• Homework 3
  • Due Friday at 2pm
• Midterm: Tuesday, March 8, 2016
Exercise: Show how to compute the maximum subrectangle sum of the $n \times n$ int[][] mat in $O(n^4)$ time.

Exercise: Show how to compute the maximum subrectangle sum of the $n \times n$ int[][] mat in $O(n^3)$ time.

Exercise: Given an array int[][] mat of zeros and ones, compute the size of the largest rectangle (in area) that is entirely composed of ones.
• Memoized Fibonacci sequence
• “Top-down”: uses recursion
• “Find the answer to my problem, only solve subproblems as needed”
• “Bottom-up” approach: iteratively build solution
• “Find solutions for subproblems, and build solution for my problem”

<table>
<thead>
<tr>
<th>$x$</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>$F[x]$</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

• A faster way to compute Fibonacci numbers?
\[
\begin{pmatrix}
1 & 1 \\
1 & 0 \\
\end{pmatrix}
\begin{pmatrix}
F(n) \\
F(n-1) \\
\end{pmatrix}
=
\begin{pmatrix}
F(n+1) \\
F(n) \\
\end{pmatrix}
\]
• Recall exponentiation by squaring to compute in O(lg n) time
\[
\begin{pmatrix}
1 & 1 \\
1 & 0 \\
\end{pmatrix}^n
\begin{pmatrix}
1 \\
0 \\
\end{pmatrix}
=
\begin{pmatrix}
F(n+1) \\
F(n) \\
\end{pmatrix}
\]
• An even faster way to compute Fibonacci numbers
• Closed-form solution:

\[ F_n = \frac{1}{\sqrt{5}} \left[ \left( \frac{1+\sqrt{5}}{2} \right)^n - \left( \frac{1-\sqrt{5}}{2} \right)^n \right] \]

• Reference
**Exercise:** Suppose you are given a list of widths and heights of blocks that each has unit depth. You can put one block on top of another exactly when the top block has width and height at most as large as the block below it. What is the height of the tallest tower of blocks that can be made?
Given an array of numbers,
- int a = new int[n];

Find the longest increasing subsequence
- a subset $s$ such that $a[s[0]] < a[s[1]] < a[s[2]] < \ldots < a[s[m-1]]$
- And $|s|$ is maximal

<table>
<thead>
<tr>
<th>i</th>
<th>0</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>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[i]</td>
<td>4</td>
<td>15</td>
<td>11</td>
<td>2</td>
<td>7</td>
<td>19</td>
<td>15</td>
<td>20</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>
• Algorithm:
  • Build a table $s$ so that $s[i]$ is the longest increasing subsequence ending with $a[i]$
    • Includes $a[i]$ in the sequence
  • $s[i] = \max \left\{ s[j] + 1 \text{ if } a[i] > a[j] \text{ for } j = 1, 2, \ldots, i-1 \right\}$
Bottom Up: Longest Increasing Subsequence

\[
\begin{array}{cccccccccc}
 i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
 a[i] & 4 & 15 & 11 & 2 & 7 & 19 & 15 & 20 & 9 & 3 \\
 s[i] & 1 & 2 & 2 & 1 & & & & & & \\
 p[i] & -1 & 0 & 0 & -1 & & & & & & \\
\end{array}
\]

\[
s[i] = \max \left\{ s[j] + 1 \text{ if } a[i] > a[j] \text{ for } j = 1, 2, \ldots, i-1 \right\}
\]

\[
p[i] = j, \text{ or } -1 \text{ if no } j \text{ exists}
\]
• A faster solution to LIS than $O(n^2)$
  • Greedy and divide & conquer approach

• Intuition:
  • For input list $A$, maintain an array $L$ where $L[i]$ is the smallest ending value of all length-$i$ LISs found so far
    • Note that because $L$ is sorted, we can binary search over it
  • Iterate over each element $A[i]$ in the input list $A$
    • Find last $L[i]$ such that $L[i] < A[i]$
    • Set $L[i+1] = A[i]$
  • Max length of $L$ is the answer
<table>
<thead>
<tr>
<th>A \ L</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7</td>
<td>-7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>-7</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>-7</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-7</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-7</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-7</td>
<td>2</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-7</td>
<td>1</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-7</td>
<td>1</td>
<td>2</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-7</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
• You are going shopping for a wedding
  • There are $1 \leq C \leq 20$ types of garments
  • Each garment type has $1 \leq K \leq 20$ items, each type with a different price
    • e.g., 2 types of shirts, 3 different belts, etc.
  • You have a budget of $1 \leq M \leq 200$
  • Task: Buy one of each type of garment, spending as much money as possible without going over budget
  • What is the maximum possible amount to spend?
## Wedding Shopping

<table>
<thead>
<tr>
<th>Item</th>
<th>M = 100</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Shirt</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Pants</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belt</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Shoes</td>
<td>50</td>
<td>14</td>
<td>23</td>
<td>8</td>
</tr>
</tbody>
</table>

**Answer:** 75

<table>
<thead>
<tr>
<th>Item</th>
<th>M = 20</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Shirt</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Pants</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belt</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

**Answer:** 19

(Multiple solutions)
### Wedding Shopping

**Answer:** No solution!

<table>
<thead>
<tr>
<th>Item</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>M = 5</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Shirt</strong></td>
<td>6</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>Pants</strong></td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Belt</strong></td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

- You can’t afford pants
- Top-down solution
- Subproblem: if we select item k for type c and price p, then the optimal solution for the other garment types must also be optimal for budget $M - p$
- State space becomes limited to $M \times C$
- Recurrence:
  - Try each item k with price p for type c
  - Solution is $\max(\text{solution for type } c+1, \text{budget } M - p)$
  - Basis state: money left after buying the last garment
• Bottom-up DP formulation
  • Define a table \( dp[moneyRem][i] \) where \( i \) is the garment index
  • \( dp[moneyRem][i] \) is true if it's possible to end up with \( moneyRem \) by choosing garments 1 through \( i \)
• This works because we're simply marking all possible states that are reachable
• The answer to “is it possible to have \( x \) remaining on \( m \) garments?” is yes if and only if \( dp[x][m] \) is true
• Solution: \( \min(moneyRem \text{ for all } dp[moneyRem][c] \text{ that are true}) \)
Example:

<table>
<thead>
<tr>
<th>Item</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = 12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shirt</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Pants</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Belt</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

\( dp[\text{moneyRem}][i] \) is true if it's possible to end up with \( \text{moneyRem} \) by choosing garments 1 through \( i \)

<table>
<thead>
<tr>
<th>i (garment index)</th>
<th>0</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>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>moneyRem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( dp )</td>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
More well-known problem with similar solution: zero-one knapsack

- Also known as subset sum
- **Problem:** given n items, each with value $v_i$ and weight $w_i$, and a knapsack that can hold weight $S$, what is the maximum value of items you can carry?

**Solution:**

1. $val(id, 0) = 0$ // if remW = 0, we cannot take anything else
2. $val(n, remW) = 0$ // if id = n, we have considered all items
3. if $W[id] > remW$, we have no choice but to ignore this item $val(id, remW) = val(id + 1, remW)$
4. if $W[id] \leq remW$, we have two choices: ignore or take this item; we take the maximum $val(id, remW) = \max(val(id + 1, remW), V[id] + val(id + 1, remW - W[id]))$
• Problem: given two strings, find the length of the longest common subsequence
• “Longest common subsequence”
  • You are given two strings, $S_1$ and $S_2$
  • $dp[i][j]$ is the length of the longest common subsequence after $i$ characters of $S_1$ and $j$ characters of $S_2$
  • $dp[i][j] = \max(dp[i-1][j], dp[i][j-1], dp[i-1][j-1] + 1 \text{ if } S_1[i] = S_2[j])$
• $dp[i][j] = \max(dp[i-1][j], dp[i][j-1], dp[i-1][j-1] + 1 \text{ if } S_1[i] = S_2[j])$

• Why this works (sketch):
  • $dp[i][j]$ contains the largest common subsequence between $S_1[0:i]$ and $S_2[0:j]$.
  • At each step, “consume” a character from either string, incrementally build upon the best answer
    • If possible (and if the answer is better), “consume” a character from both and increment the subsolution
  • Therefore the largest common subsequence is in $dp[n][m]$ where $|S_1| = n$ and $|S_2| = m$
• **Top-down DP**
  - Pro: Natural transformation from recursion
  - Pro: Computes subproblems only when necessary
  - Con: May be slower due to recursion overhead
  - Con: Uses exactly $O(\text{states})$ table size

• **Bottom-up DP**
  - Pro: Faster if many sub-problems visited, no recursion
  - Pro: Can save memory space
  - Con: May not be as intuitive
  - Con: Fills values for all the states, does not skip unreachable states
Exercise: You are given 12 points in the plane with coordinates \((x_0, y_0), \ldots, (x_{11}, y_{11})\). Compute the shortest route (in Euclidean distance) beginning at \((x_0, y_0)\) that visits each of the 12 points exactly once.
Competitive Programming 3.5