

CSE 417 Algorithms and Complexity

Richard Anderson

Lecture 22

Longest Common Subsequence

Longest Common Subsequence

- $C=c_1\dots c_g$ is a subsequence of $A=a_1\dots a_m$ 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

ocurranec

attacggct

occurrence

tacgacca

Determine the LCS of the following strings

BARTHOLEMEWSIMPSON

KRUSTYTHECLOWN

String Alignment Problem

- Align sequences with gaps

CAT TGA AT

CAGAT AGGA

- Charge δ_x if character x is unmatched
- Charge γ_{xy} if character x is matched to character y

Note: the problem is often expressed as a minimization problem,
with $\gamma_{xx} = 0$ and $\delta_x > 0$

LCS Optimization

- $A = a_1a_2\dots a_m$
- $B = b_1b_2\dots b_n$

- $\text{Opt}[j, k]$ is the length of $\text{LCS}(a_1a_2\dots a_j, b_1b_2\dots b_k)$

Optimization recurrence

If $a_j = b_k$, $\text{Opt}[j,k] = 1 + \text{Opt}[j-1, k-1]$

If $a_j \neq b_k$, $\text{Opt}[j,k] = \max(\text{Opt}[j-1,k], \text{Opt}[j,k-1])$

Give the Optimization Recurrence for the String Alignment Problem

- Charge δ_x if character x is unmatched
- Charge γ_{xy} if character x is matched to character y

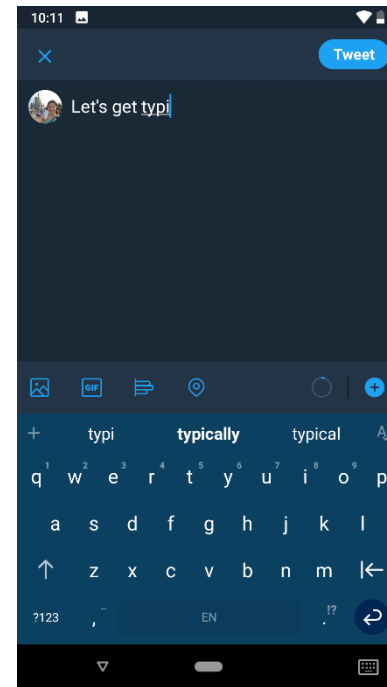
Opt[j , k] =

Let $a_j = x$ and $b_k = y$

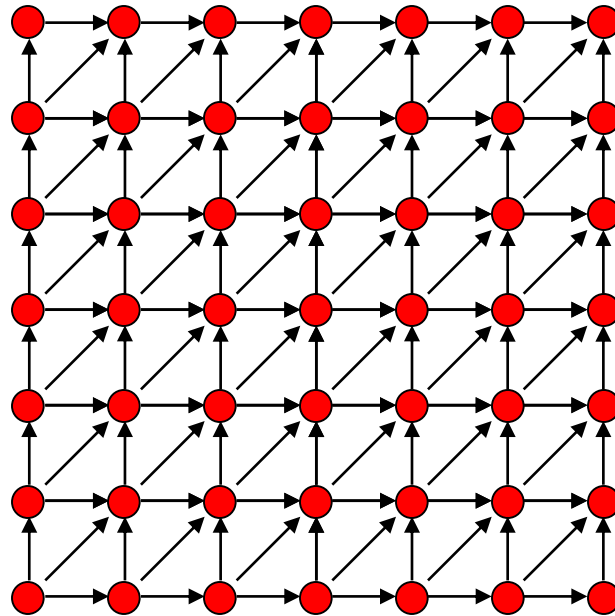
Express as minimization

String edit with Typo Distance

- Find closest dictionary word to typed word
- $\text{Dist}('a', 's') = 1$
- $\text{Dist}('a', 'u') = 6$
- Capture the likelihood of mistyping characters
- Different distance model for T9 on basic mobile phone



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

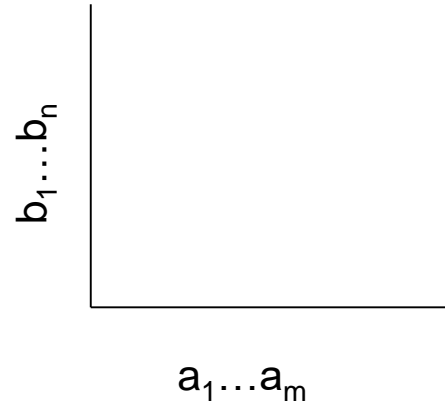
 for $j := 1$ to n

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

 else if $Opt[i-1, j] \geq Opt[i, j-1]$

 { $Opt[i, j] := Opt[i-1, j]$, $Best[i, j] := \text{Left}$; }

 else { $Opt[i, j] := Opt[i, j-1]$, $Best[i, j] := \text{Down}$; }



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*

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

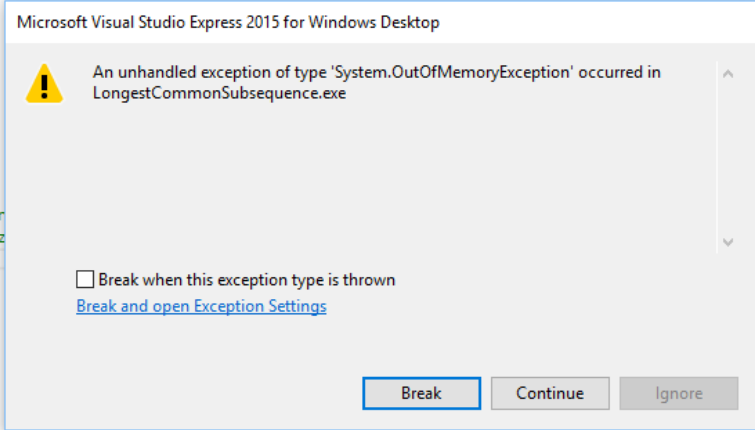
        int[,] opt;

        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 zero
             */
            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];
        }
    }
}
```



* 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

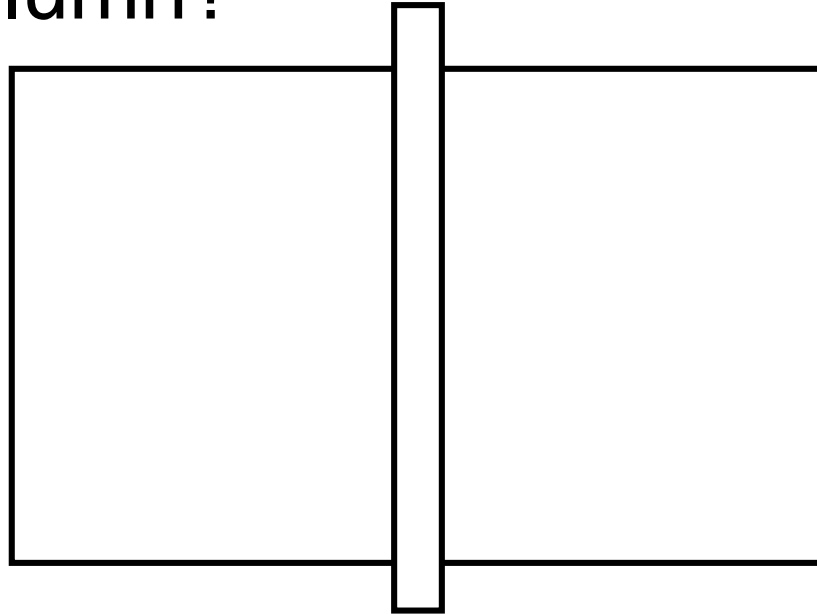
- The computation can be done in $O(m+n)$ space if we only need one column of the Opt values or Best Values
- The computation requires $O(nm)$ space if we store all of the string information

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: $T(m,n) \leq 2cnm$

