Shortest Paths

Chapter 24

Read sections 24.1 – 24.3

Ignore theorems, corollaries, lemmas and proofs

<u>but know the results</u> discussed in class

Shortest Paths Problem

Given a weighted directed graph g=(V,E), with each edge having real-valued weights, the weight of a path from node u to node v is the sum of the weights of its edges. The shortest path from u to v is the path with the minimum path weight (if no such path exists, its weight is ∞.

Single Source Shortest Paths Problem

Given a single start node s, find shortest paths from s to all other nodes.

Single Source Shortest Paths

- The breadth-first search problem finds the shortest paths from the start node to all other nodes.
- This is the same as finding the shortests paths if all edge weights are 1.
- Thinking Assignment: Will the BFS algorithm find the shortest paths if all edges had the same weight c?

Single Source Shortest Paths

An algorithm fo finding SSSPs can also solve the following problems:

1. Single Destination Shortest Paths
2. Single Pair Shortest Path
3. All Pairs Shortest Paths

Optimal Substructure Property

Lemma 24.1

If $p=<v_0,\ldots,v_k>$ is a shortest path from v_0 to v_k in graph $G=(V,E)$, then, for any i and j such that $0 \le i \le j \le k$, let p_{ij} be the subpath of p from v_i to v_j. Then pij is the shortest path from v_i to v_j.

Proof: Trivial!

Negative weight edges

- Edges with negative weights pose no problem.
- Shortest path weight may be negative in this case.
- But if there is a negative weight cycle that is reachable from the start node s, it is a problem! Why?
- In this case the shortest path is not defined and shortest path weight is $-\infty$

Positive or zero weight cycles

- Thus, a shortest path cannot contain a negative weight cycle.
- Can a shortest path contain any positive or zero weight cycles. Why or why not?

Shortest Paths

- Thus we can conclude that shortest paths must be simple paths (a simple path is one that contains no cycles).
- Any acyclic path in a graph G with n nodes and m edges can only contain at most n nodes and n-1 edges. Why?

Shortest Path Algorithms

- Use the attribute predecessor or previous (π) attached to nodes.
- Use the attribute distance (d) attached to nodes.
- Computes the Shortest-paths tree (like breadth-first trees computed by BFS).
- Thinking assignment: is a shortest path tree the same as a minimum spanning tree? why or why not?

Shortest-Paths Tree

If G=(V,E) is a weighted directed graph and s is the start node in G, and if there are no negative-weight cycles reachable from s, the shortest-paths tree is a directed subgraph G' of G, G'=(V',E') where $V' \subseteq V \text{ } and \text{ } E' \subseteq E$, such that:

1. V' is the set of nodes reachable from s
2. G' is a tree with root s
3. for all v∈V', the path from s to v in G' is the shortest path from s to v in G.

Relaxation

INITIALIZE-SINGLE-SOURCE (G,s) Complexity Θ(n)

```
1 for each node v in G.V
2     v.d = ∞
3     v.π = nil
4 s.d = 0
```

v.d = upper bound on the weight of a shortest path from s to v

v.π = previous node on the shortest path from s to v

G is the adjacency list representation; Each node in the adjacency list of a vertex contains the edge weight too, so it can be read in constant time.

Relaxing an edge (u,v): testing if the currently known shortest path from s to v can be improved by going through the currently known shortest path from s to u and then from u to v along the edge (u,v). This is the only way in which a current estimate of a shortest path can change.

RELAX (u,v,w) Complexity Θ(1)

```
1 if v.d>u.d+w(u,v)
2     v.d=u.d+w(u,v)
3     v.π = u
```

Properties of Relaxation

Triangle inequality: for any edge (u,v) in G, shortest-path-cost(s,v)≤ shortest-path-cost(s,u)+w(u,v).

Upper-bound property: for all vertices v in G, we always have v.d≥ shortest-path-cost(s,v), and once v.d= shortest-path-cost(s,v) through mutliple relaxations, its value will never change.

No-path property: If there is no path from s to v, then v.d= shortest-path-cost(s,v)= ∞

Shortest Path Algorithms

All algorithms go through the initialization step and then relax graph edges repeatedly. They differ in the number of times and order in which edges are relaxed.

1. Bellman-Ford
   - each edge is relaxed exactly |V|-1=n-1 times
   - works on graphs with negative weight edges
   - is able to detect negative weight cycles
   - complexity O(mn)
2. Shortest Paths in Directed Acyclic Graphs
   - each edge is relaxed exactly once
   - works only on acyclic graphs
   - linear algorithm: Θ(m+n)
3. Dijkstra’s Algorithm
   - each edge is relaxed exactly once
   - edge weights must be nonnegative
   - complexity $O(n^2)$

Bellman-Ford Algorithm

Returns a Boolean indicating whether there is a negative weight cycle reachable from source node s. If not, it determines shortest paths from s to all other vertices.

```
BELLMAN-FORD(G,w,s)
1 INITIALIZE-SINGLE-SOURCE(G,s)
2 for i=1 to n-1
3         for each edge (u,v) in G.E
4                 RELAX (u,v,w)
5 for each edge (u,v) in G.E
6         if v.d>u.d+w(u,v)
7                 return false
8 return true
```

Thinking Assignment: Which step checks for the existence of a negative weight cycle? How is it detected? Hint: Triangle inequality

DAG Algorithm

By relaxing the edges of a weighted directed acyclic graph according to the topological order of its nodes, shortest paths can be computed in Θ(m+n) time. Why?

```
DAG-SHORTEST_PATHS(G,w,s)
1 topologically sort vertices of G
2 INITIALIZE-SINGLE-SOURCE(G,s)
3 for each vertex u taken in topological order
4         for each vertex v in G.AdjacencyList[u]
5             RELAX(u,v,w)
```

Thinking Assignment: Understand how to modify this algorithm to find the longest paths from the discussion on p.657

Dijkstra’s Algorithm

Edge weights must not be negative. Maintains a set S of vertices whose shortest paths from s have already been determined. Repeatedly selects a vertex u from V-S with a minimum shortest-path weight estimate u.d (use a min priority queue based on the d attribute), and relaxes all edges leaving u.

```
DIJKSTRA(G,w,s)
1 INITIALIZE-SINGLE-SOURCE(G,s)
2 S=empty set
3 Build min priority queue Q with nodes in G.V based on d
4 while Q≠empty
5           u=EXTRACT-MIN(Q)
6           S=S U {u}
7           for each vertex v in G.AdjacencyList[u]
8               RELAX(u,v,w) {substitute step 2 with a DECREASE-KEY operation}
```

Thinking Assignment: Understand the informal complexity analysis of this algorithms on p. 661-662. Ask me next class if you have a question. The discussion should be understandable even though we did not discuss Fibonacci Heaps.

Thinking Assignments

- Why/how does Dijkstra’s algorithm fail if there are negative cost edges?
- Why does the algorithm work correctly if all edge costs are positive? I.e., why is it certain that when a node is removed from the priority Q by the algorithm, the shortest path to it has already been found?
- Create such a graph and run the algorithm on that graph to be able to answer this.