Graph traversal

anhtt-fit@mail.hut.edu.vn

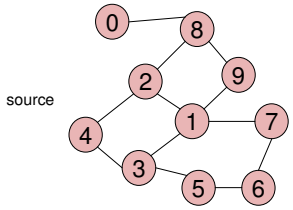
Graph Traversal

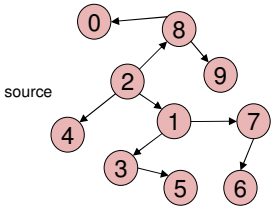
- We need also algorithm to traverse a graph like for a tree
- Graph traversal may start at an arbitrary vertex. (Tree traversal generally starts at root vertex)
- Two difficulties in graph traversal, but not in tree traversal:
  - The graph may contain cycles;
  - The graph may not be connected.
- There are two important traversal methods:
  - Breadth-first traversal, based on breadth-first search (BFS).
  - Depth-first traversal, based on depth-first search (DFS).

Breadth-First Search Traversal

Breadth-first traversal of a graph:

- Is roughly analogous to level-by-level traversal of an ordered tree
- Start the traversal from an arbitrary vertex;
- Visit all of its adjacent vertices;
- Then, visit all unvisited adjacent vertices of those visited vertices in last level;
- Continue this process, until all vertices have been visited.





Breadth-First Traversal

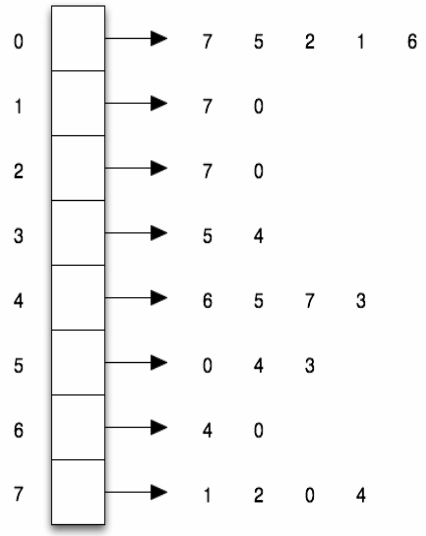
The pseudocode of breadth-first traversal algorithm:

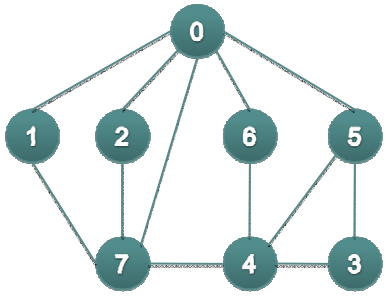
```
BFS(G,s)
   for each vertex u in V
         do visited[u] = false
   Report(s)
   visited[s] = true

   initialize an empty Q
   Enqueue(Q,s)

   While Q is not empty
       do u = Dequeue(Q)
            for each v in Adj[u]
                  do if visited[v] = false
                       then Report(v)
                            visited[v] = true
                            Enqueue(Q,v)
```

An Example

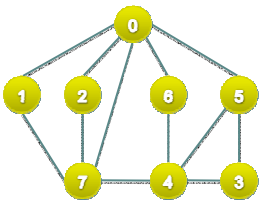




Breadth-First Search Traversal

Example of breadth-first traversal

- Visit the first vertex (in this example 0)
- Visit its adjacent nodes in *Adj*[0] :7 5 2 1 6
- Visit adjacent unvisited nodes of the those visited in last level
  - Visit adjacent nodes of 7 in *Adj*[7] : 4
  - Visit adjacent nodes of 5 in *Adj*[5] : 3
  - Visit adjacent nodes of 2 in *Adj*[2] : none
  - Visit adjacent nodes of 1 in *Adj*[1] : none
  - Visit adjacent nodes of 6 in *Adj*[6] : none
- Visit adjacent unvisited nodes of the those visited in last level
  - Visit adjacent nodes of 4 in *Adj*[4] : none
  - Visit adjacent nodes of 3 in *Adj[3]* : none
- Done



Breadth-First Traversal

Breadth-first traversal of a graph:

- Implemented with queue;
- Visit an adjacent unvisited vertex to the current vertex, mark it, insert the vertex into the queue, visit next.
- If no more adjacent vertex to visit, remove a vertex from the queue (if possible) and make it the current vertex.
- If the queue is empty and there is no vertex to insert into the queue, then the traversal process finishes.

Quiz 1

- Let implement a graph using the red black tree as in the previous lab.

```
typedef JRB Graph;
Graph createGraph();
void setEdge(Graph* graph, int v1, int v2);
int connected(Graph* graph, int v1, int v2);
```

- Write a function to traverse the graph using BFS algorithm

```
void BFS(Graph* graph, int s, int (*func)(int));
// func is a pointer to the function that process on the
   visited vertices
```

Unweighted Shortest Path Problem

- Unweighted shortest-path problem: Given as input an unweighted graph, G = (V,E), and a distinguished vertex, s, find the shortest unweighted path from s to every other vertex in G.
- After running BFS algorithm with s as starting vertex, the shortest path length from s to i is given by d[i].

Pseudo Algorithm

```
BFS(G,s)
   for each vertex u in V
         do
              visited[u] = false; d[u]=  ∞

   Report(s)
   visited[s] = true

   initialize an empty Q
   Enqueue(Q,s); d[s]=0;

   While Q is not empty
       do u = Dequeue(Q)
            for each v in Adj[u]
                 do if visited[v] = false
                      then Report(v)
                           d[v]=d[u]+1;
                           visited[v] = true
                           Enqueue(Q,v)
```

Quiz 1.1

- Continue with the exercise about Metro stations.
- Write a function that print out a unweighted shortest path between two given vertices and return its length.

```
int UShortestPath(Graph* graph, int v1, int v2);
```