Traversing a graph: BFS and DFS

(CLRS 22.2, 22.3)

The most fundamental graph problem is traversing the graph.

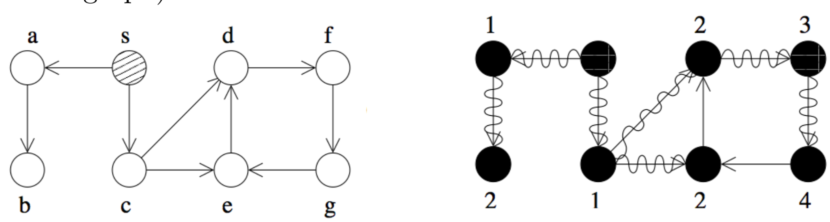
- There are two standard (and simple) ways of traversing all vertices/edges in a graph in a systematic way: BFS and DFS.
- Most fundamental algorithms on graphs (e.g finding cycles, connected components) are applications of graph traversal.
- Like finding the way out of a maze (maze = graph). Need to be careful to not get stuck in the graph, so we need to mark vertices that we've encountered; and we need to make sure we don't skip anything.
- Basic idea: over the course of the traversal a vertex progresses from undiscovered, to discovered, to completely-discovered:
  - undiscovered: initially (WHITE)
  - discovered: after it's encountered, but before it's completely explored (GRAY)
  - completely explored: the vertex after we visited all its incident edges (BLACK)
- We start with a single vertex and evaluate its outgoing edges:
  - If an edge goes to an undiscoverd vertex, we mark it as discovered and add it to the list of discovered vertices.
  - If an edge goes to a completely explored vertex, we ignore it (we've already been there)
  - If an edge goes to an already discovered vertex, we ignore it (it's on the list).
- Analysis: Each edge is visited once (for directed graphs), or twice (undirected graphs — once when exploring each endpoint) $\Rightarrow O(|V| + |E|)$
- Depending on how we store the list of discovered vertices we get BFS or DFS:
  - queue: explore oldest vertex first. The exploration propagates in layers form the starting vertex.
  - stack: explore newest vertex first. The exploration goes along a path, and backs up only when new unexplored vertices are not available.

Breadth-first search (BFS)

- We use a queue Q to hold all gray vertices—vertices we have seen but are still not done with.
- We remember from which vertex a given vertex v is colored gray – i.e. the node that discovered v first; this is called parent[v].
- We also maintain $d[v]$, the length of the path from s to v. Initially $d[s] = 0$.

```
BFS(s)
    color[s] = gray
    d[s] = 0
    ENQUEUE(Q, s)
    WHILE Q not empty DO
        DEQUEUE(Q, u)
        FOR each v ∈ adj[u] DO
            IF color[v] = white THEN
                color[v] = gray
                d[v] = d[u] + 1
                parent[v] = u //(u,v) is a tree-edge
                ENQUEUE(Q, v)
            //ELSE v is not white, (u,v) is non-tree edge
            color[u] = black
```

- Example (for directed graph):



- If graph is not connected we start the traversal at all nodes until the entire graph is explored.

```
BFS(G)

    FOR each vertex u ∈ V DO
        IF color[u] = white THEN BFS(u)
```

Properties of BFS

- During BFS(v) each edge in G is classified as:
  - tree edge: an edge leading to an unmarked vertex
  - non-tree edge: an edge leading to a marked vertex.
- Each vertex, except the source vertex s, has a parent; these edges $(v, parent[v])$ define a tree, called the *BFS-tree*.
- **Lemma:** On a directed graph, BFS(s) reaches all vertices reachable from s. On an undirected graph, BFS(s) visits all vertices in the connected component (CC) of s, and the BFS-tree obtained is a spanning tree of $CC(s)$.

  Proof idea: Assume by contradiction that there is a vertex v in CC(u) that is not reached by BFS(u). Since u, v are in same CC, there must exist a path $v_0 = u, v_1, v_2, ..., v_k, v$ connecting u to v. Let v_i be the last vertex on this path that is reached by BFS(u) (v_i could be u). When exploring v_i, BFS must have explored edge (v_i, v_{i+1}),..., leading eventually to v. Contradiction.
- **Lemma:** BFS(s) runs in $O(|V_c| + |E_c|)$, where V_c, E_c are the number of vertices and edges in CC(s). When run on the entire graph, BFS(G) runs in $O(|V| + |E|)$ time. Put differently, BFS runs in linear time in the size of the graph.

  Proof: It explores every vertex once. Once a vertex is marked, it's not explored again. It traverses each edge twice. Overall, $O(|V| + |E|)$.
- **Lemma:** Let x be a vertex reached in BFS(s). Its distance $d[x]$ represents the the shortest path from s to x in G.

  Proof idea: All vertices v which are one edge away from s are discovered when exploring s and are set with $d[v] = 1$. Similarly all vertices that are one edge away from vertices at distance 1, are explored and their distance set to $d = 2$. And so on.
- **Lemma:** For undirected graphs, for any non-tree edge (x, y) in BFS(v), the level of x and y differ by at most one.

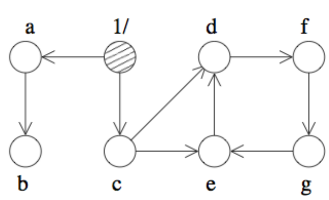
  Proof idea: Observe that, at any point in time, the vertices in the queue have distances that differ by at most 1. Let's say x comes out first from the queue; at this time y must be already marked (because otherwise (x, y) would be a tree edge). Furthermore y has to be in the queue, because, if it wasn't, it means it was already deleted from the queue and we assumed x was first. So y has to be in the queue, and we have $|d(y) - d(x)| \leq 1$ by above observation.

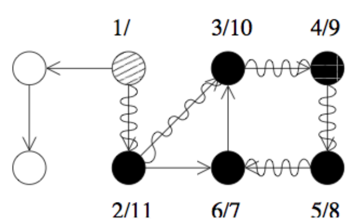
Depth-first search (DFS)

- Use stack instead of queue to hold discovered vertices:
  - We go "as deep as possible", go back until we find first unexplored adjacent vertex
- Useful to compute "start time" and "finish time" of vertex u
  - *Start time* d[u]: time when a vertex is first visited.
  - *Finish time* f[u]: time when all adjacent vertices of u have been visited.
- We can write DFS iteratively using the same algorithm as for BFS but with a STACK instead of a QUEUE, or, we can write a recursive DFS procedure

```
DFS(u)
    color[u] = gray
    d[u] = time
    time = time + 1
    FOR each v ∈ adj[u] DO
        IF color[v] = white THEN
            parent[v] = u
            DFS(v)
    color[u] = black
    f[u] = time
    time = time + 1
```

- Example:





DFS Properties:

- DFS(u) reaches all vertices reachable from u. On undirected graphs, DFS(u) visits all vertices in CC(u), and the DFS-tree obtained is a spanning tree of G.
- Analysis: DFS(s) runs in $O(|V_c| + |E_c|)$, where V_c, E_c are the number of vertices and edges in CC(s) (reachable from s, for directed graphs). When run on the entire graph, DFS(G) runs in $O(|V| + |E|)$ time. Put differently, DFS runs in linear time in the size of the graph.
- As with BFS $(v, parent[v])$ forms a tree, the *DFS-tree*
- Nesting of descendants: If u is descendent of v in DFS-tree then $d[v] < d[u] < f[u] < f[v]$.