Q.40	MCQ	2	**D**
Q.41	MCQ	2	**A**
Q.42	MCQ	2	**D**
Q.43	MCQ	2	**B**
Q.44	MCQ	2	**B**
Q.45	MCQ	2	**B**
Q.46	MSQ	2	**A, C, D**
Q.47	MSQ	2	**A**
Q.48	MSQ	2	**B, D**
Q.49	MSQ	2	**A, D**
Q.50	MSQ	2	**B, D**
Q.51	MSQ	2	**C**
Q.52	MSQ	2	**B, D**
Q.53	MSQ	2	**A, B, D**
Q.54	MSQ	2	**B, C**
Q.55	MSQ	2	**B, C**
Q.56	MSQ	2	**A, B, C**
Q.57	NAT	2	**34**
Q.58	NAT	2	**4.24 to 4.26**
Q.59	NAT	2	**77.3**
Q.60	NAT	2	**95**
Q.61	NAT	2	**65536**
Q.62	NAT	2	**7**
Q.63	NAT	2	**4**
Q.64	NAT	2	**9.5**
Q.65	NAT	2	**6**

General Aptitude Q.1 to Q.10

Q.1 MCQ · 1 MARK

General Aptitude › Grammar & vocabulary

The antonym of the word protagonist is _____.

(A) agnostic

(B) antagonist

(C) arsonist

(D) anarchist

Official answer: **B**

SOLUTION

1. A **protagonist** is the leading character or main figure in a story or a cause, the one who drives the action.
2. An antonym is a word of opposite meaning, so we need the word for the figure who stands against the protagonist.
3. (A) agnostic: a person who holds that the existence of God cannot be known. This concerns belief, not a role in a story, so it is not the opposite. (C) arsonist: a person who deliberately sets fire to property. Unrelated. (D) anarchist: a person who rejects government or authority. Unrelated to the protagonist's role.
4. (B) antagonist: the character or force that opposes the protagonist. This is the direct opposite in role (pro- means for or in favour of; ant(i)- means against).
5. Hence the antonym of protagonist is antagonist, option (B).

KEY IDEA

In literature, the protagonist is the main character and the antagonist is the character or force opposing them. The prefixes pro- (for) and ant(i)- (against) mark the contrast.

WATCH OUT FOR

Picking a word that merely sounds similar or starts with the same letters (agnostic, arsonist, anarchist) because all four options begin with "a". None of them has any link to the protagonist's role.

Q.2 MCQ · 1 MARK

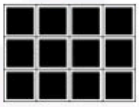
General Aptitude › Spatial

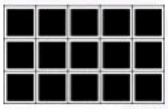
Q.2 The figure shows two 4-tile patterns.

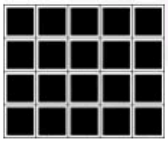


Either one or both of the patterns can be used any number of times and in any orientation to construct a new pattern. Which one of the options below **cannot** be constructed by using only these two 4-tile patterns assuming there are no overlaps among them?

(A) 

(B) 

(C) 

(D) 

Official answer: **C**

SOLUTION

1. Both building patterns, the 2×2 square and the 1×4 strip, contain exactly 4 tiles. With no overlaps, any pattern built from them has a number of tiles that is a multiple of 4.
2. Count the tiles in each option: (A) $2 \times 4 = 8$, (B) $3 \times 4 = 12$, (C) $3 \times 5 = 15$, (D) $4 \times 5 = 20$.
3. 15 is not a multiple of 4, so option (C) cannot be built from these patterns.
4. Check the others by construction. (A) Two 2×2 squares side by side fill the 2×4 rectangle.
5. (B) Three 1×4 strips, each turned to lie horizontally, stacked one above another fill the 3×4 rectangle.
6. (D) Five vertical 1×4 strips placed side by side fill the 4×5 rectangle.
7. Hence only (C) is impossible.

KEY IDEA
Every building piece has 4 tiles, so any pattern made from them has a tile count that is a multiple of 4. Count the tiles in each option first, then confirm the others by tiling.

WATCH OUT FOR
Trying to tile all four options by hand without counting first, or forgetting that the pieces may be rotated. The strip can lie horizontally, which is what makes the 3-row rectangle in (B) possible.

Q.3 MCQ · 1 MARK

General Aptitude › Numerical

Consider a knock-out women's badminton singles tournament where there are no ties. The loser in each game is eliminated from the tournament. Every player plays until she is defeated or remains the last undefeated player. The last undefeated player is declared the winner of the tournament. If there are 64 players in the beginning of the tournament, how many games should be played in total to declare the winner of the tournament?

(A) 127
(B) 64
(C) 63
(D) 32

Official answer: **C**

SOLUTION

1. Each game has exactly one loser (there are no ties), and the loser is eliminated. So every game removes exactly one player.
2. The tournament ends when only one undefeated player is left. Going from 64 players to 1 means $64 - 1 = 63$ players must be eliminated.
3. Number of games = number of eliminations = 63.
4. Check by rounds: $32 + 16 + 8 + 4 + 2 + 1 = 63$.

KEY IDEA
In a knock-out tournament each game eliminates exactly one player. Games played = players - 1, because everyone except the winner must lose exactly once.

WATCH OUT FOR
Adding the players as if every player played a separate game, or doubling up to get 127 or 64. The question asks for games, not players: a game involves two players but eliminates only one.

Q.4 MCQ · 1 MARK

General Aptitude › Numerical

A student needs to enroll for a minimum of 60 credits. A student cannot enroll for more than 70 credits. The credits are divided amongst project and three distinct sets of courses namely, core courses, specialization courses, and elective courses. It is compulsory for a student to enroll for exactly 15 credits of core courses and exactly 20 credits of project. In addition, a student has to enroll for a minimum of 10 credits of specialization courses. The maximum credits of elective courses that a student can enroll for is _____

(A) 10
(B) 15
(C) 20

(D) 25

Official answer: **D**

SOLUTION

1. Let the credits be core C, project P, specialization S and elective E. The total is $T = C + P + S + E$, with $60 \le T \le 70$.
2. The fixed values are $C = 15$ and $P = 20$, so $C + P = 35$.
3. Specialization has a minimum: $S \ge 10$.
4. Then $E = T - 35 - S$. To make E as large as possible, take the largest total T and the smallest S.
5. The largest total is $T = 70$ and the smallest specialization is $S = 10$.
6. $$E_{\max} = 70 - 15 - 20 - 10 = 25.$$
7. Check: $15 + 20 + 10 + 25 = 70$. This does not exceed 70 and is at least 60, so the choice is allowed.

KEY IDEA

To maximise one quantity under a total cap, set the total to its upper limit and every other component to its minimum. The fixed parts are subtracted from the cap, and a lower bound on another part is used at its lowest value.

WATCH OUT FOR

Subtracting 15 + 20 from the minimum total of 60 gives 25 minus the specialization minimum, which is 15 (option B). The maximum elective credits must be found against the maximum total of 70, not the minimum of 60.

Q.5 MCQ · 1 MARK

General Aptitude › Reasoning

'When the teacher is in the room, all students stand silently.'
If the above statement is true, which one of the following statements is not necessarily true?

(A) If any student is not standing silently, then the teacher is not in the room.

(B) When the teacher is in the room, all students are silent.

(C) If all students are standing, then the teacher is in the room.

(D) When the teacher is in the room, all students are standing.

Official answer: **C**

SOLUTION

1. Write the given statement as an implication $T \Rightarrow S$, where T = the teacher is in the room and S = all students stand silently (that is, every student is standing and every student is silent).
2. Since S is a conjunction of "all students are standing" and "all students are silent", the given statement also gives $T \Rightarrow \text{all standing}$ and $T \Rightarrow \text{all silent}$.
3. (B) When the teacher is in the room, all students are silent. This follows from step 2, so it is necessarily true.
4. (D) When the teacher is in the room, all students are standing. This also follows from step 2, so it is necessarily true.
5. (A) If any student is not standing silently, then the teacher is not in the room. This is the contrapositive $\neg S \Rightarrow \neg T$ of the given statement, and a statement is always equivalent to its contrapositive, so it is necessarily true.
6. (C) If all students are standing, then the teacher is in the room. This has the form $\text{standing} \Rightarrow T$, the converse of part of the given statement. A statement does not imply its converse: the given statement says nothing about what happens when the teacher is absent, so students may well be standing with no teacher in the room. This is not necessarily true.
7. Hence the statement that is not necessarily true is (C).

KEY IDEA
An implication $P \Rightarrow Q$ is equivalent to its contrapositive $\neg Q \Rightarrow \neg P$ and implies each part of a conjunction Q. It does not imply its converse $Q \Rightarrow P$.

WATCH OUT FOR
Reading the statement as "if and only if", so that students standing is taken to show the teacher is present. The statement only says what happens when the teacher is in the room, so its converse need not hold.

Q.6 MCQ · 2 MARKS

General Aptitude › Reading

Combinatorics deals with problems involving counting. For example, “How many distinct arrangements of N distinct objects in M spaces on a circle are possible?” is a typical problem in combinatorics. This kind of counting is sometimes used in the modeling of several physical phenomena. Often, in such models, the different combinatorial possibilities are assigned probability values. Assigning probabilities enables the computation of the average values of physical quantities.

Consider the following statements:

P: Combinatorics is always invoked in the modeling of physical phenomena.

Q: Modeling some physical phenomena involves assigning probabilities to combinatorial possibilities in order to compute average values of physical quantities.

Based on the passage above, what can be inferred about statements P and Q?

(A) P is False and Q is False

(B) P is False and Q is True

(C) P is True and Q is False

(D) P is True and Q is True

Official answer: **B**

SOLUTION

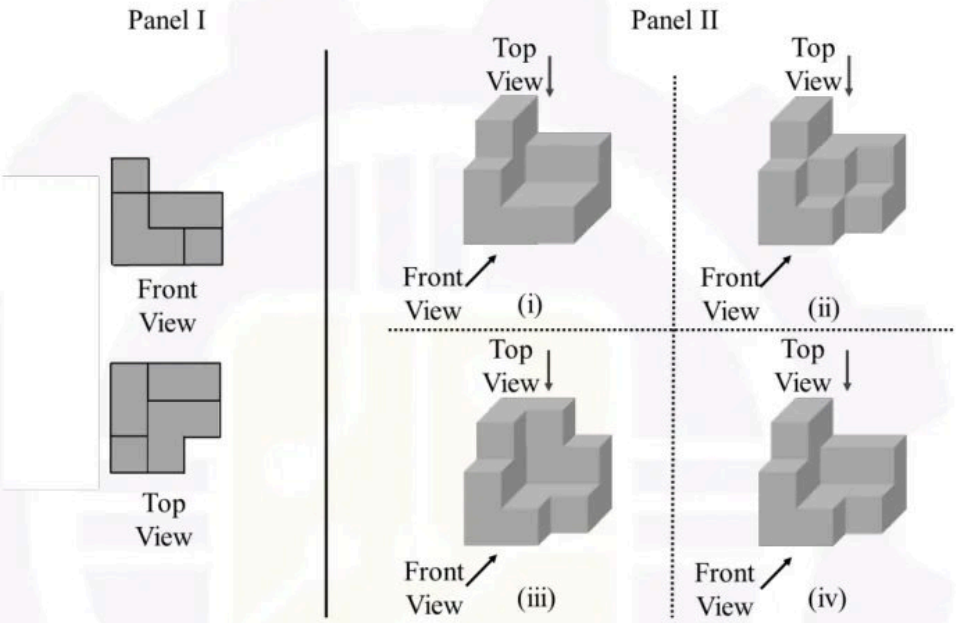
1. Statement P says combinatorics is **always** invoked in modeling physical phenomena. The passage says only that this kind of counting is "sometimes used in the modeling of several physical phenomena". "Sometimes" does not support "always", so P cannot be inferred. P is False.
2. Statement Q says that modeling some physical phenomena involves assigning probabilities to combinatorial possibilities in order to compute average values of physical quantities.
3. The passage says that "often, in such models, the different combinatorial possibilities are assigned probability values" and that "assigning probabilities enables the computation of the average values of physical quantities". If this happens often, it happens in some models. Q is supported, so Q is True.
4. P is False and Q is True, which is option (B).

KEY IDEA
In inference questions, compare quantifier words. "Always" is stronger than the passage's "sometimes", so it is not supported. "Some" is weaker than "often", so it is supported.

WATCH OUT FOR
Treating the passage's topic as support for a universal claim and choosing P True, or rejecting Q because it does not repeat the passage's word "often".

Q.7 MCQ · 2 MARKS

General Aptitude › Spatial

Q.7	In Panel I of the figure below, the front view and top view of a structure are shown. Which one of the 3D structures shown in Panel II possesses the views shown in Panel I? 
(A)	(i)
(B)	(ii)
(C)	(iii)
(D)	(iv)

Official answer: **D**

Q.8 MCQ · 2 MARKS

General Aptitude › Data interpretation

Q.8

For positive real numbers S and K, the function $H_K(S)$ is defined as: $H_K(S) = \max(S - K, 0)$. The max function is defined as:

$$\max(a, b) = \begin{cases} a, & \text{when } a > b \\ b, & \text{when } a \leq b \end{cases}$$

The graph below shows the plot of a function $N(S)$ versus S.

$N(S)$ can be expressed as _____.



(A) $H_{10}(S) - H_{20}(S)$

(B) $H_{10}(S) - 2H_{20}(S)$

(C) $-H_{10}(S) + H_{20}(S)$

(D) $H_{15}(S) - H_{20}(S)$

Official answer: **A**

SOLUTION

1. Read the graph: $N(S) = 0$ for $S \leq 10$; from $(10, 0)$ to $(20, 10)$ it is a straight line of slope $\dfrac{10 - 0}{20 - 10} = 1$; for $S \geq 20$ it stays at 10.
2. By definition, $H_K(S) = \max(S - K, 0)$ equals 0 for $S \leq K$ and $S - K$ for $S > K$. It is a ramp of slope 1 that starts at $S = K$.
3. A ramp of slope 1 starting at $S = 10$ is $H_{10}(S)$. It matches the graph for $S \leq 20$.

4. To make the graph flat after $S = 20$, subtract a slope-1 ramp that starts at $S = 20$, which is $H_{20}(S)$. This suggests $N(S) = H_{10}(S) - H_{20}(S)$.
5. Check $S \leq 10$: $H_{10} - H_{20} = 0 - 0 = 0$. Check $10 < S \leq 20$: $(S - 10) - 0 = S - 10$, which is 0 at $S = 10$ and 10 at $S = 20$, a line of slope 1.
6. Check $S > 20$: $(S - 10) - (S - 20) = 10$, a constant, which matches the graph.
7. Other options: (B) gives $(S - 10) - 2(S - 20) = 30 - S$ for $S > 20$, which decreases, so it is wrong. (C) is the negative of (A), so it is ≤ 0 everywhere, so it is wrong. (D) starts rising at $S = 15$ and gives 5 for $S > 20$, but the graph starts rising at 10 and ends at 10, so it is wrong.
8. Therefore the answer is (A) $H_{10}(S) - H_{20}(S)$.

KEY IDEA

$H_K(S)$ is a slope-1 ramp that starts at $S = K$. Subtracting a ramp that starts later cancels the slope from that point on, so $H_{10} - H_{20}$ rises from 10 to 20 and then stays flat at 10.

WATCH OUT FOR

Choosing (C), which has the right ramps in the wrong order and so is non-positive. Or choosing (B), where subtracting two ramps makes the graph turn downward instead of flat.

Q.9 MCQ · 2 MARKS

General Aptitude › Reasoning

In the 2020 summer Olympics' Javelin throw finals, Neeraj Chopra exhibited a spectacular performance to win the gold medal. The silver medal was won by Jakub Vadlejch and the bronze medal was won by Vitezlav Vesely. There were six rounds of throws with each athlete having one throw per round. The best of all the throws of each athlete is considered for the medal. Following were the observations about the throws:

i. The first and second rounds were dominated by Neeraj Chopra with a gold medal performance in his second throw, while the other two athletes did not have any medal winning throws in these rounds.

ii. The throws in the last round by both Jakub Vadlejch and Vitezlav Vesely were fouls and were not considered for scoring.

iii. After four rounds, Vitezlav Vesely was in the second position and could not improve upon his best throw in the succeeding rounds.

iv. In the fourth round, the throw by Jakub Vadlejch was the best in that round.

In which round did Vitezlav Vesely have his best throw?

(A) Third

(B) Fourth

(C) Fifth

(D) Sixth

Official answer: **A**

SOLUTION

1. By (iii), Vesely could not improve on his best throw after round 4, so his best throw came in rounds 1 to 4. Round 6 was a foul (ii), and any throw after round 4 cannot be his best. This rules out rounds 5 and 6.
2. By (i), Vesely had no medal-winning throw in rounds 1 and 2. His best throw is the one that won him the bronze medal, so it is not in round 1 or 2. That leaves round 3 or round 4.
3. Test round 4. Suppose Vesely's best throw was in round 4. By (iv), Vadlejch's round-4 throw was the best in that round, so it was longer than Vesely's round-4 throw, which is Vesely's best.
4. Then after four rounds Vadlejch's best throw is at least as long as that round-4 throw, so it is longer than Vesely's best. Vadlejch would be ahead of Vesely. This contradicts (iii), which puts Vesely in

second position after four rounds.

5. So round 4 is impossible, and Vesely's best throw was in the third round. The answer is (A) Third.

KEY IDEA

Use elimination. Statement (iii) limits the best throw to rounds 1 to 4, statement (i) removes rounds 1 and 2, and statements (iii) and (iv) together remove round 4. Only round 3 is left.

WATCH OUT FOR

Picking round 4 because the observations name it. Statement (iv) shows that Vadlejch out-threw Vesely in that round, so Vesely's best there would put Vadlejch ahead of him and contradict his second place after four rounds.

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 number appearing in the second roll is an integer multiple of the number appearing in the first roll is _____

(A) $\frac{1}{6}$

(B) $\frac{5}{18}$

(C) $\frac{7}{18}$

(D) $\frac{5}{6}$

Official answer: **C**

SOLUTION

1. Two rolls give $6 \times 6 = 36$ equally likely ordered outcomes (a, b), where a is the first roll and b the second.
2. We need b to be an integer multiple of a, so count the favourable outcomes by fixing the first roll a.
3. $a = 1$: every $b \in \{1, \ldots, 6\}$ works, so 6 outcomes.
4. $a = 2$: $b \in \{2, 4, 6\}$, so 3 outcomes.
5. $a = 3$: $b \in \{3, 6\}$, so 2 outcomes.
6. $a = 4$, $a = 5$, $a = 6$: the next multiple (8, 10, 12) exceeds 6, so only $b = a$ works, giving 1 outcome each, 3 in all.
7. Favourable outcomes $= 6 + 3 + 2 + 1 + 1 + 1 = 14$.
8. $P = \frac{14}{36} = \frac{7}{18}$, which is option (C).

KEY IDEA

Fix the first roll a and count the values $b \leq 6$ that are multiples of a, which is $\lfloor 6/a \rfloor$. Add these counts over $a = 1, \ldots, 6$ and divide by 36.

WATCH OUT FOR

Counting only pairs with $b > a$, or leaving out the case $b = a$ (every number is a multiple of itself). Another slip is counting the 14 favourable pairs as unordered, or dividing by 6 instead of 36.

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

Q.11 MCQ · 1 MARK

Engineering Mathematics › Probability & statistics

An urn contains one red ball and one blue ball. At each step, a ball is picked uniformly at random from the urn, and this ball together with another ball of the same color is put back in the urn. The probability that there are equal number of red and blue balls after two steps is

(A) 1/4

(B) 1/3

(C) 1/2

(D) 2/3

Official answer: **B**

SOLUTION

1. Each step adds one ball, so after two steps the urn holds 4 balls. Equal numbers means exactly 2 red and 2 blue, so one step must add a red ball and the other a blue ball.
2. Case 1, red then blue. Step 1: $P(\text{red}) = \frac{1}{2}$, and the urn becomes 2 red, 1 blue. Step 2: $P(\text{blue}) = \frac{1}{3}$. The product is $\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$.
3. Case 2, blue then red. By symmetry the urn becomes 1 red, 2 blue, and then $P(\text{red}) = \frac{1}{3}$. The product is $\frac{1}{2} \cdot \frac{1}{3} = \frac{1}{6}$.
4. The two cases are mutually exclusive, so add them using the total probability rule: $\frac{1}{6} + \frac{1}{6} = \frac{1}{3}$.

KEY IDEA

This is a Polya urn. After two draws the urn has 4 balls, so equality needs exactly one red addition and one blue addition. Multiply along each ordering using the urn's composition at each step, then add the two orderings.

WATCH OUT FOR

Using $\frac{1}{2}$ for the second draw as well, which gives $\frac{1}{2}$ in total. The urn changes after the first step: the drawn colour's count rises, so the second-draw probabilities are $\frac{1}{3}$ and $\frac{2}{3}$, not $\frac{1}{2}$.

Q.12 MCQ · 1 MARK

Discrete Mathematics › Counting

Consider 4×4 matrices with their elements from $\{0, 1\}$. The number of such matrices with even number of 1s in every row and every column is

(A) 512

(B) 1025

(C) 1023

(D) 255

Official answer: **A**

SOLUTION

1. Every row sum and every column sum must be even, that is $\equiv 0 \pmod 2$.
2. Fill the top-left 3×3 block with any entries from $\{0, 1\}$. It has 9 cells, so there are 2^9 choices.
3. Entries a_{14}, a_{24}, a_{34} are each forced: each is the parity bit that makes its own row sum even.
4. Entries a_{41}, a_{42}, a_{43} are each forced the same way: each is the parity bit that makes its own column sum even.
5. Choose a_{44} so that row 4 has an even sum. Then rows 1 to 4 are all even, so the sum of all 16 entries is even. The first three columns each have an even sum, so column 4 has sum $(\text{total}) - (\text{columns } 1 \text{ to } 3)$, which is even. So this one choice of a_{44} satisfies row 4 and column 4 together.

6. Hence every choice of the 3×3 block extends to exactly one valid matrix. Conversely, any valid matrix is fixed by its top-left 3×3 block, because all the other entries were forced above. Distinct blocks give distinct matrices.
7. Count $= 2^{(4-1)(4-1)} = 2^9 = 512$. Check on a 2×2 matrix: the formula gives $2^1 = 2$, and the valid matrices are the all-zero matrix and the all-one matrix. This agrees.

KEY IDEA

Parity constraints on all rows and columns are fixed by the free top-left $(n-1) \times (n-1)$ block. The last row and last column are forced, and the corner is consistent automatically. The count is $2^{(n-1)^2}$.

WATCH OUT FOR

Counting the corner a_{44} as a separate free or doubly constrained choice, or treating the row and column constraints as independent and getting $2^{16}/2^8 = 256$. The 8 constraints are not independent, because the sum of all row parities equals the sum of all column parities. Only 7 are independent, so the count is $2^{16-7} = 2^9$.

Q.13 MCQ · 1 MARK

Engineering Mathematics › Linear algebra

For $n > 1$, the maximum multiplicity of any eigenvalue of an $n \times n$ matrix with elements from $\mathbb{R}$ is

(A) n

(B) $n - 1$

(C) 1

(D) $n + 1$

Official answer: **A**

SOLUTION

1. The eigenvalues of an $n \times n$ real matrix M are the roots of its characteristic polynomial $\det(M - \lambda I)$, which has degree exactly n.
2. The multiplicity of an eigenvalue is its multiplicity as a root of this polynomial. A degree-n polynomial has at most n roots counted with multiplicity, so no eigenvalue can have multiplicity greater than n. This rules out $n + 1$.
3. The bound n is attained. For $M = I_n$ (or $M = cI_n$ with c real), $\det(M - \lambda I) = (1 - \lambda)^n$, so the eigenvalue 1 has multiplicity n. This holds for every $n > 1$, so the maximum is not $n - 1$ or 1.
4. A non-diagonalizable example also reaches n. A single $n \times n$ Jordan block with real diagonal entry c has characteristic polynomial $(c - \lambda)^n$, so its eigenvalue c again has multiplicity n.
5. Hence the maximum possible multiplicity of an eigenvalue is n, which is option (A).

KEY IDEA

Multiplicity is a root multiplicity of the degree-n characteristic polynomial, so it is at most n. A scalar matrix attains this bound.

WATCH OUT FOR

Thinking that a repeated eigenvalue of multiplicity n forces the matrix to be non-diagonalizable or to have fewer independent eigenvectors, and choosing $n - 1$ or 1. Multiplicity n is attained by I_n itself, and the identity is diagonalizable.

Q.14 MCQ · 1 MARK

Computer Organization and Architecture › Instructions & addressing

Q.14 Match each addressing mode in **List I** with a data element or an element of a data structure (in a high-level language) in **List II**:

List I	List II
P. Immediate	1. Element of an array
Q. Indirect	2. Pointer
R. Base with index	3. Element of a record
S. Base with offset/displacement	4. Constant

(A) P–4, Q–3, R–1, S–2

(B) P–4, Q–2, R–1, S–3

(C) P–1, Q–4, R–3, S–2

(D) P–2, Q–3, R–1, S–4

Official answer: **B**

SOLUTION

1. P. Immediate addressing: the operand's value is embedded directly in the instruction itself, with no memory reference needed; this is exactly how a constant/literal is used in a high-level language. So P → 4 (Constant).
2. Q. Indirect addressing: the instruction's address field holds the address of a memory location that itself holds the operand's address (or, in single-level indirect, holds the address of the operand); the defining property is that you dereference through a stored address to reach the data. This is exactly how a pointer variable behaves: the pointer stores an address, and you follow it to get the value. So Q → 2 (Pointer).
3. R. Base with index addressing: effective address = base register + index register, where the index register's value is typically varied at run time (e.g., incremented in a loop). This models array access: the base register holds the array's starting address and the index register holds the element number, so varying the index steps through consecutive elements. So R → 1 (Element of an array).
4. S. Base with offset/displacement addressing: effective address = base register + a fixed (compile-time constant) displacement encoded in the instruction. This models accessing a field at a fixed, known offset within a structure/record: the base register holds the record's starting address and the displacement is the field's fixed position within it. So S → 3 (Element of a record).
5. Combining: P–4, Q–2, R–1, S–3, which is exactly option (B).

KEY IDEA

Each addressing mode's structural definition maps directly onto a matching high-level-language construct: immediate (value in the instruction) ↔ constant; indirect (dereference through a stored address) ↔ pointer; base+index (base + a value that varies to step through elements) ↔ array element; base+displacement (base + a fixed, compile-time offset) ↔ record/structure field.

WATCH OUT FOR

Swapping base-with-index and base-with-offset/displacement (R and S): both add a base register to something, so it's easy to conflate them. The distinguishing feature is whether that "something" varies at run time (index register → array traversal) or is a fixed constant baked into the instruction (displacement → a specific, unchanging field offset in a record).

Q.15 MCQ · 1 MARK

Computer Organization and Architecture › Instructions & addressing

Consider a processor P whose instruction set architecture is the load-store architecture. The instruction format is such that the first operand of any instruction is the destination operand.

Which one of the following sequences of instructions corresponds to the high-level language statement $Z = X + Y$?

Note: X, Y, and Z are memory operands. R0, R1, and R2 are registers.

(A) ADD Z, X, Y

(B)
LOAD R0, X
ADD Z, R0, Y

(C)
ADD R0, X, Y
STORE Z, R0

(D)
LOAD R0, X
LOAD R1, Y
ADD R2, R0, R1
STORE Z, R2

Official answer: **D**

SOLUTION

1. In a load-store architecture, the memory can be touched only by LOAD (memory to register) and STORE (register to memory); every other instruction, including ADD, must operate purely on registers.
2. Option (A) `ADD Z, X, Y` performs an ALU operation with all three operands in memory. This is an ALU instruction directly on memory operands, which a load-store architecture forbids. Eliminated.
3. Option (B): `LOAD R0, X` brings X into R0, but the following `ADD Z, R0, Y` still writes its result directly to memory (Z) and reads one operand (Y) directly from memory. An ADD may not touch memory at all in a load-store machine. Eliminated.
4. Option (C): `ADD R0, X, Y` reads both X and Y directly from memory (only the destination R0 is a register), which is disallowed for an ALU instruction; `STORE Z, R0` alone does not fix the illegal ADD. Eliminated.
5. Option (D): `LOAD R0, X` (memory to register, allowed), `LOAD R1, Y` (memory to register, allowed), `ADD R2, R0, R1` (all three operands are registers, no memory access, allowed), `STORE Z, R2` (register to memory, allowed). Every instruction obeys the load-store constraint, and the first operand is consistently the destination throughout.
6. Only sequence (D) is fully compliant with the load-store architecture.

KEY IDEA
In a load-store (register-register) architecture, only LOAD and STORE instructions may reference memory; every arithmetic/logic instruction (like ADD) must have all its operands, source and destination, in registers.

WATCH OUT FOR
Assuming an ADD can have one memory operand as long as it also uses a register (as in options B and C), rather than recognizing that a load-store ISA forbids ANY memory operand on a non-load/store instruction.

Q.16 MCQ · 1 MARK

Computer Organization and Architecture › Pipelining

Which one of the following dependencies among the register operands of different instructions can cause a data hazard in a pipelined processor?

(A) Read-after-read

(B) Read-after-write

(C) Write-after-read

(D) Write-after-write

Official answer: **B**

SOLUTION

1. A data hazard occurs when overlapping (pipelined) execution of instructions can change the outcome relative to strictly sequential execution, due to a dependency through a shared register operand.
2. Read-after-read (RAR): both instructions only read the register. Reads don't change the register's value, so the order in which they read it doesn't matter: no hazard.
3. Write-after-read (WAR) and write-after-write (WAW): in a simple single-issue, in-order pipeline, every instruction reads its operands (ID stage) and writes its result (WB stage) strictly in program order, so a later instruction can never overtake an earlier one's read or write. WAR/WAW can only cause trouble under out-of-order or overlapped-write execution, not in this standard in-order pipeline, so they are not the answer here.
4. Read-after-write (RAW): a later instruction needs to read a register that an earlier instruction has not yet finished computing/writing. If the read (in ID) happens before the earlier instruction's write (in WB) has completed, the pipeline reads a stale value, producing a wrong result unless stalled or the value is forwarded. This is a genuine data hazard even in a simple in-order pipeline.
5. Since RAW is the only one of the four dependencies that can actually corrupt a result under ordinary in-order pipelined execution, it is the dependency that causes a data hazard.
6. Hence the answer is Read-after-write, option (B).

KEY IDEA
In a simple in-order pipeline, only a true (flow) dependency, a later instruction reading a register before an earlier instruction has finished writing it, can produce an incorrect result; this is the Read-After-Write (RAW) hazard.

WATCH OUT FOR
Confusing RAW (a true data dependency that causes a real hazard even in a simple in-order pipeline) with WAR/WAW (anti- and output-dependencies, which only matter under out-of-order or overlapped-write execution), or picking RAR by mistakenly treating any repeated access as hazardous.

Q.17 MCQ · 1 MARK

Data Structures and Algorithms › Complexity

Consider the following recurrence relations:

For all $n > 1$,

$$T_1(n) = 4T_1\left(\frac{n}{2}\right) + T_2(n)$$

$$T_2(n) = 5T_2\left(\frac{n}{4}\right) + \Theta(\log_2 n)$$

Assume that for all $n \leq 1$, $T_1(n) = 1$ and $T_2(n) = 1$.

Which one of the following options is correct?

(A) $T_1(n) = \Theta(n^2)$

(B) $T_1(n) = \Theta(n^2 \log_2 n)$

(C) $T_1(n) = \Theta(n^{\log_4 5})$

(D) $T_1(n) = \Theta(n^{\log_4 5} \log_2 n)$

Official answer: **A**

SOLUTION

1. Solve the inner recurrence T2(n) = 5T2(n/4) + Θ(log₂n) via the Master theorem: a=5, b=4, so the watershed function is n^{log_4 5} ≈ n^{1.161}.
2. Since f(n) = Θ(log₂n) = O(n^{log_4 5 - ε}) for ε = 1 > 0, Master theorem Case 1 applies, giving T2(n) = Θ(n^{log_4 5}).
3. Substitute this closed form into the outer recurrence: T1(n) = 4T1(n/2) + Θ(n^{log_4 5}).
4. Apply the Master theorem to T1: a=4, b=2, so the watershed function is n^{log_2 4} = n².
5. Compare f(n) = Θ(n^{log_4 5}) ≈ Θ(n^{1.161}) with n²: since log_4 5 < 2, f(n) = O(n^{2-ε}) for ε = 2 - log_4 5 > 0, so Master theorem Case 1 applies again.
6. Therefore T1(n) = Θ(n²), which is option (A).

KEY IDEA

Solve the inner recurrence T2 first with the Master theorem, then substitute the resulting Θ-bound into the outer recurrence T1 and apply the Master theorem a second time.

WATCH OUT FOR

Applying the Master theorem to T1 using the printed f(n) = T2(n) symbolically (or assuming T2(n) = Θ(n) / Θ(log n) without first solving it), instead of substituting the actual closed form Θ(n^{log_4 5}) for T2(n) before comparing against n^{log_2 4} = n².

Q.18 MCQ · 1 MARK

Computer Networks › TCP

With respect to a TCP connection between a client and a server, which one of the following statements is true?

(A) The client and server use a two-way handshake mechanism before the start of data transmission

(B) The server cannot initiate closing of the connection before the client initiates closing of the connection

(C) The TCP connection is half-duplex

(D) The client and server can initiate closing of the connection at the same time

Official answer: **D**

SOLUTION

1. Option (A): TCP opens a connection with a **three-way handshake** (SYN, SYN-ACK, ACK), not a two-way one. False.
2. Option (C): a TCP connection carries data in both directions at the same time, so it is **full-duplex**, not half-duplex. False.
3. Option (B): closing is symmetric. Each direction is closed independently by its own FIN, and either side may send the first FIN, the server included. So the server can start closing before the client does. False.
4. Option (D): if both sides send a FIN at about the same time, each moves from `ESTABLISHED` to `FIN_WAIT_1`. Each then receives the other's FIN before its own FIN is acknowledged, and moves to

CLOSING . After the ACKs arrive, each moves to TIME_WAIT . This is the **simultaneous close**, and TCP defines it. True.

5. Hence only (D) is a true statement.

KEY IDEA

TCP opens with a three-way handshake, is full-duplex, and closes each direction independently with its own FIN. Either side may send the first FIN, and both may send one at once (simultaneous close, via the `CLOSING` state).

WATCH OUT FOR

Thinking the client must always close first because it opened the connection. Close is not tied to who opened. Another slip is mixing up the three-way open with the four-segment close.

Q.19 MSQ · 1 MARK

Computer Networks › DNS & HTTP

Which of the following statements is/are true with respect to the interaction of a web browser with a web server using HTTP 1.1?

(A) HTTP 1.1 facilitates downloading multiple objects of the same webpage over the same TCP connection, if the objects are stored in the same server

(B) HTTP 1.1 facilitates downloading multiple objects of the same webpage over the same TCP connection, even if they are stored in different servers

(C) HTTP 1.1 facilitates sending a request for downloading one object without waiting for a previously requested object to be downloaded completely

(D) HTTP 1.1 facilitates downloading multiple webpages on the same server to be downloaded over a single TCP connection

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

SOLUTION

1. HTTP 1.1 uses persistent connections by default. The TCP connection stays open after a response, so later requests reuse it instead of opening a new connection for each object.
2. A TCP connection joins the browser to one server, identified by its IP address and port. Reuse is therefore possible only for objects held on that same server. So (A) is true and (B) is false, because objects on a different server need a separate connection.
3. Persistence is not tied to one page. The connection stays open for any further request to that server, so objects of several webpages can use it one after another. So (D) is true.
4. HTTP 1.1 also allows pipelining. The client may send several requests back to back without waiting for the earlier responses to finish, and the server returns the responses in the order the requests arrived. So (C) is true.
5. True statements: (A), (C) and (D).

KEY IDEA

HTTP 1.1 keeps one TCP connection open per server (persistent connection) and allows pipelined requests. Whatever goes over that connection, whether objects or whole pages, must come from the same server.

WATCH OUT FOR

Thinking one connection can carry objects from different servers (B). A TCP connection ends at a single server. Another slip is to treat persistence as limited to one webpage, which makes (D) look false.

Q.20 MSQ · 1 MARK

Engineering Mathematics › Linear algebra

Let $n > 1$. Consider an $n \times n$ matrix M with its elements from $\mathbb{R}$. Let the vector $(0, 1, 0, 0, \ldots, 0) \in \mathbb{R}^n$ be in the null space of M.

Which of the following options is/are always correct?

(A) Determinant of M is 1

(B) Determinant of M is 0

(C) Rank of M is 1

(D) There are at least two non-zero vectors in the null space of M

Official answer: **B, D**

SOLUTION

1. Let $v = (0, 1, 0, \ldots, 0)^T$. It is non-zero and $Mv = 0$, so the null space of M contains a non-zero vector.
2. A square matrix whose null space contains a non-zero vector is singular, so $\det M = 0$. Hence (B) is always true and (A) is always false.
3. By rank-nullity, $\text{rank}(M) = n - \text{nullity}(M) \leq n - 1$. The rank can be anything from 0 to $n - 1$.
4. Check (C) with examples. $M = 0$ has rank 0. For $n = 3$, $M = \text{diag}(1, 0, 1)$ has rank 2 and still sends v to 0. So rank 1 is not forced, and (C) is not always correct.
5. The null space is a subspace, so it is closed under scalar multiplication. For every $c \neq 0$ in $\mathbb{R}$, cv is a non-zero vector in it. So there are infinitely many non-zero vectors in the null space, in particular at least two, and (D) is always true.
6. The options that are always correct are (B) and (D).

KEY IDEA

A non-zero vector in the null space of a square matrix makes the matrix singular, so its determinant is 0. A non-trivial null space is a subspace over $\mathbb{R}$, so it has infinitely many non-zero vectors.

WATCH OUT FOR

Reading "at least two non-zero vectors" as needing a second independent vector, or assuming the rank must be exactly 1. Only v is guaranteed to lie in the null space, so the rank can be anything from 0 to $n - 1$, but scalar multiples of v already give infinitely many non-zero vectors.

Q.21 MSQ · 1 MARK

Digital Electronics › Boolean logic

Consider the following Boolean expression of a function F:

$$F(P, Q) = (\overline{P} + Q) \oplus (\overline{P}Q)$$

Which of the following expressions is/are equivalent to F?

(A) $\overline{P \oplus Q}$

(B) $P \oplus Q$

(C) $\overline{P} \oplus Q$

(D) $\overline{P} \oplus \overline{Q}$

Official answer: **A, C**

SOLUTION

1. Evaluate $F(P, Q) = (\overline{P} + Q) \oplus (\overline{P}Q)$ for all four input pairs.
2. $P = 0, Q = 0$: $(1 + 0) \oplus (1 \cdot 0) = 1 \oplus 0 = 1$.
3. $P = 0, Q = 1$: $(1 + 1) \oplus (1 \cdot 1) = 1 \oplus 1 = 0$.
4. $P = 1, Q = 0$: $(0 + 0) \oplus (0 \cdot 0) = 0 \oplus 0 = 0$.
5. $P = 1, Q = 1$: $(0 + 1) \oplus (0 \cdot 1) = 1 \oplus 0 = 1$.

6.

P	Q	F
0	0	1
0	1	0
1	0	0
1	1	1

So $F = 1$ exactly when $P = Q$, which means $F = P \odot Q = \overline{P \oplus Q}$.

7. (A) $\overline{P \oplus Q}$ is XNOR, so it matches. (B) $P \oplus Q$ is the complement of F, so it does not match.

8. (C) $\overline{P} \oplus Q = \overline{P \oplus Q}$, because complementing one XOR input complements the output. It matches. (D) $\overline{P} \oplus \overline{Q} = P \oplus Q$, because complementing both inputs leaves XOR unchanged. It does not match.

KEY IDEA

With only two variables, a truth table settles the question quickly. Here F comes out as XNOR. Complementing one XOR input complements the output, so $\overline{P} \oplus Q$ is also XNOR.

WATCH OUT FOR

Treating $\overline{P} \oplus \overline{Q}$ as the XNOR. Complementing both inputs of an XOR leaves it unchanged, so that option is $P \oplus Q$.

Q.22 MSQ · 1 MARK

Digital Electronics › Number systems

Consider the 8-bit signed integers X, Y and Z represented using the sign-magnitude form. The binary representations of X and Y are as follows:

X: 10110100 Y: 01001100

Which of the following operations to compute Z result(s) in an arithmetic overflow?

(A) $Z = X + Y$

(B) $Z = X - Y$

(C) $Z = -X + Y$

(D) $Z = -X - Y$

Official answer: **B, C**

SOLUTION

1. Sign-magnitude with 8 bits uses 1 sign bit and 7 magnitude bits, so the representable range is $[-127, +127]$. Any result whose magnitude is 128 or more overflows.
2. $X = 10110100$: the sign bit is 1 (negative) and the magnitude is $0110100_2 = 32 + 16 + 4 = 52$, so $X = -52$.
3. $Y = 01001100$: the sign bit is 0 (positive) and the magnitude is $1001100_2 = 64 + 8 + 4 = 76$, so $Y = +76$.
4. (A) $Z = X + Y = -52 + 76 = 24$. This is within range, so there is no overflow.
5. (B) $Z = X - Y = -52 - 76 = -128$. This is below -127, so it overflows.
6. (C) $Z = -X + Y = 52 + 76 = 128$. This is above $+127$, so it overflows.
7. (D) $Z = -X - Y = 52 - 76 = -24$. This is within range, so there is no overflow.
8. Overflow occurs for options (B) and (C).

KEY IDEA

Sign-magnitude with 8 bits has only 7 magnitude bits, so the range is -127 to $+127$. Decode X and Y to decimal and check whether each result's magnitude exceeds 127.

WATCH OUT FOR

Using the two's complement range $[-128, +127]$ makes -128 look representable and drops option B. In sign-magnitude, -128 has no encoding. Another slip is reading X as $-(\text{two's complement})$ instead of sign bit plus magnitude.

Q.23 MSQ · 1 MARK

Data Structures and Algorithms › Trees & heaps

Let n be an odd number greater than 100. Consider a binary minheap with n elements stored in an array P whose index starts from 1.

Which of the following indices of P do/does NOT correspond to any leaf node of the minheap?

(A) $\frac{n+1}{2}$

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

(C) $\frac{n-3}{2}$

(D) n

Official answer: **B, C**

SOLUTION

1. Rule: in a 1-indexed array-stored binary heap of n nodes, node i has children at 2i and 2i+1. Node i is internal iff 2i <= n, i.e., i <= floor(n/2); node i is a leaf iff i > floor(n/2).
2. Since n is odd, floor(n/2) = (n-1)/2. So internal (non-leaf) indices run from 1 to (n-1)/2, and leaf indices run from (n-1)/2 + 1 = (n+1)/2 up to n.
3. Option (A) (n+1)/2 equals floor(n/2) + 1, the very first leaf index, so it IS a leaf: it is NOT one of the required answers.
4. Option (B) (n-1)/2 equals floor(n/2) exactly, the last internal node, so it is NOT a leaf: this IS one of the required answers.
5. Option (C) (n-3)/2 equals floor(n/2) - 1, strictly inside the internal range, so it is NOT a leaf: this IS one of the required answers.
6. Option (D) n is the last stored element; its would-be children 2n and 2n+1 both exceed n, so it has no children and IS a leaf: it is NOT one of the required answers.
7. Sanity check with n = 101: floor(101/2) = 50, so leaves are indices 51 through 101. (A) 51 is a leaf. (B) 50 is internal (not a leaf). (C) 49 is internal (not a leaf). (D) 101 is a leaf. This confirms that (B) and (C) are exactly the indices that do NOT correspond to a leaf node.

KEY IDEA

In a 1-indexed array representation of an n-node binary heap, node i is internal iff $1 \le i \le \left\lfloor \frac{n}{2} \right\rfloor$ (its children $2i, 2i+1$ still lie within $1..n$), and node i is a leaf iff $i > \left\lfloor \frac{n}{2} \right\rfloor$. For odd n, $\left\lfloor \frac{n}{2} \right\rfloor = \frac{n-1}{2}$, so the last internal index is $\frac{n-1}{2}$ and the first leaf index is $\frac{n+1}{2}$.

WATCH OUT FOR

Confusing the leaf-boundary index (the first leaf) with the last internal node. Since $\left\lfloor \frac{n}{2} \right\rfloor$ is itself the LAST internal node (not a leaf), and $\left\lfloor \frac{n}{2} \right\rfloor + 1$ is the FIRST leaf, it is easy to mislabel option (A), $\frac{n+1}{2}$, as non-leaf, or to mislabel (B), $\frac{n-1}{2}$, as a leaf, by an off-by-one slip in the floor computation.

Q.24 MSQ · 1 MARK

Data Structures and Algorithms › Sorting & hashing

Consider a hash table $P[0, 1, \ldots, 10]$ that is initially empty. The hash table is maintained using open addressing with linear probing. The hash function used is $h(x) = (x + 7) \bmod 11$.

Consider the following sequence of insertions performed on P:

$$1, 13, 22, 15, 11, 24$$

Which of the following positions in the hash table is/are empty after these insertions are performed?

(A) 0

(B) 10

(C) 2

(D) 1

Official answer: **C**

SOLUTION

1. The hash table P[0..10] has size 11, all slots initially empty. h(x) = (x+7) mod 11.
2. Insert 1: h(1) = 8 mod 11 = 8. Slot 8 is empty → P[8] = 1.
3. Insert 13: h(13) = 20 mod 11 = 9. Slot 9 is empty → P[9] = 13.
4. Insert 22: h(22) = 29 mod 11 = 7. Slot 7 is empty → P[7] = 22.
5. Insert 15: h(15) = 22 mod 11 = 0. Slot 0 is empty → P[0] = 15.
6. Insert 11: h(11) = 18 mod 11 = 7. Slot 7 is occupied (22); probe slot 8 (occupied by 1), slot 9 (occupied by 13), slot 10 (empty) → P[10] = 11.
7. Insert 24: h(24) = 31 mod 11 = 9. Slot 9 is occupied (13); probe slot 10 (occupied by 11), slot 0 (occupied by 15), slot 1 (empty) → P[1] = 24.
8. Final table: P[0]=15, P[1]=24, P[2..6]=empty, P[7]=22, P[8]=1, P[9]=13, P[10]=11.
9. Empty positions are {2, 3, 4, 5, 6}. Among the given options (0, 10, 2, 1), only position 2 (option C) is empty; the answer is (C).

KEY IDEA

Simulate linear probing step by step: compute h(x)= (x+7) mod 11 for each key, and if that slot is occupied, probe the next slots (wrapping around) until an empty one is found.

WATCH OUT FOR

Forgetting to wrap the probe sequence around from index 10 back to index 0 (linear probing is circular), or mis-tracking which slots are already occupied by the time a later collision occurs.

Q.25 MSQ · 1 MARK

Theory of Computation › CFG & PDA

Consider the following grammar where S is the start symbol, and a and b are terminal symbols.

$$S \to aSbS \mid bS \mid \varepsilon$$

Which of the following statements is/are true?

(A) The grammar is ambiguous

(B) The string abb has two distinct derivations in this grammar

(C) The string $abab$ has only one rightmost derivation

(D) The language generated by the grammar is undecidable

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

SOLUTION

1. The grammar is $S \to aSbS \mid bS \mid \varepsilon$. A nonempty string that starts with b can only come from $S \to bS$. A string that starts with a can only come from $S \to aSbS$, which splits it as $w = a\,x\,b\,y$, where x and y are both strings that S derives. Because $S \to bS$ can produce extra b's anywhere, x may itself contain unmatched b's. So the b written in $aSbS$ is not forced to be one particular b of w.
2. Option (B), the string abb. It starts with a, so the first step is $S \Rightarrow aSbS$, and we need $x\,b\,y = bb$. There are two splits, and both work. Split 1: $x = \varepsilon$, $y = b$ (from $S \to bS \to b$). Split 2: $x = b$ (from $S \to bS \to b$), $y = \varepsilon$.
3. These give two different leftmost derivations of abb:

$$S \Rightarrow aSbS \Rightarrow abS \Rightarrow abbS \Rightarrow abb$$
$$S \Rightarrow aSbS \Rightarrow abSbS \Rightarrow abbS \Rightarrow abb$$

In the first, the inner S goes to ε and the last S produces the second b. In the second, the inner S produces the first b and the last S goes to ε. The parse trees are different, so abb has two distinct derivations. (B) is true.

4. Option (A). A grammar is ambiguous if some string has two different parse trees (equivalently, two leftmost derivations). By step 3, abb is such a string, so the grammar is ambiguous. (A) is true.
5. Option (C), the string $abab$. It starts with a, so $S \Rightarrow aSbS$ and we need $x\,b\,y = bab$. The b of the production is either the 1st or the 3rd symbol of bab. If it is the 3rd, then $x = ba$. But S cannot derive ba: after $S \to bS$ the rest would be a, and every a that S produces comes from $aSbS$, which always puts a b after it. So the b must be the 1st symbol: $x = \varepsilon$, $y = ab$.
6. Now $y = ab$ starts with a, so $S \to aSbS$ with $x'\,b\,y' = b$, which forces $x' = y' = \varepsilon$. Every choice is forced, so $abab$ has exactly one parse tree. A parse tree gives exactly one rightmost derivation:
$$S \Rightarrow aSbS \Rightarrow aSbaSbS \Rightarrow aSbaSb \Rightarrow aSbab \Rightarrow abab$$
So $abab$ has only one rightmost derivation. (C) is true.
7. Option (D). The language is generated by a context-free grammar, so it is context-free. Every context-free language is decidable: the CYK algorithm decides membership for any string. (D) is false.
8. The true statements are (A), (B) and (C).

KEY IDEA

In $S \to aSbS$, the b written in the production does not have to be the first b that closes the a. The production $S \to bS$ lets the inner S soak up extra b's. So whenever several b's follow an a, check whether more than one of them can play the part of that b. In abb two can, which gives two parse trees. In $abab$ only one can, because S cannot derive ba.

WATCH OUT FOR

Treating $S \to aSbS$ like the balanced-parentheses grammar $S \to aSbS \mid \varepsilon$, which is unambiguous, and so assuming the b in the production must be the first b that balances the a. Adding $S \to bS$ breaks this: in abb, either b can be the matching one. A second slip is to answer (D) as true or uncertain. Every context-free language is decidable.

Q.26 MSQ · 1 MARK

Theory of Computation › Finite automata

Let M be a nondeterministic finite automaton (NFA) with 6 states over a finite alphabet.

Which of the following options CANNOT be the number of states in the minimal deterministic finite automaton (DFA) that is equivalent to M ?

(A) 32

(B) 65

(C) 1

(D) 128

Official answer: **B, D**

SOLUTION

1. Subset construction turns an NFA with n states into a DFA whose states are subsets of the NFA's states, so there are at most 2^n of them.
2. For $n = 6$ this gives at most $2^6 = 64$ states. The empty subset (the dead state) is already counted in the 64.
3. The minimal DFA is never larger than the DFA from the subset construction, so it has at most 64 states. Hence 65 (option B) and 128 (option D) are impossible.
4. Option C is possible. Take a 6-state NFA for Σ^*: its start state is accepting and loops on every symbol, and the other five states are unreachable. The minimal DFA for Σ^* has one state.

5. Option A is possible. Let L be the set of strings over $\{a, b\}$ whose 5th symbol from the end is a. A 6-state NFA accepts L. State q_0 loops on a and b and moves to q_1 on a. States $q_1, \ldots, q_4$ each move to the next state on both a and b. State q_5 is accepting. The minimal DFA for L must remember the last 5 symbols, so it has exactly $2^5 = 32$ states.
6. So 32 and 1 can occur, and 65 and 128 cannot. The options that CANNOT be the size of the minimal DFA are B and D.

KEY IDEA
An NFA with n states gives a DFA with at most 2^n states. The minimal DFA can be anywhere from 1 up to 2^n states, so $2^6 = 64$ is the upper limit.

WATCH OUT FOR
Forgetting that the bound is 2^n and not $2^n + 1$ (the dead state is the empty subset, already counted), or assuming the minimal DFA must be at least as large as the NFA.

Q.27 MSQ · 1 MARK

Compiler Design › Lexical analysis

Consider the following C statements:

```
 char *str1 = "Hello;    /* Statement S1 */
char *str2 = "Hello;";  /* Statement S2 */
int *str3 = "Hello";    /* Statement S3 */
```

Which of the following options is/are correct?

(A) S1 and S2 have syntactic errors

(B) S2 has a lexical error and S3 has a syntactic error

(C) S1 has a lexical error and S3 has a semantic error

(D) S1 has a syntactic error and S3 has a semantic error

Official answer: **C**

SOLUTION

1. A lexical error is a malformed token, found by the scanner. A syntactic error breaks the grammar. A semantic error is well-formed code that breaks a type or meaning rule.
2. S1: `char *str1 = "Hello; /* Statement S1 */` opens a string literal with `"` but has no closing `"` before the end of the line. The scanner cannot form a valid string-literal token, so this is a lexical error. The `/*` sits inside the unfinished literal, so it does not start a comment.
3. S2: `"Hello;"` is a complete string literal, because the `;` is an ordinary character inside it. The statement continues with its own `;` and then a comment, so it is fully valid. S2 has no error, which rules out A and B.
4. S3: `int *str3 = "Hello";` is a grammatically valid declaration with an initializer, so there is no syntactic error. The literal has type `char *` after array decay, and it is used to initialize an `int *`. This incompatible pointer type is a type-checking violation, which is a semantic error.
5. So S1 has a lexical error and S3 has a semantic error. This is option C. Option D is wrong because S1 fails at the token level, not at the grammar level.

KEY IDEA
Classify each error by the compiler phase that catches it. An unterminated string literal fails in the scanner (lexical). A well-formed declaration that assigns a char pointer to an int pointer fails in type checking (semantic).

WATCH OUT FOR
Calling S1 a syntactic error because the statement looks broken, or reading the `;` inside the quotes in S2 as the end of the statement and so marking S2 as an error.

Q.28 MSQ · 1 MARK

Compiler Design › Parsing

Which of the following statements is/are true?

(A) LL(1) parser uses backtracking

(B) For a grammar to be LL(1), it must be left-recursive

(C) For a grammar to be LL(1), it must be left-factored

(D) The LL(1) parsers are more powerful than the SLR parsers

Official answer: **C**

Q.29 MSQ · 1 MARK

Operating Systems › Deadlock

With respect to deadlocks in an operating system, which of the following statements is/are FALSE?

(A) Banker's algorithm is used to prevent deadlocks

(B) Deadlock formation can be prevented by ensuring that the hold and wait condition is not allowed

(C) An assignment edge in a resource allocation graph is marked from a process to a resource

(D) A safe state guarantees that all processes can finish without formation of a deadlock

Official answer: **A, C**

SOLUTION

1. The question asks for the FALSE statements, so judge each statement as true or false and report the false ones.
2. (A) Banker's algorithm checks, at each request, whether granting it leaves the system in a safe state. That is deadlock **avoidance**. Prevention works differently: it negates one of the four necessary conditions in advance. Stating that Banker's algorithm is used to prevent deadlocks is therefore false.
3. (B) The four necessary conditions are mutual exclusion, hold and wait, no preemption and circular wait. If a process is never allowed to hold one resource while waiting for another, hold and wait is negated, so deadlock cannot form. (B) is true.
4. (C) In a resource allocation graph, a request edge goes $P_i \rightarrow R_j$ and an assignment edge goes $R_j \rightarrow P_i$, from the resource to the process holding it. An assignment edge marked from a process to a resource is wrong. (C) is false.
5. (D) A safe state has a safe sequence in which each process can obtain its maximum remaining demand and finish. If the system follows that sequence, no deadlock forms. (D) is true.
6. The false statements are (A) and (C).

KEY IDEA

Banker's algorithm is deadlock avoidance, which needs a safe-state check at each request, not prevention. In a resource allocation graph an assignment edge points from the resource to the process, and a request edge points from the process to the resource.

WATCH OUT FOR

Treating avoidance and prevention as the same thing, and so calling (A) true. Reversing the direction of request and assignment edges, and so calling (C) true.

Q.30 MSQ · 1 MARK

Database Management and Warehousing › Normal forms

Let P, Q, R and S be the attributes of a relation in a relational schema. Let $X \rightarrow Y$ indicate functional dependency in the context of a relational database, where $X, Y \subseteq \{P, Q, R, S\}$.

Which of the following options is/are always true?

(A) If ($\{P, Q\} \to \{R\}$ and $\{P\} \to \{R\}$), then $\{Q\} \to \{R\}$

(B) If $\{P, Q\} \to \{R\}$, then ($\{P\} \to \{R\}$ or $\{Q\} \to \{R\}$)

(C) If ($\{P\} \to \{R\}$ and $\{Q\} \to \{S\}$), then $\{P, Q\} \to \{R, S\}$

(D) If $\{P\} \to \{R\}$, then $\{P, Q\} \to \{R\}$

Official answer: **C, D**

SOLUTION

1. Test each implication with Armstrong's axioms, or with a small counterexample relation. In a counterexample, the attribute S can be given the same value in every tuple, so it plays no part.
2. (A) $\{P\} \to \{R\}$ already implies $\{P, Q\} \to \{R\}$, so the premise tells us nothing about Q alone. Take the tuples $(P, Q, R) = (1, 1, 1),\ (1, 2, 1),\ (2, 3, 5),\ (3, 3, 6)$. Then $P \to R$ holds, because $P = 1$ gives only $R = 1$ and the other P values are distinct. So $PQ \to R$ holds too. But $Q = 3$ gives both $R = 5$ and $R = 6$, so $Q \to R$ fails. (A) is not always true.
3. (B) Take $(P, Q, R) = (1, 1, 1),\ (1, 2, 2),\ (2, 1, 2)$. All (P, Q) pairs are distinct, so $PQ \to R$ holds. But $P = 1$ gives $R = 1$ and $R = 2$, so $P \to R$ fails. Also $Q = 1$ gives $R = 1$ and $R = 2$, so $Q \to R$ fails. Neither disjunct holds, so (B) is not always true.
4. (C) Augment $P \to R$ with Q to get $PQ \to RQ$. Augment $Q \to S$ with R to get $QR \to RS$. By transitivity, $PQ \to RQ$ and $RQ \to RS$ give $PQ \to RS$. (C) is always true. This is the union rule applied to the augmented dependencies.
5. (D) Augment $P \to R$ with Q to get $PQ \to RQ$. Since $R \subseteq \{R, Q\}$, decomposition gives $PQ \to R$. (D) is always true.
6. Hence the options that are always true are (C) and (D).

KEY IDEA

Check each implication with Armstrong's axioms (augmentation, transitivity, decomposition, union). If an implication cannot be derived, build a small relation that satisfies the premise and violates the conclusion.

WATCH OUT FOR

Thinking that a determinant made of two attributes can be split, so that $PQ \to R$ would imply $P \to R$ or $Q \to R$ (option B). Another slip is reading $P \to R$ together with $PQ \to R$ as saying something about Q alone (option A). Adding attributes to the left side of a dependency is always safe, but removing them is not.

Q.31 MSQ · 1 MARK

Database Management and Warehousing › Normal forms

In the context of relational database normalization, which of the following statements is/are true?

(A) It is always possible to obtain a dependency-preserving 3NF decomposition of a relation

(B) It is always possible to obtain a dependency-preserving 1NF decomposition of a relation

(C) It is not always possible to obtain a dependency-preserving BCNF decomposition of a relation

(D) It is not always possible to obtain a dependency-preserving 2NF decomposition of a relation

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

SOLUTION

1. (A) The 3NF synthesis algorithm starts from a minimal cover, makes one relation per functional dependency (merging those with the same left side), and adds a relation holding a candidate key if none is present. The result is lossless and dependency-preserving, so (A) is true.
2. (B) Every relation in the relational model has atomic attribute values, so it is already in 1NF. The trivial decomposition, the relation itself, is in 1NF and preserves every dependency. A dependency-preserving 1NF decomposition therefore always exists, and (B) is true.

3. (C) Take $R(C, S, Z)$ with $CS \to Z$ and $Z \to C$. The candidate keys are CS and SZ. The dependency $Z \to C$ violates BCNF because Z is not a superkey. The only lossless BCNF decomposition is $\{Z, C\}$ and $\{Z, S\}$, since the other two splits of a three-attribute relation, $\{C, Z\}, \{C, S\}$ and $\{C, S\}, \{S, Z\}$, share an attribute that determines neither side. That decomposition cannot enforce $CS \to Z$ locally, because C and S are in different relations. So a dependency-preserving BCNF decomposition does not always exist, and (C) is true.
4. (D) Every 3NF relation is also in 2NF. By (A), a dependency-preserving 3NF decomposition always exists, and it is also a dependency-preserving 2NF decomposition. So it is always possible, the statement that it is not always possible is false, and (D) is false.
5. Hence the true statements are A, B and C.

KEY IDEA
A lossless, dependency-preserving 3NF decomposition always exists, so the same holds for the weaker 2NF and 1NF. BCNF can lose dependencies, for example in $R(C, S, Z)$ with $CS \to Z$ and $Z \to C$.

WATCH OUT FOR
Assuming BCNF decomposition can always preserve dependencies, as 3NF synthesis does. Or reading (D) as true, forgetting that 2NF is weaker than 3NF, so the 3NF guarantee carries over.

Q.32 NAT · 1 MARK

Engineering Mathematics › Calculus

Consider the function $f : \mathbb{R} \to \mathbb{R}$ defined as follows:

$$f(x) = \begin{cases} c_1 e^x - c_2 \log_e\left(\frac{1}{x}\right), & \text{if } x > 0 \\ 3, & \text{otherwise} \end{cases}$$

where $c_1, c_2 \in \mathbb{R}$.

If f is continuous at $x = 0$, then $c_1 + c_2 =$ _____. (answer in integer)

Official answer: **3**

SOLUTION

1. Continuity at $x = 0$ requires $\lim_{x\to 0^-} f(x) = f(0) = \lim_{x\to 0^+} f(x)$. For $x \le 0$ the function is the constant 3 (the case "otherwise"), so $f(0) = 3$ and the left limit is 3.
2. For $x > 0$, $\log_e\left(\frac{1}{x}\right) = -\ln x$, so $f(x) = c_1 e^x + c_2 \ln x$.
3. As $x \to 0^+$, $c_1 e^x \to c_1$ (a finite number), while $\ln x \to -\infty$.
4. If $c_2 \neq 0$, then $c_2 \ln x \to \pm\infty$ and added to the finite term $c_1 e^x$ the sum still diverges, so the right-hand limit does not exist and f cannot be continuous. Hence $c_2 = 0$.
5. With $c_2 = 0$ the right-hand limit is $\lim_{x\to 0^+} c_1 e^x = c_1$. Equating it to $f(0) = 3$ gives $c_1 = 3$.
6. Therefore $c_1 + c_2 = 3 + 0 = 3$.

KEY IDEA
A logarithm blows up at 0^+, so its coefficient must be zero for the right-hand limit to exist. Then the surviving term must match the value 3 on the other side.

WATCH OUT FOR
Setting $\ln(1/x)$ to a finite value at $x \to 0^+$ and trying to solve $c_1 - c_2 \cdot (\text{something}) = 3$, instead of noticing that $\ln(1/x) \to +\infty$ forces $c_2 = 0$.

Q.33 NAT · 1 MARK

Data Structures and Algorithms › Trees & heaps

The height of a binary tree is the number of edges in the longest path from the root to a leaf in the tree. The

maximum possible height of a full binary tree with 23 nodes is _____. (answer in integer)

Official answer: **11**

SOLUTION

1. A full binary tree requires every node to have exactly 0 or 2 children (no node has exactly 1 child).
2. If there are i internal (2-child) nodes, the number of leaves is $i + 1$, so the total number of nodes is $n = i + (i + 1) = 2i + 1$.
3. For $n = 23$: $2i + 1 = 23 \Rightarrow i = 11$ internal nodes, and leaves $= 12$.
4. To maximise height, make the tree as skewed as possible: chain the internal nodes down one side of the tree, so each internal node (except the last one in the chain) has one leaf child and one child that continues the chain.
5. Place the 11 internal nodes at depths $0, 1, 2, \ldots, 10$. The internal nodes at depths 0 through 9 each spawn one leaf child (using 10 leaves) and one child continuing the chain to the next depth.
6. The internal node at depth 10 is the last internal node in the chain: it cannot continue the chain (that would require a 12th internal node, leaving too few nodes to keep $n = 23$), so both of its children are leaves, at depth 11.
7. Leaf count check: 10 leaves from the chain (depths 0–9) $+ 2$ leaves from the last internal node $= 12$ leaves, matching $i + 1 = 12$.
8. The longest root-to-leaf path passes through all 11 internal nodes (depths 0 to 10) and ends at a leaf at depth 11, i.e. it has 11 edges.
9. So the maximum possible height is $i = 11$, consistent with the general result $\text{height}_{\max} = \dfrac{n-1}{2} = \dfrac{23-1}{2} = 11$ for a full binary tree with n nodes.

KEY IDEA

In a full binary tree (every node has 0 or 2 children) with i internal nodes, leaves $= i + 1$, so $n = 2i + 1$; to maximise height, chain the internal nodes into a single skewed path, giving one leaf off each internal node except the last (which contributes two leaves), so $\text{height}_{\max} = i = (n-1)/2$.

WATCH OUT FOR

Confusing "full" binary tree (every node has 0 or 2 children; degenerate/skewed shapes are still allowed as long as no node has exactly one child) with "complete" or "perfect" binary tree, and instead computing the height of a perfect/complete tree with 23 nodes (which gives a much smaller height, around $\lceil \log_2(24) \rceil - 1 = 4$), or forgetting that the last internal node in the skewed chain must take two leaf children (not one), which would misalign the leaf count.

Q.34 NAT · 1 MARK

Programming in C › Functions & scope

Consider the following program in C:

```
#include <stdio.h>

void func(int i, int j) {
 if(i < j) {
  int i = 0;
  while (i < 10) {
   j += 2;
   i++;
  }
 }
 printf("%d", i);
}

int main() {
 int i = 9, j = 10;
 func(i, j);
 return 0;
}
```

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

Note: Assume that the program compiles and runs successfully.

Official answer: **9**

SOLUTION

1. main sets i = 9 , j = 10 and calls func(9, 10) . Inside func the parameters are i = 9 and j = 10 .
2. The condition i < j is $9 < 10$, which is true, so control enters the if block.
3. Inside the if block, int i = 0; declares a new variable i . It hides the parameter i , but only until the } that closes the if block.
4. The while loop uses this inner i . The body runs for inner i $= 0, 1, \ldots, 9$, which is 10 times, so j grows by $2 \times 10 = 20$ and becomes 30. The inner i ends at 10 and the loop stops. None of this changes the parameter i .
5. Look at where the braces fall in the printed program: the } that closes the if block comes before printf("%d", i); . So the printf is outside the if block, where the inner i no longer exists.
6. In that position the name i refers to the parameter i . Nothing ever assigned to the parameter, so it still holds 9.
7. The program prints 9 .

KEY IDEA

A variable declared inside a block { ... } exists only until that block's closing } . While it exists it hides any outer variable with the same name. Once the block closes, the name refers to the outer variable again, which in this program is the parameter i .

WATCH OUT FOR

Treating the printf as part of the if block and printing the inner i , which ends at 10. The if block closes before the printf , so the inner i is out of scope at that point and the parameter i = 9 is printed.

Q.35 NAT · 1 MARK

Operating Systems › Deadlock

Consider a system consisting of k instances of a resource R, being shared by 5 processes. Assume that each process requires a maximum of two instances of resource R and a process can request or release only one instance at a time. Further, a process can request the second instance of the resource only after acquiring the first instance.

The minimum value of k for the system to be deadlock-free is _____. (answer in integer)

Official answer: **6**

SOLUTION

1. Deadlock needs every process to hold part of its need and wait for more, with no free instance left.
2. Each process needs at most 2 instances, so the worst case is that every process holds exactly 1 instance and waits for its second.
3. In that worst case the number of instances held is $5 \times (2-1) = 5$. With $k = 5$ all instances are held and nobody can get a second one, so the system deadlocks.
4. With $k = 6$, even if all 5 processes hold 1 instance each, one instance is still free.
5. Give the free instance to one process. It now holds its maximum of 2, finishes and releases both, so 2 instances are free. The next process takes one, finishes, and releases 2, leaving 3 free, and so on until all 5 have finished. No process can wait forever.
6. General rule: deadlock-free if $k \geq \sum_i (m_i - 1) + 1 = 5(2-1) + 1 = 6$.
7. The minimum value of k is 6.

KEY IDEA

The worst case for deadlock is every process holding one less than its maximum need. With one more instance than that total, some process can always finish, so $k_{\min} = \sum(m_i - 1) + 1$.

WATCH OUT FOR

Answering 5 (one instance per process) or 10 (the sum of the maximum needs). Five instances can all be held with every process waiting for a second, which is a deadlock. Ten is enough but not the minimum.

Q.36 MCQ · 2 MARKS

Digital Electronics › Number systems

Consider the real valued variables X, Y and Z represented using the IEEE 754 single-precision floating-point format. The binary representations of X and Y in hexadecimal notation are as follows:

X: 35C00000 Y: 34A00000

Let Z = X + Y.

Which one of the following is the binary representation of Z, in hexadecimal notation?

(A) 35C80000

(B) 35CC0000

(C) 35E80000

(D) 35EC0000

Official answer: **C**

SOLUTION

1. IEEE 754 single precision: $(-1)^s \times 1.f \times 2^{e-127}$, with 1 sign bit, 8 exponent bits and 23 fraction bits.
2. $X =$ 35C00000 $= 0011\,0101\,1100\,0000\ldots$. Sign $s = 0$, exponent bits $01101011 = 107$, fraction bits $100\,0\ldots$, so $1.f = 1.5$. Hence $X = 1.5 \times 2^{107-127} = 1.5 \times 2^{-20}$.
3. $Y =$ 34A00000 $= 0011\,0100\,1010\,0000\ldots$. Sign $s = 0$, exponent bits $01101001 = 105$, fraction bits $010\,0\ldots$, so $1.f = 1.25$. Hence $Y = 1.25 \times 2^{105-127} = 1.25 \times 2^{-22}$.
4. Align to the larger exponent: $Y = 1.25 \times 2^{-22} = 0.3125 \times 2^{-20}$.
5. Add the significands: $1.5 + 0.3125 = 1.8125$, so $Z = 1.8125 \times 2^{-20}$. Since $1 \leq 1.8125 < 2$ it is already normalised, the exponent field stays 107, and the fraction $0.8125 = 0.1101_2$ needs only 4 bits, so no rounding is needed.
6. Assemble: sign 0, exponent 01101011, fraction $1101\,0000\ldots$. The bit string is $0\,01101011\,1101000\ldots = 0011\,0101\,1110\,1000\,0000\ldots$, which is 35E80000 , option (C).

KEY IDEA

Decode both numbers as $1.f \times 2^{e-127}$, shift the smaller one to the larger exponent, add the significands, check that the result is still normalised, then re-encode the fraction and exponent in hexadecimal.

WATCH OUT FOR

Misreading the exponent field by one bit, because the exponent straddles two hex digits (the first bit of the third digit belongs to the exponent). A second slip is shifting the smaller operand the wrong way, or by 2 places instead of $107 - 105 = 2$ exponent steps correctly applied as a divide by 4.

Q.37 MCQ · 2 MARKS

Digital Electronics › Sequential

Q.37 Consider a 2-bit saturating up/down counter that performs the saturating up count when the input P is 0, and the saturating down count when P is 1. The Next State table of the counter is as shown. The counter is built as a synchronous sequential circuit using D flip-flops.

Input	Current State		Next State	
P	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	0	1
1	1	1	1	0

Which one of the following options corresponds to the expressions for the inputs of the D flip-flops, D_1 and D_0?

(A) $D_1 = P\,Q_1 + \bar{P}Q_0 + Q_1Q_0$ $\quad D_0 = P\,Q_0 + \bar{P}\,Q_1 + Q_1\,\overline{Q_0}$

(B) $D_1 = \bar{P}\,Q_1 + \bar{P}Q_0 + Q_1Q_0$ $\quad D_0 = \bar{P}\,\overline{Q_0} + \bar{P}\,Q_1 + Q_1\,\overline{Q_0}$

(C) $D_1 = \bar{P}\,\overline{Q_1} + \bar{P}\,Q_0 + Q_1\,Q_0$ $\quad D_0 = \bar{P}\,Q_0 + \bar{P}\,Q_1 + Q_1\,\overline{Q_0}$

(D) $D_1 = P\,\overline{Q_1} + \bar{P}\,Q_0 + Q_1\,Q_0$ $\quad D_0 = P\,\overline{Q_0} + \bar{P}\,Q_1 + Q_1\,\overline{Q_0}$

Official answer: **B**

SOLUTION

1. For a D flip-flop the input equals the next state, so $D_1 = Q_1^+$ and $D_0 = Q_0^+$.
2. Read Q_1^+ from the table. For $P = 0$, with $Q_1Q_0 = 00, 01, 10, 11$, it is $0, 1, 1, 1$, which is $Q_1 + Q_0$. For $P = 1$ it is $0, 0, 0, 1$, which is Q_1Q_0.
3. Combine: $D_1 = \overline{P}\,(Q_1 + Q_0) + P\,Q_1Q_0$. The term Q_1Q_0 is already true wherever PQ_1Q_0 is. At $P = 0$ it is contained in $Q_1 + Q_0$, so adding it changes nothing. Replacing PQ_1Q_0 by Q_1Q_0 therefore gives

$D_1 = \overline{P}Q_1 + \overline{P}Q_0 + Q_1Q_0$.

4. Read Q_0^+. For $P = 0$ it is $1, 0, 1, 1$ for $Q_1Q_0 = 00, 01, 10, 11$, which is $\overline{Q_0} + Q_1$. For $P = 1$ it is $0, 0, 1, 0$, which is $Q_1\overline{Q_0}$.
5. Combine: $D_0 = \overline{P}\,(\overline{Q_0} + Q_1) + P\,Q_1\overline{Q_0}$. At $P = 0$ the term $Q_1\overline{Q_0}$ is contained in $\overline{P}Q_1$, so replacing $PQ_1\overline{Q_0}$ by $Q_1\overline{Q_0}$ changes nothing. Hence $D_0 = \overline{P}\,\overline{Q_0} + \overline{P}Q_1 + Q_1\overline{Q_0}$.
6. Check against the table. At $P = 1$, $Q_1Q_0 = 11$: the $\overline{P}$ terms vanish and $Q_1\overline{Q_0} = 0$, so $D_0 = 0$, as in the table. At $P = 1$, $Q_1Q_0 = 10$: the $\overline{P}$ terms vanish and $Q_1Q_0 = 0$, so $D_1 = 0$, as in the table.
7. The pair matches option (B). The others fail on a single row. (A) gives $D_1 = 1$ at $P = 1$, $Q_1Q_0 = 10$, but the table has 0. (C) gives $D_1 = 1$ at $P = 0$, $Q_1Q_0 = 00$, but the table has 0. (D) gives $D_1 = 1$ at $P = 1$, $Q_1Q_0 = 00$, but the table has 0.

KEY IDEA
For a D flip-flop, D equals the next state. Split the next-state column by P, write each half as a small expression in Q_1, Q_0, and add a redundant consensus term to match the options.

WATCH OUT FOR
Using P in place of $\overline{P}$ for the up-count half, which swaps the up and down behaviour. Another slip is misreading the Q_0^+ column, which toggles on up count and is not a simple copy of Q_0.

Q.38 MCQ · 2 MARKS

Computer Organization and Architecture › Memory & cache

The size of the physical address space of a processor is 2^{32} bytes. The capacity of a cache memory unit is 2^{23} bytes. The cache block size is 128 bytes. The cache memory unit can be built as a direct mapped cache or as a K-way set-associative cache, where $K = 2^L$ and $L \in \{1, 2, 3\}$. Let the length of the TAG field be M bits for the direct mapped cache, and N bits for the set-associative cache.

Which one of the following options is true?

(A) $N = M + L$
(B) $N = M - L$
(C) $N = M + K$
(D) $N = M - K$

Official answer: **A**

SOLUTION

1. Block offset bits: block size = 128 = 2^7 bytes, so offset = 7 bits. This is the same for the direct-mapped and set-associative organizations, since block size is unchanged.
2. Total number of blocks (lines) the cache can hold = $\dfrac{2^{23}}{2^7} = 2^{16}$.
3. Direct-mapped cache: each set holds exactly 1 block, so number of sets = number of lines = 2^{16}, giving index length = 16 bits. Then $M = 32 - 7 - 16 = 9$ bits.
4. K-way set-associative cache: each set holds $K = 2^L$ blocks, so number of sets $= \dfrac{2^{16}}{2^L} = 2^{16-L}$, giving index length = $(16 - L)$ bits.
5. Tag length: $N = 32 - 7 - (16 - L) = 9 + L$.
6. Since $M = 9$, this gives $N = M + L$, so option (A) is correct.

KEY IDEA

Tag = Total address bits − Block offset bits − Index (set) bits. Block offset is fixed (7 bits) regardless of mapping scheme; only the number of sets (hence index bits) changes between direct-mapped and K-way set-associative caches of the same total capacity and block size.

WATCH OUT FOR

Forgetting that increasing associativity (K-way) shrinks the number of sets by a factor of K, so it REDUCES the index bits and thus INCREASES the tag bits: students often think higher associativity increases index bits, leading them to pick B (N = M − L) instead of A.

Q.39 MCQ · 2 MARKS

Data Structures and Algorithms › Linear structures

Consider the following code snippet in C language that computes the number of nodes in a non-empty singly linked list pointed to by the pointer variable head.

```
struct node{
 int elt;
 struct node *next;
};

int getListSize (struct node *head)
{
 if( E1 ) return 1;
 return E2;
}
```

Which one of the following options gives the correct replacements for the expressions E1 and E2?

(A) E1: head == NULL
E2: 1 + getListSize(head)

(B) E1: head→next == NULL
E2: 1 + getListSize(head→next)

(C) E1: head == NULL
E2: 1 + getListSize(head→next)

(D) E1: head→next == NULL
E2: 1 + getListSize(head)

Official answer: **B**

SOLUTION

1. The function is documented to be called only on a non-empty list, so `head` is never `NULL` at the point of the call; the base case must detect the LAST node reached during recursion, not a NULL head.
2. The last node in the list satisfies `head→next == NULL`, and that single node contributes exactly 1 to the total count. So the base case should be `E1: head→next == NULL`, returning 1 there.
3. For any non-last node, the recursive step must add 1 for the current node and advance the pointer to the next node so the recursion makes progress: `E2: 1 + getListSize(head→next)`.
4. Rule out (A): E1 = head==NULL, E2 = 1+getListSize(head). The recursive call passes `head` unchanged, so the pointer never advances; this recurses forever on the same node (stack overflow), never terminating.
5. Rule out (D): E1 = head→next==NULL, E2 = 1+getListSize(head). Same flaw as (A): the recursive call never advances the pointer, causing infinite recursion.
6. Rule out (C): E1 = head==NULL, E2 = 1+getListSize(head→next). Trace a 3-node list A→B→C→NULL: getListSize(A) = 1+getListSize(B) = 1+(1+getListSize(C)) = 1+(1+(1+getListSize(C→next=NULL)));

getListSize(NULL) hits head==NULL and returns 1. Total = 1+1+1+1 = 4, but the list has only 3 nodes; this makes one extra call past the last node into NULL, overcounting by 1.

7. Check (B): E1 = head→next==NULL, E2 = 1+getListSize(head→next). Trace the same 3-node list: getListSize(A): A→next=B≠NULL, so 1+getListSize(B); getListSize(B): B→next=C≠NULL, so 1+getListSize(C); getListSize(C): C→next==NULL (last node), returns 1 directly. Total = 1+1+1 = 3, matching the true node count.
8. Sanity check the single-node case for (B): head→next==NULL holds immediately, so the base case fires at once and returns 1: correct.
9. Therefore (B) is the only option that both terminates and computes the correct count.

KEY IDEA

A recursive size/length function over a linked list needs its base case and recursive step to move in lockstep: the base case must fire exactly once per traversal (at the true last node, detected by `head→next == NULL`, not by overshooting to `head == NULL`), and the recursive call must always advance the pointer it tests (`getListSize(head→next)`, never the unchanged `head`) or it never terminates.

WATCH OUT FOR

Picking option (C) (E1: head==NULL, E2: 1+getListSize(head→next)). This looks like the "natural" NULL-terminated recursion pattern but it makes one extra call past the last node (into NULL), overcounting every list by exactly 1. Equally common is picking (A) or (D) where E2 recurses on the unchanged `head` pointer, causing infinite recursion / a crash rather than a wrong count.

Q.40 MCQ · 2 MARKS

Data Structures and Algorithms › Trees & heaps

Let P be the set of all integers from 1 to 15. Consider any order of insertion of the elements of P into a binary search tree that creates a complete binary tree.

Which one of the following elements can NEVER be the third element that is inserted?

(A) 4

(B) 2

(C) 10

(D) 5

Official answer: **D**

Q.41 MCQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

Let $G(V, E)$ be an undirected, edge-weighted graph with integer weights. The weight of a path is the sum of the weights of the edges in that path. The length of a path is the number of edges in that path.

Let $s \in V$ be a vertex in G. For every $u \in V$ and for every $k \geq 0$, let $d_k(u)$ denote the weight of a shortest path (in terms of weight) from s to u of length at most k. If there is no path from s to u of length at most k, then $d_k(u) = \infty$.

Consider the statements:

S1: For every $k \geq 0$ and $u \in V$, $d_{k+1}(u) \leq d_k(u)$.

S2: For every $(u, v) \in E$, if (u, v) is part of a shortest path (in terms of weight) from s to v, then for every $k \geq 0$, $d_k(u) \leq d_k(v)$.

Which one of the following options is correct?

(A) Only S1 is true

(B) Only S2 is true

(C) Both S1 and S2 are true

(D) Neither S1 nor S2 is true

Official answer: **A**

SOLUTION

1. S1: A path counted for d_k(u) is also a path counted for d_{k+1}(u), so the feasible path set only grows as k increases, and the minimum weight over a larger set can never exceed the minimum over a subset; hence d_{k+1}(u) <= d_k(u) for every k and u, unconditionally. S1 is TRUE.
2. S2: Build the counterexample graph: vertices s, x, u, v with undirected edges s-x (weight 0), x-u (weight 0), u-v (weight 0), s-v (weight 1).
3. The true shortest path from s to v is s-x-u-v with weight 0 (versus the direct edge s-v of weight 1), and it uses edge (u,v), so S2's hypothesis holds for this edge.
4. At k=1: d_1(v)=1 (only the direct edge s-v has length <=1), while d_1(u)=infinity (no edge s-u exists, so u is unreachable within 1 hop).
5. Checking S2's required inequality: d_1(u) <= d_1(v) becomes infinity <= 1, which is false, so S2 fails despite its hypothesis holding. S2 is FALSE.
6. Since S1 is always true and S2 is false, the correct choice is Only S1 is true, option A.

KEY IDEA

d_k(u) is monotone non-increasing in k because the set of length-<=k paths is a subset of the length-<=(k+1) paths, so the minimum can only get smaller or stay the same (S1). The claim that d_k(u) <= d_k(v) whenever (u,v) lies on a shortest path to v is false because reaching u may require strictly more hops than reaching v via a different, cheaper route (S2).

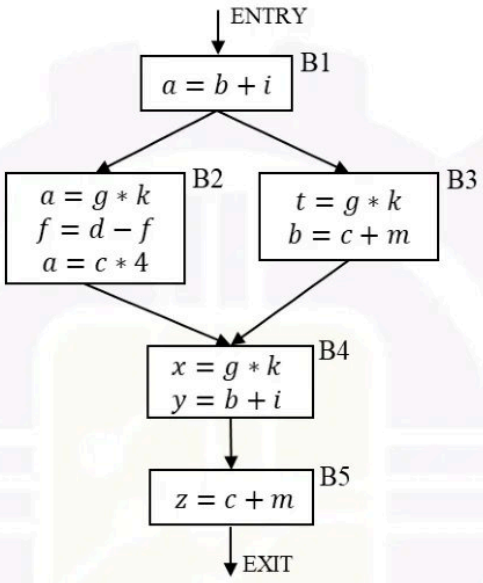
WATCH OUT FOR

Assuming S2 must be true by analogy with the Bellman-Ford relaxation step d_{k+1}(v) <= d_k(u) + w(u,v); that inequality involves k and k+1 with an added edge weight, not d_k(u) <= d_k(v) at the same k with no weight term, so the analogy does not transfer and a hop-count counterexample breaks it.

Q.42 MCQ · 2 MARKS

Compiler Design › Optimisation

Q.42 Consider the control flow graph shown in the figure.



Which one of the following options correctly lists the set of redundant expressions (common subexpressions) in the basic blocks B4 and B5?

Note: All the variables are integers.

(A) B4: $\{ b + i \}$
B5: $\{ c + m \}$

(B) B4: $\{ g * k \}$
B5: $\{ c + m \}$

(C) B4: $\{ g * k,\ b + i \}$
B5: $\{ \}$

(D) B4: $\{ g * k \}$
B5: $\{ \}$

Official answer: **D**

SOLUTION

1. An expression is redundant at a point if it is available there: it has been computed on every path from ENTRY, and none of its operands has been redefined since.
2. Available-expression equations: $\mathrm{IN}(B) = \bigcap_{P \in \mathrm{pred}(B)} \mathrm{OUT}(P)$ and $\mathrm{OUT}(B) = \mathrm{gen}(B) \cup (\mathrm{IN}(B) - \mathrm{kill}(B))$, with $\mathrm{IN}(B1) = \{\}$.

3. B1 has `a = b + i`. It generates $b + i$. The variable a is not an operand of any expression in play, so nothing is killed. $\text{OUT}(B1) = \{b + i\}$.
4. B2 has three statements. `a = g * k` generates $g * k$. `f = d - f` generates $d - f$ and then kills it, because f is redefined by the same statement. `a = c * 4` generates $c * 4$. No statement redefines b, i, g or k, so $b + i$ survives. $\text{OUT}(B2) = \{b + i,\ g * k,\ c * 4\}$.
5. B3 has `t = g * k`, which generates $g * k$. Then `b = c + m` generates $c + m$ and kills $b + i$, because b is redefined. $\text{OUT}(B3) = \{g * k,\ c + m\}$.
6. $\text{IN}(B4) = \text{OUT}(B2) \cap \text{OUT}(B3) = \{g * k\}$. So `g * k` in B4 is redundant. `b + i` in B4 is not redundant, because on the path through B3 it was killed.
7. B4 evaluates `g * k` (already available, nothing new generated) and `b + i`, which becomes available. Neither x nor y is an operand of any expression in play. $\text{OUT}(B4) = \{g * k,\ b + i\}$, so $\text{IN}(B5) = \{g * k,\ b + i\}$.
8. B5 computes `c + m`, which is not in $\text{IN}(B5)$. The path through B2 never computes $c + m$. So B5 has no redundant expression.
9. Result: B4: $\{g * k\}$ and B5: $\{\ \}$. This is option (D).

KEY IDEA

A common subexpression is redundant only if it is available on every path into the block. Available expressions use a forward "must" analysis, so IN is the intersection of the predecessors' OUT sets. An expression is killed when any of its operands is reassigned.

WATCH OUT FOR

Taking the union of the predecessors' OUT sets, instead of the intersection, makes $b + i$ and $c + m$ look redundant. They are not: $b + i$ is killed on the B3 path by `b = c + m`, and $c + m$ is computed only on the B3 path, never on the B2 path.

Q.43 MCQ · 2 MARKS

Database Management and Warehousing › Relational algebra

Consider a relational database schema with two relations $R(P, Q)$ and $S(X, Y)$.

Let $E = \{\langle u \rangle \mid \exists v\ \exists w\ \langle u, v \rangle \in R \wedge \langle v, w \rangle \in S\}$ be a tuple relational calculus expression.

Which one of the following relational algebraic expressions is equivalent to E ?

(A) $\Pi_P(R \bowtie_{R.P=S.X} S)$

(B) $\Pi_P(S \bowtie_{S.X=R.Q} R)$

(C) $\Pi_P(R \bowtie_{R.P=S.Y} S)$

(D) $\Pi_P(S \bowtie_{S.Y=R.Q} R)$

Official answer: **B**

SOLUTION

1. Read the calculus expression. The answer tuple is $\langle u \rangle$, so the output has one column. Because $\langle u, v \rangle \in R$ and R has attributes (P, Q), u is the value of P and v is the value of Q.
2. The condition $\langle v, w \rangle \in S$ with $S(X, Y)$ means v is the value of X and w is the value of Y.
3. The variable v appears in both tuples. A tuple of R and a tuple of S must therefore agree with $R.Q = S.X$.
4. The variable w is only existentially quantified. It is not in the output, so Y is not used and simply drops out. The same holds for v, which is quantified and not output.
5. So $E = \Pi_P\big(R \bowtie_{R.Q=S.X} S\big)$. A theta join is symmetric in its operands, so $E = \Pi_P\big(S \bowtie_{S.X=R.Q} R\big)$.
6. Check the options. (A) joins on $R.P = S.X$ and (C) on $R.P = S.Y$. Both use P, the wrong attribute of R. (D) joins $S.Y = R.Q$, which uses Y, the wrong attribute of S. Only (B) joins $S.X = R.Q$ and projects P.

KEY IDEA
A variable shared by two tuples in a calculus expression becomes an equality join condition. Existentially quantified variables that are not output are projected away.

WATCH OUT FOR
Joining on the first attribute of each relation ($R.P$ with $S.X$), or on the second attribute of S (Y). The shared variable v is the second position of the R tuple (attribute Q) and the first position of the S tuple (attribute X).

Q.44 MCQ · 2 MARKS

Computer Networks › TCP

A TCP sender successfully establishes a connection with a TCP receiver and starts the transmission of segments. The TCP congestion control mechanism's slow-start threshold is set to 10000 segments. Assume that the round-trip time is fixed at 1 millisecond. Assume that the sender always has data to send, the segments are numbered from 1, and no segment is lost. Let t denote the time (in milliseconds) at which the transmission of segment number 2000 starts.

Which one of the following options is correct?

(A) 9 \le t < 10

(B) 10 \le t < 11

(C) 11 \le t < 12

(D) 12 \le t < 13

Official answer: **B**

SOLUTION

1. The threshold is 10000 segments, far above 2000, so the sender stays in slow start. With no loss, the congestion window starts at 1 MSS and doubles every round-trip time (RTT = 1 ms).
2. Take round k (for $k = 1, 2, \ldots$) to begin at $t = (k - 1)$ ms. It sends 2^{k-1} segments.
3. Segments sent in rounds 1 to k in total: $1 + 2 + 4 + \cdots + 2^{k-1} = 2^k - 1$.
4.

Round	Starts at (ms)	Segments sent
1	0	1 (segment 1)
2	1	2 (segments 2 to 3)
10	9	512 (segments 512 to 1023)
11	10	1024 (segments 1024 to 2047)

5. After round 10, $2^{10} - 1 = 1023$ segments have been sent. The window in round 11 is $1024 < 10000$, so it is still slow start.
6. Segment 2000 satisfies $1024 \le 2000 \le 2047$, so it is sent in round 11, which begins at $t = 10$ ms.
7. The segments of round 11 go out as ACKs return during that one RTT, before round 12 begins at $t = 11$ ms. Hence $10 \le t < 11$, which is option (B).

KEY IDEA
In slow start with no loss the window doubles each RTT, so round k sends 2^(k-1) segments and the first k rounds send 2^k - 1 in total. Find the round containing segment 2000, then read its start time.

WATCH OUT FOR
Being off by one round. Segment 1 goes out in round 1 at t = 0, so round k starts at (k-1) ms, not k ms. Using 2^k instead of 2^k - 1 for the cumulative count also shifts the answer to the wrong interval.

Q.45 MCQ · 2 MARKS

Computer Networks › Performance

Consider the implementation of sliding window protocol over a lossless link, with a window size of W frames, where each frame is of size 1000 bits (including header). The bandwidth of the link is 100 kbps (1k = 10^3) and the one-way propagation delay is 100 milliseconds. Assume that processing times at the sender

and receiver are zero and the transmission time of acknowledgements is also zero. Which one of the following options gives the minimum size of W (in number of frames) required to achieve 100% link utilization?

(A) 10

(B) 21

(C) 20

(D) 11

Official answer: **B**

SOLUTION

1. Frame transmission time: $T_t = \dfrac{1000 \text{ bits}}{100 \times 10^3 \text{ bps}} = 10 \text{ ms}$.
2. One-way propagation delay is $T_p = 100 \text{ ms}$, so the round trip is $2T_p = 200 \text{ ms}$. Acknowledgements take zero time to transmit and nothing is spent on processing.
3. The sender starts the first frame at time 0. Its last bit leaves at 10 ms, reaches the receiver at 110 ms, and the acknowledgement is back at 210 ms. One full cycle is $T_t + 2T_p = 210 \text{ ms}$.
4. For 100% utilization the sender must keep transmitting for the whole cycle, so the window must cover the cycle: $W \times T_t \geq T_t + 2T_p$.
5. $W \geq \dfrac{10 + 200}{10} = 21$.
6. Check: with $W = 20$ the sender is busy for 200 ms and then idles for 10 ms until the first acknowledgement arrives at 210 ms, so utilization is $\dfrac{20}{21} < 100\%$. With $W = 21$ the first acknowledgement arrives just as the 21st frame finishes, so the sender never idles.
7. Minimum $W = 21$ frames, option (B).

KEY IDEA

For 100% utilization the window must cover one full cycle: $W = \lceil (T_t + 2T_p)/T_t \rceil = 1 + 2T_p/T_t$. With $T_t = 10$ ms and $2T_p = 200$ ms this gives 21.

WATCH OUT FOR

Using $W = 2T_p/T_t = 20$ and forgetting the one frame time T_t needed to send the first frame, which gives option (C). The cycle is $T_t + 2T_p$, not $2T_p$ alone.

Q.46 MSQ · 2 MARKS

Engineering Mathematics › Calculus

Let $f : \mathbb{R} \to \mathbb{R}$ be defined as follows:

$$f(x) = \left(\frac{|x|}{2} - x\right)\left(x - \frac{|x|}{2}\right)$$

Which of the following statements is/are true?

(A) f has a local maximum

(B) f has a local minimum

(C) f' is continuous over $\mathbb{R}$

(D) f' is not differentiable over $\mathbb{R}$

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

SOLUTION

1. The two factors are negatives of each other: $\frac{|x|}{2} - x = -\left(x - \frac{|x|}{2}\right)$. So $f(x) = -\left(x - \frac{|x|}{2}\right)^2 \leq 0$ for all x.
2. For $x \geq 0$: $|x| = x$, so $x - \frac{x}{2} = \frac{x}{2}$ and $f(x) = -\frac{x^2}{4}$.

3. For $x < 0$: $|x| = -x$, so $x - \frac{|x|}{2} = x + \frac{x}{2} = \frac{3x}{2}$ and $f(x) = -\frac{9x^2}{4}$.
4. $f(0) = 0$ and $f(x) < 0$ for every $x \neq 0$, so $x = 0$ is a local (indeed global) maximum. (A) is true.
5. For $x > 0$, $f'(x) = -\frac{x}{2} \neq 0$, and for $x < 0$, $f'(x) = -\frac{9x}{2} \neq 0$. So f has no stationary point other than $x = 0$, and no local extremum can occur at any $x \neq 0$. At $x = 0$ the value 0 is a maximum, not a minimum, since $f < 0$ nearby. So f has no local minimum. (B) is false.
6. At 0: the difference quotient is $\frac{f(h)-f(0)}{h} = -\frac{h}{4}$ for $h > 0$ and $-\frac{9h}{4}$ for $h < 0$, and both tend to 0. So $f'(0) = 0$. Also $f'(x) = -\frac{x}{2} \to 0$ as $x \to 0^+$ and $f'(x) = -\frac{9x}{2} \to 0$ as $x \to 0^-$, matching $f'(0) = 0$. Away from 0, f' is a polynomial, hence continuous. So f' is continuous over $\mathbb{R}$. (C) is true.
7. Differentiability of f' at 0: $\frac{f'(h)-f'(0)}{h} = -\frac{1}{2}$ for $h > 0$ and $-\frac{9}{2}$ for $h < 0$. The one-sided limits differ, so f' is not differentiable at 0. (D) is true.
8. Hence the true statements are (A), (C) and (D).

KEY IDEA

Remove the absolute value by splitting at $x = 0$. The function is a downward parabola piece on each side, with different curvature on the two sides, so f' is continuous at 0 but has a kink there.

WATCH OUT FOR

Assuming a function with a corner-free graph is smooth. Here $f'(0) = 0$ from both sides, but the slopes of f' are $-\frac{1}{2}$ and $-\frac{9}{2}$, so $f''(0)$ does not exist. Another slip is treating the product as a single square with the same coefficient on both sides.

Q.47 MSQ · 2 MARKS

Discrete Mathematics › Graph theory

Let $G(V, E)$ be a simple, undirected graph. A vertex cover of G is a subset $V' \subseteq V$ such that for every $(u, v) \in E$, $u \in V'$ or $v \in V'$. Let the size of the smallest vertex cover in G be k. Let S be any vertex cover of size k.

For a vertex $v \in V$, which of the following constraints will always ensure that $v \in S$?

(A) The degree of v is at least $k + 1$

(B) The vertex v is on a path of length $k + 1$

(C) The vertex v is on a cycle of length $k + 1$

(D) The vertex v is a part of a clique of size k

Official answer: **A**

SOLUTION

1. Let S be a vertex cover of the smallest size k. If a vertex $v \notin S$, every edge (v, u) must be covered by u, so all neighbours of v lie in S.
2. (A) Suppose $\deg(v) \geq k + 1$ and $v \notin S$. Then S contains at least $k + 1$ distinct neighbours of v, so $|S| \geq k + 1 > k$, a contradiction. Hence $v \in S$ for every minimum cover S. A always ensures $v \in S$.
3. (B) Take the path $a - b - c$. The smallest cover is $\{b\}$, so $k = 1$, and the path has length $2 = k + 1$. The end vertex a is on this path but $a \notin S$. B fails.
4. (C) Take the triangle K_3, a cycle of length 3. Its smallest cover has size $k = 2$, so the cycle has length $k + 1 = 3$. The cover $\{a, b\}$ has size 2 and omits c, which lies on the cycle. C fails.
5. (D) Take K_3 with vertices a, b, c. Again $k = 2$. The vertex c is part of the clique $\{a, c\}$ of size $2 = k$. The minimum cover $\{a, b\}$ omits c. D fails.
6. Only (A) is guaranteed, so the answer is A.

KEY IDEA
A vertex left out of a cover forces all its neighbours into it, so a vertex with more than k neighbours cannot be left out of a cover of size k.

WATCH OUT FOR
Assuming that lying on a long path or cycle, or inside a clique, forces membership. In a cycle or clique, a cover can skip any one vertex, so the structure alone does not force v into S.

Q.48 MSQ · 2 MARKS

Digital Electronics › Boolean logic

Consider a Boolean function F with the following minterm expression: $F(P, Q, R, S) = \sum m(1, 2, 3, 4, 5, 7, 10, 12, 13, 14)$. Which of the following options is/are the minimal sum-of-products expression(s) of F?

(A) $\overline{P}S + Q\overline{R} + \overline{P}\,\overline{Q}R + \overline{Q}R\overline{S}$

(B) $\overline{P}S + Q\overline{R} + \overline{P}\,\overline{Q}R + PR\overline{S}$

(C) $\overline{P}S + Q\overline{R} + PQ\overline{S} + PR\overline{S}$

(D) $\overline{P}S + Q\overline{R} + PQ\overline{S} + \overline{Q}R\overline{S}$

Official answer: **B, D**

SOLUTION

1. Write the minterms in binary ($PQRS$): $m_1 = 0001$, $m_2 = 0010$, $m_3 = 0011$, $m_4 = 0100$, $m_5 = 0101$, $m_7 = 0111$, $m_{10} = 1010$, $m_{12} = 1100$, $m_{13} = 1101$, $m_{14} = 1110$. The remaining cells $m_0, m_6, m_8, m_9, m_{11}, m_{15}$ are 0.
2. Prime implicants from the K-map: $\overline{P}S = m(1, 3, 5, 7)$; $Q\overline{R} = m(4, 5, 12, 13)$; $\overline{P}\,\overline{Q}R = m(2, 3)$; $\overline{Q}R\overline{S} = m(2, 10)$; $PR\overline{S} = m(10, 14)$; $PQ\overline{S} = m(12, 14)$. Each pair is a legal group because the neighbouring cells $m_6, m_{15}, m_{11}, m_0, m_8, m_9$ are all 0, so none of these can be enlarged.
3. Essential prime implicants: m_1 and m_7 are covered only by $\overline{P}S$, and m_{13} only by $Q\overline{R}$. So both terms must appear in every cover. Together they cover $m(1, 3, 4, 5, 7, 12, 13)$.
4. Minterms still uncovered: m_2, m_{10}, m_{14}. Here $m_2 \in \{\overline{P}\,\overline{Q}R,\ \overline{Q}R\overline{S}\}$, $m_{10} \in \{\overline{Q}R\overline{S},\ PR\overline{S}\}$, $m_{14} \in \{PR\overline{S},\ PQ\overline{S}\}$. No single implicant covers all three, so at least two more terms are needed. The minimum is therefore 4 terms.
5. Pairs that cover $\{2, 10, 14\}$: $\{\overline{Q}R\overline{S},\ PR\overline{S}\}$ (not offered), $\{\overline{Q}R\overline{S},\ PQ\overline{S}\}$ (option D), $\{\overline{P}\,\overline{Q}R,\ PR\overline{S}\}$ (option B). The pair $\{\overline{P}\,\overline{Q}R,\ PQ\overline{S}\}$ misses m_{10}. Each valid cover has 4 terms and 10 literals.
6. Option A uses $\overline{P}\,\overline{Q}R + \overline{Q}R\overline{S}$, which covers only m_2, m_3, m_{10} of the three missing cells, so m_{14} is uncovered and A is not equal to F. Option C uses $PQ\overline{S} + PR\overline{S}$, which leaves m_2 uncovered, so C is not equal to F.
7. B and D both equal F and have the minimum cost of 4 terms, so both are minimal sum-of-products expressions.

KEY IDEA
Take the essential prime implicants first, then find the cheapest set of remaining prime implicants that covers the leftover minterms. Several covers can tie for minimum cost, so more than one option can be correct.

WATCH OUT FOR
Stopping at the first minimal cover found, or choosing an option that looks similar but leaves a minterm uncovered (A misses m_{14}, C misses m_2). Always check that every minterm is covered.

Q.49 MSQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

Let $G(V, E)$ be a simple, undirected, edge-weighted graph with unique edge weights.

Which of the following statements about the minimum spanning trees (MST) of G is/are true?

(A) In every cycle C of G, the edge with the largest weight in C is not in any MST

(B) In every cycle C of G, the edge with the smallest weight in C is in every MST

(C) For every vertex $v \in V$, the edge with the largest weight incident on v is not in any MST

(D) For every vertex $v \in V$, the edge with the smallest weight incident on v is in every MST

Official answer: **A, D**

SOLUTION

1. Since all edge weights in G are distinct, G has a unique MST, and both the cycle property and the cut property hold in their strict (no-tie) form.
2. **Cycle property (Statement A):** In any cycle C, let e be the strictly maximum-weight edge in C. If e were in some spanning tree T, removing e splits T into two components; since e lies on a cycle, some other edge f of C (with $f \neq e$) crosses back between these two components. Replacing e with f gives another spanning tree that is no heavier, and strictly lighter because weights are unique, so T was not minimum. Hence the max-weight edge of every cycle is excluded from every MST: **Statement A is TRUE.**
3. **Testing Statement B with a counterexample:** Take vertices v_1, v_2, v_3, v_4 with edges $v_1v_4 = 1$, $v_2v_4 = 1$, $v_1v_2 = 3$, $v_2v_3 = 4$, $v_1v_3 = 5$. The cycle v_1-v_2-v_3-v_1 has weights $\{3, 4, 5\}$, so its minimum-weight edge is v_1v_2 (weight 3).
4. Running Kruskal's algorithm (edges in increasing order $1, 1, 3, 4, 5$): add v_1v_4; add v_2v_4 (now $\{v_1, v_2, v_4\}$ connected); skip v_1v_2 since it would close a cycle; add v_2v_3 to connect v_3. The unique MST is $\{v_1v_4,\ v_2v_4,\ v_2v_3\}$, which does **not** contain v_1v_2, the minimum edge of that cycle. So a cycle's minimum edge need not be in the MST: **Statement B is FALSE.**
5. **Testing Statement C:** Take any vertex v of degree 1 (a leaf), for instance in a simple path graph. Its single incident edge is simultaneously the largest- and smallest-weight edge incident on it. Since removing this edge would disconnect v from the rest of the graph, this edge must belong to every spanning tree, including the MST, so the 'largest incident edge' at v IS in the MST, contradicting the claim that it is in no MST: **Statement C is FALSE.**
6. **Cut property (used for Statement D):** For any cut $(S, V \setminus S)$, if e is the strictly minimum-weight edge crossing the cut, then e lies in every MST: any spanning tree lacking e can be improved by swapping in e for some heavier crossing edge that its addition would make redundant.
7. **Statement D:** Apply the cut property to the trivial cut $(\{v\}, V \setminus \{v\})$ for an arbitrary vertex v. The edges crossing this cut are exactly the edges incident on v. Therefore the minimum-weight edge incident on v is the unique minimum-weight crossing edge of this cut, and by the cut property it must lie in every MST. Since v was arbitrary, this holds for every vertex: **Statement D is TRUE.** (This is also the principle behind Borůvka's algorithm, where each component picks its own cheapest outgoing edge.)
8. Hence the true statements are **A and D**.

KEY IDEA

Because all edge weights are unique, G has a unique MST, and two classical results pin down every option: the **cycle property** (in any cycle, the strictly max-weight edge is never in an MST) and the **cut property** (across any cut, the strictly min-weight crossing edge is in every MST). Applying the cut property to the trivial cut $(\{v\}, V \setminus \{v\})$, whose crossing edges are exactly v 's incident edges, shows the minimum edge at each vertex is forced into every MST.

WATCH OUT FOR

Confusing "every cycle's minimum edge is safe/must be in the MST" (false: it can be excluded once a cheaper path already connects its endpoints, as Kruskal shows) with the correct cut-property statement about a vertex's minimum incident edge; and assuming an edge's local extremal property (largest/smallest at a vertex) behaves symmetrically to its extremal property in a cycle, when cycles and cuts are genuinely dual/opposite structures in MST theory.

Q.50 MSQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

Q.50 Consider the following pseudocode for depth-first search (DFS) algorithm which takes a directed graph $G(V, E)$ as input, where $d[v]$ and $f[v]$ are the discovery time and finishing time, respectively, of the vertex $v \in V$.

```
DFS(G):
  unmark all v ∈ V
  t ← 0
  for each v ∈ V
     if v is unmarked
        t ← Explore(G, v, t)
     end if
  end for
```

```
Explore(G, v, t):
   mark v
   t ← t + 1
   d[v] ← t
   for each (v, w) ∈ E
      if w is unmarked
         t ← Explore(G, w, t)
      end if
   end for
   t ← t + 1
   f[v] ← t
   return t
```

Suppose that the input directed graph $G(V, E)$ is a directed acyclic graph (DAG).

For an edge $(u, v) \in E$, which of the following options will NEVER be correct?

(A) $d[u] < d[v] < f[v] < f[u]$

(B) $d[v] < d[u] < f[u] < f[v]$

(C) $d[v] < f[v] < d[u] < f[u]$

(D) $d[u] < d[v] < f[u] < f[v]$

Official answer: **B, D**

SOLUTION

1. Recall the DFS Parenthesis Theorem: for any edge (u,v) processed during DFS, the discovery/finish intervals [d[u],f[u]] and [d[v],f[v]] must be either completely nested or completely disjoint (never partially overlapping), regardless of whether the graph is cyclic or acyclic.
2. Option (A): d[u]<d[v]<f[v]<f[u] is the tree/forward-edge pattern (v's interval nested inside u's, u discovers v). This is a valid, common occurrence, including in a DAG. So (A) CAN be correct.

3. Option (C): d[v]<f[v]<d[u]<f[u] is the cross-edge pattern (v's interval entirely finished before u's interval starts, e.g. edge into an already-explored earlier subtree). This is allowed in a DAG since it creates no cycle. So (C) CAN be correct.
4. Option (B): d[v]<d[u]<f[u]<f[v] is the back-edge pattern (u's interval nested inside v's, meaning v is an ancestor of u still open on the DFS stack when edge (u,v) is traversed). A back edge always implies a cycle, formed by the tree-edge path from v down to u plus the back edge (u,v) closing it. Since G is a DAG, it has no cycles, so it has no back edges. Hence pattern (B) NEVER occurs.
5. Option (D): d[u]<d[v]<f[u]<f[v] describes intervals that partially overlap: neither nested nor disjoint. By the parenthesis theorem, this configuration is structurally impossible for ANY DFS on ANY directed graph (the recursive call/return discipline forces nesting or disjointness), independent of the DAG condition. Hence pattern (D) also NEVER occurs.
6. Therefore the two interval patterns that will never be correct for an edge (u,v) are (B) and (D).

KEY IDEA

The DFS Parenthesis Theorem gives exactly four possible relations between the [d,f] intervals of two vertices u, v for an edge (u,v): tree/forward (nested, u outer), back (nested, v outer), cross (disjoint, v's interval entirely before u's), and a fourth "partial overlap" pattern, which the theorem rules out for ANY DFS on ANY directed graph; it is not graph-type dependent. Separately, a DAG has no back edges by definition (a back edge always creates a cycle with the tree-path from v to u). So one excluded pattern (D) comes from the parenthesis theorem itself, and the other (B) comes specifically from the DAG constraint. The two impossibilities have different sources, but both must be excluded here.

WATCH OUT FOR

Two distinct traps: (1) treating this as "there is only one impossible pattern": students often stop after correctly ruling out the back-edge pattern via the DAG property and forget that option D is ALSO impossible, but for the separate, more fundamental reason (parenthesis theorem) that has nothing to do with the graph being acyclic. (2) Confusing which nesting direction is the back edge: back edge is v (ancestor, still open) containing u (descendant), i.e., d[v] < d[u] < f[u] < f[v], not the other nesting direction, which is the tree/forward-edge pattern.

Q.51 MSQ · 2 MARKS

Theory of Computation › Regular languages

Let L_1 and L_2 be two languages over a finite alphabet, such that $L_1 \cap L_2$ and L_2 are regular languages.

Which of the following statements is/are always true?

(A) L_1 is regular

(B) $L_1 \cup L_2$ is regular

(C) $\overline{L_2}$ is context-free

(D) L_1 is context-free

Official answer: **C**

SOLUTION

1. The only facts given are that $L_1 \cap L_2$ and L_2 are regular. Nothing else is assumed about L_1, so a statement is "always true" only if it holds for every pair (L_1, L_2) that satisfies these two conditions.
2. Counterexample for L_1: take $L_2 = \varnothing$, which is regular. Let L_1 be any language that is not regular. Then $L_1 \cap L_2 = \varnothing$ is regular, so both hypotheses hold while L_1 is arbitrary.
3. (A) L_1 is regular: false. With $L_2 = \varnothing$ and $L_1 = \{a^n b^n \mid n \geq 0\}$, the hypotheses hold but L_1 is not regular.
4. (D) L_1 is context-free: false. With $L_2 = \varnothing$ and $L_1 = \{a^n b^n c^n \mid n \geq 0\}$, which is not context-free, the hypotheses hold but L_1 is not context-free.

5. (B) $L_1 \cup L_2$ is regular: false. With $L_2 = \varnothing$ we get $L_1 \cup L_2 = L_1$, which is not regular for the chosen L_1. Also, $(L_1 \cap L_2) \cup L_2 = L_2$, which is not the same set as $L_1 \cup L_2$.
6. (C) $\overline{L_2}$ is context-free: true. Regular languages are closed under complement (swap the accepting and non-accepting states of a complete DFA), so $\overline{L_2}$ is regular. Every regular language is context-free, so $\overline{L_2}$ is context-free for every valid choice of L_1 and L_2.
7. Only (C) is always true.

KEY IDEA

The hypotheses constrain only L_2 and the intersection, so L_1 can be any language, for example when $L_2 = \varnothing$. Only a statement that depends on L_2 alone can be always true. Regular languages are closed under complement and are a subset of the context-free languages.

WATCH OUT FOR

Reading $L_1 \cap L_2$ regular as a restriction on L_1. Another slip is treating $(L_1 \cap L_2) \cup L_2$ as $L_1 \cup L_2$ and concluding (B). The union simplifies to L_2, not $L_1 \cup L_2$.

Q.52 MSQ · 2 MARKS

Theory of Computation › CFG & PDA

Consider the following context-free grammar G.

$$S \to abaABAbba$$
$$A \to aaBBAb \mid bBabaa$$
$$B \to aBb \mid ab$$

In the above grammar, S is the start symbol, a and b are terminal symbols, and A and B are non-terminal symbols.

Let $L(G)$ be the language generated by the grammar G. For a string $s \in L(G)$, let $n_1(s)$ be the number of a 's in s and $n_2(s)$ be the number of b's in s.

Which of the following statements is/are true?

(A) There is a string $s \in L(G)$ such that $n_1(s) < n_2(s)$

(B) For every string $s \in L(G)$, $n_1(s) \geq n_2(s)$

(C) There is a string $s \in L(G)$ such that $n_1(s) > 2n_2(s)$

(D) For every string $s \in L(G)$, $n_1(s) \leq 2n_2(s)$

Official answer: **B, D**

SOLUTION

1. Let $d(w) = n_1(w) - n_2(w)$ for a derived string w. The grammar is context-free and d is additive over concatenation, so we can compute d for each non-terminal separately.
2. $B \to aBb \mid ab$ generates exactly $a^k b^k$ with $k \geq 1$. So $d(B) = 0$ and B always contributes at least one b.
3. $A \to bBabaa$: the terminals b, a, b, a, a contribute 3 a's and 2 b's, so $d = 1$ plus $d(B) = 0$. This gives $d(A) = 1$ and at least $2 + 1 = 3$ b's.
4. $A \to aaBBAb$: the terminals a, a, b contribute $d = 2 - 1 = 1$. The two B's contribute 0 and add at least 2 b's. Using this rule m times and then $A \to bBabaa$ gives $d(A) = 1 + m$ and $n_2(A) \geq 3 + 3m$.
5. $S \to abaABAbba$ contains TWO copies of A. The terminals aba give $d = 1$ and bba give $d = -1$, and $d(B) = 0$. So with $m_1, m_2 \geq 0$ uses of the recursive rule in the two copies, $d(s) = 1 + (1 + m_1) + 0 + (1 + m_2) - 1 = 2 + m_1 + m_2 \geq 2$.
6. Hence $n_1(s) - n_2(s) \geq 2 > 0$ for every $s \in L(G)$. So $n_1 > n_2$ always: (A) is false and (B) is true.
7. Count the b's. The b's come from aba (1), bba (2), the middle B (at least 1), and the two A's (at least $3 + 3m_1$ and $3 + 3m_2$). So $n_2(s) \geq 10 + 3(m_1 + m_2)$.

8. Put $M = m_1 + m_2$. Then $d(s) = 2 + M$ and $n_2(s) \geq 10 + 3M$, so $d(s) \leq n_2(s)$ for every s. Therefore $n_1 = n_2 + d \leq 2n_2$ for every string: (D) is true and (C) is false.
9. Check on the smallest string: $B \to ab$ everywhere and $A \to bBabaa$ in both copies. Then $n_1 = 12$ and $n_2 = 10$, so $n_1 > n_2$ and $n_1 \leq 2n_2 = 20$. This agrees with the above.

KEY IDEA

Track the difference $n_1 - n_2$ for each non-terminal. B gives 0, each A gives $1 + m$, and S has two copies of A, so the difference is at least 2. The b-count grows at least three times as fast as the difference, which gives $n_1 \leq 2n_2$.

WATCH OUT FOR

Treating $S \to abaABAbba$ as if it had only one A. It has two. Dropping one A gives the wrong string counts, such as $n_2 = 7$ and $n_1 = 8$ for the smallest string. The correct smallest string has $n_2 = 10$ and $n_1 = 12$. The conclusions (B) and (D) still hold.

Q.53 MSQ · 2 MARKS

Compiler Design › Syntax-directed translation

Q.53 Consider the following two syntax-directed definitions SDD1 and SDD2 for type declarations.

SDD1	
Grammar (G1)	Semantic Rules
$D \rightarrow T\ V$	$D.type = T.type$ $V.type = T.type$
$T \rightarrow int$	$T.type = int$
$T \rightarrow float$	$T.type = float$
$V \rightarrow V_1\ id$	$V_1.type = V.type$ $put(id.entry, V.type)$
$V \rightarrow id$	$put(id.entry, V.type)$

SDD2	
Grammar (G2)	Semantic Rules
$D \rightarrow D_1\ id$	$D.type = D_1.type$ $put(id.entry, D_1.type)$
$D \rightarrow T\ id$	$D.type = T.type$ $put(id.entry, T.type)$
$T \rightarrow int$	$T.type = int$
$T \rightarrow float$	$T.type = float$

D is the start symbol, and int, $float$ and id are the three terminals. The non-terminal V_1 is the same as V and the non-terminal D_1 is the same as D. Here, the subscript is used to differentiate the grammar symbols on the two sides of a production. The function put updates the symbol table with the type information for an identifier.

Let P and Q be the languages specified by grammars G1 and G2, respectively.

Which of the following statements is/are true?

(A) The languages P and Q are the same

(B) SDD2 is S-attributed and contains only synthesized attributes

(C) SDD1 is L-attributed and contains only inherited attributes

(D) The specifications of SDD1 and SDD2 are such that the same entries get added to the symbol table

Official answer: **A, B, D**

SOLUTION

1. Language of G1: from $D \rightarrow T\,V$ and $V \rightarrow V_1\,\mathrm{id} \mid \mathrm{id}$, V derives id^+ (one or more ids), so $P = (\mathrm{int} \mid \mathrm{float})\,\mathrm{id}^+$.
2. Language of G2: from $D \rightarrow D_1\,\mathrm{id} \mid T\,\mathrm{id}$, D derives $T\,\mathrm{id}\,\mathrm{id}^*$, so $Q = (\mathrm{int} \mid \mathrm{float})\,\mathrm{id}^+$. Hence $P = Q$ and (A) is true.

3. SDD2: every rule defines the head's attribute from attributes of body symbols ($D.\text{type}$ from $D_1.\text{type}$ or $T.\text{type}$; $T.\text{type}$ from a constant). The put calls only read $D_1.\text{type}$ or $T.\text{type}$, and id.entry is a lexical value supplied by the scanner. So every attribute is synthesized, SDD2 is S-attributed, and (B) is true.
4. SDD1: $V.\text{type} = T.\text{type}$ and $V_1.\text{type} = V.\text{type}$ define an attribute of a body symbol from the parent's or a left sibling's attribute, so $V.\text{type}$ is inherited and the definition is L-attributed. But $T.\text{type}$ and $D.\text{type}$ are defined from the head's own production body or a constant, so they are synthesized. SDD1 therefore does not contain only inherited attributes, and (C) is false.
5. Symbol table entries: in SDD1 each id receives $V.\text{type}$, which is copied from $T.\text{type}$ down the chain of V nodes. In SDD2 each id receives $T.\text{type}$ (first id) or $D_1.\text{type}$ (later ids), which is the same declared type passed up the D chain. In both, every id of a declaration is entered with the declared type, so the same entries are added and (D) is true.
6. Answer: A, B and D.

KEY IDEA
Synthesized attributes depend on the children (or constants); inherited attributes depend on the parent or left siblings. S-attributed means synthesized only. Compare grammars by the languages they derive, and compare SDDs by what they put in the symbol table.

WATCH OUT FOR
Calling SDD1 'only inherited' because $V.\text{type}$ is inherited, forgetting that $T.\text{type}$ and $D.\text{type}$ are synthesized. Another slip is thinking G2's left recursion changes the language, when both grammars generate the same strings.

Q.54 MSQ · 2 MARKS

Computer Organization and Architecture › Memory & cache

Consider a system that has a cache memory unit and a memory management unit (MMU). The address input to the cache memory is a physical address. The MMU has a translation lookaside buffer (TLB). Assume that when a page is evicted from the main memory, the corresponding blocks in the cache are marked as invalid.

For a given memory reference, which of the following sequences of events can NEVER happen?

(A) TLB miss, Page table hit, Cache hit

(B) TLB hit, Page table miss, Cache hit

(C) TLB miss, Page table miss, Cache hit

(D) TLB miss, Page table miss, Cache miss

Official answer: **B, C**

Q.55 MSQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

An undirected, unweighted, simple graph $G(V, E)$ is said to be 2-colorable if there exists a function $c : V \to \{0, 1\}$ such that for every $(u, v) \in E$, $c(u) \neq c(v)$.

Which of the following statements about 2-colorable graphs is/are true?

(A) If G is 2-colorable, then G may contain cycles of odd length

(B) If G is 2-colorable, then G may contain cycles of even length

(C) An optimal algorithm for testing whether G is 2-colorable runs in time $\Theta(|V| + |E|)$, if G is represented as an adjacency list

(D) An optimal algorithm for testing whether G is 2-colorable runs in time $\Theta(|E| \log |V|)$, if G is represented as an adjacency list

Official answer: **B, C**

SOLUTION

1. Restate the theorem: a graph is 2-colorable (admits a proper coloring with 2 colors) if and only if it is bipartite, if and only if it contains no cycle of odd length.
2. Check (A): if G contained an odd-length cycle, traversing the cycle and alternating colors 0,1,0,1,... would force the last vertex back to color 0, which is adjacent to the starting vertex also colored 0, violating $c(u) \neq c(v)$ on that edge. Hence a 2-colorable graph can never contain an odd cycle, so (A) is false.
3. Check (B): a 2-colorable graph can contain even-length cycles: for instance C_4 (a 4-cycle) admits the coloring $0, 1, 0, 1$ around the cycle, which is a valid 2-coloring. So (B) is true.
4. Check (C): a single BFS or DFS from each uncolored vertex, assigning the opposite color to each newly discovered neighbor and checking every edge for a same-color violation, visits every vertex once and examines every edge a constant number of times, giving $\Theta(|V| + |E|)$ time on an adjacency-list representation. Since any correct algorithm must read the whole input (every vertex and edge) at least once, $\Omega(|V| + |E|)$ is a lower bound, so this algorithm is optimal. Hence (C) is true.
5. Check (D): $\Theta(|E| \log |V|)$ is asymptotically worse than the achievable and optimal $\Theta(|V| + |E|)$ bound (whenever $|V| \geq 2$), so it cannot describe an optimal algorithm's running time. Hence (D) is false.
6. Conclusion: the true statements are (B) and (C).

KEY IDEA

2-colorability of a graph is exactly bipartiteness, and bipartiteness is testable in linear time by a single BFS/DFS coloring pass.

WATCH OUT FOR

Confusing "2-colorable" with "properly colored graphs can have any cycle type"; a bipartite (2-colorable) graph can never contain an odd cycle, since traversing an odd cycle forces two adjacent vertices to receive the same color; students sometimes pick (A) by only checking small examples instead of proving the general impossibility.

Q.56 MSQ · 2 MARKS

Computer Networks › IPv4

An ISP having an address block 202.16.0.0/15 assigns a block of 6000 IP addresses to a client, using the classless internet domain routing (CIDR) super-netting approach. Which of the following address blocks can be assigned by the ISP?

(A) 202.16.0.0/19

(B) 202.17.64.0/19

(C) 202.16.32.0/19

(D) 202.17.24.0/19

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

SOLUTION

1. The ISP block 202.16.0.0/15 has $32 - 15 = 17$ host bits, so it holds $2^{17} = 131072$ addresses. A /15 fixes the first 15 bits, so the second octet may differ only in its last bit. The block therefore runs from 202.16.0.0 to 202.17.255.255.
2. A block of 6000 addresses needs $\lceil \log_2 6000 \rceil = 13$ host bits, because $2^{12} = 4096 < 6000 \leq 8192 = 2^{13}$. The client's block is therefore a /19 ($32 - 13 = 19$), which holds 8192 addresses.

3. A /19 block must be aligned. Its network address has all 13 host bits equal to zero. A /19 fixes 3 bits of the third octet, so the third octet must be a multiple of $2^5 = 32$, that is 0, 32, 64, 96, and so on.
4. A candidate must also lie inside the ISP's range, 202.16.0.0 to 202.17.255.255.
5. (A) 202.16.0.0/19: the third octet 0 is a multiple of 32. The block runs from 202.16.0.0 to 202.16.31.255, inside the ISP range. Valid.
6. (B) 202.17.64.0/19: the third octet 64 is a multiple of 32. The block runs from 202.17.64.0 to 202.17.95.255, inside the ISP range. Valid.
7. (C) 202.16.32.0/19: the third octet 32 is a multiple of 32. The block runs from 202.16.32.0 to 202.16.63.255, inside the ISP range. Valid.
8. (D) 202.17.24.0/19: 24 is not a multiple of 32, so the host bits are not all zero. The /19 that contains this address is 202.17.0.0/19, so 202.17.24.0/19 is not a valid block. Invalid.
9. The blocks that can be assigned are A, B and C.

KEY IDEA

6000 addresses need 13 host bits, so the client gets a /19 of 8192 addresses. A valid /19 starts on a multiple of 32 in the third octet and lies inside the ISP's /15, which spans 202.16.0.0 to 202.17.255.255.

WATCH OUT FOR

Checking only that the block lies inside the ISP's range and forgetting alignment. 202.17.24.0 is inside the range, but a /19 must start on a multiple of 32 in the third octet, and 24 is not one. Another slip is rounding 6000 down to a /20 (4096 addresses), which is too small.

Q.57 NAT · 2 MARKS

Discrete Mathematics › Graph theory

Let G be an undirected graph, which is a path on 8 vertices. The number of matchings in G is _____. (answer in integer)

Official answer: **34**

SOLUTION

1. Let m_n be the number of matchings (including the empty one) in the path P_n on n vertices. P_1 has no edges, so $m_1 = 1$; P_2 has one edge, so $m_2 = 2$ (the empty matching, or the single edge).
2. Split on the last vertex v_n. If v_n is unmatched, the rest is any matching of P_{n-1}, giving m_{n-1} choices. If v_n is matched, it must be matched to its only neighbour v_{n-1}, and the rest is any matching of P_{n-2}, giving m_{n-2} choices. The two cases are disjoint and cover everything.
3. So $m_n = m_{n-1} + m_{n-2}$, a Fibonacci recurrence.
4. Compute: $m_3 = 3$, $m_4 = 5$, $m_5 = 8$, $m_6 = 13$, $m_7 = 21$, $m_8 = 34$.
5. Check by counting on edges. A matching with k edges in the path P_8 (7 edges) is a choice of k pairwise non-adjacent edges, which can be done in $\binom{8-k}{k}$ ways. So the count is $\binom{8}{0} + \binom{7}{1} + \binom{6}{2} + \binom{5}{3} + \binom{4}{4} = 1 + 7 + 15 + 10 + 1 = 34$.
6. The empty matching is included, because a matching is any set of pairwise non-adjacent edges, and the empty set qualifies. Both methods agree: the answer is 34.

KEY IDEA

Count matchings of a path with a recurrence on the last vertex: it is either unmatched or matched to its single neighbour, so $m_n = m_{n-1} + m_{n-2}$, which gives the Fibonacci numbers. The empty matching counts.

WATCH OUT FOR

Leaving out the empty matching and answering 33, or counting only perfect matchings (which gives 1) or only maximal ones. Another slip is starting the recurrence with $m_1 = 0$ or $m_2 = 1$.

Q.58 NAT · 2 MARKS

Engineering Mathematics › Probability & statistics

Let X be a random variable which takes values in the set $\{1, 2, 3, 4, 5, 6, 7, 8\}$. Further, $\Pr(X = 1) = \Pr(X = 2) = \Pr(X = 5) = \Pr(X = 7) = \frac{1}{6}$ and $\Pr(X = 3) = \Pr(X = 4) = \Pr(X = 6) = \Pr(X = 8) = \frac{1}{12}$. The expected value of X, denoted by $E[X]$, is equal to _____. (rounded off to two decimal places)

Official answer: **4.24 to 4.26**

SOLUTION

1. The probabilities must sum to 1: $4 \cdot \frac{1}{6} + 4 \cdot \frac{1}{12} = \frac{2}{3} + \frac{1}{3} = 1$, so the distribution is valid.
2. The expected value is $E[X] = \sum_x x \Pr(X = x)$.
3. The values with probability $\frac{1}{6}$ are $1, 2, 5, 7$. Their contribution is $\frac{1+2+5+7}{6} = \frac{15}{6} = 2.5$.
4. The values with probability $\frac{1}{12}$ are $3, 4, 6, 8$. Their contribution is $\frac{3+4+6+8}{12} = \frac{21}{12} = 1.75$.
5. Add the two contributions: $E[X] = 2.5 + 1.75 = 4.25$.

KEY IDEA

Group the values by their probability. Then sum $x \cdot \Pr(X = x)$ over each group and add the groups.

WATCH OUT FOR

Assuming a uniform distribution gives $\frac{1+2+\cdots+8}{8} = 4.5$. The probabilities are not equal, so each value must be weighted by its own probability.

Q.59 NAT · 2 MARKS

Operating Systems › File systems & disks

Consider a hard disk with a rotational speed of 15000 rpm. The time to move the read/write head from a track to its adjacent track is 1 millisecond. Initially, the head is on track 0. The number of sectors per track is 400. The sector size is 1024 bytes. It is necessary to transfer data from 10 randomly located sectors in each of the following tracks in the order: 5, 12 and 7.

The total time for the data transfer (in milliseconds) from the hard disk is _____. (rounded off to one decimal place)

Official answer: **77.3**

SOLUTION

1. One revolution takes $\frac{60000 \text{ ms}}{15000} = 4 \text{ ms}$.
2. The sectors are randomly located, so the average rotational latency before each sector is half a revolution: $\frac{4}{2} = 2 \text{ ms}$.
3. A track holds 400 sectors and one revolution passes all of them, so one sector is transferred in $\frac{4}{400} = 0.01 \text{ ms}$. The sector size (1024 bytes) is not needed.
4. Time per sector $= 2 + 0.01 = 2.01 \text{ ms}$. There are $3 \times 10 = 30$ sectors, so the rotational latency and transfer time total $30 \times 2.01 = 60.3 \text{ ms}$.
5. The head starts on track 0 and visits tracks 5, 12 and 7 in that order, moving $5 + 7 + 5 = 17$ tracks. At 1 ms per track, the seek time is 17 ms.
6. Total time $= 17 + 60.3 = 77.3 \text{ ms}$.

KEY IDEA

Disk access time is seek time plus rotational latency plus transfer time. Random sectors each cost half a revolution of latency, and the seek is the total track distance travelled in the given visiting order.

WATCH OUT FOR

Counting the seeks as 5 + 12 + 7 = 24 tracks, as if the head returned to track 0 each time. The head moves from track 0 to 5, then 5 to 12, then 12 to 7, which is 17 tracks. Another slip is to add a full revolution of latency per sector, or to count the latency only once per track.

Q.60 NAT · 2 MARKS

Q.60 The EX stage of a pipelined processor performs the memory read operations for LOAD instructions, and the operations for the arithmetic and logic instructions. Let t_{EX} denote the time taken by the EX stage to perform the operation for an instruction. For each instruction type, the values of t_{EX} and M (the number of instructions of that type in a sequence of 100 instructions for a program P), are given in the table below.

The duration of the pipeline clock cycle is 1 nanosecond. Assume that the latch time for the interstage buffers in the pipeline is negligible.

Instruction	t_{EX} in nanoseconds	M
LOAD	1.8	15
IMUL	1.5	10
IDIV	2.5	5
FADD	1.7	10
FSUB	1.7	5
FMUL	2.8	15
FDIV	3.2	5
All other instructions	Less than 1.0	35

When program P is executed, the number of clock cycles for which the pipeline is stalled due to structural hazards in the EX stage is _____. (*answer in integer*)

Official answer: **95**

SOLUTION

1. Since the clock cycle is 1 ns and the EX stage is a single shared functional unit, an instruction whose t_EX exceeds 1 ns must occupy the EX stage for multiple consecutive clock cycles before the next instruction can enter it: this is a structural hazard on the EX stage.
2. An instruction with execution time t_EX occupies the EX stage for ceil(t_EX / 1 ns) clock cycles, so it causes ceil(t_EX) − 1 stall cycles for the instruction immediately behind it (0 stalls if t_EX ≤ 1 ns).
3. Compute ceil(t_EX) and the resulting stall cycles per instruction, per type: LOAD 1.8 ns → 2 cycles → 1 stall; IMUL 1.5 ns → 2 cycles → 1 stall; IDIV 2.5 ns → 3 cycles → 2 stalls; FADD 1.7 ns → 2 cycles → 1 stall; FSUB 1.7 ns → 2 cycles → 1 stall; FMUL 2.8 ns → 3 cycles → 2 stalls; FDIV 3.2 ns → 4 cycles → 3 stalls; 'All other instructions' (t_EX < 1.0 ns) → 1 cycle → 0 stalls.
4. Multiply each type's per-instruction stall count by its count M in the 100-instruction sequence and sum: LOAD 1×15=15; IMUL 1×10=10; IDIV 2×5=10; FADD 1×10=10; FSUB 1×5=5; FMUL 2×15=30; FDIV 3×5=15; others 0×35=0.
5. Total stall cycles = 15+10+10+10+5+30+15+0 = 95.

KEY IDEA

When a functional unit (here, the EX stage) is not internally pipelined and one instruction's operation takes longer than one clock cycle, the unit must hold that instruction for multiple cycles before it can accept the next one: a structural hazard. An instruction whose EX time is t nanoseconds occupies the EX stage for $\lceil t \rceil$ clock cycles (since the clock period is 1 ns), so it delays (stalls) the instruction behind it by $\lceil t \rceil - 1$ cycles. Total stall cycles for the program = $\sum(\lceil t_{EX} \rceil - 1) \times M$ over all instruction types.

WATCH OUT FOR

Two common errors: (1) using $\lceil t \rceil$ itself (the total cycles occupied) instead of $\lceil t \rceil - 1$ (the stall cycles caused) as the per-instruction penalty, which overcounts by exactly M_{total} instructions with $t_{EX} > 1$; (2) rounding t_{EX} to the nearest integer instead of taking the ceiling (e.g., treating 1.5 ns or 2.5 ns as needing only 1–2 cycles), which understates the number of clock cycles actually required to complete the operation.

Q.61 NAT · 2 MARKS

Programming in C › Recursion

Consider the recursive functions represented by the following code segment:

```
int bar(int n){
  if (n == 1) return 0;
  else return 1 + bar(n/2);
}

int foo(int n){
  if (n == 1) return 1;
  else return 1 + foo(bar(n));
}
```

The smallest positive integer n for which foo(n) returns 5 is _____. (answer in integer)

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

Official answer: **65536**

SOLUTION

1. foo(n) unwinds as 1 + foo(bar(n)) until the argument becomes 1, where foo(1)=1. So foo(n)=5 means exactly 4 applications of bar are needed to bring n down to 1: n0=n → n1=bar(n0) → n2=bar(n1) → n3=bar(n2) → n4=bar(n3)=1, with n0, n1, n2, n3 all ≠ 1.
2. bar(m) repeatedly integer-halves m until it reaches 1, counting the halvings; tracing the recursion shows this is exactly bar(m) = $\lfloor \log_2 m \rfloor$ for every m ≥ 1 (verified on bar(2)=1, bar(3)=1, bar(4)=2, bar(7)=2, bar(8)=3).
3. bar(n3)=1 requires $\lfloor \log_2 n3 \rfloor = 1$, i.e. n3 ∈ {2,3}. Since $\lfloor \log_2 \cdot \rfloor$ is non-decreasing, the smallest n3 gives the smallest possible value to build back from at every later stage, so take the minimal n3 = 2.
4. bar(n2) = n3 = 2 requires $\lfloor \log_2 n2 \rfloor = 2$, i.e. n2 ∈ {4,5,6,7}; the minimal choice is n2 = 4.
5. bar(n1) = n2 = 4 requires $\lfloor \log_2 n1 \rfloor = 4$, i.e. n1 ∈ {16,...,31}; the minimal choice is n1 = 16.
6. bar(n0) = n1 = 16 requires $\lfloor \log_2 n0 \rfloor = 16$, i.e. n0 ∈ {65536,...,131071}; the minimal choice is n0 = 65536.
7. Verification by direct trace: bar(65536)=16, bar(16)=4, bar(4)=2, bar(2)=1, bar(1)=0, so foo(65536) = 1+foo(16) = 2+foo(4) = 3+foo(2) = 4+foo(1) = 4+1 = 5. Correct, and no smaller n achieves this since each stage above used the smallest possible value.

KEY IDEA

`foo` counts how many times `bar` must be applied to bring `n` down to 1, plus one; `bar` itself computes $\lfloor \log_2 m \rfloor$ (the number of integer-halvings needed to reach 1). Since $\lfloor \log_2 m \rfloor$ is non-decreasing in m, the smallest n reaching a target value after k nested applications of `bar` is built from the *inside out*: at each stage pick the smallest number whose $\lfloor \log_2 \cdot \rfloor$ equals the previous stage's smallest value, i.e. exactly 2^{value}.

WATCH OUT FOR

Confusing `bar(n)` with `n` halved once (giving `bar(n) = n/2`) instead of recognizing it as the *iterated*-halving count $\lfloor \log_2 n \rfloor$; this leads to solving $n/2^4 = 1 \Rightarrow n = 16$, missing that `foo` calls `bar` fresh on each level (nesting $\lfloor \log_2 \rfloor$ four times, not dividing by 2 four times).

Q.62 NAT · 2 MARKS

Data Structures and Algorithms › Trees & heaps

The following sequence corresponds to the preorder traversal of a binary search tree T:

$$50, 25, 13, 40, 30, 47, 75, 60, 70, 80, 77$$

The position of the element 60 in the postorder traversal of T is _____. (answer in integer)

Note: The position begins with 1.

Official answer: **7**

SOLUTION

1. Reconstruct the BST from preorder using the property: first element is the root; among the remaining elements, those less than the root form the left subtree's preorder, those greater form the right subtree's preorder, recursively.
2. Root = 50. Left subtree preorder (values < 50): 25, 13, 40, 30, 47. Right subtree preorder (values > 50): 75, 60, 70, 80, 77.
3. Left subtree: root 25; left of 25 = {13}; remaining 40, 30, 47 (all > 25) form the right-of-25 preorder: root 40, left = {30}, right = {47}. So left subtree = 25(13, 40(30, 47)).
4. Right subtree: root 75; left-of-75 preorder is 60, 70 (both < 75): root 60, and since 70 > 60 it is the right child of 60, giving 60(-, 70); right-of-75 preorder is 80, 77 (both > 75): root 80, and since 77 < 80 it is the left child of 80, giving 80(77, -). So right subtree = 75(60(-, 70), 80(77, -)).
5. Postorder (left, right, root) of the 25-subtree: postorder(13) = 13; postorder(40-subtree) = postorder(30), postorder(47), 40 = 30, 47, 40; so postorder(25-subtree) = 13, 30, 47, 40, 25.
6. Postorder of the 75-subtree: postorder(60-subtree) = (nothing), 70, 60 = 70, 60; postorder(80-subtree) = 77, (nothing), 80 = 77, 80; so postorder(75-subtree) = 70, 60, 77, 80, 75.
7. Full postorder of T = [13, 30, 47, 40, 25] + [70, 60, 77, 80, 75] + [50] = 13, 30, 47, 40, 25, 70, 60, 77, 80, 75, 50.
8. Counting from position 1: 13(1), 30(2), 47(3), 40(4), 25(5), 70(6), 60(7), 77(8), 80(9), 75(10), 50(11). The element 60 is at position 7.

KEY IDEA

In a BST, a preorder listing splits uniquely at each node into (root, values less than root = left subtree preorder, values greater than root = right subtree preorder). Recursively rebuild both subtrees this way, then compute postorder (left, right, root) on the rebuilt tree.

WATCH OUT FOR

Trying to compute the postorder position of 60 without fully separating each subtree's preorder segment first, e.g. misplacing 70 as a left child of 75 instead of the right child of 60, or misplacing 77 as the right child of 75 instead of the left child of 80, which shifts the final position count.

Q.63 NAT · 2 MARKS

Operating Systems › Processes & threads

Consider the following program snippet. Assume that the program compiles and runs successfully. Further, assume that the `fork()` system call is always successful in creating a process.

```
int main () {
  int i;
  for (i = 0; i < 3; i++){
    if (fork() == 0){
      continue;
    }
    break;
  }
  printf("Hello!");
  return 0;
}
```

The total number of times that the `printf` statement gets executed is _____. (answer in integer)

Official answer: **4**

SOLUTION

1. fork() returns 0 in the child and a positive PID in the parent. The child runs `continue` and moves to the next iteration. The parent skips the `if` body, runs `break`, leaves the loop and reaches `printf`.
2. Iteration $i = 0$: the original process P_0 forks P_1. P_0 breaks out and prints (1 print). P_1 continues with $i = 1$.
3. Iteration $i = 1$: P_1 forks P_2. P_1 breaks out and prints (2 prints so far). P_2 continues with $i = 2$.
4. Iteration $i = 2$: P_2 forks P_3. P_2 breaks out and prints (3 prints so far). P_3 continues and i becomes 3.
5. For P_3 the condition $i < 3$ is false, so it leaves the loop normally and prints (4 prints in total). P_3 forks nothing more.
6. Only one process stays in the loop at each level, so the processes form a chain $P_0 \to P_1 \to P_2 \to P_3$, not a tree. Each of the four processes reaches `printf` exactly once. Total prints $= 3$ (parents that break) $+1$ (last child) $= 4$.

KEY IDEA

Here the child continues the loop and the parent breaks out. So the processes form a chain, with one new process per iteration, instead of the doubling that happens when every process keeps looping. The count is the number of loop iterations plus the original process: $3 + 1 = 4$.

WATCH OUT FOR

Treating this like the usual `for` loop with `fork()`, where the number of processes doubles each iteration, and answering $2^3 = 8$. The `break` in the parent and the `continue` in the child stop the tree from growing.

Q.64 NAT · 2 MARKS

Operating Systems › CPU scheduling

Consider a CPU that has to execute two types of processes. The first type, Actuators (A), requires a CPU burst of 6 seconds. The second type, Controllers (C), requires a CPU burst of 8 seconds. A new process of type A arrives at time $t = 10$, 20, 30, 40, and 50 (in seconds). Similarly, a new process of type C arrives at time $t = 11$, 22, 33, 44, and 55 (in seconds). The CPU scheduling policy is First Come First Serve (FCFS).

The first process of type A starts running at $t = 10$ seconds. The average waiting time (in seconds) for the 10 processes is _____. (rounded off to one decimal place)

Official answer: **9.5**

SOLUTION

1. Arrival order (FCFS serves in this order): $A_1(10), C_1(11), A_2(20), C_2(22), A_3(30), C_3(33), A_4(40), C_4(44), A_5(50), C_5(55)$. Bursts: A = 6 s, C = 8 s.
2. Gantt chart, starting at $t = 10$. Each process starts when the previous one finishes, or on arrival if the CPU is idle. From $t = 10$ the queue is never empty, so the CPU never idles: A_1 : 10–16, C_1 : 16–24, A_2 : 24–30, C_2 : 30–38, A_3 : 38–44, C_3 : 44–52, A_4 : 52–58, C_4 : 58–66, A_5 : 66–72, C_5 : 72–80.
3. Waiting time = start time − arrival time (each process runs without preemption):

Process	Arrival	Start	Wait
A_1	10	10	0
C_1	11	16	5
A_2	20	24	4
C_2	22	30	8
A_3	30	38	8
C_3	33	44	11
A_4	40	52	12
C_4	44	58	14
A_5	50	66	16
C_5	55	72	17

4. Total waiting time $= 0 + 5 + 4 + 8 + 8 + 11 + 12 + 14 + 16 + 17 = 95$ seconds.
5. Average waiting time $= 95/10 = 9.5$ seconds.

KEY IDEA

Under FCFS the processes run in arrival order without preemption. Each one starts at the later of its own arrival time and the previous process's finish time. Its waiting time is its start time minus its arrival time.

WATCH OUT FOR

Averaging turnaround times instead of waiting times, or assuming the CPU idles between jobs. Arrivals are 10 s and 11 s apart but each pair needs 14 s of CPU, so the queue keeps growing and waiting times rise from 0 to 17.

Q.65 NAT · 2 MARKS

Database Management and Warehousing › Normal forms

Consider a relational database schema with a relation $R(A, B, C, D)$. If $\{A, B\}$ and $\{A, C\}$ are the only two candidate keys of the relation R, then the number of superkeys of relation R is _____. (answer in integer)

Official answer: **6**

SOLUTION

1. A superkey is any attribute set that contains a candidate key. The only candidate keys are $\{A, B\}$ and $\{A, C\}$, so a set is a superkey exactly when it contains $\{A, B\}$ or contains $\{A, C\}$.
2. Let X be the family of supersets of $\{A, B\}$. The attributes C and D may each be present or absent, so $|X| = 2^2 = 4$: $AB, ABC, ABD, ABCD$.
3. Let Y be the family of supersets of $\{A, C\}$. The attributes B and D are free, so $|Y| = 2^2 = 4$: $AC, ABC, ACD, ABCD$.
4. A set in both families contains A, B and C, with D free. So $|X \cap Y| = 2^1 = 2$: ABC and $ABCD$.
5. By inclusion–exclusion, $|X \cup Y| = 4 + 4 - 2 = 6$.
6. Listing them: $AB, AC, ABC, ABD, ACD, ABCD$. There are 6 superkeys.

KEY IDEA

A superkey is any set that contains a candidate key. Count the supersets of each candidate key and use inclusion–exclusion so that sets containing both keys are not counted twice.

WATCH OUT FOR

Adding $4 + 4 = 8$ and forgetting that ABC and $ABCD$ contain both candidate keys. Another slip is to count only the candidate keys themselves, or to leave out the supersets that include D.

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 1 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