6.006 Recitation
Build 2008.38
6.006 Proudly Presents

- Warmup: Maxing out sums
- Fun: Tetris pwnage
- Bonus:
  - Pwn Mario v2: mushrooms, monsters
Max. Sum Sub-array

- a is a list of real numbers
- want i, j so that $\sum a[i:j]$ is as large as possible
- want to compute this as fast as possible

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>a[i]</td>
<td>31</td>
<td>-41</td>
<td>59</td>
<td>26</td>
<td>-53</td>
<td>58</td>
<td>97</td>
<td>-93</td>
<td>-23</td>
<td>84</td>
</tr>
</tbody>
</table>

- answer for this case
  - i = 2
  - j = 6
  - sum = 187
Max. Sum Sub-array: Naive Solution

Max. Sum Sub-array: Naive Solution

<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>31</td>
<td>-41</td>
<td>59</td>
<td>26</td>
<td>-53</td>
<td>58</td>
<td>97</td>
<td>-93</td>
<td>-23</td>
<td>84</td>
</tr>
</tbody>
</table>

- \( \text{max}_\text{sum}, \text{max}_i, \text{max}_j = 0, 0, 0 \)
- for \( i \) in 0:len(a)  
  - for \( j \) in \( i: \) len(a)  
    - if \( \text{max}_\text{sum} < \sum a[i:j] \)  
      - \( \text{max}_\text{sum}, \text{max}_i, \text{max}_j = \sum a[i:j], i, j \)
Running Time for Naive Solution

• $i, j$ go through all possible intervals $a[i:j]$  
• $O(N^2)$ intervals  
• Evaluating $\sum a[i:j]$ at each interval  
• $O(N)$ work per interval  
• $O(N^3)$ total
Max. Sub-Array: Smarter Solution A

- Notice that $\sum a[i:j] = \sum a[i:j-1] + a[j]$
- Rewrite inner block to eliminate computing $\sum a[i:j]$, replace with a running sum
- Running time: work per interval drops to $O(1)$, total work drops to $O(N^2)$
Max. Sub-Array: Smarter Solution B

• Hints
  • we’re using a ‘fancy’ data structure
  • \( s[i] = \Sigma a[0:i] \)
  • again, we’re trying to cut the work per interval
Max. Sub-Array: Smarter Solution B

- Notice that $\Sigma a[i:j] = \Sigma a[0:j] - \Sigma a[0:i-1]$
- Pre-compute $\Sigma a[0:i]$ into $s[i]$
- Rewrite the inner block of the naive algorithm to compute $\Sigma a[i:j]$ in $O(1)$
- Running time: again $O(N^2)$
Max. Sub-Array: Uber-Pro Solution Hint

• Hint: we will go through the motions of DP, but arrive at a very interesting conclusion

• Hint II: so start thinking of the optimal sub-structure
Max. Sub-Array: Uber-Pro Solution 1

• Problem: the max. sum sub-array in a

• Sub-problem
  \[ s[i] = \text{max. sum sub-array ending at } a[i] \]

• Optimal sub-structure: if the max. sub-array includes \( a[i] \), it starts with the max. sum sub-array ending at \( a[i] \)
Max. Sub-Array: Uber-Pro Solution II

• $s[i] = \max(s[i - 1] + a[i], a[i])$

• So we keep adding to the current sub-array until the sub-array sum becomes negative

• Discussion: bottom-up implementation, constant-space implementation
Tetris pwnage
Tetris Pwnage: This is How Pros Do It

- For each piece
  1. Instantly rotate and move the piece
  2. Let the piece drop
- Don’t care about making lines disappear; if you pwn it, they will come
- Last for as many pieces as possible
Tetris Pwnage: Formal Problem

- Board of width N
- K pieces, each of its own shape
- Must fit as many pieces as possible
- For each piece, must return rotation and position where it falls from
Tetris Pwnage:
The Vision

This is a game. Act accordingly.
Tetris Pwnage: The Approach

1. Find all the variables that make a position
2. Reduce the position representation
3. Use BFS
4. Figure out a way to do this bottom-up
Tetris Pwnage: The Solution I

• A configuration is the # of pieces on the board and the “skyline”

• Pieces can’t go through other pieces, so it doesn’t matter what’s under the “skyline”

• Example at the right: 6 pieces, (5 4 4 5 1 4 4)
**Tetris Pwnage: The Solution II**

- Bottom-line solution: configurations of P pieces only depend on configurations of P-1 pieces

- \(d[p][skyline] = 1\) if can use p pieces to achieve the given skyline
Bonus Discussion: Mario v2

- Monsters 1...m patrol platforms
- Monster i moves between platforms m[i][0], m[i][1]...m[i][mp_i], 1 ≤ mp_i ≤ 4
- Special platforms contain mushrooms
- Mushroom state is an extra life - lost when in the same position as a monster