Last Time

• Greedy Algorithms
  – Take the best available solution at each step
  – Need to show correctness and optimality

• Proof by Exchange Argument
  – Minimum Average Waiting Time problem
  – Minimum Spanning Tree problem
Dynamic Programming

• **Dynamic Programming** – Solving a problem by breaking it down divide-and-conquer style into a set of smaller programs.
  – Only solve each subproblem once, no repeat computations.

• Basic approach:
  1. Design a top-down recursive algorithm.
  2. Convert it to a bottom-up iterative algorithm.
Toy Example: Fibonacci

• Top-down recursion:

```c
int fibo(int val) {
    if (val == 0 || val == 1) {
        return 0;
    } else {
        return fibo(val-1) + fibo(val-2);
    } //if-else
} //fibo
```
Toy Example: Fibonacci

- Bottom-up iterative:

```c
int fibo(int stop) {
    int val[0] = 0, val[1] = 0;
    for (int i = 2; i <= stop; i++) {
        val[i] = val[i-1] + val[i-2];
    } //for
    return val[stop];
} //fibo
```
Toy Example: Fibonacci

• So what did we do?
  – Start with a recursive algorithm that was $O(2^n)$.
  – Pull out the base cases and initialize them with appropriate values.
  – Remove the recursion and turn it into a for loop, counting up from the base cases instead of counting down to them.
Longest Common Subsequence

• **Input:**
  – String $A = A_1 A_2 \ldots A_m$
  – String $B = B_1 B_2 \ldots B_n$

• **Output:**
  – Length of the longest common subsequence (can skip letters).

• Example input: $A = \text{xyyzzxyyz}$, $B = \text{xyxxxyz}$
• Example output: 5 (xyxyz)
Longest Common Subsequence

- Start with a recursive algorithm.

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>X</th>
<th>m-1</th>
<th>X</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>X</td>
<td>n-1</td>
<td>X</td>
<td>n</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>X</td>
<td>m-1</td>
<td>X</td>
<td>m</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>Y</td>
<td>n-1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Longest Common Subsequence

• Start with a recursive algorithm.

```cpp
LCS(int m, int n) {
    if (m == 0 || n == 0) {
        return 0;
    } else if (A[m] = B[n]) {
        return LCS(m-1, n-1) + 1;
    } else {
        return max(LCS(m, n-1), LCS(m-1, n));
    } //if-else
} //LCS
```
Longest Common Subsequence

• Convert to iterative, bottom-up algorithm

```plaintext
LCS(int m, int n) {
    for (int i = 0; i <= m; i++)
        LCS[i][0] = 0;
    for (int i = 0; i <= n; i++)
        LCS[0][i] = 0;
    for (int i = 1; i <= m; i++)
        for (int j = 1; j <= n; j++)
            if (A[i] == B[j])
                LCS[i][j] = LCS[i-1][j-1]+1;
            else
                LCS[i][j] = max(LCS[i][j-1], LCS[i-1][j]);
    return LCS[m][n];
} //LCS
```
Longest Common Subsequence

- Solve $LCS(\text{xyxx}, \text{ yxxxy})$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

The longest common subsequence is $\text{xy}$. 
Longest Common Subsequence

- Can we recover that longest subsequence?

<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>x</th>
<th>x</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Dynamic Programming

• Alternate approach: the Pruning Method.
  – At every decision, two options: (1) take the choice, or (2) don’t take the choice.
  – Build a tree based off of the two options.

• Pruning rules:
  1. If a node ever goes beyond the bounds of the problem, prune it.
  2. If there are 2+ nodes on the same level of the tree with the same value, prune all but one of them.
Subset Sum Problem

• **Input:**
  – Coin values $v_1, v_2, \ldots, v_n$
  – Maximum value $L$

• **Output:**
  – Change for $L$ cents, if possible

• Example Input: 7, 2, 11, 9, 16, 13; $L = 41$
• Example Output: 11, 16, 14 <or> 2, 9, 16, 14
Subset Sum Problem

• Tree construction: Either take coin $v_i$ or don’t.
Subset Sum Problem

• Pruning Rules:
  1. If the coins at a node sum to $>L$, then prune the subtree at that node.
  2. If there are 2+ nodes on the same level with coins summing to the same value, prune all but one of those nodes.
Subset Sum Problem

• \( v_1 = 1, v_2 = 1, v_3 = 3 \); \( L = 3 \)
Subset Sum Problem

• So what does pruning get us?
  – There are at most \((n + 1)\) levels in the tree if we have \(n\) coins.
  – Each level can have at most \(L+1\) sums, because pruning guarantees unique sums capped at \(L\).

• So we fill an \((n + 1) \times (L + 1)\) matrix!
Subset Sum Problem

SubsetSum(int[] coins, int L) {
    A[0,0] = 1;
    for (int lvl = 0; lvl <= n; lvl++) {
        for (int s = 0; s <= L; s++) {
            if (A[lvl, s]) {
                A[lvl+1, s] = 1;  //left child
                A[lvl+1, s+coin[lvl+1]] = 1 //r
            }
        }
    }
} //SubsetSum

A[lvl,s] = 1 if ∃ subset of coins 1..lvl that sums to s
Subset Sum Problem

• Solve $v_1 = 1, v_2 = 2, v_3 = 3, v_4 = 4; L = 7$

<table>
<thead>
<tr>
<th>lvl</th>
<th>s</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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</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>1</td>
<td>1</td>
<td>1</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>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

04/21/2015 Dynamic Programming
Any Questions?