1 Weighted Graphs and Representations

So far we have only considered unweighted graphs. Today we will consider the shortest paths problem in weighted graphs, so let us define them.

A weighted graph is a graph $G = (V, E)$ equipped with a weight function $w : E \to \mathbb{Z}$. Sometimes the weights can be arbitrary real numbers, but in this class we will assume that they are integers.

Expanding on the representations of unweighted graphs, there are two standard representations of weighted graphs:

1. **Adjacency lists:** Just like with unweighted graphs, a weighted graph $G = (V, E)$ with weights w is represented with a dictionary/array Adj indexed by the vertices in V, where $Adj[v]$ now is a list of pairs $(u, w(v, u))$ where u is a neighbor of v (out-neighbor for directed graphs) and $w(v, u)$ is the weight of the edge from v to u. Similarly to the adjacency lists for unweighted graphs, most operations are quite efficient, except for checking whether (v, u) is an edge. However, just like with unweighted adjacency lists, one can augment the weighted adjacency lists with a hash table containing the graph's edges, so that checking if (v, u) is an edge can be performed in expected constant time.

2. **Generalized adjacency matrix:** Here we have a $|V| \times |V|$ matrix A whose rows and columns are indexed by the vertices, and now $A[u, u] = 0$, $A[u, v] = w(u, v)$ if $(u, v) \in E$, and otherwise $A[u, v] = \infty$ if $(u, v) \notin E$ and $u \neq v$. This adjacency matrix representation has similar advantages and disadvantages to the unweighted adjacency matrix.

2 Weighted Shortest Paths

Given a path $s = v_0, v_1, \ldots, v_{k-1}, v_k = t$, its weight is $\sum_{i=0}^{k-1} w(v_i, v_{i+1})$.

A shortest s-t path is a path from s to t with minimum weight among all s-t paths in the graph, and the distance $d(s, t)$ is the weight of a shortest s-t path, and ∞ if s cannot reach t at all.

This notion of shortest path and distance is, however, not well-defined in the presence of a negative weight cycle, a path beginning and ending at the same vertex, so that the weight of the path is < 0. Suppose that C is a negative weight cycle and a vertex s can reach C via a path P_1 and t can be reached from C via some path P_2. Then, one can obtain successively shorter s-t paths starting with P_1, going around C more and more times and ending with P_2. The more times one goes around the cycle C, the shorter the path gets as the weight of C is negative and we keep subtracting more and more. Thus, in this situation, we say that the distance from s to t is $-\infty$. In fact, our shortest paths algorithms will try to detect whether a negative cycle exists and throw an exception if it does, saying that the distances are undefined.

On the other hand, if there is no negative weight cycle in the graph, then the notion of distance is well-defined. In particular, in this case for every s and t in the graph where s can reach t, there is a *simple* shortest path, i.e. one that does not go through any vertex more than once. To see this, suppose that P is a shortest s-t path that goes through a vertex x more than once:

$$P = s \to v_1 \to \ldots \to v_j \to x \to \ldots \to x \to \ldots v_r \to t.$$

Let P_1 be the path from s to the first occurrence of x on P, let C be the cycle from the first occurrence to the last occurrence of x on P, and let P_2 be the path from the last occurrence of x on P to t. Then $P = P_1 \odot C \odot P_2$ where $\odot$ is concatenation.

Since there are no negative cycles, the weight $w(C)$ of C is ≥ 0. Thus, removing C from C yields a path $P_1 \odot P_2$ from s to t that goes through x only once and has weight $d(s, t) - w(C) \leq d(s, t)$, and hence this

path must also be a shortest s-t path. Doing this to all vertices that appear more than once yields a simple shortest path.

As shortest paths are now without loss of generality simple, their number of edges is bounded by $n-1$ and the distance notion is well-defined.

3 Single Source Shortest Paths (SSSP)

Two lectures ago we used BFS to solve the SSSP problem in unweighted graphs. Today we revisit SSSP in weighted graphs.

Single Source Shortest Paths (SSSP) in weighted graphs: Given a directed or undirected $G = (V, E)$ weighted by $w : E \to \mathbb{R}$, and a source vertex $s \in V$, either detect a negative weight cycle (and throw an exception) or compute for every vertex v, the distance $d(s, v)$.

Just like with unweighted graphs we might want to get the shortest paths besides the distances. As shortest paths are without loss of generality simple (and hence on at most $n-1$ edges each), if we wanted to, we could return n simple shortest paths, but then just the output would be quadratic.

In BFS, when we considered unweighted SSSP, we returned predecessors, $\pi(v)$ for each v, so that $\pi(v)$ was the node before v on an s to v shortest path. This defined a shortest paths tree rooted at v and was a compressed representation of the shortest paths. Can we do something like this here?

Why did it work before? Well, because if $s = v_1, v_2, \ldots, v_k = t$ is a shortest path, then any subpath $v_i, \ldots, v_j$ is also a shortest path. This property is still true for weighted paths when there are no negative weight cycles. (Otherwise, if $v_i, \ldots, v_j$ were not a shortest path, we could replace it with a v_i-v_j shortest path and get a shorter s-t path, a contradiction.)

Because of this, we can still output a consistent $\pi(v)$ for each v and there is a shortest paths tree rooted at s. Let's see why: Start with s. Then process the vertices t in nondecreasing order of the min number of edges of a shortest s-t path. Assume so far we had a tree of vertices containing all vertices v with shortest s-v paths on at most j edges (and maybe some with $j+1$ edges). We want to add another vertex t with a $j+1$-edge shortest path P_{st} to the tree. Say y is the node before t on P_{st}. As the s-y subpath of P_{st} is a shortest s-y path on at most j edges, y is in the current tree, so we can append t to it with edge (y, t) and the s-t path in the tree will be a shortest s-t path. We can also have $\pi(t)$ be y.

This isn't really an algorithm since we don't actually know how many edges a shortest s-t path can have for each t, but it's getting close.

The way we do weighted shortest paths algorithms is that we maintain an estimate $d'[s, t]$ of the distance $d(s, t)$ for each t so that at all times $d'[s, t] \geq d(s, t)$. Initially all estimates are ∞, and for SSSP from s we also set $d'[s, s] = 0$. This is actually $d(s, s)$ as long as there is no negative cycle through s.

During the algorithms we scan edges (u, v) and **"relax"** them. This corresponds to setting

$$d'[s, v] = \min\{d'[s, v], d'[s, u] + w(u, v)\}.$$

Relaxation maintains the **invariant** that $d'[s, v] \geq d(s, v)$. This is due to the *triangle inequality*: for every u, v, w in a graph, $d(u, w) + d(w, v) \geq d(u, v)$.

Using the triangle inequality and assuming that before relaxing (u, v) we had $d'[s, v] \geq d(s, v)$ and $d'[s, u] \geq d(s, u)$, we get that after relaxation, (1) either $d'[s, v]$ stayed the same, in which case we still have $d'[s, v] \geq d(s, v)$, or

$$d'[s, v] = d'[s, u] + w(u, v) \geq d(s, u) + w(u, v) \geq d(s, u) + d(u, v) \geq d(u, v).$$

Here, the first inequality follows since we assumed that $d'[s, u] \geq d(s, u)$, the second inequality follows since (u, v) is a potential $u - v$ path of weight $w(u, v)$ so $w(u, v) \geq d(u, v)$ and the last inequality is the triangle inequality.

In fact, due to the nature of edge relaxations, $d'[s, v]$ is always the weight of some s-v path (you can prove this by induction).

Relaxations can reduce our distance estimate. We can thus propose the following algorithm:

BrainlessRelax(G): "While there is an edge (u, v) whose relaxation will reduce $d'[s, v]$, relax (u, v)."

If there are negative cycles reachable from s, this algorithm will never finish. If there are no negative cycles, it might still take a long time. Even if we run it on a DAG. (Notably DAGs have no cycles so definitely no negative cycles.) There is a nasty example in the recitation notes.

DAGs however are very special, as we saw last time. So we can actually fix the above algorithm.

Suppose that the algorithm does terminate. Let's show that then for every $v \in V$, $d'[s, v] = d(s, v)$, and hence the algorithm will have solved SSSP. The simplest way to prove this is by contradiction. Suppose that there is some v for which $d'[s, v] > d(s, v)$. Consider a shortest s-v path $s = v_1, v_2, \ldots, v_j = v$. Let v_ℓ be the first on this path for which $d'[s, v_\ell] = d(s, v_\ell)$ but $d'[s, v_{\ell+1}] > d(s, v_{\ell+1})$. Such a vertex must exist since $d'[s, s] = d(s, s) = 0$ but $d'[s, v] > d(s, v)$. But then, relaxing $(v_\ell, v_{\ell+1})$ would reduce $d'[s, v_{\ell+1}]$, as

$$d'[s, v_{\ell+1}] > d(s, v_{\ell+1}) = d(s, v_\ell) + w(v_\ell, v_{\ell+1}) = d'[s, v_\ell] + w(v_\ell, v_{\ell+1}).$$

We get a contradiction.

Thus, we have

Lemma 3.1 (Termination Lemma)**.** *If G has no negative cycles reachable from s, and there is no edge whose relaxation would improve a distance estimate, then for every v, $d'[s, v] = d(s, v)$.*

4 SSSP in weighted DAGs

Let $G = (V, E)$ be a given DAG. First compute the topological order. s will be at the front. (If not, remove all vertices in front of it and set their distances to ∞. Let our distance estimates be $d[v]$, i.e. $d[v]$ corresponds to $d'[s, v]$. We set $d[v] = \infty$ for all v. Then set $d[s] = 0$. Now, process the vertices in the topological order (can use a list, pop from the front). Let u be the next vertex to be processed (originally s). We go through all neighbors v of u, and relax (u, v): $d[v] = \min\{d[v], d[u] + w(u, v)\}$. Then process the next vertex in the topological order etc until all vertices are processed.

The running time is linear: $O(m + n)$ to find the topological order via DFS and $O(m + n)$ to go through the order and relax the m edges.

Let us prove the correctness now. It will be by induction. Let the topological order is $v_1, v_2, \ldots, v_n$.

The inductive hypothesis for $i = 1, \ldots, n$ is that when v_i is popped from the topological order (and before its edges are relaxed), for every $j \leq i$, $d[v_j] = d(s, v_j)$. The base case is $i = 1$, when $v_1 = s$, we have $d[s] = 0 = d(s, s)$.

Now suppose that the inductive hypothesis is true for $i - 1$. Let's show it is true for i. We only need to show that by the time v_i is popped, $d[v_i]$ will be set to $d(s, v_i)$. The rest of the distance estimates for v_k with $k < i$ cannot change, as any distance estimate is the weight of some path and can only decrease via relaxations; since $d[v_k]$ has already reached its minimum value (the weight of a shortest path to v_k), it cannot decrease anymore and will stay unchanged.

Let P be a shortest path from s to v_i. Let v_k be the vertex right before v_i on P. Since G is a DAG, $k < i$, i.e. v_k must be before v_i in the topological order as $(v_k, v_i) \in E$. Thus v_k was popped at some point before v_i, and by the inductive hypothesis, at that point $d[v_k] = d(s, v_k)$. Then all edges out of v_k were relaxed, and in particular so was (v_k, v_i) and then $d[v_i]$ was set to the min of $d[v_i]$ and $d[v_k] + w(v_k, v_i) = d(s, v_k)$. As $d[v_i]$ is the weight of some s-v_i path, $d[v_i] \geq d(s, v_i)$, and so $d[v_i]$ was set to $d(s, v_i)$, completing the induction.