# Algorithms

Final term report writting

Name : Jannatul Ferdous Umama

ID : 20-42616-1

Sec : P

Jannatul Ferdous Umama
ID: 20-42616-1

# Final Term Report Writing

**Dynamic algorithm:** Dynamic Programming (DP) is an algorithmic technique for solving an optimization problem by breaking it down into simpler subproblems and utilizing the fact that the optimal solution to the overall problem depends upon the optimal solution to its subproblems.

**Development of Dynamic Programming Algorithm:**

It can be broken into four steps:

1. Characterize the structure of an optimal solution.
2. Recursively defined the value of the optimal parts recursively. This helps to determine what the solution will look like.
3. Compute the value of the optimal solution from the bottom up (starting with the smallest subproblems)
4. Construct the optimal solution for the entire problem form the computed values of smaller subproblems.

Jannatul Ferdous Umama
ID: 20-42616-1

Applications of dynamic programming:

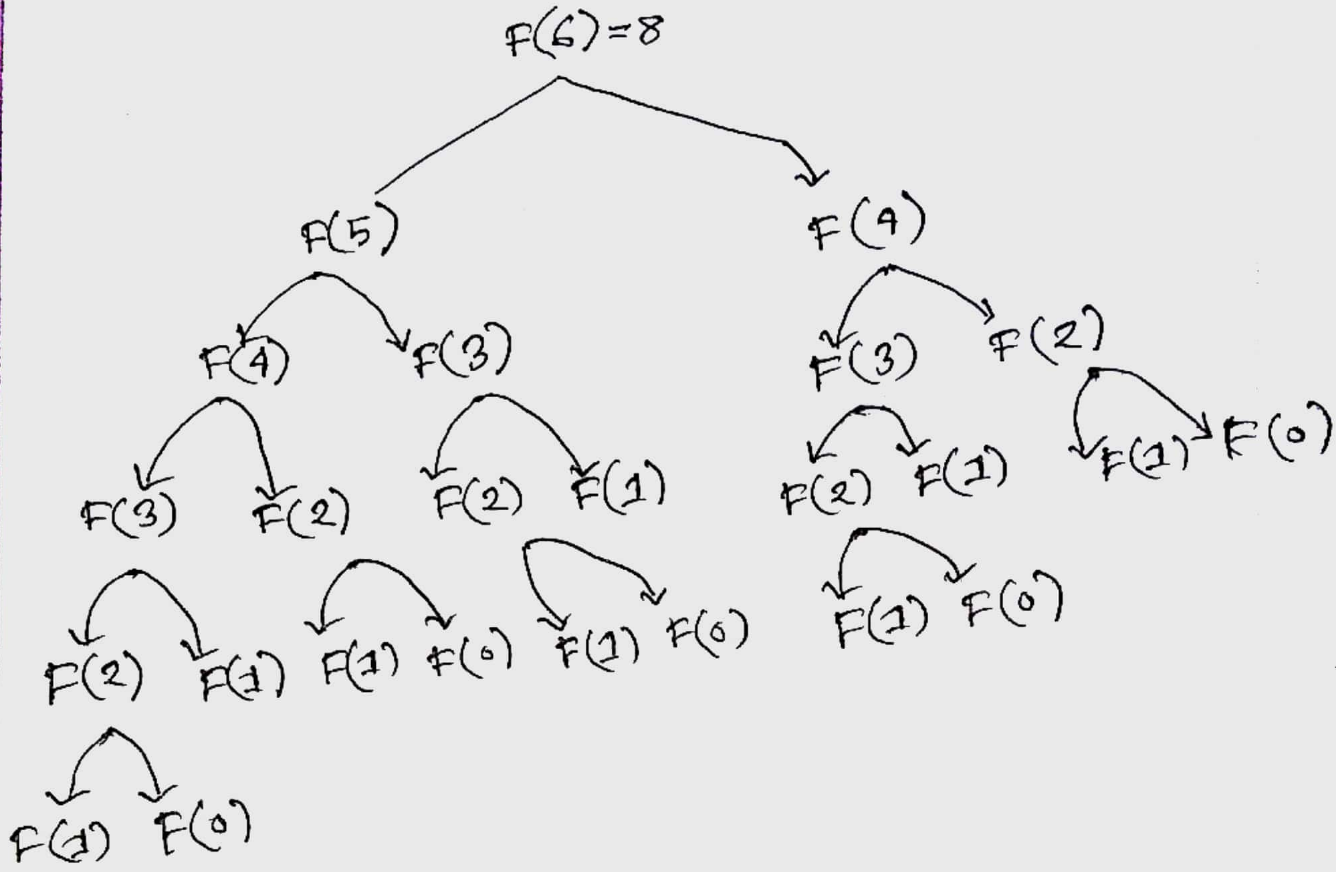
1. 0/1 knapsack problem
2. Mathematical optimization problem.
3. All pair Shortest path problem.
4. Reliability design problem.
5. Longest common subsequence (LCS)
6. Flight control and robotics control
7. Time-sharing: It schedules the job to maximize CPU usage.

Example:
The following computer problems can be solved using dynamic programming approach-

- Fibonacci number series
- Knapsack problem.
- Tower of Hanoi
- All pair shortest path by Floyd-Warshall
- Shortest path by Dijkstra
- Project scheduling.

Jannatul Ferdous Umama
ID : 20-42616-1

Example : Fibonacci Numbers :

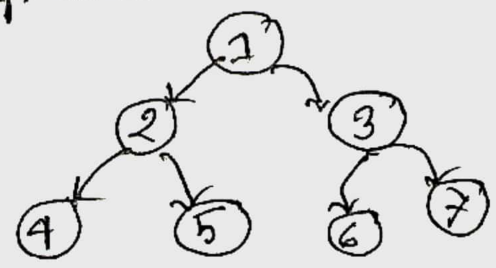



## Trees and graphs :

Tree : A tree is a finite set of one or more nodes such that -

1. There is a specially designated node called root.
2. The remaining nodes are partitioned into $n>=0$ disjoint sets $T_1, T_2, .. T_n$. where $T_1, T_2, T_3 .. T_n$ is called the subtrees of the root.

The concept of tree is represented :-

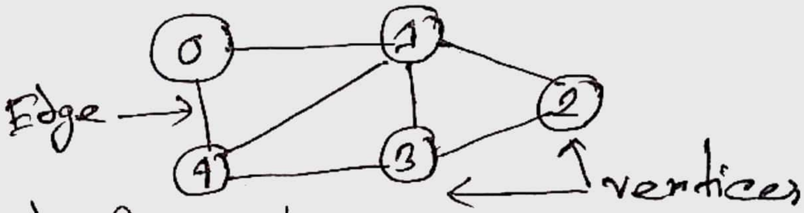


Jannatul Ferdous Umama
ID : 20-42616-1

Graph : A graph is collection of two sets V and E where V is a finite non-empty set of vertices and E is a finite non-empty set of edges.

1. Vertices are nothing but the nodes in the graph
2. Two adjacent vertices are joined by edges.
3. Any graph is denoted as $G = \{V, E\}$.

Example :




Kruskal algorithm and Prim's algorithm :

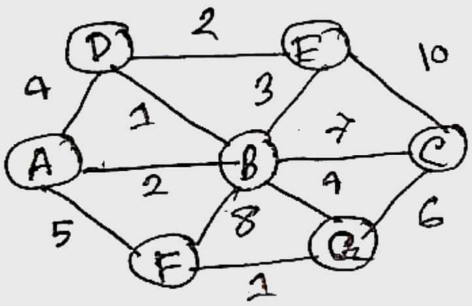
Kruskal's algorithm for MST : Given a connected and
undirected graph, a spanning tree of that graph is a
subgraph that is a tree and connects all the vertices
together. A single graph can have many different
spanning trees. A minimum spanning tree (MST) or
minimum weight spanning tree for a weighted, connected
and undirected graph is a spanning tree with weight
less than or equal to the weight of every other
spanning tree. The weight of a spanning tree is
the sum of weights given to each edge of the
spanning tree.

Jannatul Ferdous Umama
ID : 20-42616-1

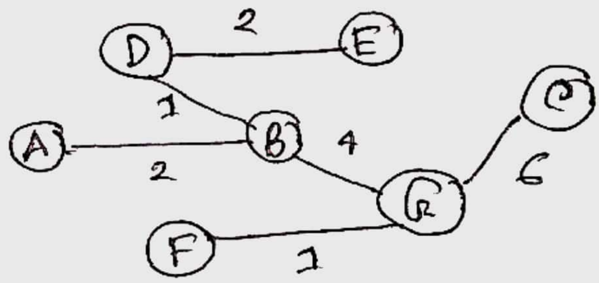
Below are the steps for finding MST using kruskal's algorithm :

1. ⇒ Sort all the edges in non-decreasing order of their weight.
2. ⇒ Pick the smallest edge. Check if it forms a cycle with the spanning-tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. ⇒ Repeat step #2 until there are (V-1) edges in the spanning tree.

Graph




Minimum Spanning Tree using kruskal's Algo




Weight = $2+1+2+4+1+6 = 16$

Prim's algorithm for MST :

Like kruskal's algorithm, Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Jannaful Ferdous Umama
ID: 20-42616-1

Below are the steps for finding MST using Prim's algorithm:

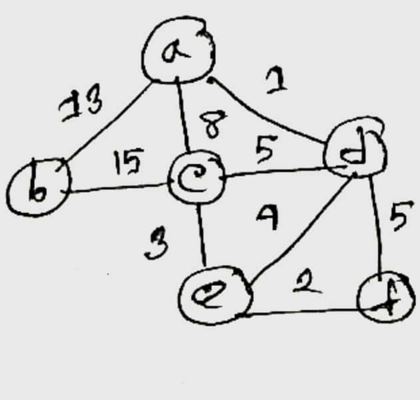
1 ⇒ Create a set mstSet that keeps track of vertices already included in MST.

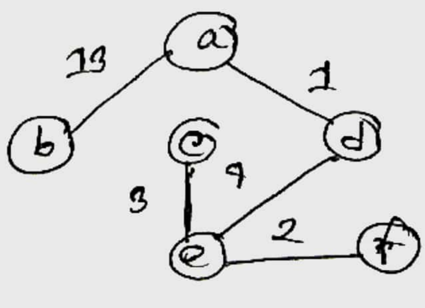
2 ⇒ Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3 ⇒ While mstSet doesn't include all vertices.

- Pick a vertex u which is not there in mstSet and has minimum key value.
- Include u to mstSet.
- Update the key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if the weight of edge u-v is less than the previous key value of v, update the key value as the weight of u-v.

Animated Example for Prim's Algorithm.



Jannatul Ferdaus Umama
ID : 20-42616-1

Matrix Chain Multiplication : 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 : $(ABC)D = (AB)(CD) = A(BCD) = \ldots$

However, the order in which we parenthsize 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,

$(AB)C = (10 \times 30 \times 5) + (10 \times 5 \times 60) = 1500 + 3000 = 4500$ operations

$A(BC) = (30 \times 5 \times 60) + (10 \times 30 \times 60) = 9000 + 18000$
$= 27000$ operations.

Clearly the first parameterization requires less number of operations. Given an array p[] which represents the chain of matrices such that the ith matrix $A_i$ is of dimension $p[i-1] \times p[i]$. We need to write a function MatrixChainOrder() that should return the minimum number of multiplications needed to multiply the chain.

Scanned with CamScanner