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 $C_{n-2} \in \Theta\left(\frac{4^n}{(n-2)^{3/2}}\right)$
The edge $q_n q_1$ is in a triangle with a third vertex $q_k$, where $k \in \{2, \ldots, n-1\}$. 
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)
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 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:

1. the triangle $q_1q_kq_n$, \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 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 $q_1q_kq_n$, (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).
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 $\text{perimeter}(q_i \ldots q_j)$

• If a given triangle $q_i, q_j, q_k$ is in the optimal solution, then $S(i, j) = S(i, k) + \Delta_{ijk} + S(k, j)$
RECURRENCE RELATION

• 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 $k = (i + 1) \ldots (j - 1)$

$S[i, k]: 
S[i, i + 1] \ldots S[i, j - 1]$

$S[k, j]:
S[i + 1, j] \ldots S[j - 1, j]$

We depend on larger $i$
And same $i$ but smaller $j$

What's a correct fill order?
for $i = n..1$, for $j = 1..n$
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
**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 Y=bbbbbb Z=LCS(X,Y)=?
  • Z=ε (empty sequence)

• X=abcde Y=bcd Z=LCS(X,Y)=?
  • Z=bcd

• X=abcde Y=labeef Z=LCS(X,Y)=?
  • 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

  • $X=abcde$  \hspace{1cm} $Y=labef$
  • $X=ab_{\text{bcd}}e$  \hspace{1cm} $Y=labef$ [no suitable $y_j$ found]
  • $X=abc_{\text{cd}}e$  \hspace{1cm} $Y=labef$ [no suitable $y_j$ found]
  • $X=abc_{\text{de}}$  \hspace{1cm} $Y=labef$
  • $Z=abe$  \hspace{1cm} 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.
  
  • $X=azbracadabra \quad Y=abracadabraz$
  
  • $X=azbracadabra \quad Y=abracadabraz$ [no $y_j$ after $z$]
  
  • $X=azbracadabra \quad Y=abracadabraz$ [no $y_j$ after $z$]
  
  • $Z=az$ Optimal?

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

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

Try both possibilities! (Brute force / dynamic programming)
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>$x_{m-1}$</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>
</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 $Y$
Consider optimal solution \( Z = \text{LCS}(X, Y) \)

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

\[ z_1 \quad z_2 \quad z_3 \quad \ldots \quad z_{\ell-1} \quad z_\ell \]

\[ x_1 \quad x_2 \quad x_3 \quad x_4 \quad \ldots \quad x_{m-1} \quad x_m \]

\[ y_1 \quad y_2 \quad y_3 \quad y_4 \quad \ldots \quad y_{n-1} \quad y_n \]

\[ a \quad b \quad r \quad a \quad c \quad a \quad a \quad b \]

\[ a \quad b \quad r \quad a \quad c \quad a \quad a \quad b \]

\[ a \quad b \quad r \quad a \quad c \quad a \quad a \quad b \]

\[ x \]

\[ y \]

\[ Z = \text{LCS}(X_{m-1}, Y) \]
# BUILDING SOLUTIONS FROM SUBPROBLEMS

**EXAMPLE #3**

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>r</th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>x₁</td>
<td>x₂</td>
<td>x₃</td>
<td>x₄</td>
<td>...</td>
<td>xₘ₋₁</td>
<td>xₘ</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>z</th>
<th>b</th>
<th>r</th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>d</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>y₁</td>
<td>y₂</td>
<td>y₃</td>
<td>y₄</td>
<td>...</td>
<td>yₙ₋₁</td>
<td>yₙ</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>r</th>
<th>a</th>
<th>c</th>
<th>a</th>
<th>a</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>z₁</td>
<td>z₂</td>
<td>z₃</td>
<td>z₄</td>
<td>...</td>
<td>zₗ₋₁</td>
<td>zₗ</td>
<td></td>
</tr>
</tbody>
</table>

This might be the final a in Z

This might be the final a in Z

Or maybe this is...

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 $y_n$ \((x_m \neq y_n)\) \(Z = LCS(X_{m-1}, Y_{n-1})\)
- $z_\ell$ matches $x_m$ but not $y_n$ \((x_m \neq y_n)\) \(Z = LCS(X_m, Y_{n-1})\)
- $z_\ell$ matches $y_n$ but not $x_m$ \((x_m \neq y_n)\) \(Z = LCS(X_{m-1}, Y_n)\)
- $z_\ell$ matches **both** \((x_m = y_n)\) \(Z = LCS(X_{m-1}, Y_{n-1}) + z_\ell\)

- Let \(c(i, j) = |LCS(X_i, Y_j)|\)

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

- General cases

\[
\begin{align*}
  c(i, j) &= c(i - 1, j - 1) + 1 \quad \text{if } x_m = y_n \\
  c(i, j) &= \max\{c(i - 1, j - 1), c(i, j - 1), c(i - 1, j)\} \quad \text{if } x_m \neq y_n
\end{align*}
\]
Combining expressions

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

Can simplify!

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

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

\[
c(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
(c(i - 1, j - 1) + 1 & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\max\{c(i, j - 1), c(i - 1, j)\} & \text{if } i, j \geq 1 \text{ and } x_i \neq y_j
\end{cases}
\]
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>$X$</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></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$i = 0$</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>$j = 0$</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>$g$</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>$v$</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$c$</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$e$</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$k$</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$s$</td>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$t$</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Algorithm: LCS1\((X = (x_1, \ldots, x_m), Y = (y_1, \ldots, y_n))\)

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

- 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 \((c[m, n])\);
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 $c[i, j]$

$$
\begin{array}{ll}
0 & \text{if } i = 0 \text{ or } j = 0 \\
\min\{c(i - 1, j - 1), c(i - 1, j), c(i, j - 1)\} & \text{if } i, j \geq 1 \text{ and } x_i = y_j \\
\end{array}
$$

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 $i, j - 1$)

In our example table we just draw an arrow to the entry...

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 $\pi$

```
1  LCS2(X[1..m], Y[1..n])
2     c = new array[0..m][0..n]
3     \pi = new array[0..m][0..n]
4
5     for i = 0..m do c[i][0] = 0
6     for j = 0..n do c[0][j] = 0
7
8     for i = 1..m
9         for j = 1..n
10            if X[i] = Y[j]
11                c[i][j] = c[i-1][j-1] + 1
12                \pi[i][j] = "IJ"
13            else if c[i][j-1] > c[i-1][j]
14                c[i][j] = c[i][j-1]
15                \pi[i][j] = "J"
16            else /\ c[i][j-1] <= c[i-1][j]
17                c[i][j] = c[i-1][j]
18                \pi[i][j] = "I"
19
20     return c, \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 $i - 1, j$)
Suppose $X = \text{gdvegta}$ and $Y = \text{gvcekst}$.

<table>
<thead>
<tr>
<th></th>
<th>$i = 0$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$Y$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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></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>v</td>
<td>2</td>
<td>0</td>
<td>↑1</td>
<td>↑1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
<td>0</td>
<td>↑1</td>
<td>↑1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
<td>0</td>
<td></td>
<td>↑2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>k</td>
<td>5</td>
<td>0</td>
<td>↑1</td>
<td>↑1</td>
<td>↑2</td>
<td>↑3</td>
<td>↑3</td>
<td>↑3</td>
</tr>
<tr>
<td>s</td>
<td>6</td>
<td>0</td>
<td>↑1</td>
<td>↑1</td>
<td>↑2</td>
<td>↑3</td>
<td>↑3</td>
<td>↑3</td>
</tr>
<tr>
<td>t</td>
<td>7</td>
<td>0</td>
<td>↑1</td>
<td>↑1</td>
<td>↑2</td>
<td>↑3</td>
<td>↑3</td>
<td>↑3</td>
</tr>
</tbody>
</table>

Done:
- seq = gvet
- seq = vet

LCS = gvet

How to obtain LCS = gvet from this table?

Example

This is a
this “a”
is not in
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)
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>

| Final recurrence relation |
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 $t$.

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

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

Final recurrence relation
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 $N[1 ... n, 0 ... T]$:

- 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)

$$N[i,t] = \begin{cases} 
\min\{j + N[i-1, t-jd_i] : 0 \leq j \leq |t/d_i|\} & \text{if } i > 2 \\
t & \text{if } i = 1, \text{OR } t = 0 
\end{cases}$$
FILLING THE ARRAY $N[1 \ldots n, 0 \ldots T]$:

No data dependencies on any other array cells.

$i$-axis (coin type)

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

$t$-axis (target sum remaining)

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

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

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

$t$-axis (target sum remaining)

Consider cell $N[i, t]

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

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

\[
N[i,t] = \begin{cases} 
\min\{j + N[i-1, t-jd_i] : 0 \leq j \leq \lfloor t/d_i \rfloor\} & \text{if } i \geq 2 \\
0 & \text{if } i = 1.
\end{cases}
\]
Exercise for later:
compute the correct output without using $J[i, t]$, (i.e., using only $N, d, T$)

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

We start at $J[n][T] = \# \text{ of coins of type } d_n \text{ used in the optimal solution}$

```
CoinChangingDP_coins(d[1..n], J[1..n][0..T])

counts = new array[1..n]
t = T
for i = n..1
    counts[i] = J[i][t]
    t = t - counts[i]*d[i]
return counts
```
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 = 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

Unit cost computational model is reasonable here

Consider instance $I = (d, T)$

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

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

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

If $T$ is small, this is much better than brute force

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

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

```c
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 \[
\begin{align*}
f_1 &\leftarrow \text{RecFib}(n - 1) \\
f_2 &\leftarrow \text{RecFib}(n - 2)
\end{align*}
\]
\[f ← f_1 + f_2\]
M[n] ← f
return (f);
```

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

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