CS 341: ALGORITHMS
Lecture 11: dynamic programming III
Readings: see website
Trevor Brown (co-teached with Anna Lubiw)
https://www.student.cs.uwaterloo.ca/~cs341

PROBLEM: LONGEST COMMON SUBSEQUENCE (LCS)

EXAMPLES
• X=aaaaa Y=bbbbb Z=LCS(X,Y)=?
• Z=ε (empty sequence)
• X=abcde Y=bcd Z=LCS(X,Y)=?
• Z=bcd
• X=abcde Y=labef Z=LCS(X,Y)=?
• Z=abe

POSSIBLE GREEDY SOLUTIONS?
Alg: for each 𝑥𝑖 ∈ 𝑋, try to choose a matching 𝑦𝑗 ∈ 𝑌 that is to the right of all previously chosen 𝑦𝑗 values
• X=abcde Y=labef Z=LCS(X,Y)=?
• X=abcde Y=labef [no suitable 𝑦𝑗 found]
• X=abcde Y=labef [no suitable 𝑦𝑗 found]
• X=abcde Y=labef
• Z=abe

DEFINING SUBPROBLEMS
• Full problem: return |LCS(X, Y)| (i.e., length of LCS)
• Reduce size by taking prefixes of 𝑋 or 𝑌
• Let 𝑋1 = (𝑥1, ..., 𝑥𝑖) and 𝑌1 = (𝑦1, ..., 𝑦𝑗)

POSSIBLE GREEDY SOLUTIONS?
Alg: for each 𝑥𝑖 ∈ 𝑋, try to choose a matching 𝑦𝑖 ∈ 𝑌 that is to the right of all previously chosen 𝑦𝑖 values

Blindly taking is bad. How to decide whether to take or leave it?

Similar greedy alg that goes right-to-left works for this input, but fails for other inputs.
Case 1: if $x_i = a$ must appear in $Y_{m-1}$

- $x_i$ is a must appear in $Y_{m-1}$

- $x_i$ = a must appear in $Y_{m-1}$

- This cannot be the final a in $Z$

- Since $Z$ is a subsequence of $X$, $x_i$ = a must appear in $X_{m-1}$

- $Y_{m-1}$ is a must appear in $Y_{m-1}$

- $x_i = a$ must be in $Y_{m-1}$

Consider optimal solution $Z = LCS(X,Y)$

Since $x_m,y_0 \notin Z$ we know $Z = LCS(x_0, Y_{m-1})$

Since $x_m,y_0 \notin Z$ we know $Z = LCS(x_0, Y_{m-1})

### Building Solutions from Subproblems

**Example #1**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>z</td>
<td>y</td>
<td>c</td>
</tr>
</tbody>
</table>

**Example #2**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>z</td>
<td>y</td>
<td>c</td>
</tr>
</tbody>
</table>

**Example #3**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>z</td>
<td>y</td>
<td>c</td>
</tr>
</tbody>
</table>

**Deriving a Recurrence**

Recall $Z = LCS(X,Y)$

- $x_i$ matches neither $x_m$ nor $y_n$  
  $Z = LCS(X_{m-1}, Y_{n-1})$

- $x_i$ matches $x_m$ but not $y_n$  
  $Z = LCS(X_{m}, Y_{n-1})$

- $x_i$ matches $y_n$ but not $x_m$  
  $Z = LCS(X_{m-1}, Y_{n})$

- $x_i$ matches both
  
  Let $c(i,j) = \left \lceil LCS(X_i,Y_j) \right \rceil$

  - Brainstorming sensible base cases
    
    - $i = 0$  
      one string is empty, so $c(0,j) = 0$ [similarly for $j = 0$]

  - General cases
    
    $c(i,j) = \max\{c(i-1,j-1), c(i-1,j), c(i,j-1)\}$

    if $x_m = y_n$

    \[
    c(i,j) = \begin{cases} 
    0 & \text{if } i = 0 \text{ or } j = 0 \\
    c(i-1,j-1)+1 & \text{if } i,j \geq 1 \text{ and } x_i = y_j \\
    \max\{c(i-1,j-1), c(i-1,j), c(i,j-1)\} & \text{otherwise}
    \end{cases}
    \]

    Can simplify!

    Observe $c(i, j - 1) \leq c(i, j) - 1$

    (former is a subproblem of the latter)

### Recurrence

Combining expressions

- $c(i,f) = \begin{cases} 
  0 & \text{if } i \geq 1 \text{ and } x_i = y_j \\
  \max\{c(i, j-1), c(i-1,j), c(i, j-1)\} & \text{otherwise}
  \end{cases}

- \text{if } i \geq 1 \text{ and } x_i = y_j

- \text{if } i = 0 \text{ or } j = 0

- \text{otherwise}

- $c(i,j) = \begin{cases} 
  0 & \text{if } i = 0 \text{ or } j = 0 \\
  c(i-1,j-1)+1 & \text{if } i,j \geq 1 \text{ and } x_i = y_j \\
  \max\{c(i-1,j-1), c(i-1,j), c(i,j-1)\} & \text{otherwise}
  \end{cases}

- \text{if } i \geq 1 \text{ and } x_i = y_j

- \text{if } i = 0 \text{ or } j = 0

- \text{otherwise}
Suppose $X = \text{gdvegta}$ and $Y = \text{gvcetak}$.

<table>
<thead>
<tr>
<th>$X$</th>
<th>$Y$</th>
<th>$i$</th>
<th>$g$</th>
<th>$d$</th>
<th>$v$</th>
<th>$e$</th>
<th>$t$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>g</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PSEUDOCODE**

**SAVING THE DIRECTION TO THE PREDECESSOR SUBPROBLEM $\pi$**

```
c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max(c(i - 1, j - 1), c(i - 1, j), c(i, j - 1)) + 1 & \text{otherwise} 
\end{cases}
```

**COMPUTING THE LCS**

**EXAMPLE**

Suppose $X = \text{gdvegta}$ and $Y = \text{gvcetak}$.

**Solved:**

- $c(2, 3) = c(1, 2) + 1$ (Case 1)
- $c(7, 4) = c(6, 3) + 1$ (Case 2)
- $c(5, 6) = c(4, 5) + 1$ (Case 3)

**Done:**

- seq-gvet
- seq-gvet
- seq-gvet
- seq-gvet
- seq-gvet
- seq-gvet
- seq-gvet
- seq-gvet
- seq-gvet

**How to obtain LCS=gvet from this table?**

- Case 1: We store "I" in $x(i, j)$ to indicate decrementing $i$ and $j$.
- Case 2: We store "V" in $x(i, j)$ to indicate decrementing $j$.
- Case 3: We store "C" in $x(i, j)$ to indicate decrementing $i$.

Recall in this case, $y_i = y_j$, so we include $x_i$ in the LCS.
FOLLOWING PREDECESSORS TO COMPUTE THE LCS

```
1: FindLCS(c[0..n][0..n], m[0..n][0..n], X[0..n])
2: lcs = new string
3: i = n
4: j = n
5: while i>0 and j>0
6:   if m[i][j] == "I"
7:     lcs.append(X[i])
8:     i--
9:   else if m[i][j] == "J"
10:    j--
11:  else lcs[i][j] == "I"
12:    i--
13: return reverse(lcs)

Complexities of this trace-back algo:
Space? Time?
(word RAM model)
space: O(n+m) words
time: O(n+m)
```

PROBLEM: MINIMUM LENGTH TRIANGULATION

Input: n points q₁, q₂, ..., qₙ in 2D space that form a convex n-gon P
Assume points are sorted clockwise around the center of P
Find: a triangulation of P such that the sum of the perimeters of the n−2 triangles is minimized
Output: the sum of the perimeters of the triangles in P

HOW HARD IS THIS PROBLEM?

The number of triangulations of a convex n-gon is the (n − 2)nd Catalan number.

It can be shown that Cₙ₋₂ ∈ Θ(4ⁿ/(n−2)²³/₂).

PROBLEM DECOMPOSITION

The edge qᵢqₖ is in a triangle with a third vertex qₙ, where k ∈ {2, ..., n−1}.

For a given k, we have:
the triangle qᵢqₙqₖ. (1)
the polygon with vertices q₁, ..., qₙ. (2)
PROBLEM DECOMPOSITION

The edge $q_i q_k$ is in a triangle with a third vertex $q_j$, where $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:
- the triangle $q_i q_k q_j$, (1)
- the polygon with vertices $q_1, \ldots, q_k$, (2)
- the polygon with vertices $q_k, \ldots, q_n$, (3)

The optimal solution will consist of optimal solutions to the two subproblems in (2) and (3), along with the triangle in (1).

PROBLEM DECOMPOSITION

The edge $q_i q_j$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n - 1\}$.

For a given $k$, we have:
- the triangle $q_i q_j q_k$, (1)
- the polygon with vertices $q_1, \ldots, q_j$, (2)
- the polygon with vertices $q_j, \ldots, q_n$, (3)

The optimal solution will consist of optimal solutions to the two subproblems in (2) and (3), along with the triangle in (1).

RECURSIVE RELATION

- Let $S(i,j)$ denote the optimal solution to the subproblem consisting of the polygon with vertices $q_i \ldots q_j$.
- Let $\Delta_{ijk}$ denote the perimeter of the subpolygon $q_i \ldots q_j$.

If a given point $q_k$ is in the optimal solution, then $S(i,j) = S(i,k) + \Delta_{ijk} + S(k,j)$.

The table $S(1..n, 1..n)$ of solutions to $S(i,j)$ for all $i, j \in \{1..n\}$

```
<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Dependencies:
- $S(i,j)$ and $S(f,j)$ for $k = \{i+1, (i+1)\}$
- $S(i,k)$ and $S(f,k)$ for $j = \{i+1, (i+1)\}$
- ... (full table)
```

FILLING IN THE TABLE

- We depend on larger $j$ and same $i$ but smaller $j$.
- What is the correct fill order?
  - For $i = 0, 1$, for $j = 1..n$.

**RUNTIME**

**WORD RAM MODEL**

- Number of subproblems: $n^2$
- Time to solve subproblem $S(i,j): O(j - i) \leq O(n)$
- So total runtime is in $O(n^3)$
- More effort needed to show $\Omega(n^3)$, since so many subproblems are base cases, which take $\Theta(1)$ steps
- Incidentally, this is polynomial time (in the input size)
- But basic runtime analysis does not require such an argument

**MEMOIZATION: AN ALTERNATIVE TO DP**

Recall that the goal of dynamic programming is to eliminate solving subproblems more than once.

Memoization is another way to accomplish the same goal.

Memoization is a recursive algorithm based on same recurrence relation as would be used by a dynamic programming algorithm.

The idea is to remember which subproblems have been solved; if the same subproblem is encountered more than once during the recursion, the solution will be looked up in a table rather than being re-calculated.

This is easy to do if initialize a table of all possible subproblems having the value undefined in every entry.

Whenever a subproblem is solved, the table entry is updated.

---

**EXAMPLE: USING MEMOIZATION TO COMPUTE FIBONACCI NUMBERS EFFICIENTLY**

```plaintext
main
for i ← 2 to n
    do M[i] ← -1
return (RecFib(n))
```

procedure RecFib(n)
    if $n = 0$ then $f ← 0$
    else if $n = 1$ then $f ← 1$
    else if $M[n] ≠ -1$ then $f ← M[n]$
    else $f ←$ RecFib(n - 2)
    return ($f$)

**BONUS CLARIFICATION MATERIAL**

**MODEL COMPARISON**

- **Word RAM**
  - Each variable (or array entry, etc.) $x$ is a word
  - All words have the same size
  - $O(\log W)$ bits where $W$ is the # of words in the input (not realistic if variables contain big numbers)
  - Runtime is the number of operations on words (where each word operation takes $O(1)$ time)
  - Read/write in $O(1)$
  - Add in $O(1)$
  - Multiply in $O(1)$
  - Space complexity is the number of words used (excluding the input)

- **Bit complexity**
  - Each variable $x$ is a bit string
  - Variables can have different numbers of bits
  - $x$ is encoded in $O(\log x)$ bits
  - Runtime is the number of operations on bits (where each bit operation takes $O(1)$ time)
  - Read/write in $O(1)$
  - Add in $x \in \mathbb{N}$
  - Multiply in $O(\log x \cdot \log x)$
  - Space complexity is the number of bits used (excluding the input)
CALCULATIONS USING INPUT SIZE

- Clarification: you can compute space/time complexity without calculating the input size
- Input size calculations are typically only needed if we ask you to show an algorithm runs in polytime
- “runs in polytime” means the runtime is at most a polynomial in the number of bits in the input
- Lots of this in tractability / NP completeness

Just trying to expose you to these ideas ahead of time...

SO... IS DP LCS A POLYTIME ALGORITHM?

- Is \( nm \) polynomial in the input size (# of bits in the input)?
- Word RAM model
  - Input contains \( \Theta(n+m) \) words
  - Word RAM model says each word stores \( \Theta(\log w) \) bits
  - where \( w = \# \) words in the input
  - So in this case, \( \Theta(\log(n+m)) \) bits per word
  - So \( S \in \Theta((n+m)\log(n+m)) \)
  - Want a term that looks like \( nm \)

SO... IS DP LCS A POLYTIME ALGORITHM?

- Try squaring: \( S^2 \in \Theta((n+m)\log(n+m))^2 \)
- \( \Theta(n^2 \log^2(n+m) + nm \log^2(n+m) + m^2 \log^2(n+m)) \)
- Of course, \( nm \in O(nm \log^2(n+m)) \)
- ... which is just one of the terms of \( S^2 \)
- \( S \) is a polynomial in the input size (in bits)
- But this was ugly. Is there a simpler approach?

SO... IS DP LCS A POLYTIME ALGORITHM?

- Calculation using words would be simpler...
  - Words in the input \( W \in O(n+m) \)
  - \( W^2 \in O((n+m)^2) = O(n^2 + nm + m^2) \)
  - \( O(nm) \subseteq O(W^2) \)
  - I.e., polynomial in the # of \textit{words} in the input

- How does this help us?
  - \# words \( W \) in the input \( \leq \# \) bits \( S \) in the input
  - \( So O(nm) \subseteq O(W^2) \implies O(nm) \subseteq O(S^2) \)

So, for showing polytime, you can calculate the input size using words (and justify like this)