Graph traversal

anhtt-fit@mail.hut.edu.vn

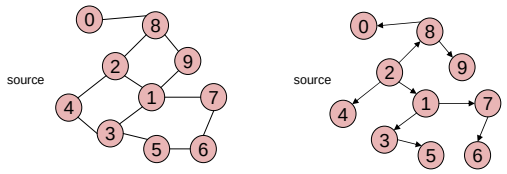
Graph Traversal

- 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

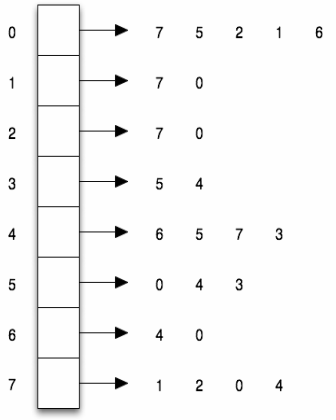
The pseudocode of breadth-first traversal algorithm:

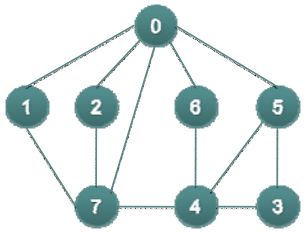
```
BFS(G,s)
   for each vertex u in V
         do visited[u] = false
   Report(s)
   visited[s] = true

   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

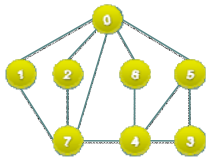




Breadth-First Search Traversal

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

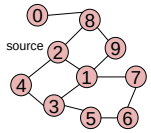
```
void BFS(Graph* graph, int s, int (*func)(int));
// func is a pointer to the function that process on the
   visited vertices
```

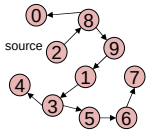
Depth-First Search

1. From the given vertex, visit one of its adjacent vertices and leave others;
2. Then visit one of the adjacent vertices of the previous vertex;
3. Continue the process, visit the graph as deep as possible until:
   - A visited vertex is reached;
   - An end vertex is reached.

Depth-First Traversal

1. Depth-first traversal of a graph:
2. Start the traversal from an arbitrary vertex;
3. Apply depth-first search;
4. When the search terminates, backtrack to the previous vertex of the finishing point,
5. Repeat depth-first search on other adjacent vertices, then backtrack to one level up.
6. Continue the process until all the vertices that are reachable from the starting vertex are visited.
7. Repeat above processes until all vertices are visited.





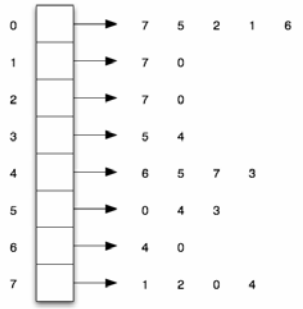
Algorithm

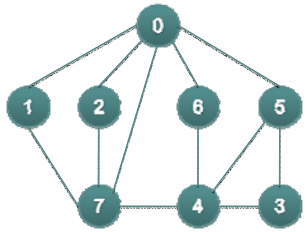
The pseudocode of depth-first traversal algorithm:

```
Boolean visited[V.size];
void DepthFirst(Graph G) {
     Vertex u;
     for each vertex u in V
        do visited[u] = false;
     for each vertex u in V
        do if visited[u] = false
              then RDFS(u);
}
 void RDFS(Vertex u){
     visited[u] = true;
     Visit(u);
     for each vertex w in Adj[u]
        do if visited[w] = false
              then RDFS(w);
}
```

Example: Depth-First Traversal

An adjacent list of a graph:

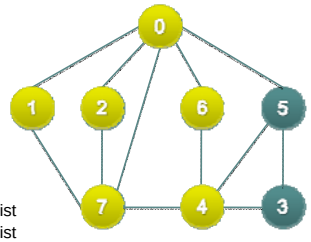




Example: Depth-First Traversal

Function calls of depth-first traversal of the graph

visit 0
- visit 7 (first on 0's list)
  - visit 1 (first on 7's list)
    - check 7 on 1's list
    - check 0 on 1's list
  - visit 2 (second on 7's list)
    - check 7 on 2's list
    - check 0 on 2's list
  - check 0 on 7's list
  - visit 4 (fourth on 7's list)
    - visit 6 (first on 4's list)
      - check 4 on 6's list
      - check 0 on 6's list

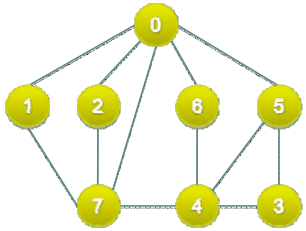


Example: Depth-First Traversal

- visit 5 (second on 4's list)
  - check 0 on 5's list
  - check 4 on 5's list
  - visit 3 (third on 5's list)
    - check 5 on 3's list
    - check 4 on 3's list
  - check 7 on 4's list
  - check 3 on 4's list

check 5 on 0's list
check 2 on 0's list
check 1 on 0's list
check 6 on 0's list

End recursive calls



Using a stack

- DFS can be implemented with stack, since recursion and programming with stacks are equivalent;
- Visit a vertex v
- Push all adjacent unvisited vertices of v onto a stack
- Pop a vertex off the stack until it is unvisited
- Repeat these steps
- If the stack is empty and there is no vertex to push onto the stack, then the traversal process finishes.

Algorithm

The pseudocode of depth-first traversal algorithm:

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

   initialize an empty stack S
   Put(S, s)

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

Quiz 2

- Continue to write a function to traverse the graph using DFS algorithm

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

Applications

- The paths traversed by BFS or DFS form a tree (called BFS tree or DFS tree).
- BFS tree is also a shortest path tree starting from its root. i.e. Every vertex v has a path to the root s in T and the path is the shortest path of v and s in G.
- DFS is used to check a the path existence between two vertices. It can be used to determine if a graph is connected.

Quiz 3

- Add a new functionality in your program in order to find a shortest path between two metro stations.