10/26/2012

Lab – 8 INFO I 500 / CSCI 609

Breadth First search (BFS)

This is a popular technique for systematically traversing vertices of a graph. The method starts traversing the graph from a given vertex V_0 which is the first vertex need to be visited. Then all the adjacent vertices of V_0 are visited.Then one of the adjacent vertices of V_n is taken and its unvisited adjacent vertices are visited next. The process continues until all of the vertices which is reachable from V_0 are visited. A BFS initiated from V_i visit all the vertices in V_n where V_n denotes the set of all the unvisited adjacent vertices of V_i. Next the process continues from any vertex in V_n. This method continues until all the vertices adjacent to V_i are fully explored. In BFS a list is maintained of vertices which have been already visited and whose adjacent vertices have not been explored. The vertices whose neighbors are yet to be visited can be stored in a queue.

This algorithm executes a depth-first search on a graph G beginning with a starting vertex A.

```
BFS ()
{

Initialize Q:
Insert the root vertex into Q.
While (Q is not empty) {
Delete an item from the Q and store it into vertex;
If (visited flag of vertex is false) {
Visit vertex;
Set visited flag of the vertex is equals to true;
Insert all the neighbors of the vertex in the Q;
}
}
```

An example of BFS is given in
http://cs.brown.edu/courses/cs016/2011/docs/files/old_lectures/BFS.pdf

Exercise 1: Imagine that you are a travel agent and a rather quarrelsome customer wants you to book a flight from New York to Los Angeles with XYZ Airlines. You try to tell the customer that XYZ does not have a direct flight from New York to Los Angeles, but the customer insists that XYZ is the only airline that he will fly. XYZ's scheduled flights are as follows:

New York to Chicago	1000 miles
Chicago to Denver	1000 miles
New York to Toronto	800 miles
New York to Denver	1,900 miles
Toronto to Calgary	1,500 miles
Toronto to Los Angeles	1,800 miles
Toronto to Chicago	1000 miles
Denver to Urbana	1,000 miles
Denver to Houston	1,500 miles
Houston to Los Angeles	1000 miles
Denver to Los Angeles	1,000 miles

You quickly see that there is a way to fly from New York to Los Angeles by using XYZ if you book connecting flights, and you book the fellow his flights.The diagram shown in Figure -1.
Write a C program to solve the problem for yourself using depth first search. The depth search won't give you an optimal solution **but BFS will give** .Change the previous given dfscode.c such that it can also find optimal paths between two places.

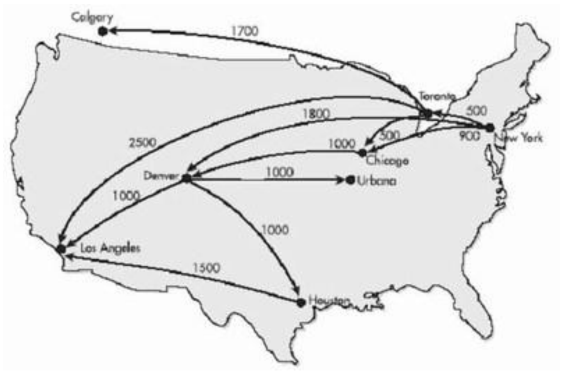

Figure -1

You can use the dfscode.c and change the code so that it can give you the optimal path for example If I want to find the optimal distance between New York to Los Angeles

It should give me New York to Toronto to Los Angeles which is 2600 miles

Other than this you may design the code in your own way and implement.

Exercise 2. Detect Cycle in a graph using DFS.

Pseudo code:

```
int n; // Number of nodes in the graph

int[] dfn[n]; // ordinal numbers for graph's visited nodes

int v, w; // represent vertices

int k=1; // ordinal number for the node to be visited next

boolean[V] inProg; // Keep track if the search on a given node is currently in progress

// Initialize indicating that no node has yet been visited

for  each v∈V  {

    dfn[v] = 0;
    inProg[v] = false;
}

// Do the search
for  each v∈V
    if (dfn[v] == 0) {
            dFS(v);
    }

void dFS(Vertex v) {

    dfn[v] = k++;
    inProg[v] = true;

    for  each w∈V such that (v,w) is an edge
            if (inProg[w])
                    declareACycle();
            else if (dfn[w] == 0)
                    dFS (w);

    inProg[v] = false;
}
```

Example 1 (no cycle should be declared) :

$$\begin{tikzcd}
C \arrow[rr] \arrow[dr] & & B \arrow[dl] & \\
& A \arrow[rr] & & E
\end{tikzcd}$$

Example 2 (cycle A, C, B, A should be declared) :

$$\begin{tikzcd}
C \arrow[rr] & & B \arrow[dl] & \\
& A \arrow[ul] \arrow[rr] & & E
\end{tikzcd}$$

Idea of the algorithm : At any time during the search from a node ***v***, all the nodes for which ***inProg*** is true, form a path. Thus, if the next node ***w*** has ***inProg*** true, there already must have been a path from ***v*** to ***w***, and therefore a cycle is formed.