Dynamic Programming
Day 2

- Priyansh Agarwal

Solving Homework Problems

Problem 1: Link

- State:
  - dp[i] = number of ways to get sum == i
- Transition:
  - dp[i] = dp[i - 1] + dp[i - 2]...... + dp[i - 6]
- Final Subproblem:
  - dp[n]

Problem 2: Link

- State:
  - dp[k] = min coins required to make sum == k
- Transition:
  - $dp[k] = 1 + \min\{dp[k - coins_i]\}$ $(0 <= i <= n - 1)$
- Final Subproblem:
  - dp[x]

Problem 3: Link

- State:
  - dp[i] = number of ways to make sum == i
- Transition:
  - dp[i] = sum of dp[i - $coins_j$] ($0 \le j \le n - 1$)
- Final Subproblem:
  - dp[x]

Recursive vs Iterative DP (in Day 2)

Recursive	Iterative
Slower (runtime)	Faster (runtime)
No need to care about the flow	Important to calculate states in a way that current state can be derived from previously calculated states
Does not evaluate unnecessary states	All states are evaluated
Cannot apply many optimizations	Can apply optimizations

General Technique to solve any DP problem

1. State

   Clearly define the subproblem. Clearly understand when you are saying dp[i][j][k], what does it represent exactly

2. Transition:

   Define a relation b/w states. Assume that states on the right side of the equation have been calculated. Don't worry about them.

3. Base Case

   When does your transition fail? Call them base cases answer before hand. Basically handle them separately.

4. Final Subproblem

   What is the problem demanding you to find?

Solving Classical Problems

Problem 1: Link

- State:
  - dp[x] = min steps to convert x to 0
- Transition:
  - dp[x] = min(dp[x - some digit of x]) + 1
- Base Case:
  - dp[0] = 0
- Final Subproblem:
  - dp[n]

Problem 2: Link

- State:
  - dp[i][j] = number of ways to go from (0, 0) to (i, j)
- Transition:
  - dp[i][j] = dp[i - 1][j] + dp[i][j - 1]
- Base Case:
  - step1: dp[0][0] = 1, step2: dp[i][j] = 0, when (i, j) is a trap
- Final Subproblem:
  - dp[n - 1][n - 1]

Problem 2: Link

- State:
  - dp[i][j] = number of ways to go from (i, j) to (n - 1, n - 1)
- Transition:
  - dp[i][j] = dp[i + 1][j] + dp[i][j + 1]
- Base Case:
  - dp[n - 1][n - 1] = 1, dp[i][j] = 0, when (i, j) is a trap
- Final Subproblem:
  - dp[0][0]

Space Optimization

- What other state does our current state depend on?
- Do we need answers to all subproblems at all times?
- Well, let’s store only relevant states then!
- But wait, does this always work?
  - What if the final subproblem requires all the states?
  - What if we need to backtrack? [more on this in later lectures]

- Fibonacci Problem
  - dp[i] depends on dp[i - 1], dp[i - 2]
- Grid Problem
  - dp[i][j] depends on dp[i + 1][j], dp[i][j + 1]
- Dice Problem in Homework
  - dp[i] depends on dp[i - k]   (1 <= k <= 6)