CSE 421 Algorithms

Autumn 2019, Lecture 20
Memory Efficient Dynamic Programming

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 \( B \)

\[
\text{LCS}({\text{BARTHOLEMES SIMPSON}, \text{ KRUSTY THE CLOWN}}) = \text{RTHOWN}
\]

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

Dynamic Programming Computation

Code to compute \( \text{Opt}[n, m] \)

```c
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

```c
for (i = n-1; i >= 0; i--)
  for (j = m-1; j >= 0; j--)
    if (A[i] == B[j])
      Best[i,j] = Diag;
    else if (Opt[i-1,j] >= Opt[i,j-1])
      Best[i,j] = Left;
    else
      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.

Implementation 1

```csharp
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

Implementation 2

```csharp
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++)
        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];
        prevRow = currRow;
    return currRow[m];
}
```

N = 300000

<table>
<thead>
<tr>
<th>N</th>
<th>Base 2 Length</th>
<th>Gamma</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>8096</td>
<td>0.8096</td>
<td>00:00:01.86</td>
</tr>
<tr>
<td>20000</td>
<td>16231</td>
<td>0.81155</td>
<td>00:00:07.45</td>
</tr>
<tr>
<td>30000</td>
<td>24317</td>
<td>0.8105667</td>
<td>00:00:16.82</td>
</tr>
<tr>
<td>40000</td>
<td>32510</td>
<td>0.81275</td>
<td>00:00:29.84</td>
</tr>
<tr>
<td>50000</td>
<td>40563</td>
<td>0.81126</td>
<td>00:00:46.78</td>
</tr>
<tr>
<td>60000</td>
<td>48700</td>
<td>0.8116667</td>
<td>00:00:58.06</td>
</tr>
<tr>
<td>70000</td>
<td>56824</td>
<td>0.8117715</td>
<td>00:00:13.36</td>
</tr>
<tr>
<td>300000</td>
<td>243605</td>
<td>0.8120167</td>
<td>00:28:07.32</td>
</tr>
</tbody>
</table>

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 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

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 $b_{j+1}, \ldots, b_n$
- $LCS_{4,3}(abacbb, cbb)$

A = RRSSRTTRTS
B = RTSRRSTST

Compute $LCS_{5,0}(A,B)$, $LCS_{5,1}(A,B)$, $\ldots$, $LCS_{5,9}(A,B)$
Computing the middle column

- From the left, compute LCS(a₁…a_{m/2}, b₁…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 = a₁,…,a_m  B = b₁,…,b_n
- Find j such that
  - LCS(a₁…a_{m/2}, b₁…b_j) and
  - LCS(a_{m/2+1}…a_m, b_{j+1}…b_n) yield optimal solution
- Recurse

Algorithm Analysis

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

Prove by induction that \( T(m,n) \leq 2cmn \)

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