CSE 373

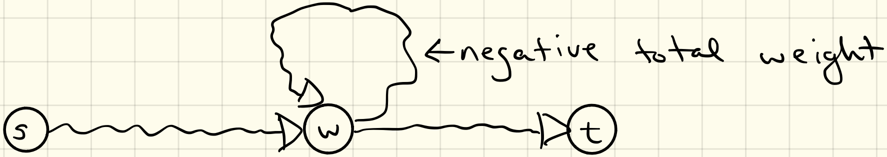
Lecture 11

Returning to the Shortest Path problem:

Shortest path with negative weights:

Given a directed graph G with weighted edges $d(u,v)$ that may be positive, 0, or negative, find the shortest path from s to t.

Complication: Negative weight cycles - If some cycle has a negative cost, we can make the length of the s-t path as small as we want!

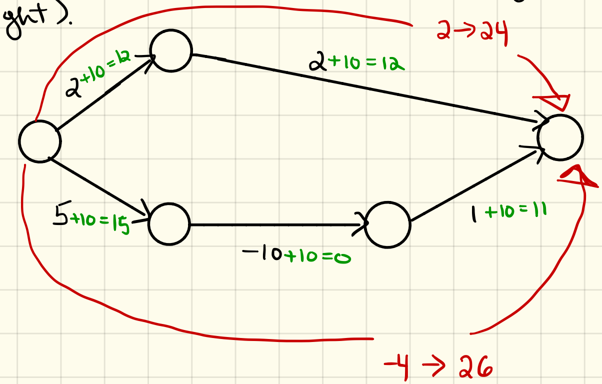


go from s to w, then traverse the cycle as much as we want (never stop). Assume no negative weight cycles.

- The negative edge weights <u>break</u> the greedy decision rule that is used by Dijkstra's algorithm; why?
  - the shortest s-t path no longer uses the shortest s-t' subpath for every t' between s and t

How do we fix this?

Idea: Just make all weights non-negative (i.e. add a big number to each edge weight).



this doesn't work because the cost becomes $\alpha \cdot \text{length}(P) + \text{cost}(P)$

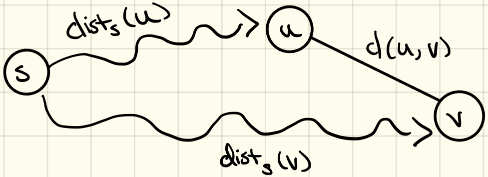
↑ adjustment factor

- that is, if paths are long in terms of # of hops, adjustment factor will dominate

Bellman-Ford

Let $\text{dist}_s(v)$ be the current estimated distance from s to v. At the start $\text{dist}_s(v) = \infty \ \forall \ v \neq s$.

Ford step: Find an edge (u,v) such that $\text{dist}_s(u) + d(u,v) < \text{dist}_s(v)$ and set $\text{dist}_s(v) = \text{dist}_s(u) + d(u,v)$



Theorem: After applying the Ford step until

$$dist_s(u) + d(u,v) \geq dist_s(v)$$

for <u>all</u> edges, $dist_s(v)$ will equal the shortest path distance from s to u for all u.

Proof: Show that for every v: (1) There is a path of length $dist_s(v)$ and (2) No path is shorter $\rightarrow$ so $dist_s(v)$ must be the shortest path length.

Lemma 1: After any number i of applications of the Ford step, either $dist_s(v) = \infty$ or there is an $s-v$ path of length $dist_s(v)$.

Proof: Let v be a vertex such that $dist_s(v) < \infty$. Proceed by induction on i

BC: $i=0$, only $dist_s(s) = 0 < \infty$, and there is a path of length 0 from s to s

IH: assume true for all $j < i$

IS: Let $dist_s(v)$ be the distance updated during the i^{th} application. It is updated using edge (u,v) with the rule $dist_s(v) = dist_s(u) + d(u,v)$. $dist_s(u)$ must be $< \infty$ and must have been updated via the Ford step at some iteration $j < i$. Therefore, by <u>IH</u>, there is a path P_{su} of length $dist_s(u)$. Now, on the i^{th} application $P_{su} + (u,v)$ is a path of length $dist_s(u) + d(u,v) = dist_s(v)$ $\blacksquare$

Lemma 2: Let P_{sv} be any path from s to v. When the Ford step can no longer be applied, $\text{length}(P_{sv}) \geq \text{dist}_s(v)$ for all paths P_{sv}.

Proof: By induction on # of edges in P_{sv}.

BC: $|P_{sv}| = 1$, it is a single edge (s,v) and because the Ford step can't be applied, $d(s,v) \geq \text{dist}_s(v)$.

IH: Assume true for P_{su} of k or fewer edges (strong induction)

IS: Let P_{sv} be an s-v path of k+1 edges. $P_{sv} = P_{su} + (u,v)$ for <u>some</u> u.

$$\text{length}(P_{sv}) = \text{length}(P_{su}) + d(u,v) \geq \text{dist}_s(u) + d(u,v) \geq \text{dist}_s(v)$$

otherwise, the Ford step could be applied. ∎

So, which edges are candidates for the Ford step?

those where $\text{dist}_s(u) + d(u,v) < \text{dist}_s(v)$

This can only become true if $\text{dist}_s(u)$ has become smaller since last we checked.

- whenever we change $\text{dist}_s(u)$ add u to a queue
- To try and apply the Ford step, take a node from the queue and try to apply the rule to all of its edges.

Implementation:

```
ShortestPath(G, s, t):
    dist[u] = ∞ ∀ u ; dist[s] = 0
    queue = [s] ; parent = {}
    while queue not empty:
        v = queue.front() ; queue.pop()
        for w ∈ neighbors(v):
            if dist[v] + d(v,w) < dist[w]:
                dist[w] = dist[v] + d(v,w)
                parent[w] = v
                if w ∉ queue: queue.append(w)
    return dist, parent
```

Question:

How is Bellman-Ford dynamic programming?

Running time:

n = # nodes

m = # edges

- After $dist_s(v)$ has been updated k times, it corresponds to a simple path of k edges.
- A path can be of length at most $n-1$ and still be simple

So, each dist[w] can be updated at most $n-1$ times.

Updating all vertices takes $O(m)$ time, since we look at each edge twice.

Total running time = $O(mn)$

Note: Slower than Dijkstra's in general.

How is BF dynamic programming?

Def: $dist_s(v,i)$ is the length of the minimum cost path from s to v using <u>at most</u> i edges.

Define $dist_s(v,i)$ recursively as

$$dist_s(v,i) = \begin{cases} dist_s(v, i-1) & \text{if the best s-v path uses at most i-1 edges} \\ dist_s(w, i-1) + d(w,v) & \text{if the best s-v path uses i edges and (w,v) is the last edge.} \end{cases}$$

Len $N(w)$ be the neighbors of w.

we can also write our recurrence as

$$dist_s(v,i) = \min \begin{cases} dist_s(v, i-1) \\ \min\limits_{w \in N(v)} dist_s(w, i-1) + d(w,v) \end{cases}$$

Base case: $dist_s(v,1) = d(s,v)$ or ∞ if $(s,v) \notin E$

Goal: Compute $dist_s(t, n-1)$

Important facts about the recurrence:

- $dist_s(v, x)$ depends only on $dist_s(w, y)$ for y wich is smaller than x
- There are only $|V| \times (|V|-1)$ possible arguments for $dist_s(\cdot,\cdot)$

\# of hops (max length of path)

	s	b	c	d	x	t	w	v
8	0							
7	0							
6	0							
5	0							
4	0							
3	0							
2	0							
1	0	∞	3	∞	6	∞	-2	∞

vertex dest

cell depends on cells of neighbors in the previous row

can fill in this matrix from the bottom up.

BellmanFord (G = (V,E), s, t):

```
dist_s[x, 1] = d(s,x) for all x ∈ V

for i = 1, ..., |V| - 1
    for v ∈ V:
        best_w = None
        for w in N(v):
            best_w = min(best_w, dist_s[w, i-1] + d(w,v))

        dist_s[v, i] = min(best_w, dist_s[v, i-1])

return dist_s[t, n-1]
```

Running time of the DP:

<u>Simple Analysis</u>

- $O(n^2)$ subproblems
- $O(n)$ time / subproblem
- $O(n^3)$ time

<u>Better Analysis</u>

- let n_v be # edges entering v
- filling each entry takes $O(n_v)$ time
- Total time is:

$$O\Big(n \cdot \sum_{v \in V} n_v\Big) = O(nm)$$