Graphs and digraphs

Note Title 3/22/2012

Graphs and Digraphs

Definitions and Representations

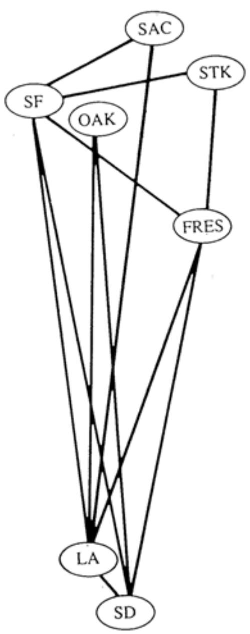
A Minimum Spanning Tree Algorithm

A Shortest-Path Algorithm

Traversing Graphs and Digraphs

Definitions and Examples

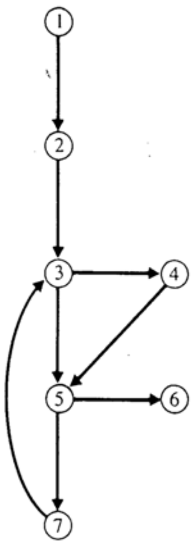
graph, G, is a pair (V, E)



$V = \{SF, OAK, SAC, STK, FRES, LA, SD\}$

$E = \{\{SF, STK\}, \{SF, SAC\}, \{SF, LA\}, \{SF, SD\}, \{SF, FRES\}, \{SD, OAK\}, \{SAC, LA\}, \{LA, OAK\}, \{LA, FRES\}, \{LA, SD\}, \{FRES, STK\}, \{SD, FRES\}\}.$

A *digraph*, G, is a pair (V, E)



$$V = \{1, 2, \ldots, 7\}$$

$$E = \{(1,2),\ (2,3),\ (3,4),\ (3,5),\ (4,5),\ (5,6),\ (5,7),\ (7,3)\}$$

(v, w) is represented in the diagrams as $v \rightarrow w$.

A *weighted graph* (or *weighted digraph*) is a triple (V, E, W)

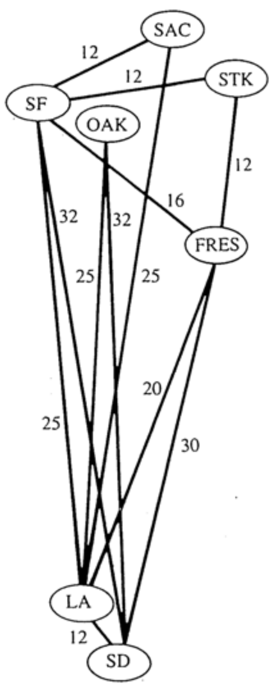


Figure 4.8 A weighted graph showing airline fares.

Finding Connected Components of a Graph

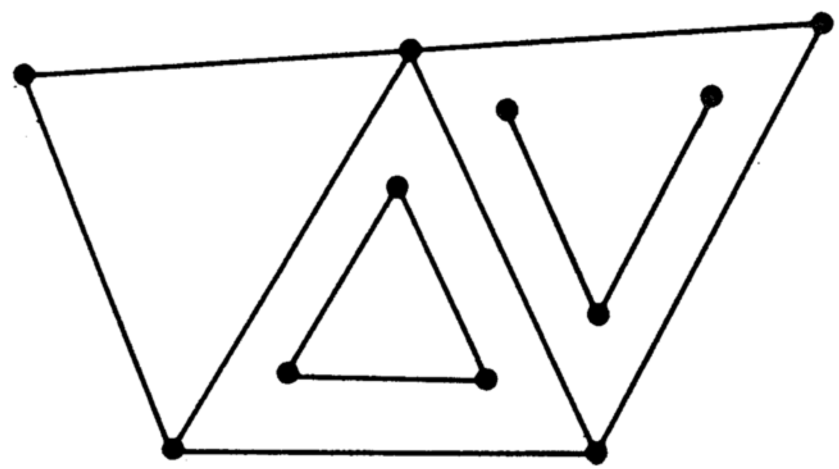
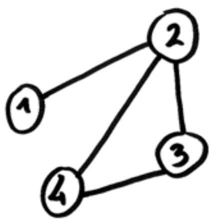
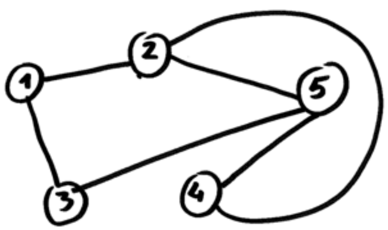


Figure 4.7 A graph with three connected components.

GRAPH



SUBGRAPH



PATH : 5 3 1 2 4

CYCLE : 5 3 1 2 4

Computer Representation of Graphs and Digraphs

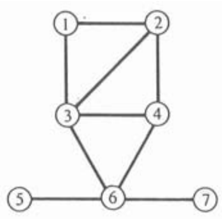
adjacency matrix $A = (a_{ij})$

Let $G = (V, E)$

$$a_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases}$$

If $G = (V, E, W)$

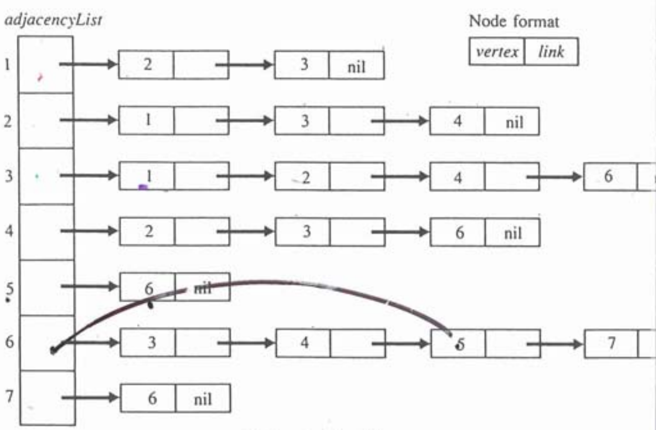
$$a_{ij} = \begin{cases} W(v_i v_j) & \text{if } v_i v_j \in E \\ c & \text{otherwise} \end{cases}$$



(a) A graph

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

(b) Its adjacency matrix.

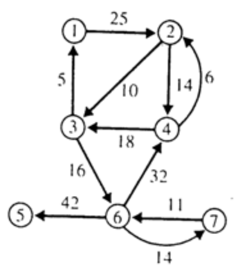


(c) Its adjacency list structure.

Figure 4.10 Representations for a graph.

```java
class DiGraph {

private int          size;    //number of vertices
private boolean[][] AdjMtrx; //adjacency matrix
private boolean[]    mark;    //to mark "visited"
```

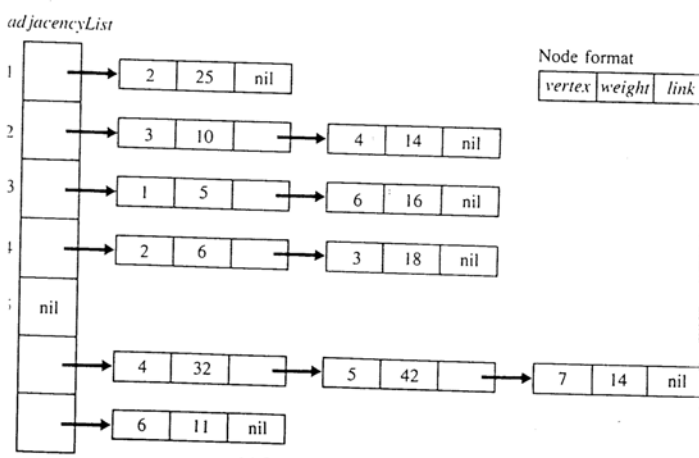
```
class DiGraph {

private int          size;    //number of vertices
private LIST[] AdjLists; //array of adjacency lists
private boolean[]   mark;    //to mark "visited"
```



(a) A weighted digraph.

$$\begin{pmatrix} 0 & 25 & \infty & \infty & \infty & \infty & \infty \\ \infty & 0 & 10 & 14 & \infty & \infty & \infty \\ 5 & \infty & 0 & \infty & \infty & 16 & \infty \\ \infty & 6 & 18 & 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & 0 & \infty & \infty \\ \infty & \infty & \infty & 32 & 42 & 0 & 14 \\ \infty & \infty & \infty & \infty & \infty & 11 & 0 \end{pmatrix}$$

(b) Its adjacency matrix.



(c) Its adjacency list structure.

```java
    private static void preOrderRec(BStree subtree)
// Traverses subtree and prints its nodes in preorder sequence
    {
        if (subtree == null) return;
        System.out.print(subtree.root + " ") ;
        preOrderRec(subtree.leftSub);
        preOrderRec(subtree.rightSub);
    }
```

Graph traversal

```
void preOrderTraversal(TreeNode T) {

    Stack  S = new Stack( );                // let S be an initially e
    TreeNode  N;                           // N points to nodes durin

    S.push(T);                // push the pointer T onto the em

    while ( !S.empty( ) ) {

        N = (TreeNode)S.pop( );                 // pop top pointer

        if (N != null) {
            System.out.print(N.info);                  // print N
            S.push(N.rlink);                   // push the right poi
            S.push(N.llink);                    // push the left poi
        }

    }
}
```

Tree : pre-order

Graph : DFS
(depth-first search)

```
void levelOrderTraversal(TreeNode T) {

    Queue Q = new Queue( );                  // let Q be a
    TreeNode N;                              // N points to

    Q.insert(T);                             // insert the

    while ( ! Q.empty( ) ) {

        N = (TreeNode) Q.remove( );                  // re

        if (N != null ) {
            System.out.print(N.info);
            Q.insert(N.llink)                // insert
            Q.insert(N.rlink)                // insert ri
        }
    }
}
```

Tree: level-by-level

Graph: BFS

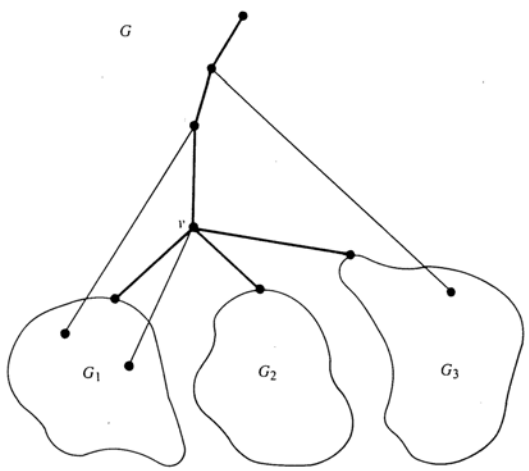
(breadth-first search)

Traversing Graphs and Digraphs

Depth-first and Breadth-first Searches

Depth-first search is a generalization of preorder traversal of trees.

In a breadth-first search, vertices are visited in order of increasing distance from the starting point

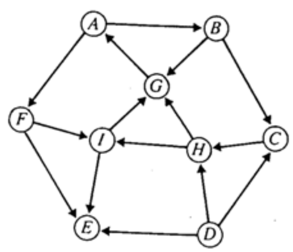


```java
public static void DFS(DiGraph G, int v) {
int w;
if (!G.IsVisited(v)) {
G.Visit(v);
for (w = G.FirstAdj(v); w != -1; w = G.NextAdj(v, w)) DFS(G, w);
System.out.println("Back out of " + v);
}
}
```

Recursive
DFS

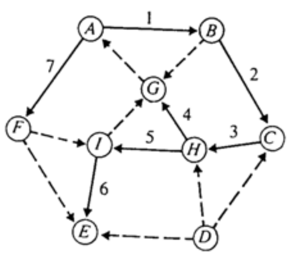
```java
public static void BFS(DiGraph G, int v) {
    if (!G.IsVisited(v)) {
        QUEUE Q = new QUEUE();
        int w;
        Q.enqueue(v);
        while (!Q.isEmpty())
        {
            v = Q.dequeue();
            if (!G.IsVisited(v)) G.Visit(v);
            for (w = G.FirstAdj(v); w != -1; w = G.NextAdj(v, w))
                if (!G.IsVisited(w)) Q.enqueue(w);
        }
    }
}
```

Non-recursive BFS

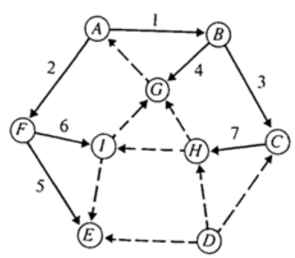


(a) A digraph.

Edges are numbered in the order traversed.



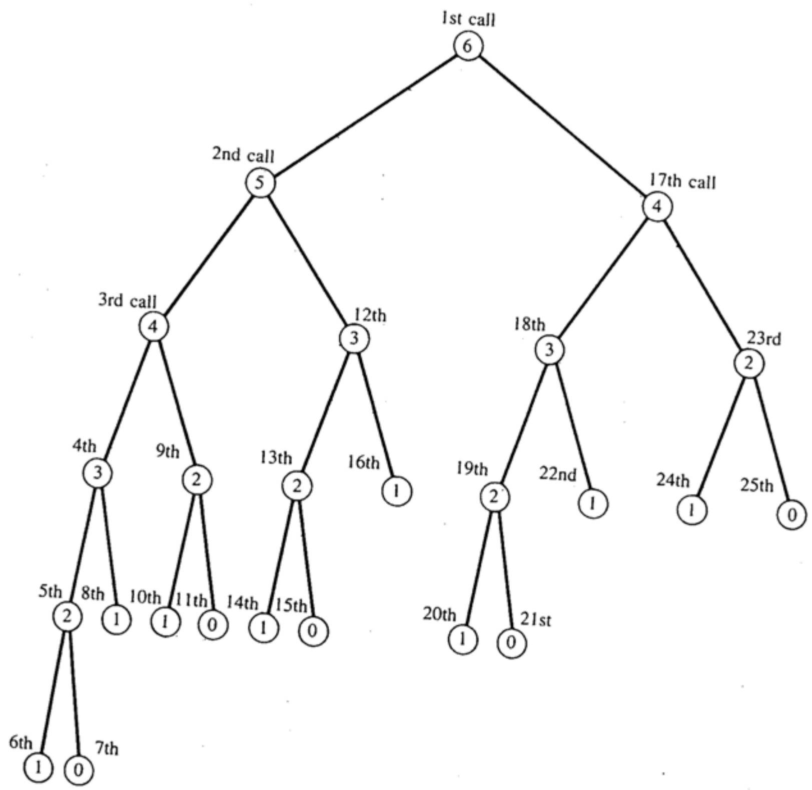
(b) Depth-first search beginning at A; order in which vertices are visited: $A\ B\ C\ H\ G\ I\ E\ F$



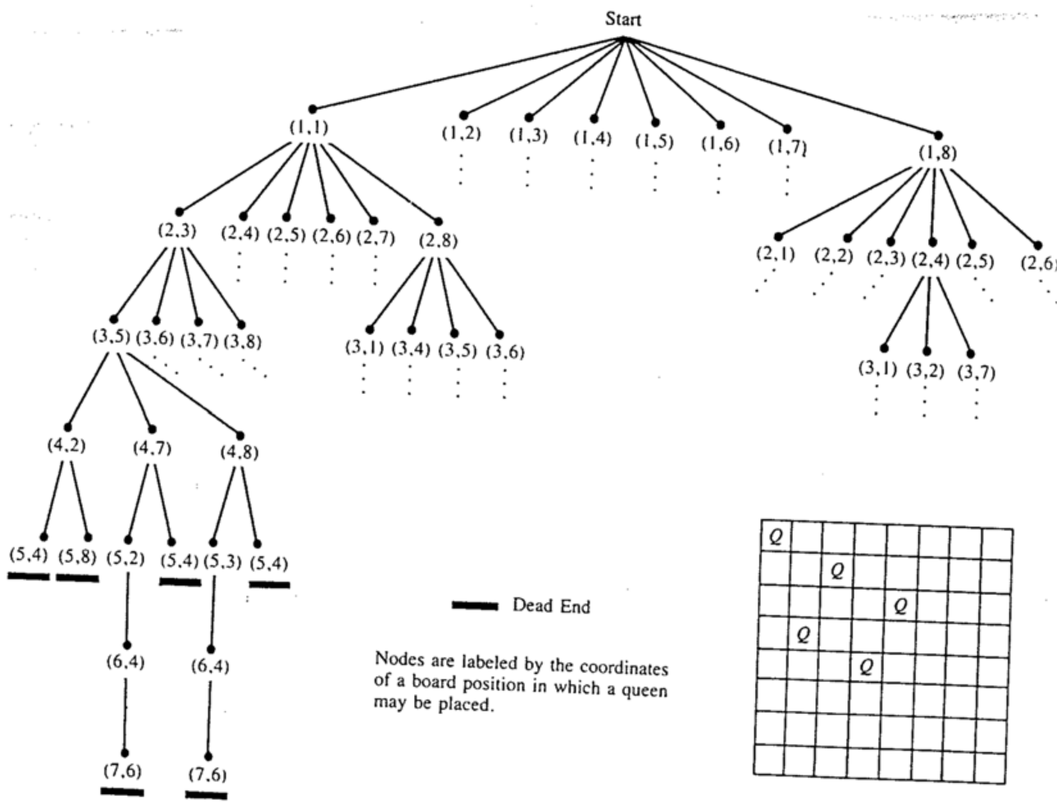
(c) Breadth-first search beginning at A; order in which vertices are visited: $A\ B\ F\ C\ G\ E\ I\ H$

DFS and BFS

spanning forests



1st call
6
2nd call
5
17th call
4
3rd call
4
12th
3
18th
3
23rd
2
4th
3
9th
2
13th
2
16th
1
19th
2
22nd
1
24th
1
25th
0
5th
2
8th
1
10th
1
11th
0
14th
1
15th
0
20th
1
21st
0
6th
1
7th
0


Start
(1,1) (1,2) (1,3) (1,4) (1,5) (1,6) (1,7) (1,8)
(2,3) (2,4) (2,5) (2,6) (2,7) (2,8)
(2,1) (2,2) (2,3) (2,4) (2,5) (2,6)
(3,5) (3,6) (3,7) (3,8)
(3,1) (3,4) (3,5) (3,6)
(3,1) (3,2) (3,7)
(4,2) (4,7) (4,8)
(5,4) (5,8) (5,2) (5,4) (5,3) (5,4)
(6,4) (6,4)
(7,6) (7,6)
Dead End
Nodes are labeled by the coordinates of a board position in which a queen may be placed.
Q
Q
Q
Q
Q