Computer Algorithms

Unit-2

The Greedy method

The Greedy method

- Straight forward design technique
- It can be applied to wide variety of problems
- Most of such problems have n inputs and required to obtain a subset that satisfies some constraints
- Any subset that satisfies these constraints is called a feasible solution
- Find feasible solution that either maximizes or minimizes a given objective function.
- The feasible solution that do this, is called optimal solution

Subset Paradigm

- Algorithm works in stages, considering one input at a time
- At each stage decision is made regarding, whether a particular input is in an optimal solution
- This is done by considering the inputs in an order determined by some selection procedure.
- If inclusion of the next input into the partially constructed optimal solution will result in an infeasible solution,
- then this input is not added to the partial solution, Otherwise it is added
- Selection procedure is based on objective function

Ordering Paradigm

- Do not call for the selection of an optimal subset
- Make decisions by considering the inputs in some order
- Each decision is made using an optimization criterion that can be computed using decisions already made

Knapsack Problem

- Given n items and a knapsack (or a bag) of capacity m.
- Associated with each item (i) is the weight (w_i) and the profit earned.
- If the fraction of item x_i is placed in knapsack
- then profit of $p_i x_i$ is earned. $0 <= x_i <= 1$

Knapsack Problem

- Objective is to obtain a filling of the knapsack that maximizes the total profit.
- Formally the problem can be stated as
- Maximize

$$\sum_{1 \leq i \leq n} p_i x_i$$

- Subject to

$$\sum_{1 \leq i \leq n} w_i x_i \leq m$$

- And $0 \leq x_i \leq 1 \qquad 1 \leq i \leq n$

Greedy Approaches

- Sort items based on profit in decreasing order place items till knapsack is full.
- Sort items based on weight in increasing order place items till knapsack is full.
- Sort items based on profit/weight decreasing, place items till knapsack is full.

Example

- Example 1
- n=3 m=20
- $(p_1, p_2, p_3) = (25,24,15)$
- $(w_1, w_2, w_3) = (18, 15, 10)$

- Example 2
  - n=7 m=15
  - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10,5,15,7,6,18,3)$
  - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2,3,5,7,1,4,1)$

- (0,1,1/2) 20 31.5

Example

- Example 3
- n=4 m=25
- $(p_1, p_2, p_3, p_4) = (2, 5, 8, 1)$
- $(w_1, w_2, w_3, w_4) = (10, 15, 6, 9)$

- Example 4
  - n=6 c=20
  - $(p_1, p_2, p_3, p_4, p_5, p_6) = (12,5,15,7,6,18)$
  - $(w_1, w_2, w_3, w_4, w_5, w_6) = (2,3,5,7,1,5)$

Example

- Example 5
- n=4 m=30
- $(p_1, p_2, p_3, p_4) = (27, 20, 24, 15)$
- $(w_1, w_2, w_3, w_4) = (15, 10, 18, 10)$

Job Sequencing with Deadlines

- Given a set of n jobs

– Each job is associated with an integer deadline $d_i \geq 0$ and profit $p_i \geq 0$

– For any job profit p_i will be earned iff the job is completed by its deadline

– Each job takes unit time

– Only one machine (server) is available.

Job Sequencing with Deadlines

– Feasible solution:

subset J of jobs such that

each job in J is completed within its deadline

– Optimal solution:

Feasible solution with highest profit

Solution

- Greedy Method
  - Sorting jobs based on profit ↓
  - Sorting jobs based on deadlines ↑
- Implementation
  - Using array

Example

12	9	10	5	15	2	20
6	1	4	6	1	3	3

20	15	12	10	9	5	2
3	1	6	4	1	6	3

Array ⇨

1	2	3	4	5	6	7

Example

- $n=4$, $(p_1, p_2, p_3, p_4) = (100,10,15,27)$
  $(d_1, d_2, d_3 , d_4) = (2,1,2,1)$

Feasible Solution	Processing Sequence	Value
• (1,2)	2,1	110
• (1,3)	1,3 or 3,1	115
• (1,4)	4,1	127
• (2,3)	2,3	25
• (3,4)	4,3	42
• (1)	1	100
• (2)	2	10
• (3)	3	15
• (4)	4	27

Example

- Example 1 $n=5$,
- $(p_1, p_2, p_3, p_4 , p_5) = (20,15,10,5,1)$
- $(d_1, d_2, d_3 , d_4 , d_5) = (2,2,1,3,3)$

- Example 2 $n=7$
- $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (3,5,20,18,1,6,30)$
- $(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1,3,4,3,2,1,2)$

Example

- Example 3 n=5 ,
- $(p_1, p_2, p_3, p_4 , p_5) = (45,15,20,7,65)$
- $(d_1, d_2, d_3 , d_4 , d_5) = (1,3,2,1,2)$

- Example 4 n=7
- $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (50,15,18,16,8,25,60)$
- $(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1,3,4,3,2,1,2)$

Example

- Example 5 n=5 ,
- $(p_1, p_2, p_3, p_4 , p_5) = (20,16,11,5,25)$
- $(d_1, d_2, d_3 , d_4 , d_5) = (2,2,1,2,1)$

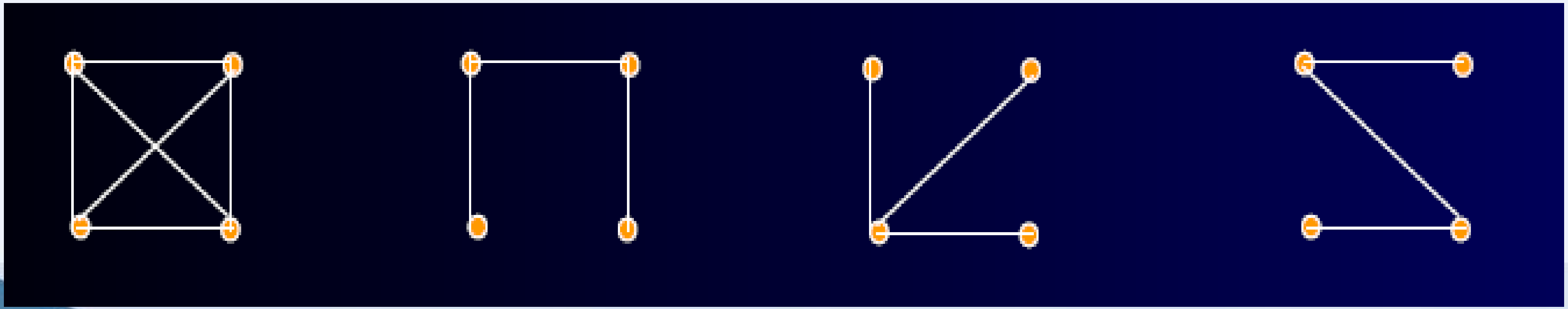
- Example 6 n=7
- $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (45,5,20,18,6,30,70)$
- $(d_1, d_2, d_3, d_4, d_5, d_6, d_7) = (1,3,4,3,2,1,2)$

Job Sequencing with Deadlines

- Variants
  - Processing time is different
  - Multiple servers
  - Profit factors different if executed on different server

Minimum Cost Spanning Tree

- Spanning Tree-
- Let G=(V,E) be undirected connected graph.
- A sub-graph t=(V,E’) of G is spanning tree
- iff t is tree.



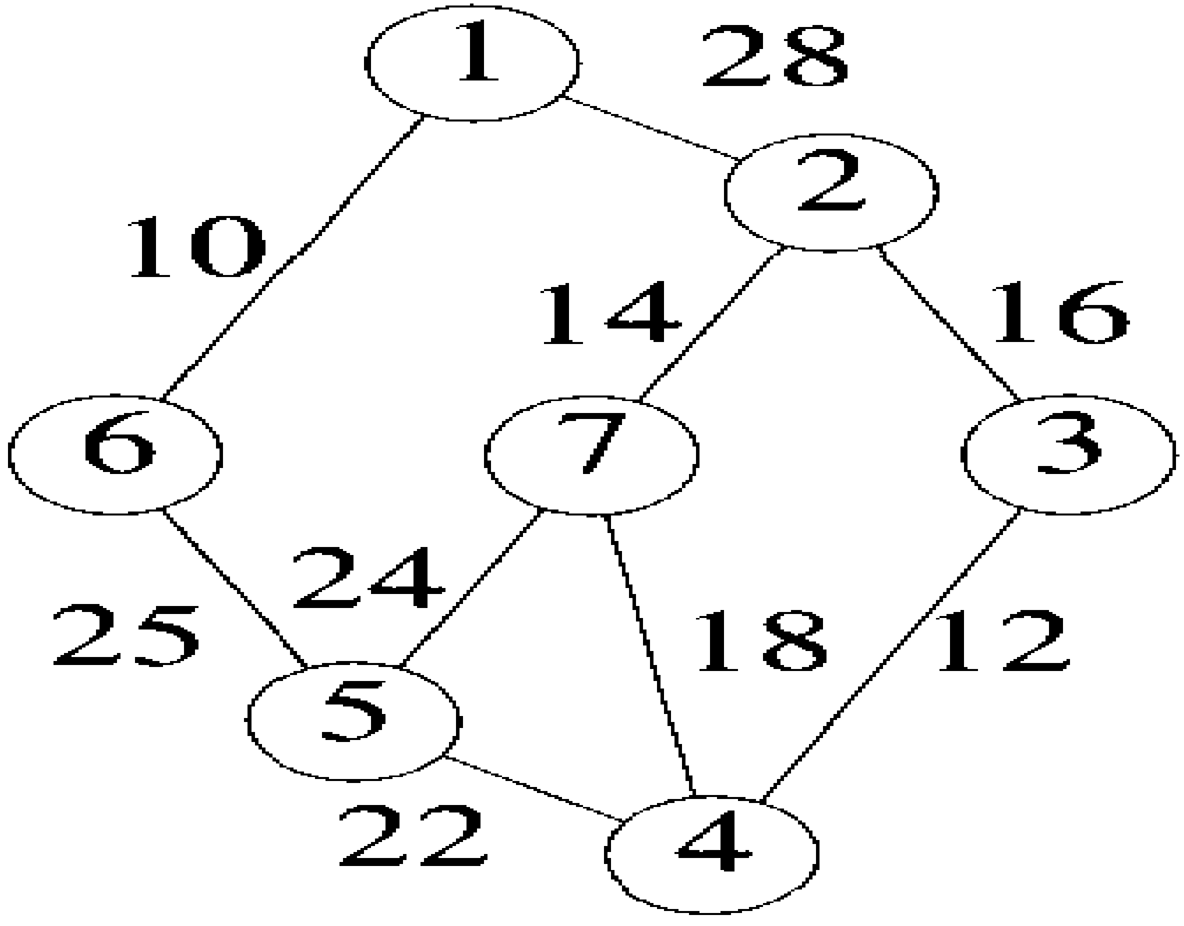
- Weighted graph:
  - Weights assigned to edges.
- Minimum Spanning Tree
  - A spanning tree with minimum sum of weights
    - It includes all vertices connected and sum of weight is minimum

Prim’s Algorithm

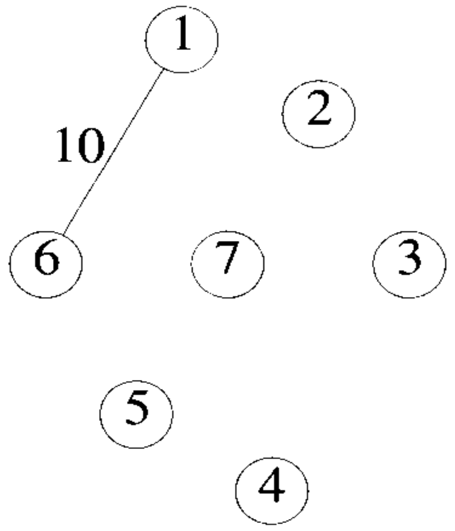
Steps

1. Select edge with smallest weight, include it in subset.
2. Select next adjacent vertex with minimum weight.
3. Include it in the subset if it does not form cycle.
4. Go to step 2 if all vertices are not included in subset.

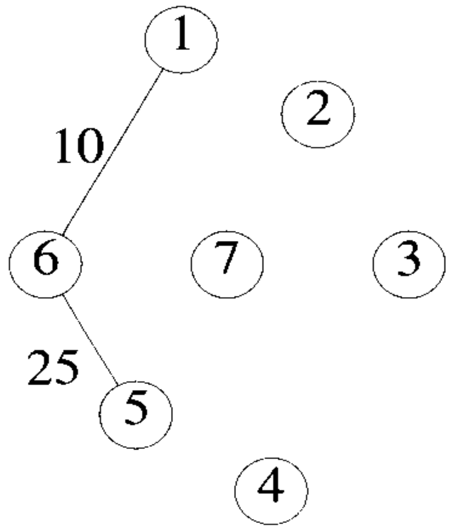
Prim's Algorithm



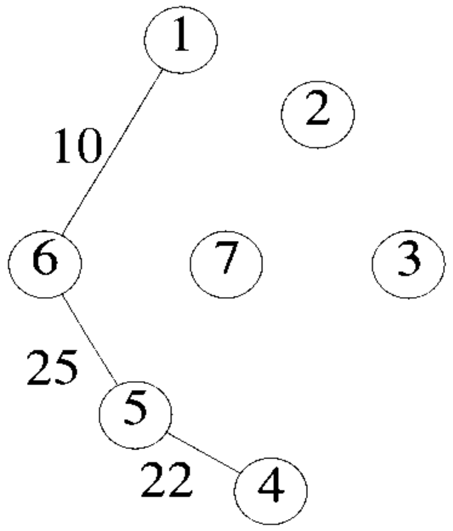
(a)


1
2
10
6
7
3
5
4

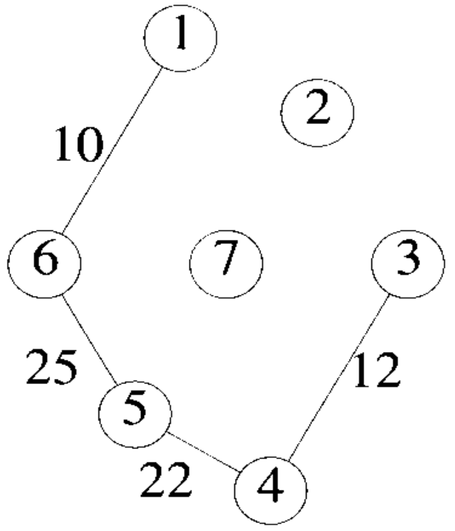
(a)


1
2
10
6
7
3
25
5
4

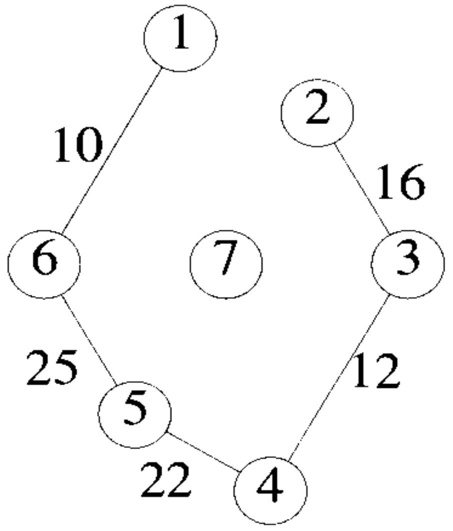
(b)


1
2
10
6
7
3
25
5
22
4

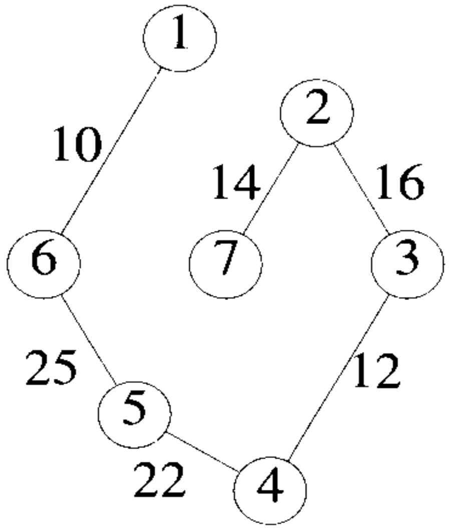
(c)


1
2
10
6
7
3
25
12
5
22
4

(d)


1
2
10
16
6
7
3
25
12
5
22
4

(e)


1
2
10
14
16
6
7
3
25
12
5
22
4

(f)

Prim’s Algorithm

- After every step, graph we get is connected

- Complexity – O(n^2)

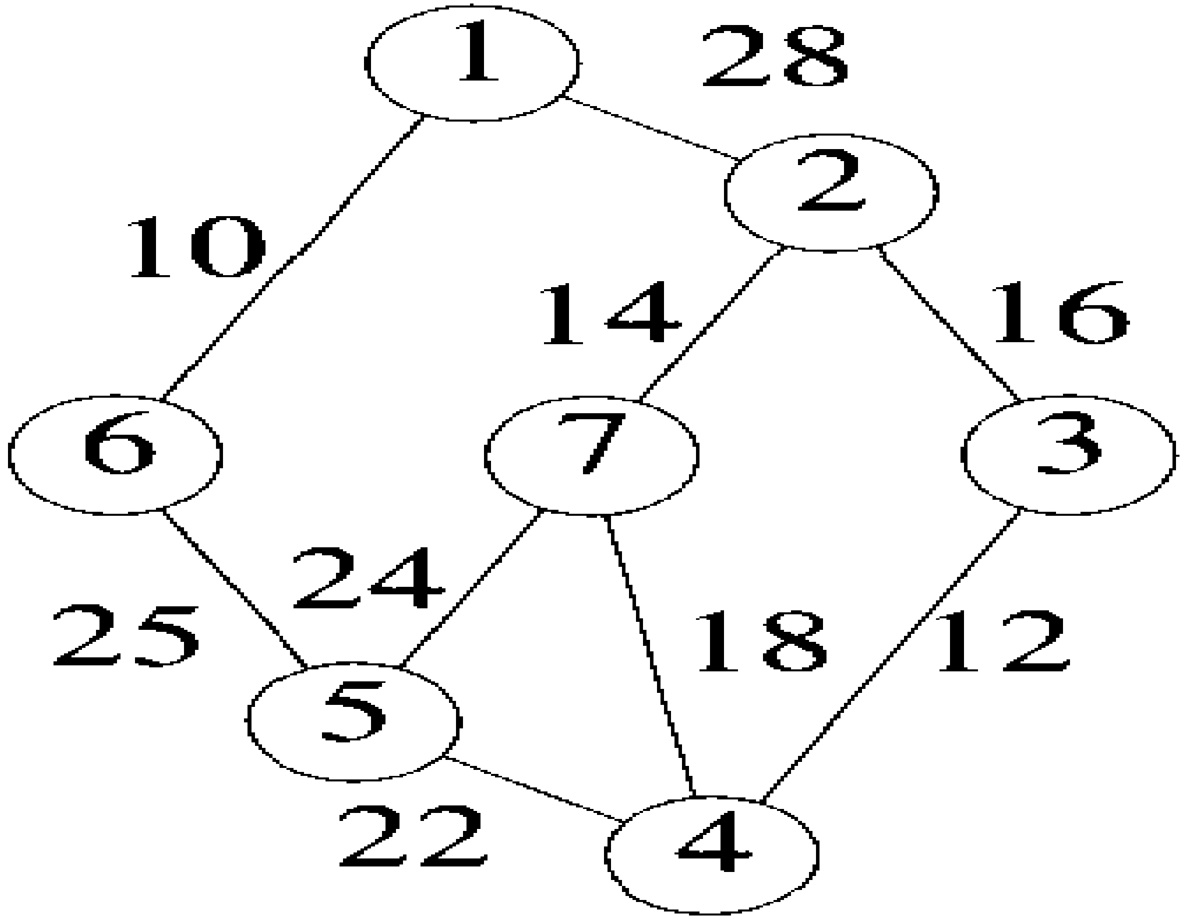
Kruskal's Algorithm

Steps

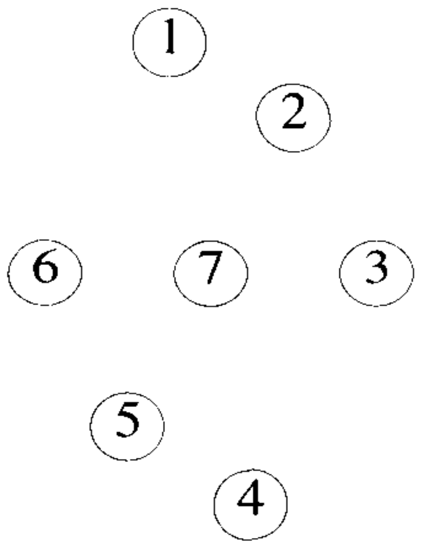
1. Sort the edge list on weight in non decreasing order.

2. Select next edge and include it in the subset if it does not form cycle.

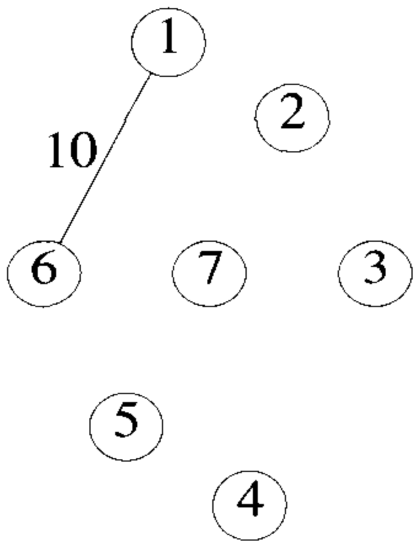
3. Go to step 2 if all vertices are not included in subset.



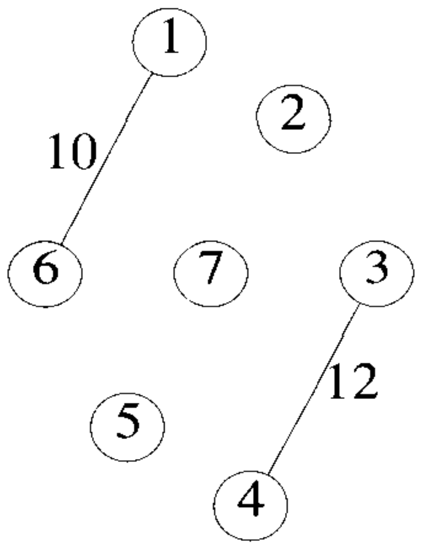
(a)


1
2
6
7
3
5
4

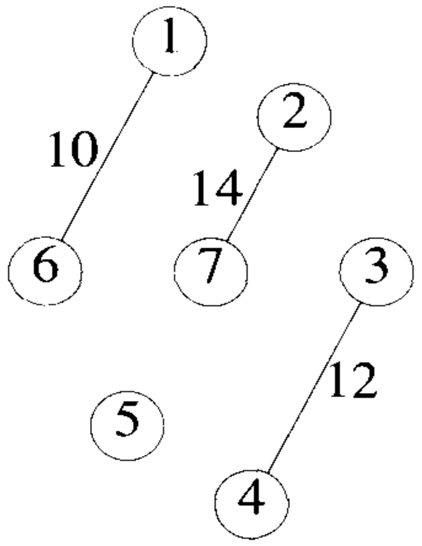
(a)


1
10
2
6
7
3
5
4

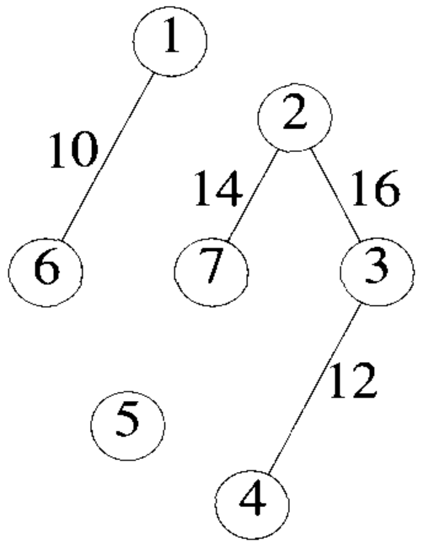
(b)


1
10
2
6
7
3
12
5
4

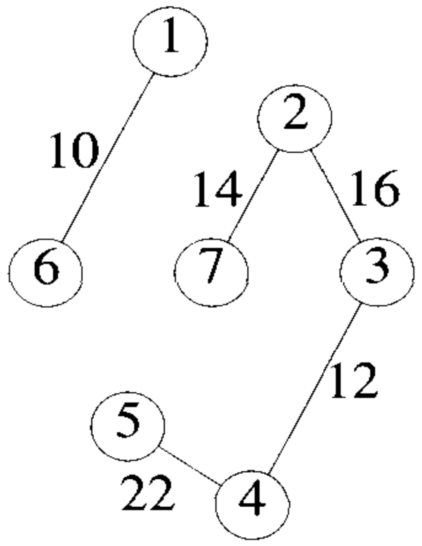
(c)


1
10
2
14
6
7
3
12
5
4

(d)


1
10
2
14
16
6
7
3
12
5
4

(e)


1
10
2
14
16
6
7
3
12
5
22
4

(f)

Kruskal’s Algorithm

- After every step graph we get may not be connected graph
- Complexity – O(E log E)

Application of MST

- Network bandwidth management-
  - Minimum bandwidth required to pass message from one node to another

Optimal Storage on Tapes

- There are n programs that are to be stored on a computer tape of length l
- Associated with each program i is a length l_i, $1 \leq i \leq n$
- All programs can be store on tape if and only if the sum of the length of the programs is at most l
- Whenever a program is to be retrieved from this tape, the tape is initially positioned at the front.

Optimal Storage on Tapes

- If the programs are stored in order
- $i_1, i_2, i_3, \ldots i_n$
- The time t_j needed to retrieve the program i_j is proportional to
  $$\sum_{1 \le k \le j} l\, i_k$$
- If all programs are retrieved equally often, then expected or mean retrieval time is (MRT) is
  $$(1/n) \sum_{1 \le j \le n} t_j$$

Optimal Storage on Tapes

- In the optimal storage on tape problem
- We are required to find a permutation for the n programs so that
- When they are stored on the tape in this order
- MRT is minimized
- This problem fits into Ordering Paradigm

Minimize $d(I) = \sum_{1\leq j\leq n} \sum_{1\leq k\leq j} l\, i_k$

O(n log n)

Sort in non-decreasing order

Example

- n=3 (l1,l2,l3)=(5,10,3)
- There are n! possible orderings

Ordering	d(I)
1,2,3	=38
1,3,2	=31
2,1,3	=43
2,3,1	=41
3,1,2	=29
3,2,1	=34

Optimal Merge Pattern

- When two or more sorted files are to be merged together,
- the merge can be accomplished by repeatedly merging sorted files in pairs.
- Approach
  - Sort the files based on no of records
  - Form pairs
  - Merge pairs, Go to above step till complete
  - Wrong results.

Optimal Merge Pattern

- Revised Approach

– Sort the files based on no of records

– Merge first two files, Go to above step till complete

– Wrong results.

Optimal Merge Pattern

- Re-revised Approach

1 Sort the files based on no of records

2 Select two smallest files,

3 merge them,

4 Place the resultant file at proper position in sorted list

5 repeat step 2 to 4 till all files are merged.

- Example.

1	2	3	4	5
10	5	9	6	12

- Find an optimal binary merge pattern for ten files whose lengths are

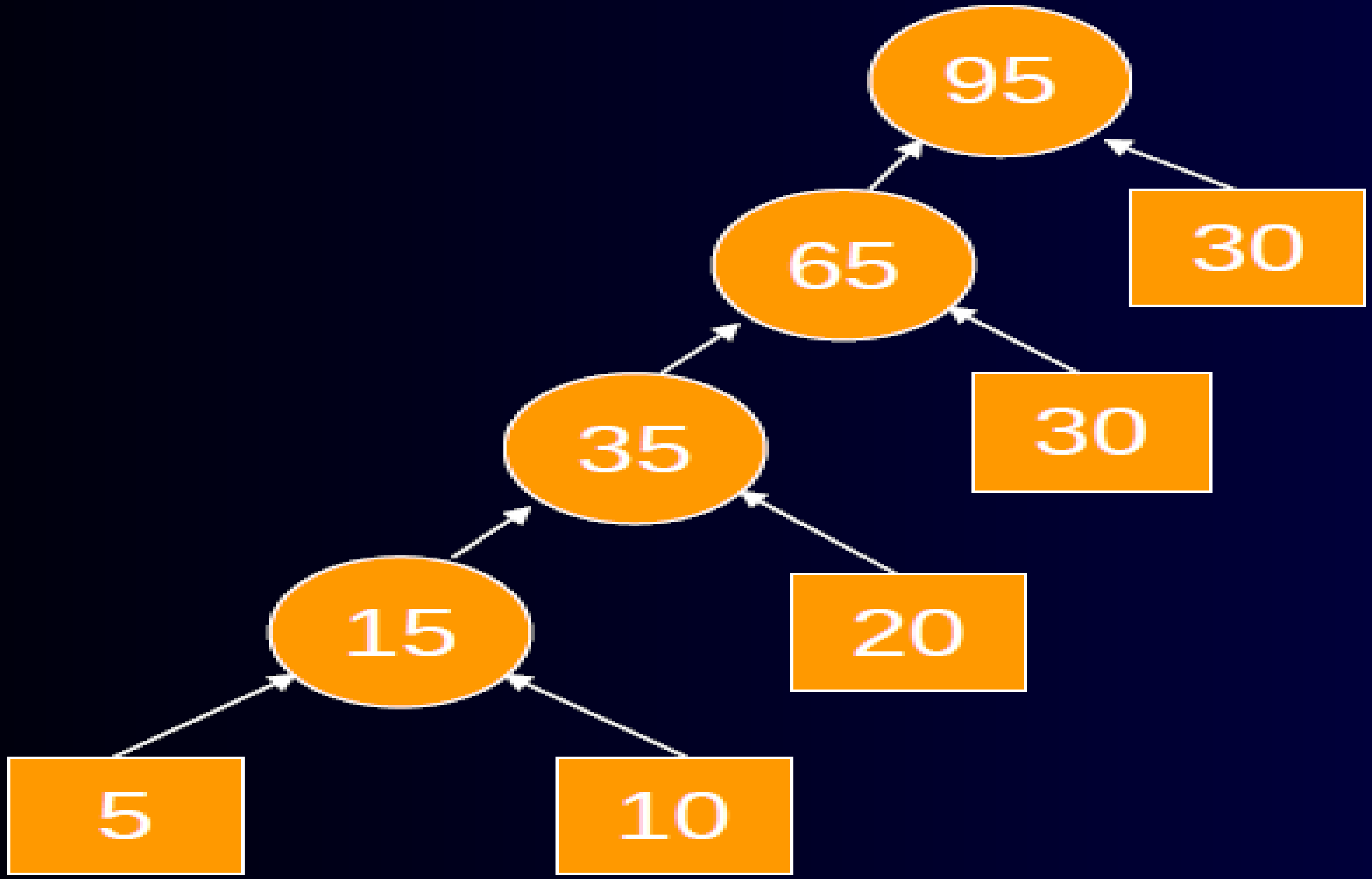
Optimal Merge Pattern

- Find an optimal binary merge pattern for ten files whose lengths are

- 28,32,12,5,84,53,91,35,3, and 11.

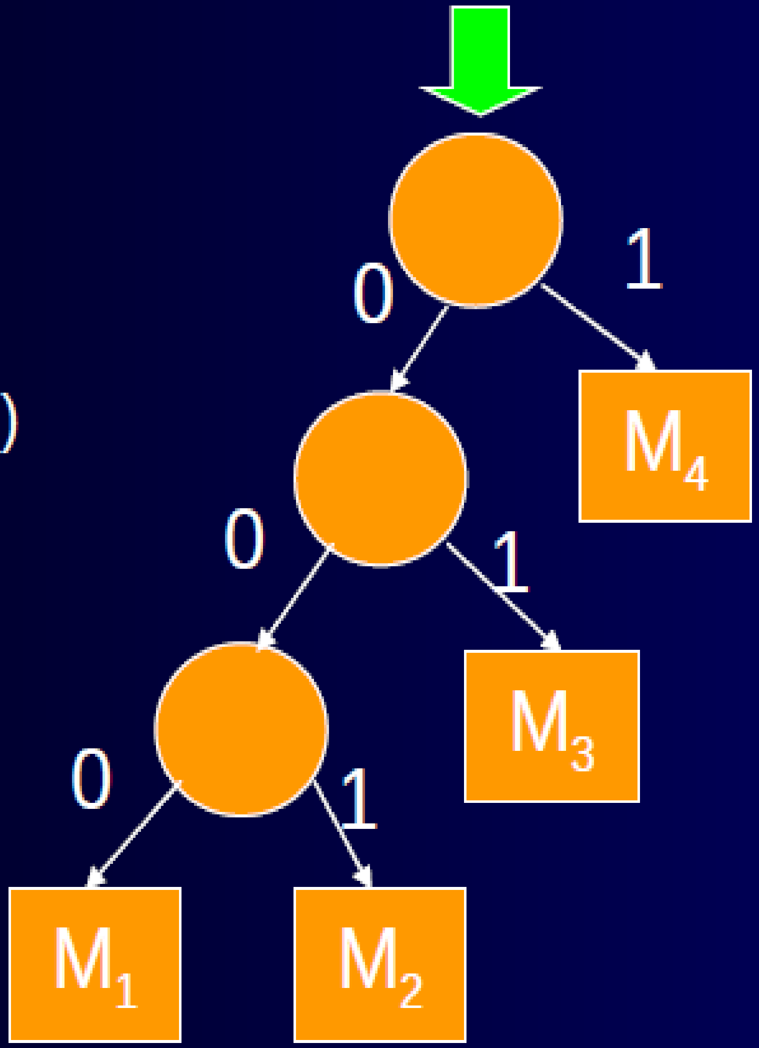
Optimal Merge Pattern

5	10	20	30	30



Optimal Merge Pattern

- Variants
  - N-way merge
  - Multiple servers
- Applicationns
  - Huffman code
  - $p(M_4) \geq p(M_3) \geq p(M_2) \geq p(M_1)$
  - $M_3 \sim 01$



Huffman Code

- Obtain a set of optimal Huffman codes for the messages (M1,..., M7) with relative frequencies
- (q1,..., q7) = (4, 5,7,8,10,12, 20).
- Draw the decode tree for this set of codes.

SINGLE-SOURCE SHORTEST PATHS

- Graphs can be used to represent the highway structure of a state or country
- vertices representing cities and edges representing roads
- The edges can then be assigned weights
- which may be either the distance between two cities connected by the edge or the average time to drive

SINGLE-SOURCE SHORTEST PATHS

- A motorist wishing to drive from city A to B
- would be interested in answers to the following questions:
- Is there a path from A to B
- If there is more than one path from A to B, which is the shortest path ?

SINGLE-SOURCE SHORTEST PATHS

- The length of a path is defined to be the sum of the weights of the edges on that path.
- The starting vertex of the path is referred to as the source, and the last vertex the destination.
- The graphs are digraphs to allow for one-way streets.
- In the problem we consider, we are given a directed graph G = (V,E),
- a weighting function cost for the edges of G,
- and a source vertex v_0.

SINGLE-SOURCE SHORTEST PATHS

- The problem is to determine the shortest paths from v_0 to all the remaining vertices of G.
- It is assumed that all the weights are positive.
- The shortest path between v_0 and some other node v is an ordering among a subset of the edges.
- Hence this problem fits the ordering paradigm.

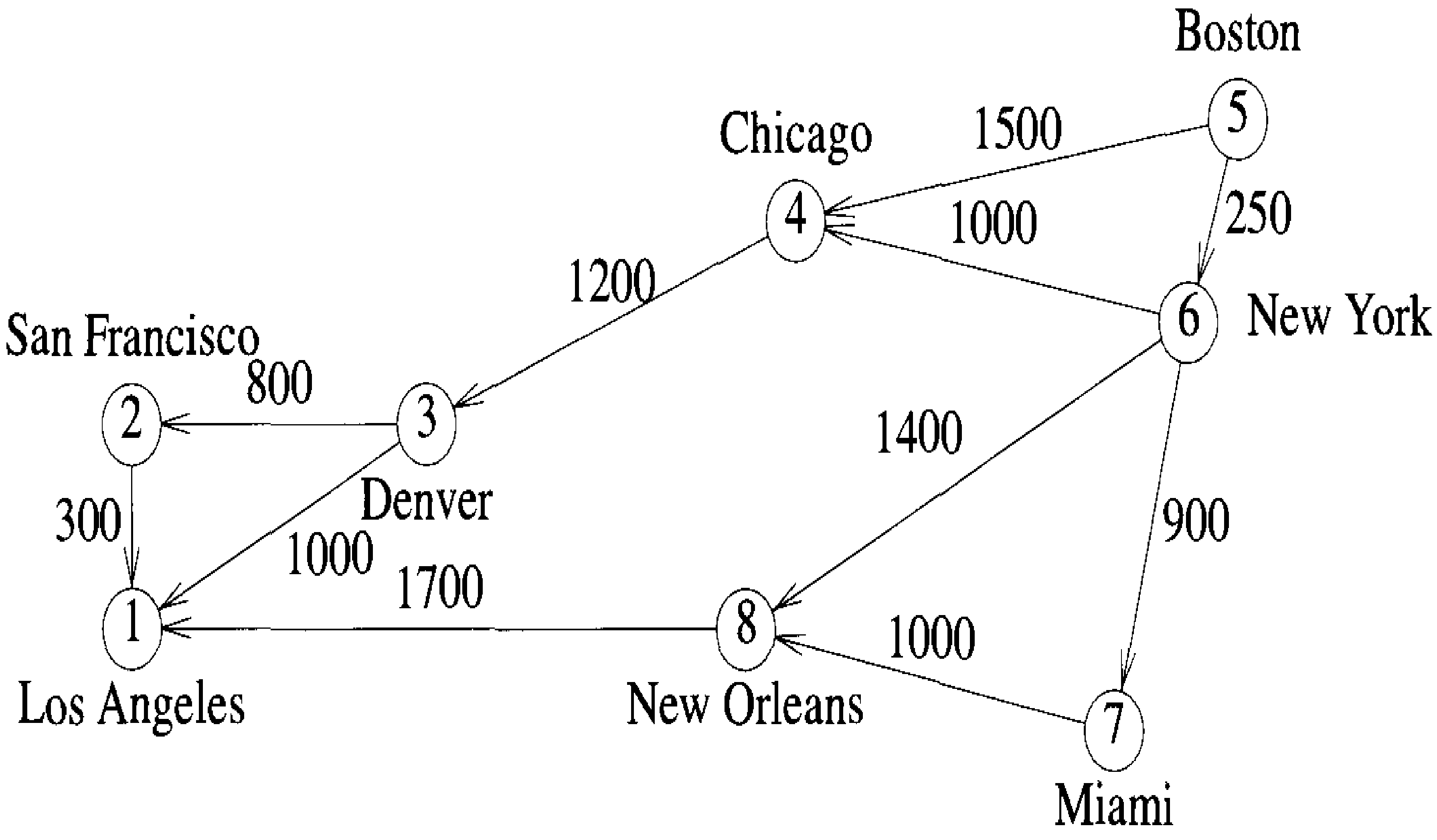
Single Source Shortest Path

- Motorist wish to visit all other cities from city A
- First shortest path to nearest city is generated, then shortest path to the second nearest city and so on…
- Form subset S containing all the cities visited.

Single Source Shortest Path: Observations

- If next shortest path is to u, then the path begins at v0 and ends at u and goes through only those vertices in S.
- Distance of next path generated must has minimum distance amongst all vertices not in S.
- Vertex u becomes member of S.

SINGLE-SOURCE SHORTEST PATHS



(a) Digraph

SINGLE-SOURCE SHORTEST PATHS

Iteration	S	Vertex	Distance							
			LA	SF	DEN	CHI	BOST	NY	MIA	NO
		selected	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
Initial	--	----	+∞	+∞	+∞	1500	0	250	+∞	+∞
1	{5}	6	+∞	+∞	+∞	1250	0	250	1150	1650
2	{5,6}	7	+∞	+∞	+∞	1250	0	250	1150	1650
3	{5,6,7}	4	+∞	+∞	2450	1250	0	250	1150	1650
4	{5,6,7,4}	8	3350	+∞	2450	1250	0	250	1150	1650
5	{5,6,7,4,8}	3	3350	3250	2450	1250	0	250	1150	1650
6	{5,6,7,4,8,3}	2	3350	3250	2450	1250	0	250	1150	1650
	{5,6,7,4,8,3,2}									

SINGLE-SOURCE SHORTEST PATHS

$$\begin{array}{c|cccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 0 & & & & & & & \\
2 & 300 & 0 & & & & & & \\
3 & 100 & 800 & 0 & & & & & \\
4 & & & 1200 & 0 & & & & \\
5 & & & & 1500 & 0 & 250 & & \\
6 & & & & 1000 & & 0 & 900 & 1400 \\
7 & & & & & & & 0 & 1000 \\
8 & 1700 & & & & & & & 0 \\
\hline
\end{array}$$

(b) Length-adjacency matrix

Thank You