CSE 421
Algorithms
Richard Anderson
Lecture 19
Memory Efficient Dynamic Programming

Announcements
• Guest lecturers
  – Wednesday, Nov 16, Shortest Paths
  – Friday, Nov 18, Network Flow
  – Monday, Nov 21, Network Flow

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 \( B \)

\[
\text{LCS}(\text{BARTHOLEMESWIMSON}, \text{KRUSTYTHECLOWN}) = \text{RTHOWN}
\]

LCS Optimization
• \( A = a_1a_2\ldots a_m \)
• \( B = b_1b_2\ldots b_n \)
• \( \text{Opt}[j, k] \) is the length of \( \text{LCS}(a_1a_2\ldots a_j, b_1b_2\ldots b_k) \)

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

Dynamic Programming Computation
**Code to compute Opt[n, m]**

```csharp
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
                         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**

```
// Reconstructing the LCS
int i = m, j = n;
while (i > 0 && j > 0)
{  if (str1[i] == str2[j])
    {  i--; j--; Opt[i, j] = Opt[i-1, j-1] + 1;  }
    else
    {  if (Opt[i, j-1] > Opt[i-1, j])
        j--;
    else
        i--;  }
}
```

**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**

```csharp
public class LCS
{
    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
                    opt[i, j] = Math.Max(opt[i-1, j], opt[i, j-1]);
        return opt[n, m];
    }
}
```

**N = 17000**

Runtime should be about 5 seconds*

* Personal PC, 3 years old
Implementation 2

```java
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];
}
```

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 algorithm can be run from either end of the strings

Computing LCS in O(nm) time and O(n+m) space

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

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 $b_j$

Constrained LCS

- LCS$_{i,j}$(A,B): The LCS such that
  - $a_1,\ldots,a_i$ paired with elements of $b_1,\ldots,b_j$
  - $a_{i+1},\ldots,a_m$ paired with elements of $b_{j+1},\ldots,b_n$
- LCS$_{4,3}$(abbacbb, cbbaa)
A = RRSSRTTRRTS
B = RTSRRSTST

Compute LCS_{5,0}(A,B), LCS_{5,1}(A,B), …, LCS_{5,9}(A,B)

Computing the middle column

- From the left, compute LCS(a_1…a_{m/2}, b_1…b_j)
- From the right, compute LCS(a_{m/2+1}…a_m, b_{j+1}…b_n)
- Add values for corresponding j's
- Note – this is space efficient

Divide and Conquer

- A = a_1,…,a_m
  B = b_1,…,b_n
- Find j such that
  - LCS(a_1…a_{m/2}, b_1…b_j) and
  - LCS(a_{m/2+1}…a_m, b_{j+1}…b_n) yield optimal solution
- Recurse

Algorithm Analysis

- T(m,n) = T(m/2, j) + T(m/2, n-j) + cmn

Prove by induction that
T(m,n) <= 2cmn
Memory Efficient LCS Summary

- We can afford $O(nm)$ time, but we can’t afford $O(nm)$ space
- If we only want to compute the length of the LCS, we can easily reduce space to $O(n+m)$
- Avoid storing the value by recomputing values
  - Divide and conquer used to reduce problem sizes