Week 13 (Notes)

Searches (DFS & BFS)

DFS

Start from the selected vertex (v), visit its unvisited neighbor (w). Repeat now the process for (w) and keep on following a path, until a "dead end is reached".

When a vertex is reached, such that all its neighbors have been visited ("dead end") back up to the last vertex visited which has still unvisited neighbors and repeat the process.

Example:

Complexity of DFS

O(n^2) if adjacency matrix is used (since there are n rows, and the length of each row is n to search for neighbors)

O(e) if adjacency list is used (since we need to examine each node in the whole list)

BFS

Algorithm:

Start from the selected vertex (v), visit all its unvisited neighbors. Then neighbors of all the neighbors and so on,

Complexity:

An O(n^2) algorithm based on adjacency matrix.

An O(e) algorithm based on adjacency list.

Example

Algorithm

```
/* Traverse: recursive traversal of a graph */
void Traverse(int v)
{
        int w;

        visited[v] = TRUE;
        Visit(v);
        for (all w adjacent to v)
                if (!visited[w])
                        Traverse(w);
}
```

```
/* BreadthFirst: perform breadth-first traversal of a graph. */
void BreadthFirst(Graph_type G)
{
        Vertexqueue_type Q;
        Boolean_type visited[MAX];
        int v, w;

        for (all v in G)
                visited[v] = FALSE;
        Initialize(Q);                  /* Set the queue to be empty.      */
        for (all v in G)
                if (!visited[v]) {
                        AddQueue(v, Q);
                        do {
                                DeleteQueue(v, Q);
                                visited[v] = TRUE;
                                Visit(v);
                                for (all w adjacent to v)
                                        if (!visited[w])
                                                AddQueue(w);
                        } while (!Empty(Q));
                }
}
```

Algorithm

Applications of Searches

- Connected and Biconnected Components
- Spanning Trees
- Sort (Topological)

Spanning Trees and Minimum Cost Spanning Trees

Definition of Spanning Tree: A tree consisting solely of edges in G and including all the vertices of G. (It is a minimal subgraph, containing all the vertices and minimum number of edges).
What is the number of edges in a spanning tree ???

Definition of Minimum Cost Spanning Tree: Edges may have costs (weights). The cost of a tree is the sum of the costs (weights) of its edges. A spanning tree with the minimum cost is the minimum cost spanning tree.

Applications: Circuit Analysis, Network Designs, Routing of Messages

Spanning Trees

Algorithm:

Use BFS or DFS sequence of vertices.

For DFS, select edge between every two adjacent vertices in the sequence, if they are neighbors. Otherwise, select the edge of the vertex under consideration to the vertex, from which DFS has chosen this vertex.

For BFS, select edges due to the neighborhood considered during BFS.