Graph Traversals: Scanning a graph or traversing it.

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

DFS: the traversal is started from an arbitrary vertex r, which is called the root of DFS. The root is <u>marked</u> as visited. An arbitrary (unmarked) vertex r_1 connected to r is then picked and a DFS starting from r_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 r_1 terminates all the vertices connected to r are already marked, then DFS for r terminates. Otherwise another arbitrary unmarked vertex r_2 connected to r is picked, a DFS starting from r_2 is performed, and so on.

```
Algorithm DFS (G,v);
 begin
   mark v; DFS_Number[v]=DFSN; DFSN++;
   Perform PreWork on v;
   for all edges (v,w) do
     if w is unmarked then DFS(G,w); add edge(v,w) to tree DFS_tree;
     perform postWork for (v,w)
 end;
```

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

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

<u>Thm:</u> If G is connected then all vertices are marked and all edges visited the DFS algorithm.

<u>Pf:</u> 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 from U is connected to a marked vertex v and thus we should visit u when we visit v by definition of the algorithm. Now since all vertices are visited and since whenever a vertex is visited all its edges are considered, all edges are considered too. ☒

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

BFS: traverses the graph level by level. If we start from a vertex v, then all v's "children" are visited first. The second level includes a visit to all the "grandchildren" and so on. The procedure is non-recursive.

Algorithm BFS (G, v)

begin
  mark v; BFS_Number[v] = BFSN=1; Level[v]=0;
  Put v in a queue {First In First out}
  while the queue is not empty do
    remove the first vertex w from the queue
    Perform pre Work on w;
    for all edges (w, x) such that x is unmarked do
      mark x; BFS_Number[x] = BFSN; BFSN++; Level[x] = level[w]+1;
      add (w, x) to the tree BFS_Tree;
      put x in the queue;
end;

complexity: Again we enqueue and dequeue each vertex once and we visit each edge only twice. So the total running time is $O(|V|+|E|)$

Note that in both DFS and BFS, we assign a DFS_Number or BFS_Number to each vertex and we create a BFS-tree or a DFS-tree.

Thm: 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 is by induction on the level of a vertex.

☆ Thm: If (v, w) is an edge of G not belonging to T, then it connects two vertices whose level numbers differ by at most 1. Pf: consider the first of v and w which is visited and the other vertex has difference at most one (potentially zero) ☒

You can see the proofs in the CLRS, the recommended book.

The above Thm☆ is the main property of BFS-trees.

☆☆ Thm (the main property of DFS-trees): Every edge in G not belonging to T, connect two vertices of G, one of which is the ancestor of the other in T (i.e., no cross edges)

Pf: 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 start 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.