Recitation 12

Bellman-Ford

In lecture, we presented a version of Bellman-Ford[1] based on graph duplication and DAG Relaxation that solves SSSPs in $O(|V||E|)$ time and space, and can return a negative-weight cycle reachable on a path from s to v, for any vertex v with $\delta(s, v) = -\infty$.

The original Bellman-Ford algorithm is easier to state but is a little less powerful. It solves SSSPs in the same time using only $O(|V|)$ space, but only detects whether a negative-weight cycle exists (will not return such a negative weight cycle). It is based on the relaxation framework discussed in R11. The algorithm is straight-forward: initialize distance estimates, and then relax every edge in the graph in $|V|-1$ rounds. The claim is that: if the graph does not contain negative-weight cycles, $d(s, v) = \delta(s, v)$ for all $v \in V$ at termination; otherwise if any edge still relaxable (i.e., still violates the triangle inequality), the graph contains a negative weight cycle. A Python implementation of the Bellman-Ford algorithm is given below.

```
1  def bellman_ford(Adj, w, s):          # Adj: adjacency list, w: weights, s: start
2      # initialization
3      infinity = float('inf')           # number greater than sum of all + weights
4      d =  [infinity for _ in Adj]      # shortest path estimates d(s, v)
5      parent = [None for _ in Adj]      # initialize parent pointers
6      d[s], parent[s] = 0, s            # initialize source
7      # construct shortest paths in rounds
8      V = len(Adj)                      # number of vertices
9      for k in range(V - 1):            # relax all edges in (V - 1) rounds
10         for u in range(V):            # loop over all edges (u, v)
11             for v in Adj[u]:          # relax edge from u to v
12                 try_to_relax(Adj, w, d, parent, u, v)
13     # check for negative weight cycles accessible from s
14     for u in range(V):                # Loop over all edges (u, v)
15         for v in Adj[u]:
16             if d[v] > d[u] + w(u,v):  # If edge relax-able, report cycle
17                 raise Exception('Ack! There is a negative weight cycle!')
18     return d, parent
```

This algorithm has the same overall structure as the general relaxation paradigm, but limits the order in which edges can be processed. In particular, the algorithm relaxes every edge of the graph (lines 10-12), in a series of $|V| - 1$ rounds (line 9). The following lemma establishes correctness of the algorithm.

[1]This algorithm is called **Bellman-Ford** after two researchers who independently proposed the same algorithm in different contexts.

Lemma 1 *At the end of relaxation round i of Bellman-Ford, $d(s, v) = \delta(s, v)$ for any vertex v that has a shortest path from s to v which traverses at most i edges.*

Proof. Proof by induction on round i. At the start of the algorithm (at end of round 0), the only vertex with shortest path from s traversing at most 0 edges is vertex s, and Bellman-Ford correctly sets $d(s, s) = 0 = \delta(s, s)$. Now suppose the claim is true at the end of round $i - 1$. Let v be a vertex containing a shortest path from s traversing at most i edges. If v has a shortest path from s traversing at most $i-1$ edges, $d(s, v) = \delta(s, v)$ prior to round i, and will continue to hold at the end of round i by the upper-bound property[2] Alternatively, $d(s, v) \neq \delta(s, v)$ prior to round i, and let u be the second to last vertex visited along some shortest path from s to v which traverses exactly i edges. Some shortest path from s to u traverses at most $i - 1$ edges, so $d(s, u) = \delta(s, u)$ prior to round i. Then after the edge from u to v is relaxed during round i, $d(s, v) = \delta(s, v)$ as desired. □

If the graph does not contain negative weight cycles, some shortest path is simple, and contains at most $|V|-1$ edges as it traverses any vertex of the graph at most once. Thus after $|V|-1$ rounds of Bellman-Ford, $d(s, v) = \delta(s, v)$ for every vertex with a simple shortest path from s to v. However, if after $|V| - 1$ rounds of relaxation, some edge (u, v) still violates the triangle inequality (lines 14-17), then there exists a path from s to v using $|V|$ edges which has lower weight than all paths using fewer edges. Such a path cannot be simple, so it must contain a negative weight cycle.

This algorithm runs $|V|$ rounds, where each round performs a constant amount of work for each edge in the graph, so Bellman-Ford runs in $O(|V||E|)$ time. Note that lines 10-11 actually take $O(|V| + |E|)$ time to loop over the entire adjacency list structure, even for vertices adjacent to no edge. If the graph contains isolated vertices that are not S, we can just remove them from Adj to ensure that $|V| = O(|E|)$. Note that if edges are processed in a topological sort order with respect to a shortest path tree from s, then Bellman-Ford will correctly compute shortest paths from s after its first round; of course, it is not easy to find such an order. However, for many graphs, significant savings can be obtained by stopping Bellman-Ford after any round for which no edge relaxation is modifying.

Note that this algorithm is different than the one presented in lecture in two important ways:

- The original Bellman-Ford only keeps track of one 'layer' of $d(s, v)$ estimates in each round, while the lecture version keeps track of $d_k(s, v)$ for $k \in \{0, \ldots, |V|\}$, which can be then used to construct negative-weight cycles.
- A distance estimate $d(s, v)$ in round k of original Bellman-Ford does not necessarily equal $d_k(s, v)$, the k-edge distance to v computed in the lecture version. This is because the original Bellman-Ford may relax multiple edges along a shortest path to v in a single round, while the lecture version relaxes at most one in each level. In other words, distance estimate $d(s, v)$ in round k of original Bellman-Ford is never larger than $d_k(s, v)$, but it may be much smaller and converge to a solution quicker than the lecture version, so may be faster in practice.

[2]Recall that the Safety Lemma from Recitation 11 ensures that relaxation maintains $\delta(s, v) \leq d(s, v)$ for all v.

Exercise: Alice, Bob, and Casey are best friends who live in different corners of a rural school district. During the summer, they decide to meet every Saturday at some intersection in the district to play tee-ball. Each child will bike to the meeting location from their home along dirt roads. Each dirt road between road intersections has a level of **fun** associated with biking along it in a certain direction, depending on the incline and quality of the road, the number of animals passed, etc. Road fun-ness may be positive, but could also be be negative, e.g. when a road is difficult to traverse in a given direction, or passes by a scary dog, etc. The children would like to: choose a road intersection to meet and play tee-ball that maximizes the total fun of all three children in **reaching** their chosen meeting location; or alternatively, abandon tee-ball altogether in favor of biking, if a loop of roads exists in their district along which they can bike all day with ever increasing fun. Help the children organize their Saturday routine by finding a tee-ball location, or determining that there exists a continuously fun bike loop in their district (for now, you do not have to find such a loop). You may assume that each child can reach any road in the district by bike.

Solution: Construct a graph on road intersections within the district, as well as the locations a, b, and c of the homes of the three children, with a directed edge from one vertex to another if there is a road between them traversable in that direction by bike, weighted by negative fun-ness of the road. If a negative weight cycle exists in this graph, such a cycle would represent a continuously fun bike loop. To check for the existence of any negative weight cycle in the graph, run Bellman-Ford from vertex a. If Bellman-Ford detects a negative weight cycle by finding an edge (u, v) that can be relaxed in round $|V|$, return that a continuously fun bike loop exists. Alternatively, if no negative weight cycle exists, minimal weighted paths correspond to bike routes that maximize fun. Running Bellman-Ford from vertex a then computes shortest paths $d(s, v)$ from a to each vertex v in the graph. Run Bellman-Ford two more times, once from vertex b and once from vertex c, computing shortest paths values $d(b, v)$ and $d(c, v)$ respectively for each vertex v in the graph. Then for each vertex v, compute the sum $d(a, v) + d(b, v) + d(c, v)$. A vertex that minimizes this sum will correspond to a road intersection that maximizes total fun of all three children in reaching it. This algorithm runs Bellman-Ford three times and then compares a constant sized sum at each vertex, so this algorithm runs in $O(|V||E|)$ time.

6.006 Introduction to Algorithms

Spring 2020