EE360C: Algorithms

Dynamic Programming (4/4)

Summer 2019

Department of Electrical and Computer Engineering
University of Texas at Austin

Longest Common Subsequence

Longest Common Subsequence

This is a common problem in biological applications: find the longest common sequence of ACTG in two different pieces of DNA

- useful in finding similar proteins, etc.

Given a sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$, a sequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ is a **subsequence** of X if there exists a strictly increasing sequence $\langle i_1, i_2, \ldots, i_k \rangle$ of indices of X such that for all $j = 1, 2, \ldots, k$, $x_{i_j} = z_j$.

(For example $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$.)

Our goal is, given two sequences X and Y, find the longest common subsequence of the two sequences.

Characterizing the Longest Common Subsequence

Theorem: Optimal Substructure of an LCS

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. if $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}

Proof (Part I)

If $z_k \neq x_m$, we could append $x_m = y_n$ to Z to get a common subsequence with length $k + 1$, which contradicts the premise. So $z_k = x_m = y_n$. And therefore Z_{k-1}, the prefix of Z with z_k removed is a longest common subsequence of X_{m-1} and Y_{n-1}.

Proof (Parts II and III)

If $z_k \neq x_m$, then Z is a common subsequence of X_{m-1} and Y. If there were a common subsequence W of X_{m-1} and Y longer than k, then W would also be a subsequence of X and Y, which contradicts the premise that Z was an LCS.

A Recursive Solution to LCS

If $x_m = y_n$, there is one subproblem: find the LCS of X_{m-1} and of Y_{n-1}. If $x_m \neq y_n$, then there are two subproblems to solve: find the LCS of X_{m-1} and Y and the LCS of X and Y_{n-1}.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Computing the Length of an LCS

We use dynamic programming to compute solutions to the $\Theta(mn)$ subproblems in a bottom-up fashion

- we create a table $c[0\,..\,m, 0\,..\,n]$ and compute its entries in row major order (filling in the first row, left to right, then the second row, etc.)
- we keep a second table $b[0\,..\,m, 0\,..\,n]$ whose entries point to the table entry that corresponds to the best subproblem for the problem $b[i, j]$ (to help in reconstructing the optimal solution)

Computing the Length of an LCS (cont.)

```
LCS-LENGTH(X, Y)
m = length[X]
n = length[Y]
for i = 1 to m
        c[i, 0] = 0
for j = 1 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i = y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = ↖
                else if c[i − 1, j] ≥ c[i, j − 1]
                                c[i, j] = c[i − 1], j]
                                b[i, j] = ↑
                        else c[i, j] = c[i, j − 1]
                                b[i, j] = ←
return c and b
```

An Example

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

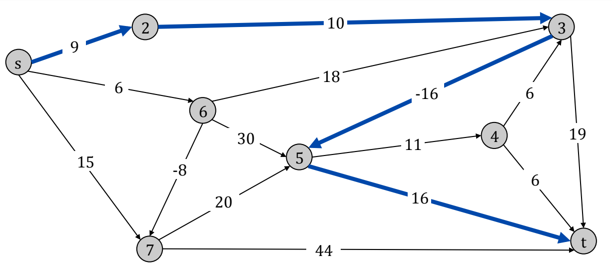
Shortest Paths

Shortest Path Problem

Given a direct graph $G = (V, E)$ with the edge weights c_{vw} (which can include negative edge weights), find the shortest path from node s to node t.

Example

Nodes represent agents in a financial setting and c_{vw} is the cost of a transaction in which we buy from agent v and sell immediately to w



Negative Cost Cycles

Observation

If some path from s to t contains a negative cost cycle, there does not exist a shortest s-t path; otherwise there exists a path that is a simple (i.e. does not repeat nodes) and hence has at most $n - 1$ edges.

Shortest Paths and Dynamic Programming

Definition: $OPT(i, v)$ is the length of the shortest v-t path P using at most i edges.

- Case 1: P uses at most $i - 1$ edges
    - $OPT(i, v) = OPT(i - 1, v)$
- Case 2: P uses exactly i edges
    - if (v, w) is the first edge, the OPT uses (v, w), and then selects the best w-t path using at most $i - 1$ edges

$$OPT(i, v) = \begin{cases} 0 & \text{if } i = 0 \\ \min(OPT(i-1, v), \min_{(v,w)\in E}(OPT(i-1, w) + c_{vw}))) & \text{otherwise} \end{cases}$$

Remark

If no negative cycles, then $OPT(n - 1, v) =$ length of shortest v-t path.

Implementation

```
Shortest-Path(G, t) {
   foreach node v ∈ V
      M[0, v] ← ∞
   M[0, t] ← 0

   for i = 1 to n-1
      foreach node v ∈ V
         M[i, v] ← M[i-1, v]
      foreach edge (v, w) ∈ E
         M[i, v] ← min { M[i, v], M[i-1, w] + c_vw }
}
```

Complexity

$O(mn)$ time, $O(n^2)$ space

Finding shortest path

Maintain a ”successor” for each table entry.

Distance Vector Routing

Communication Network

- nodes = routers
- edges = direct communication link
- cost of edge = delay on link (which is naturally non-negative, but we use Bellman-Ford algorithm anyway!)

Dijkstra's Algorithm

Requires global information of network

Bellman-Ford

Uses only local knowledge of neighboring nodes

Synchronization

We don't expect the routers to run in lock-step; Bellman-Ford can tolerate asynchronous routing updates and can be shown to still converge on the correct shortest path.

Distance Vector Protocol

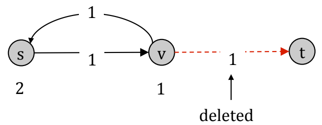
- Each router maintains a vector of shortest path lengths to every other node (distances) and the first hop on each path (directions)
- Algorithm: each router performs *n* separate computations, one for each potential destination node
- “Routing by rumor”

Examples

RIP, Xerox XNS RIP, Novell’s IPX RIP, Cisco’s IGRP, DEC’s DNA Phase IV, AppleTalk’s RTMP

Caveat

Edge costs may change during the algorithm; inconsistent router state can lead to the counting to infinity problem



Path Vector Protocols

Link State Routing

- Each router stores the entire path (not just the distance and next hop)
- Based on Dijkstra's algorithm
- Avoids the counting to infinity problem
- Requires significantly more storage

Examples

Border Gateway Protocol (BGP), Open Shortest Path First (OSPF)

Questions