SPRING END SEMESTER EXAMINATION-2018

4th Semester B.Tech & B.Tech Dual Degree

Design & Analysis of Algorithms

CS-2008

[For 2017 (LE), 2016 & Previous Admitted Batches]

Time: 3 Hours **Full Marks: 60**

Answer any six questions including question No.1 which is compulsory.
The figures in the margin indicate full marks.
Candidates are required to give their answers in their own words as far as practicable and all parts of a question should be answered at one place only.

DAA SAMPLE ANSWER(S) & EVALUATION SCHEME

Q1 Answer the following questions: [2 x 10]

a) Write the time complexity for finding the 3rd maximum in an (unordered) array of n distinct elements.

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

n

b) Write the time complexity for addition of two matrices of size m x n each.

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

m x n

c) Write the asymptotic tight bound of n log(n!)

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

$n^2 \log n$

d) A[1..n] is a list of n distinct numbers. Given that x belongs to A, $\sum_{i=1}^{n}|A[i]-x|$ is minimumif x is : (a) mean (b) median (c) some element of A between (and including) mean and median (d) not necessarily any of these

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

(b) median

e) Define binary max heap. Name an operation which is efficient on a binary max heap.

Evaluation Scheme:

- Defining max. heap : 1 Mark
- Efficient operation name: 1 Mark

Answer:

Binary Max. Heap

- A binary heap is a heap data structure that takes the form of a binary tree.
- A binary heap is defined as a binary tree with two additional constraints:
  a) **Shape property:** a binary heap is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
  b) **Heap property:** the key stored in each node is either greater than or equal to (≥) or less than or equal to (≤) the keys in the node's children, according to some total order.
- Heaps where the parent key is greater than or equal to (≥) the child keys are called **max-heaps**; those where it is less than or equal to (≤) are called **min-heaps.**

Name of Operations: Getting largest element/ Insertion & Deletion

f) Minimum number of comparison required to merge two sorted files of size m and n is MIN(m, n). How many minimum comparisons are required to merge 3 sorted files of size 10, 20, and 30

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

30

g) Write the worst-case and the average-case time complexities of heap sort.

Evaluation Scheme:

- Correct Answer : 2 Marks
- Wrong Answer : Zero

Answer:

n log n

h) Write the space complexities for adjacency-list representation and adjacency-matrix representation of an arbitrary tree having n nodes.

Evaluation Scheme:

- Each Correct Answer : 1 Mark
- Wrong Answer : Zero

Answer:

Space complexity for adjacency-list representation: O(n)

Space complexity for adjacency-matrix representation: $O(n^2)$

i) **Which one among the following problems is easiest and which one hardest (measured in terms of worst-case time complexity of the best-known algorithm)?**

P1: Finding the maximum in an unordered list.

P2: Finding the minimum in an unordered list.

P3: Searching a key in an ordered list..

P4: Searching a key in an unordered list.

P5: Finding a pair of elements in an unordered list such that their difference is minimum.

P6: Partitioning an unordered list into sub-lists such that:

-they are of equal size or differs in size by at most one;

-each element in the first sub-list is less than eachelement in the second sub-list.

(Assume that every time the list is implemented by a 1-D array and contains n distinct elements

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

P2: Finding the minimum in an unordered list (easiest)

P5: Finding a pair of elements in an unordered list such that their difference is minimum. (hardest)

j) **Which of the following problems is/are NP-Complete?**

P1 : Matrix Chain Multiplication Problem

P2: Minimum Spanning Tree Problem

P3: Circuit Satisfiability Problem

P4: Binary Knapsack Problem

P5: Traveling Salesman Problem

Pb: Median Finding Problem

Evaluation Scheme:

- Correct Answer : 2 mark
- Wrong Answer : Zero

Answer:

P3: Circuit Satisfiability Problem

P4: Binary Knapsack Problem

P5: Traveling Salesman Problem

Q2 a) Deduce the asymptotic tight bound, if possible, for **[4]**

$$T(n) = \begin{cases} \Theta(1) \text{ if } n < 40, \\ 4T(n/5) + n \quad \text{otherwise.} \end{cases}$$

If asymptotic tight bound is not possible, then justify the reason and deduce the upper asymptotic bound.

Evaluation Scheme:

- Correct answer with valid steps: 4 Marks
- Incorrect answer with some valid steps: 1-3 Marks

Answer:

$T(n) = \Theta(n)$

Explanation

Applying master theorem

a=4, b=5, f(n)=n

Step-1(Guess)

$n^{\log}b^{a}=n^{\log}5^{4}$

Comparing $n^{\log_b a}$ with f(n), f(n) is found asymptotically larger than $n^{\log_b a}$. So case-3 of master theorem is guessed.

Step-2 (Verify)

As per case-3 of master theorem,

Let $f(n)=\Omega(n^{\log_b a+\epsilon})$ is true

$\Rightarrow f(n) \geq c.\ n^{\log_b a+\epsilon}$

$\Rightarrow n \geq c.n^{\log_5 4+\epsilon}$

$\Rightarrow n \geq c.n^{\log_5 4} \times n^{\epsilon}$

$\Rightarrow \epsilon < 1$ the above inequality is valid with c=1. As per case-3 of master theorem

$af(n/b) \leq cf(n)$ must valid

$\Rightarrow 4 \times n/5 \leq c\ n$

$\Rightarrow c \geq 0.8$ (True as c is a valid constant < 1)

So the solution is $T(n) = \Theta(f(n) = \Theta(n)$

b) State and explain Master's Method and use this method to give tight asymptotic bounds for the recurrence $T(n) = 4T(n/2) + n^3$. [4]

Evaluation Scheme:

- State & Explain Master Theorem: 2 Marks
- Solving the recurrence: 2 Marks

Answer:

$T(n) = \Theta(f(n) = \Theta(n^3)$

Explanation

<table>
<tr><th colspan="2">Type:-1 (Master Theorem as per CLRS)</th></tr>
<tr><th>Master Theorem</th><th>Solution of recurrence
$T(n) = 4T(n/2) + n^3$</th></tr>
<tr><td>The Master Theorem applies to recurrences of the following form:
$T(n) = aT(n/b) + f(n)$
where $a \geq 1$ and $b > 1$ are constants and f(n) is an asymptotically positive function. T(n) is defined on the non-negative integers by the recurrence.
T(n) can be bounded asymptotically as follows: There are 3 cases:
a) Case-1: If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$,
then $\mathbf{T(n) = \Theta(n^{\log_b a})}$
b) Case- 2: If $f(n) = \Theta(n^{\log_b a})$, then $\mathbf{T(n) = \Theta(n^{\log_b a} \log n)}$
c) Case-3: If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and : $af(n/b) \leq cf(n)$, then $T(n) = \Theta(f(n))$, for some constant $c < 1$ and all sufficiently large n, then $\mathbf{T(n) = \Theta(f(n))}$</td><td>Given, a=4, b=2, $f(n)=n^3$
<u>Step-1 (Guess)</u>
$n^{\log_b a}=n^{\log_2 4}=n^2$, Comparing $n^{\log_b a}$ with f(n), f(n) is found asymptotically larger than $n^{\log_b a}$. So case-3 of master theorem is guessed.

<u>Step-2 (Verify)</u>
As per case-3 of master theorem,
Let $f(n)=\Omega(n^{\log_b a+\epsilon})$ is true
$\Rightarrow f(n) \geq c.n^{\log_b a+\epsilon}$
$\Rightarrow n^3 \geq c.n^{2+\epsilon}$
$\Rightarrow n \geq c.n^{\epsilon}$, This inequality is valid for $c=1$ and $0<\epsilon \leq 1$. Now
$af(n/b) \leq cf(n)$ must valid
$\Rightarrow 4(n/2)^3 \leq c.n^3$
$\Rightarrow c \geq 0.5$ (True as c is a valid constant < 1)
So the solution is
$T(n) = \Theta(f(n) = \Theta(n^3)$</td></tr>
<tr><th colspan="2">Type:-2 (Master Theorem)</th></tr>
<tr><th>Master Theorem</th><th>Solution of recurrence
$T(n) = 4T(n/2) + n^3$</th></tr>
<tr><td>If the recurrence is of the form
$T(n) = aT(n/b) + n^k \log^p n$,
where $a \geq 1$, $b > 1$, $k \geq 0$ and p is a real number, then comapre a with b^k and conclude the solution as per the following cases.

Case-1: If $a > b^k$, then $T(n) = \Theta(n^{\log_b a})$

Case-2: If $a = b^k$, then
a) If $p > -1$, then
$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$
b) If $p = -1$, then
$T(n) = \Theta(n^{\log_b a} \log\log n)$
c) If $p < -1$, then $T(n) = \Theta(n^{\log_b a})$

Case-3: If $a < b^k$, then
a) If $p \geq 0$, then $T(n) = \Theta(n^k \log^p n)$
b) If $p < 0$, then $T(n) = \Theta(n^k)$</td><td>Here, a=4, b=2, k=3, p=0
$b^k = 2^3 = 8$
Comparing a with b^k, we found a is less than b^k, so this will fit to case-3.
Now p=0, so case-3.a solution is the recurrence solution.
$T(n)= \Theta(n^3 \log^0 n)=\Theta(n^3)$</td></tr>
</table>

Q3 a) Derive the average-case time complexity for linear search in an unordered list. [4]

Evaluation Scheme:

- Explanation of average case time complexity : 2 Marks.
- Derivation of answer: 2 Marks

Answer:

Average Case Analysis of Linear Search

In average case analysis, we take all possible inputs and calculate computing time for all of the inputs. Sum all the calculated values and divide the sum by total number of inputs. We must know (or predict) distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in array). So we sum all the cases and divide the sum by (n+1). Following is the value of average case time complexity.

$$\text{Average Case Time} = \frac{\sum_{i=1}^{n+1} \theta(i)}{(n+1)} = (1+2+3+4+\ldots\ldots.+n+n+1)/(n+1)=\Theta(n)$$

Where, θ(i) is the number of comparision required to find an element x found in i^{th} position of array.

b) Write the algorithm for insertion sort and use step count method to analyze the time complexities of Insertion Sort. **[4]**

Evaluation Scheme:

- Insertion Algorithm: 1.5 Marks
- Explanation through step count method : 2.5 Marks

Answer:

Line No.	Insertion Sort Algorithm	Cost	Times
1	INSERTION-SORT(A)	0	
2	{	0	
3	for j←2 to length[A]	c1	n
4	{	0	
5	key←A[j]	c2	n-1
6	//Insert A[j] into the sorted sequence A[1..j-1]	0	
7	i←j-1	c3	n-1
8	while(i>0 and A[i]>key)	c4	$\sum_{j=2}^{n} tj$
9	{	0	
10	A[i+1]←A[i]	c5	$\sum_{j=2}^{n} (tj-1)$
11	i←i-1	c6	$\sum_{j=2}^{n} (tj-1)$
12	}	0	
13	A[i+1]←key	c7	n-1
14	}	0	
15	}	0	

To compute T(n), the running time of INSERTION-SORT on an input of n values, we sum the products of the cost and times column, obtaining

$T(n)=c1n + c2(n-1) + c3(n-1) + c4\sum_{j=2}^{n} tj + c5\sum_{j=2}^{n} (tj - 1) + c6\sum_{j=2}^{n} (tj - 1) + c7(n-1)$ ..... (1)

Best case Analysis:

- Best case occurs if the array is already sorted.
- For each j=2 to n, we find that A[i]≤key in line number 8 when i has its initial value of j-1. Thus tj=1 for j=2 to n and the best case running time is $T(n)=c1n + c2(n-1) + c3(n-1) + c4(n-1) + c7(n-1) = O(n)$

Worst case Analysis:

- Worst case occurs if the array is already sorted in reverse order
- In this case, we compare each element A[j] with each element in the entire sorted subarray A[1..j-1], so tj=j for j=2 to n.
- The worst case running time is

$T(n)=c1n + c2(n-1) + c3(n-1) + c4(n(n+1)/2-1) + c5(n(n-1)/2) + c6(n(n-1)/2) + c7(n-1)=O(n^2)$

Q4 **Given a set of n intervals, $R = \{I_I, 1_2, I_3, ..., I_n\}$ where each $I_j = [a_j, b_j]$ is an interval from the real line having length $b_j - a_j$. The objective is to find the largest set of intervals $S \subseteq R$ such that no two intervals in S overlap each other and the sum of lengths of the intervals in S is maximized. Suggest an efficient algorithm for this and derive its time complexity.** **[5+3]**

Evaluation Scheme:

- D
- M

Answer:

Q5 **3n distinct elements are given in a one-dimensional array A. Suggest an efficient algorithm to partition A into B, C, D, such that i) each of B, C, D contains n elements, ii) each element of B is less than each element of C, and each element of C is less than each element of D. Derive its time complexity.** **[5+3]**

Evaluation Scheme:

- Correct Algorithm that runs in O(n) time: 8 Marks
- Correct Algorithm that runs in O(nlog n) time: 5 Marks
- Algorithm correct or some valid steps,
  runs other than these above times: 1-4 Marks

Answer:

```
/*Array A is having m=3n elements, and each array of B, C & D, contain n elements.
Indexes are 1, 2, 3….m/3, m/3+1,……2m/3, 2m/2+1,……..m*/
PARTITION-FUN(A, m, B, C, D, n)
{
  /*Finding n^th smallest element in an array of 3n elements in O(n) time*/
  fPV=KTH-SMALLEST(A, m, n);
 /*Finding n^th largest element in an array of 3n elements in O(n) time*/
  sPV=KTH-LARGEST(A, m, n);
  /*Three way partitioning of an array around a given range fPV and sPV in O(n) time*/
   THREE-WAY-PARTITION(A, m, fPV, sPV);
   p=q=r=1;
   /*Transfering all elements of array A into B, C & D*/
  for (i=1; i<=m; i++)
  {
      if (A[i]<=fPV)
              B[p++]=A[i];
      else if (A[i]<=sPV)
              C[q++]=A[i];
       else   D[r++]=A[i];
  }
}
```

OR

```
/*Array A is having m=3n elements, and each array of B, C & D, contain n elements.
Indexes are 1, 2, 3….m/3, m/3+1,……2m/3, 2m/2+1,……..m*/
PARTITION-FUN(A, lb, ub, B, C, D)
{
  /*If lb>m/2 && ub<2m/3, then stop, else continue*/
  if(lb>m/2 && ub<2m/3) return;
  pivot=PARTITION(A, lb, ub);
  if (pivot==m/3)
```

```
        PARTITION(A, pivot+1, ub);
    else if (pivot==2m/3)
        PARTITION-FUN(A, lb, pivot-1);
    else
    {
        PARTITION-FUN(A, lb, pivot-1);
        PARTITION-FUN(A, pivot+1,ub);
    }
}

TRANSFER-ARRAY(A, m, B, C, D, n)
{
   p=q=r=1;
   /*Transfering all elements of array A into B, C & D*/
  for (i=1; i<=m; i++)
  {
      if (i<=m/3)
              B[p++]=A[i];
      else if (i<=2m/3)
              C[q++]=A[i];
       else   D[r++]=A[i];
  }
}
```

The time Complexity of the above algorithm is O(n).

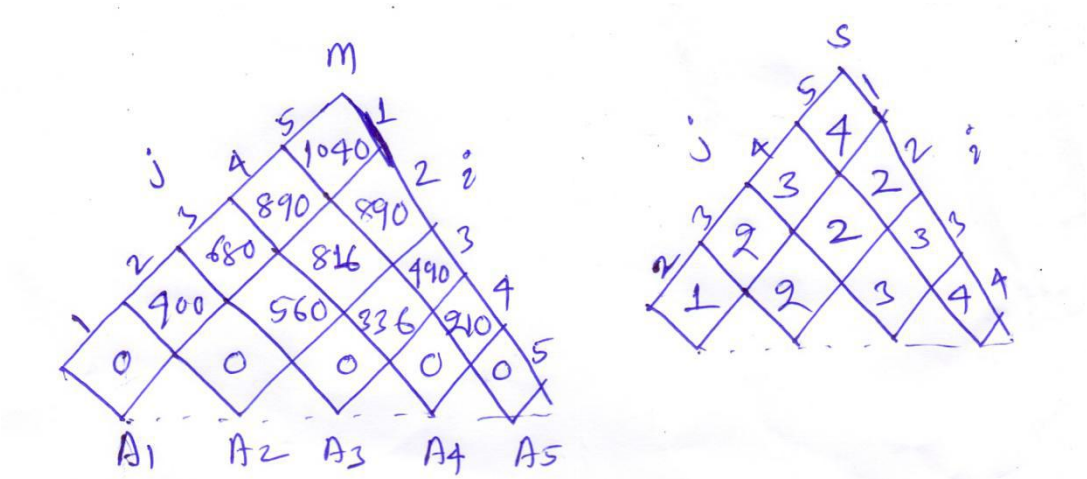
Q6 a) Find the optimal parenthesization of a Matrix-Chain product for minimizing total number of scalar multiplications, whose sequence of dimensions is <5x10, 10x8, 8x7, 7x6, 6x5 >. **[4]**

Evaluation Scheme:

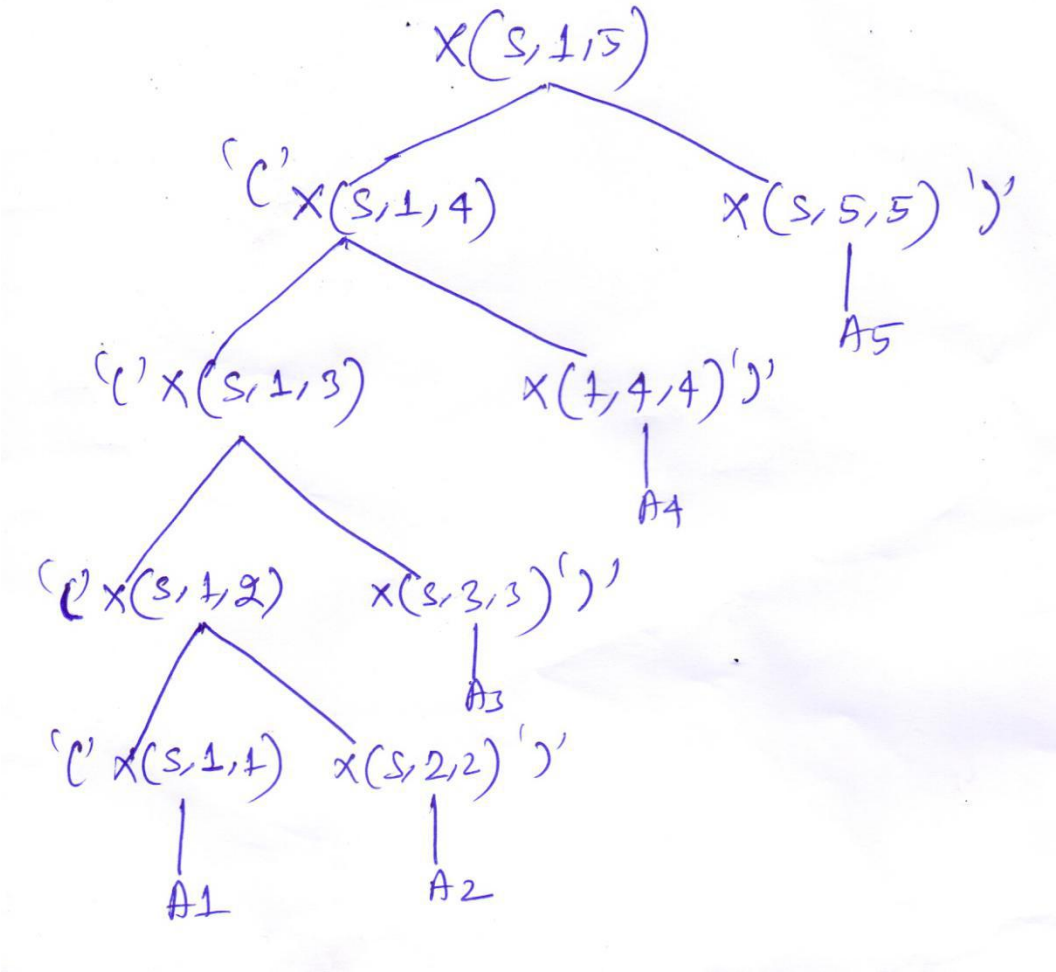
- Construction of m and s table with correct data: 4 Marks
- Correct optimal parenthesization of a Matrix-Chain product: 1 Mark

Answer:

- The sequence of dimensions is <5, 10, 8, 7, 6, 5>
- So construct the required m and s tables are as follows:



Let the PRINT-OPTIMAL-PARENS(s, i, j) function is renamed as X(s, i, j). As there are 5 matrices, so the initial call is X(s, 1, 5). The recursive PRINT-OPTIMAL-PARENS(s, i, j) algo is representd as follows:



Referring the PRINT-OPTIMAL-PARENS algorithm through the above diagram, the optimal parenthesization of a Matrix-Chain product is drawn as follows for minimizing total number of scalar multiplications.

(((A1A2)A3)A4)A5)

Filling of m[i][j] and s[i][j] values

Step-1

m[1]2]=p0p1p2=5x10x8=400, k=1 => s[1]2]=1
m[2][3]=p1p2p3=10x8x7=560, k=2 => s[2][3]=2
m[3][4]=p2p3p4=8x7x6=336, k=3 => s[3][4]=3
m[4][5]=p3p4p5=7x6x5=210, k=4 => s[4][5]=4

Step-2

m[1][3]=?

m[1][3] = m[1][1] + m[2][3] + p0p1p3 = 0 + 560 + 5x10x7 = 560 + 350 = 910
k=1
m[1][3] = m[1][2] + m[3][3] + p0p2p3 =400 + 0 + 5x8x7 = 400 + 280 = 680 **(min)**
k=2
So m[1][3]=680 with k=2 => s[1][3]=2
…….
m[2][4]=?

m[2][4] = m[2][2] + m[3][4] + p1p2p4 = 0 + 336 + 10x8x6 = 336 + 480 = 816 **(min)**
k=2
m[2][4] = m[2][3] + m[4][4] + p1p3p4 = 560 + 0 + 10x7x6 = 560 + 420 = 980
k=3
So m[2][4]=816 with k=2 => s[2][4]=2
…….
m[3][5]=?

m[3][5] = m[3][3] + m[4][5] + p2p3p5 = 0 + 210 + 8x7x5 = 210 + 280 = 490 **(min)**
k=3
m[3][5] = m[3][4] + m[5][5] + p2p4p5 = 336 + 0 + 8x6x5 = 336 + 240 = 576
k=4
So m[3][5]=490 with k=3 => s[3][5]=3

…….
m[3][5]=?

m[3][5] = m[3][3] + m[4][5] + p2p3p5 = 0 + 210 + 8x7x5 = 210 + 280 = 490 **(min)**
k=3
m[3][5] = m[3][4] + m[5][5] + p2p4p5 = 336 + 0 + 8x6x5 = 336 + 240 = 576
k=4
So m[3][5]=490 with k=3

Step-3

m[1][4]=?

m[1][4] = m[1][1] + m[2][4] + p0p1p4 = 0 + 816 + 5x10x6 = 816 + 300 = 1116
k=1
m[1][4] = m[1][2] + m[3][4] + p0p2p4 = 400 + 336 + 5x8x6
k=2 = 400 + 336 + 240 = 976

m[1][4] = m[1][3] + m[4][4] + p0p3p4 = 680 + 0 + 5x7x6 = 680 + 210 = 890 **(min)**
k=3

So m[1][4]=890 with k=3 => s[1][4]=3

…….

m[2][5]=?

m[2][5] = m[2][2] + m[3][5] + p1p2p5 = 0 + 490 + 10x8x5= 490 + 400 = 890 **(min)**
k=2
m[2][5] = m[2][3] + m[4][5] + p1p3p5 = 560 + 210 + 10x7x5
k=3 = 560 + 210 + 350 = 1120

m[2][5] = m[2][4] + m[5][5] + p1p4p5 = 816 + 0 + 10x6x5 = 816 +300 =1116
k=4

So m[2][5]=890 with k=2 => s[2][5]=2

…….

<u>Step-4</u>

m[1][5]=?

m[1][5] = m[1][1] + m[2][5] + p0p1p5 = 0 + 890 + 5x10x5= 890 + 250 = 1140
k=1
m[1][5] = m[1][2] + m[3][5] + p0p2p5 = 400 + 490 + 5x8x5
k=2 = 890 + 250 + 200= 1090
m[1][5] = m[1][3] + m[4][5] + p0p3p5 = 680 + 210 + 5x7x5
k=3 = 680 + 210 + 175= 1065
m[1][5] = m[1][4] + m[5][5] + p0p4p5 = 890 + 0 + 5x6x5= 890 + 250 = 1040 **(min)**
k=4

So m[1][5]=1040 with k=4 => s[1][5]=4

b) Use Dynamic Programming Algorithm to find the Longest Common Subsequence of the following two sequences: [4]

X = < ababaabaa >

Y = < aababaab >

Evaluation Scheme:

- Table with correct information
  with the use of dynamic programming : 4 Marks
- Correct LCS: 1 Mark

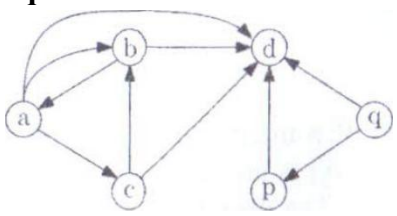
Answer:

- The LCS is **a a b a b a a**

	y_j	a	a	b	a	b	a	a	b
x_i	0	0	0	0	0	0	0	0	0
a	0	1 ↖	1 ↖	1 ←	1 ↖	1 ←	1 ↖	1 ↖	1 ←
b	0	1 ↑	1 ↑	2 ↖	2 ←	2 ↖	2 ←	2 ←	2 ↖
a	0	1 ↖	2 ↖	2 ↑	3 ↖	3 ←	3 ↖	3 ↖	3 ←
b	0	1 ↑	2 ↑	3 ↖	3 ←	4 ↖	4 ←	4 ←	4 ↖
a	0	1 ↖	2 ↖	3 ↑	4 ↖	4 ↑	5 ↖	5 ↖	4 ←
a	0	1 ↖	2 ↖	3 ↑	4 ↖	4 ↑	5 ↖	6 ↖	6 ←
b	0	1 ↑	2 ↑	3 ↖	4 ↑	5 ↖	5 ↑	6 ↑	7 ↖
a	0	1 ↖	2 ↖	3 ↑	4 ↖	5 ↑	6 ↖	6 ↖	6 ↑
a	0	1 ↖	2 ↖	3 ↑	4 ↖	5 ↑	6 ↖	7 ↖	7 ←

The LCS is aababaa

Q7 **Construct the adjacency list of the following directed graph and demonstrate the DFS (depth-first search) algorithm on it. Write the initialization and explain how the relevant parameters and data structures are updated during the execution. In the final step, you should write the DFS tree/trees, and also the forward edges, cross edges, and back edges, if any. Use node 'a' as source node while answering the question.** [5+3]



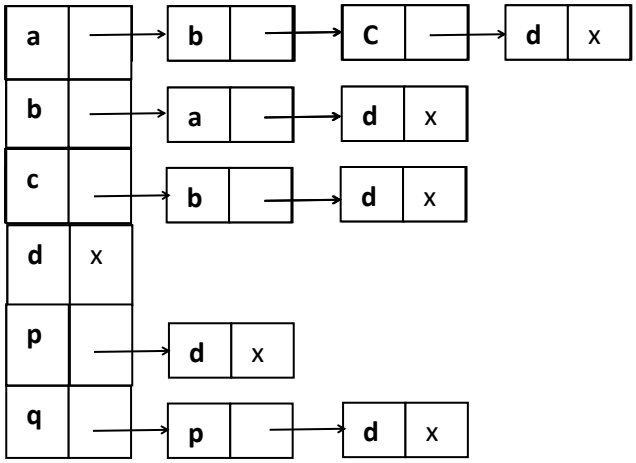
Evaluation Scheme:

- Construction of adjacency list: 2 Mark
- Explanation of DFS algorithm ended with DFS tree/trees: 4 Marks
- Distinguish the edges as tree edge/forward edge/back edge/cross edge: 2

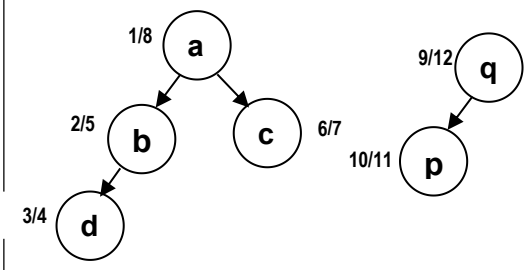
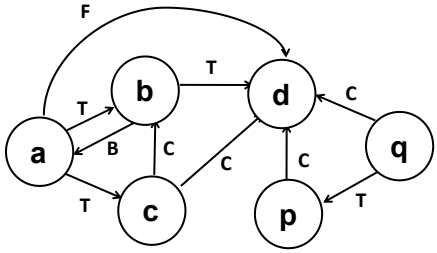
Marks

Answer:

- A graph can also be represented using alinked list. For each vertex, a list of adjacent vertices is maintained using a linked list.
- It creates a separate linked list for each vertex Vi inthe graph G = (V, E).
- Adjacency list of a graph with n nodes can be represented by an array of pointers. Each pointer points to a linked list of the corresponding vertex.
- Below figure shows the adjacency list representation of a graph.

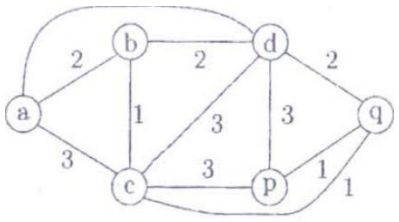


(Adjacent List)

DFS tree/forest with each node labelled with time stamp (discover time/finishing time)	DFS tree/forest with tree edge (Labelled as T) forward edge (labeled as F), back age (labeled as B) and cross edge (labeled as C)
	
DFS Sequence for this DFS tree/forest is **a, b, d, c, q, p**	

Q8 a) Construct the adjacency list of the following weighted graph and demonstrate on it the Kruskal's algorithm for minimum spanning tree. Explain how the relevant parameters and data structures are updated during the execution. In the final step, you should write the MST and its cost. Is this MST unique for this graph? [5]

Use node `a' as source node while answering the question.



Evaluation Scheme:

- Construction of adjacency list: 2 Mark
- Explanation of Kruskal's/Prim's algorithm through diagram reflected with relevant parameters and data structures are updated during the execution: 4 Marks
- Correct MST cost: 1 Mark
- Correct answer for the MST unique or not: 1 Mark

Answer:

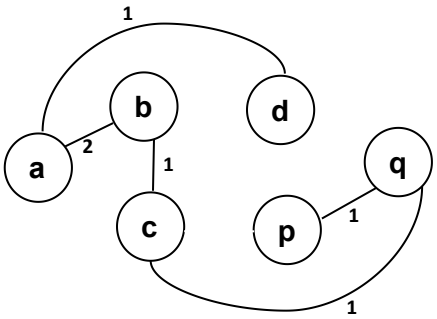
Kruskal's algorithm for Minimum Spanning Tree

- Adjacency list of the given graph is as follows:
  a → b → c → d
  b → a → c → d
  c → a → b → d → p → q
  d → a → b → c → p → q
  q → d → p
- The Cost of MST = 1+1+1+1+2=6
- The MST is not unique.

The edges of the graph in ascending order of their weight

Sl. No.	Edge	Weight	Selection
1	(a, d)	1	√
2	(b, c)	1	√
3	(c, q)	1	√
4	(p, q)	1	√
5	(a, b)	2	√
6	(b, d)	2	x
7	(d, q)	2	x
8	(a, c)	3	x
9	(c, d)	3	x
10	(c, p)	3	x
11	(p, d)	3	x

MST (Final Figure)



b) Define the classes P and NP. Mention one problem that belongs to NP but not to P. [5]

Evaluation Scheme:

- Defining the classes P and NP: 3 Marks
- Mentioning atleast one problem that belongs to NP but not to P : 2 Marks

Answer:

The classes P and NP Problems

P

- P stands for Polynomial.
- The problems that are **solvable** in polynomial time (that is the time $O(n^k)$, for some constant k, where n is the size of the input to the problem), are called class of **P problems**.

NP

- NP stands for Non-deterministic Polynomial.
- The problems that are **verifiable** in polynomial time, are called class of N**P problems**.
- Here the term verifiable mean is that if we were somehow given a certificate of a solution, then we would verify the certificate is correct in polynomial time.
- In other words, a problem is in NP if we can quickly (in polynomial time) test whether a solution is correct without worrying about how hard it might be to find the solution.
- Problems in NP are still relatively easy, if only we could guess the right solution, we could then quickly test it.
- Any problem in P is also in NP, since if a problem is in P then we can solve it in polynomial time without even being given a certificate. We believe P is a subset of NP.

Example

- Determining whether a directed graph has a Hamiltonian cycle is NP-Complete.
- Circuit Satisfiability Problem
- Binary Knapsack Problem
- Traveling Salesman Problem

	y_j	a	a	b	a	b	a	a	b
x_i	0	0	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1	1	1
b	0	1	1	2	2	2	2	2	2
a	0	1	2	2	3	3	3	3	3
b	0	1	2	3	3	4	4	4	4
a	0	1	2	3	4	4	5	5	4
a	0	1	2	3	4	4	5	6	6
b	0	1	2	3	4	5	5	6	7
a	0	1	2	3	4	5	6	6	7
a	0	1	2	3	4	5	6	7	7

https://www.geeksforgeeks.org/three-way-partitioning-of-an-array-around-a-given-range/

https://www.geeksforgeeks.org/3-way-quicksort-dutch-national-flag/

https://www.geeksforgeeks.org/3-way-quicksort-dutch-national-flag/