Part-A-DAA_IAT-2

1. How dynamic programming is used to solve knapsack problem?

The basic idea of Knapsack dynamic programming is to use a table to store the solutions of solved subproblems. If you face a subproblem again, you just need to take the solution in the table without having to solve it again. Therefore, the algorithms designed by dynamic programming are very effective.

2. Define OBST.

An Optimal Binary Search Tree (OBST), also known as a Weighted Binary Search Tree, is a binary search tree that minimizes the expected search cost. In a binary search tree, the search cost is the number of comparisons required to search for a given key.

3. What are the applications of transitive closure of a digraph?

- interested in a matrix containing the information about the existence of directed paths of arbitrary lengths between vertices of a given graph.
- Such a matrix, called the transitive closure of the digraph, would allow us to determine in constant time whether the j th vertex is reachable from the ith vertex.

4. What does Floyd's algorithm?

Floyd's algorithm is an application, which is used to find all pairs shortest paths problem. Floyd's algorithm is applicable to both directed and undirected weighted graph, but they do not contain a cycle of a negative length

5. Define the principle of optimality.

It states that an optimal sequence of decisions has the property that whenever the initial stage or decisions must constitute an optimal sequence with regard to stage resulting from the first decision.

6. What do you mean by dynamic programming?

Dynamic programming is a technique for solving problems with overlapping subproblems. Typically, these subproblems arise from a recurrence relating a given problem's solution to solutions of its smaller subproblems. Rather than solving overlapping subproblems again and again, dynamic programming suggests solving each of the smaller subproblems only once and recording the results in a table from which a solution to the original problem can then be obtained.

7. Compare Divide and Conquer and Dynamic Programming.

Divide and Conquer	**Dynamic Programming**
Subproblems are solved independently, and finally all solutions are collected to arrive at the final answers.	Dynamic programming considers a large number of decision sequences and all the overlapping substances.
The divide and conquer strategy is slower than the dynamic programming approach.	The dynamic programming strategy is slower than the divide and conquer approach.
Maximize time for execution.	Reduce the amount of time spent on execution by consuming less time.
Recursive techniques are used in Divide and Conquer.	Non-Recursive techniques are used in Dynamic programming.
A top-down approach is used in Divide and Conquer.	In a dynamic programming solution, the bottom-up approach is used.
The problems that are part of a Divide and Conquer strategy are independent of each other.	A dynamic programming subproblem is dependent upon other sub-problems.

One of the best examples of this strategy is a binary search.	One of the best examples of this strategy is the longest common subsequence.
No results are stored when completing sub-problems.	The solutions to sub-problems are saved in the table.
Repeating tasks.	There is no repeating task.
At a specified point, the split input splits big problems into smaller ones.	Every point in the split input is processed.
The divide and conquer strategy is simple to solve.	A dynamic programming solution can sometimes be complicated and challenging to solve.
Not more than one decision sequence is generated.	More than one decision sequence is generated.

8. Compute the order of growth of the following recurrence: T(n) = 4T(n/2) + n, T(1) = 1.

The recurrence can be written in the form T(n) = aT(n/b) + f(n), where a = 4, b = 2, and f(n) = n.

Using the master theorem, we need to compare f(n) = n to n^logb(a) = n^2. Since f(n) = n is polynomially smaller than n^2, we are in case 1 of the master theorem. Therefore, the solution to the recurrence is T(n) = Θ(n^2).

Thus, the order of growth of the recurrence T(n) = 4T(n/2) + n is Θ(n^2).

9. Compute the maximal length of a codeword possible in a Huffman encoding of an alphabet of n characters?

The longest codeword can be of length n − 1. An encoding of n symbols with n − 2 of them having probabilities 1/2,1/4,...,1/2n-2 and two of them having probability 1/2n-1 achieves this value. No codeword can ever by longer than length n − 1.

10. State coin row problem.

 There is a row of n coins whose values are some positive integers C1, C2…..,Cn, not necessarily distinct.

 The goal is to pick up the maximum amount of money subject to the constraint that no two coins adjacent in the initial row can be picked up.

11. What is all pairs shortest path algorithm?

Given a weighted connected graph (undirected or directed), the ***all-pairs shortest paths problem*** asks to find the distances—i.e., the lengths of the shortest paths— from each vertex to all other vertices.

12. Define Transitive Closure of digraph.

The transitive closure of a directed graph with n vertices can be defined as the $n \times n$ boolean matrix $T = \{tij\}$, in which the element in the ith row and the jth column is 1 if there exists a nontrivial path (i.e., directed path of a positive length) from the ith vertex to the jth vertex; otherwise, tij is 0.

13. What is the formula for Binomial coefficient? And Explain how Binomial Coefficient is computed?

The binomial co-efficient C(n,k) is the number of ways of choosing a subset of k elements from a set of n elements.

$$C(n,k) = \begin{cases} 1 & \text{if } k=0 \\ 1 & \text{if } n = k \\ C(n-1,k-1) + C(n-1, k) & \text{if } n>k>0 \end{cases}$$

14. Compare Dynamic Programming and Greedy Technique.

Greedy method	Dynamic programming
1. Only one sequence of decision is generated.	1. Many number of decisions are generated.
2.It does not guarantee to give an optimal solution always	2.It definitely gives an optimal solution always.

15. Outline the general procedure of dynamic programming.

The development of dynamic programming algorithm can be broken into a sequence of 4 steps.
1. Characterize the structure of an optimal solution.
2. Recursively defines the value of the optimal solution.
3. Compute the value of an optimal solution in the bottom-up fashion.
4. Construct an optimal solution from the computed information.

16. What are the different types of coding schemes available?

BCD Code. BCD stands for binary coded decimal. It is a 4-bit code. ...

EBCDIC Code. EBCDIC stands for extended binary coded decimal interchange code. It is an 8-bit code. ...

ASCII. ASCII stands for American standard code for information interchange. ...

Unicode. Unicode is a 16-bit code.

17. Define feasible and optimal solution.

Feasible : It has to satisfy the problem‘s constraints

 locally optimal: It has to be the best local choice among all feasible choices available on that step

18. What are the applications of Huffman Codes?

Huffman coding is used in conventional compression formats like GZIP, BZIP2, PKZIP, etc. For text and fax transmissions.

19. Prove that any comparison sort algorithm requires Ω (n log n) comparisons in the worst case.

***theriyalaye**

20. Compare divide and conquer and dynamic programming approach.

Typically, divide-and-conquer divides an instance into smaller instances with no intersection whereas dynamic programming deals with problems in which smaller instances overlap. Consequently, divide-and-conquer algorithms do not explicitly store

solutions to smaller instances and dynamic programming algorithms do.

21. Show - Is Merge Sort and Quick Sort a stable sorting algorithm?

Merge sort is a stable sorting algorithm, i.e., it maintains the relative order of two equal elements. Quicksort is an unstable sorting algorithm, i.e., it might change the relative order of two equal elements.

22. What is the worst case complexity of binary search?

O(log n)

23. Define extreme points.

Any linear programming problem with a nonempty bounded feasible region has an optimal solution; moreover, an optimal solution can always be found at an extreme point of the problem's feasible region

24. Write the applications of closest pair problem.

Applications:

- Points in question can represent such physical objects as airplanes or post offices as well as database records, statistical samples, DNA sequences, and so on.
- An air-traffic controller might be interested in two closest planes as the most probable collision candidates.
- A regional postal service manager might need a solution to the closest pair problem to find candidate post-office locations to be closed.

Part-B

1.Solve the instance of the problem to find OBST.

key	A	B	C	D
probability	0.1	0.2	0.4	0.3

Obtain a optimal binary search tree for the following nodes do, if, int, while with the following probabilities pair

0.1, 0.2, 0.4, 0.3

C(1,2)
C(2,3)
C(3,4)
C(1,3)
C(2,4)
C(1,4)

Sol:

Cost table:

$j \longrightarrow$, $i \downarrow$

	0	1	2	3	4
1	0	0.1	0.4	1.1	1.7
2			0.2	0.8	1.4
3				0.4	1
4					0.3
5					0

Root table:

	1	2	3	4
1	1	2	3	3
2		2	3	3
3			3	3
4				4

$$c[i,j] = \min_{i \le k \le j} \{ c[i, k-1] + c[k+1, j] \} + \sum_{s=i}^{j} p_s$$

$c[1,2]$ min

$\xrightarrow{k=1}$ $c[1,0] + c[2,2] + p_1 + p_2$

$= 0 + 0.2 + 0.1 + 0.2 = 0.5$

$\xrightarrow{k=2}$ $c[1,1] + c[3,2] + p_1 + p_2$

$= 0.1 + 0 + 0.1 + 0.2 = 0.4$ (min)

$c[2,3]$

$\xrightarrow{k=2}$ $c[2,1] + c[3,3] + p_2 + p_3$

$= 0 + 0.4 + 0.2 + 0.4 = 1$

$\xrightarrow{k=3}$ $c[2,2] + c[4,3] + p_2 + p_3$

$= 0.2 + 0 + 0.6 = 0.8$ (min)

$c[3,4]$

$\xrightarrow{k=3}$ $c[3,2] + c[4,4] + p_2 + p_4$

$= 0 + 0.3 + 0.7 = \cancel{0.9}\ 1.0$ (min)

$\xrightarrow{k=4}$ $c[3,3] + c[5,4] + p_3 + p_4$

$0.4 + 0 + 0.7 = 1.1$

$c[1,3]$ min

$\xrightarrow{k=1}$ $c[1,0] + c[2,3] + p_1 + p_2 + p_3$

$= 0 + 0.8 + 0.1 + 0.2 + 0.4$

$= 1.5$

$\xrightarrow{k=2}$ $c[1,1] + c[3,3] + 0.7$

$= 0.1 + 0.4 + 0.7 = 1.2$

$\xrightarrow{k=3}$ $c[1,2] + c[4,3] + 0.7$

$0.4 + 0 + 0.7 = 1.1$ (min).

$c[2,4]$ min

$\xrightarrow{k=2}$ $c[2,1] + c[3,4] + p_2 + p_3 + p_4$

$= 0 + 1 + 0.9 = 1.9$

$\xrightarrow{k=3}$ $c[2,2] + c[4,4] + 0.9$

$= 0.2 + 0.3 + 0.9 = 1.4$ (min)

$\xrightarrow{k=4}$ $c[2,3] + c[5,4] + 0.9$

$= 0.8 + 0 + 0.9 = 1.7$

$c[1,4] \xrightarrow{\min}$

$k=1 \rightarrow c[1,0] + c[2,4] + P_1 + P_2 + P_3 + P_4$

$= 0 + 1.4 + 1 = 2.4.$

$k=2 \rightarrow c[1,1] + c[3,4] + 1 = 0.1 + 1 + 1$

$= 2.1$

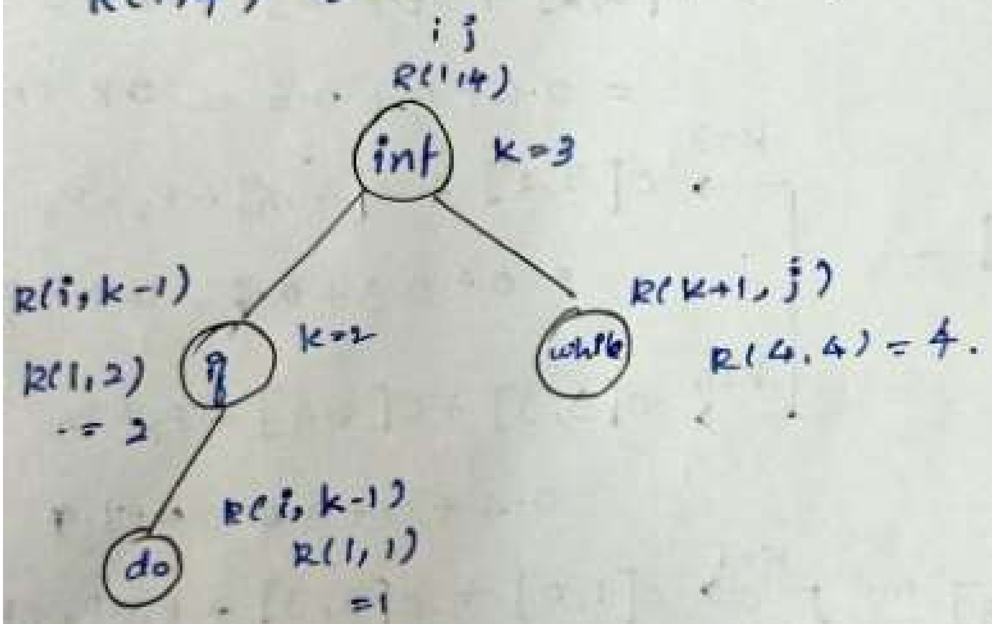
$k=3 \rightarrow c[1,2] + c[4,4] + 1 = 0.4 + 0.3 + 1$

$= 1.7$ (min)

$k=4 \rightarrow c[1,3] + c[5,4] + 1 = 1.1 + 0 + 1$

$= 2.1$

$R(1,4) = 3$



2.Solve the all-pairs shortest-path problem for the digraph with the following weight matrix:

$$\begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix}$$

Given:

Solve the following matrix using floyd's algorithm:

$$D^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & \infty \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & \infty & \infty & \infty & 0 \end{bmatrix} \end{array}$$

$$D^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 2 & \infty & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & \infty & 4 & 0 \end{bmatrix} \end{array}$$

$$D^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{bmatrix} 0 & 2 & 5 & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \infty & \infty & 0 & 4 & \infty \\ \infty & \infty & 2 & 0 & 3 \\ 3 & 5 & 8 & 4 & 0 \end{bmatrix} \end{array}$$

$$D^{(3)} = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 2 & 5 & 1 & 8 \\ b & 6 & 0 & 3 & 2 & 14 \\ c & \alpha & \alpha & 0 & 4 & \alpha \\ d & \alpha & \alpha & 2 & 0 & 3 \\ e & 3 & 5 & 8 & 4 & 0 \end{array}$$

$$\begin{array}{ccccc} 0 & 2 & 5 & 1 & 8 \\ 6 & 0 & 3 & 2 & 14 \\ \alpha & \alpha & 0 & 4 & \alpha \\ \alpha & \alpha & 2 & 0 & 3 \\ 3 & 5 & 8 & 4 & 0 \end{array}$$

$$D^{(4)} = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 2 & 3 & 1 & 4 \\ b & 6 & 0 & 3 & 2 & \cancel{4}\,5 \\ c & \alpha & \alpha & 0 & 4 & 7 \\ d & \alpha & \alpha & 2 & 0 & 3 \\ e & 3 & 5 & 6 & 4 & 0 \end{array}$$

$$D^{(5)} = \begin{array}{c|ccccc} & a & b & c & d & e \\ \hline a & 0 & 2 & 3 & 1 & 4 \\ b & 6 & 0 & 3 & 2 & 5 \\ c & 10 & 12 & 0 & 4 & 7 \\ d & 6 & 8 & 2 & 0 & 3 \\ e & 3 & 5 & 6 & 4 & 0 \end{array}$$

3. Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

item	weight	value
1	3	$25
2	2	$20
3	1	$15
4	4	$40
5	5	$50

capacity $W = 6$.

Sol: W=6 Items=5

Item \ capacity	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	-1	-1	-1	-1	-1	-1
2	0	-1	-1	-1	-1	-1	-1
3	0	-1	-1	-1	-1	-1	-1
4	0	-1	-1	-1	-1	-1	-1
5	0	-1	-1	-1	-1	-1	-1

i=5 j=6
$w_5 = 5$
$j-w_5 = 6-5$

$V[5,6] = \max\{V[4,6], 50 + V[4,1]\}$

$V[4,6]$: $i=4$, $j=6$ $\quad w_4 = 4 \quad v_4 = 40$.

$j-w_i = 6-4 = 2 \geq 0$.

$= \max\{V[3,6], 40 + V[3,2]\}$

$V[4,1]$: $i=4$ $j=1$ $\quad w_4 = 4 \quad v_4 = 40$.

$j - w_i = 1-4 = -3 < 0$

$V[4,1] = V[3,1]$:

$v[3,6]$: $i=3$ $j=6$ $W_3=1$ $V_3=15$

$j-w_i = 6-1 = 5 > 0.$

$v[3,6] = \max\{v[2,6],\ 15+v[2,5]\}$

$v[3,2]$: $i=3$ $j=2$ $W_3=1$ $V_3=15$

$j-w_i = 2-1 = 1 > 0.$

$v[3,2] = \max\{v[2,2],\ 15+v[2,1]\}$

$v[3,1]$ = $i=3$ $j=1$ $W_3=1$ $V_3=15$

$j-w_i = 1-1 = 0 \le 0.$

$v[3,1] = \max\{ v[2,1], 15 + v[2,0] \}$

$v[2,1]$: $i = 2$ $j = 1$ $w_2 = 2$ $v_2 = 20$.

$j - w_i = 1 - 2 = -1 < 0$.

$v[i-1, j] = v[1,1]$

$v[2,6] = \max \{ v[1,6], 20 + v[1,4] \}$ (25 , 20+25)

$i = 2$ $j = 6$ $w_2 = 2$ $v_2 = 20$ $j - w_i = 6 - 2 = 4$

$v[2,5]$: $i = 2$ $j = 5$. $w_2 = 2$ $v_2 = 20$.

$j - w_i = 5 - 2 = 3 > 0$.

$= \max\{ v[1,5], 20 + v[1,3] \}$

$v[2,2]$: $i = 2$ $j = 2$ $w_2 = 2$ $v_2 = 20$

$j - w_i = 2 - 2 = 0$

$= \max\{ v[1,2], 20 + v[1,0] \}$

$v[2,1]$: $i = 2$ $j = 1$ $w_2 = 2$ $v_2 = 20$

$j - w_i = 1 - 2 = -1 < 0$

$v[i-1, j] = v[1,1]$

$v[1,6]$: $i = 1$ $j = 6$ $w_1 = 3$ $v_1 = 25$

$j - w_i = 6 - 3 = 3 > 0$

$v[1,6] = \max \{ v[0,6], 25 + v[0,3] \}$

$= 25$

$v[1,4]$ = $i = 1$ $j = 4$ $w_1 = 3$ $v_1 = 25$

$j - w_i = 4 - 3 = 1 > 0$.

$v[1,4] = \max\{ v[0,4], 25 + v[0,1] \}$

$= 25$

$v[1,5]$ = $i = 1$ $j = 5$ $w_1 = 3$ $v_1 = 25$

$j - w_i = 5 - 3 = 2 > 0$

$v[1,5] = \max\{ v[0,5], 25 + v[0,2] \} = 25$

$V[1,1]$: $i=1$ $j=1$ $w_1 = 3$ $V_1 = 25$

$j - w_i = 1 - 3 = -2 < 0$

$V[i-1, j] = V[0,1] = 0$.

$V[1,3] = 25$

$V[1,2]$: $i=1$ $j=2$ $w_1 = 3$ $V_1 = 25$

$j - w_i = 2 - 3 = -1 < 0$.

$V[i-1, j] = V[0,2] = 0$.

$V[2,2] = \max\{0, 20\} = 20$

$V[2,5] = \max\{25, 20+25\} = 45$

$V[3,2] = \max\{20, 15\} = 20$.

$V[3,6] = \max\{45, 60\} = 60$

$V[4,6] = \max\{60, 40+20\} = 60$.

$V[5,6] = \max\{60, 50+15\} = \max\{60, 65\} = 65$

Item \ Cap	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0.	0.	25	25	25	25
2	0	0	20.	-1	-1	45.	45
3	0	15	20	-1	-1	-1	60
4	0	15	-1	-1	-1	-1	60
5	0	-1	-1	-1	-1	-1	65

4.

a. Construct a Huffman code for the following data: character

character	A	B	C	D	_
probability	0.4	0.1	0.2	0.15	0.15

b. Encode the text ABACABAD using the code of question a.

Decode the text whose encoding is 100010111001010 in the code of question a.

Construct ~~an~~ huffman code for the following data.

character	A	B	C	D	–
probability	0.4	0.1	0.2	0.15	0.15

(a) Encode the text ABACABAD

(b) Decode the text 100010111001010

Sol:

n = 5 {A, B, C, D, –}.

step 1:

A	B	C	D	–
0.4	0.1	0.2	0.15	0.15

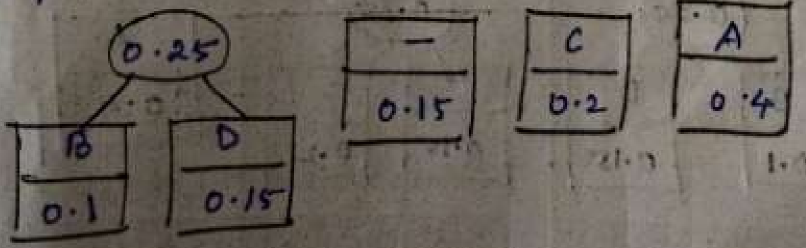
step 2: Arrange in Ascending order

B	D	–	C	A
0.1	0.15	0.15	0.2	0.4

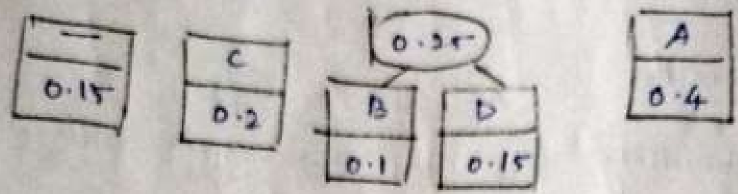
step 3: combine 2 elements

B	D	–	C	A
0.1	0.15	0.15	0.2	0.4

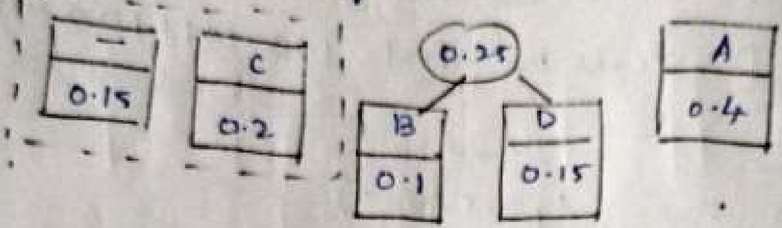
step 4:



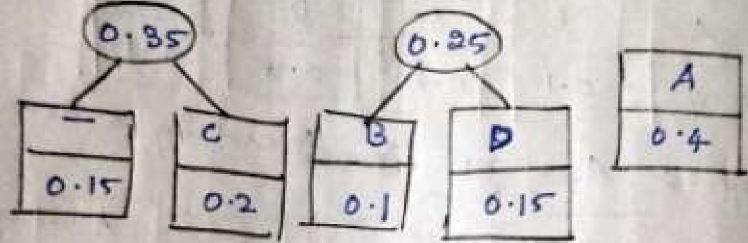
step 5: Arrange in Ascending order.



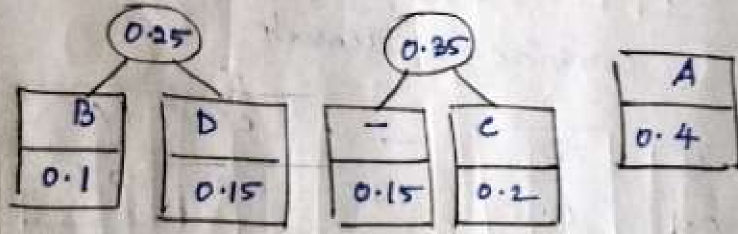
step 6: Combine first 2 elements.



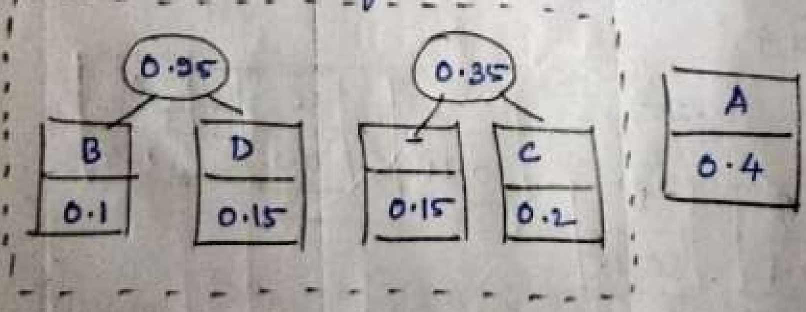
step 7:



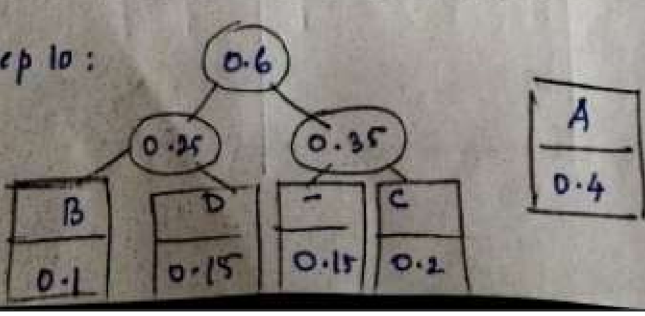
step 8: Arrange in Ascending order.



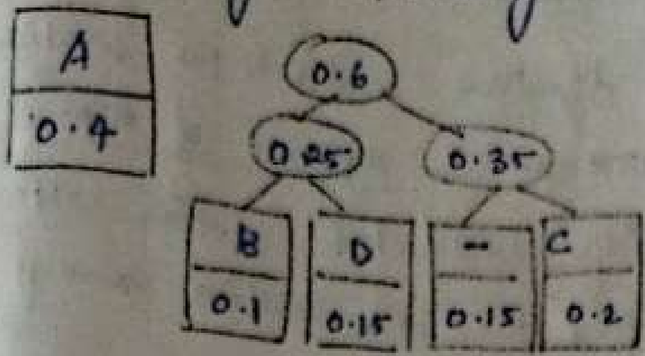
step 9: Combine first 2 elements.



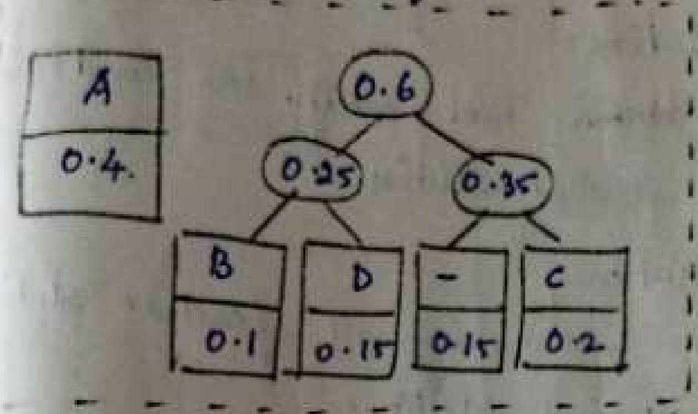
step 10:



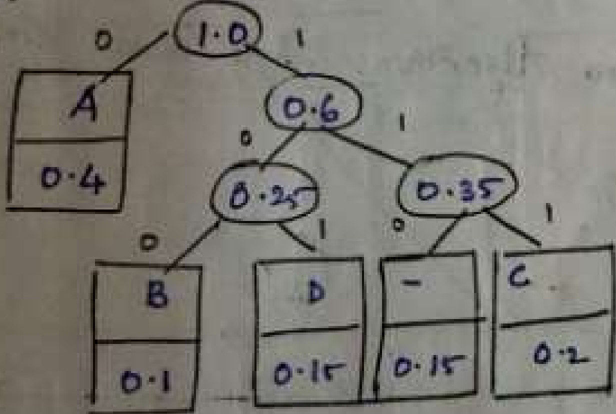
Step 11: Arrange in Ascending order



Step 12: Combine first 2 elements



Step 13:



character	A	B	C	D	-
code Word	0	100	111	101	110

(a) ABACABAD - 0100011101000101

(b) 1 0 0|0|1 0 1|1 1 0|0|1 0 1|0|
B A D - A D A

5.Outline the algorithm for quick sort. Provide a complete analysis of quick sort for the given set of numbers 12, 33, 23, 43, 44, 55, 64, 77 and 76.

*therincha Send Please

6. Compute the multiplication of any two matrices of size 2 × 2 using Strassen's matrix multiplication method and analyze its time complexity.

use Strassen's matrix multiplication to multiply 2 matrices $\begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix} \begin{bmatrix} 2 & 7 \\ 8 & 3 \end{bmatrix}$

Sol:

$$\begin{bmatrix} \underset{a_{00}}{3} & \underset{a_{01}}{5} \\ \underset{a_{10}}{4} & \underset{a_{11}}{6} \end{bmatrix} \begin{bmatrix} \underset{b_{00}}{2} & \underset{b_{01}}{7} \\ \underset{b_{10}}{8} & \underset{b_{11}}{3} \end{bmatrix}$$

$m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$

$= (3+6) * (2+3) = 9 * 5 = 45$

$m_2 = (a_{10} + a_{11}) * b_{00}$

$= (4+6) * 2 = 10 * 2 = 20$

$m_3 = a_{00} * (b_{01} - b_{11})$

$= 3 * (7-3) = 3 * 4 = 12$

$m_4 = a_{11} * (b_{10} - b_{00})$

$= 6 * (8 - 2) = 6 * 6 = 36.$

$m_5 = (a_{00} + a_{01}) * b_{11}$

$= (3 + 5) * 3 = 8 * 3 = 24.$

$m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$

$= (4 - 3) * (2 + 7) = 1 * 9 = 9$

$m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$

$= (5 - 6) * (8 + 3) = -1 * 11 = -11.$

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m \end{bmatrix}$$

$$= \begin{bmatrix} 46 & 36 \\ 56 & 46 \end{bmatrix}$$

Analysis.

$$M(n) = 7M(n/2)$$

$n = 2^k$

For $n > 1$

$M(1) = 1$

$$M(2^k) = 7M(2^k/2)$$

$$= 7M(2^{k-1})$$

$$= 7(7M(2^{k-2}))$$

$$= 7(7(7(M\,2^{k-3})))$$

$$= 7^{k-1}M(2^{k-k})$$

$$= 7^k M(2^0)$$

$$= 7^k.$$

$$= k = \log 2^n$$

$$M(n) = 7^{\log 2^n}$$

$$= n^{\log_2 7}$$

$$\boxed{M(n) = n^{2.807}}$$

7. Compute 2101 * 1130 by applying the divide-and-conquer algorithm

1. Compute 2101 * 1130

Solution

$a = a_1 a_0$ implies that $a = a_1 10^{n/2} + a_0$

$a_1 = 21$, $a_0 = 01$

$b = b_1 b_0$ implies that $b = b_1 10^{n/2} + b_0$

$b_1 = 11$, $b_0 = 30$

$c = a * b = (a_1\ 10^{n/2} + a_0) * (b_1\ 10^{n/2} + b_0)$

$= (a_1 * b_1)\ 10^n + (a_1 * b_0 + a_0 * b_1)\ 10^{n/2} + (a_0 * b_0)$

$= c_2 10^n + c_1 10^{n/2} + c_0$

$c_2 = a_1 * b_1$ is the product of their first halves,

$c_0 = a_0 * b_0$ is the product of their second halves,

$c_1 = (a_1 + a_0) * (b_1 + b_0) - (c_2 + c_0)$ is the product of the sum of the *a*'s halves and the sum of the *b*'s halves minus the sum of c_2 and c_0.

For 2101* 1130:

$c_2 = 21 * 11$

$c_0 = 01 * 30$

$c_1 = (21 + 01) * (11 + 30) - (c_2 + c_0) = 22 * 41 - 21 * 11 - 01 * 30$

For 21 * 11:

$c_2 = 2*1 = 2$

$c_0 = 1*1 = 1$

$c_1 = (2 + 1) *(1 + 1) - (2 + 1) = 3*2-3 = 3.$

So, $21*11 = 2.10^2 + 3.10^1 + 1 = 231.$

For 01 * 30:

$c_2 = 0*3 = 0$

$c_0 = 1*0 = 0$

$c_1 = (0+1)*(3 + 0)-(0 + 0) = 1*3-0 = 3$

So, $01*30 = 0.10^2 + 3.10^1 + 0 = 30.$

For 22 * 41:

$c_2 = 2*4 = 8$

$c_0 = 2*1 = 2$

$c_1 = (2 + 2)* (4 + 1)-(8 + 2) = 4*5-10 = 10.$

So, $22 * 41 = 8.10^2 + 10.10^1 + 2 = 902.$

Hence

$2101 * 1130 = 231.10^4 + (902-231-30).10^2 + 30 = 2,374,130$

8. Apply quick sort to sort the list E, X, A, M, P, L, E in alphabetical order. Draw the tree of the recursive calls made.

5. Apply quicksort to sort the list E, X, A, M, P, L, E

Solution

Given : E X A M P L E

Pivot = E

(i) Search from the backward to find < pivot, and create a vacant position in the index.

A X – M P L E

(ii) Search from the forward to find ≥ pivot and create a vacant position in the index.

A – X M P L E

(iii) Process continues to find the position for pivot, which divides the array into sub ranges.

A E X M P L E

(iv) Partition of second section (after pivot ≥ E)

X M P L E
pivot

– M P L E

E M P L –

E M P L X

(v) Partition of left subsection:

E M P L
pivot

(vi) Partition of right subsection

M P L
pivot

– P L

L P –

L – P

L M P

(vii) Combining the resultant of each section.

A E E L M P X

Part-C

1. Apply Warshall's algorithm to find the transitive closure of the digraph defined by the following

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$R^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R^{(3)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

$$R^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

2. Solve C(3,2) using Dynamic Programming.

$c(3,2)$

$$c(n,n) = 1$$

$$c(n,0) = 1$$

$$c(n,k) = c(n-1, k-1) + c(n-1, k)$$

n \ k	0	1	2	3
0	1	—	—	—
1	1	1	—	—
2	1	2	1	
3	1	3	$\boxed{3}$	1
4	1	4	6	4

$$c(\overset{3}{n}, \overset{2}{k}) = c(n-1, k-1) + c(n-1, k)$$

$$= c(2,1) + c(2,2)$$

$$= c(2,1) + 1$$

$$c(\overset{n}{2}, \overset{k}{1}) = c(1,0) + c(1,1)$$

$$= 1 + 1 = 2$$

$$c(3,2) = c(2,1) + 1$$

$$= 2 + 1 = 3$$

$$c(3,2) = 3 \ //$$

3. Solve the instance 5, 1, 2, 10, 6 of the coin-row problem

ALGORITHM CoinRow(C[1..n])

//Input: Array C[1..n] of positive integers indicating the coin values
//Output: The maximum amount of money that can be picked up

F [0] ← 0; F [1] ← C[1]
for i ← 2 to n do
F [i] ← max(C[i] + F [i − 2], F [i − 1])
return F [n]

The application of the algorithm to the coin row of denominations 5, 1, 2, 10, 6, 2
It yields the maximum amount of 17

$F[0] = 0$, $F[1] = c_1 = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

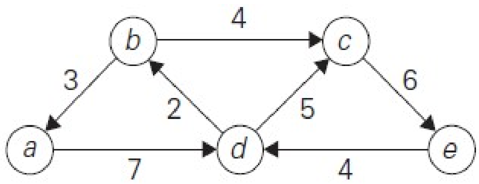
index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

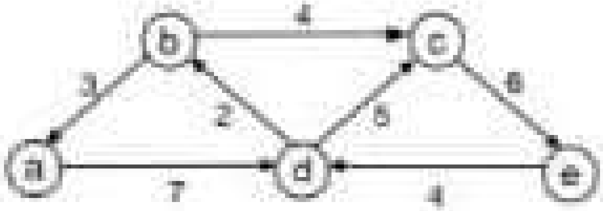
$F[6] = \max\{2 + 15, 15\} = 17$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	**17**

.1 Solving the coin-row problem by dynamic programming for the coin row 5, 1, 2, 10, 6, 2.

4. Solve the following instances of the single-source shortest-paths problem with vertex *a* as the





Tree vertices	Remaining vertices
a(-,0)	b(-,∞) c(-,∞) **d(a,7)** e(-,∞)
d(a,7)	**b(d,7+2)** c(d,7+5) e(-,∞)
b(d,9)	**c(d,12)** e(-,∞)
c(d,12)	**e(c,12+6)**
e(c,18)	

The shortest paths (identified by following nonnumeric labels backwards from a destination vertex to the source) and their lengths are:

from a to d:	$a - d$	of length 7
from a to b:	$a - d - b$	of length 9
from a to c:	$a - d - c$	of length 12
from a to e:	$a - d - c - e$	of length 18