6.006 Lecture 17: More Efficient Algorithms for special cases.

- Taking advantage of absence of negative edges: Dijksta's algorithm
- Taking advantage of absence of cycles
- CLRS 24.2, 24.3

The trouble with Bellman-Ford

$\Theta(VE)$ running time

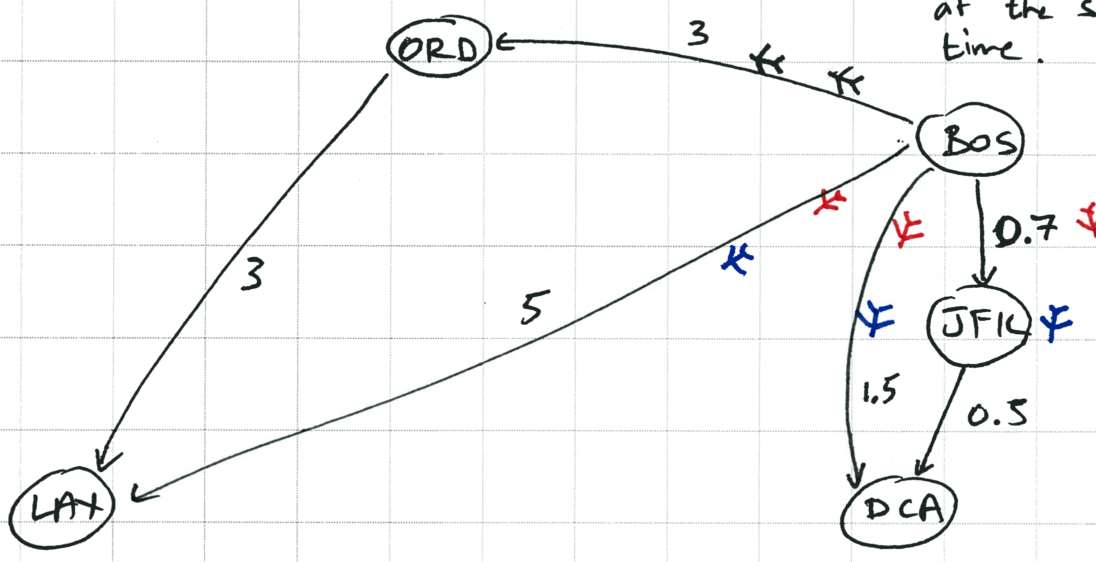
More specifically: relaxing (or ~~trying~~ trying to) every edge $|V|-1$ times.

The key to more efficient algorithms: relax each edge at most once. Can be done if we delay relaxing (u,v) until we are sure $d[u] = \delta(s,v)$.

Also in this lecture: building up. We'll be using two algorithmic building blocks that we studied earlier.

Flight times from Logan

we start by sending a plane in each direction at the same time.



a plane that left BOS 0.3 hours ago

a plane that left BOS 0.7 hours ago

Since there are no negative flight time, when our first plane lands (at JFK), we know that the total flight time to JFK is 0.7. Any indirect route will take longer since even the first leg takes > 0.7.

When all the planes land, we relax:

$d[lax]=5$, $d[ORD]=3$, $d[DCA]=1.5$, $\delta[JFK]=d[JFK]=0.7$

Now move all the planes to JFK and repeat.

Effectively, we now relax the edges going out of JFK. Since we do this ~~whe~~ only when $d[JFK] = \delta(Bos, JFK)$, $d[JFK]$ cannot go down any more, so we will not need to relax this edge ever again.

The result of $d[v] = d[u] + w(u,v)$ can change only ~~w~~ if $d[u]$ changes

Algorithm

```
d = [float('inf') for i in range(n)]
d[s] = 0
Q = d
do n-1 times:          for i in range(n)
    u = min_index(Q)   not built into Python but self explanatory
    for (u,v) in E:
        if d[v] > d[u] + w(u,v):
            d[u] = q[u] = d[v] + w(u,v)
    q.pop(u)
```

Running time:

Initialization $= \Theta(V)$ notation for $|V|$

Loop: $(|V| - 1) \times (\underbrace{\Theta(V)}_{\text{min. index}} + \underbrace{\Theta(V)}_{\text{pop}})$

$+$

$\Theta(E)$ every edge relaxed at most once

Total: $\Theta(V^2)$

Better than Bellman-Ford but still slow.

Dijkstra's Algorithm

The key is a more efficient data structure for Q.

Abstract operations:

- Initialize Q with a given set of values
- Extract-Min
- Decrease-Key (in $q[u] = d[v] + w(u,v)$)

$\Rightarrow$ Let's use a heap

```
d=[float('inf') for i in range(n)]
d[s]=0
Q = Build-Heap(d)
for i in range(n-1):
    u = Extract-Min(Q)      get the index, not the value!
    for (u,v) in E:
        if d[v] > d[u]+w(u,v):
            d[v] = d[u]+w(u,v)
            Decrease-Key(Q, v, d[v])
                                need to maintain a
                                mapping from vertices to
                                locations in the heap.
```

Running time

Initialization: $\Theta(V)$ including Build-Heap

Extract-Mins: $\Theta(V \cdot \lg V)$ # calls, cost of each call

Relaxations: $\Theta(E \cdot 1)$

Decrease-Keys: $\Theta(E \cdot \lg V)$ maybe less

Total: $\Theta((V+E)\lg V)$ much better than Bellman Ford

Can we do better? Yes, with Fibonacci heaps, in which Decrease-Key costs only $\Theta(1)$ amortized. Ch. 20 in text but not coverd in 6.006

Correctness of Dijkstra:

Q: set of vertices in priority queue

$S = V \setminus Q$

Claim: (1) for $u \in S$, $d[u] = \delta(s,u)$

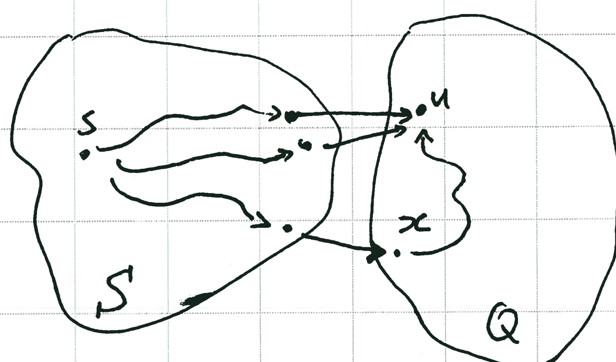
(2) for $v \in Q$, $d[v] = \min\limits_{u \in S} \{\delta(s,u) + w(u,v)\}$

If claim is true, Dijkstra is correct because of (1).

Proof of claim: we need to show that (1) holds

for $u = \text{Extract-Min}(Q)$, & using induction.

Suppose not:



If the path through x is shorter,

the subpath from x to u must be

negative: contradiction.

DAG's: another way to relax every edge only once

Can we order the vertices so that all the vertices that appear in all the paths from s to v are ordered before v?

If G is a DAG (contains no cycles), then YES.

```
Q = Topological-Sort(V,E,s)
d = [float('inf') for i in range(n)]
d[s] = 0
for i in range(n):
    u = Q[i]
    for (u,v) in E
        if d[v] > d[u] + w(u,v): d[v] = d[u] + w(u,v)
```

Running time: $\Theta(V+E)$

even better than Dijkstra.