

Solutions -- Topic 12, Dynamic Programming, Class 03/07

Copyright (c) 2016 Dan Suthers. All rights reserved. These solution notes may only be used by students in ICS 311 Spring 2016 at the University of Hawaii.

Longest Simple Path in a Directed Acyclic Graph

Given a **directed weighted acyclic graph** $G=(V,E)$ and two vertices **s** (start) and **t** (target), develop a **dynamic programming approach** for finding a **longest weighted simple path** from **s** to **t**.

1. Characterize the Structure of an Optimal Solution

Let **p** be a **longest path from u to t**. If $u = t$ then **p** is simply $\langle u \rangle$ and has zero weight. Consider when $u \neq t$. Then **p** has at least two vertices and looks like: **p** = $\langle u, v \dots t \rangle$ (it is possible that $v = t$).

Let **p'** = $\langle v \dots t \rangle$ and **prove that p' must be a longest simple path from v to t**. (This is a proof of optimal substructure.)

Solution:

Suppose that **p'** as defined above were *not* a longest simple path from **v** to **t**.

Then there must exist some path **p''** that is a longer simple path from **v** to **t**; that is (extending our w notation), $w(p'') > w(p')$. We can construct a new path **p*** from **u** to **t** consisting of $u \rightarrow v \rightarrow p'' \rightarrow t$. This is a legal path because **G** is acyclic, so **u** cannot occur in **p''**. The length of this path **p*** is

$$w(p^*) = w(u,v) + w(p'') > w(u,v) + w(p') = w(p)$$

contradicting our definition of **p** as the longest simple path from **u** to **t**. Therefore, **p'** must be a longest simple path from **v** to **t**, and the problem exhibits optimal substructure.

2. Recursively define the value of an optimal solution:

Let **dist[u]** be the distance of a longest path from **u** to **t**. **Fill out the definition to reflect the above structure:**

$$\text{dist}[u] = \begin{array}{ll} 0 & \text{if } u = t \\ \max_{v \in \text{Adj}[u]} \{w(u,v) + \text{dist}[v]\} & \text{if } u \neq t \end{array}$$

3. Compute the value of an optimal solution (simple recursive version):

Write a recursive procedure that computes the value of an optimal solution as defined by the above recursive definition. Do not memoize yet; that's the next step.

Solution: (Notice how the code follows the mathematical definition.)

```
Longest-Path-Value-Recursive (G, u, t)
    if u = t
        return 0
    else
        max =  $-\infty$ 
        for each v in G.Adj[u]
            alt = w(u,v)
                + Longest-Path-Value-Recursive(G, v, t)
            if max < alt
                max = alt
        return max
```

4. Compute the value of an optimal solution (dynamic programming version):

Memoize your procedure by passing the array `dist[1..|V|]` that records longest path distances `dist[u]` from each vertex `u` to `t`. Assume that the **caller has initialized all entries of `dist` to $-\infty$.**

Solution (additions highlighted):

```
Longest-Path-Value-Memoized (G, u, t, dist)
    if u = t
        dist[u] = 0 // caller may expect all entries
defined
        return 0
    if dist[u] >  $-\infty$ 
        return dist[u] // this is the reuse of prior result
    else
        // dist[u] =  $-\infty$  is not necessary as caller did it
        for each v in G.Adj[u]
            alt = w(u,v)
                + Longest-Path-Value-Memoized(G, v, t, dist)
            if dist[u] < alt
                dist[u] = alt
        return dist[u]
```

5. Extra Credit: analyze the runtime of your solution in #4 in terms of $|V|$ and $|E|$

Include (a) the runtime to initialize `dist` and (b) the runtime of **Longest-Path-Value-Memoized** itself. This requires aggregating across loops.

(a) $\Theta(|V|)$ to fill in the entries.

(b) $\Theta(|E|)$ since in aggregate across all calls each edge is processed once, and all other operations are constant.

(c) Total: $\Theta(|V| + |E|)$.