CSE101 Quiz1 Review

This review doc summarizes the essential algorithms covered in lectures. It follows the section order to textbook to organize everything. Created by M. and Yilin, feel free to collaborate.

Topics1

- Max bandwidth path
  - Path
    - Simple path
      - No two edges are the same
    - **A single vertex is a trivial path to itself**
  - Objective
    - Over all possible paths p between v and u, find max BW(p)
- Graph reachability
  - Given a graph **G** and starting vertex **s**, give all reachable vertices v from s
    - **X:** a set of explored vertices
    - **F:** a set of reached but not explored vertices
    - **U:** a set of unreached vertices
  - procedure GraphSearch (G: directed graph, s: vertex)
    -
    - `Initialize X = empty, F = {s}, U = V – F.`
    - `While F is not empty:`
      - `Pick v in F.`
      - `For each neighbor u of v:`
        - `If u is not in X or F:`
          - `move u from U to F.`
      - `Move v from F to X.`
    - `Return X.`
  - Time analysis: **O(|V| + |E|)**

- **chapter 3.1,3.2**

- How big is ur graph
  - |E|
    - As small as |V|, if smaller, then the graph degenerates - **sparse**
    - As large as $|V|^2$, all possible connections - **dense**
  - Adjacency matrix vs adjacency list
    - Matrix always takes $O(|V|^2)$ space, so if the graph is **sparse**, that would be wasteful.
    - List always takes O(|E|)
- DFS in **undirected graphs**
  - **procedure** explore(G, v):
    - visited(v);

        - previsited(v):
        - for each edge (v, u) ∈ E:
            - If not visited(u): explore(G, u)
        - postvisit(v);

    - **procedure** dfs(G):
        - for all v ∈ V:
            - visited(v) = false;
        - for all v ∈ V:
            - If not visited(v): explore(v)
    - Two steps when considering the runtime
        - Mark each spot as visited - **O(|V|)**
        - A loop in which scanned all edges - **O(|E|)**
- **Connectivity**
    - Connected components:
        - Subgraph internally connected but has edges to remaining graph
        - Each time DFS called explore, one connected component is picked out
- Previsit and povisit ordering
    - Define a counter **clock, initialized as 1**
        - **procedure** previsit(v):
            - Pre[v] = clock;
            - Clock++;
        - **procedure** postvisit(v):
            - Post[v] = clock;
            - Clock++;
    - **Property:**
        - For any nodes u and v, the two intervals [pre[u], post[u]] and [pre[v], post[v]] are either disjoint or contained within the other.

- chapter 3.3,3.4

- **Types of edges**
    - Forward edges
        - Lead to a non-child descendant
        - If u is an ancestor of v, **$[_u$, $[_v$, $]_v$, $]_u$**
    - Back edges
        - Lead to ancestor
        - **$[_v$, $[_u$, $]_u$, $]_v$**
    - Cross edges
        - Lead to a node that has been explored.
        - **$[_v$, $]_v$, $[_u$, $]_u$**
- Directed acyclic graphs
    - **Property**
        - DFS reveals a back edge **iff** this G has a cycle

- Every edges leads to a vertex with lower post number in a **DAG**
  - Sink: smallest post number
  - Source: highest post number
- **Every DAG has at least one source and at least one sink**
  - Linearization:
    - Find a **source**, output it and delete from G
    - Repeat until the graph is empty

- **Strongly connected components (SCCs)**
  - Two nodes u and v are connected in a graph if there is a path from u to v and a path from v to u.
  - Property
    - Every directed graph is a DAG of its SCCs.
- Decomposition of SCCs
  - How to locate the **sink**
    - The SCC of highest post number must be a **source** SCC
    - We only need to reverse the whole graph, the **source SCC** of G' will be the **sink SCC** of the original graph.
  - How to continue once the first **sink** is discovered
    - Run DFS on G^R (step 1)
    - Run the directed connected components algorithm on G, and during DFS, process the vertices in decreasing order of their post numbers from step 1

- chapter 4.1,4.2,4.3

- Distances
  - The distance between two vertices is the length of the shortest path between them

- BFS
  - **procedure** bfs(G, s):
    - for all u ∈ V:
      - dist(v) = infinite
    - dist(s) = 0;
    - Q = [s] (queue containing only s)
    - while Q is not empty:
      - u = eject(Q)
      - for all edges (u, v) ∈ E:
        - If dist(v) = infinite:
          - inject(Q, v)
          - dist(v) = dist(u) + 1
- Lengths on edges

- chapter 4.4,4.5

- **Dijkstra's algorithm**
  - All edges have positive length
  - Using priority queue
    - Insert
      - Add a new element to the set
    - Decrease key
      - Decrease the value of certain key
    - Delete min
      - Return the element with the smallest key,
      - remove it from the set
    - Make heap
      - Build a priority queue out of given elements
- Priority Queue Implementations
  - Array
    - M

- chapter 4.5,5.1