CS 97SI: INTRODUCTION TO PROGRAMMING CONTESTS

Jaehyun Park

Today's Lecture: Dynamic Programming

- What is DP?
- 1-dimensional DP
- 2-dimensional DP
- Interval DP
- Tree DP
- Subset DP

What is DP?

- Wikipedia definition: “a method for solving complex problems by breaking them down into simpler subproblems”

- This definition will make sense once we see some examples
  - Actually, we’ll only see problem solving examples today

Steps for solving DP problems

- 1. Define subproblems
- 2. Write down the recurrence that relates subproblems
- 3. Recognize and solve the base cases

- Each step is very important!

1-dimensional DP Example

- Problem: given n, find the number of different ways to write n as the sum of 1, 3, 4
- Example: for $n = 5$, the answer is 6
  - $5 = 1+1+1+1+1$
    $= 1+1+3$
    $= 1+3+1$
    $= 3+1+1$
    $= 1+4$
    $= 4+1$

1-dimensional DP Example

- Define subproblems
  - Let D_n be the number of ways to write n as the sum of 1, 3, 4
- Find the recurrence
  - Consider one possible solution $n = x_1 + x_2 + \cdots + x_m$
  - If x_m is 1, the rest of the terms must sum to $n - 1$
  - Thus, the number of sums that end with $x_m = 1$ is equal to D_{n-1}
  - Take other cases into account ($x_m = 3$, $x_m = 4$)

1-dimensional DP Example

- Find the recurrence
  - $D_n = D_{n-1} + D_{n-3} + D_{n-4}$
- Solve the base cases
  - $D_0 = 1$
  - $D_n = 0$ for all negative n
  - Alternatively, can set: $D_0 = D_1 = D_2 = 1, D_3 = 2$
- We're basically done!

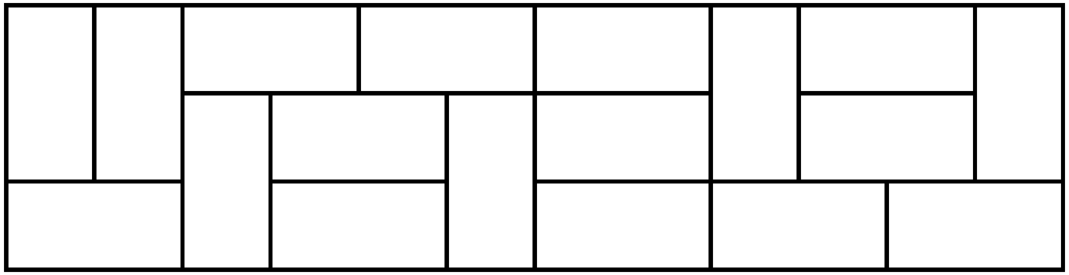
Implementation

```
D[0] = D[1] = D[2] = 1; D[3] = 2;
for(i = 4; i <= n; i++)
     D[i] = D[i-1] + D[i-3] + D[i-4];
```

- Too short that it's almost disappointing ☺
- Extension: solving this for huge n
  - Recall the matrix form of Fibonacci numbers

POJ 2663: Tri Tiling

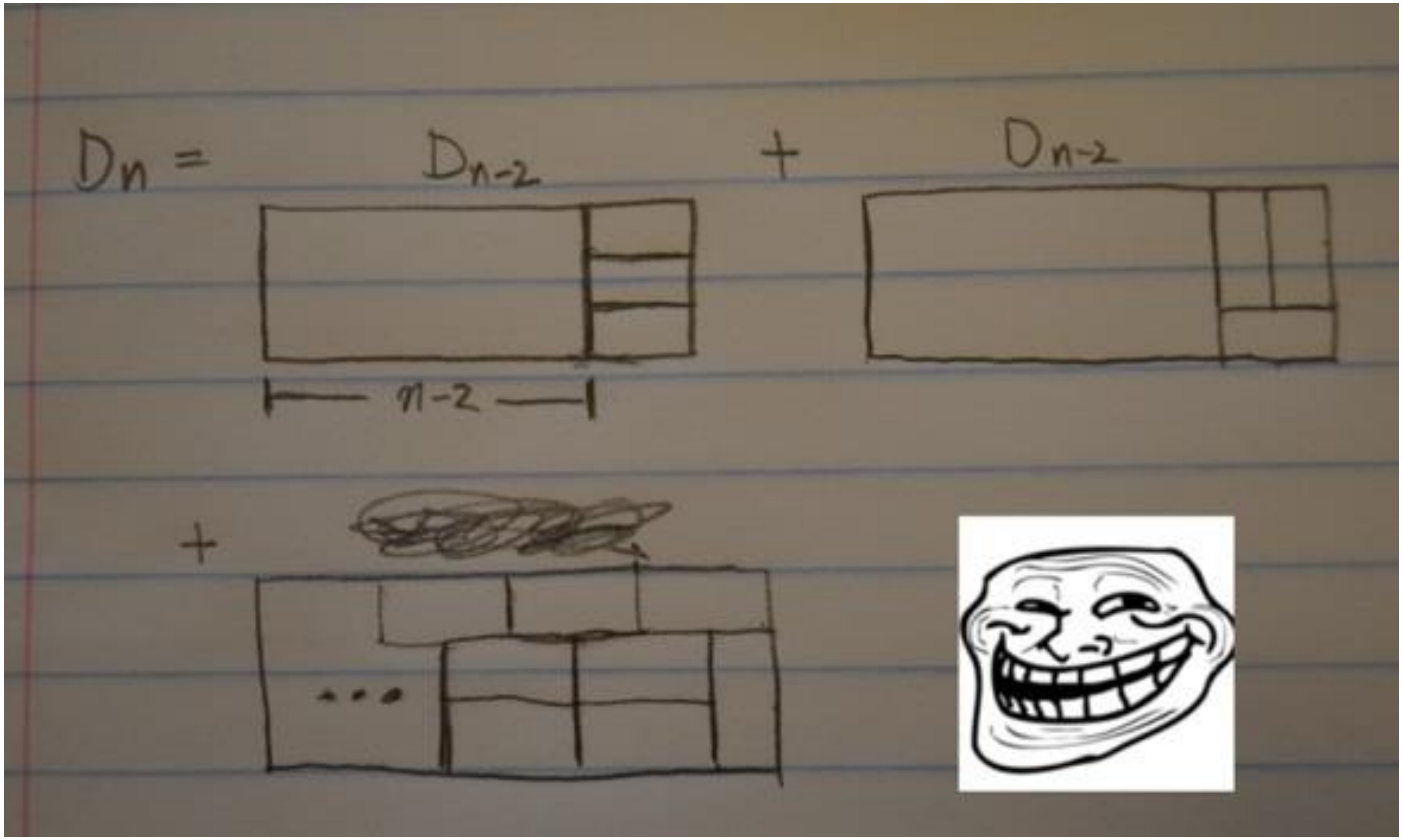
- Given n, find the number of ways to fill a $3 \times n$ board with dominoes
- Here is one possible solution for $n = 12$



POJ 2663: Tri Tiling

- Define subproblems
  - Define D_n as the number of ways to tile a $3 \times n$ board
- Find recurrence
  - Uuuhhhh…

Troll Tiling

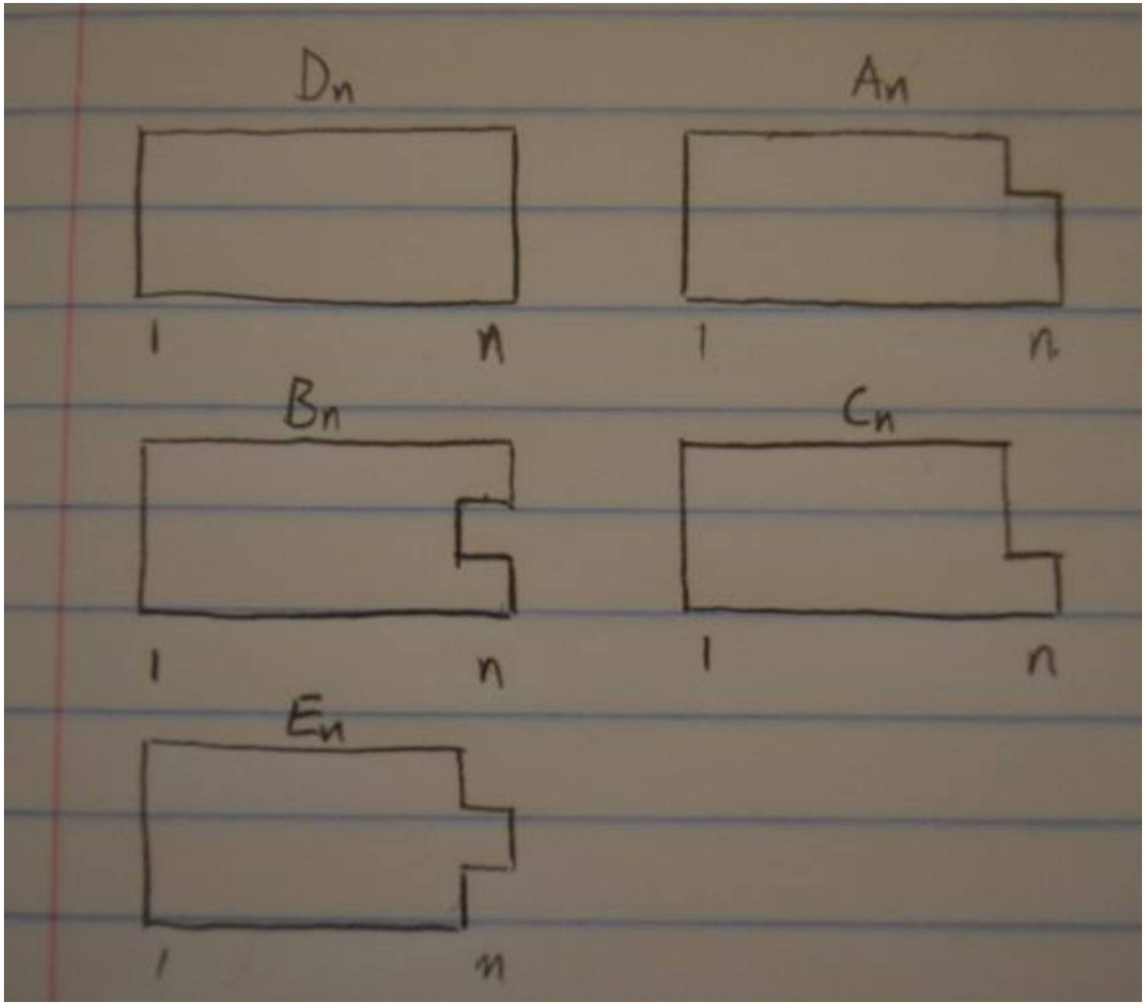


Defining Subproblems

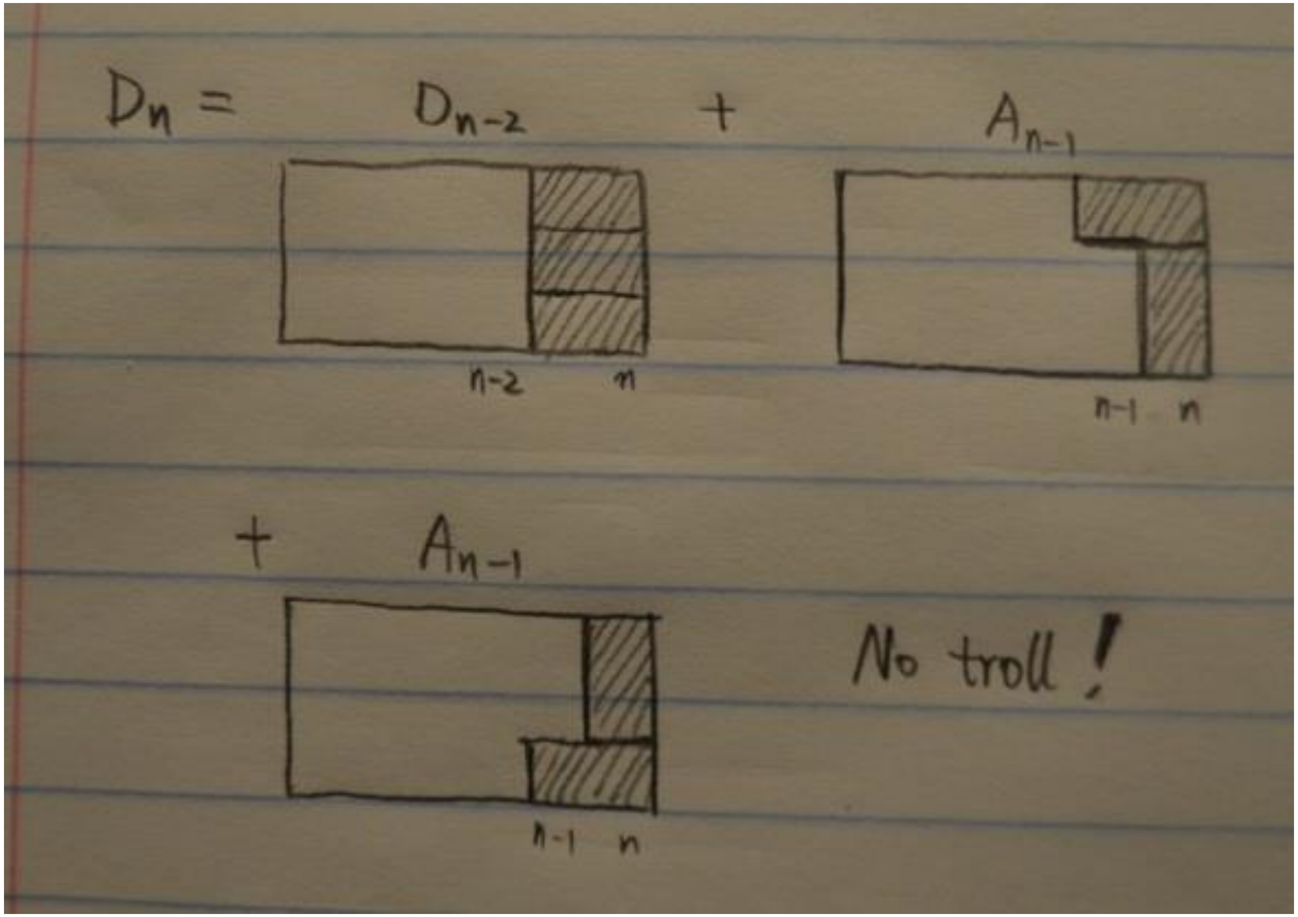
- Obviously, the previous definition didn't work very well
  - D_n's don't relate in simple terms

- What if we introduce more subproblems?

Defining Subproblems



Finding Recurrences



Finding Recurrences

- Consider different ways to fill the nth column
  - And see what the remaining shape is
- Exercise:
  - Finding recurrences for A_n, B_n, C_n
  - Understanding why E_n is always zero
- Extension: solving the problem for $n \times m$ grids, where n is small ($n \leq 10$)
  - How many subproblems should we consider?

2-dimensional DP Example

- Problem: given two strings x and y, find the longest common subsequence (LCS) and print its length
- Example:
  - x: A**BC**BD**AB**
  - y: **B**D**CAB**C
  - “BCAB” is the longest subsequence found in both sequences, so the answer is 4

Solving the LCS Problem

- Define subproblems
  - Let $D_{i,j}$ be the length of the LCS of $x_{1\ldots i}$ and $y_{1\ldots j}$
- Find the recurrence
  - If $x_i = y_j$, they both contribute to the LCS
    - $D_{i,j} = D_{i-1,j-1} + 1$
  - Otherwise, either x_i or y_j does **not** contribute to the LCS, so one can be dropped
    - $D_{i,j} = \max\left(D_{i-1,j}, D_{i,j-1}\right)$
- Find and solve the base cases: $D_{i,0} = D_{0,j} = 0$

Implementation

```
for(i = 0; i <= n; i++) D[i][0] = 0;
for(j = 0; j <= m; j++) D[0][j] = 0;
for(i = 1; i <= n; i++) {
  for(j = 1; j <= m; j++) {
    if(x[i] == y[j]) D[i][j] = D[i-1][j-1] + 1;
    else D[i][j] = max(D[i-1][j], D[i][j-1]);
  }
}
```

- Again, very short ☺

Interval DP Example

- Problem: given a string $x = x_{1 \ldots n}$, find the minimum number of characters that need to be inserted to make it a palindrome
- Example:
  - x: Ab3bd
  - Can get “**d**Ab3b**A**d” or “A**d**b3bd**A**” by inserting 2 characters (one ‘d’, one ‘A’)

Interval DP Example

- Define subproblems
  - Let $D_{i,j}$ be the minimum number of characters that need to be inserted to make $x_{i\dots j}$ into a palindrome
- Find the recurrence
  - Consider an optimal solution $y_{1\dots k}$ for $x_{i\dots j}$
  - Either $y_1 = x_i$ or $y_k = x_j$ (why?)
  - $y_{2\dots k-1}$ is then an optimal solution for $x_{i+1\dots j}$ or $x_{i\dots j-1}$ or $x_{i+1\dots j-1}$
    - Last case possible only if $y_1 = y_k = x_i = x_j$

Interval DP Example

- Find the recurrence
  - $D_{i,j} = 1 + \min\left(D_{i+1,j}, D_{i,j-1}\right)$, if $x_i \neq x_j$
  - $D_{i,j} = D_{i+1,j-1}$, if $x_i = x_j$
- Find and solve the base cases
  - $D_{i,i} = D_{i,i-1} = 0$ for all i

- The entries of D must be filled in increasing order of $j - i$

Implementation

```
// fill in base cases here
for(t = 2; t <= n; t++)
     for(i = 1, j = t; j <= n; i++, j++)
          // fill in D[i][j] here
```

- Note how we use an additional variable t to fill the table in correct order
- And yes, for loops can work with multiple variables

An Alternate Solution

- Reverse x to get x^R
- The answer is $n - L$, where L is the length of the LCS of x and x^R
- Think about why this works

Tree DP Example

- Problem: given a tree, color nodes black as many as possible without coloring two adjacent nodes
- Subproblems:
  - First, we arbitrarily decide the root node r
  - B_v: the optimal solution for a subtree having v as the root, where we color v black
  - W_v: the optimal solution for a subtree having v as the root, where we don't color v
  - The answer is $\max(B_r, W_r)$

Tree DP Example

- Find the recurrence
  - Crucial observation: once v's color is determined, its subtrees can be solved independently
  - If v is colored, its children must not be colored
    - $B_v = 1 + \sum_{u:v'\text{s child}} W_u$
  - If v is not colored, its children can have any color
    - $W_v = \sum_{u:v'\text{s child}} \max(B_u, W_u)$

- Base cases: leaf nodes

Subset DP Example

- Problem: given a weighted graph with n nodes, find the shortest path that visits every node exactly once (Traveling Salesman Problem)

- Wait, isn't this an NP-hard problem?
  - Yes, but we can solve it in $O(n^2 2^n)$ time
  - Note: brute force algorithm takes $O(n!)$ time

Subset DP Example

- Define subproblems
  - $D_{S,v}$: the length of the optimal path that visits every node in S exactly once and ends at v
  - There are approximately $n2^n$ subproblems
  - The answer is $\min_{v \in V} D_{V,v}$, where V is the set of nodes

- Let's solve the base cases first
  - For each node v, $D_{\{v\},v} = 0$

Subset DP Example

- Find the recurrence
  - Consider a path that visits all nodes in S exactly once and ends at v
  - The path must have come from some node u in $S - \{v\}$ right before arriving v
  - And that subpath has to be the optimal one that covers $S - \{v\}$
  - $D_{S,v} = \min_{u \in S - \{v\}} \left(D_{S-\{v\},u} + \text{cost}(u, v) \right)$

Working with Subsets

- We work with all subsets of V, so it's good to have a nice representation of them
- Number the nodes from 0 and use bitmask!
  - Use an integer to represent a subset
  - If the ith (least significant) digit is 1, i is in the subset
  - If the ith digit is 0, i is not in the subset
  - e.g. 010011 in binary represent a set {0, 1, 4}

Using Bitmasks

- Union of two sets `x` and `y`: `x | y`
- Intersection: `x & y`
- Symmetric difference: `x ^ y`
- Singleton set $\{i\}$: `1 << i`
- Membership test: `x & (1 << i) == 0`

- Can easily work with a small set and its subsets

Conclusion

- Wikipedia definition: “a method for solving complex problems by breaking them down into simpler subproblems”
  - Does this make sense now?

- Remember the three steps!
  - Defining subproblems
  - Finding recurrences
  - Solving the base cases