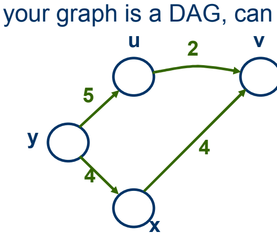
Tirgul 13

(and more)

sample questions

(back to previous week)

Q. We've seen that solving the shortest paths problem requires O(VE) time using the Belman-Ford algorithm or O((V+E)logV) using the Dijkstra algorithm (might be even O(VlogV+E) when using Fibonacci heaps). Suppose you know that your graph is a DAG, can you do better?



A. Yep, the algorithm is:

```
Dag-Shortest-Paths(G,w,s)
Θ(|V+E|)   1. topologically sort the vertices of G
Θ(|V|)     2. Initialize-Single-Source(G,s)
           3. For each vertex u (taken in topologically    (Θ(|V|))
              sorted order)
Θ(|E|)     4.   For each vertex v ∈ Adj[u]
           5.       Relax(u,v,w)
```

This procedure runs in Θ(V+E)

Proof outline

Triangle Inequality:
For all $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$.

Lower-bound Property
Always have $d[v] \ge \delta(s, v)$ for all v. Once $d[v] = \delta(s, v)$, it never changes.

No-path Property
If $\delta(s, v)=\infty$, then $d[v]=\infty$ always.

Convergence Property
If $s \rightsquigarrow u \rightarrow v$ is a shortest path, $d[u] = \delta(s, u)$, and we call RELAX(u,v,w), then $d[v] = \delta(s, v)$ afterward.

Path Relaxation Property
Let $p = v0, v1, \ldots, vk$ be a shortest path from $s = v0$ to vk.
If we relax, *in order*, $(v0, v1), (v1, v2), \ldots, (vk\text{-}1, vk)$, even intermixed with other relaxations, then $d[v\,k] = \delta(s, v\,k)$.

Proof

Triangle Inequality:

For all $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$.

Proof:

Weight of shortest path $s \rightsquigarrow v$ is ≤ weight of any other path from s to v.

Path $s \rightsquigarrow u \rightarrow v$ is a path $s \rightsquigarrow v$, and if we use a shortest path $s \rightsquigarrow u$, its weight is $\delta(s, u) + w(u, v)$.

Proof

Lower-bound Property:

We always have $d[v] \ge \delta(s, v)$ for all v. Once $d[v] = \delta(s, v)$, it never changes.

Proof:

Initially true.

Suppose there exists a vertex such that $d[v] < \delta(s, v)$.

Without loss of generality, v is first vertex for which this happens. Let u be the vertex that causes $d[v]$ to change.

Then $d[v] = d[u] + w(u, v)$.

So, $d[v] < \delta(s, v) \le \delta(s, u) + w(u, v)$ (triangle inequality)

$\le d[u] + w(u, v)$ (v is first violation) $\Rightarrow d[v] < d[u] + w(u, v)$.

Contradicts $d[v] = d[u] + w(u, v)$.

Once $d[v]$ reaches $\delta(s, v)$, it never goes lower. It never goes up, since relaxations only lower shortest-path estimates.

Proof

No-path Property:

If $\delta(s, v)=\infty$, then $d[v]=\infty$ always.

Proof:

$d[v] \geq \delta(s, v)=\infty \Rightarrow d[v]=\infty$.

Proof

Convergence Property

If $s \rightsquigarrow u \rightarrow v$ is a shortest path, $d[u] = \delta(s, u)$, and we call RELAX(u,v,w), then $d[v] = \delta(s, v)$ afterward.

Proof

After relaxation:

$d[v] \leq d[u] + w(u, v)$ (RELAX code)

$= \delta(s, u) + w(u, v)$

$= \delta(s, v)$

Since $d[v] \geq \delta(s, v)$, must have $d[v] = \delta(s, v)$.

Proof

Path Relaxation Property

Let $p = v_0, v_1, \ldots, v_k$ be a shortest path from $s = v_0$ to v_k.

If we relax, *in order*, $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, even intermixed with other relaxations, then $d[v_k] = \delta(s, v_k)$.

Proof

Induction to show that $d[v_i] = \delta(s, v_i)$ after (v_{i-1}, v_i) is relaxed.

Basis: $i = 0$. Initially, $d[v_0] = 0 = \delta(s, v_0) = \delta(s, s)$.

Inductive step:

Assume $d[v_{i-1}] = \delta(s, v_{i-1})$.

Relax (v_{i-1}, v_i).

By convergence property, $d[v_i] = \delta(s, v_i)$ afterward and $d[v_i]$ never changes.

Hashing

Consider a hash table of size 13 with hash function *h*(key) = key mod 13. Insert following items, in the given order, into an initially empty table:

46, 11, 42, 29, 36, 22, 20, 3, 10

Draw the resulting table for each of the four collision resolution strategies:

1. *linear probing*
2. *quadratic probing*
3. *double hashing with h2(key) = 5 – (key mod 5)*
4. *chaining*

Hashing

Linear probing: $h(k,i) = (h'(k)+i) mod(m))$

46, 11, 42, 29, 36, 22, 20, 3, 10

			42	29	3		46	20	22	36	11	10
0	1	2	3	4	5	6	7	8	9	10	11	12

Suppose we have a hash table with m entries with a primary clustering problem, we have m/2 members in the table, all forming one primary cluster, what is the average complexity of inserting a new member?

Hashing

Quadratic probing: $h(k,i) = (h'(k)+c_1 i+c_2 i^2) mod(m))$

46, 11, 42, 29, 36, 22, 20, 3, 10

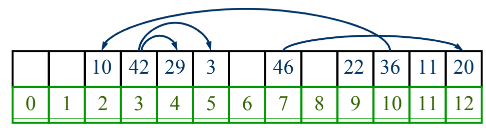
$(c_1 = c_2 = 1)$

20		3	42		29		46		22	36	11	10
0	1	2	3	4	5	6	7	8	9	10	11	12

Hashing

Double hashing with $h_2(key) = 5 - (key \bmod 5)$:

46, 11, 42, 29, 36, 22, 20, 3, 10



* Note that although the range of h_1 is $[0..m-1]$, the range of h_2 is $[1..m-1]$, why?