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 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.

BFS algorithm

- 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.

BFS demo

- 51demo-bfs.ppt

Pseudocode

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

    initialize an empty Q
    Enqueue(Q,s)

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

Quiz 1

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

```
typedef JRB Graph;
Graph createGraph();
void addEdge(Graph graph, int v1, int v2);
int adjacent(Graph graph, int v1, int v2);
…
```

- Write a function to traverse the graph using BFS algorithm

```
void BFS(Graph graph, int start, int stop, void (*func)(int));
```

  - start is the first vertex to visit
  - stop is the vertex to be visited at the end, if stop = -1, all the vertices may be visited
  - func is a pointer to the function that process on the visited vertices

Example

```
void printVertex(int v) { printf(“%4d”, v); }

    Graph g = createGraph();
    addEdge(g, 0, 1);
    addEdge(g, 1, 2);
    addEdge(g, 1, 3);
    addEdge(g, 2, 3);
    addEdge(g, 2, 4);
    addEdge(g, 4, 5);
    printf("\nBFS: start from node 1 to   5 : ");
    BFS(g, 1, 4, printVertex);
    printf("\nBFS: start from node 1 to all : ");
    BFS(g, 1, -1, printVertex);
```

Instruction

- Use the double linked list data structure in libfdr to represent a queue as the following
- To create a queue
  - Dllist queue = new_dllist();
- To add a visited node
  - dll_append(queue, new_jval_i(v))
- To check if the queue is empty
  - dll_empty(queue)
- To get a vertex from the queue
  - node = dll_first(queue)
  - v = jval_i(node->val)
  - dll_delete_node(node)

Depth-First Search

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

Depth-First Traversal

- Start the traversal from an arbitrary vertex;
- Apply depth-first search;
- When the search terminates, backtrack to the previous vertex of the finishing point,
- Repeat depth-first search on other adjacent vertices, then backtrack to one level up.
- Continue the process until all the vertices that are reachable from the starting vertex are visited.
- Repeat above processes until all vertices are visited.

DFS algorithm

- 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.

DFS demo

- demo-dfs-undirected.ppt

Pseudocde

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

   initialize an empty stack S
   Put(S, s)

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

Quiz 2

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

  ```
  void DFS(Graph graph, int start, int stop, void (*func)(int));
  ```

  - start is the first vertex to visit
  - stop is the vertex to be visited at the end, if stop = -1, all the vertices may be visited
  - func is a pointer to the function that process on the visited vertices

Solution

- graph_traversal.c

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 the metro program in order to find a shortest path between two metro stations.