Computing binomial coefficients, 1

\[ \text{binom}(n, k) = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n; \\ \text{binom}(n-1, k-1) + \text{binom}(n-1, k), & \text{otherwise.} \end{cases} \]

Computing binomial coefficients, 2

\[ \text{binom}(n, k) = \begin{cases} 1, & \text{if } k = 0 \text{ or } k = n; \\ \text{binom}(n-1, k-1) + \text{binom}(n-1, k), & \text{otherwise.} \end{cases} \]

Computing binomial coefficients, 3

**Before Memoization**

```
function binom(n, k)
    if k = 0 or k = n then return 1
    else return binom(n - 1, k - 1) + binom(n - 1, k)
end binom
```

**After Memoization**

```
function binom(n, k)
    for m ← 0, 1, ..., n do
        b[m, 0] ← 1; b[m, m] ← 1
    for m ← 2, 3, ..., n do
        for ℓ ← 1, 2, ..., m - 1 do
            b[m, ℓ] ← b[m - 1, ℓ] + b[m - 1, ℓ - 1]
        return helper(n, k)
end binom
```

**After Memoization (Continued)**

```
function helper(m, ℓ)
    if b[m, ℓ] = 0 then
        b[m, ℓ] ← helper(m - 1, ℓ - 1)
        return helper(m - 1, ℓ)
    else
        return b[m, ℓ]
end helper
```

Trace binom(5,3)
Computing binomial coefficients, 4

Building the Table Directly

function \text{binom}(n,k)
for \(m \leftarrow 0, 1, \ldots, n\) do
    \(b[m,0] \leftarrow 1\); \(b[m,m] \leftarrow 1\)
for \(m = 2, 3, \ldots, n\) do
    for \(\ell = 1, 2, \ldots, m-1\) do
        \(b[m,\ell] \leftarrow b[m-1,\ell-1] + b[m-1,\ell]\)
return \(b[n,k]\)

Trace \text{binom}(5,3)

Computing binomial coefficients, 5

Going from a recursion to a table-building computation.

Step 1. Give a recursive definition.
(For many problems, this is the hard part.)

Step 2. Memoize to exploit repeated subproblems.
(If there are few repeated subproblems, memoization will not help.)

Step 3. Build the table directly to cut down overhead.
(If the answer depends on a small part of the table, the recursion can be faster.)

Making Change—Again, 1

The Making Change Problem (MCP)

Given: coin denominations \(d_1 < d_2 < \cdots < d_k\) and an amount \(a\).
Find: the smallest collection of coins that is worth amount \(a\).

Example

\(d_1 = 1, d_2 = 4, d_3 = 6; \ a = 8\).
\(\triangle\) The optimal choice is \(\{4, 4\}\).
\(\triangle\) The greedy algorithm produces \(\{6, 1, 1\}\).

The Optimal Substructure of MCP

If you use a \(d_i\)-coin in an optimal solution of the MCP for \(a\),
then the rest of the coins give an opt. soln of the MCP for \(a - d_i\).
(Why?)

Making Change—Again, 2

The Making Change Problem (MCP)

Given: coin denominations \(d_1 < d_2 < \cdots < d_k\) and an amount \(a\).
Find: the smallest collection of coins that is worth amount \(a\).

\[
\text{mcn}(a) \equiv \begin{cases} 
0, & \text{if } a = 0; \\
1 + \min \{ \text{mcn}(a - d_i) : d_i \leq a \ & \text{& } 1 \leq i \leq k \}, & \text{if } a > 0.
\end{cases}
\]

Example

\[
\text{mcn}(0) = 0, \quad \text{mcn}(1) = 1 + \min \{ \text{mcn}(0) \} = 1, \\
\text{mcn}(2) = 1 + \min \{ \text{mcn}(1) \} = 2, \\
\text{mcn}(3) = 1 + \min \{ \text{mcn}(2) \} = 3, \\
\text{mcn}(4) = 1 + \min \{ \text{mcn}(3), \text{mcn}(0) \} = 1, \\
\text{mcn}(5) = 1 + \min \{ \text{mcn}(4), \text{mcn}(1) \} = 2, \\
\text{mcn}(6) = 1 + \min \{ \text{mcn}(5), \text{mcn}(2), \text{mcn}(0) \} = 1, \\
\text{mcn}(7) = 1 + \min \{ \text{mcn}(6), \text{mcn}(3), \text{mcn}(1) \} = 2, \\
\text{mcn}(8) = 1 + \min \{ \text{mcn}(7), \text{mcn}(4), \text{mcn}(2) \} = 2.
\]
Making Change—Again, 3

The Making Change Problem (MCP)

Given: coin denominations $d_1 < d_2 < \cdots < d_k$ and an amount $a$.
Find: the smallest collection of coins that is worth amount $a$.

$$mcn(a) = \begin{cases} 0, & \text{if } a = 0; \\ 1 + \min \{ mcn(a - d_i) ; d_i \leq a & \text{and } 1 \leq i \leq k \}, & \text{if } a > 0. \end{cases}$$

```python
function mcn(d1, ..., dk; a) integer array num[0..a]
for d' ← 1 to a do // Trace mcn(1, 4, 6; 8)
    num[a'] ← ∞
    for i ← 1 to k do
        if d_i ≤ a' then num[a'] ← min(num[a'], 1 + num[a' - d_i])
    return num[a]
```

Making Change—Again, 4

Reconstructing the Solution to the MCP

Given: $num[i, a']$ (the min number of coins of denominations $d_1, \ldots, d_i$ need to make change for amount $a'$ where $0 \leq i \leq k$ and $0 \leq a' \leq a$)
Find: What coins make up the optimal solution.

```python
function reconstruct(d1, ..., dk; a, num[0..k,0..a])
    coins ← the empty list
    a' ← a; i ← k
    while a' > 0 do
        while a' > 0 do
            if (d_i ≤ a' & num[i, a'] ≠ num[i - 1, a'])
                then Add i to the coins list; a' ← a' - d_i
            else i ← i - 1
        return coins
```

Longest Increasing Subsequences, 1

Definition
Suppose $S = a_1, \ldots, a_n$ is a sequence of numbers.

(a) A subsequence of $S$ is a sequence of numbers $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ such that $1 \leq i_1 < i_2 < \cdots < i_k \leq n$.

(b) Such a subsequence is increasing when $a_{i_1} < a_{i_2} < \cdots < a_{i_k}$.

Longest Increasing Subsequence Problem (LISP)

Given: A sequence of numbers.
Find: An increasing subsequence of maximal length.

Example
For $S = 5, 2, 8, 6, 3, 6, 9, 7$; a longest increasing subsequence is: $2, 3, 6, 9$. 
Longest Increasing Subsequences, 3

Longest Increasing Subsequence Problem (LISP)

Given: A sequence of numbers. Find: A max-length increasing subsequence.

Given \( S = a_1, \ldots, a_n \), we can turn this into a graph problem as follows:

Let \( V = \{1, \ldots, n\} \), \( E = \{(i, j) : i < j \land a_i < a_j\} \), and \( G = (V, E) \).

\( G \) is a dag. \textbf{(Why?)}

\( \ast \), a longest increasing sequence in \( S \) \( \equiv \) a longest path in \( G \).

Example (for \( S = 5, 2, 8, 6, 3, 6, 9, 7 \))

![Graph Diagram](image)

### Transition Table

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 1 )</th>
<th>( 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Longest Increasing Subsequences, 3

Longest Increasing Subsequence Problem (LISP)

Given: A sequence of numbers. Find: A max-length increasing subsequence.

\( G = (V, E) \), where \( V = \{1, \ldots, n\} \), \( E = \{(i, j) : i < j \land a_i < a_j\} \).

\[ L(j) = \text{the length of a longest increasing subseq. ending at } j \]
\[ = 1 + \max\{L(i) : (i, j) \in E\}. \]

Convention: \( \max(\emptyset) = 0. \text{ (So, for example, } L(1) = 1.\text{)} \)

Example (for \( S = 5, 2, 8, 6, 3, 6, 9, 7 \))

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 1 )</th>
<th>( 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Transition Table

<table>
<thead>
<tr>
<th>( i )</th>
<th>( 1 )</th>
<th>( 2 )</th>
<th>( 3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>5</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Longest Increasing Subsequences, 3

Longest Increasing Subsequence Problem (LISP)

*Given:* A sequence of numbers.  
*Find:* A max-length increasing subsequence.

\[ G = (V, E), \text{ where } V = \{1, \ldots, n\}, E = \{(i, j) : i < j \& a_i < a_j \}. \]

\[ L(j) = \text{ the length of a longest increasing subseq. ending at } j \]
\[ = 1 + \max\{ L(i) \mid (i, j) \in E \}. \]

**Convention:** \( \max(\emptyset) = 0. \)  
*(So, for example, \( L(1) = 1. \))

**Example (for \( S = 5, 2, 8, 6, 3, 6, 9, 7 \))**

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

---

Longest Increasing Subsequences, 3

Longest Increasing Subsequence Problem (LISP)

*Given:* A sequence of numbers.  
*Find:* A max-length increasing subsequence.

\[ G = (V, E), \text{ where } V = \{1, \ldots, n\}, E = \{(i, j) : i < j \& a_i < a_j \}. \]

\[ L(j) = \text{ the length of a longest increasing subseq. ending at } j \]
\[ = 1 + \max\{ L(i) \mid (i, j) \in E \}. \]

**Convention:** \( \max(\emptyset) = 0. \)  
*(So, for example, \( L(1) = 1. \))

**Example (for \( S = 5, 2, 8, 6, 3, 6, 9, 7 \))**

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

---

Longest Increasing Subsequences, 3

Longest Increasing Subsequence Problem (LISP)

*Given:* A sequence of numbers.  
*Find:* A max-length increasing subsequence.

\[ G = (V, E), \text{ where } V = \{1, \ldots, n\}, E = \{(i, j) : i < j \& a_i < a_j \}. \]

\[ L(j) = \text{ the length of a longest increasing subseq. ending at } j \]
\[ = 1 + \max\{ L(i) \mid (i, j) \in E \}. \]

**Convention:** \( \max(\emptyset) = 0. \)  
*(So, for example, \( L(1) = 1. \))

**Example (for \( S = 5, 2, 8, 6, 3, 6, 9, 7 \))**

<table>
<thead>
<tr>
<th>( i )</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>( a_i )</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
Longest Increasing Subsequences, 3

Longest Increasing Subsequence Problem (LISP)

Given: A sequence of numbers.
Find: A max-length increasing subsequence.

\[ G = (V, E), \text{ where } V = \{ 1, \ldots, n \}, E = \{ (i, j) : i < j \& a_i < a_j \}. \]

\[ L(j) = \text{the length of a longest increasing subseq. ending at } j \]
\[ = 1 + \max\{ L(i) : (i, j) \in E \}. \]

Convention: \( \max(\emptyset) = 0. \) (So, for example, \( L(1) = 1. \))

Example (for \( S = 5, 2, 8, 6, 3, 6, 9, 7 \))

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_i )</td>
<td>5</td>
<td>2</td>
<td>8</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>( L(i) )</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>( \text{prev} )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Optimal Substructure

A problem has optimal substructure when an optimal solution is made up of optimal solutions to its subproblems.

Examples

(a) Shortest paths in a graph.
(b) Making change.
(c) ...

Non-examples

(a) Longest paths in a graph.
(b) Cheapest airline ticket from A to B.

Edit Distance, 1

Edit Operations:

- Insert a character
- Delete a character
- Substitute a character

The Edit Distance Problem

The edit distance between strings \( x[1..m] \) and \( y[1..n] \)
= the minimal number of edit operations to change \( x[1..m] \) to \( y[1..n] \).

\[ \text{SNOWY} \xrightarrow{\text{ins}} \text{SUNOWY} \xrightarrow{\text{sub}} \text{SUNNWY} \xrightarrow{\text{del}} \text{SUNNY} \]

<table>
<thead>
<tr>
<th>( S )</th>
<th>( U )</th>
<th>( N )</th>
<th>( O )</th>
<th>( W )</th>
<th>( Y )</th>
<th></th>
</tr>
</thead>
</table>

To solve this with dynamic programming, we have to figure out good subproblems.

Edit Distance, 2

The Edit Distance Problem

The edit distance between strings \( x[1..m] \) and \( y[1..n] \)
= the minimal number of edit operations to change \( x[1..m] \) to \( y[1..n] \).

The \( E(i, j) \) Problem

What is the edit distance of \( x[1..i] \) and \( y[1..j] \).

Goal: \( E(m, n) \).
Strategy: Solve \( E(i, j) \) for \( i = 1, \ldots, m, \ j = 1, \ldots, n. \)

\[ E(i, j) = \left\{ \begin{array}{l} \text{some combination of the solutions} \\ \text{to smaller } E(i', j') \text{ problems} \end{array} \right. \]

Observation

In an optimal alignment for \( E(i, j) \) the last column must look like:

\[
\begin{array}{c|c|c|c|c}
| x_i | & \text{or} & | y_j | & \text{or} & | x_j | \\
\hline
& - & & - & & \end{array}
\]
Edit Distance, 3

The $E(i,j)$ Problem
What is the edit distance of $x[1..i]$ and $y[1..j]$ (where $i, j > 0$)?

$$
\begin{align*}
\text{Case } & \lessdot x_i : \quad \text{The cost is } 1 + E(i-1,j). \\
\text{Case } & \lessdot y_j : \quad \text{The cost is } 1 + E(i,j-1). \\
\text{Case } & \lessdot x_i, y_j : \\
\text{Subcase } & \lessdot x_i = y_j : \quad \text{The cost is } E(i-1,j-1). \\
\text{Subcase } & \lessdot x_i \neq y_j : \quad \text{The cost is } 1 + E(i-1,j-1). \\
\end{align*}
$$

Edit Distance, 4

The $E(i,j)$ Problem
What is the edit distance of $x[1..i]$ and $y[1..j]$ (where $i, j > 0$)?

$$
\therefore E(i,j) = \min(1+E(i-1,j), 1+E(i,j-1), \text{diff}(i,j) + E(i-1,j-1))
$$

$$
\text{diff}(i,j) = \begin{cases} 
0, & \text{if } x_i = y_j; \\
1, & \text{otherwise.}
\end{cases}
$$

Edit Distance, 5

The $E(i,j)$ Problem
What is the edit distance of $x[1..i]$ and $y[1..j]$ (where $i, j > 0$)?

Compute $E(i,j) = \text{the edit distance of } x[1..i] \text{ and } y[1..j]$.

```python
function edcost(x[1..m], y[1..n])
for i <- 0 to m do E[i,0] <- i
for j <- 1 to n do E[0,j] <- j
for i <- 1 to m do
for j <- 1 to n do
E[i,j] <-
\quad \min(1+E[i-1,j],
\quad 1+E[i,j-1],
\quad \text{diff}(i,j) + E[i-1,j-1])
\quad return E[m,n]
```

See http://www.alistair.c.com/search/it/ 
Sawmahtbui/edit_distance.html for a nice animation.
The underlying dependencies dag and a cost-6 path.