Week 9 Part1 Discussion

Sandra Batista

1.1-1.2

BFS Correctness

```
BFS(G,u)
1  for each vertex v
2      pred[v] = nil, d[v] = inf.
3  Q = new Queue
4  Q.enqueue(u),  d[u]=0
5  while Q is not empty
6     v = Q.front(); Q.dequeue()
7     foreach neighbor, w, of v:
8        if pred[w] == nil // w not found
9           Q.enqueue(w)
10          pred[w] = v,  d[w] = d[v] + 1
```

Define

- dist(s,v) = correct shortest distance
- d[v] = BFS computed distance
- p[v] = predecessor of v

Loop invariant

- All vertices with p[v] != nil (i.e. already in the queue or popped from queue) have d[v] = dist(s,v)
- The distance of the nodes in the queue are sorted

  If $Q = \{v_1, v_2, \ldots, v_r\}$ then $d[v_1] <= d[v_2] <= \ldots <= d[v_r]$

- The nodes in the queue are from 2 adjacent layers/levels

  i.e. $d[v_k] <= d[v_1] + 1$

  Suppose there is a node from a 3^{rd} level ($d[v_1] + 2$), it must have been found by some, vi, where $d[v_i] = d[v_1]+1$

Breadth-First Search

Analyze the run time of BFS for a graph with n vertices and m edges

- Find T(n)

How many times does loop on line 5 iterate?

- N times (one iteration per vertex)

How many times loop on line 7 iterate?

- For each vertex, v, the loop executes deg(v) times
- $= \sum_{v \in V} \theta[1 + \deg(v)]$
- $= \theta(\sum_{v} 1) + \theta(\sum_{v} \deg(v))$
- = Θ (n) + Θ (m)

Total = Θ(n+m)

```
BFS(G,u)
1  for each vertex v
2      pred[v] = nil, d[v] = inf.
3  Q = new Queue
4  Q.enqueue(u),  d[u]=0
5  while Q is not empty
6     v = Q.front(); Q.dequeue()
7     foreach neighbor, w, of v:
8        if pred[w] == nil // w not found
9           Q.enqueue(w)
10          pred[w] = v,  d[w] = d[v] + 1
```

DFS Analysis

1. What is the loop invariant of DFS?

2. The stack maintains all found vertices and only all nodes that need to be visited are reachable from a node on the stack. If a node v is on the stack there exists a path from start node to v. If a node is marked visited or gray, then its neighbors have been found and pushed on the stack.

3. What is the runtime of DFS?

```
DFS-Visit (G, start_node)
1    for each vertex u
2      u.color = WHITE
3      u.pred = nil
4    st = new Stack
5    st.push_back(start_node)
6    while st not empty
7       u = st.top(); st.pop()
8       if u.color == WHITE
9          u.color = GRAY
10         foreach vertex v in Adj(u) do
11            st.push_back(v)
```

Dijkstra's Run-time Analysis

What is the run-time of Dijkstra's algorithm?

While loop on line 8:

- At most once for each vertex since only removed from PQ when it is shortest distance to source
- **Each call to remove_min() costs log(V)** [at most V items in PQ]

For loop on line 10:

- Once for each of v's neighbors, deg(v)
- **Each call to decreaseKey() is log(V)**

```
1.    SSSP(G, s)
2.      PQ = empty PQ
3.      s.dist = 0;  s.pred = NULL
4.      PQ.insert(s)
5.      For all v in vertices
6.          if v != s then v.dist = inf;
7.          PQ.insert(v)
8.      while PQ is not empty
9.          v = min(); PQ.remove_min()
10.         for u in neighbors(v)
11.             w = weight(v,u)
12.             if(v.dist + w < u.dist)
13.                 u.pred = v
14.                 u.dist = v.dist + w;
15.                 PQ.decreaseKey(u, u.dist)
```

Total runtime =

$$\sum_{v \in V} [\log(V) + \sum_{i=1}^{\deg(v)} \log(V)] = O(|V| \log(V) + |E| \log(V))$$

$$= O((|V| + |E|) \log(V))$$

Dijkstra's Algorithm Analysis

What is the loop invariant?

- The vertex v removed from PQ is guaranteed to be the vertex with shortest path to source of all vertices in PQ and its distance is set to its shortest distance to the source.
- All vertices whose distances and predecessors are set have shortest known distance thus far to source.

Proof sketch by induction

- First node from PQ is source itself with distance 0 to itself.
- Decrease the distance to its neighbors; its neighbor with the shortest distance will be at front of PQ
- No shorter path from source to vertex at front of PQ; any other path would use some edge from the start having greater distance