Shortest Paths

Weighted Directed Graph – Weight on edges for distance

Variants of Shortest Path Problem

1. Single source shortest paths
   a. Finds all shortest path of vertices reachable from single source vertex

2. Single destination shortest-path
   a. By reversing direction of each edge in the graph, we can reduce this problem to single-source problem

3. Single-pair shortest-path
   a. No algorithm for this problem are known that run faster than best single-source algorithm in worst case.

4. All-pairs shortest-path

Relaxation - For each vertex v in V, we maintain an attribute d[v], which is upper bound on weight of shortest path from source s to v.

- Dijkstra’s Algorithm – no negative edge
- Bellman-Ford Algorithm – Running time O(VE)
  o Order in which edges are processed affects how quickly it converges
  o Correctness: show d[v] = δ(s,v) after |V|-1 passes
  o Prove: after |V| -1 passes, all d values correct

Properties of Shortest Paths

- Triangle Inequality
  o For any edge(u,v) in E, δ(s,v) <= δ(s,v)+w(u,v)
- Upper bound property
  o D[v] >= δ(s,v)
- Monotonic property
  o D[v] never increase
- No-path property
  o If v is not reachable then d[v] = δ(s,v) = infty
- Convergence property
- Path-relaxation property
- Predecessor-subgraph property
### DFS, DAG and SCC

- **Directed Acyclic Graph (DAG)** – directed graph with no directed cycle
  - Theorem: a directed graph G is acyclic iff a DFS of G yields no back edges

- **Topological Sort – of DAG**
  - Linear ordering of all vertices in graph G such that vertex u comes before vertex v if edge(u,v) in G
    - Example: scheduling dependent graph, find feasible course plan for university
  - Topological-Sort()
    1. Call DFS to compute finish time f[v] for each vertex
    2. As each vertex is finished, insert into onto front of linked list
    3. Return the linked list of vertices
    - Running Time: O(V+E)

- **Strongly Connected Directed Graph**
  - Every pairs of vertices are reachable from each other
  - Strongly-Connected – Graph G is strongly connected if, for every u and v in V, there is some path from u to v and some path from v to u
  - Strongly Connected Components – strongly connected component of graph is maximal subset of nodes that is strongly connected. Node share a strongly connected component if they are inter-reachable.
  - Transpose of G = (V,E): G transpose = (V, E transpose), where E transpose = \{(u,v): (v,u) in E\}
    - If G is DAG then G transpose is also D
  - Finding SCC
    - Input: directed graph G = (V,E)
    - Output: partition of V into disjoint sets so that each set defines SCC of G
    - Algorithm
      1. Call DFS(G) to compute finishing time f[u] for each vertex u.
      2. Compute G transpose
      3. Call DFS(G transpose), but in the main loop of DFS, consider vertices in order of decreasing f[u]
      4. Output the vertices of each tree in depth-first forest of step 3 as separate SCC.
    - Running time: SCC(G) = \( \Theta (E+V) \)