CSE 417 Algorithms and Complexity

Richard Anderson
Lecture 21
DP and Shortest Paths
Longest Common Subsequence

- $C = c_1 \ldots c_g$ is a subsequence of $A = a_1 \ldots a_m$ if $C$ can be obtained by removing elements from $A$ (but retaining order).

- $\text{LCS}(A, B)$: A maximum length sequence that is a subsequence of both $A$ and $B$

$LCS(\text{BARTHOLEMEWSIMPSON, KRUSTYTHECLOWN}) = \text{RTHOWN}$
LCS Optimization

• $A = a_1 a_2 \ldots a_m$, $B = b_1 b_2 \ldots b_n$
• $\text{Opt}[j, k]$ is the length of $\text{LCS}(a_1 a_2 \ldots a_j, b_1 b_2 \ldots b_k)$
• Optimization Recurrence:
  – If $a_j = b_k$, $\text{Opt}[j, k] = 1 + \text{Opt}[j-1, k-1]$
  – If $a_j \neq b_k$, $\text{Opt}[j, k] = \max(\text{Opt}[j-1, k], \text{Opt}[j, k-1])$
Code to compute Opt[n, m]

for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
        if (A[i] == B[j])
            Opt[i,j] = Opt[i-1, j-1] + 1;
        else if (Opt[i-1, j] >= Opt[i, j-1])
            Opt[i, j] := Opt[i-1, j];
        else
            Opt[i, j] := Opt[i, j-1];
Storing the path information

A[1..m], B[1..n]

for i := 1 to m  Opt[i, 0] := 0;
for j := 1 to n  Opt[0,j] := 0;
Opt[0,0] := 0;

for i := 1 to m
  for j := 1 to n
    else if Opt[i-1, j] >= Opt[i, j-1]
      {  Opt[i, j] := Opt[i-1, j], Best[i,j] := Left;  }
    else  {  Opt[i, j] := Opt[i, j-1], Best[i,j] := Down;  }
Reconstructing Path from Distances
How good is this algorithm?

• Is it feasible to compute the LCS of two strings of length 300,000 on a standard desktop PC? Why or why not.
public int ComputeLCS() {
    int n = str1.Length;
    int m = str2.Length;

    int[,] opt = new int[n + 1, m + 1];
    for (int i = 0; i <= n; i++)
        opt[i, 0] = 0;
    for (int j = 0; j <= m; j++)
        opt[0, j] = 0;

    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (str1[i - 1] == str2[j - 1])
                opt[i, j] = opt[i - 1, j - 1] + 1;
            else if (opt[i - 1, j] >= opt[i, j - 1])
                opt[i, j] = opt[i - 1, j];
            else
                opt[i, j] = opt[i, j - 1];

    return opt[n, m];
}
N = 17000

Runtime should be about 5 seconds*

* Personal PC, 6 years old

```csharp
namespace LongestCommonSubsequence
{
    class LcsAlgorithm
    {
        int[] str1;
        int[] str2;

        int[,] opt;

        public LcsAlgorithm(int[] str1, int[] str2)
        {
            this.str1 = str1;
            this.str2 = str2;
        }

        public int ComputeLcs() {
            int n = str1.Length;
            int m = str2.Length;

            /* Adding an extra row and column to the array.
               This means the strings are indexed from 0 to n-1, m-1.
            */
            opt = new int[n + 1, m + 1];
            for (int i = 0; i <= n; i++)
                opt[i, 0] = 0;
            for (int j = 0; j <= m; j++)
                opt[0, j] = 0;

            for (int i = 1; i <= n; i++)
                for (int j = 1; j <= m; j++)
                    if (str1[i - 1] == str2[j - 1])
                        opt[i, j] = opt[i - 1, j - 1] + 1;
                    else
                        opt[i, j] = Math.Max(opt[i - 1, j], opt[i, j - 1]);
        }
    }
}
```
public int SpaceEfficientLCS() {
    int n = str1.Length;
    int m = str2.Length;
    int[] prevRow = new int[m + 1];
    int[] currRow = new int[m + 1];

    for (int j = 0; j <= m; j++)
        prevRow[j] = 0;

    for (int i = 1; i <= n; i++) {
        currRow[0] = 0;
        for (int j = 1; j <= m; j++) {
            if (str1[i - 1] == str2[j - 1])
                currRow[j] = prevRow[j - 1] + 1;
            else if (prevRow[j] >= currRow[j - 1])
                currRow[j] = prevRow[j];
            else
                currRow[j] = currRow[j - 1];
        }
        for (int j = 1; j <= m; j++)
            prevRow[j] = currRow[j];
    }

    return currRow[m];
}
N = 300000

N: 10000 Base 2 Length: 8096   Gamma:  0.8096   Runtime:00:00:01.86
N: 20000 Base 2 Length: 16231  Gamma:  0.81155  Runtime:00:00:07.45
N: 30000 Base 2 Length: 24317  Gamma:  0.8105667 Runtime:00:00:16.82
N: 40000 Base 2 Length: 32510  Gamma:  0.81275   Runtime:00:00:29.84
N: 50000 Base 2 Length: 40563  Gamma:  0.81126   Runtime:00:00:46.78
N: 60000 Base 2 Length: 48700  Gamma:  0.8116667 Runtime:00:01:08.06
N: 70000 Base 2 Length: 56824  Gamma:  0.8117715 Runtime:00:01:33.36

N: 300000 Base 2 Length: 243605  Gamma:  0.8120167  Runtime:00:28:07.32
Observations about the Algorithm

• The computation can be done in $O(m+n)$ space if we only need one column of the Opt values or Best Values

• The computation requires $O(nm)$ space if we store all of the string information
Computing LCS in $O(nm)$ time and $O(n+m)$ space

• Divide and conquer algorithm
• Recomputing values used to save space

• Section 6.7 of the text, but we will not have time to cover in detail (so you are not responsible for section 6.7)
Divide and Conquer Algorithm

• Where does the best path cross the middle column?

• For a fixed $i$, and for each $j$, compute the LCS that has $a_i$ matched with $b_j$
Algorithm Analysis

- $T(m,n) = T(m/2, j) + T(m/2, n-j) + cnm$
- Solution: $T(m,n) \leq 2cnm$
Prove by induction that
\[ T(m,n) \leq 2^{cn} \]

\[ T(m,n) = T(m/2, j) + T(m/2, n-j) + cnm \]
Shortest Paths with Dynamic Programming
Shortest Path Problem

• Dijkstra’s Single Source Shortest Paths Algorithm
  – $O(m \log n)$ time, positive cost edges
• Bellman-Ford Algorithm
  – $O(mn)$ time for graphs which can have negative cost edges
Lemma

• If a graph has no negative cost cycles, then the **shortest** paths are **simple** paths

• Shortest paths have at most n-1 edges
Shortest paths with a fixed number of edges

• Find the shortest path from s to w with exactly k edges
Express as a recurrence

• Compute distance from starting vertex s

• $\text{Opt}_k(w) = \min_x [\text{Opt}_{k-1}(x) + c_{xw}]$

• $\text{Opt}_0(w) = 0$ if $w = s$ and infinity otherwise
Algorithm, Version 1

for each $w$

\[
M[0, w] = \text{infinity};
\]

\[
M[0, s] = 0;
\]

for $i = 1$ to $n-1$

for each $w$

\[
M[i, w] = \min_x (M[i-1, x] + \text{cost}[x, w]);
\]
for each w
    \[ M[0, w] = \text{infinity}; \]
M[0, s] = 0;
for i = 1 to n-1
    for each w
        \[ M[i, w] = \min(M[i-1, w], \min_x(M[i-1, x] + \text{cost}[x, w])); \]
Algorithm, Version 3

for each \( w \)

\[ M[w] = \infty; \]

\[ M[s] = 0; \]

for \( i = 1 \) to \( n-1 \)

for each \( w \)

\[ M[w] = \min(M[w], \min_x (M[x] + \text{cost}[x,w])); \]
Example:
Correctness Proof for Algorithm 3

• Key lemma – at the end of iteration $i$, for all $w$, $M[w] \leq M[i, w]$;
Algorithm, Version 4

for each w
  
  $M[w] = \infty$;

$M[s] = 0$;

for i = 1 to n-1
  
  for each w
    
    for each x
      
      if ($M[w] > M[x] + \text{cost}[x,w]$)
        
        $P[w] = x$;

        $M[w] = M[x] + \text{cost}[x,w]$ ;
Theorem

If the pointer graph has a cycle, then the graph has a negative cost cycle.

Proof: See text.
Negative Cycles

• If the pointer graph has a cycle, then the graph has a negative cycle
• Therefore: if the graph has no negative cycles, then the pointer graph has no negative cycles
Finding negative cost cycles

What if you want to find negative cost cycles?
What about finding Longest Paths

• Can we just change Min to Max?
Foreign Exchange Arbitrage

<table>
<thead>
<tr>
<th></th>
<th>USD</th>
<th>EUR</th>
<th>CAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>----</td>
<td>0.8</td>
<td>1.2</td>
</tr>
<tr>
<td>EUR</td>
<td>1.2</td>
<td>----</td>
<td>1.6</td>
</tr>
<tr>
<td>CAD</td>
<td>0.8</td>
<td>0.6</td>
<td>----</td>
</tr>
</tbody>
</table>