CSE 417 Algorithms and Complexity

Richard Anderson

Lecture 21

DP and Shortest Paths

Longest Common Subsequence

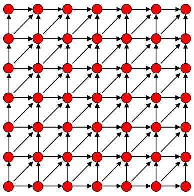
- $C=c_1\ldots c_g$ is a subsequence of $A=a_1\ldots a_m$ if C can be obtained by removing elements from A (but retaining order)
- LCS(A, B): A maximum length sequence that is a subsequence of both A and B

LCS(BARTHOLEMEWSIMPSON, KRUSTYTHECLOWN)
= RTHOWN

LCS Optimization

- $A = a_1a_2\ldots a_m$, $B = b_1b_2\ldots b_n$
- Opt[j, k] is the length of LCS($a_1a_2\ldots a_j$, $b_1b_2\ldots b_k$)
- Optimization Recurrence:
  - If $a_j = b_k$, Opt[j,k] = 1 + Opt[j-1, k-1]
  - If $a_j \mathrel{!}= b_k$, Opt[j,k] = max(Opt[j-1,k], Opt[j,k-1])

Dynamic Programming Computation

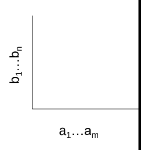


Code to compute Opt[n, m]

```
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
        if A[i] = B[j]  {  Opt[i,j] := 1 + Opt[i-1,j-1];  Best[i,j] := Diag; }
        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

```
LCS Arguments
 211031321102033212120000321302
 100222010121310130323233121011
_0 _0 _0 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1 _1
_1 _1 _1 _1 _1 _1 _1 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2 _2
_1 _1 _1 _1 _1 _1 _1 _2 _2 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3 _3
_1 _2 _2 _2 _2 _2 _2 _2 _3 _3 _3 _3 _3 _3 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4 _4
_1 _2 _2 _2 _2 _2 _2 _2 _3 _3 _3 _3 _4 _4 _4 _4 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5
_1 _2 _2 _2 _2 _2 _2 _3 _3 _4 _4 _4 _4 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _5 _6 _6 _6 _6 _6 _6
_1 _2 _2 _2 _2 _2 _2 _3 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6 _6
_1 _2 _2 _3 _3 _3 _3 _3 _3 _4 _5 _5 _5 _5 _5 _5 _6 _6 _6 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7
_1 _2 _2 _3 _3 _3 _3 _4 _4 _4 _5 _6 _6 _6 _6 _6 _6 _6 _6 _7 _7 _7 _7 _7 _8 _8 _8 _8 _8 _8
_1 _2 _2 _3 _3 _3 _3 _4 _4 _5 _5 _6 _6 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7 _7 _8 _8 _9 _9 _9 _9
_1 _2 _3 _3 _3 _3 _4 _4 _5 _5 _5 _6 _6 _7 _8 _8 _8 _8 _8 _8 _8 _8 _8 _8 _8 _8 _9 10 10 10
_1 _2 _3 _4 _4 _4 _4 _4 _5 _5 _6 _6 _6 _7 _8 _8 _8 _8 _8 _9 _9 _9 _9 _9 _9 _9 _9 10 10 10
_1 _2 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _6 _7 _8 _8 _8 _9 _9 _9 _9 _9 _9 _9 _9 _9 _9 10 10 10
_1 _2 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _7 _7 _8 _8 _9 _9 10 10 10 10 10 10 10 10 10 10 10 10
_1 _2 _3 _4 _4 _4 _5 _5 _5 _5 _6 _6 _7 _7 _8 _8 _9 _9 10 10 11 11 11 11 11 11 11 11 11 11
_1 _2 _3 _4 _5 _5 _5 _5 _5 _5 _6 _6 _7 _7 _8 _8 _9 _9 10 11 11 12 12 12 12 12 12 12 12 12
_1 _2 _3 _4 _5 _5 _5 _6 _6 _6 _6 _7 _7 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 13 13 13 13 13
_1 _2 _3 _4 _5 _6 _6 _6 _6 _6 _7 _7 _7 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 14 14 14 14 14
_1 _2 _3 _4 _5 _6 _6 _7 _7 _7 _7 _8 _8 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 14 15 15 15 15
_1 _2 _3 _4 _5 _6 _6 _7 _7 _7 _8 _8 _8 _8 _8 _9 _9 _9 10 11 11 12 12 12 13 14 15 15 15 15
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _8 _8 _9 _9 _9 10 10 11 11 12 12 12 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _8 _8 _9 _9 _9 _9 10 10 11 11 12 12 13 13 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _7 _8 _8 _9 _9 _9 _9 _9 _9 10 10 11 12 12 13 13 13 13 14 15 16 16 16
_1 _2 _3 _4 _5 _6 _7 _8 _8 _9 _9 10 10 10 10 10 10 10 11 12 12 13 13 13 14 14 15 16 17 17
_1 _2 _3 _4 _5 _6 _7 _8 _8 _9 _9 10 11 11 11 11 11 11 11 12 13 13 14 14 14 14 15 16 17 17
_1 _2 _3 _4 _5 _6 _7 _8 _9 _9 _9 10 11 11 12 12 12 12 12 12 13 13 14 14 14 14 15 16 17 17
_1 _2 _3 _4 _5 _6 _7 _8 _9 _9 10 10 11 11 12 12 12 12 12 13 13 14 14 14 14 15 15 16 17 17
```

How good is this algorithm?

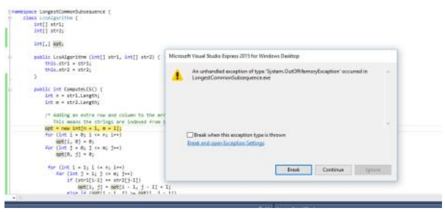
- Is it feasible to compute the LCS of two strings of length 300,000 on a standard desktop PC? Why or why not.

Implementation 1

```
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

Manufacturer: Dell
Model: Optiplex 990
Processor: Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz 3.10 GHz
Installed memory (RAM): 8.00 GB (7.88 GB usable)
System type: 64-bit Operating System, x64-based processor

Implementation 2

```
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

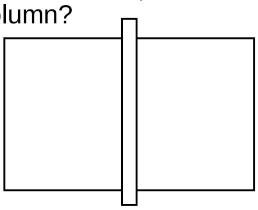
- The computation can be done in O(m+n) space if we only need one column of the Opt values or Best Values
- The computation requires O(nm) space if we store all of the string information

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

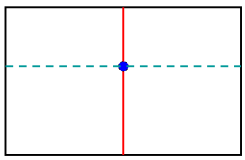
- Where does the best path cross the middle column?



- 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) <= 2cnm



Prove by induction that T(m,n) <= 2cmn

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(mlog 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
- $Opt_k(w) = \min_x [Opt_{k-1}(x) + c_{xw}]$
- $Opt_0(w) = 0$ if w = s and infinity otherwise

Algorithm, Version 1

```
for each w
   M[0, w] = infinity;
M[0, s] = 0;
for i = 1 to n-1
   for each w
      M[i, w] = min_x(M[i-1,x] + cost[x,w]);
```

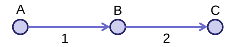
Algorithm, Version 2

```
for each w
   M[0, w] = 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] + cost[x,w]));
```

Algorithm, Version 3

```
for each w
   M[w] = infinity;
M[s] = 0;
for i = 1 to n-1
   for each w
      M[w] = min(M[w], min_x(M[x] + cost[x,w]));
```

Example:



Correctness Proof for Algorithm 3

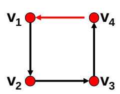
- Key lemma – at the end of iteration i, for all w, M[w] <= M[i, w];

Algorithm, Version 4

```
for each w
   M[w] = infinity;
M[s] = 0;
for i = 1 to n-1
   for each w
      for each x
         if (M[w] > M[x] + cost[x,w])
            P[w] = x;
            M[w] = M[x] + cost[x,w] ;
```

Theorem

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



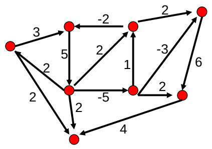
Proof: See text.

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

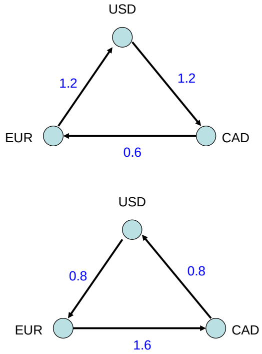
- What if you want to find negative cost cycles?



What about finding Longest Paths

- Can we just change Min to Max?

Foreign Exchange Arbitrage



	USD	EUR	CAD
USD	------	0.8	1.2
EUR	1.2	------	1.6
CAD	0.8	0.6	-----