1 Introduction

In this lecture we will look at Graph Traversals: DFS and BFS.

2 Graph Traversals

There are two famous algorithms for graph traversal: Depth-first search (DFS) and Breadth-first search (BFS). Both algorithms work for both undirected and directed graphs, though we focus more on undirected graphs.

3 DFS

The traversal is started from an arbitrary vertex \( v \), which is called as the root of the DFS. The root is marked as visited. An arbitrary (unmarked) vertex \( v_1 \) connected to \( v \) is then picked and a DFS starting from \( v_1 \) is performed recursively. The recursion stops when it reaches a vertex \( v \) such that all the vertices connected to \( v \) are already marked. If after the DFS from \( v_1 \) terminates all the vertices connected to \( v \) are also marked then the DFS for \( v \) also terminates. Otherwise another arbitrary unmarked vertex \( v_2 \) connected to \( v \) is picked and a DFS starting from \( v_2 \) is performed, and so on.

To incorporate different applications, we do pre-work at the time the vertex is marked and post-work after we backtrack from an edge and find that the edge leads to a marked vertex.

**Complexity:** Each edge is looked exactly twice (once from each end). So the running time is \( O(|E|) \) if the graph is connected and \( O(|V| + |E|) \) in general due to vertices that are not connected to anything.

**Theorem 1** If \( G \) is connected, then all vertices are marked and all edges are visited by the DFS.
Algorithm 1 DFS(G, v)
1: Mark v
2: DFS-Number[v]=DFS-N, DFS-N++
3: Perform pre-work on v;
4: for all edges v, w do
5:  if w is unmarked then
6:    DFS(G, w); add edge (v, w) to DFS-Tree;
7:    Perform post-work for (v, w);

Proof: The proof is by contradiction. Let U be the set of unmarked vertices at the end. Since G is connected at least one vertex u ∈ U is connected to a marked vertex say v and thus we should have had visited u when we visited v by definition of the algorithm. Now since all vertices are visited and since whenever a vertex is visited all its incident edges are considered, all the edges in the graph are considered too.

Thus above theorem gives an application of DFS for checking the connectivity of the graph, i.e., the number of marked vertices that we count should be n for the graph to be connected. DFS has various other applications as well.

4 BFS

It traverses the graph level-by-level. If we start from a vertex v, then all “children” of v are visited first. The second level includes a visit to all the “grandchildren” of v and so on. The procedure is non-recursive.

Algorithm 2 BFS(G, v)
1: Mark v
2: BFS-Number[v]=BFS-N=1, level[v]=0;
3: Put v in a queue (First In First Out);
4: while The queue is not empty do
5:  Remove the first vertex w from the queue;
6:  Perform pre-work on w;
7:  for All edges (w, x) such that x is unmarked do
8:    Mark x; BFS-Number[x]=BFS-N, BFS-N++; level[x]=level[w]+1;
9:    Add (w, x) to the BFS-Tree
10:   Put x in the queue.

Note that in both DFS and BFS, we assign a DFS-Number or BFS-Number to each vertex and we create a DFS-Tree or a BFS-Tree.

4.1 Complexity

Again we enqueue and dequeue each vertex once and we visit each edge only twice. So the total running time is O(|V| + |E|).
Theorem 2  For each vertex \( w \), the path from the root to \( w \) in the BFS-Tree is a shortest path from the root to \( w \) in \( G \).

Proof: Proof follows by induction of the level of a vertex. ■

5 Main Properties of DFS and BFS Trees

Theorem 3 (Main Property of BFS-Trees) If \((u,v)\) is an edge of \( G \) not belonging to \( T \), then it connects two vertices whose level numbers differ by at most 1.

Proof: The idea is to consider the first of \( v \) and \( u \) which is visited and then the other vertex has difference at most one (or potentially zero). You can see detailed proofs in CLRS or the recommended book [1]. ■

Theorem 4 (Main Property of DFS-Trees) Every edge in \( G \) not belonging to \( T \), connects two vertices of \( G \), one of which is the ancestor of the other in \( T \), i.e., there are no cross-edges.

Proof: Let \((v,u)\) be an edge of \( G \), and suppose that \( v \) is visited by DFS before \( u \). After \( v \) is marked, we perform DFS starting from all its neighbors. Since \( u \) is a neighbor of \( v \), DFS either starts from \( u \) in which case \((v,u)\) belongs to \( T \) or the DFS will visit \( u \) before it backtracks from \( v \), in which case \( u \) is a descendant of \( v \) in \( T \).

See the properties of DFS in directed graphs in the book [1].

References