CS 341: ALGORITHMS

Lecture 9: dynamic programming III

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca
**PROBLEM: MINIMUM LENGTH TRIANGULATION**

- **Input:** \( n \) points \( q_1, \ldots, q_n \) 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 many triangulations are there?

Number of triangulations of a convex \( n \)-gon = the \((n-2)\)nd Catalan number

This is \( C_{n-2} = \frac{1}{n-1} \binom{2n-4}{n-2} \)

It can be shown that \( C_{n-2} \in \Theta(4^n/(n-2)^{3/2}) \)
PROBLEM DECOMPOSITION

The edge $q_nq_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n-1\}$. 
PROBLEM DECOMPOSITION

The edge $q_nq_1$ 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 $q_1q_kq_n$, \hspace{1cm} (1)
The edge $q_n q_1$ 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 $q_1 q_k q_n$, \hspace{1cm} (1)
- the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
PROBLEM DECOMPOSITION

The edge $q_n q_1$ 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 $q_1 q_k q_n$, \hspace{1cm} (1)
- the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
- the polygon with vertices $q_k, \ldots, q_n$. \hspace{1cm} (3)
PROBLEM DECOMPOSITION

The edge \( q_n q_1 \) is in a triangle with a third vertex \( q_k \), where \( k \in \{2, \ldots, n-1\} \).

For a given \( k \), we have:

1. the triangle \( q_1 q_k q_n \), \hspace{1cm} (1)
2. the polygon with vertices \( q_1, \ldots, q_k \), \hspace{1cm} (2)
3. the polygon with vertices \( q_k, \ldots, q_n \). \hspace{1cm} (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 $q_nq_1$ 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 $q_1q_kq_n$, \hspace{1cm} (1)
- the polygon with vertices $q_1, \ldots, q_k$, \hspace{1cm} (2)
- the polygon with vertices $q_k, \ldots, q_n$. \hspace{1cm} (3)

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

- Let $S(i, j) = \text{optimal solution to the subproblem consisting of the polygon with vertices } q_i \ldots q_j$
- Let $\Delta_{ijk}$ denote $\text{perimeter}(q_i q_j q_k)$
- **If** a given triangle $q_i, q_j, q_k$ is in the **optimal** solution, then $S(i, j) = S(i, k) + \Delta_{ijk} + S(k, j)$
But we don’t know the optimal $k$

Minimize over all $k$ strictly between $i$ and $j$

$$S(i, j) = \begin{cases} 
\min_{i < k < j} \{S(i, k) + \Delta_{ijk} + S(k, j)\} & \text{if } j \geq i + 2 \\
0 & \text{otherwise}
\end{cases}$$
FILLING IN THE TABLE

Table \( S[1..n, 1..n] \) of solutions to \( S(i, j) \) for all \( i, j \in \{1..n\} \)

\[
S(i, j) = \begin{cases} 
\min_{i < k < j} \{S(i, k) + \Delta_{ijk} + S(k, j)\} & \text{if } j \geq i + 2 \\
0 & \text{otherwise}
\end{cases}
\]

Dependencies:
\( S[i, k] \) and \( S[k, j] \)
For \( k = (i + 1) \ldots (j - 1) \)

We depend on larger \( i \)
And same \( i \) but smaller \( j \)

What's a correct fill order?
for \( i = n \ldots 1 \), for \( j = 1 \ldots n \)
RUNTIME
WORD RAM MODEL

- Number of subproblems: $n^2$
- Time to solve subproblem $S(i, j)$: $O(j - i) \subseteq O(n)$
- So total runtime is in $O(n^3)$
  - Some 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

$$S(i, j) = \begin{cases} \min_{i < k < j} \{ S(i, k) + \Delta_{ijk} + S(k, j) \} & \text{if } j \geq i + 2 \\ 0 & \text{otherwise} \end{cases}$$
PROBLEM: LONGEST COMMON SUBSEQUENCE (LCS)

Problem 5.3
Longest Common Subsequence
Instance: Two sequences $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$ over some finite alphabet $\Gamma$.
Find: A maximum length sequence $Z$ that is a subsequence of both $X$ and $Y$.

$Z = (z_1, \ldots, z_\ell)$ is a subsequence of $X$ if there exist indices $1 \leq i_1 < \cdots < i_\ell \leq m$ such that $z_j = x_{i_j}, 1 \leq j \leq \ell$.

Similarly, $Z$ is a subsequence of $Y$ if there exist (possibly different) indices $1 \leq h_1 < \cdots < h_\ell \leq n$ such that $z_j = y_{h_j}, 1 \leq j \leq \ell$.

Let’s first solve for the length of the LCS.
EXAMPLES

- $X=aaaaa \quad Y=bbbbb \quad Z=LCS(X,Y)=\epsilon$ (empty sequence)
- $X=abcde \quad Y=bcd \quad Z=bcd$
- $X=abcde \quad Y=labeef \quad Z=abe$
POSSIBLE GREEDY SOLUTIONS?

- Alg: for each $x_i \in X$, try to choose a matching $y_j \in Y$ that is to the right of all previously chosen $y_j$ values
  - $X=\text{abcde}$, $Y=\text{labef}$
  - $X=\text{ab}_c\text{de}$, $Y=\text{labef}$
  - $X=\text{ab}_c\text{de}$, $Y=\text{labef}$ [no suitable $y_j$ found]
  - $X=\text{abc}_d\text{e}$, $Y=\text{labef}$ [no suitable $y_j$ found]
  - $X=\text{abcd}_e$, $Y=\text{labef}$
  - $Z=\text{abe}$, Optimal?
POSSIBLE GREEDY SOLUTIONS?

- Alg: for each $x_i \in X$, try to choose a matching $y_j \in Y$ that is to the right of all previously chosen $y_j$ values
  
  - $X=azbracadabra$ $Y=abracadabraz$
  - $X=azbracadabra$ $Y=abracadabrazz$ [no $y_j$ after $z$]
  - $X=azbracadabra$ $Y=abracadabraz$ [no $y_j$ after $z$]
  - $X=azbracadabra$ $Y=abracadabra$ $Y=abracadabraz$ [no $y_j$ after $z$]

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

Try both possibilities! (Brute force / dynamic programming)

Similar greedy alg that goes right-to-left works for this input, but fails for other inputs.

Z-optimal?
DEFINING SUBPROBLEMS

- **Full problem:** $|\text{LCS}(X, Y)|$ (i.e., length of LCS)
  - Reduce size by taking **prefixes** of $X$ or $Y$
  - Let $X_i = (x_1, ..., x_i)$ and $Y_i = (y_1, ..., y_i)$

<table>
<thead>
<tr>
<th>$X_m$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>...</th>
<th>...</th>
<th>$x_{m-1}$</th>
<th>$x_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_4$</td>
<td>$x_1$</td>
<td>$x_2$</td>
<td>$x_3$</td>
<td>$x_4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Note $X = X_m$ and $Y = Y_n$

- **Subproblem:** $|\text{LCS}(X_i, Y_j)|$

- **Shrinking the problem:** remove the last letter of $X$ or $Y$
### BUILDING SOLUTIONS FROM SUBPROBLEMS

#### EXAMPLE #1 TO BUILD INTUITION

**X**

| a | b | r | a | c | a | z |

**Y**

| a | z | b | r | a | c | a | d | a | b |

**Z**

| a | b | r | a | c | a |

---

Consider optimal solution $Z = \text{LCS}(X, Y)$

Since $x_m, y_n \notin Z$ we know $Z = \text{LCS}(X_{m-1}, Y_{n-1})$

---

Neither of these is part of $Z$

This cannot be the final $a$ in $Z$

This cannot be the final $a$ in $Z$

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

$z_\ell = a$ must be in $Y_{n-1}$
BUILDING SOLUTIONS FROM SUBPROBLEMS

EXAMPLE #2

Since \( Z \) is a subsequence of \( Y \), \( z_\ell = a \) must appear in \( Y_{n-1} \).

Since \( y_n \notin Z \) we know \( Z = \text{LCS}(X, Y_{n-1}) \).

Case \( x_m \notin Z, y_n \in Z \) is symmetric.

\[ Z = \text{LCS}(X_{m-1}, Y) \]
Building Solutions from Subproblems

Example #3

\[ X = a \ b \ r \ a \ c \ a \ z \ a \]

\[ Y = a \ z \ b \ r \ a \ c \ a \ d \ d \ a \]

\[ Z = a \ b \ r \ a \ c \ a \ a \]

Or maybe this is...

This might be the final \( a \) in \( Z \)

This might be the final \( a \) in \( Z \)

Might as well match \( x_m \) and \( y_n \) with \( z_\ell \)

Then we have \( Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell \)
SUMMARIZING CASES

- \( z_\ell \) matches **neither** \( x_m \) nor \( y_n \)  
  \[ Z = \text{LCS}(X_{m-1}, Y_{n-1}) \]

- \( z_\ell \) matches \( x_m \) but not \( y_n \)  
  \[ Z = \text{LCS}(X_m, Y_{n-1}) \]

- \( z_\ell \) matches \( y_n \) but not \( x_m \)  
  \[ Z = \text{LCS}(X_{m-1}, Y_n) \]

- \( z_\ell \) matches **both**  
  \[ Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell \]

- ... but we don’t know \( z_\ell \)
  - Try all cases and maximize
  - **Careful:** last case is only valid if \( x_m = y_n \)
  - Also note \( x_m = y_n \) only holds in the last case
  - Cases 2&3: trivial
  - Case 1: if \( x_m = y_n \neq z_\ell \) then we can improve \( Z \) (contra)
DERIVING A RECURRENCE

- $z_\ell$ matches **neither** $x_m$ nor $y_n$ \((x_m \neq y_n)\) \(Z = \text{LCS}(X_{m-1}, Y_{n-1})\)
- $z_\ell$ matches $x_m$ but not $y_n$ \((x_m \neq y_n)\) \(Z = \text{LCS}(X_m, Y_{n-1})\)
- $z_\ell$ matches $y_n$ but not $x_m$ \((x_m \neq y_n)\) \(Z = \text{LCS}(X_{m-1}, Y_n)\)
- $z_\ell$ matches **both** \((x_m = y_n)\) \(Z = \text{LCS}(X_{m-1}, Y_{n-1}) + z_\ell\)

Let \(c(i, j) = |\text{LCS}(X_i, Y_j)|\)

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

- General cases

<table>
<thead>
<tr>
<th>(c(i, j))</th>
<th>Condition</th>
<th>Formulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(c(i, j) = c(i - 1, j - 1) + 1)</td>
<td>(x_m = y_n)</td>
<td>()</td>
</tr>
<tr>
<td>(c(i, j) = \max{c(i - 1, j - 1), c(i, j - 1), c(i - 1, j)})</td>
<td>(x_m \neq y_n)</td>
<td>()</td>
</tr>
</tbody>
</table>
RECURRENCE

• Combining expressions

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

• Can simplify!

• Observe \( c(i - 1, j - 1) \leq c(i - 1, j) \)
  (former is a subproblem of the latter)

\[ c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\max\{c(i, j - 1), c(i - 1, j), c(i - 1, j - 1)\} & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max\{c(i, j - 1), c(i - 1, j), c(i - 1, j - 1)\} & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j 
\end{cases} \]
Suppose $X = \text{gdvegta}$ and $Y = \text{gvcekst}$

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

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

**Question 1**

- **Q2**
- **Q3**
- **Q4**
- **Q5**
- **Q6**
- **Q7**

...
Suppose \( X = \text{gdvegta} \) and \( Y = \text{gvcekst} \)

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

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

Algorithm: LCS1($X = (x_1, \ldots, x_m), Y = (y_1, \ldots, y_n)$)

for $i \leftarrow 0$ to $m$
    $c[i, 0] \leftarrow 0$

for $j \leftarrow 0$ to $n$
    $c[0, j] \leftarrow 0$

for $i \leftarrow 1$ to $m$
    for $j \leftarrow 1$ to $n$
        if $x_i = y_j$
            then $c[i, j] \leftarrow c[i - 1, j - 1] + 1$
        else $c[i, j] \leftarrow \max\{c[i, j - 1], c[i - 1, j]\}$

return $(c[m, n])$;

Complexity:
Space? Time?
(word RAM model)

$\Theta(nm)$ for both
COMPUTING THE LCS NOT JUST ITS LENGTH

To make it easy to find the actual LCS (not just its length),

Consider **which table entry** was used to calculate $c[i, j]$

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

We store the **direction** to that entry in an array $\pi[i, j]$.

**Case 1:** $c(i, j) = c(i, j - 1)$

We store “J” in $\pi[i, j]$ to indicate decrementing $j$ (to get $i, j - 1$).

**Case 2:** $c(i, j) = c(i - 1, j)$

We store “I” in $\pi[i, j]$ to indicate decrementing $i$ (to get $i - 1, j$).

**Case 3:** $c(i, j) = c(i - 1, j - 1) + 1$

We store “IJ” in $\pi[i, j]$ to indicate decrementing both $i$ and $j$.

Recall in this case, $x_i = y_j$ so we **include $x_i$ in the LCS**.
SAVING THE DIRECTION TO THE PREDECESSOR SUBPROBLEM $\pi$

Case: $c(i, j) = c(i - 1, j - 1) + 1$
We store “IJ” in $\pi[i,j]$ to indicate decrementing both $i$ and $j$

Recall in this case, $x_i = y_j$ so we include $x_i$ in the LCS

Case: $c(i,j) = c(i,j-1)$
We store “J” in $\pi[i,j]$ to indicate decrementing $j$ (to get $i,j-1$)

Case: $c(i,j) = c(i-1,j)$
We store “I” in $\pi[i,j]$ to indicate decrementing $i$ (to get $i-1,j$)

LCS2(X[1..m], Y[1..n])

- $c = \text{new array}[0..m][0..n]$
- $\pi = \text{new array}[0..m][0..n]$

for $i = 0..m$ do $c[i][0] = 0$
for $j = 0..n$ do $c[0][j] = 0$

for $i = 1..m$
  for $j = 1..n$
    if $X[i] = Y[j]$
      $c[i][j] = c[i-1][j-1] + 1$
      $\pi[i][j] = \text{"IJ"}$
    else if $c[i][j-1] > c[i-1][j]$
      $c[i][j] = c[i][j-1]$
      $\pi[i][j] = \text{"J"}$
    else // $c[i][j-1] \leq c[i-1][j]$
      $c[i][j] = c[i-1][j]$
      $\pi[i][j] = \text{"I"}$

return $c$, $\pi$
Suppose $X = \text{gdvegta}$ and $Y = \text{gvcekst}$.

### Example

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

**Done:**
- seq=gvet

**seq=gvet**

**seq=vet**

**seq=et**

**this is.**
- seq=t

**this “a” is not in**

**How to obtain LCS=gvet from this table?**
FOLLOWING PREDECESSORS TO COMPUTE THE LCS

Complexities of this trace-back algo:
Space? Time?
(word RAM model)

space: $O(n+m)$ words

time: $O(n+m)$

```plaintext
FindLCS(c[0..m][0..n], π[0..m][0..n], X[0..m])
  lcs = new string
  i = m
  j = n

  while i>0 and j>0
    if π[i][j] == "IJ"
      lcs.append(X[i])
      i--
      j--
    else if π[i][j] == "J"
      j--
    else // π[i][j] == "I"
      i--

  return reverse(lcs)
```
UNLIKELY TO GET THIS FAR

So this is likely just an exercise for you...
COIN CHANGING
## Coin Changing

### Problem 5.2

**Coin Changing**

**Instance:** A list of coin denominations, \(1 = d_1, d_2, \ldots, d_n\), and a positive integer \(T\), which is called the target sum.

**Find:** An \(n\)-tuple of non-negative integers, say \(A = [a_1, \ldots, a_n]\), such that \(T = \sum_{i=1}^{n} a_i d_i\) and such that \(N = \sum_{i=1}^{n} a_i\) is minimized.

---

What subproblems should be considered?

What table of values should we fill in?

---

There is a denomination with unit value!

In 0-1 knapsack, we only considered **two subproblems** in our recurrence: taking an item, or not.

Here we can do **more than** use a coin denomination or not.
Let $N[i, t]$ denote the optimal solution to the subproblem consisting of the first $i$ coin denominations $d_1, \ldots, d_i$ and target sum $t$.

<table>
<thead>
<tr>
<th>Exploring: some sensible base case(s)?</th>
</tr>
</thead>
<tbody>
<tr>
<td>General case:</td>
</tr>
<tr>
<td>What are the different ways we could use coin denomination $d_i$?</td>
</tr>
<tr>
<td>What subproblems / solutions should we use?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Final recurrence relation</th>
</tr>
</thead>
</table>
Let $N[i, t]$ denote the optimal solution to the subproblem consisting of the first $i$ coin denominations $d_1, \ldots, d_i$ and target sum $t$. Since $d_1 = 1$, we immediately have $N[1, t] = t$ for all $t$.

General case:
What are the different ways we could use coin denomination $d_i$?
What subproblems / solutions should we use?

Final recurrence relation

Also $N[i, 0] = 0$ for all $i$
Let $N[i, t]$ denote the optimal solution to the subproblem consisting of the first $i$ coin denominations $d_1, \ldots, d_i$ and target sum $t$. Since $d_1 = 1$, we immediately have $N[1, t] = t$ for all $t$.

For $i \geq 2$, the number of coins of denomination $d_i$ is an integer $j$ where $0 \leq j \leq \lfloor t/d_i \rfloor$.

If we use $j$ coins of denomination $d_i$, then the target sum is reduced to $t - jd_i$, which we must achieve using the first $i - 1$ coin denominations. Thus we have the following recurrence relation:

$$
N[i, t] = \begin{cases} 
\min\{j + N[i - 1, t - jd_i] : 0 \leq j \leq \lfloor t/d_i \rfloor\} & \text{if } i \geq 2 \\
t & \text{if } i = 1 \text{ OR } t = 0
\end{cases}
$$

Also $N[i, 0] = 0$ for all $i$. 

37
FILLING THE ARRAY

\[ N[1 \ldots n, 0 \ldots T] : \]

\[ N[i, t] = \begin{cases} 
\min\{ j + N[i-1, t-jd_i] : 0 < j < |t/d_i| \} & \text{if } i > 2 \\
t & \text{if } i = 1, \text{ OR } t = 0
\end{cases} \]

No data dependencies on any other array cells.

\textit{i-axis} (coin type)

(recall: \( N[i, t] \) uses coin types \( 1 \ldots i \))
FILLING THE ARRAY

\[ N[1 \ldots n, 0 \ldots T] : \]

\[
N[i, t] = \begin{cases} 
\min\{j + N[i - 1, t - jd_i] : 0 \leq j \leq \lfloor t/d_i \rfloor\} & \text{if } i \geq 2 \\
t & \text{if } i = 1.
\end{cases}
\]

OR \[ t = 0 \]

No data dependencies on any other array cells.

\( i \)-axis (coin type)

(recall: \( N[i, t] \) uses coin types 1..i)

\( t \)-axis (target sum remaining)
FILLING THE ARRAY $N[1 \ldots n, 0 \ldots T]$: 

\[
N[i, t] = \begin{cases} 
\min\{j + N[i - 1, t - jd_i] : 0 \leq j \leq \lfloor t/d_i \rfloor\} & \text{if } i \geq 2 \\
0 & \text{if } i = 1.
\end{cases}
\]

OR $t = 0$

We only look at the previous $i$-row!

Consider cell $N[i, t]$

It is sufficient to fill:
row $i=1$ (base case), then
for $(i = 2 \ldots n)$, for $(t = 0 \ldots T)$

$i$-axis (coin type)
(recall: $N[i, t]$ uses coin types $1 \ldots i$)

$t$-axis (target sum remaining)
CoinChangingDP(\(d[1..n], T\))

\[ N[i, t] = \begin{cases} \min\{j + N[i - 1, t - jd_i] : 0 \leq j \leq \lfloor t/d_i \rfloor\} & \text{if } i \geq 2 \\ t & \text{if } i = 1. \end{cases} \]

\(J[i, t] = \# \text{ of coins of type } d_i \text{ used in } N[i, t]\)

using other coin types

\(i.e., \text{ using coin } d_1 = 1\)

Compute \(\min\{\ldots\}\) over \(j = 0 \ldots \lfloor t/d_i \rfloor\)
Exercise for later:
compute the correct output
without using \( J[i, t] \)
(i.e., using only \( N, d, T \))
CoinChangingDP(d[1..n], T)
N = new table[1..n][0..T]
J = new table[1..n][0..T]

for t = 0..T    // base cases where i=1
    N[1][t] = t
    J[1][t] = t

for i = 2..n    // general cases
    for t = 0..T
        // initially best solution is 0 of d[i]
        N[i][t] = N[i-1][t]
        J[i][t] = 0

        // try j>0 coins of type d[i]
        for j = 1..floor(t / d[i])
            if j + N[i-1][t-j*d[i]] < N[i][t]
                N[i][t] = j + N[i-1][t-j*d[i]]
                J[i][t] = j // best is currently j of d[i]

return N[n][T]    // can also return N, J

Consider instance $I = (d, T)$

Runtime $R(I) \in O \left( \sum_{i=2}^{n} \sum_{t=0}^{T} \left\lceil \frac{t}{d_i} \right\rceil \right)$

$R(I) \in O \left( \sum_{i=2}^{n} \frac{1}{d_i} \sum_{t=0}^{T} t \right)$

$R(I) \in O \left( \sum_{i=2}^{n} \frac{1}{d_i} \left( \frac{T(T+1)}{2} \right) \right)$

$R(I) \in O(DT^2)$  
where $D = \sum_{i=2}^{n} \frac{1}{d_i} < n.$

If $T$ is small, this is much better than brute force.
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
        f1 ← RecFib(n - 1)
        f2 ← RecFib(n - 2)
        f ← f1 + f2
        M[n] ← f
    return (f);
```

If M[n] is already computed, don’t recurse!
VISUALIZING MEMOIZATION

If $M[n]$ is already computed, don't recurse!

```
procedure RecFib(n)
    if $n = 0$ then $f \leftarrow 0$
    else if $n = 1$ then $f \leftarrow 1$
    else if $M[n] \neq -1$ then $f \leftarrow M[n]$
        \[ f_1 \leftarrow \text{RecFib}(n-1) \]
        \[ f_2 \leftarrow \text{RecFib}(n-2) \]
    else \[ f \leftarrow f_1 + f_2 \]
        \[ M[n] \leftarrow f \]
    return $(f)$;
```