11. Graphs

Dr. Swati Agarwal

Agenda

1. Graph Search
2. Graph Traversal
3. Depth First Search
4. Breadth First Search

Graph Search

- One of the most fundamental tasks on graphs is searching a graph by starting at some source vertex, or set of vertices, and visiting new vertices by crossing (out) edges until there is nothing left to search.
- We visit every vertex that is reachable from the source exactly once. This will require recording what vertices have already been visited so they are not visited a second time.
- Graph searching can be used to determine various properties of graphs:
  - whether the graph is connected or whether it is bipartite
  - whether a vertex u is reachable from v
  - finding the shortest path between vertices u and v

Graph Traversal

- Problem: Search for a certain node or traverse all nodes in the graph.
- Depth First Search: Once a possible path is found, continue the search until the end of the path
- Breadth First Search: Start several paths at a time, and advance in each one step at a time

Depth First Search

- A depth first search (DFS) in a graph G is like wandering in a labyrinth with a string and a can of paint without getting lost.
- We start at vertex s, tying the end of our string to the point and painting s "visited". Next we label s as our current vertex called u.
- Now we travel along an arbitrary edge (u, v). If this edge leads to an already visited vertex, we return to u.
- If vertex *v* is unvisited, we unroll our string and move to v, paint *v* "visited", set *v* as our current vertex, and repeat the previous steps.
- The process terminates when our backtracking leads us back to the start index s, and there are no more unexplored edges incident on s.

Algorithm DFS(v,G)

Input: A vertex v in a graph G
Output: A labeling of the edges as "discovery" edges and "back" edges
mark v as a visited vertex
for each edge e incident on v **do**
    if edge e is unexplored **then** let w be the other endpoint of e
    if vertex w is unexplored **then** label e as a discovery edge
        recursively call DFS(w)
    else label e as a backedge

Algorithm DFS(v,G)

Input: A vertex v in a graph G
Output: A labeling of the edges as "discovery" edges and "back" edges
mark v as a visited vertex
for each edge e incident on v **do**
    if edge e is unexplored **then** let w be the other endpoint of e
    if vertex w is unexplored **then** label e as a discovery edge
        recursively call DFS(w)
    else label e as a backedge

Property

DFS(v,G) visits all the vertices and edges in the connected component of v

Analysis of DFS

- Setting/getting a vertex/edge label takes O(1) time
- Each vertex is labeled twice: once as UNEXPLORED and once as VISITED
- Each edge is labeled twice – once as UNEXPLORED and once as DISCOVERY or BACK
- Method incidentEdges is called once for each vertex
- DFS runs in O(n + m) time provided the graph is represented by the adjacency list structure

Breadth First Search

- Instead of going as far as possible, BFS tries to search all paths.
- BFS makes use of a queue to store visited (but not dead) vertices, expanding the path from the earliest visited vertices
- The starting vertex s has level 0
- In the first round, the string is unrolled the length of one edge, and all of the nodes that are only one edge away from the anchor are visited. These edges are placed into level 1
- In the second round, all the new edges that can be reached by unrolling the string 2 edges are visited and placed in level 2.
- This continues until every vertex has been assigned a level.
- The label of any vertex v corresponds to the length of the shortest path from s to v.

Algorithm BFS(s,G)

Input: A vertex s in a graph G
Output: A labeling of the edges as "discovery" edges and "cross" edges
initialize container L0 to contain vertex s
$i \leftarrow 0$
while L_i is not empty **do**
    create container L_{i+1} to initially be empty
    for each vertex v in L_i **do**
        if edge e incident on v **do**
            let w be the other endpoint of e
            if vertex w is unexplored **then**
                label e as a discovery edge
                insert w into L_{i+1}
            else label e as a cross edge

Analysis of BFS

- Setting/getting a vertex/edge label takes O(1) time
- Each vertex is labeled twice: once as UNEXPLORED, once as VISITED
- Each edge is labeled twice: once as UNEXPLORED, once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence L_i
- Method incidentEdges is called once for each vertex

Analysis of BFS

- Setting/getting a vertex/edge label takes O(1) time
- Each vertex is labeled twice: once as UNEXPLORED, once as VISITED
- Each edge is labeled twice: once as UNEXPLORED, once as DISCOVERY or CROSS
- Each vertex is inserted once into a sequence L_i
- Method incidentEdges is called once for each vertex
- BFS runs in O(n + m) time provided the graph is represented by the adjacency list structure

Applications

We can specialize the BFS or DFS traversal of a graph G to solve the following problems in $O(n + m)$ time

- Compute the connected components of G
- Find a simple cycle in G, or report that G is a forest
- Given two vertices of G, and a path in G between them with the minimum number of edges, or report that no such path exists