CS Lunch
Reports on Summer Activities from Computer Science Students - REUs and Internships

Wednesday, 12:15 to 1:00
Alyx Burns ’17
Tricia Chaffey ’18
Sarah Robinson ’17
Moe Phyu Pwint ’18
Ngoc Vu ’17

Grace Hopper???
October 19 & 21
If going, let me know on your index card

O() ?

BFS (G) {
    R = \{s\}
    layer0 = \{s\}
    i = 1
    while layeri is not empty {
        for each node u in layeri-1 {
            while there is an edge (u,v) such that v \notin R{
                add v to layeri and to R
            }
        }
        i = i + 1
    }
}

DFS(u) {
    mark u as "explored"
    add u to R
    for each edge (u,v) incident on u {
        if (v is not marked as "explored") {
            DFS(v);
        }
    }
}
Representing Graphs: Adjacency Matrix

Adjacency matrix. An n-by-n matrix with $A_{uv} = 1$ if (u, v) is an edge.
- Two representations of each edge.
- How much memory?
- How long does it take to find out if a specific edge exists?
- How long does it take to find all edges incident on a node?

Adjacency list. Node indexed array of lists.
- Two representations of each edge.
- How much memory?
- How long to find a specific edge?
- How long to find all edges incident on a node?

Order in which we add edges?

BFS (G) {
    R = {s}
    layer0 = {s}
    i = 1
    while layeri is not empty {
        for each node u in layeri-1 {
            while there is an edge (u,v) such that v $\notin$ R{
                add v to layer and to R
            }
        }
        i = i + 1
    }
}

DFS(u) {
    mark u as "explored"
    add u to R
    for each edge (u,v) incident on u {
        if (v is not marked as "explored") {
            DFS(v);
        }
    }
}
Implementing BFS

R = {s}
layer_0 = {s}
i = 1
for each node u in layer_{i-1} {
    while there is an edge (u,v) such that v \notin R{
        add v to layer_i and to R
    }
    i = i + 1
}

What functionality do we need?
How should we represent the graph?
How should we keep track of the nodes we need to visit?

Implementing DFS

DFS(u) {
    mark u as "explored"
    add u to R
    for each edge (u,v) incident on u {
        if (v is not marked as "explored") {
            DFS(v);
        }
    }
}

What functionality do we need?
How should we represent the graph?
How should we keep track of the nodes we need to visit?

Summary

- Adjacency lists generally use less memory and are faster than adjacency matrices
- BFS and DFS differ only in the order in which nodes are visited and have cost \(O(m+n)\)
Party Problem

- You want to throw a party at which there are no pairs of guests that do not get along.
- You want to invite as many guests as possible.
- How would you solve this?

The party problem

- Represent each guest as a node.
- Draw an edge between guests who do not get along.
- Find the largest set of nodes where there is no edge between any pair of nodes in the set.

Bipartite Graphs

A bipartite graph is an undirected graph $G = (V, E)$ in which the nodes can be colored red or blue such that every edge has one red and one blue end.

- is a bipartite graph
- is NOT a bipartite graph
A bipartite solution
A bipartite solution

Who should you invite?

BFS and Bipartite Graphs

Lemma. Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).
Lemma. Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).