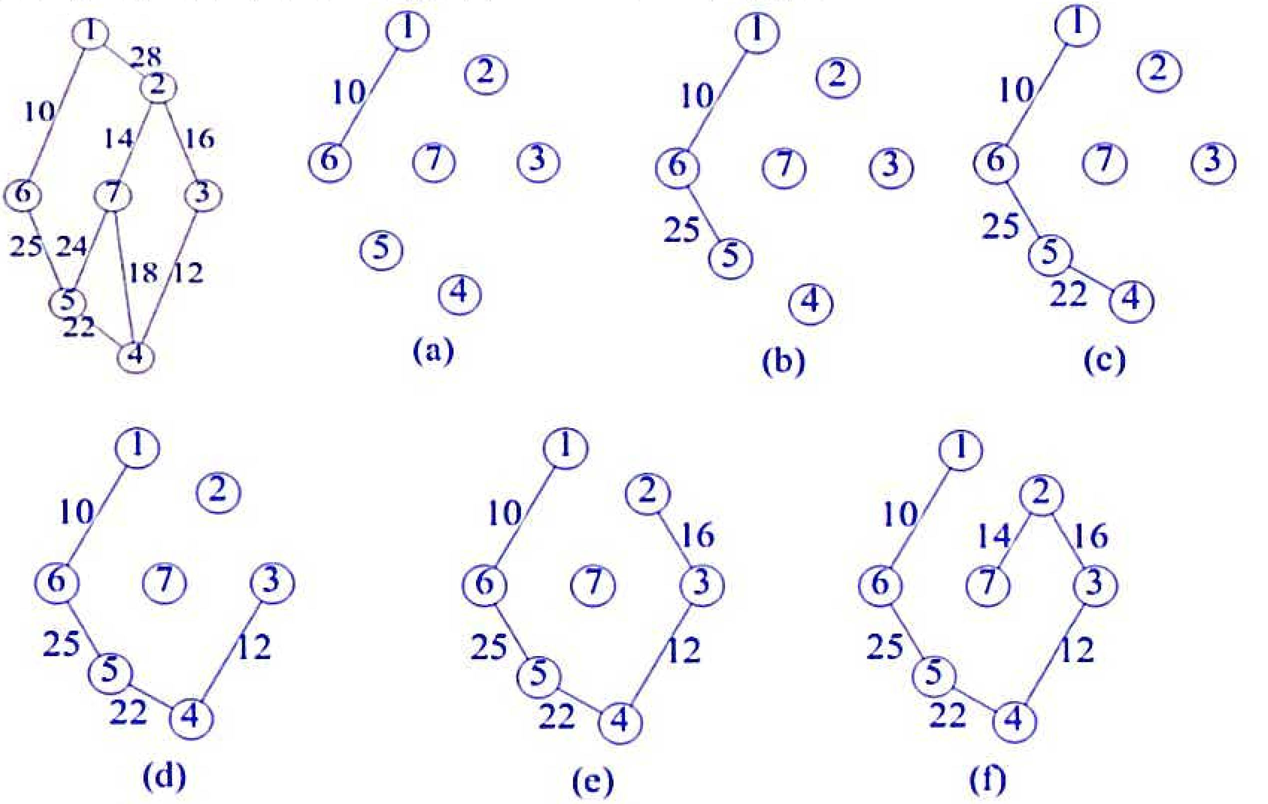
Prim's Algorithm:

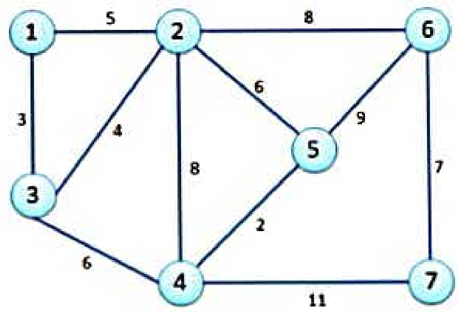
- The set of edges selected by this algorithm should form a tree.
2 • Start from an arbitrary vertex and store it in A.
- Thus, if A is the set of edges selected so far, then A forms a tree.
- The next edge (u, v) to be included in A is a minimum-cost edge not in A with the property that $AU\{(u,v)\}$ is also a tree.

Following figures shows the working of prim's method on a graph.



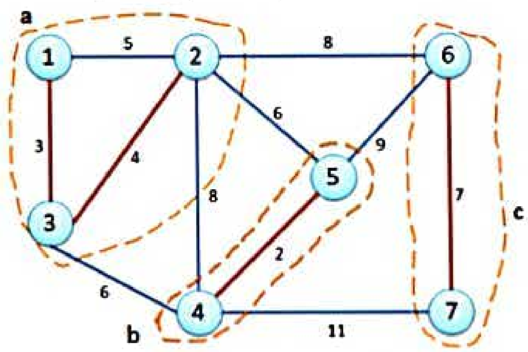
Example :

Given input graph G is 3

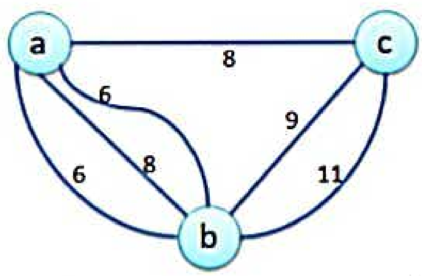


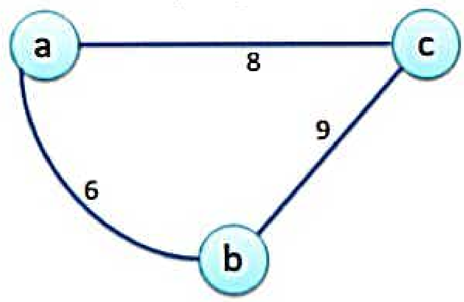
Randomly sampled 4 edges (1,3) , (2,3), (4,5) and (6,7)

Selected edges are forming three trees (forest). They should be minimum spanning trees.

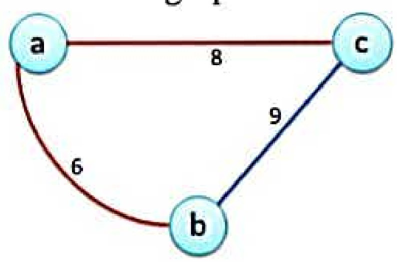


Using F translate the given Graph G into G1 and remove the heavy edges.

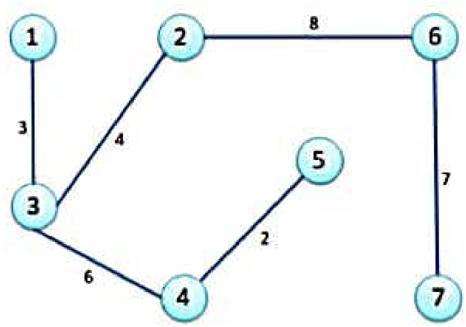




Apply the same process recursively on obtained graph



The resultant minimum spanning tree is



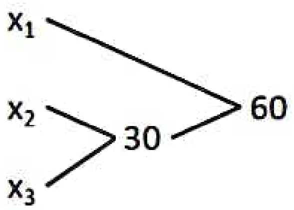
4

Optimal Merge Patterns

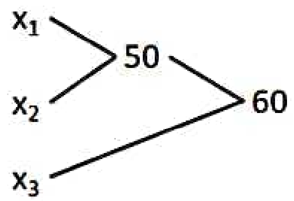
Problem : Determining an optimal way (one requiring the fewest comparisons) to pair-wise merge n sorted files.

- Merging two sorted files having n and m records to obtain one sorted file takes $O(n+m)$ time.
- When more than two sorted files are to be merged together, the merge can be accomplished by repeatedly merging sorted files in pairs.
- Given n sorted files, there are many ways in which to pairwise merge them into a single sorted file.
- Different pairings require different computing time.

Eg: Consider three files x1, x2, x3 with record lengths 30, 20, 10.



This merge pattern is taking 90 comparisons



This merge pattern is taking 110 comparisons.

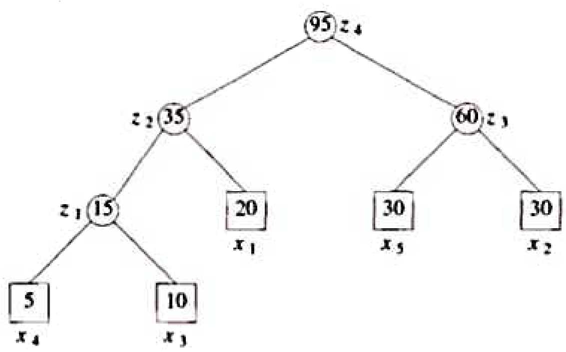
<u>Greedy method to obtain an optimal merge pattern :</u>

<u>Selection criterion:</u> since merging n-record and m-record files requires $n+m$ record moves, the obvious choice is at each step merge the two smallest size files together.

Eg: no.of files = 5
(x1, x2, x3, x4, x5) sizes (20, 30, 10, 5, 30)
Merge x3 and x4 to get z1 → |z1| = 15
Merge z1 and x1 to get z2 → |z2| = 35
Merge x2 and x5 to get z3 → |z3| = 60
Merge z2 and z3 to get z4 → |z4| = 95

The total number of record moves = 205

It can be represented as a binary merge tree.



- The leaf nodes are drawn as squares and represent the given five files. These nodes are called external nodes.
- Remaining nodes are drawn as circles and are called internal nodes.
- Each internal node has exactly two children.
- The number in each node is the length (no.of records) of the file represented by that node.

- The external node x4 is at a distance of 3 from the root node z4. i.e., the records of file x4 are moved three times, once to get z1, once to get z2 and finally to get z4.
- Total number of record moves for the binary merge tree is $\sum_{i=1}^{n} d_i \, q_i$
    - di is the distance from the root to the external node for file xi
    - q_i is the length of x_i
- This sum is the weighted external path length of the tree.

Algorithm to generate a 2-way merge tree :

```
node = record {
        node    *lchild, *rchild;
        integer weight;
}
Algorithm Tree(n)
{
        //list is a global list of n single node binary trees
        for i := 1 to n-1 do
        {
                pt := new node;
                pt->lchild := Least(list);
                pt->rchild := Least(list);
                pt->weight := pt->lchild->weight + pt->rchild->weight;
                insert(list, pt);
        }
        return (Least(list))
}
```

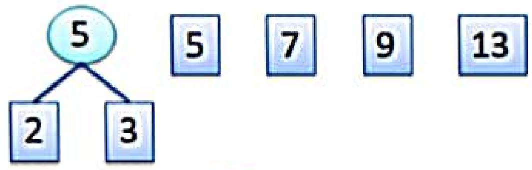
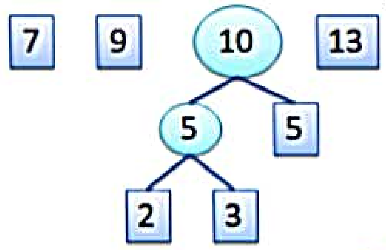
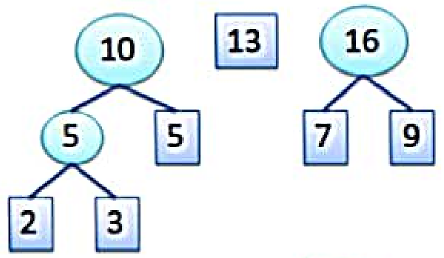
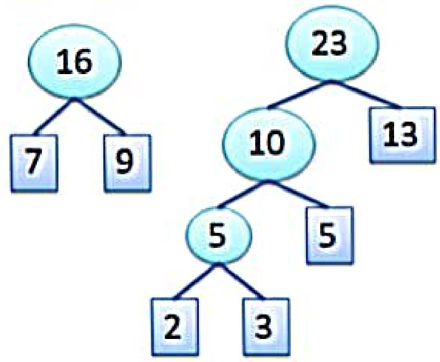
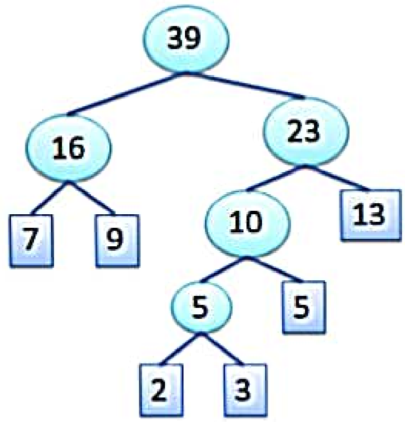
- Input to this algorithm is list of n trees. Each node in a tree has 3 fields lchild, rchild, weight.
- Initially, each tree in list has exactly one node.
- Least(list) function finds a tree in list whose root has least weight.
- Insert() function is used to insert a node into the list.

Analysis :

- Main for loop is executed *(n – 1)* times
- If list is kept in nondecreasing order according to weight value in the roots, then **Least(list)** requires O(1) time
- **Insert(list, t)** can be done in O(n) time.
- Hence, the total time taken is $O(n^2)$

Eg: Trace the algorithm for 6 files with lengths 2, 3, 5, 7, 9, 13

Iteration	List
Initially	2 3 5 7 9 13

1	
2	
3	
4	
5	

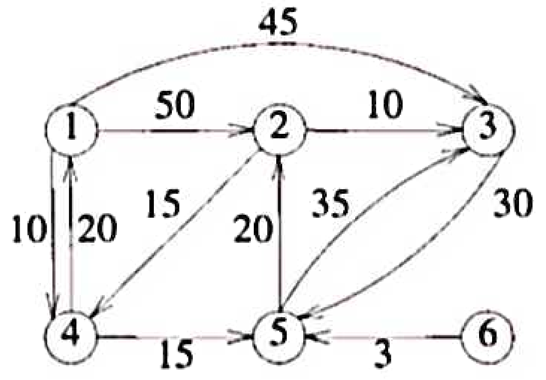
Optimal merge pattern

1. Merge files whose lengths are 2 and 3
2. Merge files whose lengths are 5 and 5
3. Merge files whose lengths are 7 and 9
4. Merge files whose lengths are 10 and 13
5. Merge files whose lengths are 16 and 23

Exercise :

Find an optimal binary merge pattern for ten files whose lengths are
28, 32, 12, 5, 84, 53, 91, 35, 3, 11

Example : Use algorithm ShortestPaths to obtain in non-decreasing order the lengths of the shortest paths from vertex 1to all remaining vertices in the digraph.



$V_0 = 1$

$$\text{Cost matrix} = \begin{bmatrix} 0 & 50 & 45 & 10 & \infty & \infty \\ \infty & 0 & 13 & 15 & \infty & \infty \\ \infty & \infty & 0 & \infty & 30 & \infty \\ 20 & \infty & \infty & 0 & 15 & \infty \\ \infty & 20 & 35 & \infty & 0 & \infty \\ \infty & \infty & \infty & \infty & 3 & 0 \end{bmatrix}$$

Trace of the algorithm

Iteration	S	Vertex selected	Distance					
			[1]	[2]	[3]	[4]	[5]	[6]
Initially	--	--	0	50	45	10	∞	∞
1	{1}	4	0	50	45	10	25	∞
2	{1, 4}	5	0	45	45	10	25	∞
3	{1,4,5}	2	0	45	45	10	25	∞
4	{1,2,4,5}	3	0	45	45	10	25	∞
5	{1,2,3,4,5}	6	0	45	45	10	25	∞
6	{1,2,3,4,5,6}							

7 Single Source Shortest Paths

- **Problem** : Given a directed graph G = (V, E), a weighting function *cost* for the edges of G, and a source vertex v_0, 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 other node v is an ordering among a subset of the edges. Hence this problem fits the ordering paradigm.
- A multistage solution must be conceived to formulate a greedy-based algorithm to generate shortest paths.
- Here the optimization measure is, each individual path must be of minimum length.
- The greedy way to generate the shortest paths from v_0 to the remaining vertices is to generate these paths in increasing order of the path length.
- First, a shortest path to the nearest vertex is generated, and then a shortest path to the second nearest vertex is generated, and so on.
- For eg : nearest vertex to (v0 = 1) is 4 cost[1, 4] = 10
  Second nearest vertex to (v0 = 1) is 5 distance = 25 path 1, 4, 5 is generated.
- Inorder to generate the shortest paths in this order, we need to determine
  1. The next vertex to which a shortest path must be generated &
  2. A shortest path to this vertex.
- Greedy algorithm to generate shortest paths is

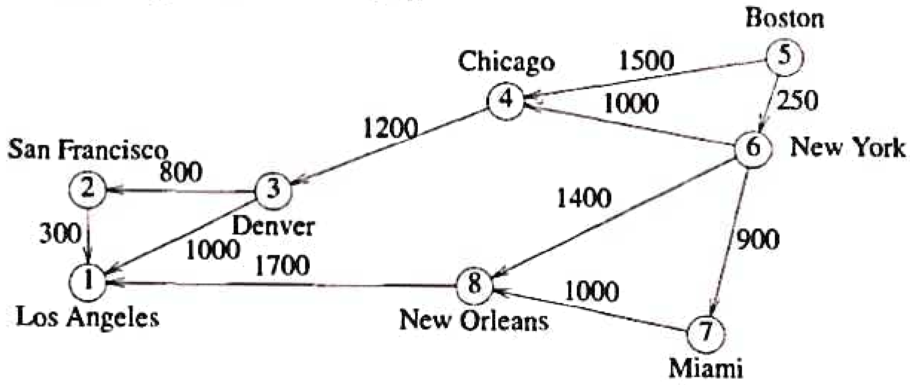
Algorithm ShortestPaths(v, cost, dist, n)

```
{
// dist[j], 1≤j ≤n, is set to the length of the shortest path from vertex v to vertex j in a digraph G with
// n  vertices. dist[v] is set to zero. G is represented by its cost adjacency matrix cost[1 :n,1:n].

        for i := 1 to n do
        {
                S[i] := false;
                dist[i] := cost[v, i];
        }
        S[v] := true;
        for num := 1 to n-1 do
        {
                choose u from among those vertices not in S such that dist[u] is minimum;
                S[u] := false;
                for (each w adjacent to u with S[w]=false) do
                        //update the distances
                        if (dist[w] > dist[u] + cost[u,w]) then
                                dist[w] := dist[u] = cost[u, w];
        }
}
```

- Let S denote the set of vertices (including v_0) to which the shortest paths have already been generated.
- For w, not in S, let dist[w] be the length of the shortest path starting from v_0, going through only those vertices that are in S, and ending at w.

8 **Example 2:** Use algorithm ShortestPaths to obtain in non-decreasing order the lengths of the shortest paths from city Boston to all remaining cities in the digraph.



Length-adjacency matrix

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

Tracing

Iteration	*S*	Vertex selected	Distance LA [1]	SF [2]	DEN [3]	CHI [4]	BOST [5]	NY [6]	MIA [7]	NO [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}									