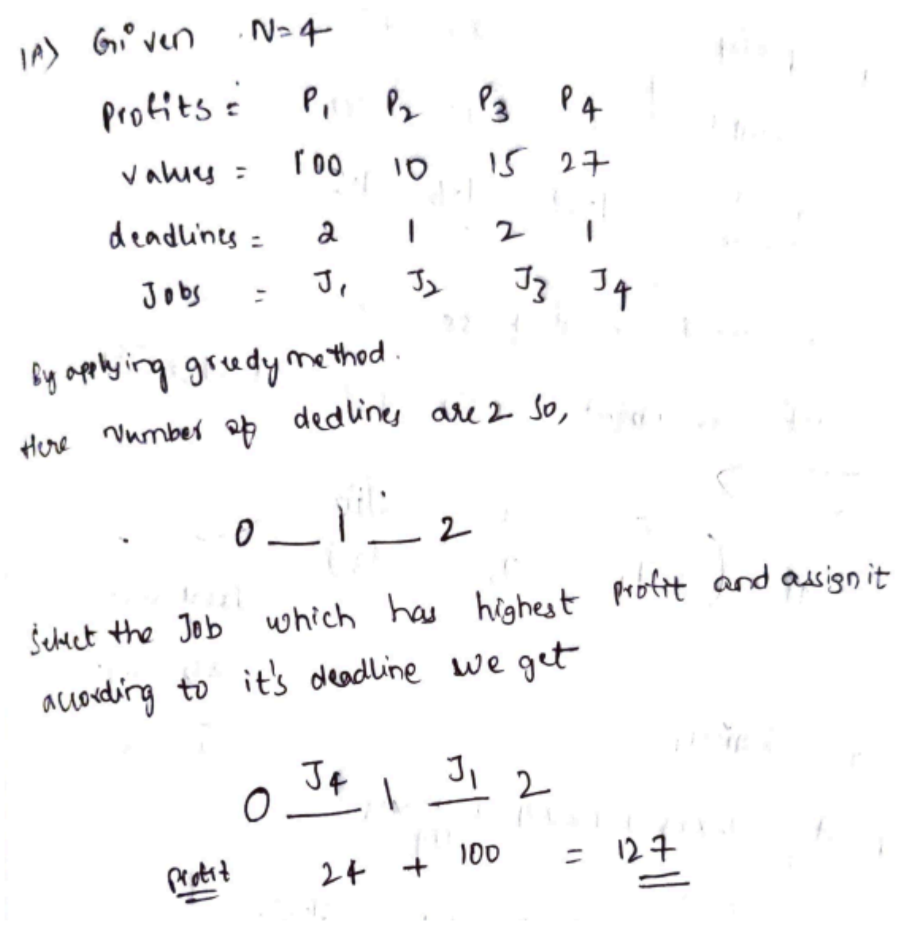
Many Pending

# DAA MODULE 3

## PART A (CIE 2)

**1. Identify the optimal solution for job sequencing with deadlines using greedy methods. N=4, profits (p1,p2 ,p3,p4) = (100,10, 15,27), Deadlines (d1,d2, d3,d4) = (2,1,2,1)**



1A) Given N=4

| | | | | |
|---|---|---|---|---|
| profits = | $P_1$ | $P_2$ | $P_3$ | $P_4$ |
| values = | 100 | 10 | 15 | 27 |
| deadlines = | 2 | 1 | 2 | 1 |
| Jobs = | $J_1$ | $J_2$ | $J_3$ | $J_4$ |

By applying greedy method.

Here number of deadlines are 2 so,

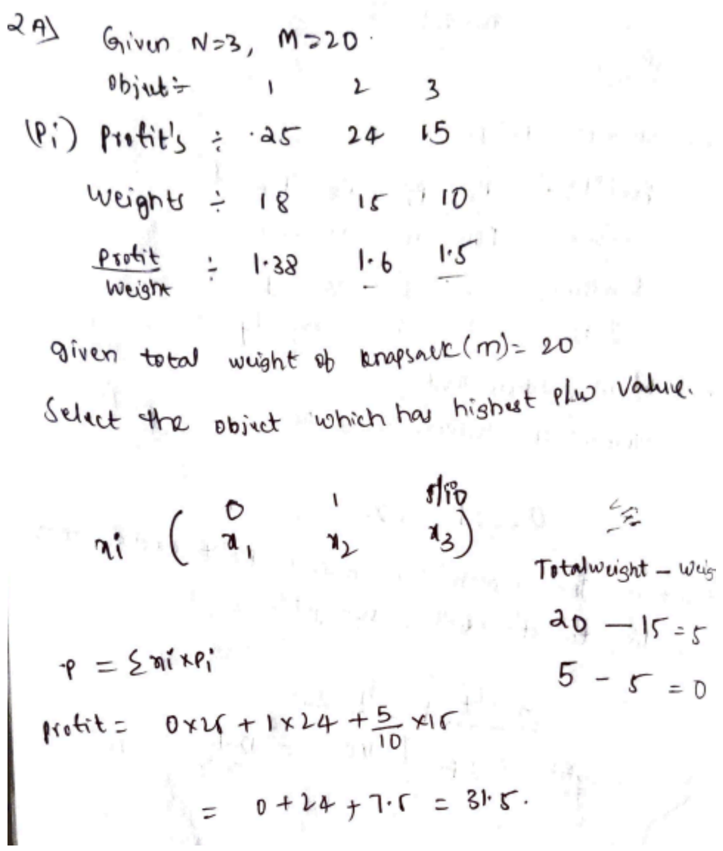
$0 \_\_ 1 \_\_ 2$

Select the Job which has highest profit and assign it according to it's deadline we get

$0 \underline{\ J_4\ } 1 \underline{\ J_1\ } 2$

Profit $27 + 100 = \underline{\underline{127}}$

**2. Identify the optimal solution for knapsack problem using greedy method N=3, M= 20, (p1,p2,p3)= (25,24,15), (w1,w2,w3) =(18,15,10)**



2A) Given $N=3$, $M=20$.

| | | | | |
|---|---|---|---|---|
| | Object :- | 1 | 2 | 3 |
| $(P_i)$ | Profit's :- | 25 | 24 | 15 |
| | Weights :- | 18 | 15 | 10 |
| | $\frac{\text{Profit}}{\text{Weight}}$ :- | 1.38 | 1.6 | 1.5 |

given total weight of knapsack(m) = 20

Select the object which has highest p/w value.

$$x_i \ (\overset{0}{x_1}, \ \overset{1}{x_2}, \ \overset{5/10}{x_3})$$

Totalweight − Weig

$20 - 15 = 5$

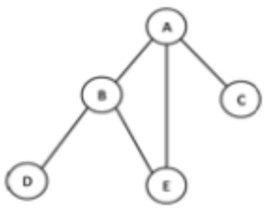
$5 - 5 = 0$

$P = \Sigma x_i \times p_i$

profit $= 0 \times 25 + 1 \times 24 + \frac{5}{10} \times 15$

$= 0 + 24 + 7.5 = 31.5.$

**6. Identify whether a given graph is connected or not using the DFS method.**



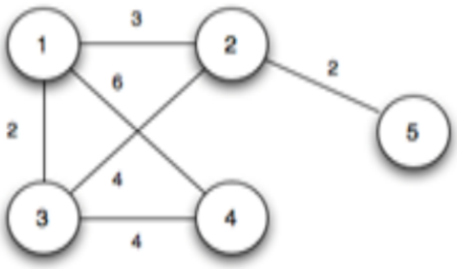
Depth-first search starts visiting vertices of a graph at an arbitrary vertex by marking it as having been visited. On each iteration, the algorithm proceeds to an unvisited vertex that is adjacent to the one it is currently in. This process continues until a vertex with no adjacent unvisited vertices is encountered.

Thus, if all the vertices of the above graph are connected, then, DFS must cover all the vertices of the graph.

The DFS of the above tree is: A → B → D → E → C.

Thus, all the vertices are covered and so, the graph is connected.

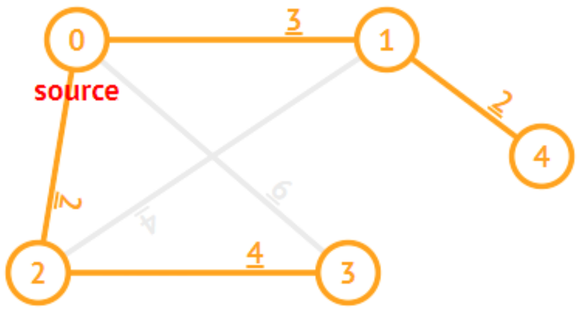
**7. Construct Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.**



Prim's algorithm is a greedy algorithm that starts from one vertex and continues to add the edges with the smallest weight until the goal is reached.

The steps to implement the prim's algorithm are given as follows -

- First, we have to initialise an MST with the randomly chosen vertex.
- Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
- Repeat step 2 until the minimum spanning tree is formed.



**8. Apply Optimal binary search tree algorithm and compute wij, cij, rij, 0¡=i¡=j¡=4,p1=1/10, p2=1/5, p3=1/10, p4=1/120, q0=1/5, q1=1/10, q2=1/5, q3=1/20,q4=1/20.**

**9. Construct optimal binary search tree for (a1, a2, a3, a4) = (do, if,int, while), p(1 : 4) = (3,3,1,1) q(0 : 4)= (2,3,1,1,1)**

**10.Solve the solution for 0/1 knapsack problem using dynamic programming (p1,p2,p3, p4) = (11, 21, 31, 33), (w1, w2, w3, w4) = (2, 11, 22, 15), M=40, n=4**

## PART B (CIE 2)

**8. Demonstrate Bellman Ford algorithm to compute shortest path.**

The problem of Dijkstra's algorithm is that it cannot deal with negative weights, Bellman-Ford works for such graphs. Bellman-Ford is also simpler than Dijkstra and suits well for distributed systems. But the time complexity of Bellman-Ford is O(VE), which is more than Dijkstra's greedy algorithm with a time complexity of O((V + E)LogV) with the use of Fibonacci Heap.

<PENDING>

**9. Explain the greedy method for generating shortest paths.**

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph. The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

**Take a simple example and find shortest path using Dijkstra's Algorithm**

**10. Explain the time complexities of Prim's and Kruskal's algorithms**

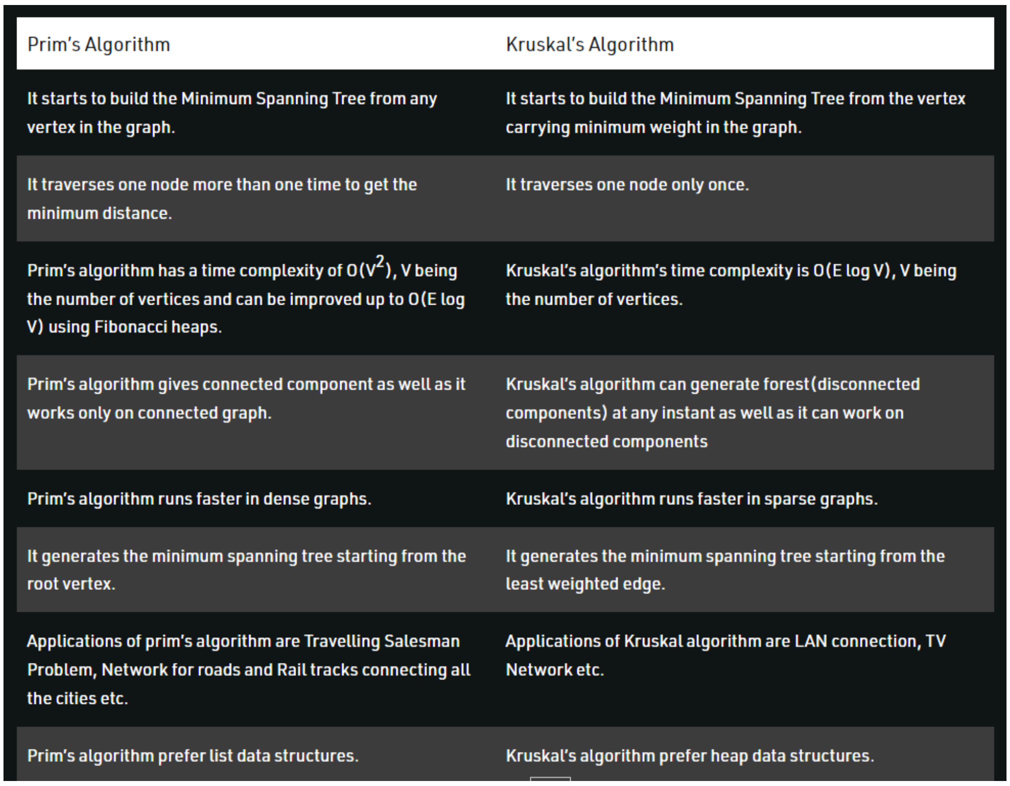
Prim's algorithm is popular as a greedy algorithm that helps in discovering the smallest spanning tree for a weighted undirected graph. That means, this algorithm tends to search the subgroup of the edges that can construct a tree, and the complete poundage of all the edges in the tree should be minimal.

The time complexity of Prim's algorithm is $O(V^2)$.

Kruskal's Algorithm is used to discover the smallest spanning tree of a connected graph and the spanning forest of an undirected edge-weighted graph. Basically, it accepts a graph as input and discovers the subgroup of the edges of that graph.

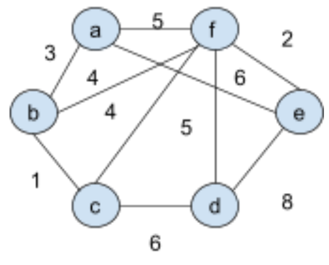
The time complexity of Kruskal's algorithm is O(E log V).

**11. Compare Prim's and Kruskal's Algorithms.**

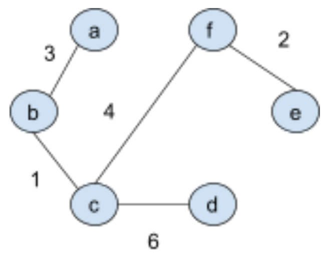


| Prim's Algorithm | Kruskal's Algorithm |
|---|---|
| It starts to build the Minimum Spanning Tree from any vertex in the graph. | It starts to build the Minimum Spanning Tree from the vertex carrying minimum weight in the graph. |
| It traverses one node more than one time to get the minimum distance. | It traverses one node only once. |
| Prim's algorithm has a time complexity of $O(V^2)$, V being the number of vertices and can be improved up to O(E log V) using Fibonacci heaps. | Kruskal's algorithm's time complexity is O(E log V), V being the number of vertices. |
| Prim's algorithm gives connected component as well as it works only on connected graph. | Kruskal's algorithm can generate forest(disconnected components) at any instant as well as it can work on disconnected components |
| Prim's algorithm runs faster in dense graphs. | Kruskal's algorithm runs faster in sparse graphs. |
| It generates the minimum spanning tree starting from the root vertex. | It generates the minimum spanning tree starting from the least weighted edge. |
| Applications of prim's algorithm are Travelling Salesman Problem, Network for roads and Rail tracks connecting all the cities etc. | Applications of Kruskal algorithm are LAN connection, TV Network etc. |
| Prim's algorithm prefer list data structures. | Kruskal's algorithm prefer heap data structures. |

**12. Make use of Prim's algorithm to find minimum cost spanning tree for a graph G(6,10) with vertices named as a,b,c,d,e,f and edges ab=3,bc=1, af=5,ae=6,ed=8, fe=2,fd=5, cd=6,cf=4 and bf=4 by showing results in each stages.**

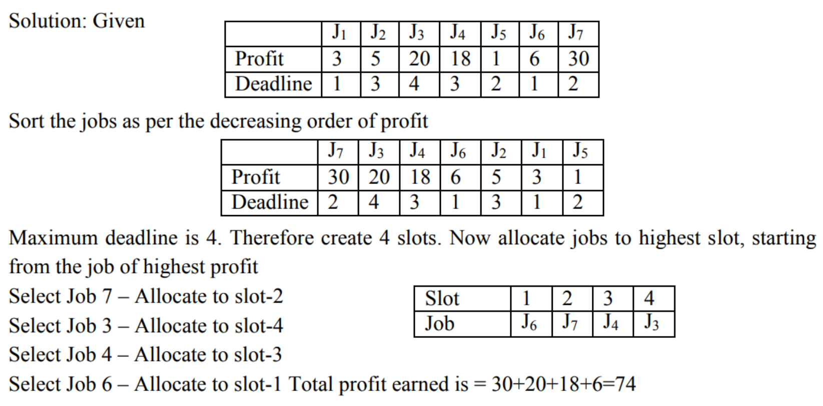


**Minimum shortest path**



**The minimum shortest path :** 3+1+4+2+6=16

**13. Make use of the control abstraction for the subset paradigm using greedy methods. Solve the job sequencing with the deadline problem using a greedy method for the given data N=7,P=3,5,20,18,1,6,30 are profits and D=1,3,4,3,5,1,2 are deadlines respectively.**



Solution: Given

| | $J_1$ | $J_2$ | $J_3$ | $J_4$ | $J_5$ | $J_6$ | $J_7$ |
|---|---|---|---|---|---|---|---|
| Profit | 3 | 5 | 20 | 18 | 1 | 6 | 30 |
| Deadline | 1 | 3 | 4 | 3 | 2 | 1 | 2 |

Sort the jobs as per the decreasing order of profit

| | $J_7$ | $J_3$ | $J_4$ | $J_6$ | $J_2$ | $J_1$ | $J_5$ |
|---|---|---|---|---|---|---|---|
| Profit | 30 | 20 | 18 | 6 | 5 | 3 | 1 |
| Deadline | 2 | 4 | 3 | 1 | 3 | 1 | 2 |

Maximum deadline is 4. Therefore create 4 slots. Now allocate jobs to highest slot, starting from the job of highest profit

Select Job 7 – Allocate to slot-2

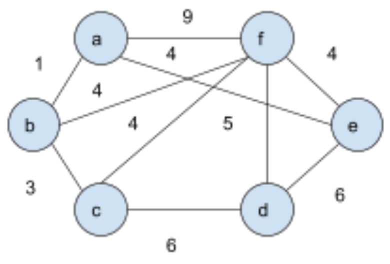
Select Job 3 – Allocate to slot-4

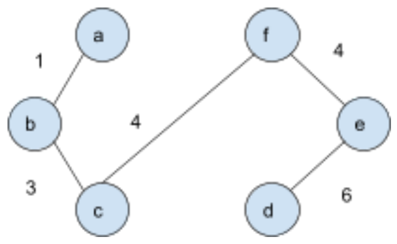
Select Job 4 – Allocate to slot-3

Select Job 6 – Allocate to slot-1 Total profit earned is = 30+20+18+6=74

| Slot | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Job | $J_6$ | $J_7$ | $J_4$ | $J_3$ |

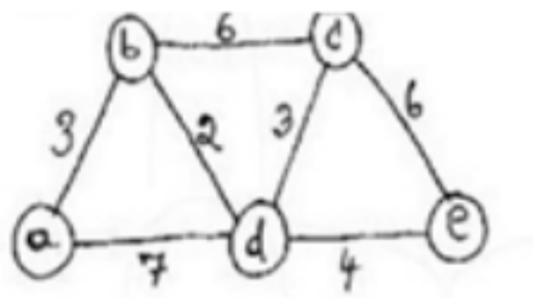
**14. Identify minimum cost spanning tree for a graph G(6,10)with vertices named as a,b,c,d,e,f and edges ab=1,bc=3, af=9,ae=4,ed=6, fe=4,fd=5,cd=6,cf=4 and bf=4 using Kruskal's algorithm and showing results in each stages.**



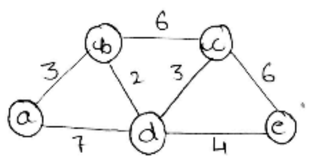


The minimum cost spanning tree is : 1+3+4+4+6=18

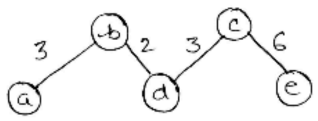
**15. Identify the shortest path from source a to all other vertices in the graph shown in below Fig. Using greedy methods. Give the greedy criterion used.**



15) Using Djikstra's Algorithm,



| visited | a | b | c | d | e |
|---|---|---|---|---|---|
| {a} | 0 | $\underline{3}$ | ∞ | 7 | ∞ |
| {a, b} | 0 | 3 | 9 | $\underline{5}$ | ∞ |
| {a, b, d} | 0 | 3 | $\underline{8}$ | 5 | 9 |
| {a, b, d, c} | 0 | 3 | 8 | 5 | $\underline{14}$ |
| {a, b, d, c, e} | 0 | 3 | 8 | 5 | $\boxed{14}$ |



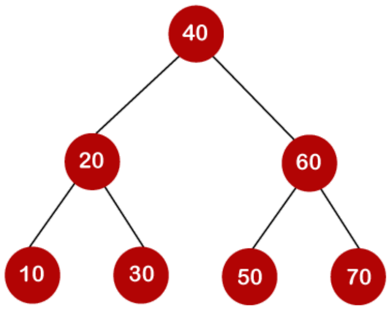
Shortest path = $\underline{\underline{14}}$

**16. Explain optimal binary search tree algorithm with example**

Optimal Binary Search Tree extends the concept of Binary search tree. Binary Search Tree (BST) is a nonlinear data structure which is used in many scientific applications for reducing the search time. In BST, the left child is smaller than root and right child is greater than root. This arrangement simplifies the search procedure.

Optimal Binary Search Tree (OBST) is very useful in dictionary search. The probability of searching is different for different words. OBST has great application in translation. If we translate the book from English to German, equivalent words are searched from the English to German dictionary and replaced in translation. Words are searched the same as in binary search tree order.

Lets understand through example



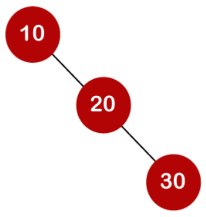
In the above tree, all the nodes on the left subtree are smaller than the value of the root node, and all the nodes on the right subtree are larger than the value of the root node. The maximum time required to search a node is equal to the minimum height of the tree, equal to logn.

The Formula for calculating the number of trees:

$$\frac{{}^{2n}C_n}{n+1}$$

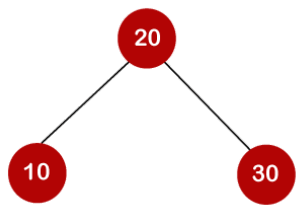
When we use the above formula, then it is found that a total 5 number of trees can be created.

The cost required for searching an element depends on the comparisons to be made to search an element. Now, we will calculate the average cost of time of the above binary search trees.



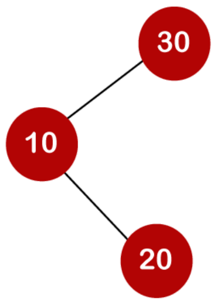
In the above tree, the average number of comparisons that can be made as:

$$average\ number\ of\ comparisons = \frac{1+2+3}{3} = 2$$



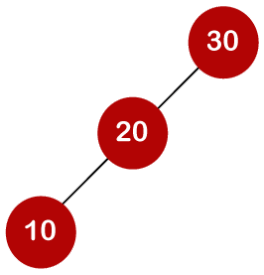
In the above tree, the average number of comparisons that can be made as:

$$average\ number\ of\ comparisons = \frac{1+2+2}{3} = 5/3$$



In the above tree, the total number of comparisons can be made as 3. Therefore, the average number of comparisons that can be made as:

$$average\ number\ of\ comparisons = \frac{1+2+3}{3} = 2$$



In the above tree, the total number of comparisons can be made as 3. Therefore, the average number of comparisons that can be made as:

$$average\ number\ of\ comparisons = \frac{1+2+3}{3} = 2$$

In the third case, the number of comparisons is less because the height of the tree is less, so it's a balanced binary search tree.

for more info go to [Optimal Binary Search Tree - javatpoint]

**17. Explain 0/1 knapsack problem with an example.**

Knapsack problem is also called the rucksack problem.

It is a problem in combinatorial optimization.

Knapsack problem states that: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

There are two versions of the problem:

a. 0/1 Knapsack Problem: Items are indivisible; you either take an item or not. Some special instances can be solved with dynamic programming.

b. Fractional knapsack problem: Items are divisible; you can take any fraction of an item.

0/1 Knapsack Problem:

i. In the 0/1 Knapsack problem, items can be entirely accepted or rejected.

ii. Given a knapsack with maximum capacity W, and a set S consisting of n items.

iii. Each item has some weight wiand benefit value bi(all wiand W are integer values).

iv. The problem is how to pack the knapsack to achieve the maximum total value of packed items.

v. For solving the knapsack problem we can generate the sequence of decisions in order to obtain the optimum selection.

vi. Let Xn be the optimum sequence and there are two instances {Xn} and {Xn-1, Xn-2… X1}.

vii. So from {Xn-1, Xn-2… X1} we will choose the optimum sequence with respect to Xn.

viii. The remaining set should fulfil the condition of filling Knapsack of capacity W with maximum profit.

ix. Thus, 0/1 Knapsack problem is solved using the principle of optimality

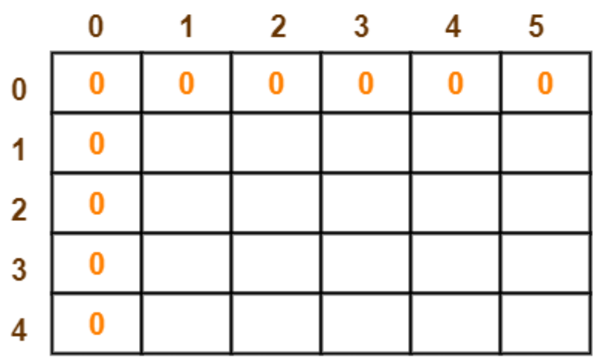
**Example:**

Find the optimal solution for the 0/1 knapsack problem making use of a dynamic programming approach. Consider-

n = 4, w = 5 kg, (w1, w2, w3, w4) = (2, 3, 4, 5), (p1, p2, p3, p4) = (3, 4, 5, 6)

**Answer:**

- Draw a table say 'T' with (n+1) = 4 + 1 = 5 number of rows and (w+1) = 5 + 1 = 6 number of columns.
- Fill all the boxes of 0th row and 0th column with 0.

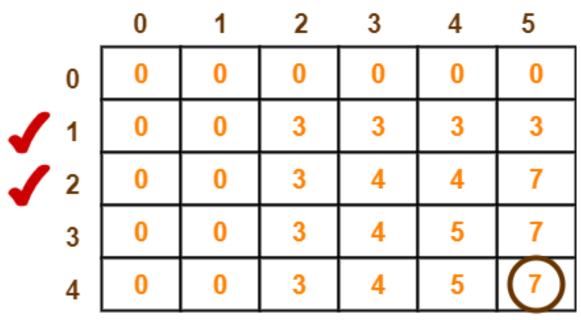


|   | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 |   |   |   |   |   |
| 2 | 0 |   |   |   |   |   |
| 3 | 0 |   |   |   |   |   |
| 4 | 0 |   |   |   |   |   |

Start filling the table row wise top to bottom from left to right using the formula-

**$T(i, j) = \max\{T(i-1, j), \text{profit}_i + T(i-1, j - \text{weight}_i)\}$**

After all the entries are computed and filled in the table, we get the following table-



| | | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ✓ | 1 | 0 | 0 | 3 | 3 | 3 | 3 |
| ✓ | 2 | 0 | 0 | 3 | 4 | 4 | 7 |
| | 3 | 0 | 0 | 3 | 4 | 5 | 7 |
| | 4 | 0 | 0 | 3 | 4 | 5 | 7 |

The last entry represents the maximum possible value that can be put into the knapsack.

So, maximum possible value that can be put into the knapsack = 7.

**18. Explain all pairs shortest path problem with example**

The all pair shortest path algorithm, also known as Floyd-Warshall algorithm, is used to find all pair shortest path problems from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.

At first the output matrix is the same as the given cost matrix of the graph. After that the output matrix will be updated with all vertices k as the intermediate vertex.

The time complexity of this algorithm is $O(V^3)$, here V is the number of vertices in the graph.

**19. Explain the travelling salesman problem and discuss how to solve it using dynamic programming?**

Travelling salesman problem is stated as, “Given a set of n cities and distance between each pair of cities, find the minimum length path such that it covers each city exactly once and terminates the tour at the starting city.”

It is not difficult to show that this problem is an NP-complete problem. There exists n! paths, a search of the optimal path becomes very slow when n is considerably large.

Each edge (u, v) in the TSP graph is assigned some non-negative weight, which represents the distance between city u and v. This problem can be solved by finding the Hamiltonian cycle of the graph.

The distance between cities is best described by the weighted graph, where edge (u, v) indicates the path from city u to v and w(u, v) represents the distance between cities u and v.

Let us formulate the solution of TSP using dynamic programming.

Algorithm for Travelling salesman problem

Step 1:

Let d[i, j] indicate the distance between cities i and j. Function C[x, V – { x }]is the cost of the path starting from city x. V is the set of cities/vertices in a given graph. The aim of TSP is to minimise the cost function.

Step 2:

Assume that graph contains n vertices V1, V2, ..., Vn. TSP finds a path covering all vertices exactly once, and at the same time it tries to minimise the overall travelling distance.

Step 3:

Mathematical formula to find minimum distance is stated below:

$C(i, V) = \min \{ d[i, j] + C(j, V - \{ j \}) \}, j \in V \text{ and } i \notin V.$

The TSP problem possesses the principle of optimality, i.e. for d[V1, Vn] to be minimum, any intermediate path (Vi, Vj) must be minimum.

From following figure, d[i, j] = min(d[i, j], d[i, k] + d[k, j])

Dynamic programming always selects the path which is minimum.

**20. Explain matrix chain multiplication with examples.**

Given a sequence of matrices, find the most efficient way to multiply these matrices together. The problem is not actually to perform the multiplications, but merely to decide in which order to perform the multiplications.

We have many options to multiply a chain of matrices because matrix multiplication is associative. In other words, no matter how we parenthesize the product, the result will be the same. For example, if we had four matrices A, B, C, and D, we would have:

(ABC)D = (AB)(CD) = A(BCD) = ....

However, the order in which we parenthesize the product affects the number of simple arithmetic operations needed to compute the product, or the efficiency. For example, suppose A is a $10 \times 30$ matrix, B is a $30 \times 5$ matrix, and C is a $5 \times 60$ matrix. Then,

$(AB)C = (10\times30\times5) + (10\times5\times60) = 1500 + 3000 = 4500$ operations

$A(BC) = (30\times5\times60) + (10\times30\times60) = 9000 + 18000 = 27000$ operations.

Clearly the first parenthesization requires less number of operations.

Given an array p[] which represents the chain of matrices such that the ith matrix Ai is of dimension p[i-1] x p[i]. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

IARE '24 DISCORD TEAM

II-II

# DAA MODULE 4 SOLUTIONS

VISHAL • UJJWAL

BACKTRACKING AND BRANCH AND BOUND