UNIT – III

DYNAMIC PROGRAMMING AND GREEDY TECHNIQUE

2 marks

1. **Define dynamic programming.**

Dynamic programming is an algorithm design method that can be used when a solution to the problem is viewed as the result of sequence of decisions.
Dynamic programming is a technique for solving problems with overlapping subproblems. These sub problems arise from a recurrence relating a solution to a given problem with solutions to its smaller sub problems only once and recording the results in a table from which the solution to the original problem is obtained. It was invented by a prominent U.S Mathematician, Richard Bellman in the 1950s.

2. **What are the features of dynamic programming?**
   - Optimal solutions to sub problems are retained so as to avoid recomputing their values.
   - Decision sequences containing subsequences that are sub optimal are not considered.
   - It definitely gives the optimal solution always.
3. **What are the drawbacks of dynamic programming?**
   - Time and space requirements are high, since storage is needed for all level.
   - Optimality should be checked at all levels.
4. **Write the general procedure of dynamic programming.**

The development of dynamic programming algorithm can be broken into a sequence of 4 steps.

- Characterize the structure of an optimal solution.
- Recursively define the value of the optimal solution.
- Compute the value of an optimal solution in the bottom-up fashion.
- Construct an optimal solution from the computed information.

5. **Define 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. **Write the difference between the Greedy method and Dynamic programming.**
   - Greedy method

     1. Only one sequence of decision is generated.
     2. It does not guarantee to give an optimal solution always.
   - Dynamic programming

     1. Many number of decisions are generated.
     2. It definitely gives an optimal solution always.
7. **What is greedy technique?**

Greedy technique suggests a greedy grab of the best alternative available in the hope that a sequence of locally optimal choices will yield a globally optimal solution to the entire problem. The choice must be made as follows

- *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.*
- *Irrevocable : Once made, it cannot be changed on a subsequent step of the algorithm*

8. **Write any two characteristics of Greedy Algorithm?**
   - To solve a problem in an optimal way construct the solution from given set of candidates.As the algorithm proceeds, two other sets get accumulated among this one set contains the candidates that have been already considered and chosen while the other set contains the candidates that have been considered but rejected.
9. **What is the Greedy choice property?**
   - The first component is greedy choice property (i.e.) a globally optimal solution can arrive at by making a locally optimal choice.
   - The choice made by greedy algorithm depends on choices made so far but it cannot depend on any future choices or on solution to the sub problem.
   - It progresses in top down fashion.

10. What is greedy method?

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one can device an algorithm that works in stages considering one input at a time.

11. What are the steps required to develop a greedy algorithm?
- Determine the optimal substructure of the problem.
- Develop a recursive solution.
- Prove that at any stage of recursion one of the optimal choices is greedy choice. Thus it is always safe to make greedy choice.
- Show that all but one of the sub problems induced by having made the greedy choice are empty.
- Develop a recursive algorithm and convert into iterative algorithm.

12. What is greedy technique?
- Greedy technique suggests a greedy grab of the best alternative available in the hope that a sequence of locally optimal choices will yield a globally optimal solution to the entire problem. The choice must be made as follows.
- 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.
- Irrevocable : Once made, it cannot be changed on a subsequent step of the algorithm

13. What are the labels in Prim's algorithm used for?

Prim's algorithm makes it necessary to provide each vertex not in the current tree with the information about the shortest edge connecting the vertex to a tree vertex. The information is provided by attaching two labels to a vertex.
- The name of the nearest tree vertex.
- The length of the corresponding edge

14. How are the vertices not in the tree split into?

The vertices that are not in the tree are split into two sets
- Fringe : It contains the vertices that are not in the tree but are adjacent to atleast one tree vertex.
- Unseen : All other vertices of the graph are called unseen because they are yet to be affected by the algorithm.

15. What are the operations to be done after identifying a vertex u* to be added to the tree?

After identifying a vertex u* to be added to the tree, the following two operations need to be performed
- Move u* from the set $V\text{-}V_T$ to the set of tree vertices V_T.
- For each remaining vertex u in $V\text{-}V_T$ that is connected to u* by a shorter edge than the u's current distance label, update its labels by u* and the weight of the edge between u* and u, respectively.

16. What is the use of Dijksra's algorithm?

Dijkstra's algorithm is used to solve the single-source shortest-paths problem: for a given vertex called the source in a weighted connected graph, find the shortest path to all its other vertices. The single-source shortest-paths problem asks for a family of paths, each leading from the source to a different vertex in the graph, though some paths may have edges in common.

17. Define Spanning tree.

Spanning tree of a connected graph *G*: a connected acyclic subgraph of *G* that includes all of *G*'s vertices

18. What is minimum spanning tree.

Minimum spanning tree of a weighted, connected graph *G*: a spanning tree of *G* of the minimum total weight

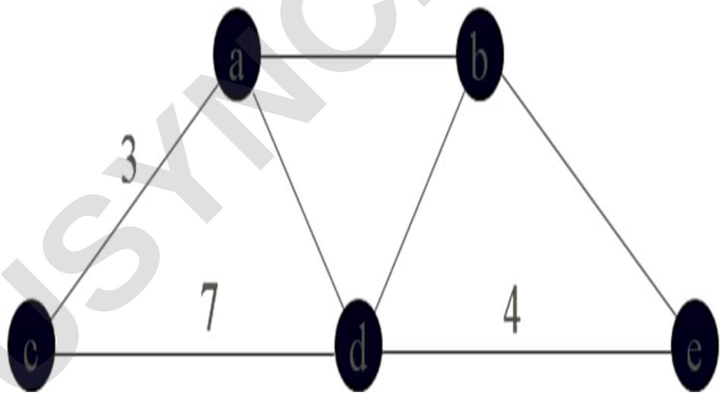
16 marks

1. **Write Short note on Dijkstra's Algorithm**

Shortest Paths – Dijkstra's Algorithm

Shortest Path Problems

- All pair shortest paths (Floy's algorithm)
- Single Source Shortest Paths Problem (Dijkstra's algorithm): Given a weighted graph G, find the shortest paths from a source vertex s to each of the other vertices.



- **Prim's and Dijkstra's Algorithms**
  - Generate different kinds of spanning trees
    - Prim's: a minimum spanning tree.
      - Dijkstra's : a spanning tree rooted at a given source s, such that the distance from s to every other vertex is the shortest.
    - Different greedy strategies
      - Prims': Always choose the closest (to the tree) vertex in the priority queue Q to add to the expanding tree V_T.
      - Dijkstra's : Always choose the closest (to the source) vertex in the priority queue Q to add to the expanding tree V_T.
    - Different labels for each vertex
      - Prims': parent vertex and the distance from the tree to the vertex.
      - Dijkstra's : parent vertex and the distance from the source to the vertex.

2. **Explain Kruskal's Algorithm**

✓ Greedy Algorithm for MST: Kruskal
  - Edges are initially sorted by increasing weight

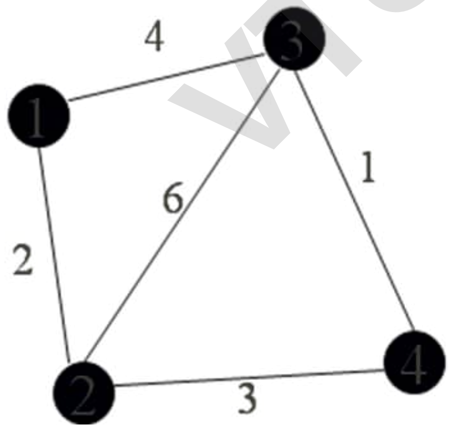
- Start with an empty forest
- "grow" MST one edge at a time
  - intermediate stages usually have forest of trees (not connected)
- at each stage add minimum weight edge among those not yet used that does not create a cycle
  - at each stage the edge may:
  - expand an existing tree
  - combine two existing trees into a single tree
  - create a new tree
- need efficient way of detecting/avoiding cycles

✓ algorithm stops when all vertices are included

```
ALGORITHM Kruscal(G)
//Input: A weighted connected graph $G = <V, E>$
//Output: $E_T$, the set of edges composing a minimum spanning tree of G.
Sort E in nondecreasing order of the edge weights
    $w(e_{i1}) <= \ldots <= w(e_{i|E|})$
$E_T \leftarrow \varnothing$; ecounter $\leftarrow 0$        //initialize the set of tree edges and its size
$k \leftarrow 0$
while encounter < $|V| - 1$ do
        $k \leftarrow k + 1$
        if $E_T \cup \{e_{ik}\}$ is acyclic
                $E_T \leftarrow E_T \cup \{e_{ik}\}$ ; ecounter $\leftarrow$ ecounter + 1
return $E_T$
```

3. Discuss Prim's Algorithm

✓ Minimum Spanning Tree (MST)
- Spanning tree of a connected graph G: a connected acyclic subgraph (tree) of G that includes all of G's vertices.
- Minimum Spanning Tree of a weighted, connected graph G: a spanning tree of G of minimum total weight.
- Example:



✓ Prim's MST algorithm

✓ Start with a tree , T_0 ,consisting of one vertex

✓ "Grow" tree one vertex/edge at a time
- Construct a series of expanding subtrees T_1, T_2, … T_{n-1}. .At each stage construct T_{i+1} from T_i by

        - adding the minimum weight edge connecting a vertex in tree (T_i) to one not yet in tree
            - choose from “fringe” edges

- ✓ (this is the “greedy” step!) Or (another way to understand it)
- ✓ expanding each tree (Ti) in a greedy manner by attaching to it the nearest vertex not in that tree. (a vertex not in the tree connected to a vertex in the tree by an edge of the smallest weight)
- ✓ Algorithm stops when all vertices are included
- ✓ **Algorithm**:

```
ALGORITHM Prim(G)
//Prim’s algorithm for constructing a minimum spanning tree
//Input A weighted connected graph G= V, E
//Output $E_T$, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow$ {v0}
$E_T \leftarrow$ F
for i $\leftarrow$ 1 to |V|-1 do
Find the minimum-weight edge e*=(v*,u*) among all the edges (v,u) such that v is in $V_T$ and u is in V-$V_T$
$V_T \leftarrow V_T$ U {u*}
$E_T \leftarrow E_T$ U {e*}
return $E_T$
```

4. Write short note on Greedy Method

- ✓ A greedy algorithm makes a locally optimal choice in the hope that this choice will lead to a globally optimal solution.
    - The choice made at each step must be:
    - Feasible
        - Satisfy the problem’s constraints
    - locally optimal
        - Be the best local choice among all feasible choices
    - Irrevocable
        - Once made, the choice can’t be changed on subsequent steps.
- ✓ Applications of the Greedy Strategy
    - Optimal solutions:
        - change making
        - Minimum Spanning Tree (MST)
        - Single-source shortest paths
        - Huffman codes
    - Approximations:
        - Traveling Salesman Problem (TSP)
        - Knapsack problem
        - other optimization problems

5. What does dynamic programming has in common with divide-and-Conquer?

- ✓ **Dynamic Programming**
- Dynamic Programming is a general algorithm design technique. “Programming” here means “planning”.
- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems
- Main idea:
    a. solve several smaller (overlapping) subproblems
    b. record solutions in a table so that each subproblem is only solved once

c. final state of the table will be (or contain) solution

- Dynamic programming vs. divide-and-conquer
  a. partition a problem into overlapping subproblems and independent ones
  b. store and not store solutions to subproblems

✓ **Example: Fibonacci numbers**

Recall definition of Fibonacci numbers:

$f(0) = 0$

$f(1) = 1$

$f(n) = f(n-1) + f(n-2)$

- Computing the n^{th} Fibonacci number recursively (top-down):

$$\begin{tikzcd}
 & & & f(n) \arrow[dll] \arrow[drr] & & & \\
 & f(n-1) \arrow[dl] \arrow[dr] & & + & & f(n-2) \arrow[dl] \arrow[dr] & \\
f(n-2) & + & f(n-3) & & f(n-3) & + & f(n-4)
\end{tikzcd}$$

...

Computing the n^{th} fibonacci number using bottom-up iteration:

$f(0) = 0$
$f(1) = 1$
$f(2) = 0+1 = 1$
$f(3) = 1+1 = 2$
$f(4) = 1+2 = 3$
$f(5) = 2+3 = 5$
$f(n-2) =$
$f(n-1) =$
$f(n) = f(n-1) + f(n-2)$

```
ALGORITHM Fib(n)
F[0] ← 0, F[1] ← 1
for i←2 to n do
        F[i] ← F[i-1] + F[i-2]
return F[n]
```

6. Disuss Warshall's Algorithm with suitable diagrams?

- Main idea: Use a bottom-up method to construct the transitive closure of a given digraph with *n* vertices through a series of *nxn* boolean matrices:

$R^{(0)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}$

$R^{(k)}$: $r_{ij}^{(k)} = 1$ in $R^{(k)}$, iff

there is an edge from i to j; or
there is a path from i to j going through vertex 1; or
there is a path from i to j going through vertex 1 and/or 2; or
...
there is a path from i to j going through 1, 2, … and/or k

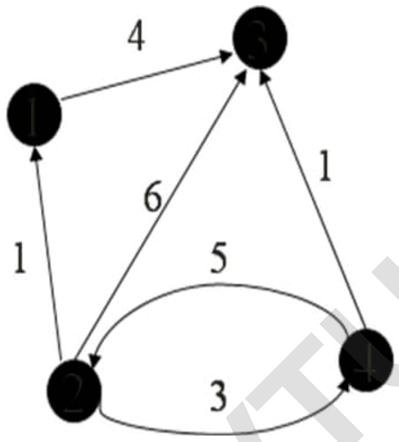
	intermediate node	

In the kth stage: to determine $R^{(k)}$ is to determine if a path exists between two vertices i, j using just vertices among 1,…,k

$$r_{ij}^{(k)} = 1: \begin{cases} r_{ij}^{(k-1)} = 1 & \text{(path using just } 1,\ldots,k\text{-1)} \\ \text{or} & \\ (r_{ik}^{(k-1)} = 1 \text{ and } r_{kj}^{(k-1)}) = 1 & \text{(path from } i \text{ to } k \\ & \text{and from } k \text{ to } i \\ & \text{using just } 1,\ldots,k\text{-1)} \end{cases}$$

7. Explain how to Floyd's Algorithm works.

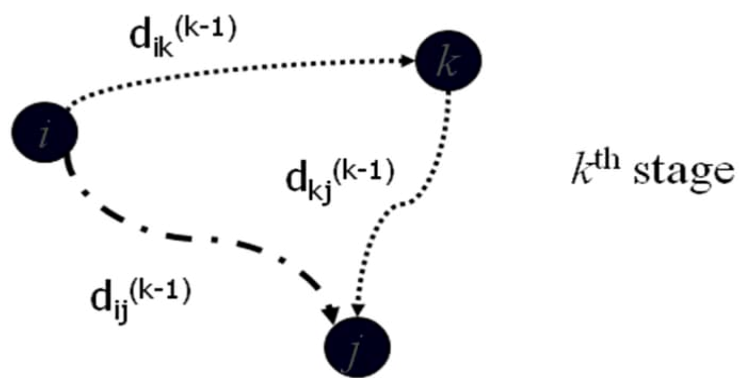
- All pairs shortest paths problem: In a weighted graph, find shortest paths between every pair of vertices.
- Applicable to: undirected and directed weighted graphs; no negative weight.
- Same idea as the Warshall's algorithm : construct solution through series of matrices D(0) , D(1), …, D(n)



$$\begin{matrix} 0 & \infty & 4 & \infty \\ 1 & 0 & 6 & 3 \\ \infty & \infty & 0 & \infty \\ \infty & 5 & 1 & 0 \end{matrix} \qquad\qquad \begin{matrix} 0 & \infty & 4 & \infty \\ 1 & 0 & 5 & 3 \\ \infty & \infty & 0 & \infty \\ 6 & 5 & 1 & 0 \end{matrix}$$

Weight matrix $\longrightarrow$ **distance matrix**

- D(k) : allow 1, 2, …, k to be intermediate vertices.
- In the kth stage, determine whether the introduction of k as a new eligible intermediate vertex will bring about a shorter path from i to j.
- dij(k) = min{dij(k-1) , dik(k-1) + dkj(k-1} for k ≥ 1, dij(0) = wij



8. Explain Knapsack problem

✓ **The problem**

- Find the most valuable subset of the given n items that fit into a knapsack of capacity W.

✓ **Consider the following sub problem P(i, j)**

- Find the most valuable subset of the first i items that fit into a knapsack of capacity j, where $1 \le i \le n$, and $1 \le j \le W$
- Let V[i, j] be the value of an optimal solution to the above subproblem P(i, j). Goal: V[n, W]

9. Explain Memory Function algorithm for the Knapsack problem

✓ The Knapsack Problem and Memory Functions

- The Recurrence
  a. Two possibilities for the most valuable subset for the subproblem P(i, j)
     i. It does not include the ith item: $V[i, j] = V[i-1, j]$
     ii. It includes the ith item: $V[i, j] = v_i + V[i-1, j - w_i]$

$$V[i, j] = \begin{cases} \max\{V[i-1, j], v_i + V[i-1, j - w_i]\}, & \text{if } j - w_i \ge 0 \\ V[i-1, j] & \text{if } j - w_i < 0 \end{cases}$$

$$V[0, j] = 0 \text{ for } j \ge 0 \text{ and } V[i, 0] = 0 \text{ for } i \ge 0$$

✓ Memory functions:

- Memory functions: a combination of the top-down and bottom-up method. The idea is to solve the subproblems that are necessary and do it only once.
- Top-down: solve common subproblems more than once.
- Bottom-up: Solve subproblems whose solution are not necessary for the solving the original problem.

✓ ALGORITHM MFKnapsack(i, j)

```
if V[i, j] < 0 //if subproblem P(i, j) hasn't been solved yet.
        if j < Weights[i]
                value ← MFKnapsack(i – 1, j)
        else
                value ←max(MFKnapsack(i – 1, j),
                                values[I] + MFKnapsck( i – 1, j – Weights[i]))
        V[i, j] ← value
return V[i, j]
```

10. **Explain in detail about Huffman tree.**

Any binary tree with edges labeled with 0's and 1's yields a prefix-free code of characters assigned to its leaves. Optimal binary tree minimizing the average length of a codeword can be constructed as follows:

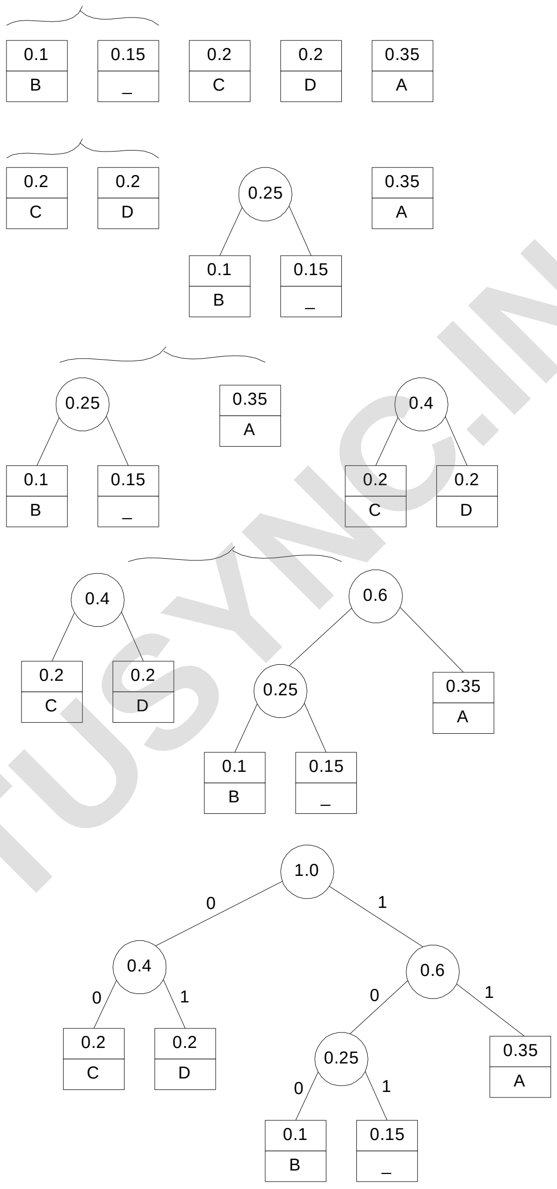
Huffman's algorithm

- Initialize *n* one-node trees with alphabet characters and the tree weights with their frequencies.

- Repeat the following step *n*-1 times: join two binary trees with smallest weights into one (as left and right subtrees) and make its weight equal the sum of the weights of the two trees.
- Mark edges leading to left and right subtrees with 0's and 1's, respectively.

Example:

character	**A**	**B**	**C**	**D**	**_**
codeword	**0.35**	**0.1**	**0.2**	**0.2**	**0.15**


0.1
B
0.15
_
0.2
C
0.2
D
0.35
A
0.25
0.4
0.6
1.0
0
1