UNIT 4

Dynamic Programming and Backtracking

CONTENTS

PART-1

Dynamic Programming with Examples such as Knapsack.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.1. What do you mean by dynamic programming ?

OR

What is dynamic programming ? How is this approach different from recursion? Explain with example.

AKTU 2019-20, Marks 07

Answer

1. Dynamic programming is a stage-wise search method suitable for optimization problems whose solutions may be viewed as the result of a sequence of decisions.
2. It is used when the sub-problems are not independent.
3. Dynamic programming takes advantage of the duplication and arranges to solve each sub-problem only once, saving the solution (in table or something) for later use.
4. Dynamic programming can be thought of as being the reverse of recursion. Recursion is a top-down mechanism *i.e.*, we take a problem, split it up, and solve the smaller problems that are created. Dynamic programming is a bottom-up mechanism *i.e.*, we solve all possible small problems and then combine them to obtain solutions for bigger problems.

Difference :

1. In recursion, sub-problems are solved multiple times but in dynamic programming sub-problems are solved only one time.
2. Recursion is slower than dynamic programming.

For example :

Consider the example of calculating n^{th} Fibonacci number.

$$\text{fibo}(n) = \text{fibo}(n-1) + \text{fibo}(n-2)$$
$$\text{fibo}(n-1) = \text{fibo}(n-2) + \text{fibo}(n-3)$$
$$\text{fibo}(n-2) = \text{fibo}(n-3) + \text{fibo}(n-4)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\text{fibo}(2) = \text{fibo}(1) + \text{fibo}(0)$$

In the first three steps, it can be clearly seen that fibo(n – 3) is calculated twice. If we use recursion, we calculate the same sub-problems again and again but with dynamic programming we calculate the sub-problems only once.

Que 4.2. What is the principle of optimality ? Also give approaches in dynamic programming.

Answer

Principle of optimality : Principle of optimality states that whatever the initial state is, remaining decisions must be optimal with regard the state following from the first decision.

Approaches in dynamic programming :

There are two approaches of solving dynamic programming problems :

1. **Bottom-up approach :** Bottom-up approach simply means storing the results of certain calculations, which are then re-used later because the same calculation is a sub-problem in a larger calculation.
2. **Top-down approach :** Top-down approach involves formulating a complex calculation as a recursive series of simpler calculations.

Que 4.3. Discuss the elements of dynamic programming.

Answer

Following are the elements of dynamic programming :

1. **Optimal sub-structure :** Optimal sub-structure holds if optimal solution contains optimal solutions to sub-problems. It is often easy to show the optimal sub-problem property as follows :
   i. Split problem into sub-problems.
   ii. Sub-problems must be optimal; otherwise the optimal splitting would not have been optimal.

   There is usually a suitable "space" of sub-problems. Some spaces are more "natural" than others. For matrix chain multiply we choose sub-problems as sub-chains.
2. **Overlapping sub-problem :**
   i. Overlapping sub-problem is found in those problems where bigger problems share the same smaller problems. This means, while solving larger problems through their sub-problems we find the same sub-problems more than once. In these cases a sub-problem is usually found to be solved previously.
   ii. Overlapping sub-problems can be found in Matrix Chain Multiplication (MCM) problem.

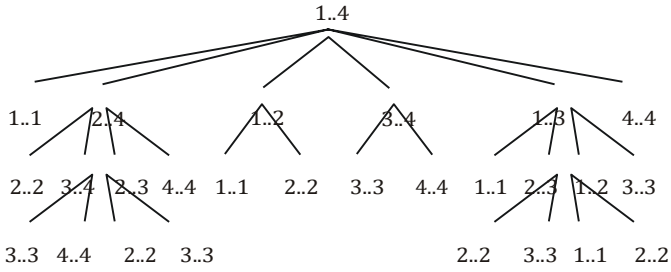


Fig. 4.3.1. The recursion tree for the computation of Recursive-Matrix-Chain (p, 1, 4).

3. Memoization :

i. The memoization technique is the method of storing values of solutions to previously solved problems.

ii. This generally means storing the values in a data structure that helps us reach them efficiently when the same problems occur during the program's execution.

iii. The data structure can be anything that helps us do that but generally a table is used.

Que 4.4. Write down an algorithm to compute Longest Common Subsequence (LCS) of two given strings and analyze its time complexity. **AKTU 2017-18, Marks 10**

Answer

LCS-Length (*X, Y*) :

1. $m \leftarrow$ length[X]
2. $n \leftarrow$ length[Y]
3. for $i \leftarrow 1$ to m
4. do $c[i, 0] \leftarrow 0$
5. for $j \leftarrow 0$ to n
6. do $c[0, j] \leftarrow 0$
7. for $i \leftarrow 1$ to m
8. do for $j \leftarrow 1$ to n
9. do if $x_i = y_j$
10. then $c[i, j] \leftarrow c[i - 1, j - 1] + 1$
11. $b[i, j] \leftarrow$ "▪"
12. else if $c[i - 1, j] \geq c[i, j - 1]$
13. then $c[i, j] \leftarrow c[i - 1, j]$
14. $b[i, j] \leftarrow$ "↑"
15. else $c[i, j] \leftarrow c[i, j - 1]$

16. $b[i, j] \leftarrow$ "$\leftarrow$"

17. return c and b

Note :

1. "▪" means both the same.
2. "↑" means $c[i-1, j] \geq c[i, j-1]$.
3. "←" means $c[i-1, j] < c[i, j-1]$.
4. The "▪" diagonal arrows lengthen the LCS.

Since, two for loops are present in LCS algorithm first for loop runs upto m times and second for loop runs upto n times. So, time complexity of LCS is $O(mn)$.

Que 4.5. Give the algorithm of dynamic 0/1-knapsack problem.

Answer

Dynamic 0/1-knapsack(v, w, n, W) :

```
1.  for (w = 0 to W) V[0, w] = 0
2.  for (i = 1 to n)
3.        for (w = 0 to W)
4.              if (w[i] ≤ w) then
5.                 V[i, w] = max{V[i – 1, w], v[i] + V[i – 1, w – w[i]]};
6.              else V[i, w] = V[i – 1, w];
7.  return V[n, W];
```

Now, as we know that $V[n, W]$ is the total value of selected items, the can be placed in the knapsack. Following steps are used repeatedly to select actual knapsack item.

Let, $i = n$ and $k = W$ then

while (1 > 0 and $k > 0$)

{

if ($V[i, k] \neq V[i-1, k]$) then

mark i^{th} item as in knapsack

$i = i - 1$ and $k = k - w_i$ // selection of i^{th} item

else

$i = i - 1$ //do not select i^{th} item

}

Que 4.6. Differentiate between dynamic programming and greedy approach. What is 0/1 knapsack problem ? Solve the following instance using dynamic programming. Write the algorithm also.

Knapsack Capacity = 10, P = < 1, 6, 18, 22, 28 > and w = < 1, 2, 5, 6, 7>.

Answer

S. No.	Dynamic programming	Greedy approach
1.	Solves every optimal sub-problem.	Do not solve every optimal problem.
2.	We make a choice at each step, but the choice may depend on the solutions to sub-problem.	We make whatever choice seems best at the moment and then solve the problem.
3.	It is bottom-up approach.	It is top-down approach.
4.	Dynamic programming works when a problem has following properties : a. Optimal sub-structure b. Overlapping sub-problems	Greedy algorithm works when a problem exhibits the following properties : a. Greedy choice property b. Optimal sub-structure

0/1-knapsack problem : Refer Q. 3.18, Page 3–17B, Unit-3.
0/1-kanpsack algorithm : Refer Q. 4.5, Page 4–5B, Unit-4.
Numerical :

Item	w_i	$p_i = v_i/w_i$ (Given)	v_i
I_1	1	1	1
I_2	2	6	12
I_3	5	18	90
I_4	6	22	132
I_5	7	28	196

Now, fill the knapsack according to given value of p_i.
First we choose item I_1 whose weight is 1, then choose item I_2 whose weight is 2 and item I_3 whose weight is 5.
$\therefore$ Total weight in knapsack : 1 + 2 + 5 = 8
Now, the next item is I_4 and its weight is 6, but we want only 2 {$\because W = 10$}. So we choose fractional part of it *i.e.*, The value of fractional part of I_4 is = 132/6 × 2 = 44 Thus the maximum value is = 1 + 12 + 90 + 44 = 147

$$\begin{array}{r} 2 \\ 5 \\ 2 \\ \hline 1 \end{array} \Big\rfloor = 10$$

Que 4.7. Discuss knapsack problem with respect to dynamic programming approach. Find the optimal solution for given problem, *w* (weight set) = {5, 10, 15, 20} and *W* (Knapsack size) = 25 and *v* = {50, 60, 120, 100}.

AKTU 2015-16, Marks 10

Answer

Knapsack problem with respect to dynamic programming approach : Refer Q. 4.5, Page 4–5B, Unit-4.

Numerical :

$w = \{5, 10, 15, 20\}$
$W = 25$
$v = \{50, 60, 120, 100\}$

Initially,

Item	w_i	v_i
I_1	5	50
I_2	10	60
I_3	15	120
I_4	20	100

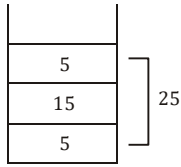
Taking value per weight ratio, *i.e.*, $p_i = v_i/w_i$

Item	w_i	v_i	$p_i = v_i/w_i$
I_1	5	50	10
I_2	10	60	6
I_3	15	120	8
I_4	20	100	5

Now, arrange the value of p_i in decreasing order.

Item	w_i	v_i	$p_i = v_i/w_i$
I_1	5	50	10
I_3	15	120	8
I_2	10	60	6
I_4	20	100	5

Now, fill the knapsack according to decreasing value of p_i.
First we choose item I_1 whose weight is 5, then choose item I_3 whose weight is 15. Now the total weight in knapsack is 5 + 15 = 20. Now, next item is I_2 and its weight is 10, but we want only 5. So, we choose fractional part of it, *i.e.*,



The value of fractional part of I_2 is,

$$= \frac{60}{10} \times 5 = 30$$

Thus, the maximum value is,

$$= 50 + 120 + 3 = 200$$

Que 4.8. Compare the various programming paradigms such as divide-and-conquer, dynamic programming and greedy approach.

AKTU 2019-20, Marks 07

Answer

S. No.	Divide and conquer approach	Dynamic programming approach	Greedy approach
1.	Optimizes by breaking down a subproblem into simpler versions of itself and using multi-threading and recursion to solve.	Same as Divide and Conquer, but optimizes by caching the answers to each subproblem as not to repeat the calculation twice.	Optimizes by making the best choice at the moment.
2.	Always finds the optimal solution, but is slower than Greedy.	Always finds the optimal solution, but cannot work on small datasets.	Does not always find the optimal solution, but is very fast.
3.	Requires some memory to remember recursive calls.	Requires a lot of memory for tabulation.	Requires almost no memory.

PART-2

All Pair Shortest Paths : Warshall's and Floyd's Algorithm, Resource Allocation Problem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.9. Describe the Warshall's and Floyd's algorithm for finding all pairs shortest paths.

Answer

1. Floyd-Warshall algorithm is a graph analysis algorithm for finding shortest paths in a weighted, directed graph.
2. A single execution of the algorithm will find the shortest path between all pairs of vertices.
3. It does so in $\Theta(V^3)$ time, where V is the number of vertices in the graph.
4. Negative-weight edges may be present, but we shall assume that there are no negative-weight cycles.
5. The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex of a simple path $p = (v_1, v_2, ..., v_m)$ is any vertex of p other than v_1 or v_m, that is, any vertex in the set $\{v_2, v_3, ..., v_{m-1}\}$.
6. Let the vertices of G be $V = \{1, 2, ..., n\}$, and consider a subset $\{1, 2, ..., k\}$ of vertices for some k.
7. For any pair of vertices $i, j \in V$, consider all paths from i to j whose intermediate vertices are all drawn from $\{1, 2, ..., k\}$, and let p be a minimum-weight path from among them.
8. Let $d_{ij}^{(k)}$ be the weight of a shortest path from vertex i to vertex j with all intermediate vertices in the set $\{1, 2, ..., k\}$.

   A recursive definition is given by

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1 \end{cases}$$

Floyd-Warshall (W) :

1. $n \leftarrow$ rows $[W]$
2. $D^{(0)} \leftarrow W$
3. for $k \leftarrow 1$ to n
4. do for $i \leftarrow 1$ to n
5. do for $j \leftarrow 1$ to n
6. do $d^{(k)} \leftarrow \min(d^{(k-1)}, d^{(k-1)} + d^{(k-1)})$

d_{ij} d_{ij} d_{ik} d_{kj}

7. return $D^{(n)}$

Que 4.10. **Define Floyd Warshall algorithm for all pair shortest path and apply the same on following graph :**

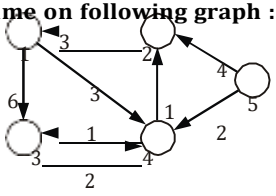


Fig. 4.10.1.

AKTU 2019-20, Marks 07

Answer

Floyd Warshall algorithm : Refer Q. 4.9, Page 4–9B, Unit-4.

Numerical :

$$d_{ij}^{(k)} = \min[d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}]$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \le d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

$$D^{(0)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & \infty & 6 & 3 & \infty \\ 2 & 3 & 0 & \infty & \infty & \infty \\ 3 & \infty & \infty & 0 & 2 & \infty \\ 4 & \infty & 1 & 1 & 0 & \infty \\ 5 & \infty & 4 & \infty & 2 & 0 \end{array}$$

$$D^{(1)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 4 & 6 & 3 & \infty \\ 2 & 3 & 0 & 9 & 6 & \infty \\ 3 & 6 & 4 & 0 & 2 & \infty \\ 4 & 4 & 1 & 1 & 0 & \infty \\ 5 & 7 & 4 & 3 & 2 & 0 \end{array}$$

$$D^{(2)} = \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & 4 & 6 & 3 & \infty \\ 2 & 3 & 0 & 7 & 6 & \infty \\ 3 & 6 & 3 & 0 & 2 & \infty \\ 4 & 4 & 1 & 1 & 0 & \infty \\ 5 & 6 & 3 & 3 & 2 & 0 \end{array}$$

Now, if we find $D^{(3)}$, $D^{(4)}$ and $D^{(5)}$ there will be no change in the entries.

Que 4.11. Give Floyd-Warshall algorithm to find the shortest path for all pairs of vertices in a graph. Give the complexity of the algorithm. Explain with example. **AKTU 2018-19, Marks 07**

Answer

Floyd-Warshall algorithm : Refer Q. 4.9, Page 4–9B, Unit-4.

Time complexity of Floyd-Warshall algorithm is $O(n^3)$.

Example :

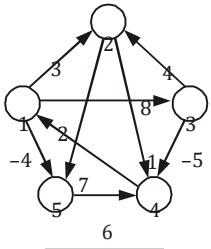


Fig. 4.11.1.

$$d_{ij}^{(k)} = \min[d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}]$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} \text{ if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} \text{ if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

$$D^{(0)} = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}; \pi^{(0)} = \begin{bmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{bmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}; \pi^{(1)} = \begin{bmatrix} NIL & 1 & 1 & NIL & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & NIL & NIL \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}; \pi^{(2)} = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 1 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}; \pi^{(3)} = \begin{bmatrix} NIL & 1 & 1 & 2 & 1 \\ NIL & NIL & NIL & 2 & 2 \\ NIL & 3 & NIL & 2 & 2 \\ 4 & 3 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}; \pi^{(4)} = \begin{bmatrix} NIL & 1 & 4 & 2 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{bmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}; \pi^{(5)} = \begin{bmatrix} NIL & 3 & 4 & 5 & 1 \\ 4 & NIL & 4 & 2 & 1 \\ 4 & 3 & NIL & 2 & 1 \\ 4 & 3 & 4 & NIL & 1 \\ 4 & 3 & 4 & 5 & NIL \end{bmatrix}$$

PART-3

Backtracking, Branch and Bound with Examples such as Travelling Salesman Problem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.12. What is backtracking ? Write general iterative algorithm for backtracking. **AKTU 2016-17, Marks 10**

Answer

1. Backtracking is a general algorithm for finding all solutions to some computational problems.
2. Backtracking is an important tool for solving constraint satisfaction problems, such as crosswords, verbal arithmetic, and many other puzzles.
3. It is often the most convenient (if not the most efficient) technique for parsing, for the knapsack problem and other combinational optimization problem.
4. It can be applied only for problems which admit the concept of a "partial candidate solution" and a relatively quick test of whether it can possibly be completed to a valid solution.

Iterative backtracking algorithm :

```
algorithm ibacktrack (n)
// Iterative backtracking process
// All solutions are generated in x[1 : n] and printed as soon as they are found
{
k = 1;
while (k != 0)
{
if ( there remains an untried x[k] in T(x[1], x[2], ..., x[k – 1])
and B_k(x[1], ..., x[k] ) is true )
{
if (x[1], ..., x[k] is a path to an answer node )
write (x[1 : k]);
k = k + 1;                    // Consider the next set
}
else
k = k – 1;                    // Backtrack to the previous set
}
}
```

Que 4.13. Describe backtracking algorithm for Travelling Salesman Problem (TSP). Show that a TSP can be solved using backtracking method in the exponential time.

OR

Explain TSP (Travelling Salesman) problem with example. Write an approach to solve TSP problem. **AKTU 2015-16, Marks 10**

Answer

Travelling Salesman Problem (TSP) :

Travelling salesman problem is the problem to find the shortest possible route for a given set of cities and distance between the pair of cities that visits every city exactly once and returns to the starting point.

Backtracking approach is used to solve TSP problem.

Backtracking algorithm for the TSP :

1. Let G be the given complete graph with positive weights on its edges.
2. Use a search tree that generates all permutations of $V = \{1 \ldots n\}$, specifically the one illustrated in Fig. 4.13.1 for the case $n = 3$.

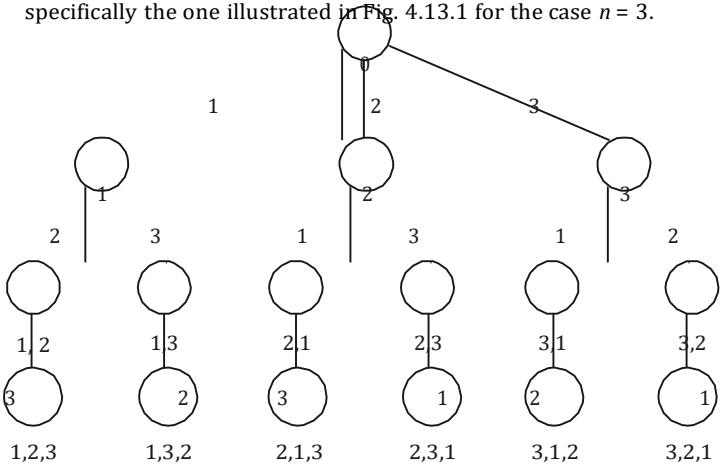


Fig. 4.13.1.

3. A node at depth i of this tree (the root is at depth 0) stores an i-permutation of $\{1, ..., n\}$. A leaf stores a permutation of $\{1, ..., n\}$, whichis equivalent to saying that it stores a particular Hamiltonian cycle (tour) of G.
4. For the travelling salesman problem, we will not do any static pruning on this tree, we will do dynamic pruning, during the search.

Proof :

1. At some point during the search, let v be a non-leaf node of this tree that is just being visited, and let w be the weight of the shortest tour found to this point.

2. Let $(\pi_1\pi \ldots _k)$ be the k-permutation of $\{1 \ldots n\}$ stored at v.

Let, $w_v = \sum_{i=1....k=1} w_{\pi_i \pi_{i+1}}$ denote the sum of the weights on edges whose endpoints are adjacent vertices in this k-permutation.

3. Then, if $w_v \geq w$, the entire subtree of the search tree rooted at v can be pruned, *i.e.*, not searched at all, since every leaf of this subtree represents a tour whose weight must be greater than w_v.
4. This follows from the fact that all edges in the graph have positive weights.
5. There are atmost $O(n*2^n)$ subproblem, and each one takes linear time to solve.
6. The total running time is therefore $O(n^2*2^n)$.
7. The time complexity is much less than $O(n!)$ but still exponential.

Hence proved.

Que 4.14. Explain the working rule for Travelling Salesman Problem using branch and bound technique.

Answer

For solving travelling salesman problem we represent the solution space by a state space tree. We define three cost functions C, l and u where $l_i \leq C_i \leq u_i$ for all the nodes i of the state space tree.

Step 1 : First find the cost matrix as given cost on the edges of the graph.

Step 2 : Now, find reduced matrix by subtracting the smallest element from row i (column j) introduce a zero in to row i (column j). Repeating this process as often as needed, the cost matrix can be reduced.

Add the total amount subtracted from the column and rows and make this as the root of the state space tree.

Step 3 : Let M be the reduced cost matrix for node A and let B be a child of A such that the tree edge (A, B) corresponds to inclusion of edge (i, j) in the tour. If B is not a leaf, then the reduced cost matrix for B may be obtained by apply following given steps :

a. Change all the entries in row i, column j of M to ∞. This includes use of any more edges leaving vertex i or entering vertex j.
b. Set $M(j, 1) = \infty$. This excludes use of the edge $(j, 1)$.
c. Reduce all the rows and columns in the resulting matrix except for rows and columns containing only ∞.

Suppose T is the total amount subtracted in step (c), then

$$I(B) = I(A) + M(i, j) + T$$

For leaf nodes $l = c$ is easily computed as each leaf defines a unique tour. For the upper bound u, we assume $u_i = \infty$ for all nodes i.

Step 4 : Find the root of the node as, combine the total amount subtracted from cost matrix to find the reduced cost matrix M.

Que 4.15. What is travelling salesman problem ? Find the solution of following travelling salesman problem using branch and bound method.

$$\text{Cost matrix} = \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 6 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

AKTU 2016-17, Marks 10

Answer

Travelling salesman problem : Refer Q. 4.13, Page 4–13B, Unit-4.

Numerical :

$$\text{Cost matrix} = \begin{bmatrix} \infty & 20 & 30 & 10 & 11 \\ 15 & \infty & 16 & 4 & 2 \\ 3 & 5 & \infty & 2 & 4 \\ 19 & 6 & 6 & \infty & 3 \\ 16 & 4 & 7 & 16 & \infty \end{bmatrix}$$

1. Reduce each column and row by reducing the minimum value from each element in row and column.

Row **Column**

$$\begin{array}{c} \\ 10 \rightarrow \\ 2 \rightarrow \\ 2 \rightarrow \\ 3 \rightarrow \\ 4 \rightarrow \end{array} \begin{bmatrix} \infty & 10 & 20 & 0 & 1 \\ 13 & \infty & 14 & 2 & 0 \\ 1 & 3 & \infty & 0 & 2 \\ 16 & 3 & 3 & \infty & 0 \\ 12 & 0 & 3 & 12 & \infty \end{bmatrix} \quad \begin{array}{c} \\ 1 \downarrow \\ \\ \\ \\ \end{array} \quad \begin{array}{c} \\ \\ \\ 3 \downarrow \\ \\ \end{array} \quad \begin{bmatrix} \infty & 10 & 17 & 0 & 1 \\ 12 & \infty & 11 & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & 0 & \infty & 0 \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix} = M_1$$

2. So, total expected cost is : 10 + 2 + 2 + 3 + 4 + 1 + 3 = 25.
3. We have discovered the root node V_1 so the next node to be expanded will be V_2, V_3, V_4, V_5. Obtain cost of expanding using cost matrix for node 2.
4. Change all the elements in 1st row and 2nd column.

$$M_2 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 0 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

5. Now, reducing M_2 in rows and columns, we get :

$$M_2 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & 0 \\ 0 & \infty & \infty & 0 & 2 \\ 15 & \infty & 0 & \infty & 0 \\ 11 & \infty & 0 & 12 & \infty \end{bmatrix}$$

$\therefore$ Total cost for $M_2 = 25 + 10 + 0 = 35$

6. Similarly, for node 3, we have :

$$M_3 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix}$$

7. Now, reducing M_3, we get :

$$M_3 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & 2 & 0 \\ 0 & 3 & \infty & 0 & 2 \\ 15 & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & 12 & \infty \end{bmatrix}$$

$\therefore$ Total cost for $M_3 = 25 + 17 + 0 = 42$

8. Similarly, for node 4, we have :

$$M_4 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ 15 & 3 & 0 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

9. Now, reducing M_4, we get :

$$M_4 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ 15 & 3 & 0 & \infty & 0 \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 0 + 0 = 25

10. Similarly, for node 5, we have :

$$M_5 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & 2 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 0 & \infty & \infty \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

11. Now, reducing M_5, we get :

$$M_5 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 10 & \infty & 9 & 0 & \infty \\ 0 & 3 & \infty & 0 & \infty \\ 15 & 3 & 0 & \infty & \infty \\ 11 & 0 & 0 & 12 & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 1 + 2 = 28

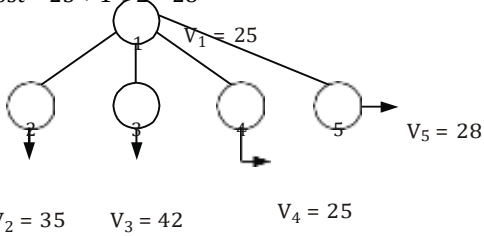


Fig. 4.15.1.

12. Now, the promising node is $V_4 = 25$. Now, we can expand V_2, V_3 and V_5. Now, the input matrix will be M_4.

13. Change all the elements in 4^{th} row and 2^{nd} column.

$$M_6 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \\ 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

14. On reducing M_6, we get :

$$\begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & 0 \end{bmatrix}$$

$$M_6 = \begin{bmatrix} 0 & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty \\ 11 & \infty & 0 & \infty & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 3 + 0 = 28

15. Similarly, for node 3, we have :

$$M_7 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

16. On reducing M_7, we get :

$$M_7 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ 0 & 3 & \infty & \infty & 2 \\ \infty & 3 & \infty & \infty & 0 \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 0 + 0 = 25

17. Similarly, for node 5, we have :

$$M_8 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & 11 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

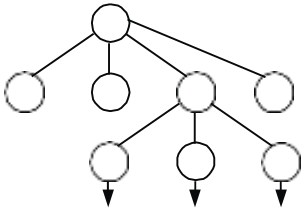
18. On reducing M_8, we get :

$$M_8 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & 0 & \infty & \infty \\ 0 & 3 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ 11 & 0 & 0 & \infty & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 0 + 11 = 36

Fig. 4.15.2.

19. Now, promising node is $V_3 = 25$. Now, we can expand V_2 and V_5. Now, the input matrix will be M_7.



20. Change all the elements in 3rd row and 2nd column.

$$M_9 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ 11 & \infty & \infty & \infty & \infty \end{bmatrix}$$

21. On reducing M_9, we get :

$$M_9 = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 1 & \infty & \infty & \infty & 0 \\ \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 \\ 0 & \infty & \infty & \infty & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 3 + 0 = 28

22. Similarly, for node 5, we have :

$$M_{10} = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 12 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 3 & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

23. On reducing M_{10}, we get :

$$M_{10} = \begin{bmatrix} \infty & \infty & \infty & \infty & \infty \\ 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty & \infty \\ 11 & 0 & \infty & \infty & \infty \end{bmatrix}$$

$\therefore$ Total cost = 25 + 2 + 12 + 3 = 42.

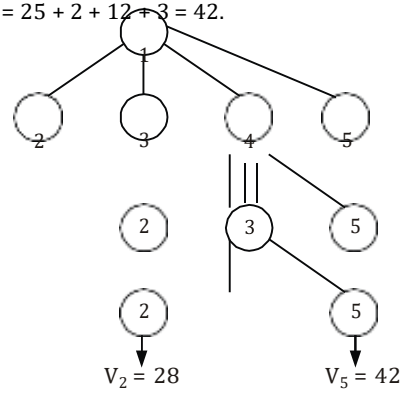


Fig. 4.15.3.

24. Here V_2 is the most promising node so next we are going to expand this node further. Now, we are left with only one node not yet traversed which is V_5.

$$V \xrightarrow[1]{10} V \xrightarrow[4]{6} V \xrightarrow[3]{5} V \xrightarrow[2]{2} V \xrightarrow[5]{16} V \xrightarrow[1]{}$$

So, total cost of traversing the graph is :

10 + 6 + 5 + 2 + 16 = 39

PART-4

Graph Colouring, N-Queen Problem.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.16. Write short notes on graph colouring.

Answer

1. Graph colouring is a simple way of labelling graph components such as vertices, edges, and regions under some constraints.
2. In a graph, no two adjacent vertices, adjacent edges, or adjacent regions are coloured with minimum number of colours. This number is called the chromatic number and the graph is called a properly coloured graph.
3. While graph colouring, the constraints that are set on the graph are colours, order of colouring, the way of assigning colour, etc.
4. A colouring is given to a vertex or a particular region. Thus, the vertices or regions having same colours form independent sets.

For example :

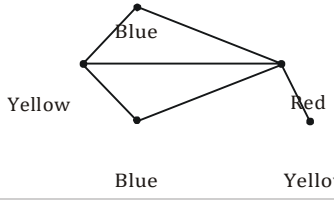


Fig. 4.16.1. Properly coloured graph.

Que 4.17. Write short notes on *N*-Queens problem.

OR

Write pseudocode for 8-Queens problem.

AKTU 2016-17, Marks 10

Answer

1. In *N*-Queens problem, the idea is to place queens one by one in different columns, starting from the leftmost column.
2. When we place a queen in a column, we check for clashes with already placed queens.
3. In the current column, if we find a row for which there is no clash, we mark this row and column as part of the solution.
4. If we do not find such a row due to clashes then we backtrack and return false.

Procedure for solving *N*-Queens problem :

1. Start from the leftmost column.
2. If all queens are placed return true.
3. Try all rows in the current column. Do following for every tried row :
   a. If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
   b. If placing queen in [row, column] leads to a solution then return true.
   c. If placing queen does not lead to a solution then unmark this [row, column] (backtrack) and go to step (a) to try other rows.
4. If all rows have been tried and nothing worked, return false to trigger backtracking.

Algorithm/pseudocode for *N*-Queens problem :

N-Queens are to be placed on an $n \times n$ chessboard so that no two attack *i.e.*, no two Queens are on the same row, column or diagonal.

PLACE (*k*, *i*)

1. for $j \leftarrow 1$ to $k - 1$
2. do if $(x(j) = i)$ or Abs $(x[j] - i) = (\text{Abs } (j - k))$
3. then return false
4. return true

Place (k, i) returns true if a queen can be placed in the k^{th} row and i^{th} column otherwise return false.

$x[\ ]$ is a global array whose first $k - 1$ values have been set. Abs(r) returns the absolute value of r.

N*-Queens *(k, n)

```
1.  for i ← 1 to n
2.      do if PLACE (k, i)
3.          then x[k] ← i
4.  if k = n, then print x[1 .... N]
5.  else N-Queens (k + 1, n)
```

[**Note :** For 8-Queen problem put $n = 8$ in the algorithm.]

Que 4.18. Write an algorithm for solving *N*-Queens problem. Show the solution of 4-Queens problem using backtracking approach. **AKTU 2015-16, Marks 10**

Answer

Algorithm for *N*-Queens problem : Refer Q. 4.17, Page 4–21B, Unit-4.

4- Queens problem :

1. Suppose we have 4 × 4 chessboard with 4-queens each to be placed in non-attacking position.

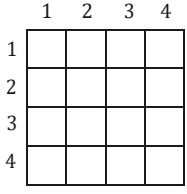


Fig. 4.18.1.

2. Now, we will place each queen on a different row such that no two queens attack each other.
3. We place the queen q_1 in the very first accept position (1, 1).
4. Now if we place queen q_2 in column 1 and 2 then the dead end is encountered.
5. Thus, the first acceptable position for queen q_2 is column 3 *i.e.*, (2, 3) but then no position is left for placing queen q_3 safely. So, we backtrack one step and place the queen q_2 in (2, 4).
6. Now, we obtain the position for placing queen q_3 which is (3, 2). But later this position lead to dead end and no place is found where queen q_2 can be placed safely.

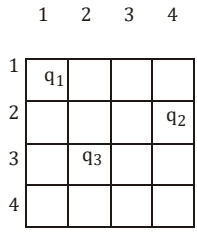


Fig. 4.18.2.

7. Then we have to backtrack till queen q_1 and place it to (1, 2) and then all the other queens are placed safely by moving queen q_2 to (2, 4), queen q_3 to (3, 1) and queen q_4 to (4, 3) *i.e.*, we get the solution < 2, 4, 1, 3>. This is one possible solution for 4-queens problem.

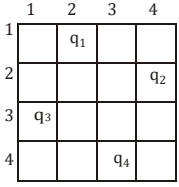


Fig. 4.18.3.

8. For other possible solution the whole method is repeated for all partial solutions. The other solution for 4-queens problem is <3, 1, 4, 2> *i.e.*,

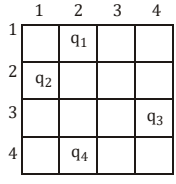


Fig. 4.18.4.

9. Now, the implicit tree for 4-queen for solution <2, 4, 1, 3> is as follows :

10. Fig. 4.18.5 shows the complete state space for 4-queens problem. But we can use backtracking method to generate the necessary node and stop if next node violates the rule *i.e.*, if two queens are attacking.

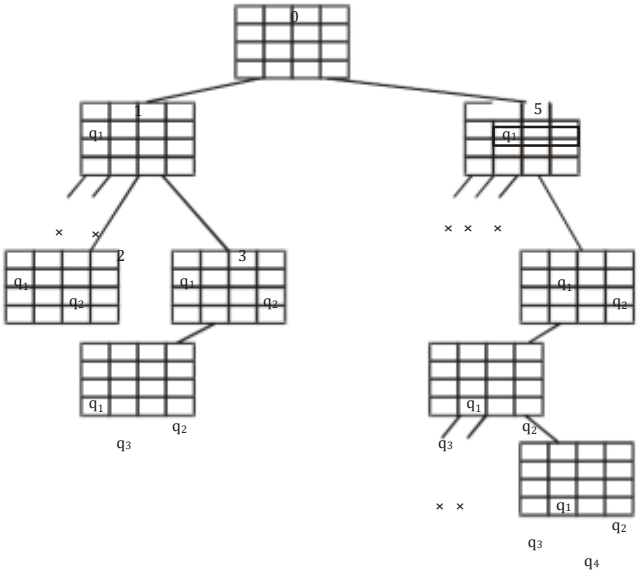


Fig. 4.18.5.

Que 4.19. What is branch and bound technique ? Find a solution to the 4-Queens problem using branch and bound strategy. Draw the solution space using necessary bounding function.

Answer

Branch and bound technique :

1. It is a systematic method for solving optimization problems.
2. It is used where backtracking and greedy method fails.
3. It is sometimes also known as best first search.
4. In this approach we calculate bound at each stage and check whether it is able to give answer or not.
5. Branch and bound procedure requires two tools :
   a. The first one is a way of covering the feasible region by several smaller feasible sub-regions. This is known as branching.
   b. Another tool is bounding, which is a fast way of finding upper and lower bounds for the optimal solution within a feasible sub-region.

Solution to 4-Queens problem :

Basically, we have to ensure 4 things :

1. No two queens share a column.

2. No two queens share a row.

3. No two queens share a top-right to left-bottom diagonal.

4. No two queens share a top-left to bottom-right diagonal.

Number 1 is automatic because of the way we store the solution. For number 2, 3 and 4, we can perform updates in O(1) time and the whole updation process is shown in Fig. 4.19.1.

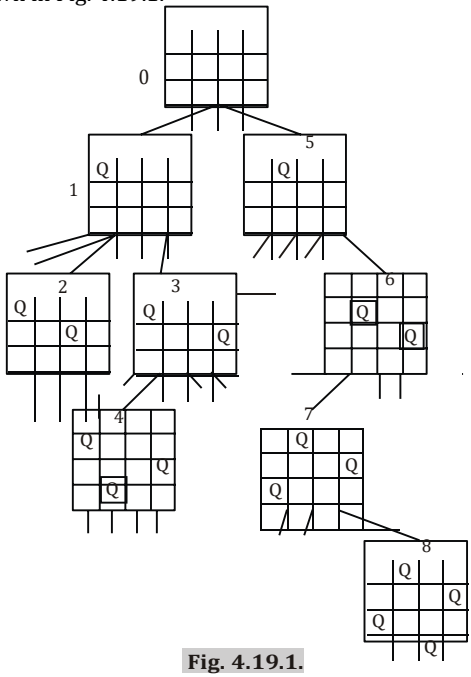


Fig. 4.19.1.

PART-5

Hamiltonian Cycles and Sum of Subsets.

Questions-Answers

Long Answer Type and Medium Answer Type Questions

Que 4.20. What is backtracking ? Discuss sum of subset problem with the help of an example. **AKTU 2017-18, Marks 10**

Answer

Backtracking : Refer Q. 4.12, Page 4–13B, Unit-4.

Sum of subset problem with example :

In the subset-sum problem we have to find a subset s' of the given set $S = (S_1, S_2, S_3, ..., S_n)$ where the elements of the set S are n positive integers in such a manner that $s' \in S$ and sum of the elements of subset 's'' is equal to some positive integer 'X'.

Algorithm for sum-subset problem :

Subset-Sum (S, t)

1. $C \leftarrow \phi$
2. $Z \leftarrow S$
3. $K \leftarrow \phi$
4. $t_1 \leftarrow t$
5. while ($Z \neq \phi$) do
6. $K \leftarrow \max(Z)$
7. if ($K < t$) then
8. $Z \leftarrow Z - K$
9. $t_1 \leftarrow t_1 - K$
10. $C \leftarrow C \cup K$
11. else $Z \leftarrow Z - K$
12. print C // Subset Sum elements whose // Sum is equal to t_1

This procedure selects those elements of S whose sum is equal to t. Every time maximum element is found from S, if it is less than t then this element is removed from Z and also it is subtracted from t.

For example :

Given S = <1, 2, 5, 7, 8, 10, 15, 20, 25> & m = 35

$$Z \leftarrow S,\ m = 35$$

$$k \leftarrow \max[Z] = 25$$

$$K < m$$

$$\therefore \quad Z = Z - K$$

i.e., Z = <1, 2, 5, 7, 8, 10, 15, 20> & $m_1 \leftarrow m$

Subtracting K from m_1, we get

New $m_1 = m_1$(old) $- K = 35 - 25 = 10$

In new step,

$$K \leftarrow \max[Z] = 20$$

$$K > m_1$$

i.e., Z = <1, 2, 5, 7, 8, 10, 15>

In new step,

$K \leftarrow \max[Z] = 15$

$K > m_1$

i.e., $Z = <1, 2, 5, 7, 8, 10>$

In new step,

$K \leftarrow \max[Z] = 10$

$K > m_1$

i.e., $Z = <1, 2, 5, 7, 8>$

In new step,

$K \leftarrow \max[Z] = 8$

$K > m_1$

i.e., $Z = <1, 2, 5, 7> \ \& \ m_2 \leftarrow m_1$

New $m_2 = m_2(\text{old}) - K = 10 - 8 = 2$

In new step

$K \leftarrow \max[Z] = 7$

$K > m_2$

i.e., $Z = <1, 2, 5>$

In new step, $K \leftarrow \max[Z] = 5$

$K > m_2$

i.e., $Z = <1, 2>$

In new step, $K \leftarrow \max[Z] = 2$

$K > m_2$

i.e., $Z = <1>$

In new step,

$K = 1$

$K < m_2$

$\therefore \quad m_3 = 01$

Now only those numbers are needed to be selected whose sum is 01, therefore only 1 is selected from Z and rest other number found as $\max[Z]$ are subtracted from Z one by one till Z become ϕ.

Que 4.21. **Solve the subset sum problem using backtracking, where $n = 4$, $m = 18$, $w[4] = \{5, 10, 8, 13\}$.** **AKTU 2018-19, Marks 07**

Answer

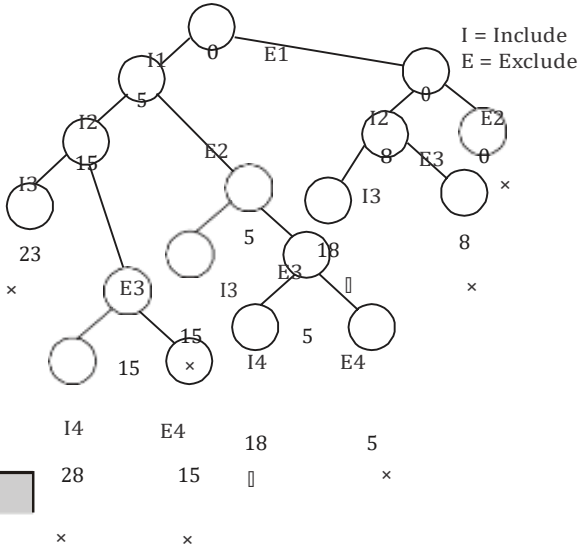
$n = 4$

$m = 18$

$w\{4\} = \{5, 10, 8, 13\}$

Sorted order : $w\{4\} = \{5, 8, 10, 13\}$

Now we construct state-space tree.



First Subset $S_1 = \{5, 13\}$

Similarly, $S_2 = \{8, 10\}$

Que 4.22. Differentiate between backtracking and branch and bound approach. Write an algorithm for sum-subset problem using backtracking approach.

Answer

Difference between backtracking and branch and bound technique :

S. No.	Backtracking	Branch and bound
1.	It is a methodological way of trying out various sequences of decisions.	It is a systematic method for solving optimization problems.
2.	It is applied in dynamic programming technique.	It is applied where greedy and dynamic programming technique fails.
3.	It is sometimes called depth first search.	It is also known as best first search.
4.	This approach is effective for decision problem.	This approach is effective for optimization problems.

5.	This algorithm is simple and easy to understand.	This algorithm is difficult to understand.

Algorithm for sum-subset problem : Refer Q. 4.20, Page 4–26B, Unit-4.

Que 4.23. Explain Hamiltonian circuit problem. Consider a graph *G* = (*V*, *E*) shown in Fig. 4.23.1 and find a Hamiltonian circuit using backtracking method.

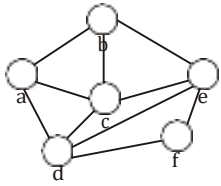


Fig. 4.23.1.

Answer

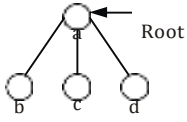
Hamiltonian circuit problem :

1. Given a graph *G* = (*V*, *E*), we have to find the Hamiltonian circuit using backtracking approach.
2. We start our search from any arbitrary vertex, say '*a*'. This vertex '*a*' becomes the root of our implicit tree.
3. The first element of our partial solution is the first intermediate vertex of the Hamiltonian cycle that is to be constructed.
4. The next adjacent vertex is selected on the basis of alphabetical (or numerical) order.
5. If at any stage any arbitrary vertex makes a cycle with any vertex other than vertex '*a*' then we say that dead end is reached.
6. In this case we backtrack one step, and again search begins by selecting another vertex and backtrack the element from the partial solution must be removed.
7. The search using backtracking is successful if a Hamiltonian cycle is obtained.

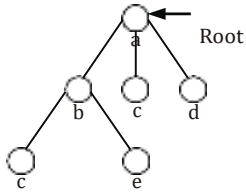
Numerical :

1. Firstly, we start our search with vertex '*a*', this vertex '*a*' becomes the root of our implicit tree.

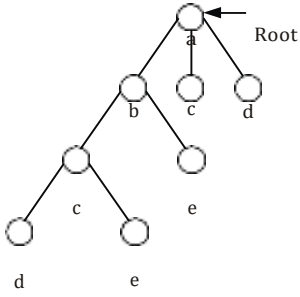
2. Next, we choose vertex '*b*' adjacent to '*a*' as it comes first in lexicographical order (*b*, *c*, *d*).



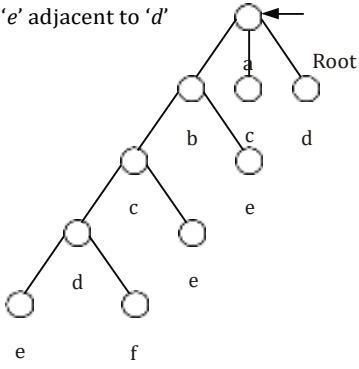
3. Next, we select '*c*' adjacent to '*b*'



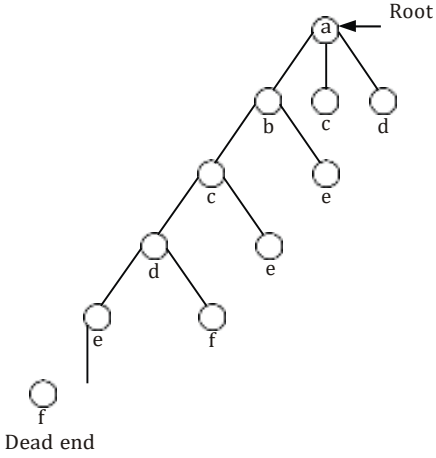
4. Next, we select '*d*' adjacent to '*c*'



5. Next, we select '*e*' adjacent to '*d*'

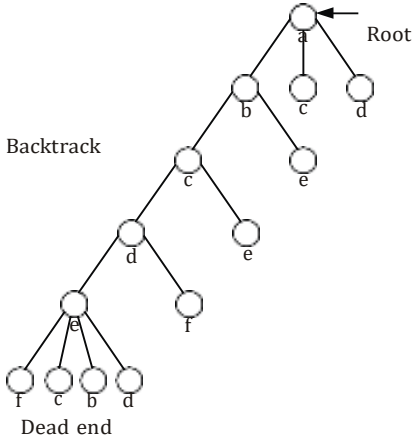


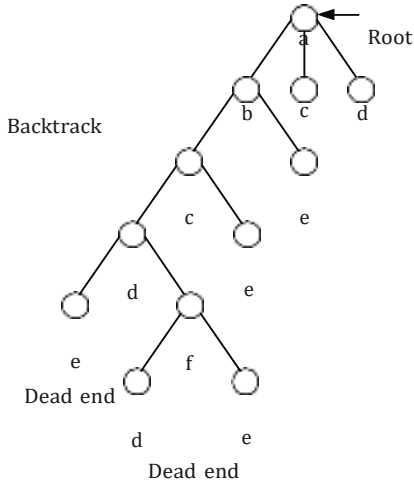
6. Next, we select vertex '*f*' adjacent to '*e*'. The vertex adjacent to '*f*' are '*d*' and '*e*' but they have already visited. Thus, we get the dead end and we backtrack one step and remove the vertex '*f*' from partial solution.



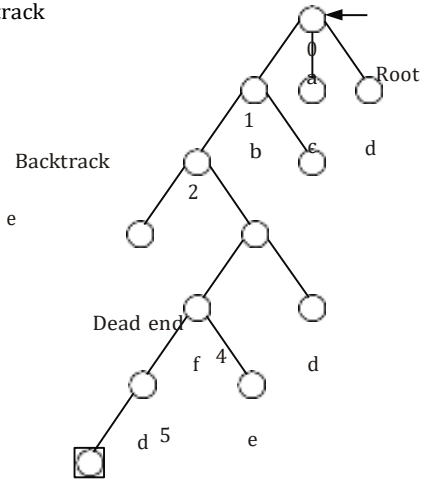
7. From backtracking, the vertex adjacent to '*e*' are '*b*', '*c*', '*d*', '*f*' from which vertex '*f* ' has already been checked and '*b*', '*c*', '*d*' have already visited. So, again we backtrack one step. Now, the vertex adjacent to '*d*' are '*e*', '*f*' from which '*e*' has already been checked and adjacent of '*f*' are '*d*' and '*e*'. If '*e*' vertex visited then again we get dead state. So again we backtrack one step.

8. Now, adjacent to '*c*' is '*e*' and adjacent to '*e*' is '*f* ' and adjacent to '*f* ' is '*d*' and adjacent to '*d*' is '*a*'. Here, we get the Hamiltonian cycle as all the vertex other than the start vertex '*a*' is visited only once (*a* – *b* – *c* – *e* – *f* – *d* – *a*).





Again backtrack



a Solution

9. Here we have generated one Hamiltonian circuit but other Hamiltonian circuit can also be obtained by considering other vertex.

VERY IMPORTANT QUESTIONS

Following questions are very important. These questions may be asked in your SESSIONALS as well as UNIVERSITY EXAMINATION.

Q. 1. What do you mean by dynamic programming ?

Ans. Refer Q. 4.1.

Q. 2. Write down an algorithm to compute longest common subsequence (LCS) of two given strings.

Ans. Refer Q. 4.4.

Q. 3. Give the algorithm of dynamic 0/1-knapsack problem.

Ans. Refer Q. 4.5.

Q. 4. Describe the Warshall's and Floyd's algorithm for finding all pair shortest paths.

Ans. Refer Q. 4.9.

Q. 5. What is backtracking ? Write general iterative algorithm for backtracking.

Ans. Refer Q. 4.12.

Q. 6. Write short notes on N-Queens problem.

Ans. Refer Q. 4.17.

Q. 7. Explain travelling salesman problem using backtracking.

Ans. Refer Q. 4.13.

Q. 8. Explain TSP using branch and bound technique.

Ans. Refer Q. 4.14.

Q. 9. Write an algorithm for sum of subset problem.

Ans. Refer Q. 4.20.

Q. 10. Compare the various programming paradigms such as divide-and-conquer, dynamic programming and greedy approach.

Ans. Refer Q. 4.8.

☺☺☺