Only some part of presentations will be shared on www.mbchandak.com

Email: chandakmb@gmail.com hodcs@rknec.edu

Unit 3 (Part-I): Greedy Algorithms

Expected Outcomes of topic (CO)

- Understand problem and its formulation to design an algorithm
- Understand various approaches for algorithm design and select an approach with objective of efficient utilization of resources.
- Demonstrate the knowledge of basic data structures and their implementation and decide upon use of particular data structure best suited for implementing solution.
- Write efficient algorithms using Greedy, Divide & Conquer, Dynamic programming, and Backtracking methods to solve the real life problems.
- Trace logic formulation, execution path of particular algorithm and data generated during execution of algorithm.
- Apply various standard algorithms to solve variety of real world problems.

Basic Characteristics

- Solution is built in small steps
- Decisions on how to build the solution are made to maximize some criterion without looking to the future
  - Want the 'best' current partial solution as if the current step were the last step

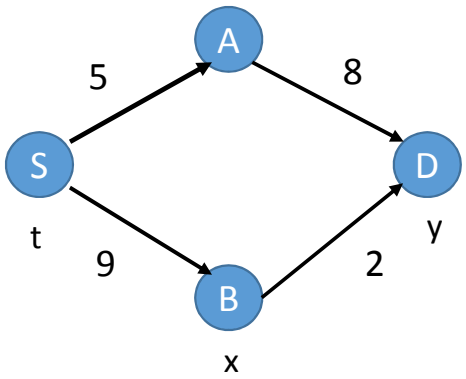
Greedy is a strategy that works well on optimization problems with the following characteristics:

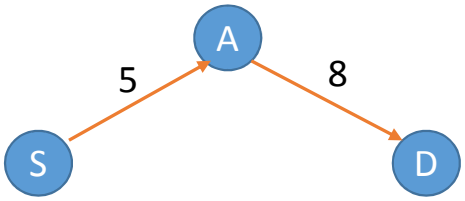
1. **Greedy-choice property:** A global optimum can be arrived at by selecting a local optimum.
2. **Optimal substructure:** An optimal solution to the problem contains an optimal solution to subproblems.

- Greedy algorithms
  - Easy to produce
  - Fast running times
  - Work only on certain classes of problems

Greedy Algorithms

- Characteristics
  - Greedy Algorithms are short – sighted.
  - Greedy Algorithms are most efficient if it works.
  - For every instance of input Greedy Algorithms makes a decision and continues to process further set of input.
  - The other input values at the instance of decision are lost and not used in further processing.

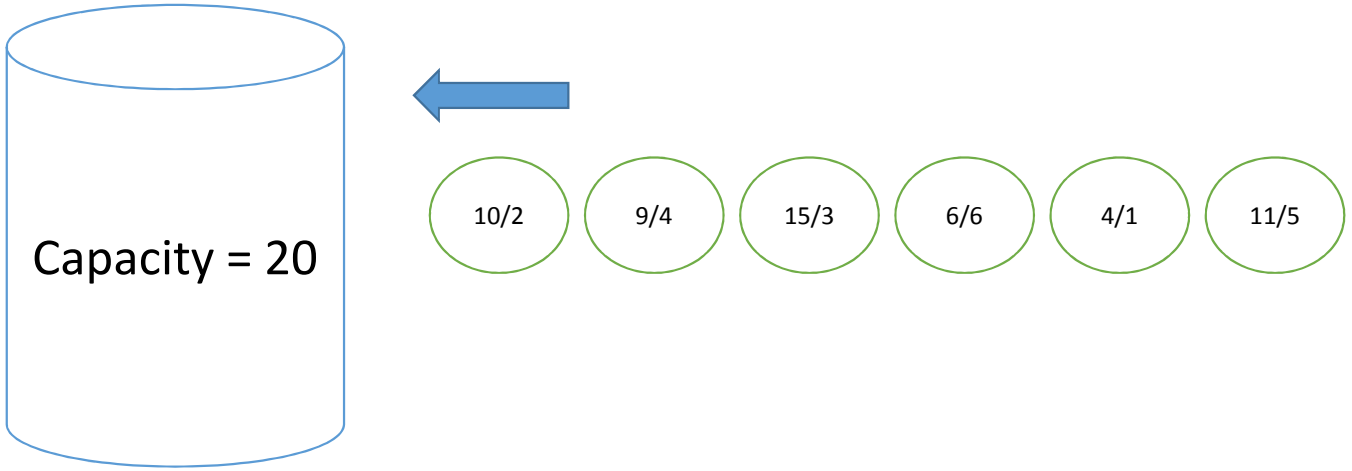




Greedy Algorithm: Knapsack Problem

- Principle: **Knapsack is a bag of fixed capacity.**
- **In this problem it is assumed that "n" objects with profit value and weight are given.**
- **The main objective is to select the objects and place them in knapsack such that the object placed in the bag will generate maximum profit.**
- Two types of Knapsack problems:
  - Fractional Knapsack problem
  - 0/1 Knapsack problem
- The Fractional Knapsack problem can be solved using Greedy approach, where as 0/1 Knapsack problem does not have greedy solution.

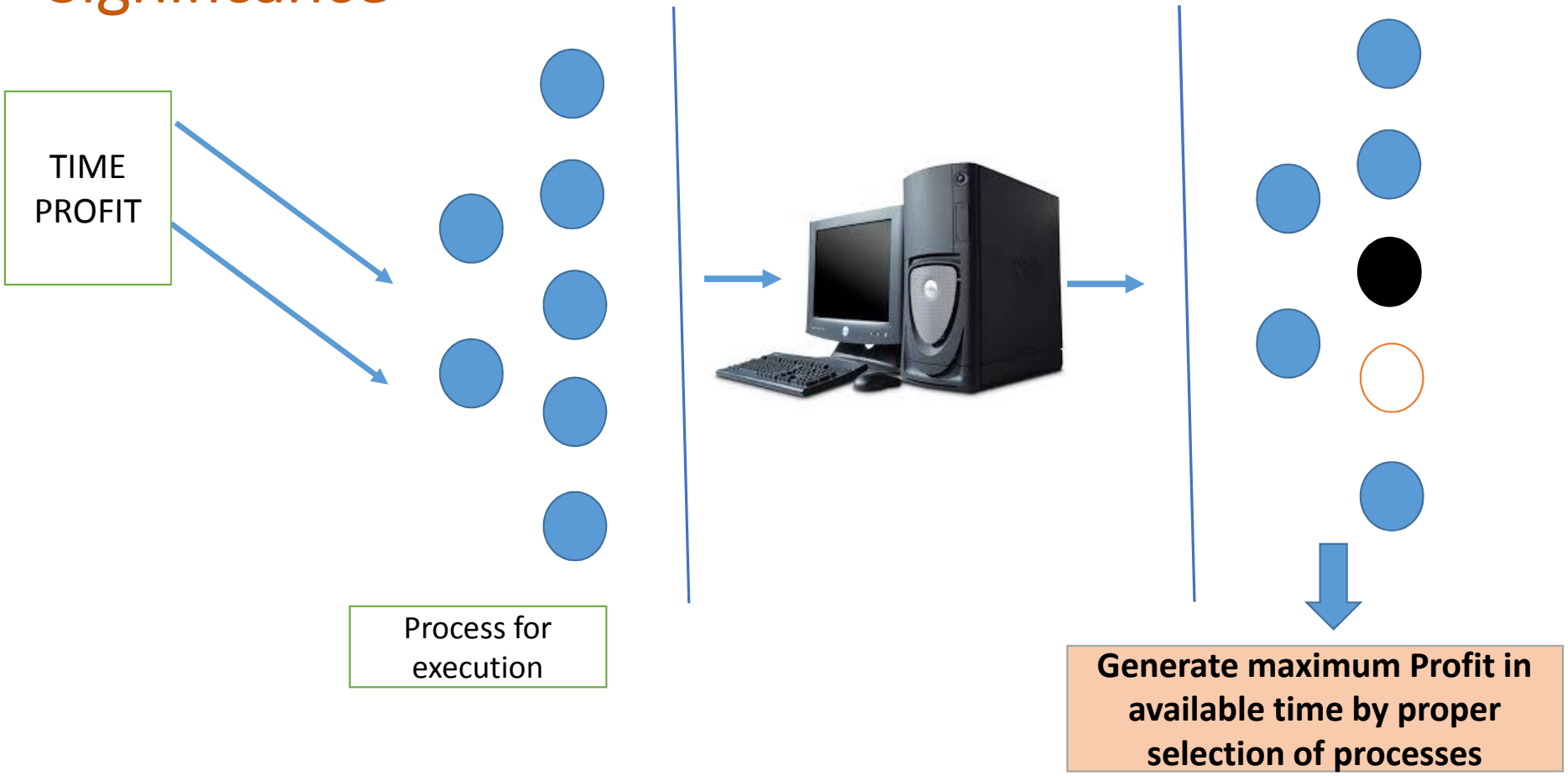
Representation



Greedy Algorithm: Knapsack Problem

- **Approaches:**
- There are three basic approaches to solve knapsack problem:
  - Select the maximum profit objects
  - Select the minimum weight objects
  - Select the objects based on Profit/Weight ratio

Significance



Knapsack Problem: Algorithm

- Data Structures:
  - Weight array: W[n] ➔ To store weight values
  - Profit array: P[n] ➔ To store profit values
  - Capacity ➔ Given and will decrease after each step
  - X[n] ➔ To store 0, 1 or fraction depending upon object placement
  - Weight = 0 (initially) and will increase towards capacity
  - Profit = 0 and will be calculated by X[i] and P[i]
- Process: To select the object with best Pi/Wi Ratio. (Sorting)

```
Algorithm Knapsack (p, w, n, Capacity : profit,x)
{Step 1
    profit = 0;
    for i = 1 to n do
        x[i] = 0;
   weight = 0;
    While (weight < capacity) do
    {
      Select an object with best pi/wi ratio and let the index of
object be "i"
       if (weight + w[i] <= capacity) then
       {
            x[i] = 1;
            weight = weight + w[i];
        }
      else
       {
       x[i] = ( capacity – weight )/w[i];
          weight = capacity;
       }
   } //End of while loop
STEP 2   for  i  = 1 to n do
   profit = profit + (p[i] * x[i])
} //End of algorithm
```

Greedy Algorithms on Graph

Prim's Algorithm

Kruskal Algorithm

Single Source shortest Path Algorithm (Dijkstra's Algorithm)

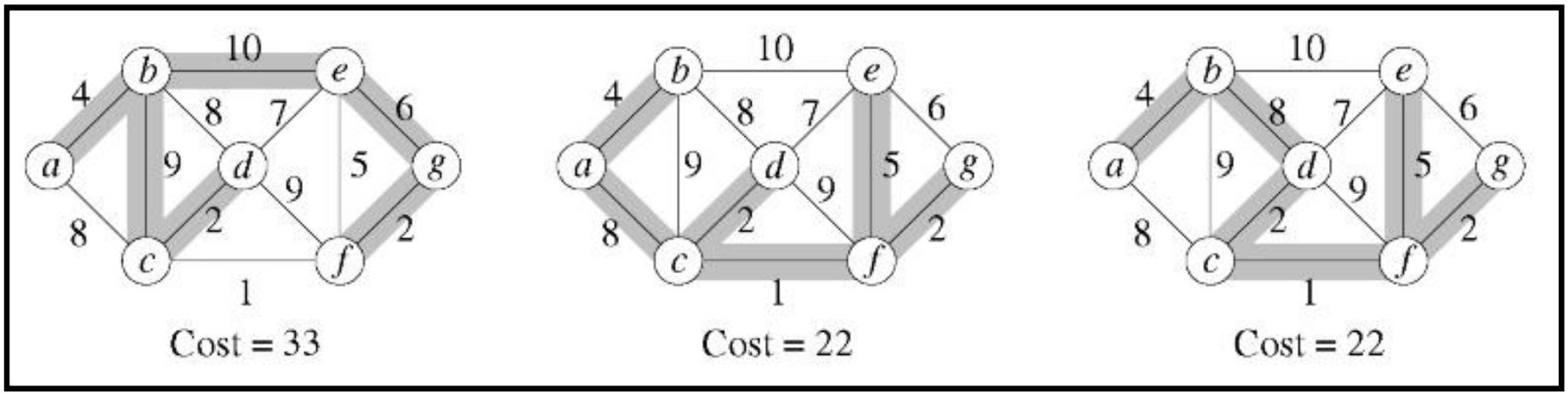
Minimum Cost Spanning Tree

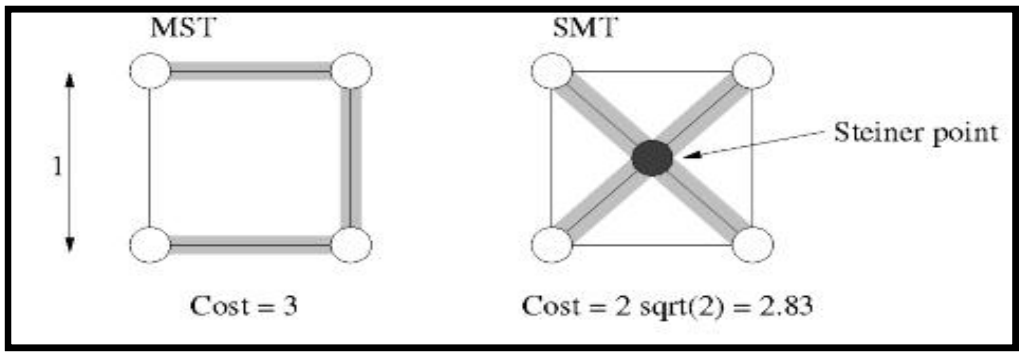
- Given a graph G =(V,E) V=Number of vertices and E=Number of edges, then Spanning tree is tree generated from graph, with characteristics
  - All the vertices present in the tree
  - There is no cycle in the tree i.e., e=v-1
- Subset of edges, that forms a tree, where total cost of edges is minimum.
- *Minimum cost spanning tree is a spanning tree with "edges of minimum cost".*
- **Applications:**
- To transmit the message without broadcasting.
- To generate travel plan in minimum cost.
- Designing network, home electric wiring etc.
- Approaches: Prims' Method and Kruskals' Method.

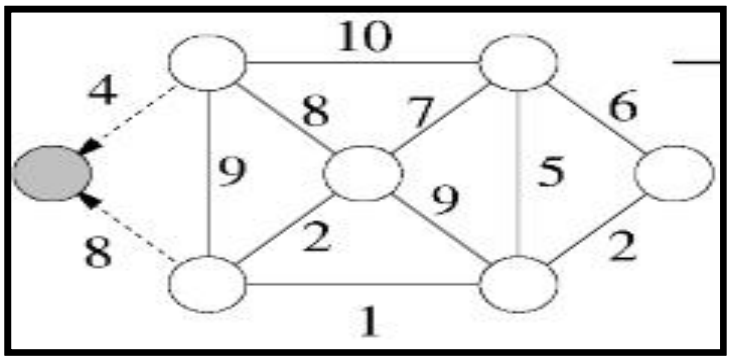
Minimum Cost Spanning Tree

- Principle:
  - Select an edge of minimum cost. The edge will derive two vertices.
  - Continue the process from one of the vertex by selecting the next edge of minimum cost.
  - Once the vertex is visited, then it is marked.
  - The process will attempt to visit unmarked vertices and terminates when all the vertices are visited.
  - Process guarantees no cycle in the tree.
- *"What if the process of tree generation starts from any arbitrary vertex".*
- **Motivation: To join points as cheaply as possible: Applications in clustering and networking**
- **Acyclic graph to connect all nodes in minimum cost.**
- **One of the term in U.S. legal code (AT&T)**

Graphs for Examples







Spanning tree: Free tree

- A free tree has following properties

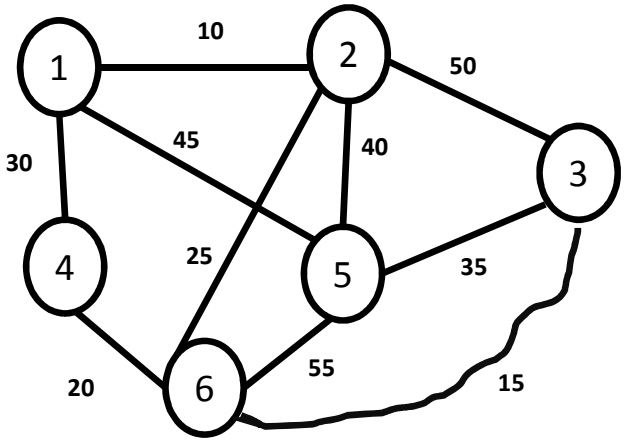
1. Exactly n-1 edges for “n” vertices
2. There exists a unique path between two vertices.
3. By adding an edge, a cycle will be created in free tree. Breaking any edge on the cycle restores the free tree.

- *Greedy: Minimization problem.*

1. Repeated selection: of minimum cost edges with certain test cases.
2. Once decision of adding edge is finalized, it cannot be revoked.
3. Basic idea: To generate subset of “E” connecting all “V”

Minimum Cost Spanning Tree: Prim's method

- Example:
- Consider the following graph:



	1	2	3	4	5	6
1	99	10	99	30	45	99
2	10	99	50	99	40	25
3	99	50	99	99	35	15
4	30	99	99	99	99	20
5	45	40	35	99	99	55
6	99	25	15	20	55	99

- ***Select an edge of minimum cost.***
- ***From selected vertex continue selecting edge of minimum cost***