IAE ACADEMY INSTITUTE FOR ENGINEERS

GATE CS · 2025 · SET 2

IAE Academy free study material · Updated 27 October 2026

GATE CS 2025 Set 2 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 Roorkee

Online, with the questions linked to their topic notes: **iaeacademy.in/study/papers/cs-2025-2**

FREE COUNSELLING

Preparing for GATE CS? Talk to IAE Academy.

Or call **+91 84204 36524**

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

Chat on WhatsApp

Free. Not for sale. Pass it on.

About this paper

65 questions, 100 marks. Question types: 32 MCQ · 18 MSQ · 15 NAT.

Official answer key: IIT Roorkee · https://gate2025.iitr.ac.in/doc/2025/2025_Key/CS2_Keys.pdf

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

This paper online, with the questions linked to their topic notes:
iaeacademy.in/study/papers/cs-2025-2

Answer key

Q	TYPE	MARKS	ANSWER
Q.1	MCQ	1	**C**
Q.2	MCQ	1	**C**
Q.3	MCQ	1	**A**
Q.4	MCQ	1	**A**
Q.5	MCQ	1	**B**
Q.6	MCQ	2	**B**
Q.7	MCQ	2	**B**
Q.8	MCQ	2	**A**
Q.9	MCQ	2	**D**
Q.10	MCQ	2	**C**
Q.11	MCQ	1	**C**
Q.12	MCQ	1	**A**
Q.13	MCQ	1	**B**
Q.14	MCQ	1	**A**
Q.15	MCQ	1	**A**
Q.16	MCQ	1	**B**
Q.17	MCQ	1	**A**
Q.18	MCQ	1	**D**
Q.19	MCQ	1	**D**
Q.20	MCQ	1	**A**
Q.21	MCQ	1	**C**
Q.22	MCQ	1	**C**
Q.23	MCQ	1	**D**
Q.24	MCQ	1	**A**
Q.25	MCQ	1	**D**
Q.26	MCQ	1	**D**
Q.27	MSQ	1	**B, C**
Q.28	MSQ	1	**D**
Q.29	MSQ	1	**B, C, D**
Q.30	MSQ	1	**B, C**
Q.31	MSQ	1	**A, B, C**
Q.32	NAT	1	**13**
Q.33	NAT	1	**21**
Q.34	NAT	1	**250**
Q.35	NAT	1	**4**
Q.36	MCQ	2	**A**
Q.37	MCQ	2	**C**
Q.38	MCQ	2	**A**
Q.39	MCQ	2	**A**
Q.40	MCQ	2	**C**
Q.41	MCQ	2	**D**
Q.42	MSQ	2	**B, C**
Q.43	MSQ	2	**A, D**
Q.44	MSQ	2	**A, B, C**
Q.45	MSQ	2	**A**
Q.46	MSQ	2	**C, D**
Q.47	MSQ	2	**D**
Q.48	MSQ	2	**C, D**
Q.49	MSQ	2	**A, B, C**
Q.50	MSQ	2	**B, C, D**
Q.51	MSQ	2	**C, D**
Q.52	MSQ	2	**A, C**
Q.53	MSQ	2	**B**
Q.54	MSQ	2	**A, C, D**
Q.55	NAT	2	**4.0**
Q.56	NAT	2	**260.20 to 261.20**
Q.57	NAT	2	**33**
Q.58	NAT	2	**11**
Q.59	NAT	2	**6**
Q.60	NAT	2	**6**
Q.61	NAT	2	**3.0**
Q.62	NAT	2	**111**
Q.63	NAT	2	**46**
Q.64	NAT	2	**0.5**
Q.65	NAT	2	**0.70 to 0.80**

General Aptitude Q.1 to Q.10

Q.1 MCQ · 1 MARK

Q.1	Despite his initial hesitation, Rehman's _____ to contribute to the success of the project never wavered. Select the most appropriate option to complete the above sentence.
(A)	ambivalence
(B)	satisfaction
(C)	resolve
(D)	revolve

Official answer: **C**

Q.2 MCQ · 1 MARK

Q.2	Bird : Nest :: Bee : _____ Select the correct option to complete the analogy.
(A)	Kennel
(B)	Hammock
(C)	Hive
(D)	Lair

Official answer: **C**

Q.3 MCQ · 1 MARK

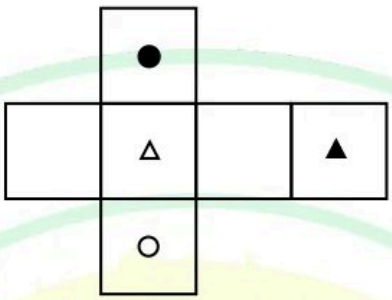
Q.3	If $Pe^{x} = Qe^{-x}$ for all real values of x, which one of the following statements is true?
(A)	$P = Q = 0$
(B)	$P = Q = 1$
(C)	$P = 1;\ Q = -1$
(D)	$\frac{P}{Q} = 0$

Official answer: **A**

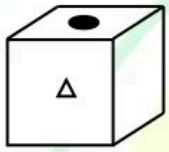
Q.4 MCQ · 1 MARK

Q.4 The paper as shown in the figure is folded to make a cube where each square corresponds to a particular face of the cube. Which one of the following options correctly represents the cube?

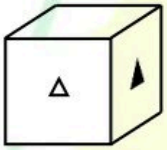
Note: The figures shown are representative.



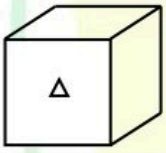
(A)



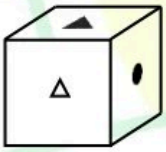
(B)



(C)



(D)



Official answer: **A**

Q.5 MCQ · 1 MARK

Q.5	Let p_1 and p_2 denote two arbitrary prime numbers. Which one of the following statements is correct for all values of p_1 and p_2?
(A)	$p_1 + p_2$ is not a prime number.
(B)	$p_1 p_2$ is not a prime number.
(C)	$p_1 + p_2 + 1$ is a prime number.
(D)	$p_1 p_2 + 1$ is a prime number.

Official answer: **B**

Q.6 MCQ · 2 MARKS

Q.6	Based only on the conversation below, identify the logically correct inference: "Even if I had known that you were in the hospital, I would not have gone there to see you", Ramya told Josephine.
(A)	Ramya knew that Josephine was in the hospital.
(B)	Ramya did not know that Josephine was in the hospital.
(C)	Ramya and Josephine were once close friends; but now, they are not.
(D)	Josephine was in the hospital due to an injury to her leg.

Official answer: **B**

Q.7 MCQ · 2 MARKS

Q.7	If IMAGE and FIELD are coded as FHBNJ and EMFJG respectively then, which one among the given options is the most appropriate code for BEACH ?
(A)	CEADP
(B)	IDBFC
(C)	JGIBC
(D)	IBCEC

Official answer: **B**

Q.8 MCQ · 2 MARKS

Q.8 Which one of the following options is correct for the given data in the table?

Iteration (i)	0	1	2	3
Input (I)	20	−4	10	15
Output (X)	20	16	26	41
Output (Y)	20	−80	−800	−12000

(A) $X(i) = X(i-1) + I(i); \quad Y(i) = Y(i-1)I(i); \quad i > 0$

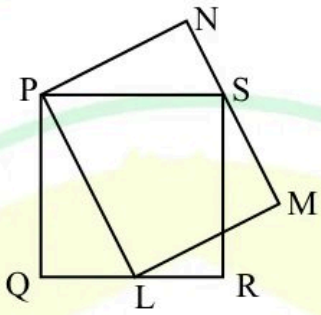
(B) $X(i) = X(i-1)I(i); \quad Y(i) = Y(i-1) + I(i); \quad i > 0$

(C) $X(i) = X(i-1)I(i); \quad Y(i) = Y(i-1)I(i); \quad i > 0$

(D) $X(i) = X(i-1) + I(i); \quad Y(i) = Y(i-1)I(i-1); \quad i > 0$

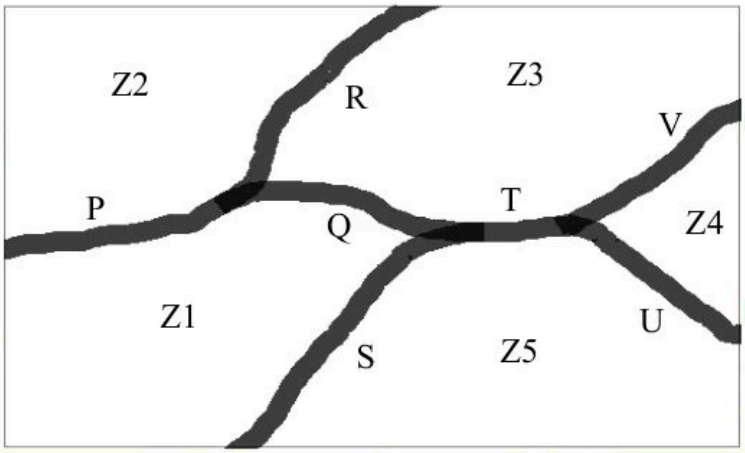
Official answer: **A**

Q.9 MCQ · 2 MARKS

Q.9	In the given figure, PQRS is a square of side 2 cm and PLMN is a rectangle. The corner L of the rectangle is on the side QR. Side MN of the rectangle passes through the corner S of the square. What is the area (in cm^2) of the rectangle PLMN? Note: The figure shown is representative.
	
(A)	$2\sqrt{2}$
(B)	2
(C)	8
(D)	4

Official answer: **D**

Q.10 MCQ · 2 MARKS

Q.10	The diagram below shows a river system consisting of 7 segments, marked P, Q, R, S, T, U, and V. It splits the land into 5 zones, marked Z1, Z2, Z3, Z4, and Z5. We need to connect these zones using the least number of bridges. Out of the following options, which one is correct? Note: The figure shown is representative.
	
(A)	Bridges on P, Q, and T
(B)	Bridges on P, Q, S, and T
(C)	Bridges on Q, R, T, and V
(D)	Bridges on P, Q, S, U, and V

Official answer: **C**

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

Q.11 MCQ · 1 MARK

Engineering Mathematics › Linear algebra

If $A = \begin{pmatrix} 1 & 2 \\ 2 & -1 \end{pmatrix}$, then which ONE of the following is A^8 ?

(A) $\begin{pmatrix} 25 & 0 \\ 0 & 25 \end{pmatrix}$

(B) $\begin{pmatrix} 125 & 0 \\ 0 & 125 \end{pmatrix}$

(C) $\begin{pmatrix} 625 & 0 \\ 0 & 625 \end{pmatrix}$

(D) $\begin{pmatrix} 3125 & 0 \\ 0 & 3125 \end{pmatrix}$

Official answer: **C**

SOLUTION

1. Do not multiply eight times. Compute A^2 first and look for a pattern.
2. $A^2 = \begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix}\begin{bmatrix} 1 & 2 \\ 2 & -1 \end{bmatrix} = \begin{bmatrix} 1+4 & 2-2 \\ 2-2 & 4+1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} = 5I.$
3. Cayley-Hamilton check: the trace is 0 and $\det A = -1 - 4 = -5$, so the characteristic equation is $\lambda^2 - 5 = 0$ and $A^2 = 5I$.
4. A scalar matrix commutes with everything, so $A^8 = (A^2)^4 = (5I)^4 = 5^4 I$.
5. $5^4 = 625$, so $A^8 = \begin{bmatrix} 625 & 0 \\ 0 & 625 \end{bmatrix}$, which is option (C).

KEY IDEA

Square the matrix first. Here $A^2 = 5I$ is a scalar matrix, so any even power is a power of 5 times I. Cayley-Hamilton confirms it: the trace is 0, so $A^2 = -\det(A)\, I$.

WATCH OUT FOR

Computing A^8 as 5^8 or 5^2 by miscounting powers. $A^8 = (A^2)^4 = 5^4 I$, not $5^8 I$ or $5^2 I$. Another slip is the arithmetic in A^2, where the off-diagonal terms $2 - 2$ cancel.

Q.12 MCQ · 1 MARK

Engineering Mathematics › Calculus

The value of x such that $x > 1$, satisfying the equation $\int_1^x t \ln t \, dt = \frac{1}{4}$ is

(A) $\sqrt{e}$

(B) e

(C) e^2

(D) $e - 1$

Official answer: **A**

SOLUTION

1. Integrate by parts with $u = \ln t$, $dv = t\,dt$, so $du = \frac{dt}{t}$ and $v = \frac{t^2}{2}$:
$$\int t \ln t \, dt = \frac{t^2}{2}\ln t - \int \frac{t}{2}\, dt = \frac{t^2}{2}\ln t - \frac{t^2}{4}.$$
2. Apply the limits 1 to x. At $t = 1$ the antiderivative equals $0 - \frac{1}{4}$, so
$$\int_1^x t \ln t \, dt = \frac{x^2}{2}\ln x - \frac{x^2}{4} + \frac{1}{4}.$$
3. Set this equal to $\frac{1}{4}$: $\frac{x^2}{2}\ln x - \frac{x^2}{4} = 0$, that is $\frac{x^2}{4}(2\ln x - 1) = 0$.

4. Since $x > 1$, $x^2 \neq 0$, so $2 \ln x = 1$. Hence $\ln x = \frac{1}{2}$ and $x = \sqrt{e}$.
5. Check: $\sqrt{e} \approx 1.65 > 1$, so the condition $x > 1$ holds. The answer is option (A).

KEY IDEA

Integrate $t \ln t$ by parts to get $\frac{t^2}{2} \ln t - \frac{t^2}{4}$. The constant $\frac{1}{4}$ from the lower limit cancels the right-hand side, leaving $\ln x = \frac{1}{2}$.

WATCH OUT FOR

Forgetting the lower-limit contribution $+\frac{1}{4}$ (or dropping the $-\frac{t^2}{4}$ term) gives a wrong equation. Another slip is integrating $t \ln t$ as $\frac{t^2}{2} \cdot \frac{1}{t}$ without the second integral.

Q.13 MCQ · 1 MARK

Data Structures and Algorithms › Trees & heaps

Consider a binary tree T in which every node has either zero or two children. Let $n > 0$ be the number of nodes in T.

Which ONE of the following is the number of nodes in T that have exactly two children?

(A) $\frac{n-2}{2}$
(B) $\frac{n-1}{2}$
(C) $\frac{n}{2}$
(D) $\frac{n+1}{2}$

Official answer: **B**

SOLUTION

1. Let n0 = number of leaf nodes (0 children) and n2 = number of nodes with exactly 2 children. Since every node has either 0 or 2 children, n1 = 0, so total nodes n = n0 + n2.
2. Count total edges two ways. Every node except the root has exactly one incoming edge from its parent, so total edges = n - 1.
3. Also, each node with exactly two children contributes exactly 2 outgoing edges (and nodes with 0 children contribute 0), so total edges = 2 * n2.
4. Equating the two counts: n - 1 = 2 * n2, so $n_2 = \frac{n-1}{2}$.
5. This matches option (B), and is the standard identity n0 = n2 + 1 for such trees (since n = n0 + n2 = (n2+1) + n2 = 2n2 + 1 gives the same result).

KEY IDEA

In a binary tree where every node has 0 or 2 children (a "full" or "proper" binary tree), counting edges two ways (n-1 total edges vs. 2 edges per 2-child node) gives n2 = (n-1)/2, equivalently the identity n0 = n2 + 1.

WATCH OUT FOR

Confusing "full binary tree" (every node has 0 or 2 children, the condition given here) with "complete binary tree" (all levels filled except possibly the last), or misremembering the identity as n0 = n2 (rather than n0 = n2 + 1), which would incorrectly give n2 = n/2, option (C).

Q.14 MCQ · 1 MARK

Engineering Mathematics › Linear algebra

Let L, M, and N be non-singular matrices of order 3 satisfying the equations $L^2 = L^{-1}$, $M = L^8$ and $N = L^2$. Which ONE of the following is the value of the determinant of $(M - N)$?

(A) 0
(B) 1
(C) 2

(D) 3

Official answer: **A**

SOLUTION

1. From $L^2 = L^{-1}$, multiply both sides on the right by L (allowed since L is non-singular): $L^3 = L^{-1}L = I$.
2. Reduce the power in M: $M = L^8 = (L^3)^2 \cdot L^2 = I \cdot L^2 = L^2$.
3. Given $N = L^2$, so $M = N$.
4. Then $M - N$ is the 3×3 zero matrix.
5. The determinant of the zero matrix is 0, so $\det(M - N) = 0$.

KEY IDEA

$L^2 = L^{-1}$ gives $L^3 = I$, so $L^8 = L^2$. Then $M = N$ and $M - N$ is the zero matrix, whose determinant is 0.

WATCH OUT FOR

Computing $\det M$ and $\det N$ separately and subtracting them. The determinant is not linear, so $\det(M - N) \neq \det M - \det N$ in general. Here $M - N$ is the zero matrix itself.

Q.15 MCQ · 1 MARK

Discrete Mathematics › Mathematical logic

Let $P(x)$ be an arbitrary predicate over the domain of natural numbers.

Which ONE of the following statements is TRUE?

(A) $(P(0) \wedge (\forall x\,[P(x) \Rightarrow P(x+1)])) \Rightarrow (\forall x\,P(x))$

(B) $(P(0) \wedge (\forall x\,[P(x) \Rightarrow P(x-1)])) \Rightarrow (\forall x\,P(x))$

(C) $(P(1000) \wedge (\forall x\,[P(x) \Rightarrow P(x-1)])) \Rightarrow (\forall x\,P(x))$

(D) $(P(1000) \wedge (\forall x\,[P(x) \Rightarrow P(x+1)])) \Rightarrow (\forall x\,P(x))$

Official answer: **A**

SOLUTION

1. Mathematical induction over the natural numbers says: if $P(0)$ holds and $\forall x\,[P(x) \Rightarrow P(x+1)]$ holds, then $\forall x\,P(x)$ holds.
2. Option (A) is exactly this principle, with base case $P(0)$ and an upward step $x \to x+1$. It is true for every predicate P over the naturals.
3. Option (B): the step goes downward. Take $P(x)$ true only for $x = 0$. Then $P(0)$ holds. For $x \geq 1$, $P(x)$ is false, so $P(x) \Rightarrow P(x-1)$ is true. For $x = 0$, $P(0) \Rightarrow P(-1)$ has a true premise, but -1 is not a natural number, so the step is only a claim over the naturals. Reading the step as ranging over $x \geq 1$ (the only values for which $x - 1$ is a natural number), the hypothesis holds while $\forall x\,P(x)$ is false. So (B) is not true for every predicate.
4. Option (C): take $P(x) \equiv (x \leq 1000)$. Then $P(1000)$ holds. For $x \geq 1$, $P(x) \Rightarrow P(x-1)$ holds, since $x \leq 1000$ implies $x - 1 \leq 1000$. The hypothesis therefore holds, but $P(1001)$ is false, so $\forall x\,P(x)$ fails.
5. Option (D): take $P(x) \equiv (x \geq 1000)$. Then $P(1000)$ holds, and $P(x) \Rightarrow P(x+1)$ holds for every x. But $P(0)$ is false, so $\forall x\,P(x)$ fails.
6. Only (A) holds for every predicate P.

KEY IDEA
A statement about an arbitrary predicate is TRUE only if it holds for every predicate. Induction needs a base case at the start of the naturals (0) and a step that goes upward.

WATCH OUT FOR
Choosing (D) or (C) because a base case and a step are present. A base case at 1000 reaches only the values above 1000 with an upward step, and only the values below it with a downward step. A downward step from 0 reaches nothing new.

Q.16 MCQ · 1 MARK

Computer Networks › TCP

Consider the following statements:
(i) Address Resolution Protocol (ARP) provides a mapping from an IP address to the corresponding hardware (link-layer) address.
(ii) A single TCP segment from a sender S to a receiver R cannot carry both data from S to R and acknowledgement for a segment from R to S.

Which ONE of the following is CORRECT?

(A) Both (i) and (ii) are TRUE
(B) (i) is TRUE and (ii) is FALSE
(C) (i) is FALSE and (ii) is TRUE
(D) Both (i) and (ii) are FALSE

Official answer: **B**

SOLUTION

1. Statement (i): ARP takes a known IP address and finds the hardware (MAC, link-layer) address of the host on the same network. So (i) is TRUE.
2. Statement (ii): TCP is full duplex. Every TCP header has a sequence number, an acknowledgement number, an ACK flag and room for a payload.
3. A segment sent from R to S that carries R's own data can also set the ACK flag and acknowledge the data received from S. This is called piggybacking, so one segment can carry both data and an acknowledgement.
4. Statement (ii) says this cannot happen, so (ii) is FALSE.
5. (i) is TRUE and (ii) is FALSE, which is option (B).

KEY IDEA
ARP maps an IP address to a link-layer address. TCP piggybacks acknowledgements on data segments, because every TCP segment has an acknowledgement field.

WATCH OUT FOR
Thinking ARP maps a hardware address to an IP address (that is the job of RARP). Also thinking a TCP segment is either data or an acknowledgement, never both.

Q.17 MCQ · 1 MARK

Computer Networks › Routing

Consider the routing protocols given in List I and the names given in List II:

List I	List II
(i) Distance vector routing	(a) Bellman-Ford
(ii) Link state routing	(b) Dijkstra

For matching of items in List I with those in List II, which ONE of the following options is CORRECT?

(A) (i) – (a) and (ii) – (b)
(B) (i) – (a) and (ii) – (a)
(C) (i) – (b) and (ii) – (a)

(D) (i) – (b) and (ii) – (b)

Official answer: **A**

SOLUTION

1. Distance vector routing: each router keeps a vector of its estimated distances to every destination and updates it using the vectors received from its neighbours. This is the distributed form of the **Bellman-Ford** equation $d_x(y) = \min_v\{c(x,v) + d_v(y)\}$, where v runs over the neighbours of x. So (i) matches (a).
2. Link state routing: each router floods the costs of its own links, so every router builds the full network topology. Each router then runs **Dijkstra's** shortest-path algorithm on that graph. So (ii) matches (b).
3. The matching is (i) - (a) and (ii) - (b), which is option (A). Options (B), (C) and (D) each pair at least one protocol with the wrong algorithm.

KEY IDEA

Distance vector routing is the distributed Bellman-Ford algorithm, where routers exchange distance vectors with neighbours only. Link state routing floods link costs so every router knows the whole map, then runs Dijkstra locally.

WATCH OUT FOR

Swapping the pairs. Dijkstra needs the whole graph, which only link state provides. Distance vector routers see only their neighbours' vectors, which is exactly the situation Bellman-Ford handles.

Q.18 MCQ · 1 MARK

Computer Networks › IPv4

A machine receives an IPv4 datagram. The protocol field of the IPv4 header has the protocol number of a protocol X.

Which ONE of the following is NOT a possible candidate for X?

(A) Internet Control Message Protocol (ICMP)

(B) Internet Group Management Protocol (IGMP)

(C) Open Shortest Path First (OSPF)

(D) Routing Information Protocol (RIP)

Official answer: **D**

SOLUTION

1. The protocol field of an IPv4 header names the protocol carried directly in the datagram's payload. So X must be a protocol that is encapsulated straight inside IP, with no transport protocol in between.
2. ICMP is carried directly in IP datagrams and has protocol number 1, so it is a possible X.
3. IGMP is carried directly in IP datagrams and has protocol number 2, so it is a possible X.
4. OSPF is carried directly in IP datagrams and has protocol number 89. It uses no TCP or UDP, so it is a possible X.
5. RIP is an application-layer protocol that runs over UDP, on port 520. An IPv4 datagram that carries a RIP message has protocol number 17, which is UDP, in its protocol field. RIP has no IP protocol number of its own.
6. So RIP can never appear in the protocol field. The answer is (D).

KEY IDEA

The IPv4 protocol field names the protocol sitting directly above IP. ICMP (1), IGMP (2) and OSPF (89) sit directly on IP. RIP sits on UDP (port 520), so the field shows 17 (UDP) for it.

WATCH OUT FOR

Treating every routing protocol as a network-layer protocol that has its own IP protocol number. OSPF does, but RIP rides on UDP, and BGP rides on TCP.

Q.19 MCQ · 1 MARK

Programming in C › Pointers & arrays

Consider the following C program:

```
#include <stdio.h>
void stringcopy(char *, char *);
int main(){
   char a[30] = "@#Hello World!";
   stringcopy(a, a + 2);
   printf("%s\n", a);
   return 0;
}
void stringcopy(char *s, char *t) {
   while(*t)
      *s++ = *t++;
}
```

Which ONE of the following will be the output of the program?

(A) @#Hello World!

(B) Hello World!

(C) ello World!

(D) Hello World!d!

Official answer: **D**

SOLUTION

1. The array is initialised: a[0..14] = '@','#','H','e','l','l','o',' ','W','o','r','l','d','!','\0'.
2. stringcopy(a, a+2) sets s = &a[0] (destination, trailing) and t = &a[2] (source, leading). Since s is always exactly 2 positions behind t, the forward loop never overwrites a byte before t has already read it. The overlapping shift is safe.
3. Trace of while(*t) *s++ = *t++; (writing a[i] = original a[i+2] for i = 0..11): a[0]='H', a[1]='e', a[2]='l', a[3]='l', a[4]='o', a[5]=' ', a[6]='W', a[7]='o', a[8]='r', a[9]='l', a[10]='d', a[11]='!'.
4. Next check: *t at t = &a[14], whose ORIGINAL value is '\0'. The while-condition is tested before any assignment, so the loop exits immediately and s never writes to index 12.
5. Therefore a[12] and a[13] keep their pre-shift original values, 'd' and '!', and a[14] is still the original '\0'. None of these three positions were ever touched by the copy.
6. Full contents of a up to the terminator: a[0..11] = "Hello World!", followed by a[12]='d', a[13]='!', a[14]='\0'.
7. printf("%s\n", a) prints from a[0] until the '\0' at index 14, producing "Hello World!d!"; option (D).

KEY IDEA

`stringcopy(a, a+2)` performs a forward, overlapping left-shift of the string by 2 positions. Because the destination pointer `s` always trails the source pointer `t` by exactly 2 (i.e., `s` is always behind `t`), each byte is read by `t` before `s` ever reaches that same array position to overwrite it, so a forward `while(*t) *s++ = *t++;` loop is safe here (no data is clobbered before being read). The loop stops the instant `*t` is the ORIGINAL null terminator, so any bytes beyond where `s` has reached at that point retain their pre-shift values rather than being copied or zeroed.

WATCH OUT FOR

Assuming the loop runs one extra time and copies the null terminator (or copies through index 13), which would wipe out the leftover 'd' and '!' and give "Hello World!" (option B). The loop condition `while(*t)` is checked BEFORE each assignment, so as soon as `t` points at the original `'\0'` (index 14), the loop exits immediately without ever writing to `s` at that position, leaving `a[12]` and `a[13]` (originally `'d'` and `'!'`) untouched from before the shift. Another common slip is forgetting these are the *original* pre-shift values at those two positions, not garbage or copies of something else.

Q.20 MCQ · 1 MARK

Data Structures and Algorithms › Sorting & hashing

Consider an unordered list of N distinct integers.

What is the minimum number of element comparisons required to find an integer in the list that is NOT the largest in the list?

(A) 1

(B) $N - 1$

(C) N

(D) $2N - 1$

Official answer: **A**

SOLUTION

1. The task is only to certify that ONE element is not the overall largest, not to find the maximum, the minimum, sort the list, or identify the second-largest.
2. Pick any two distinct elements x and y from the list (valid since all N integers are distinct and N ≥ 2).
3. Compare x and y with a single comparison: exactly one of x < y or y < x holds.
4. The smaller of the two (say x, if x < y) is certified NOT to be the largest element in the whole list, because y is strictly greater than x.
5. Zero comparisons cannot work: with no information gathered, every element remains a possible candidate for being the largest, so no arbitrary choice can be guaranteed correct.
6. Hence 1 comparison is both sufficient and necessary: the minimum number of element comparisons required is 1, matching option A.

KEY IDEA

To show an element is not the largest, it is enough to find one element bigger than it: a single comparison between any two elements does that, and the smaller of the two is the answer.

WATCH OUT FOR

Confusing this with "find the maximum" (which needs $N - 1$ comparisons) or "find the second largest" (needs $N + \lceil \log_2 N \rceil - 2$ comparisons via a tournament). The question only asks for ANY element guaranteed not to be the largest, which is a much weaker requirement solvable in a single comparison.

Q.21 MCQ · 1 MARK

Compiler Design › Intermediate code

Consider the following statements about the use of backpatching in a compiler for intermediate code generation:

(I) Backpatching can be used to generate code for Boolean expression in one pass.
(II) Backpatching can be used to generate code for flow-of-control statements in one pass.
Which ONE of the following options is CORRECT?

(A) Only (I) is correct.

(B) Only (II) is correct.

(C) Both (I) and (II) are correct.

(D) Neither (I) nor (II) is correct.

Official answer: **C**

SOLUTION

1. Problem: in one-pass code generation, a jump to a label that is not yet known cannot be filled in when the jump is emitted.
2. Backpatching fixes this. Each jump with an unknown target is put on a list, using makelist(i) . The list is filled in later with backpatch(list, label) , once the target address is known.
3. Boolean expressions: for $E \to E_1 \vee M\, E_2$ the translation keeps $E.\text{truelist}$ and $E.\text{falselist}$. The marker M records the index of the next instruction. Then $\text{backpatch}(E_1.\text{falselist}, M.\text{instr})$, $E.\text{truelist} = \text{merge}(E_1.\text{truelist}, E_2.\text{truelist})$ and $E.\text{falselist} = E_2.\text{falselist}$. So statement (I) is true.
4. Flow-of-control: for $S \to$ if (B) M S_1 we do $\text{backpatch}(B.\text{truelist}, M.\text{instr})$ and $S.\text{nextlist} = \text{merge}(B.\text{falselist}, S_1.\text{nextlist})$. The same pattern handles if-else , while and statement sequences. So statement (II) is true.
5. Both statements hold, so the answer is (C).

KEY IDEA
Backpatching leaves jump targets blank, records them on truelist, falselist or nextlist, and fills them in once the target is known. This lets a single pass generate code for both Boolean expressions and flow-of-control statements.

WATCH OUT FOR
Thinking backpatching is only for flow-of-control statements, or only for Boolean expressions. It is used for both, and Boolean expressions are where truelist and falselist first appear.

Q.22 MCQ · 1 MARK

Compiler Design › Syntax-directed translation

Given the following syntax directed translation rules:

Rule 1: $R \to AB$ { $B.i = R.i - 1$; $A.i = B.i$; $R.i = A.i + 1$; }
Rule 2: $P \to CD$ { $P.i = C.i + D.i$; $D.i = C.i + 2$; }
Rule 3: $Q \to EF$ { $Q.i = E.i + F.i$; }

Which ONE is the CORRECT option among the following?

(A) Rule 1 is S-attributed and L-attributed; Rule 2 is S-attributed and not L-attributed; Rule 3 is neither S-attributed nor L-attributed

(B) Rule 1 is neither S-attributed nor L-attributed; Rule 2 is S-attributed and L-attributed; Rule 3 is S-attributed and L-attributed

(C) Rule 1 is neither S-attributed nor L-attributed; Rule 2 is not S-attributed and is L-attributed; Rule 3 is S-attributed and L-attributed

(D) Rule 1 is S-attributed and not L-attributed; Rule 2 is not S-attributed and is L-attributed; Rule 3 is S-attributed and L-attributed

Official answer: **C**

SOLUTION

1. An SDD is S-attributed if every attribute is synthesized, so each value depends only on attributes of the node's children. It is L-attributed if each inherited attribute of a right-hand-side symbol depends only on the inherited attributes of the parent and on attributes of symbols to its left in the production.
2. Rule 1, $R \to AB$: $B.i = R.i - 1$ defines an attribute of the right-hand-side symbol B, so $B.i$ is inherited and the rule is not S-attributed. Also $A.i = B.i$ defines the inherited $A.i$ from $B.i$, and B lies to the right of A. This violates the L-attributed condition. So Rule 1 is neither S-attributed nor L-attributed.
3. Rule 2, $P \to CD$: $P.i = C.i + D.i$ defines an attribute of the left-hand side from its children, so it is synthesized. $D.i = C.i + 2$ defines an attribute of the right-hand-side symbol D, so it is inherited. It depends only on C, which lies to the left of D, so the L-attributed condition holds. The rule is L-attributed but not S-attributed, because it has an inherited attribute.
4. Rule 3, $Q \to EF$: the only definition is $Q.i = E.i + F.i$, which is synthesized from the children. So Rule 3 is S-attributed. Every S-attributed definition is also L-attributed, because it has no inherited attributes to constrain. So Rule 3 is both.
5. Summary: Rule 1 is neither, Rule 2 is not S-attributed but is L-attributed, and Rule 3 is both S-attributed and L-attributed. This is option (C). Option (A) is wrong about Rule 1 and Rule 3. Option (B) is wrong about Rule 2. Option (D) is wrong about Rule 1.

KEY IDEA

Classify each attribute by where it is defined. If it sits on the left-hand side and is computed from the children, it is synthesized. If it sits on a right-hand-side symbol, it is inherited. S-attributed means no inherited attributes. L-attributed means every inherited attribute uses only the parent's inherited attributes and attributes of symbols to its left. S-attributed implies L-attributed.

WATCH OUT FOR

Treating $A.i = B.i$ as harmless because it looks like a simple copy. $A.i$ is inherited, and B is to the right of A, so the rule breaks the left-to-right dependency condition. Another slip is forgetting that a rule with any inherited attribute cannot be S-attributed, which wrongly makes Rule 2 S-attributed.

Q.23 MCQ · 1 MARK

Computer Networks › IPv4

Consider a network that uses Ethernet and IPv4. Assume that IPv4 headers do not use any options field. Each Ethernet frame can carry a maximum of 1500 bytes in its data field. A UDP segment is transmitted. The payload (data) in the UDP segment is 7488 bytes.

Which ONE of the following choices has the CORRECT total number of fragments transmitted and the size of the last fragment including IPv4 header?

(A) 5 fragments, 1488 bytes

(B) 6 fragments, 88 bytes

(C) 6 fragments, 108 bytes

(D) 6 fragments, 116 bytes

Official answer: **D**

SOLUTION

1. A UDP segment has an 8-byte header, so the IP datagram's data (the UDP segment) is $7488 + 8 = 7496$ bytes.

2. The Ethernet data field carries at most 1500 bytes, so the MTU is 1500 bytes. With a 20-byte IPv4 header (no options), each fragment can carry at most $1500 - 20 = 1480$ bytes of data.
3. Fragment data (except in the last fragment) must be a multiple of 8 bytes. Since $1480 = 185 \times 8$, a full fragment carries exactly 1480 bytes.
4. Number of fragments $= \lceil 7496/1480 \rceil = \lceil 5.065 \rceil = 6$.
5. The first five fragments carry $5 \times 1480 = 7400$ bytes of data.
6. The last fragment carries $7496 - 7400 = 96$ bytes of data.
7. Its size including the IPv4 header is $96 + 20 = 116$ bytes.
8. So there are 6 fragments and the last one is 116 bytes, which is option D.

KEY IDEA
The UDP header is part of the data that IP fragments. Each fragment holds the MTU minus the 20-byte IP header, rounded down to a multiple of 8 bytes. The last fragment holds the remainder plus a fresh 20-byte header.

WATCH OUT FOR
Forgetting the 8-byte UDP header and fragmenting only 7488 bytes. That gives a last fragment of 88 + 20 = 108 bytes (option C). Another slip is to divide by 1500 and forget that every fragment carries its own IP header.

Q.24 MCQ · 1 MARK

Theory of Computation › CFG & PDA

Which ONE of the following languages is accepted by a deterministic pushdown automaton?

(A) Any regular language.

(B) Any context-free language.

(C) Any language accepted by a non-deterministic pushdown automaton.

(D) Any decidable language.

Official answer: **A**

SOLUTION

1. A deterministic pushdown automaton (DPDA) accepts exactly the deterministic context-free languages (DCFLs). The question asks which option holds for every language in the class it names, so we test each class against the DCFLs.
2. (A) Any DFA is a DPDA that never touches its stack, so every regular language is accepted by a DPDA. This option is true for any regular language.
3. (B) The language $\{ww^R \mid w \in \{a, b\}^*\}$ is context-free, but a machine must guess where the middle of the input is, and a DPDA cannot do that. No DPDA accepts it, so not every context-free language is accepted by a DPDA.
4. (C) A language accepted by a non-deterministic pushdown automaton is exactly a context-free language. The same language ww^R is accepted by an NPDA but by no DPDA, so (C) fails.
5. (D) The language $\{a^n b^n c^n \mid n \geq 0\}$ is decidable but not context-free. No PDA of either kind accepts it, so a decidable language need not be accepted by a DPDA, and (D) fails.
6. The classes nest as regular $\subset$ DCFL $\subset$ CFL $\subset$ decidable. Only the regular languages are all inside the DCFLs, so the answer is (A).

KEY IDEA
The languages a DPDA accepts (DCFLs) lie strictly between the regular languages and the context-free languages. Only the regular languages are guaranteed to be inside the DCFLs.

WATCH OUT FOR
Assuming a DPDA is as powerful as an NPDA, and picking (B) or (C). Languages like ww^R need a guess of the midpoint, which only a non-deterministic machine can make.

Q.25 MCQ · 1 MARK

Theory of Computation › Turing machines

Let G_1, G_2 be Context Free Grammars (CFGs) and R be a regular expression. For a grammar G, let $L(G)$ denote the language generated by G.

Which ONE among the following questions is decidable?

(A) Is $L(G_1) = L(G_2)$?

(B) Is $L(G_1) \cap L(G_2) = \emptyset$?

(C) Is $L(G_1) = L(R)$?

(D) Is $L(G_1) = \emptyset$?

Official answer: **D**

SOLUTION

1. Recall the decidability results for context-free languages. Emptiness, finiteness and membership are decidable. Equivalence of two CFGs, universality ($L = \Sigma^*$) and emptiness of an intersection of two CFLs are undecidable.
2. (A) $L(G_1) = L(G_2)$ is the equivalence problem for CFGs, which is undecidable.
3. (B) $L(G_1) \cap L(G_2) = \emptyset$ is undecidable. Post's Correspondence Problem reduces to it: for a PCP instance one can build two CFGs whose languages intersect exactly when the instance has a solution.
4. (C) $L(G_1) = L(R)$ is undecidable. Take $R = \Sigma^*$. The question becomes whether $L(G_1) = \Sigma^*$, which is CFG universality, and that is undecidable. A problem that is undecidable for one choice of R is undecidable in general.
5. (D) $L(G_1) = \emptyset$ is decidable. Compute the set of generating variables. Start with the variables that have a production whose right side contains only terminals. Repeatedly add a variable that has a production whose right side contains only terminals and variables already in the set. The set grows at most once per variable, so this stops after finitely many rounds. Then $L(G_1) \neq \emptyset$ exactly when the start symbol S is in the set.
6. Only (D) is decidable, so the answer is (D).

KEY IDEA

Emptiness of a CFG is decidable by finding the generating variables and checking whether the start symbol is one of them. Equivalence, universality and intersection-emptiness for CFLs are undecidable.

WATCH OUT FOR

Treating (C) as decidable because R is regular. Regular-versus-CFL equality is still undecidable, because choosing $R = \Sigma^*$ gives CFG universality. Another slip is assuming (B) is decidable because the emptiness of one CFL is.

Q.26 MCQ · 1 MARK

Operating Systems › CPU scheduling

Processes $P1, P2, P3, P4$ arrive in that order at times 0, 1, 2, and 8 milliseconds respectively, and have execution times of 10, 13, 6, and 9 milliseconds respectively. Shortest Remaining Time First (SRTF) algorithm is used as the CPU scheduling policy. Ignore context switching times.

Which ONE of the following correctly gives the average turnaround time of the four processes in milliseconds?

(A) 22

(B) 15

(C) 37

(D) 19

Official answer: **D**

SOLUTION

1. Tabulate the processes:

Process	Arrival	Burst
P_1	0	10
P_2	1	13
P_3	2	6
P_4	8	9

   SRTF always runs the process with the least remaining time, and re-decides whenever a process arrives.
2. From $t = 0$ to 1 only P_1 is present and runs, so its remaining time falls to 9. At $t = 1$, P_2 arrives with 13, which is more than 9, so P_1 continues.
3. At $t = 2$, P_1 has 8 left and P_3 arrives with 6. Since $6 < 8$, P_3 preempts P_1.
4. P_3 runs from 2 to 8 and completes at $t = 8$. No other arrival happens before that, so it is not interrupted. P_4 arrives at $t = 8$.
5. At $t = 8$ the remaining times are $P_1 = 8$, $P_2 = 13$, $P_4 = 9$. P_1 is the shortest, so it runs from 8 to 16 and completes at 16. No new process arrives during this time.
6. Next, P_4 (9) is shorter than P_2 (13), so P_4 runs from 16 to 25 and then P_2 runs from 25 to 38.
7. Gantt chart:

$$P_1\,[0,2]\ P_3\,[2,8]\ P_1\,[8,16]\ P_4\,[16,25]\ P_2\,[25,38]$$

8. Turnaround time = completion time − arrival time: $P_1 = 16 - 0 = 16$, $P_2 = 38 - 1 = 37$, $P_3 = 8 - 2 = 6$, $P_4 = 25 - 8 = 17$.
9. Average $= \dfrac{16 + 37 + 6 + 17}{4} = \dfrac{76}{4} = 19$ ms, which is option (D).

KEY IDEA

SRTF is preemptive shortest-job-first: at every arrival and completion, run the process with the least remaining time. Turnaround time is completion time minus arrival time, not minus the start of execution.

WATCH OUT FOR

Treating the schedule as non-preemptive SJF, or using waiting time or completion time instead of turnaround time. Another slip is to let P_3 (6) wait for P_1 to finish, or to subtract 0 instead of the arrival time for P_2 and P_4.

Q.27 MSQ · 1 MARK

Database Management and Warehousing › Transactions

An audit of a banking transactions system has found that on an earlier occasion, two joint holders of account A attempted simultaneous transfers of Rs. 10000 each from account A to account B. Both transactions read the same value, Rs. 11000, as the initial balance in A and were allowed to go through. B was credited Rs. 10000 twice. A was debited only once and ended up with a balance of Rs. 1000.

Which of the following properties is/are certain to have been violated by the system?

(A) Atomicity

(B) Consistency

(C) Isolation

(D) Durability

Official answer: **B, C**

SOLUTION

1. Write the two transfers as T_1, T_2. Each one reads A, computes $A - 10000$, writes A, then adds 10000 to B.
2. Both read $A = 11000$. Each computes $11000 - 10000 = 1000$ and writes 1000. The second write overwrites the first, so one debit is lost. This is a lost update.
3. Isolation: the final state matches no serial order. In a serial run, the second transaction would read 1000 and could not debit 10000 again. So the two transactions were not isolated, and Isolation is certainly violated.
4. Consistency: a transfer must leave the total $A + B$ unchanged. Before, the total is $11000 + B_0$. After, it is $1000 + (B_0 + 20000) = 21000 + B_0$. So 10000 appeared from nothing, the database reached an invalid state, and Consistency is certainly violated.
5. Atomicity: nothing says either transaction ran only partly. Each may have executed all its steps, so Atomicity is not certain to be violated.
6. Durability: no committed update was lost after a failure. The lost debit came from concurrent overwriting, not from a crash, so Durability is not certain to be violated.
7. Hence the properties certain to be violated are Consistency (B) and Isolation (C).

KEY IDEA

A lost update from two interleaved read-modify-write transactions breaks Isolation (the result matches no serial order). It also breaks the transfer's invariant that the total across A and B stays the same, which is Consistency.

WATCH OUT FOR

Calling this an Atomicity or Durability failure. Both need a partial execution or a crash, and the question gives neither. Only the two violations that follow from the stated balances are certain.

Q.28 MSQ · 1 MARK

Computer Organization and Architecture › Instructions & addressing

Which of the following is/are part of an Instruction Set Architecture of a processor?

(A) The size of the cache memory

(B) The clock frequency of the processor

(C) The number of cache memory levels

(D) The total number of registers

Official answer: **D**

SOLUTION

1. ISA (Instruction Set Architecture) is the abstract interface between hardware and software that is visible to the programmer/compiler: the instruction set, the addressing modes, and the architecturally visible register set (how many registers exist and how they are named/encoded).
2. Cache memory size (A) and the number of cache levels (C) are microarchitectural implementation choices, not part of the ISA. The ISA does not expose or specify cache organization at all, and two processors implementing the identical ISA (e.g., different processors in the x86 family) can have completely different cache hierarchies.
3. Clock frequency (B) is a physical/implementation parameter of a specific chip, not part of the ISA: different speed grades of the same processor, all implementing the same ISA, run at different clock frequencies.

4. The total number of (architectural) registers (D) is fixed by the ISA specification itself, since machine instructions must reference these registers by a fixed-width encoding/name defined in the ISA (e.g., x86-64 defines 16 general-purpose registers, MIPS defines 32). This is a software-visible property, unlike microarchitectural resources such as physical register files or reorder buffers.
5. Hence only option D is part of the Instruction Set Architecture; A, B, and C are microarchitecture/implementation details.

KEY IDEA

The Instruction Set Architecture (ISA) is the programmer/compiler-visible contract between hardware and software: the instruction set, addressing modes, and the architecturally visible register set (how many registers a program can name and how they're encoded). Everything about *how* that contract is physically realized (cache size, number of cache levels, clock frequency) is microarchitecture/implementation, and can differ across chips that implement the identical ISA.

WATCH OUT FOR

Treating any processor specification listed in a datasheet (clock speed, cache size, cache levels) as part of the "architecture," when ISA specifically means the software-visible contract, not the hardware implementation details that vary between different chips sharing the same ISA.

Q.29 MSQ · 1 MARK

Data Structures and Algorithms › Graph algorithms

Which of the following statements regarding Breadth First Search (BFS) and Depth First Search (DFS) on an undirected simple graph G is/are TRUE?

(A) A DFS tree of G is a Shortest Path tree of G.

(B) Every non-tree edge of G with respect to a DFS tree is a forward/back edge.

(C) If (u, v) is a non-tree edge of G with respect to a BFS tree, then the distances from the source vertex s to u and v in the BFS tree are within ± 1 of each other.

(D) Both BFS and DFS can be used to find the connected components of G.

Official answer: **B, C, D**

SOLUTION

1. Statement (A): A DFS tree records the order/structure of recursive exploration (which branch was visited first), not edge counts or weights. There is no guarantee that the tree-path distance from the root to any vertex in a DFS tree equals the shortest-path (BFS) distance in G; a DFS can plunge deep along one branch before backtracking to a vertex that is actually very close to the root. Hence (A) is FALSE.
2. Statement (B): For DFS on an UNDIRECTED graph, every edge of G is classified as either a tree edge or a back edge: forward edges and cross edges, which can occur in directed-graph DFS, cannot occur here. This is because an undirected edge (x, y) is explored from whichever endpoint DFS reaches first; by the time the other endpoint is visited, the edge is already either a tree edge or connects back to an ancestor still on the recursion stack (a back edge). So every non-tree edge is a back edge, which is one of the two cases named in statement (B) ('forward/back edge'). Hence (B) is TRUE.
3. Statement (C): This is the standard BFS-tree lemma. BFS assigns each vertex a level equal to its shortest-path distance from source s, discovering all level-k vertices before any level-$(k+1)$ vertex. If (u, v) is a non-tree edge found while scanning u's neighbours, v must already be at level $\text{level}(u) - 1$, $\text{level}(u)$, or $\text{level}(u) + 1$; it cannot be two or more levels away, because BFS would then have discovered v (or u) earlier through that very edge, contradicting how BFS proceeds strictly level by level. Hence the distances from s to u and v differ by at most 1, so (C) is TRUE.

4. Statement (D): Running BFS (or equivalently DFS) from any unvisited vertex visits exactly the set of vertices reachable from it, which in an undirected graph is precisely its connected component. Repeating this process (pick an unvisited vertex, traverse, mark its whole component visited) until every vertex has been visited enumerates all connected components. This works identically whether BFS or DFS is used as the traversal. Hence (D) is TRUE.
5. Combining all four: the TRUE statements are (B), (C), (D), and (A) is FALSE.

KEY IDEA
In an undirected graph, DFS classifies every non-tree edge as a back edge (never forward or cross), and BFS levels correspond exactly to shortest-path distances from the source, so any non-tree edge in a BFS tree can only connect vertices at levels differing by at most 1. Both traversals visit exactly the reachable set from a start vertex, so repeating from each unvisited vertex finds all connected components.

WATCH OUT FOR
Assuming forward and cross edges can occur in a DFS of an undirected graph (as they can in a directed graph), which leads students to wrongly mark (B) as false; also confusing a DFS tree (built for reachability, not distance) with a BFS tree when judging shortest-path claims.

Q.30 MSQ · 1 MARK

Theory of Computation › Context-free languages

Consider the two lists List I and List II given below:

List I	List II
(i) Context free languages	(a) Closed under union
(ii) Recursive languages	(b) Not closed under complementation
(iii) Regular languages	(c) Closed under intersection

For matching of items in List I with those in List II, which of the following option(s) is/are CORRECT?

(A) (i) – (a), (ii) – (b), and (iii) – (c)

(B) (i) – (b), (ii) – (a), and (iii) – (c)

(C) (i) – (b), (ii) – (c), and (iii) – (a)

(D) (i) – (a), (ii) – (c), and (iii) – (b)

Official answer: **B, C**

SOLUTION

1. Closure facts. Regular languages are closed under union, intersection and complementation. Recursive languages are closed under union, intersection and complementation. Context-free languages are closed under union, but not under intersection or complementation.
2. Decide which List II property is true for each class. Context-free languages: (a) true, (b) true, (c) false. Recursive languages: (a) true, (b) false, (c) true. Regular languages: (a) true, (b) false, (c) true.
3. An option is correct only if all three of its pairings are true statements. Option A pairs (ii) with (b), but recursive languages are closed under complementation, so (ii)-(b) is false. A is incorrect.
4. Option B: (i)-(b) is true, because CFLs are not closed under complementation. (ii)-(a) is true, because recursive languages are closed under union. (iii)-(c) is true, because regular languages are closed under intersection. B is correct.
5. Option C: (i)-(b) is true. (ii)-(c) is true, because recursive languages are closed under intersection. (iii)-(a) is true, because regular languages are closed under union. C is correct.
6. Option D pairs (iii) with (b), but regular languages are closed under complementation, so (iii)-(b) is false. D is incorrect.
7. Hence the correct options are B and C.

KEY IDEA

Test each pairing as a true or false statement about closure properties. Only an option whose three pairings are all true is correct. Several options can qualify because (a) and (c) both hold for recursive and regular languages.

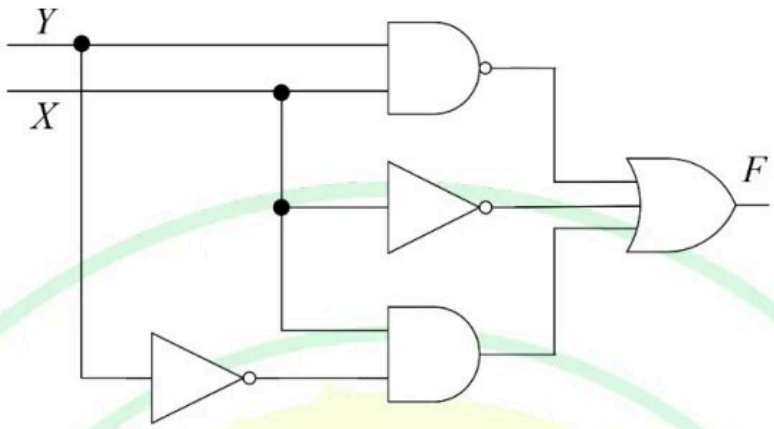
WATCH OUT FOR

Treating the question as a one-to-one matching with a single right answer. Another slip is forgetting that recursive languages are closed under complementation, or that CFLs are closed under union but not under intersection or complementation.

Q.31 MSQ · 1 MARK

Digital Electronics › Boolean logic

Q.31 Consider the following logic circuit diagram.



Which is/are the CORRECT option(s) for the output function F?

(A)	$\overline{X\,Y}$
(B)	$\bar{X} + \bar{Y} + X\bar{Y}$
(C)	$\overline{X\,Y} + \bar{X} + X\bar{Y}$
(D)	$X + \bar{Y}$

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

SOLUTION

1. Trace the circuit. The top gate is a NAND with inputs X and Y, so its output is $\overline{XY}$.
2. The middle gate is an inverter on X, so its output is $\overline{X}$.
3. In the bottom path, Y goes through an inverter to give $\overline{Y}$. An AND gate combines this with X, so its output is $X\overline{Y}$.
4. The three outputs feed a 3-input OR gate, so $F = \overline{XY} + \overline{X} + X\overline{Y}$. This is exactly option (C).
5. Apply De Morgan's law, $\overline{XY} = \overline{X} + \overline{Y}$. Then $F = \overline{X} + \overline{Y} + X\overline{Y}$, which is option (B).
6. Absorb the last term: $\overline{Y} + X\overline{Y} = \overline{Y}(1 + X) = \overline{Y}$. So $F = \overline{X} + \overline{Y} = \overline{XY}$, which is option (A).
7. Check with a truth table:

X	Y	$\overline{XY}+\overline{X}+X\overline{Y}$	$X+\overline{Y}$
0	0	1	1
0	1	1	0
1	0	1	1
1	1	0	1

The F column is $1, 1, 1, 0$, which matches $\overline{XY}$ exactly.

8. Option (D) differs from F at $X = 0, Y = 1$ (it gives 0, F gives 1) and at $X = 1, Y = 1$ (it gives 1, F gives 0). So (D) is wrong. Options (A), (B) and (C) are all equal to F.

KEY IDEA

Read the gate expressions off the circuit, then simplify with De Morgan's law and the absorption identity $\overline{Y} + X\overline{Y} = \overline{Y}$. All three correct options are different forms of the same function, $\overline{XY}$.

WATCH OUT FOR

Stopping after finding one option equal to F and not checking the others. In an MSQ, every algebraically equivalent form is correct, including the raw circuit expression in (C) and its De Morgan form in (B). Option (D) looks similar but fails the truth table.

Q.32 NAT · 1 MARK

Computer Organization and Architecture › ALU & control unit

The following two signed 2's complement numbers (multiplicand M and multiplier Q) are being multiplied using Booth's algorithm:

M: 1100 1101 1110 1101 and Q: 1010 0100 1010 1010

The total number of addition and subtraction operations to be performed is _____. (Answer in integer)

Official answer: **13**

SOLUTION

1. Booth's algorithm runs for $n = 16$ cycles (Q is 16 bits). At each cycle it inspects the pair (Q_0, Q_{-1}), where Q_{-1} starts at 0: pair 01 adds M to the partial product, pair 10 subtracts M, and pair 00 or 11 does no arithmetic (only an arithmetic right shift).
2. Since Q_{-1} at cycle k is just the original bit Q_{k-1} (shifted in from the previous cycle), the pair examined at cycle k is (Q_k, Q_{k-1}) for $k = 0, \ldots, 15$, with $Q_{-1} = 0$. So the number of add/subtract operations equals the number of $0 \to 1$ or $1 \to 0$ transitions scanning Q's bits from LSB to MSB with an extra leading $Q_{-1} = 0$. M is irrelevant to this count.
3. Q = 1010 0100 1010 1010 (MSB to LSB), so reading off individual bits: $Q_{15} = 1, Q_{14} = 0, Q_{13} = 1, Q_{12} = 0, Q_{11} = 0, Q_{10} = 1, Q_9 = 0, Q_8 = 0, Q_7 = 1, Q_6 = 0, Q_5 = 1, Q_4 = 0, Q_3 = 1, Q_2 = 0, Q_1 = 1, Q_0 = 0$.
4. Prefixing $Q_{-1} = 0$ gives the sequence $Q_{-1}, Q_0, Q_1, \ldots, Q_{15} = 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1$.
5. Comparing each adjacent pair (Q_i, Q_{i-1}) for $i = 0$ to 15: i=0:(0,0) same; i=1:(1,0) diff; i=2:(0,1) diff; i=3:(1,0) diff; i=4:(0,1) diff; i=5:(1,0) diff; i=6:(0,1) diff; i=7:(1,0) diff; i=8:(0,1) diff; i=9:(0,0) same; i=10:(1,0) diff; i=11:(0,1) diff; i=12:(0,0) same; i=13:(1,0) diff; i=14:(0,1) diff; i=15:(1,0) diff.
6. Same-pairs occur at $i = 0, 9, 12$ (3 positions, pure shifts, no arithmetic); every other position ($i = 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 13, 14, 15$) is a diff-pair, giving $16 - 3 = 13$ transitions.
7. Total addition + subtraction operations $= 13$.

KEY IDEA

In Booth's algorithm, at each of the n cycles the ALU performs add M, subtract M, or nothing, based on the pair (Q_0, Q_{-1}) (with Q_{-1} starting at 0): pair 01 adds M, pair 10 subtracts M, and 00 or 11 do nothing except an arithmetic shift. Because Q_{-1} at cycle k is just the original bit Q_{k-1} (shifted in from the previous cycle), the total add/subtract count equals the number of bit-to-bit transitions ($0 \to 1$ or $1 \to 0$) when scanning the multiplier Q from its LSB toward its MSB, with an extra leading $Q_{-1} = 0$ prepended, independent of the multiplicand M.

WATCH OUT FOR

Assuming the number of add/subtract operations depends on the multiplicand M, or forgetting to prepend the initial $Q_{-1} = 0$ bit: omitting it undercounts the transition at $i = 0$ and gives 12 instead of 13.

Q.33 NAT · 1 MARK

Programming in C › C basics

```
 int x=126,y=105;
do {
    if(x>y) x=x-y;
    else y=y-x;
} while(x!=y);

printf("%d",x);
```

The output of the given C code segment is _____. (Answer in integer)

Official answer: **21**

SOLUTION

1. Since it is a do-while loop, the body runs once before the condition x!=y is checked.
2. Initial: x=126, y=105.
3. Iteration 1: x>y (126>105) is true, so x=x-y=126-105=21. Now x=21, y=105. Check x!=y: 21!=105 is true, so the loop continues.
4. Iteration 2: x>y (21>105) is false, so y=y-x=105-21=84. Now x=21, y=84. Check x!=y: true, continue.
5. Iteration 3: x>y (21>84) is false, so y=y-x=84-21=63. Now x=21, y=63. Check x!=y: true, continue.
6. Iteration 4: x>y (21>63) is false, so y=y-x=63-21=42. Now x=21, y=42. Check x!=y: true, continue.
7. Iteration 5: x>y (21>42) is false, so y=y-x=42-21=21. Now x=21, y=21. Check x!=y: 21!=21 is false, so the loop terminates.
8. The loop is the subtraction-based Euclidean algorithm, so the terminal value of x equals gcd(126,105).
9. gcd(126,105): $126 = 2 \cdot 3^2 \cdot 7$ and $105 = 3 \cdot 5 \cdot 7$, so gcd$= 3 \cdot 7 = 21$, matching the trace.
10. printf("%d",x) prints x=21, which is the required output.

KEY IDEA

This do-while loop is the subtraction-based (slow) Euclidean algorithm: it repeatedly replaces the larger of x, y by their difference until x equals y, at which point both variables hold gcd(x, y).

WATCH OUT FOR

Treating it as a while loop and skipping the first pass's guaranteed subtraction, or stopping the trace as soon as x first becomes 21 (after iteration 1) without continuing until x!=y actually turns false; x stays 21 throughout, but y still needs four more iterations to reach 21.

Q.34 NAT · 1 MARK

Digital Electronics › Sequential

In a 4-bit ripple counter, if the period of the waveform at the last flip-flop is 64 microseconds, then the frequency of the ripple counter in kHz is _____. (Answer in integer)

Official answer: **250**

SOLUTION

1. Each flip-flop in a ripple counter is a toggle stage: it changes state once per clock period of its own input, so its output period is twice its input period (frequency halves).
2. A 4-bit ripple counter has 4 such stages in cascade, so the last flip-flop's output frequency is $f_{\text{last}} = \frac{f_{\text{clk}}}{2^4} = \frac{f_{\text{clk}}}{16}$.
3. In terms of period, $T_{\text{last}} = 16\,T_{\text{clk}}$.
4. Substitute $T_{\text{last}} = 64\ \mu\text{s}$: $T_{\text{clk}} = \frac{64}{16} = 4\ \mu\text{s}$.
5. The counter's operating (input clock) frequency is $f = \frac{1}{4\ \mu\text{s}} = 250\ \text{kHz}$.

KEY IDEA

Every flip-flop in a ripple counter divides the frequency by 2, so an n-bit counter's last output has period 2^n times the clock period.

WATCH OUT FOR

Dividing by 4 (the number of bits) instead of $2^4 = 16$, or forgetting to convert microseconds to kHz: $1/(4\ \mu\text{s}) = 250$ kHz, not 250 MHz.

Q.35 NAT · 1 MARK

Data Structures and Algorithms › Trees & heaps

Suppose the values $10, -4, 15, 30, 20, 5, 60, 19$ are inserted in that order into an initially empty binary search tree. Let T be the resulting binary search tree. The number of edges in the path from the node containing 19 to the root node of T is _____. (Answer in integer)

Official answer: **4**

SOLUTION

1. Insert 10 as the root.
2. Insert -4: compare with 10 (-4<10) → becomes left child of 10.
3. Insert 15: compare with 10 (15>10) → becomes right child of 10.
4. Insert 30: compare with 10 (30>10, go right to 15), then with 15 (30>15) → becomes right child of 15.
5. Insert 20: compare with 10 (go right to 15), with 15 (go right to 30), with 30 (20<30) → becomes left child of 30.
6. Insert 5: compare with 10 (5<10, go left to -4), with -4 (5>-4) → becomes right child of -4.
7. Insert 60: compare with 10 (go right to 15), with 15 (go right to 30), with 30 (60>30) → becomes right child of 30.
8. Insert 19: compare with 10 (go right to 15), with 15 (go right to 30), with 30 (19<30, go left to 20), with 20 (19<20) → becomes left child of 20.
9. Path from 19 to the root: 19 → 20 → 30 → 15 → 10. This path has 4 edges.

KEY IDEA
Standard BST insertion: each new key is compared against nodes starting from the root, going left if smaller and right if larger, until an empty spot is found; the depth of a node equals the number of edges on its unique root path.

WATCH OUT FOR
Counting the number of nodes on the path (5) instead of the number of edges (4), or inserting 5 relative to 10 instead of continuing the BST search down to -4's subtree (5 must be compared against -4, since 5 < 10 sends it left first).

Q.36 MCQ · 2 MARKS

Computer Networks › Performance

Suppose we are transmitting frames between two nodes using Stop-and-Wait protocol. The frame size is 3000 bits. The transmission rate of the channel is 2000 bps (bits/second) and the propagation delay between the two nodes is 100 milliseconds. Assume that the processing times at the source and destination are negligible. Also, assume that the size of the acknowledgement packet is negligible. Which ONE of the following most accurately gives the channel utilization for the above scenario in percentage?

(A) 88.23

(B) 93.75

(C) 85.44

(D) 66.67

Official answer: **A**

SOLUTION

1. Transmission time of one frame: $T_t = \dfrac{3000 \text{ bits}}{2000 \text{ bps}} = 1.5 \text{ s}$.
2. Propagation delay: $T_p = 100 \text{ ms} = 0.1 \text{ s}$. The acknowledgement is negligible in size, so its transmission time is 0, and processing times are negligible.
3. In Stop-and-Wait the sender sends one frame, then waits for the acknowledgement. One cycle takes $T_t + 2T_p = 1.5 + 0.2 = 1.7 \text{ s}$.
4. Utilization $U = \dfrac{T_t}{T_t + 2T_p} = \dfrac{1.5}{1.7} = 0.88235\ldots$
5. As a percentage this is about 88.2353%. The closest option is 88.23, which is option (A).

KEY IDEA
In Stop-and-Wait the channel is busy only for the frame's transmission time T_t in each cycle of $T_t + 2T_p$ (with a negligible acknowledgement), so $U = T_t/(T_t + 2T_p)$.

WATCH OUT FOR
Using $T_t + T_p$ as the cycle time, or forgetting that the acknowledgement must travel back, which gives $1.5/1.6 = 93.75\%$ (option B). Another slip is using the window-size formula $1/(1 + 2a)$ with a computed from mismatched units.

Q.37 MCQ · 2 MARKS

Data Structures and Algorithms › Graph algorithms

Let G be an edge-weighted undirected graph with positive edge weights. Suppose a positive constant α is added to the weight of every edge.

Which ONE of the following statements is TRUE about the minimum spanning trees (MSTs) and shortest paths (SPs) in G before and after the edge weight update?

(A) Every MST remains an MST, and every SP remains an SP.

(B) MSTs need not remain MSTs, and every SP remains an SP.

(C) Every MST remains an MST, and SPs need not remain SPs.

(D) MSTs need not remain MSTs, and SPs need not remain SPs.

Official answer: **C**

SOLUTION

1. Since G has n vertices, every spanning tree of G has exactly $n-1$ edges (a basic property of trees); this holds for EVERY spanning tree, not just the MST.
2. After adding α to every edge weight, the new total weight of any spanning tree T is $\text{weight}_{\text{new}}(T) = \text{weight}_{\text{old}}(T) + \alpha(n-1)$, because T contributes exactly $n-1$ edges to the sum, each of which increased by α.
3. The shift $\alpha(n-1)$ is the SAME constant for every spanning tree of G: it depends only on n and α, never on which particular tree is chosen.
4. Adding an identical constant to every value in a set cannot change which value is the minimum. So whichever spanning tree(s) had minimum weight before the update still have minimum weight after it: every MST remains an MST.
5. Now consider shortest paths between two vertices u and v. Unlike spanning trees, different candidate paths from u to v can use different numbers of edges; say one path uses k_1 edges and another uses $k_2 \neq k_1$ edges.
6. After adding α to every edge, a path using k edges gains total weight $k\alpha$. Since $k_1 \neq k_2$ in general, the two candidate paths are shifted by DIFFERENT amounts, not the same amount as in the spanning-tree case.
7. This unequal shift can reverse the relative order: a path that was shortest before the update (possibly using more edges but a smaller original total weight) can end up longer than a path using fewer edges once α is added to every edge; a standard counterexample exists for any such setup.
8. Hence shortest paths need not remain shortest paths after the update, while every MST is guaranteed to remain an MST.
9. This matches option (C): Every MST remains an MST, and SPs need not remain SPs.

KEY IDEA

Every spanning tree of a connected graph on n vertices has exactly $n-1$ edges, so adding a constant α to every edge shifts every spanning tree's total weight by the SAME fixed amount $\alpha(n-1)$; this cannot change which tree has minimum weight. But different simple paths between two vertices can use different numbers of edges, so the same shift adds different amounts to different candidate paths, which can flip which path is shortest.

WATCH OUT FOR

Assuming that because MSTs are preserved under a uniform additive shift, shortest paths must be too; students forget that different u-v paths can use different numbers of edges, so the shift is NOT applied equally to all candidate paths (unlike spanning trees, which always have exactly n−1 edges regardless of which tree is chosen).

Q.38 MCQ · 2 MARKS

Data Structures and Algorithms › Complexity

A meld operation on two instances of a data structure combines them into one single instance of the same data structure. Consider the following data structures:

P: Unsorted doubly linked list with pointers to the head node and tail node of the list.

Q: Min-heap implemented using an array.

R: Binary Search Tree.

Which ONE of the following options gives the worst-case time complexities for meld operation on instances of size n of these data structures?

(A) P: $\Theta(1)$, Q: $\Theta(n)$, R: $\Theta(n)$

(B) P: $\Theta(1)$, Q: $\Theta(n \log n)$, R: $\Theta(n)$

(C) P: $\Theta(n)$, Q: $\Theta(n \log n)$, R: $\Theta(n^2)$

(D) P: $\Theta(1)$, Q: $\Theta(n)$, R: $\Theta(n \log n)$

Official answer: **A**

SOLUTION

1. P: Unsorted doubly linked list with head and tail pointers. Melding two such lists means splicing the tail of one list to the head of the other (updating three or four pointers) and setting the new tail pointer. This is a fixed number of pointer operations regardless of list size, so P is $\Theta(1)$.
2. Q: Min-heap implemented as an array. Repeated insertion of one heap's n elements into the other would cost $\Theta(n \log n)$, but that is not optimal. Instead, concatenate the two underlying arrays into one array of size $2n$ (a $\Theta(n)$ copy) and then run the standard bottom-up build-heap (heapify) procedure on the combined array. Build-heap on an arbitrary array of size m runs in $\Theta(m)$ time (the classical linear-time heap-construction result), so the whole meld is $\Theta(n)$, not $\Theta(n \log n)$.
3. R: Binary Search Tree with no balance guarantee. The worst-case-optimal way to meld two arbitrary BSTs of size n is: an in-order traversal of each tree to get its sorted key sequence ($\Theta(n)$ each), a linear merge of the two sorted sequences ($\Theta(n)$), and then building a fresh balanced BST from the merged sorted sequence by repeatedly choosing the middle element as root ($\Theta(n)$). Every step is linear, so the total worst-case cost is $\Theta(n)$.
4. Matching P: $\Theta(1)$, Q: $\Theta(n)$, R: $\Theta(n)$ to the options identifies option (A) as correct. Option (B) wrongly assumes heap melding needs $\Theta(n \log n)$ (that is the cost of repeated insertion, not the optimal build-heap approach); options (C) and (D) overstate R's or Q's/P's complexity.

KEY IDEA

Melding two structures optimally means using each structure's cheapest correct rebuild strategy: pointer splicing for the linked list (Θ(1)), linear-time array concatenation + bottom-up build-heap for the heap (Θ(n), not the Θ(n log n) of repeated insertion), and in-order flatten + merge + balanced rebuild for the BST (Θ(n) total).

WATCH OUT FOR

Assuming meld of two heaps costs Θ(n log n) by imagining n repeated insertions into one heap, instead of recognizing that concatenating the arrays and running the linear-time (Floyd's) build-heap procedure achieves Θ(n).

Q.39 MCQ · 2 MARKS

Computer Organization and Architecture › Memory & cache

For a direct-mapped cache, 4 bits are used for the tag field and 12 bits are used to index into a cache block. The size of each cache block is one byte. Assume that there is no other information stored for each cache block.

Which ONE of the following is the CORRECT option for the sizes of the main memory and the cache memory in this system (byte addressable), respectively?

(A) 64 KB and 4 KB

(B) 128 KB and 16 KB

(C) 64 KB and 8 KB

(D) 128 KB and 6 KB

Official answer: **A**

SOLUTION

1. Address structure for a direct-mapped, byte-addressable cache: total address bits = tag bits + index bits + block-offset bits.
2. Block size = 1 byte ⇒ block-offset bits = $\log_2(1) = 0$.

3. Total address bits = $4 + 12 + 0 = 16$ bits.
4. Main memory size = 2^{16} bytes $= 65536$ bytes $= 64$ KB.
5. Number of cache lines (blocks) = $2^{\text{index bits}} = 2^{12} = 4096$.
6. Cache memory size = (number of lines) × (block size) = 4096×1 byte $= 4$ KB (no extra tag/valid-bit storage is counted, per the question's assumption).
7. Main memory = 64 KB and Cache memory = 4 KB, which is option (A).

KEY IDEA
In a direct-mapped, byte-addressable cache, the memory address splits into Tag + Index + Block-offset bits. Main memory size = $2^{\text{total address bits}}$ bytes; number of cache lines = $2^{\text{index bits}}$; cache size = (number of lines) × (block size).

WATCH OUT FOR
Forgetting that a block-offset field needs $\log_2(\text{block size})$ bits and wrongly adding an extra offset bit even when block size is 1 byte (offset = 0 bits), or confusing "index bits" with "number of blocks" incorrectly when computing cache size.

Q.40 MCQ · 2 MARKS

Compiler Design › Parsing

Given a Context-Free Grammar G as follows:

$S \to Aa \mid bAc \mid dc \mid bda$

$A \to d$

Which ONE of the following statements is TRUE?

(A) G is neither LALR(1) nor SLR(1)

(B) G is CLR(1), not LALR(1)

(C) G is LALR(1), not SLR(1)

(D) G is LALR(1), also SLR(1)

Official answer: **C**

SOLUTION

1. Compute the follow set: A occurs in $S \to Aa$ and $S \to bAc$, so $\text{FOLLOW}(A) = \{a, c\}$.
2. Start state I_0 contains $S \to \cdot Aa$, $S \to \cdot bAc$, $S \to \cdot dc$, $S \to \cdot bda$ and $A \to \cdot d$. On d it moves to $I_d = \{A \to d\cdot,\ S \to d \cdot c\}$.
3. SLR(1) test on I_d: the reduce $A \to d$ is placed on every symbol of $\text{FOLLOW}(A) = \{a, c\}$, and the shift is on c . The entry for c holds both a reduce and a shift, a shift-reduce conflict, so G is not SLR(1). (The state reached on b then d has the same problem on a .)
4. CLR(1) lookaheads in I_d: this $A \to d\cdot$ comes only from $S \to \cdot Aa$, so its lookahead is $\{a\}$. The reduce is on a and the shift is on c , so there is no conflict.
5. Reaching d after b gives $I_{bd} = \{A \to d\cdot,\ S \to bd \cdot a\}$. Here $A \to d\cdot$ comes from $S \to b \cdot Ac$, so its lookahead is $\{c\}$, and the shift is on a . There is no conflict, so G is CLR(1).
6. LALR(1) merges only CLR(1) states that have identical LR(0) cores. I_d has core $\{A \to d\cdot,\ S \to d \cdot c\}$ and I_{bd} has core $\{A \to d\cdot,\ S \to bd \cdot a\}$. The cores differ, so no states merge and the LALR(1) table equals the CLR(1) table, which is conflict-free.
7. Hence G is LALR(1) (and CLR(1)) but not SLR(1). Option (B) is false because G is LALR(1), (A) and (D) are false for the same reason or because G is not SLR(1). The answer is (C).

KEY IDEA

SLR(1) reduces on the whole FOLLOW set, while CLR(1) and LALR(1) use the exact lookahead of each item. The two states containing $A \to d\cdot$ have different LR(0) cores, so LALR(1) merges nothing and keeps the CLR(1) table, which has no conflict.

WATCH OUT FOR

Assuming that the two states holding $A \to d\cdot$ merge in LALR(1) and create a reduce-reduce or shift-reduce conflict. Their cores differ ($S \to d \cdot c$ against $S \to bd \cdot a$), so they stay separate. Another slip is using FOLLOW(A) = {a, c} as the lookahead, which makes it look SLR-conflict-free or CLR-conflicted.

Q.41 MCQ · 2 MARKS

Data Structures and Algorithms › Sorting & hashing

An array A of length n with distinct elements is said to be bitonic if there is an index $1 \leq i \leq n$ such that $A[1..i]$ is sorted in the non-decreasing order and $A[i+1..n]$ is sorted in the non-increasing order.

Which ONE of the following represents the best possible asymptotic bound for the worst-case number of comparisons by an algorithm that searches for an element in a bitonic array A?

(A) $\Theta(n)$

(B) $\Theta(1)$

(C) $\Theta(\log^2 n)$

(D) $\Theta(\log n)$

Official answer: **D**

SOLUTION

1. A bitonic array rises (non-decreasing) up to some peak index i and then falls (non-increasing) after it; since all elements are distinct, both the rising part $A[1..i]$ and the falling part $A[i+1..n]$ are strictly monotonic.
2. Stage 1 (locate the peak in $O(\log n)$ comparisons): at a probe index m, compare $A[m]$ with its neighbours. If $A[m-1] < A[m] < A[m+1]$ the peak lies to the right; if $A[m-1] > A[m] > A[m+1]$ it lies to the left; otherwise $A[m]$ itself is the peak. Each such probe (a constant number of comparisons) discards half the remaining range, exactly like binary search, so the peak index i is found using $O(\log n)$ comparisons.
3. Stage 2 (search each monotonic half): once i is known, $A[1..i]$ is sorted ascending and $A[i+1..n]$ is sorted descending. An ordinary binary search for the target on $A[1..i]$ costs $O(\log n)$ comparisons, and a binary search (with the inequality direction reversed) on $A[i+1..n]$ also costs $O(\log n)$ comparisons. In the worst case the target can require probing both halves, but that only adds two $O(\log n)$ terms.
4. Total worst-case cost: $O(\log n)$ (find peak) $+ O(\log n)$ (binary search the halves) $= O(\log n)$.
5. This bound is also the best possible one: searching for an arbitrary target among n distinct candidate values, using only comparisons, is a decision problem with n (or $n+1$, including "not found") possible outcomes, so any comparison-based search algorithm needs $\Omega(\log n)$ comparisons in the worst case, on any array, bitonic or otherwise. Combining the matching upper and lower bounds gives a worst-case comparison count of $\Theta(\log n)$.
6. This rules out the other options: $\Theta(n)$ is what a naive linear scan gives, far more than necessary; $\Theta(1)$ is impossible because the target can be any one of n values and the algorithm must in general inspect enough elements to distinguish among them; $\Theta(\log^2 n)$ is achievable (e.g. binary searching both halves without first locating the peak) but is not the *best possible* bound, since the two-stage algorithm above already achieves the smaller $\Theta(\log n)$. Hence the correct choice is option D.

KEY IDEA
A bitonic array can be searched in two binary-search-style stages: first locate the peak (the point where the array switches from ascending to descending), then binary-search whichever monotonic half is needed, giving $O(\log n)$ comparisons; since any comparison-based search over n distinct values needs $\Omega(\log n)$ comparisons in the worst case, the tight, best-possible bound is $\Theta(\log n)$.

WATCH OUT FOR
Assuming a bitonic array needs $\Theta(\log^2 n)$ comparisons because it "looks like" you must binary-search twice (once to find the peak, once inside a half) and multiplying the two $O(\log n)$ stages together instead of adding them. The two stages are sequential (add), not nested (multiply), so the total stays $O(\log n)$, not $O(\log^2 n)$.

Q.42 MSQ · 2 MARKS

Discrete Mathematics › Sets & relations

Let $\mathcal{F}$ be the set of all functions from $\{1, \ldots, n\}$ to $\{0, 1\}$. Define the binary relation $\preccurlyeq$ on $\mathcal{F}$ as follows:

$\forall f, g \in \mathcal{F},\ f \preccurlyeq g$ if and only if $\forall x \in \{1, \ldots, n\},\ f(x) \leq g(x)$, where $0 \leq 1$.

Which of the following statement(s) is/are TRUE?

(A) $\preccurlyeq$ is a symmetric relation

(B) $(\mathcal{F}, \preccurlyeq)$ is a partial order

(C) $(\mathcal{F}, \preccurlyeq)$ is a lattice

(D) $\preccurlyeq$ is an equivalence relation

Official answer: **B, C**

SOLUTION

1. Each $f \in \mathcal{F}$ is a bit vector $(f(1), \ldots, f(n))$, and $f \preccurlyeq g$ means $f(x) \leq g(x)$ in every position. This is the componentwise order on $\{0, 1\}^n$.
2. Reflexive: $f(x) \leq f(x)$ for all x, so $f \preccurlyeq f$.
3. Antisymmetric: if $f \preccurlyeq g$ and $g \preccurlyeq f$, then $f(x) \leq g(x)$ and $g(x) \leq f(x)$ for every x. So $f(x) = g(x)$ for every x, and $f = g$.
4. Transitive: if $f \preccurlyeq g$ and $g \preccurlyeq h$, then $f(x) \leq g(x) \leq h(x)$ for all x, so $f \preccurlyeq h$. The relation is reflexive, antisymmetric and transitive, so $(\mathcal{F}, \preccurlyeq)$ is a partial order. (B) is true.
5. Lattice: for any f, g define $(f \vee g)(x) = \max(f(x), g(x))$ and $(f \wedge g)(x) = \min(f(x), g(x))$. Both take values in $\{0, 1\}$, so both lie in $\mathcal{F}$. The first is an upper bound of f and g, and any upper bound h has $h(x) \geq \max(f(x), g(x))$ at each x, so it is the least upper bound. Dually, the second is the greatest lower bound. Every pair has a join and a meet, so $(\mathcal{F}, \preccurlyeq)$ is a lattice (the Boolean lattice 2^n). (C) is true.
6. Symmetric fails: take $f \equiv 0$ and $g \equiv 1$. Then $f \preccurlyeq g$, but $g \not\preccurlyeq f$ because $1 \not\leq 0$. So (A) is false.
7. An equivalence relation must be symmetric, and this one is not, so (D) is false. The true statements are (B) and (C).

KEY IDEA
Comparing bit vectors position by position gives the componentwise order on $\{0, 1\}^n$. It is a partial order, and position-wise max and min give the join and meet, so it is the Boolean lattice 2^n.

WATCH OUT FOR
Treating the relation as symmetric or an equivalence because "$\leq$" looks like "=", or checking only the partial-order axioms and forgetting that every pair needs a join and a meet.

Q.43 MSQ · 2 MARKS

Digital Electronics › Boolean logic

Q.43 Given the following Karnaugh Map for a Boolean function $F(w, x, y, z)$:

wx \ yz	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

Which one or more of the following Boolean expression(s) represent(s) F?

(A) $\bar{w}\bar{x}\bar{y}\bar{z} + w\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z} + w\bar{x}y\bar{z} + xz$

(B) $\bar{w}\bar{x}\bar{y}\bar{z} + \bar{w}\bar{x}y\bar{z} + w\bar{x}yz + xz$

(C) $\bar{w}\bar{x}\bar{y}\bar{z} + w\bar{x}\bar{y}\bar{z} + w\bar{x}\bar{y}z + xz$

(D) $\bar{x}\bar{z} + xz$

Official answer: **A, D**

SOLUTION

1. Number each cell by $8w + 4x + 2y + z$. Reading the map with rows $wx = 00, 01, 11, 10$ and columns $yz = 00, 01, 11, 10$: row 00 has 1s at $yz = 00$ and 10, which are minterms 0 and 2. Row 01 has 1s at $yz = 01$ and 11, which are minterms 5 and 7. Row 11 has 1s at minterms 13 and 15. Row 10 has 1s at minterms 8 and 10.
2. So $F = \sum m(0, 2, 5, 7, 8, 10, 13, 15)$. All other minterms (1, 3, 4, 6, 9, 11, 12, 14) have $F = 0$.
3. Grouping: the four corners 0, 2, 8, 10 form a quad in which $x = 0$ and $z = 0$ are fixed, giving $\bar{x}\,\bar{z}$. The centre block 5, 7, 13, 15 is a quad in which $x = 1$ and $z = 1$ are fixed, giving xz. Hence $F = \bar{x}\,\bar{z} + xz$, which is option (D).
4. Option (A): the four product terms are $\bar{w}\,\bar{x}\,\bar{y}\,\bar{z}$ (minterm 0), $w\bar{x}\,\bar{y}\,\bar{z}$ (minterm 8), $\bar{w}\,\bar{x}y\bar{z}$ (minterm 2) and $w\bar{x}y\bar{z}$ (minterm 10). Together they are exactly the corner quad $\bar{x}\,\bar{z}$, and xz covers 5, 7, 13, 15. So (A) $= \sum m(0, 2, 8, 10, 5, 7, 13, 15) = F$. It is correct.
5. Option (B): the terms $\bar{w}\,\bar{x}\,\bar{y}\,\bar{z}$ and $\bar{w}\,\bar{x}y\bar{z}$ are minterms 0 and 2. The term $w\bar{x}yz$ is minterm 11, where $F = 0$. Minterms 8 and 10, where $F = 1$, are not covered. So (B) $\neq F$.

6. Option (C): the terms $\overline{w}\,\overline{x}\,\overline{y}\,\overline{z}$ and $w\overline{x}\,\overline{y}\,\overline{z}$ are minterms 0 and 8. The term $w\overline{x}\,\overline{y}z$ is minterm $8+1=9$, where $F=0$. Minterms 2 and 10, where $F=1$, are not covered. So (C) $\neq F$.
7. Therefore the expressions that represent F are (A) and (D).

KEY IDEA

Write the 1-cells of the map as the minterm list $\sum m(0,2,5,7,8,10,13,15)$. This is two quads, the four corners ($\overline{x}\,\overline{z}$) and the centre block ($xz$). Then check each option by converting every one of its product terms to a minterm number and comparing the set with F.

WATCH OUT FOR

Judging an option by the shape of its terms instead of converting each term to a minterm. A term that differs from a covered minterm in just one bar, such as $w\overline{x}\,\overline{y}z$ (minterm 9) in place of $w\overline{x}y\overline{z}$ (minterm 10), looks almost right but lands on a 0-cell. Another slip is writing a minterm with the wrong variable weights ($8w+4x+2y+z$).

Q.44 MSQ · 2 MARKS

Engineering Mathematics › Linear algebra

Consider a system of linear equations $PX=Q$ where $P\in\mathbb{R}^{3\times3}$ and $Q\in\mathbb{R}^{3\times1}$. Suppose P has an LU decomposition, $P=LU$, where $L=\begin{bmatrix}1&0&0\\l_{21}&1&0\\l_{31}&l_{32}&1\end{bmatrix}$ and $U=\begin{bmatrix}u_{11}&u_{12}&u_{13}\\0&u_{22}&u_{23}\\0&0&u_{33}\end{bmatrix}$. Which of the following statement(s) is/are TRUE?

(A) The system $PX=Q$ can be solved by first solving $LY=Q$ and then $UX=Y$.

(B) If P is invertible, then both L and U are invertible.

(C) If P is singular, then at least one of the diagonal elements of U is zero.

(D) If P is symmetric, then both L and U are symmetric.

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

SOLUTION

1. Substitute $P=LU$: $PX=L(UX)=Q$. Put $Y=UX$. Then $LY=Q$ is solved by forward substitution, and $UX=Y$ by back substitution. So (A) is true.
2. L is lower triangular with all diagonal entries 1, so $\det L=1\neq0$ and L is always invertible.
3. U is upper triangular, so $\det U=u_{11}u_{22}u_{33}$.
4. Hence $\det P=\det L\cdot\det U=u_{11}u_{22}u_{33}$.
5. (B) If P is invertible, $\det P\neq0$, so $u_{11}u_{22}u_{33}\neq0$ and every $u_{ii}\neq0$. Then U is invertible, and L is invertible by step 2. True.
6. (C) If P is singular, $\det P=0$, so $u_{11}u_{22}u_{33}=0$. At least one diagonal element of U is zero. True.
7. (D) Take the symmetric matrix $P=\begin{bmatrix}1&2&0\\2&5&0\\0&0&1\end{bmatrix}$. Eliminating with multiplier $l_{21}=2$ gives $L=\begin{bmatrix}1&0&0\\2&1&0\\0&0&1\end{bmatrix}$ and $U=\begin{bmatrix}1&2&0\\0&1&0\\0&0&1\end{bmatrix}$. Check: $LU=\begin{bmatrix}1&2&0\\2&5&0\\0&0&1\end{bmatrix}=P$. Neither L nor U is symmetric, so (D) is false.
8. Therefore the true statements are (A), (B) and (C).

KEY IDEA

With $P=LU$ and L unit lower triangular, $\det P=u_{11}u_{22}u_{33}$. So P is invertible exactly when every diagonal entry of U is nonzero, and $PX=Q$ is solved by forward then back substitution.

WATCH OUT FOR

Assuming a symmetric P gives symmetric L and U. In fact symmetry gives $P=LDL^T$ (when the pivots are nonzero), and $U=DL^T$ is not symmetric unless D is the identity.

Q.45 MSQ · 2 MARKS

Data Structures and Algorithms › Linear structures

Consider a stack data structure into which we can PUSH and POP records. Assume that each record pushed in the stack has a positive integer key and that all keys are distinct.

We wish to augment the stack data structure with an $O(1)$ time MIN operation that returns a pointer to the record with smallest key present in the stack

1) without deleting the corresponding record, and
2) without increasing the complexities of the standard stack operations.

Which one or more of the following approaches can achieve it?

(A) Keep with every record in the stack, a pointer to the record with the smallest key below it.

(B) Keep a pointer to the record with the smallest key in the stack.

(C) Keep an auxiliary array in which the key values of the records in the stack are maintained in sorted order.

(D) Keep a Min-Heap in which the key values of the records in the stack are maintained.

Official answer: **A**

SOLUTION

1. Requirement: PUSH and POP must stay O(1) (the standard stack complexities), and MIN must become O(1), without deleting/losing any record.
2. (A) Augmented pointer per record: when a record is pushed, compare its key with the running minimum recorded at the (soon-to-be-previous) top; the new record stores a pointer to whichever is smaller (itself or that running minimum) as its own 'min-so-far' pointer. PUSH: O(1), one comparison and one pointer write. POP: O(1), simply discard the top; the record now exposed as the new top already carries the correct minimum for the reduced stack, computed at the time it was pushed. MIN: O(1), read the pointer stored at the current top. No case increases the standard operations' complexity, so (A) works.
3. (B) Single global min pointer: PUSH is still O(1) (update the pointer only if the new key is smaller). But POP fails when the popped record is the one the pointer references; the old minimum-before-that-push is nowhere recorded, so recovering the new minimum needs an O(n) scan of the remaining stack in the worst case. This increases POP's complexity, so (B) is rejected.
4. (C) Auxiliary sorted array: inserting a new key at its correct sorted position requires shifting up to n array elements, making PUSH O(n) in the worst case; removing a specific (not necessarily top-of-array) key on POP similarly needs an O(n) search-and-shift. Both standard operations' complexities increase, so (C) is rejected.
5. (D) Min-heap of keys: PUSH now requires a heap insertion, O(log n) rather than O(1). POP must remove the exact record being popped from the stack from the heap (not necessarily the heap's current minimum), which needs O(log n) with extra index bookkeeping rather than O(1). Both standard operations get slower than O(1), so (D) is rejected.
6. Only (A) delivers O(1) MIN while leaving PUSH and POP at O(1); the answer is (A) only.

KEY IDEA

To add O(1) MIN to a stack without touching PUSH/POP's O(1) cost, the extra information must be attachable and readable in O(1) per operation and must never need to be recomputed from scratch on POP. Storing, with each element as it is pushed, a pointer to the minimum of everything below it achieves this because that stored value is fixed forever once written and POP never has to search for a replacement minimum: the newly exposed top already carries it.

WATCH OUT FOR

Thinking any structure that can report the minimum in O(1) is good enough, and overlooking that POP must also stay O(1): a single global min pointer (B) answers MIN in O(1) but forces an O(n) rescan on POP when the current minimum is popped, and a sorted array (C) or min-heap (D) cannot be kept updated in O(1) per PUSH/POP even though they can report the minimum quickly.

Q.46 MSQ · 2 MARKS

Database Management and Warehousing › Normal forms

Consider the following relational schema along with all the functional dependencies that hold on them.

$R1(A, B, C, D, E)$: $\{D \to E,\ EA \to B,\ EB \to C\}$

$R2(A, B, C, D)$: $\{A \to D,\ A \to B,\ C \to A\}$

Which of the following statement(s) is/are TRUE?

(A) R1 is in 3NF

(B) R2 is in 3NF

(C) R1 is NOT in 3NF

(D) R2 is NOT in 3NF

Official answer: **C, D**

SOLUTION

1. **R1, find the key.** A and D never appear on a right-hand side, so every candidate key contains AD. Compute $\{A, D\}^+$: $D \to E$ gives E; $EA \to B$ gives B; $EB \to C$ gives C. So $\{A, D\}^+ = \{A, B, C, D, E\}$, and AD is the only candidate key. The prime attributes are A and D.
2. **R1, test 3NF.** A dependency $X \to Y$ meets 3NF if X is a superkey or Y is prime (or $Y \subseteq X$). Take $D \to E$: $D^+ = \{D, E\}$, so D is not a superkey, and E is not prime. Both conditions fail, so $R1$ is NOT in 3NF. (It is a partial dependency on the key AD.)
3. **R2, find the key.** C never appears on a right-hand side, so every key contains C. Compute C^+: $C \to A$ gives A; $A \to D$ gives D; $A \to B$ gives B. So $C^+ = \{A, B, C, D\}$, and C is the only candidate key. The only prime attribute is C.
4. **R2, test 3NF.** Take $A \to D$: $A^+ = \{A, B, D\}$ does not contain C, so A is not a superkey, and D is not prime. So $A \to D$ violates 3NF (and $A \to B$ does too). $R2$ is NOT in 3NF. (It is a transitive dependency $C \to A \to D$.)
5. Hence (C) and (D) are true, and (A) and (B) are false.

KEY IDEA

Find the candidate keys by closure, then look for any functional dependency whose left side is not a superkey and whose right side is not a prime attribute. One such dependency is enough to rule out 3NF.

WATCH OUT FOR

Forgetting that a non-key attribute can depend on part of the key (D to E in R1) or on another non-key attribute (A to D in R2). Both break 3NF, and both are easy to miss when the key is only checked against the dependency that uses it.

Q.47 MSQ · 2 MARKS

Operating Systems › Virtual memory

Q.47 Consider a demand paging system with three frames, and the following page reference string: 1 2 3 4 5 4 1 6 4 5 1 3 2. The contents of the frames are as follows initially and after each reference (from left to right):

initially	*after*												
-	1*	2*	3*	4*	5*	4	1	6*	4	5	1*	3*	2*
-	1	1	1	1	1	1	1	6	6	6	6	6	2
-	-	2	2	4	4	4	4	4	4	4	1	1	1
-	-	-	3	3	5	5	5	5	5	5	5	3	3

The *-marked references cause page replacements.

Which one or more of the following could be the page replacement policy/policies in use?

(A) Least Recently Used page replacement policy

(B) Least Frequently Used page replacement policy

(C) Most Frequently Used page replacement policy

(D) Optimal page replacement policy

Official answer: **D**

SOLUTION

1. Number the references by position:

Position	1	2	3	4	5	6	7	8	9	10	11	12	13
Reference	1	2	3	4	5	4	1	6	4	5	1	3	2

   From the table, the evictions are: position 4 (page 4) evicts 2, position 5 (page 5) evicts 3, position 8 (page 6) evicts 1, position 11 (page 1) evicts 4, position 12 (page 3) evicts 5, and position 13 (page 2) evicts 6. Positions 6, 7, 9 and 10 are hits.

2. **LRU (A).** At position 4 the frames hold 1, 2 and 3, last used at positions 1, 2 and 3. The least recently used page is 1, so LRU evicts 1. The table evicts 2, so the policy is not LRU.

3. **Frequency counts.** For LFU and MFU, a page's frequency is the number of times it has been referenced in the string so far. Before position 8 the counts are $1:2$ (positions 1 and 7), $4:2$ (positions

4 and 6) and $5:1$ (position 5). Before position 12 the pages in frames have counts $6:1$ (position 8), $1:3$ (positions 1, 7 and 11) and $5:2$ (positions 5 and 10).

4. **LFU (B).** At position 8 the frames hold 1, 4 and 5 with counts 2, 2 and 1. LFU must evict 5, the least frequently used page. The table evicts 1, so the policy is not LFU. All three pages entered memory once and were never evicted before this point, so the counts are the same whether or not a count is reset on eviction. LFU fails either way.
5. **MFU (C).** At position 12 the frames hold 6, 1 and 5 with counts 1, 3 and 2. MFU must evict page 1, the most frequently used page. The table evicts 5, so the policy is not MFU. There is a second, separate failure at position 4. The three pages in frames each have count 1, so MFU has a tie, and both usual tie-breaks pick page 1: oldest-loaded first (FIFO) and least recently used. The table evicts 2.
6. **Optimal (D).** At each replacement, evict the page whose next use is farthest away, or a page that is never used again. Position 4: frames 1, 2, 3, next used at 7, 13 and 12, so evict 2. Matches. Position 5: frames 1, 4, 3, next used at 7, 6 and 12, so evict 3. Matches. Position 8: frames 1, 4, 5, next used at 11, 9 and 10, so evict 1. Matches.
7. **Optimal, continued.** At position 11 the frames hold 6, 4 and 5, and the rest of the string is 3, 2. None of the three pages is used again, so evicting any of them is optimal, including the table's choice of 4. At position 12 the frames hold 6, 1 and 5, and only 2 follows, so evicting 5 is optimal. At position 13 nothing follows, so evicting 6 is optimal. Every replacement in the table is one that Optimal can make.
8. Only the Optimal policy fits every replacement in the table. The answer is (D).

KEY IDEA

Test each policy at every replacement in the table. The first replacement where the policy's choice differs from the table's rules it out. Optimal works because each eviction removes the page needed farthest in the future, and after position 10 none of the pages in memory is used again.

WATCH OUT FOR

Keeping MFU (C) by resetting a page's count to zero when it is evicted and breaking every tie in whichever way matches the table. With counts taken over the whole reference string, page 1 has 3 references at position 12, so MFU would evict it rather than page 5. With either usual tie-break (FIFO or LRU), MFU already fails at position 4.

Q.48 MSQ · 2 MARKS

Operating Systems › Deadlock

$P = \{P_1, P_2, P_3, P_4\}$ consists of all active processes in an operating system. $R = \{R_1, R_2, R_3, R_4\}$ consists of single instances of distinct types of resources in the system.

The resource allocation graph has the following assignment and claim edges.

Assignment edges: $R_1 \to P_1,\ R_2 \to P_2,\ R_3 \to P_3,\ R_4 \to P_4$ (the assignment edge $R_1 \to P_1$ means resource R_1 is assigned to process P_1, and so on for others)

Claim edges: $P_1 \to R_2,\ P_2 \to R_3,\ P_3 \to R_1,\ P_2 \to R_4,\ P_4 \to R_2$ (the claim edge $P_1 \to R_2$ means process P_1 is waiting for resource R_2, and so on for others)

Which of the following statement(s) is/are CORRECT?

(A) Aborting P_1 makes the system deadlock free.

(B) Aborting P_3 makes the system deadlock free.

(C) Aborting P_2 makes the system deadlock free.

(D) Aborting P_1 and P_4 makes the system deadlock free.

Official answer: **C, D**

SOLUTION

1. Every resource has a single instance, so a cycle in the resource allocation graph means deadlock, and no cycle means no deadlock. Convert to a wait-for graph: $P_i \to P_j$ if P_i waits for a resource held by P_j.

2. Holders: R_1 is held by P_1, R_2 by P_2, R_3 by P_3, R_4 by P_4.
3. Waiting edges give: $P_1 \to P_2$ (via R_2), $P_2 \to P_3$ (via R_3), $P_2 \to P_4$ (via R_4), $P_3 \to P_1$ (via R_1), $P_4 \to P_2$ (via R_2).
4. Cycles in the wait-for graph: $P_1 \to P_2 \to P_3 \to P_1$ and $P_2 \to P_4 \to P_2$. Both pass through P_2.
5. (A) Abort P_1: the edges $P_2 \to P_4$ and $P_4 \to P_2$ remain, so the cycle $P_2 \leftrightarrow P_4$ remains. Deadlock persists. False.
6. (B) Abort P_3: the edges $P_2 \to P_4$ and $P_4 \to P_2$ remain. Deadlock persists. False.
7. (C) Abort P_2: it releases R_2, R_3 and R_4 and withdraws its requests. Every cycle used P_2, so the only edge left is $P_3 \to P_1$, which has no cycle. True.
8. (D) Abort P_1 and P_4: both edges out of and into P_1 and P_4 disappear, so only $P_2 \to P_3$ remains. No cycle. True.
9. Correct options: (C) and (D).

KEY IDEA

With single-instance resources, deadlock is exactly a cycle in the wait-for graph. Here both cycles pass through P_2, so aborting P_2 breaks both. Aborting $\{P_1, P_4\}$ also breaks both, one process per cycle.

WATCH OUT FOR

Aborting only one process of the three-process cycle (P_1 or P_3) and forgetting the second cycle $P_2 \leftrightarrow P_4$ that does not contain it.

Q.49 MSQ · 2 MARKS

Digital Electronics › Number systems

Three floating point numbers X, Y, and Z are stored in three registers R_X, R_Y, and R_Z, respectively in IEEE 754 single precision format as given below in hexadecimal:

R_X = 0xC1100000 , R_Y = 0x40C00000 , and R_Z = 0x41400000

Which of the following option(s) is/are CORRECT?

(A) $4(X+Y)+Z=0$

(B) $2Y-Z=0$

(C) $4X+3Z=0$

(D) $X+Y+Z=0$

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

SOLUTION

1. IEEE 754 single precision: value $= (-1)^s \times 1.f \times 2^{e-127}$, with 1 sign bit, 8 exponent bits and 23 fraction bits.
2. 0xC1100000 = 1100 0001 0001 0000 ... , so $s=1$, exponent field 10000010 $=130$ (exponent 3), fraction 0010000... $=0.125$. Hence $X = -1.125 \times 2^3 = -9$.
3. 0x40C00000 = 0100 0000 1100 0000 ... , so $s=0$, exponent field 10000001 $=129$ (exponent 2), fraction 1000000... $=0.5$. Hence $Y = 1.5 \times 2^2 = 6$.
4. 0x41400000 = 0100 0001 0100 0000 ... , so $s=0$, exponent field 10000010 $=130$ (exponent 3), fraction 1000000... $=0.5$. Hence $Z = 1.5 \times 2^3 = 12$.
5. (A) $4(X+Y)+Z = 4(-9+6)+12 = -12+12 = 0$. Correct.
6. (B) $2Y - Z = 12 - 12 = 0$. Correct.
7. (C) $4X + 3Z = -36 + 36 = 0$. Correct.
8. (D) $X+Y+Z = -9+6+12 = 9 \neq 0$. Wrong.
9. So the correct options are A, B and C.

KEY IDEA
Split each hexadecimal word into sign, 8 exponent bits and 23 fraction bits, decode with bias 127 and the implicit leading 1, then test each relation with the decoded values.

WATCH OUT FOR
Forgetting the implicit leading 1 in the mantissa, or reading the exponent field without subtracting the bias 127. Misaligning the hex digits against the 1+8+23 bit fields is another frequent slip.

Q.50 MSQ · 2 MARKS

Digital Electronics › Boolean logic

Which of the following Boolean algebraic equation(s) is/are CORRECT?

(A) $\overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + ABC = BC + \overline{B}\,\overline{C} + \overline{A}\,\overline{B}$

(B) $AB + \overline{A}C + BC = AB + \overline{A}C$

(C) $(A + C)(\overline{A} + B) = AB + \overline{A}C$

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

Official answer: **B, C, D**

SOLUTION

1. (A) Index the variables as ABC with A as the most significant bit. The left side has $\overline{A}BC = 011 = 3$, $A\overline{B}\,\overline{C} = 100 = 4$, $\overline{A}\,\overline{B}\,\overline{C} = 000 = 0$, $A\overline{B}C = 101 = 5$ and $ABC = 111 = 7$. So the left side is $\sum m(0, 3, 4, 5, 7)$.
2. The right side has $BC = m(3, 7)$, $\overline{B}\,\overline{C} = m(0, 4)$ and $\overline{A}\,\overline{B} = m(0, 1)$. So it is $\sum m(0, 1, 3, 4, 7)$. Minterm 5 is only on the left and minterm 1 is only on the right, so (A) is false.
3. (B) The consensus theorem says $XY + \overline{X}Z + YZ = XY + \overline{X}Z$. With $X = A$, $Y = B$, $Z = C$, the term BC is redundant. So (B) is true.
4. (C) Expand: $(A + C)(\overline{A} + B) = A\overline{A} + AB + \overline{A}C + BC = AB + \overline{A}C + BC$. By consensus, BC is redundant, so this equals $AB + \overline{A}C$. So (C) is true.
5. (D) The bar covers the whole product, so De Morgan gives the sum of the complements of the four factors: $\overline{(A + \overline{B} + \overline{D})} = \overline{A}BD$, $\overline{(C + D)} = \overline{C}\,\overline{D}$, $\overline{(\overline{A} + C + D)} = A\overline{C}\,\overline{D}$ and $\overline{(A + B + \overline{D})} = \overline{A}\,\overline{B}D$.
6. Add them: $\overline{A}BD + \overline{C}\,\overline{D} + A\overline{C}\,\overline{D} + \overline{A}\,\overline{B}D = \overline{C}\,\overline{D}(1 + A) + \overline{A}D(B + \overline{B}) = \overline{C}\,\overline{D} + \overline{A}D$. So (D) is true.
7. The correct equations are (B), (C) and (D).

KEY IDEA
Check each identity by minterm lists or by the consensus theorem $XY + \overline{X}Z + YZ = XY + \overline{X}Z$. For a complemented product, apply De Morgan to the whole expression.

WATCH OUT FOR
Treating (A) as true because the two sides share most minterms. Minterm 5 appears only on the left and minterm 1 only on the right. Another slip is complementing only some factors of the product in (D).

Q.51 MSQ · 2 MARKS

Compiler Design › Parsing

Consider two grammars G_1 and G_2 with the production rules given below:

G_1: $S \rightarrow if\ E\ then\ S \mid if\ E\ then\ S\ else\ S \mid a$; $E \rightarrow b$

G_2: $S \rightarrow if\ E\ then\ S \mid M$; $M \rightarrow if\ E\ then\ M\ else\ S \mid c$; $E \rightarrow b$

where $if, then, else, a, b, c$ are the terminals. Which of the following option(s) is/are CORRECT?

(A) G_1 is not $LL(1)$ and G_2 is $LL(1)$.

(B) G_1 is $LL(1)$ and G_2 is not $LL(1)$.

(C) G_1 and G_2 are not $LL(1)$.

(D) G_1 and G_2 are ambiguous.

Official answer: **C, D**

SOLUTION

1. An $LL(1)$ grammar needs, for every nonterminal, the FIRST sets of its alternatives to be pairwise disjoint (and, for a nullable alternative, disjoint from FOLLOW). No nonterminal here is nullable, so only the FIRST sets matter. Any ambiguous grammar is also not $LL(1)$.
2. G_1: both $S \to if\ E\ then\ S$ and $S \to if\ E\ then\ S\ else\ S$ have $\text{FIRST} = \{if\}$. They overlap, so G_1 is not $LL(1)$.
3. G_1 is the dangling-else grammar, so it is ambiguous. The string `if b then if b then a else a` has two parse trees: the `else` can belong to the inner `if` ($S \to if\ E\ then\ S$ outside, $S \to if\ E\ then\ S\ else\ S$ inside) or to the outer `if` ($S \to if\ E\ then\ S\ else\ S$ outside, $S \to if\ E\ then\ S$ inside).
4. G_2: $\text{FIRST}(M) = \{if, c\}$. The alternatives of S are $if\ E\ then\ S$ with $\text{FIRST} = \{if\}$ and M with $\text{FIRST} = \{if, c\}$. Both contain if, so on input if the parser cannot choose between them. G_2 is not $LL(1)$. So (A) and (B) are false and (C) is true.
5. Is G_2 ambiguous? The else-part of $M \to if\ E\ then\ M\ else\ S$ is an S, and an S can be an unmatched $if\ E\ then\ S$. So an `else` written after it can attach in two places. Take the string `if b then if b then c else if b then c else c`, and number its keywords `if1 ... if2 ... else1 if3 ... else2`.
6. Parse 1 (`else2` belongs to `if1`): $S \Rightarrow M \Rightarrow if_1\ E\ then\ M_1\ else_2\ S$ with $S \Rightarrow M \Rightarrow c$. Here $M_1 \Rightarrow if_2\ E\ then\ M\ else_1\ S'$ with $M \Rightarrow c$ and $S' \Rightarrow if_3\ E\ then\ S \Rightarrow if_3\ b\ then\ c$ (using $S \to M \to c$). Every step uses a production of G_2.
7. Parse 2 (`else2` belongs to `if3`): $S \Rightarrow if_1\ E\ then\ S_1$. Then $S_1 \Rightarrow M \Rightarrow if_2\ E\ then\ M\ else_1\ S_2$ with $M \Rightarrow c$, and $S_2 \Rightarrow M \Rightarrow if_3\ E\ then\ M\ else_2\ S$ with $M \Rightarrow c$ and $S \Rightarrow M \Rightarrow c$. Every step is again valid.
8. The two trees are different (in one `else2` pairs with `if1`, in the other with `if3`) and they produce the same string. So G_2 is ambiguous. It is not the standard unambiguous matched/unmatched grammar, because there the else-part of a matched statement must itself be matched, while G_2 allows any S.
9. Both grammars are ambiguous, so (D) is true. (C) is true as well. Correct options: (C) and (D).

KEY IDEA

FIRST sets clash for both grammars on the token if, so neither is $LL(1)$. G_1 is the classic dangling-else grammar. G_2 only looks like the fix: the else-part of M is a general S, which can be an unmatched `if`, so a later `else` can still attach in two places, as in `if b then if b then c else if b then c else c`.

WATCH OUT FOR

Treating G_2 as the textbook unambiguous dangling-else grammar. In that grammar the else-part of a matched statement must itself be matched. G_2 writes $M \to if\ E\ then\ M\ else\ S$, which lets an unmatched `if` sit inside a matched one. A short test string like `if b then if b then c else c` has only one parse, so you need a longer string with two elses to see the ambiguity.

Q.52 MSQ · 2 MARKS

Theory of Computation › Regular languages

Let $\Sigma = \{a, b, c\}$. For $x \in \Sigma^*$ and $\alpha \in \Sigma$, let $\#_\alpha(x)$ denote the number of occurrences of α in x.

Which one or more of the following option(s) define(s) regular language(s)?

(A) $\{a^m b^n \mid m, n \geq 0\}$

(B) $\{a, b\}^* \cap \{a^m b^n c^{m-n} \mid m \geq n \geq 0\}$

(C) $\{w \mid w \in \{a, b\}^*,\ \#_a(w) \equiv 2 \pmod 7,\ \text{and } \#_b(w) \equiv 3 \pmod 9\}$

(D) $\{w \mid w \in \{a,b\}^*,\ \#_a(w) \equiv 2 \pmod 7,\ \text{and } \#_a(w) = \#_b(w)\}$

Official answer: **A, C**

SOLUTION

1. (A) $\{a^m b^n \mid m, n \geq 0\}$ is exactly the language of the regular expression a^*b^*, so it is regular.
2. (B) A string in $\{a,b\}^*$ has no c, so a string $a^m b^n c^{m-n}$ can lie in $\{a,b\}^*$ only when $m - n = 0$. The intersection is therefore $\{a^n b^n \mid n \geq 0\}$. This language is not regular, by the pumping lemma. So (B) is not regular.
3. (C) The two conditions count different letters, so they can be tracked independently. A DFA keeps the pair $(\#_a \bmod 7,\ \#_b \bmod 9)$ in $7 \times 9 = 63$ states. It starts at $(0,0)$, reading a adds 1 to the first coordinate (mod 7), reading b adds 1 to the second coordinate (mod 9), and the only accepting state is $(2,3)$. So (C) is regular.
4. (D) Call the language L_D. Regular languages are closed under intersection, so if L_D were regular then $L_D \cap a^*b^*$ would be regular. A string $a^i b^j$ in L_D needs $i = j$ and $i \equiv 2 \pmod 7$, so $L_D \cap a^*b^* = \{a^n b^n \mid n \equiv 2 \pmod 7\}$.
5. This language is not regular. Suppose it had pumping length p_0. Choose $n \geq p_0$ with $n \equiv 2 \pmod 7$ and let $w = a^n b^n$. The pumped part y lies inside the first p_0 symbols, so $y = a^k$ with $k \geq 1$. Then $xy^2z = a^{n+k}b^n$ has $\#_a \neq \#_b$, so it is not in the language. This contradicts the pumping lemma, so $L_D \cap a^*b^*$ is not regular, and therefore L_D is not regular.
6. The regular languages are (A) and (C).

KEY IDEA

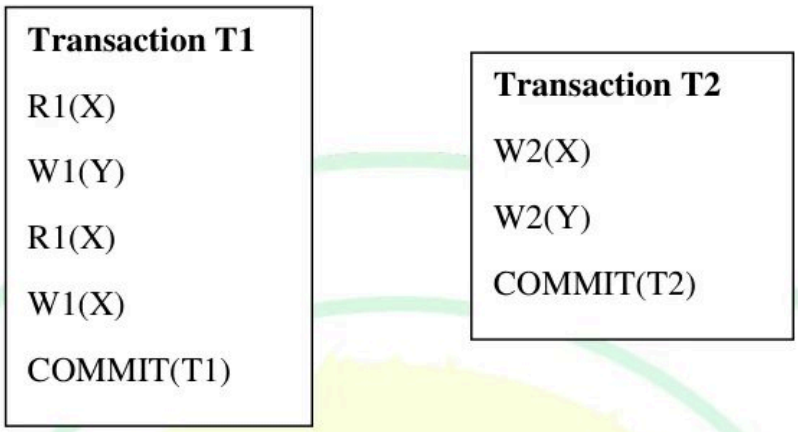
Check each language by either giving a finite automaton (independent modular counters give a product DFA) or by intersecting with a simple regular set to expose a non-regular core such as $a^n b^n$.

WATCH OUT FOR

Treating (D) as regular because of the mod 7 condition, or missing that in (B) the intersection with $\{a,b\}^*$ removes every c and forces $m = n$, which leaves $a^n b^n$.

Q.53 MSQ · 2 MARKS

Q.53 Consider the database transactions T1 and T2, and data items X and Y. Which of the schedule(s) is/are conflict serializable?



(A) R1(X), W2(X), W1(Y), W2(Y), R1(X), W1(X), COMMIT(T2), COMMIT(T1)

(B) W2(X), R1(X), W2(Y), W1(Y), R1(X), COMMIT(T2), W1(X), COMMIT(T1)

(C) R1(X), W1(Y), W2(X), W2(Y), R1(X), W1(X), COMMIT(T1), COMMIT(T2)

(D) W2(X), R1(X), W1(Y), W2(Y), R1(X), COMMIT(T2), W1(X), COMMIT(T1)

Official answer: **B**

SOLUTION

1. Method: draw the precedence graph. Add an edge $T_i \to T_j$ when an operation of T_i comes before a conflicting operation of T_j on the same item (different transactions, same item, at least one a write). A schedule is conflict serializable exactly when this graph has no cycle.
2. (A) $R_1(X)$ comes before $W_2(X)$, giving $T_1 \to T_2$. $W_2(X)$ comes before the second $R_1(X)$, giving $T_2 \to T_1$. The graph has a cycle, so (A) is not conflict serializable.
3. (B) $W_2(X)$ precedes $R_1(X)$, so $T_2 \to T_1$. $W_2(Y)$ precedes $W_1(Y)$, so $T_2 \to T_1$. $W_2(X)$ precedes $W_1(X)$, so $T_2 \to T_1$. Every conflicting pair gives an edge $T_2 \to T_1$ and there is no $T_1 \to T_2$ edge, so the graph is acyclic. The schedule is equivalent to the serial order T_2, T_1. (B) is conflict serializable.
4. (C) $R_1(X)$ comes before $W_2(X)$, giving $T_1 \to T_2$. $W_2(X)$ comes before the second $R_1(X)$, giving $T_2 \to T_1$. The graph has a cycle, so (C) is not conflict serializable.

5. (D) $W_2(X)$ precedes $R_1(X)$, giving $T_2 \to T_1$. $W_1(Y)$ precedes $W_2(Y)$, giving $T_1 \to T_2$. The graph has a cycle, so (D) is not conflict serializable.
6. Only (B) is conflict serializable.

KEY IDEA
Build the precedence graph from conflicting operations (same item, different transactions, at least one write). The schedule is conflict serializable exactly when the graph has no cycle.

WATCH OUT FOR
Looking only at the first conflict between the transactions and missing the opposite-direction edge from a later operation, such as the second R1(X) after W2(X) in (A) and (C), or W1(Y) before W2(Y) in (D).

Q.54 MSQ · 2 MARKS

Database Management and Warehousing › SQL

Consider the following relational schema:

Students (rollno: integer, name: string, age: integer, cgpa: real)
Courses (courseno: integer, cname: string, credits: integer)
Enrolled (rollno: integer, courseno: integer, grade: string)

(Primary keys underlined: rollno in Students; courseno in Courses; (rollno, courseno) in Enrolled.)

Which of the following options is/are correct SQL query/queries to retrieve the names of the students enrolled in course number (i.e., courseno) 1470?

(A)

```
SELECT S.name
FROM Students S
WHERE EXISTS (SELECT * FROM Enrolled E
              WHERE E.courseno = 1470
              AND E.rollno = S.rollno);
```

(B)

```
SELECT S.name
FROM Students S
WHERE SIZEOF (SELECT * FROM Enrolled E
              WHERE E.courseno = 1470
              AND E.rollno = S.rollno) > 0;
```

(C)

```
SELECT S.name
FROM Students S
WHERE  0 < (SELECT COUNT(*)
            FROM Enrolled E
            WHERE E.courseno = 1470
            AND E.rollno = S.rollno);
```

(D)

```
SELECT S.name
FROM Students S NATURAL JOIN Enrolled E
WHERE E.courseno = 1470;
```

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

SOLUTION

1. Goal: return the names of students who have an `Enrolled` row with `courseno = 1470`. Check each option for validity and meaning.
2. (A) The correlated subquery selects `Enrolled` rows with `courseno = 1470` and the same `rollno` as the current student `S`. `EXISTS` is true exactly when at least one such row exists. This is valid and correct.
3. (B) `SIZEOF` is not an SQL function or operator, so the query is rejected as a syntax error. Incorrect.

4. (C) For each student, the subquery counts the `Enrolled` rows with `courseno = 1470` and that student's `rollno`. `0 < COUNT(*)` holds exactly when at least one such row exists, which is the same condition as in (A). Valid and correct.
5. (D) `NATURAL JOIN` joins on all attribute names the two tables share. `Students` has $\{rollno, name, age, cgpa\}$ and `Enrolled` has $\{rollno, courseno, grade\}$. The only shared attribute is `rollno`, so each student is paired with their own enrolments. The `WHERE` clause then keeps the rows with `courseno = 1470`. Valid and correct.
6. Duplicates: $(rollno, courseno)$ is the key of `Enrolled`. A student therefore has at most one row for course 1470, so (A), (C) and (D) return each such student's name once. Two different students with the same name would still produce a repeated name in all three, equally.
7. Valid and correct queries: A, C and D.

KEY IDEA
To test whether a related row exists, SQL uses `EXISTS`, a count compared with 0, or a join. A natural join matches on every shared column name, and here the only one is `rollno`.

WATCH OUT FOR
Treating `SIZEOF` as a standard SQL operator because it sounds like a set-size test. Another slip is doubting that `NATURAL JOIN` works without an explicit `ON` clause, even though `rollno` is the only shared column.

Q.55 NAT · 2 MARKS

Computer Organization and Architecture › Memory & cache

Given a computing system with two levels of cache (L1 and L2) and a main memory. The first level (L1) cache access time is 1 nanosecond (ns) and the "hit rate" for L1 cache is 90% while the processor is accessing the data from L1 cache. Whereas, for the second level (L2) cache, the "hit rate" is 80% and the "miss penalty" for transferring data from L2 cache to L1 cache is 10 ns. The "miss penalty" for the data to be transferred from main memory to L2 cache is 100 ns.

Then the average memory access time in this system in nanoseconds is _____ . (rounded off to one decimal place)

Official answer: **4.0**

SOLUTION

1. Write AMAT for a two-level cache hierarchy as a recursive expression: AMAT = L1 access time + (L1 miss rate) × (penalty on an L1 miss), where the L1-miss penalty is itself the cost of consulting L2.
2. L1 access time = 1 ns. L1 hit rate = 90% ⇒ L1 miss rate = 1 − 0.9 = 0.1.
3. On an L1 miss, the data is sought in L2. If L2 hits, the cost charged is the stated "miss penalty for transferring data from L2 cache to L1 cache" = 10 ns. L2 hit rate = 80% ⇒ L2 miss rate = 1 − 0.8 = 0.2.
4. If L2 also misses (probability 0.2, given an L1 miss), an additional "miss penalty for the data to be transferred from main memory to L2 cache" = 100 ns is incurred on top of the 10 ns.
5. So the expected extra time on an L1 miss = 10 + (L2 miss rate) × 100 = 10 + 0.2 × 100 = 10 + 20 = 30 ns.
6. AMAT = L1 access time + (L1 miss rate) × (extra time on L1 miss) = 1 + 0.1 × 30 = 1 + 3 = 4.0 ns.

KEY IDEA

In a multi-level cache, AMAT is built recursively from the innermost level outward: AMAT = (access time of level i) + (miss rate of level i) × (penalty incurred on a miss at level i), where that miss penalty is itself the access/transfer cost to the next level plus (that level's own miss rate × the penalty of going further down).

WATCH OUT FOR

Treating the 10 ns and 100 ns figures as independent, unconditional penalties and simply adding them (e.g., $1 + 0.1 \times 10 + 0.08 \times 100$, using the L2 miss rate against the *overall* access count rather than conditioning it on an L1 miss first), or forgetting that the L2 miss penalty (100 ns) is only incurred on the fraction of accesses that miss in *both* L1 and L2, not on every L1 miss.

Q.56 NAT · 2 MARKS

Computer Organization and Architecture › Pipelining

A 5-stage instruction pipeline has stage delays of 180, 250, 150, 170, and 250, respectively, in nanoseconds. The delay of an inter-stage latch is 10 nanoseconds. Assume that there are no pipeline stalls due to branches and other hazards. The time taken to process 1000 instructions in microseconds is _____ . (rounded off to two decimal places)

Official answer: **260.20 to 261.20**

SOLUTION

1. Clock period must accommodate the slowest stage plus the inter-stage latch delay: $T_{clk} = \max(180, 250, 150, 170, 250) + 10 = 250 + 10 = 260$ ns.
2. For $n = 1000$ instructions through a $k = 5$-stage pipeline with no stalls, the total number of clock cycles is $k + (n - 1) = 5 + 999 = 1004$ cycles.
3. Total time $= 1004 \times 260$ ns $= 261040$ ns.
4. Converting to microseconds: 261040 ns $= 261.04$ μs.

KEY IDEA

In a synchronous pipeline, the clock cycle time must be long enough for the slowest stage plus the inter-stage latch overhead: $T_{clk} = \max(\text{stage delays}) + t_{latch}$. For n instructions through a k-stage pipeline with no stalls, the total number of cycles is $k + (n - 1)$ (fill the pipeline in k cycles for the first instruction, then one more cycle per remaining instruction).

WATCH OUT FOR

A common error is using $n \times k$ cycles (as if every instruction needed all k cycles independently) instead of the correct $k + (n - 1)$ pipelined cycle count, or forgetting to add the latch delay to the slowest stage when computing the clock period.

Q.57 NAT · 2 MARKS

Data Structures and Algorithms › Trees & heaps

In a B$^+$-tree where each node can hold at most four key values, a root to leaf path consists of the following nodes:

$$A = (49, 77, 83, -), \quad B = (7, 19, 33, 44), \quad C = (20^*, 22^*, 25^*, 26^*)$$

The *-marked keys signify that these are data entries in a leaf.

Assume that a pointer between keys k_1 and k_2 points to a subtree containing keys in $[k_1, k_2)$, and that when a leaf is created, the smallest key in it is copied up into its parent.

A record with key value 23 is inserted into the B$^+$-tree.

The smallest key value in the parent of the leaf that contains 25* is _____. (Answer in integer)

Official answer: **33**

SOLUTION

1. Determine the tree structure from the given root-to-leaf path. Root A = (49, 77, 83): all of B's keys (7, 19, 33, 44) are < 49, so B is A's leftmost child (the subtree for keys < 49).
2. Within B = (7, 19, 33, 44), C's keys (20, 22, 25, 26) all lie in [19, 33), so C is B's child pointer between keys 19 and 33.
3. Insert key 23 into leaf C. Sorted, the leaf becomes (20, 22, 23, 25, 26): 5 keys, which exceeds the maximum of 4, so the leaf overflows and must split.
4. Leaf split (order 4, so 5 entries split as $\lceil 5/2 \rceil = 3$ and $\lfloor 5/2 \rfloor = 2$): the left leaf keeps (20, 22, 23); the right leaf keeps (25, 26). Per the stated rule for a newly created leaf, the smallest key of the new right leaf, 25, is copied up into the parent B (the original leaf's key stays as-is; only the new leaf's smallest key is inserted into the parent).
5. Insert 25 into B in the slot that separates the two split leaves, i.e. between 19 and 33: B becomes (7, 19, 25, 33, 44), 5 keys, exceeding the maximum of 4, so B (an internal node) also overflows and must split.
6. Internal-node split (5 keys: 7, 19, 25, 33, 44): the middle key, 25, is pushed up into the parent A and removed from B (an internal-node split pushes the middle key up rather than copying it, unlike a leaf split). The left node keeps (7, 19); the right node keeps (33, 44).
7. The right half of the split leaf, (25∗, 26∗), is attached as the child of the new right internal node (the one carrying keys 33, 44), since its keys start at 25.
8. The parent of the leaf (25∗, 26∗) is therefore the node with keys (33, 44); its smallest key is 33.

KEY IDEA

Trace the B+-tree insertion as a cascading split: first split the overflowing leaf (copy up the new right leaf's smallest key), then check whether that insertion overflows the parent and split it too (push up the middle key), and finally identify which of the resulting internal nodes the new leaf actually hangs from. The answer is the smallest key in *that* node, not in the original parent B.

WATCH OUT FOR

Stopping after the leaf split and reporting the value copied up into the (pre-split) parent B, e.g. giving 25, or giving 19 by keeping the leaf attached to the original left-hand side, instead of continuing to split the overflowing internal node B and re-checking which of the two resulting nodes (7,19) or (33,44) the (25∗,26∗) leaf now belongs to.

Q.58 NAT · 2 MARKS

Operating Systems › Memory management

A computer system supports a logical address space of 2^{32} bytes. It uses two-level hierarchical paging with a page size of 4096 bytes. A logical address is divided into a b-bit index to the outer page table, an offset within the page of the inner page table, and an offset within the desired page. Each entry of the inner page table uses eight bytes. All the pages in the system have the same size.

The value of b is _____ . (Answer in integer)

Official answer: **11**

SOLUTION

1. The logical address space is 2^{32} bytes, so a logical address has 32 bits.
2. The page size is $4096 = 2^{12}$ bytes, so the offset within the desired page takes 12 bits.
3. All pages in the system have the same size, so each page of the inner page table is also 4096 bytes.
4. Each inner page table entry is $8 = 2^3$ bytes, so one inner-table page holds $\frac{4096}{8} = 512 = 2^9$ entries. Indexing into it takes 9 bits.

5. The remaining high-order bits index the outer page table: $b = 32 - 9 - 12 = 11$.
6. Check: $11 + 9 + 12 = 32$ bits, so the outer page table has 2^{11} entries.

KEY IDEA
The three fields of the address must add up to 32 bits. The two lower fields come from the page size and from how many 8-byte entries fit in one page. The outer index takes whatever bits remain.

WATCH OUT FOR
Taking the inner-table index as the offset within a page, or ignoring the 8-byte entry size. Another slip is using the 4096-byte page size directly as the number of entries, which gives $b = 8$ instead of 11.

Q.59 NAT · 2 MARKS

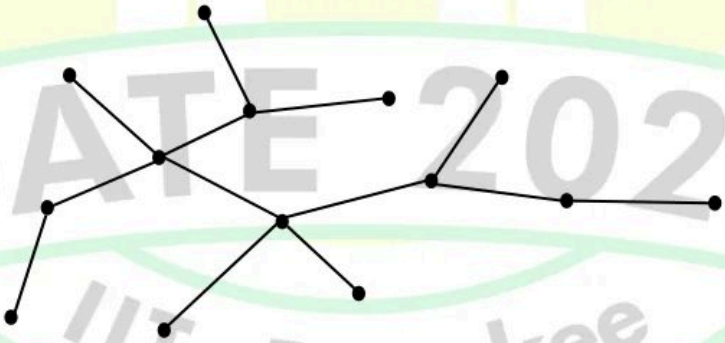
Data Structures and Algorithms › Graph algorithms

Q.59 Consider the following algorithm `someAlgo` that takes an undirected graph G as input.

```
someAlgo(G)

1. Let v be any vertex in G. Run BFS on G starting at
   v. Let u be a vertex in G at maximum distance from
   v as given by the BFS.
2. Run BFS on G again with u as the starting vertex.
   Let z be the vertex at maximum distance from u as
   given by the BFS.
3. Output the distance between u and z in G.
```

The output of `someAlgo(T)` for the tree shown in the given figure is _____. (Answer in integer)



Official answer: **6**

SOLUTION

1. someAlgo is the classical double-BFS algorithm for a tree's diameter: BFS from any vertex v finds a vertex u that is a guaranteed endpoint of some longest path (diameter path) in the tree; BFS from u then finds the true farthest vertex z, and the reported distance u-z equals the diameter of T. This is a proven property specific to trees.
2. Read the tree T from the figure: two degree-4 hub vertices P and Q are joined by a single edge. P also connects to a leaf L1, to a degree-3 vertex M that itself branches to two leaves T1 and R1, and to a 2-

edge chain P-L2-L3 ending in leaf L3. Q also connects to two leaves B1 and B2, and to a chain Q-R2-R3-R4 where R2 additionally branches to a leaf T2 and R3 is degree 2.

3. Because the whole tree hangs off the single bridge edge P-Q, the diameter equals (farthest reach from P, not through Q) + 1 (the P-Q edge) + (farthest reach from Q, not through P).
4. Farthest reach from P away from Q: L1 is at distance 1; the M-branch reaches T1 or R1 at distance 2 (P-M-T1); the L2 chain reaches L3 at distance 2 (P-L2-L3). The maximum is 2.
5. Farthest reach from Q away from P: B1 and B2 are at distance 1; the R2 branch reaches T2 at distance 2 (Q-R2-T2); the chain Q-R2-R3-R4 reaches R4 at distance 3. The maximum is 3.
6. Diameter = 2 (P side) + 1 (P-Q edge) + 3 (Q side) = 6, realized for example by the path T1-M-P-Q-R2-R3-R4 (6 edges).
7. Since double-BFS is guaranteed to output the diameter of a tree, someAlgo(T) outputs 6.

KEY IDEA

The double-BFS algorithm (BFS from an arbitrary vertex to find one diameter endpoint, then BFS from that endpoint to find the other) correctly computes the diameter of a tree, a classical, provable result that fails for general graphs but is guaranteed for trees.

WATCH OUT FOR

Treating the two hub vertices' branch depths asymmetrically or forgetting to add the connecting P-Q edge itself (giving 5 instead of 6), or missing the T2 branch off R2 and wrongly taking the Q-side depth as only 2 (via B1/B2) instead of 3 (via R2-R3-R4).

Q.60 NAT · 2 MARKS

Theory of Computation › Finite automata

Let $\Sigma = \{1, 2, 3, 4\}$. For $x \in \Sigma^*$, let $prod(x)$ be the product of symbols in x modulo 7. We take $prod(\epsilon) = 1$, where ϵ is the null string.

For example, $prod(124) = (1 \times 2 \times 4) \bmod 7 = 1$.

Define $L = \{x \in \Sigma^* \mid prod(x) = 2\}$.

The number of states in a minimum state DFA for L is _____. (Answer in integer)

Official answer: **6**

SOLUTION

1. After reading a string x, the only thing that matters for the future is $prod(x) \bmod 7$. Take one state q_r for each residue r. The start state is q_1, since $prod(\varepsilon) = 1$. The accepting state is q_2.
2. The transitions are $\delta(q_r, a) = q_{(r \cdot a) \bmod 7}$ for $a \in \{1, 2, 3, 4\}$. No symbol is a multiple of 7, and 7 is prime, so the residue 0 is never reached. No q_0 and no dead state is needed.
3. Reachability from q_1: reading $1, 2, 3, 4$ gives residues $1, 2, 3, 4$. Then $3 \cdot 4 = 12 \equiv 5$ and $2 \cdot 3 = 6$. So all of $\{1, 2, 3, 4, 5, 6\}$ are reachable, and no state is unreachable.
4. Distinguishability: take residues $r \neq s$ in $\{1, \ldots, 6\}$. Every nonzero residue modulo 7 has an inverse. Every nonzero residue is the product of some string over $\{1, 2, 3, 4\}$ (the empty string gives 1, and $5 = 3 \cdot 4$, $6 = 2 \cdot 3$). So pick a suffix w with $prod(w) \equiv 2r^{-1} \pmod 7$.
5. Then $r \cdot prod(w) \equiv 2$, so a string in state q_r followed by w is accepted. For s, $s \cdot prod(w) \equiv 2sr^{-1} \not\equiv 2$, because $s \not\equiv r$. So that string is rejected, and q_r and q_s are distinguishable.
6. All 6 states are reachable and pairwise distinguishable, so none can be merged. The minimum DFA has 6 states.

KEY IDEA

The state of the DFA is the running product modulo 7. Because 7 is prime and the symbols 1 to 4 generate every nonzero residue, all six nonzero residues are reachable and pairwise distinguishable.

WATCH OUT FOR

Counting a state for residue 0 or a dead state (giving 7 or 8). Residue 0 can never occur, since no symbol is a multiple of 7.

Q.61 NAT · 2 MARKS

Computer Organization and Architecture › Pipelining

Q.61 An application executes 6.4×10^8 number of instructions in 6.3 seconds. There are four types of instructions, the details of which are given in the table. The duration of a clock cycle in nanoseconds is _____. (rounded off to one decimal place)

Instruction type	Clock cycles required per instruction (CPI)	Number of instructions executed
Branch	2	2.25×10^8
Load	5	1.20×10^8
Store	4	1.65×10^8
Arithmetic	3	1.30×10^8

Official answer: **3.0**

SOLUTION

1. Total clock cycles = Σ(CPI_i × instructions_i) over the four instruction types.
2. Branch: $2 \times 2.25 \times 10^8 = 4.50 \times 10^8$ cycles.
3. Load: $5 \times 1.20 \times 10^8 = 6.00 \times 10^8$ cycles.
4. Store: $4 \times 1.65 \times 10^8 = 6.60 \times 10^8$ cycles.
5. Arithmetic: $3 \times 1.30 \times 10^8 = 3.90 \times 10^8$ cycles.
6. Total cycles $= 4.50 + 6.00 + 6.60 + 3.90 = 21.00 \times 10^8 = 2.1 \times 10^9$ cycles.
7. Total execution time $= 6.3\ \text{s} = 6.3 \times 10^9\ \text{ns}$.
8. Clock cycle time $= \dfrac{\text{Total time}}{\text{Total cycles}} = \dfrac{6.3 \times 10^9\ \text{ns}}{2.1 \times 10^9} = 3.0\ \text{ns}$.

KEY IDEA

Total clock cycles = Σ(CPI_i × count_i) for each instruction class; total execution time = total cycles × clock cycle time, so clock cycle time = total execution time / total clock cycles.

WATCH OUT FOR

Dividing the given total time (6.3 s) by the given total instruction count (6.4×10^8) to get an "average time per instruction" instead of first computing total clock cycles from the CPI table and dividing time by cycles. This conflates cycles with instructions and ignores the CPI values entirely.

Q.62 NAT · 2 MARKS

Programming in C › Pointers & arrays

Consider the following C program:

```c
#include <stdio.h>
int main(){
   int a;
   int arr[5] = {30,50,10};
   int *ptr;
   ptr = &arr[0] + 1;
   a = *ptr;
   (*ptr)++;
   ptr++;
   printf("%d", a + (*ptr) + arr[1]);
   return 0;
}
```

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

Official answer: **111**

SOLUTION

1. **Initialize the array.** int arr[5] = {30,50,10}; explicitly sets arr[0]=30, arr[1]=50, arr[2]=10; C zero-fills the remaining elements, so arr[3]=0, arr[4]=0.
2. **Compute the pointer.** ptr = &arr[0] + 1; : pointer arithmetic advances by one int per unit (not one byte), so ptr now equals &arr[1] , i.e., it points at the element holding 50.
3. **Read through the pointer into a .** a = *ptr; copies the current value at that address into a , so a = 50 . This is a snapshot: later changes to *ptr do not retroactively change a .
4. **Mutate through the pointer.** (*ptr)++; increments the object ptr points to. Since ptr currently aliases arr[1] , this changes arr[1] itself, from 50 to 51. (a , already assigned in the previous step, is unaffected.)
5. **Advance the pointer.** ptr++; moves ptr to the next int , so it now points at arr[2] , whose value is 10 (never modified).
6. **Evaluate the printf arguments.** At this point: a = 50 , *ptr = arr[2] = 10 , arr[1] = 51 (post-increment). So the expression a + (*ptr) + arr[1] = 50 + 10 + 51 .
7. **Sum and print.** 50 + 10 + 51 = 111 . The program prints 111 .

KEY IDEA

Partial array initializers zero-fill the remaining elements; pointer arithmetic (`&arr[0] + 1`) moves by one element's width, not one byte; and `*ptr` and `arr[1]` refer to the *same* memory once `ptr` points into `arr[1]` , so a change through one is visible through the other.

WATCH OUT FOR

Treating `a` , `*ptr` , and `arr[1]` as three independent values once `(*ptr)++` fires, instead of recognizing `ptr` aliases `arr[1]` at that point (so the increment changes `arr[1]` itself), or forgetting that `a` already captured the old value of 50 before the increment happened, and wrongly using the post-increment value for `a` too.

Q.63 NAT · 2 MARKS

Programming in C › Functions & scope

Consider the following C program:

```
#include <stdio.h>

int g(int n) {
   return (n+10);
}

int f(int n) {
   return g(n*2);
}

int main() {
   int sum, n;
   sum=0;
   for (n=1; n<3; n++)
      sum += g(f(n));
   printf ("%d", sum);
   return 0;
}
```

The output of the given C program is _____. (Answer in integer)

Official answer: **46**

SOLUTION

1. g(n) returns n+10, so f(n) = g(n*2) = 2n+10.
2. The for-loop runs for n=1 and n=2 (loop stops when n<3 fails, i.e. at n=3).
3. n=1: f(1) = 2(1)+10 = 12. g(f(1)) = g(12) = 12+10 = 22. sum = 0+22 = 22.
4. n=2: f(2) = 2(2)+10 = 14. g(f(2)) = g(14) = 14+10 = 24. sum = 22+24 = 46.
5. The loop ends; printf outputs the value of sum, which is 46.

KEY IDEA

Trace nested function calls exactly in evaluation order: `g(f(n))` first fully evaluates `f(n)`, which itself calls `g(n*2)`, before the outer `g` is applied to that result.

WATCH OUT FOR

Applying `g` only once per loop iteration (forgetting that `f(n)` internally already calls `g`), e.g. computing `sum += n*2+10` instead of `sum += g(f(n))` = `(n*2+10)+10`, which gives 36 instead of 46.

Q.64 NAT · 2 MARKS

Discrete Mathematics › Counting

A quadratic polynomial $(x-\alpha)(x-\beta)$ over complex numbers is said to be square invariant if $(x-\alpha)(x-\beta) = (x-\alpha^2)(x-\beta^2)$. Suppose from the set of all square invariant quadratic polynomials we choose one at random.

The probability that the roots of the chosen polynomial are equal is _____. (rounded off to one decimal place)

Official answer: **0.5**

SOLUTION

1. Two monic quadratics are equal exactly when their root multisets are equal, so square invariance means $\{\alpha, \beta\} = \{\alpha^2, \beta^2\}$ as multisets.
2. Matching 1: $\alpha^2 = \alpha$ and $\beta^2 = \beta$. Then $\alpha, \beta \in \{0, 1\}$, giving the polynomials x^2, $x(x-1)$ and $(x-1)^2$.
3. Matching 2: $\alpha^2 = \beta$ and $\beta^2 = \alpha$. Then $\alpha^4 = \alpha$, so $\alpha(\alpha^3 - 1) = 0$ and $\alpha \in \{0, 1, \omega, \omega^2\}$, where ω is a primitive cube root of unity.

4. For $\alpha = 0$ or 1 this gives $\beta = \alpha$, already counted. For $\alpha = \omega$ we get $\beta = \omega^2$, and $\alpha = \omega^2$ gives $\beta = \omega$. Both give the same polynomial $(x - \omega)(x - \omega^2) = x^2 + x + 1$. Check: the squares of its roots are ω^2 and $\omega^4 = \omega$, the same pair.
5. So there are exactly 4 distinct square invariant polynomials: x^2, $x^2 - x$, $x^2 - 2x + 1$ and $x^2 + x + 1$.
6. Equal roots occur for x^2 (roots $0, 0$) and $(x - 1)^2$ (roots $1, 1$). That is 2 of the 4. The other two have distinct roots ($0, 1$ and ω, ω^2).
7. Probability $= 2/4 = 0.5$.

KEY IDEA
Square invariance means the root multiset is unchanged by squaring, so the roots are either each fixed by squaring (0 or 1) or swapped by squaring (the two primitive cube roots of unity). List the polynomials and count.

WATCH OUT FOR
Counting only the fixed-point case and forgetting that the two roots can be swapped by squaring (ω and ω^2). That gives 3 polynomials and the wrong probability $2/3$.

Q.65 NAT · 2 MARKS

Engineering Mathematics › Probability & statistics

The unit interval $(0, 1)$ is divided at a point chosen uniformly distributed over $(0, 1)$ in $\mathbb{R}$ into two disjoint subintervals.

The expected length of the subinterval that contains 0.4 is _____. (rounded off to two decimal places)

Official answer: **0.70 to 0.80**

SOLUTION

1. Let the cut point be $X \sim U(0, 1)$ with density $f(x) = 1$ on $(0, 1)$. The two pieces are $(0, X)$ and $(X, 1)$.
2. If $X > 0.4$, the point 0.4 lies in $(0, X)$, so the length is $L = X$.
3. If $X < 0.4$, the point 0.4 lies in $(X, 1)$, so the length is $L = 1 - X$.
4. Split the expectation at 0.4:
$$E[L] = \int_0^{0.4} (1 - x)\, dx + \int_{0.4}^{1} x\, dx$$
5. First integral: $\left[x - \frac{x^2}{2}\right]_0^{0.4} = 0.4 - 0.08 = 0.32$.
6. Second integral: $\left[\frac{x^2}{2}\right]_{0.4}^{1} = 0.5 - 0.08 = 0.42$.
7. Add the two parts: $E[L] = 0.32 + 0.42 = 0.74$. As a check, for a general point p the result is $\frac{1}{2} + p(1 - p)$, which at $p = 0.4$ gives $0.5 + 0.24 = 0.74$.

KEY IDEA
The length of the piece containing a fixed point depends on which side of that point the random cut falls. Split the expectation at the point and integrate each case. In general the answer is $\frac{1}{2} + p(1 - p)$.

WATCH OUT FOR
Taking the expected length of a randomly chosen piece, which is 0.5. The piece containing a fixed point is length-biased toward the longer piece, so its expected length is larger than 0.5.

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 2025 Set 2 question paper, organised by IIT Roorkee; 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 Roorkee 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