Dynamic Programming

Announcements

- Dynamic Programming Reading:
  - Weighted Interval Scheduling, Segmented Least Squares, Knapsack and Subset Sum
  - 6.6 String Alignment
  - 6.8 Shortest Paths (Bellman-Ford)
- Midterm, Friday, Feb 9
  - Material through 6.3 and HW 5
  - Feb 8 Section will be Midterm review
    • Old exam problems on course homepage
    • Homework 6 due Feb 14

Longest Common Subsequence

- $C=c_1…c_g$ is a subsequence of $A=a_1…a_m$ 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 $B$

$LCS(BARTHOLEMEWSIMPSON, KRUSTYTHECLOWN) = RTHOWN$

Optimization recurrence

If $a_j = b_k$, $Opt[j,k] = 1 + Opt[j-1, k-1]$

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

Code to compute $Opt[n, m]$

```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

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

for \( i := 1 \) to \( m \) \n\hspace{10pt} Opt\[i, 0\] := 0;
for \( j := 1 \) to \( n \) \n\hspace{10pt} Opt\[0,j\] := 0;
Opt\[0,0\] := 0;
for \( i := 1 \) to \( m \)
\hspace{10pt} for \( j := 1 \) to \( n \)
\hspace{20pt} if \( A[i] = B[j] \) \{ \hspace{10pt} Opt\[i,j\] := 1 + Opt\[i-1,j-1\]; \hspace{10pt} Best\[i,j\] := Diag; \}
\hspace{10pt} else if Opt\[i-1,j\] >= Opt\[i,j-1\] \{ \hspace{10pt} Opt\[i,j\] := Opt\[i-1,j\], \hspace{10pt} Best\[i,j\] := Left; \}
\hspace{10pt} else \{ \hspace{10pt} Opt\[i,j\] := Opt\[i,j-1\], \hspace{10pt} 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];
}
```

* Personal PC, 10 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++)
    {
        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] >= prevRow[j - 1])
                currRow[j] = prevRow[j];
            else
                currRow[j] = prevRow[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

• $\text{LCS}_{i,j}(A,B)$: 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 $b_{j+1},\ldots,b_n$

• $\text{LCS}_{4,3}(abbacbb, cbbaa)$

A = RRSSRTTRTS
B=RTSRRRSTST
Compute $\text{LCS}_{5,0}(A,B)$, $\text{LCS}_{5,1}(A,B),\ldots,\text{LCS}_{5,5}(A,B)$
A = RRSSRTTTRS
B = RTSRRSTST

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

<table>
<thead>
<tr>
<th>j</th>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

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) + cnm

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
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.

String Alignment Problem

- Align sequences with gaps
  
  \[
  \begin{align*}
  \text{CAT TGA AT} & \\
  \text{CAGAT AGGA} & 
  \end{align*}
  \]

- Charge $\delta_x$ if character $x$ is unmatched
- Charge $\gamma_{xy}$ if character $x$ is matched to character $y$.

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

Give the Optimization Recurrence for the String Alignment Problem

- Charge $\delta_x$ if character $x$ is unmatched
- Charge $\gamma_{xy}$ if character $x$ is matched to character $y$

\[
\text{Opt}[j, k] =
\]

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

Express as minimization.

String edit with Typo Distance

- Find closest dictionary word to typed word
- $\text{Dist}(\text{'a'}, \text{'s'}) = 1$
- $\text{Dist}(\text{'a'}, \text{'u'}) = 6$
- Capture the likelihood of mistyping characters.