# Dynamic Programming
# Day 3

- Priyansh Agarwal

# Solving Homework Problems

# Problem 1: [Link]

- State:
  - dp[k][i] => current sum = k, standing at $i^{th}$ coin, number of ways to make sum = X
- Transition:
  - $dp[k][i] = dp[k + coin_i][i] + dp[k][i + 1]$
- Base Case:
  - $dp[X][i] = 1$, $dp[> X][i] = 0$, $dp[Y][n] = 0$
- Final Subproblem:
  - dp[0][0]

# Problem 2: Link

- State:
  - dp[i][j] = max pages you can read when allowed to read the first i books and allowed cost is j
- Transition:
  - dp[i][j] = max(dp[i - 1][j], pages[i - 1] + dp[i - 1][j - cost[i - 1]])
- Base Case:
  - dp[0][anything] = 0
- Final Subproblem:
  - dp[n][x]

# Problem 3: Link

- State:
  - dp[i][x] = number of ways to fill [0..i] s.t the ith element is x
- Transition:
  - dp[i][x] = dp[i - 1][x - 1] + dp[i - 1][x] + dp[i - 1][x + 1]
- Base Case:
  - dp[0][x] = (arr[0] == 0 | | arr[0] == x ? 1 : 0)
- Final Subproblem:
  - dp[n - 1][1] + dp[n - 1][2] .......... + dp[n - 1][m]

# State Optimization

- Ask yourself do you need all the parameters in the dp state?
- If you have dp[a][b][c], and a + b = c, do you need to store c as a parameter or can you just compute it on spot?
- If you can compute a parameter in dp state from other parameters, no need to store it.
- Which parameters should you remove? Highest

# Transition Optimization

- Observe the transition equation.
- Can you do some pre-computation to evaluate the equation faster?
- Using clever observations.
- Using range query data structures

# Solving More Classical Problems

# Problem 1: Link

- State:
  - dp[i][0] = number of ways to fill rows from (i to n) such that last row had 2 growing blocks
  - dp[i][1] = number of ways to fill rows from (i to n) such that last row had a (1 * 2) type of growing block

# Problem 1: Link

- Transition:
  - for dp[i][0]
    - pos1: close both blocks -> dp[i + 1][0] + dp[i + 1][1]
      - pos11: start 2 blocks of 1 * 1 each = dp[i + 1][0]
      - pos12: start 1 block of 1 * 2 = dp[i + 1][1]
    - pos2: don't close any of them -> dp[i + 1][0]
    - pos3 and 4: close one of them -> 2 * dp[i + 1][0]

  - dp[i][0] = pos1 + pos2 + pos3 + pos4 =
      - = dp[i + 1][0] + dp[i + 1][1] + dp[i + 1][0] + 2 * dp[i + 1][0]
      - = 4 * dp[i + 1][0] + dp[i + 1][1]

# Problem 1: Link

- Transition:
    - for dp[i][1]
        - pos1: close the complete block:
            - pos11: start 2 blocks of 1 * 1 each = dp[i + 1][0]
            - pos12: start 1 block of 1 * 2  = dp[i + 1][1]
        - pos2: don't close the block -> dp[i + 1][1]

    - dp[i][1] = pos1 + pos2
        - = dp[i + 1][0] + dp[i + 1][1] + dp[i + 1][1]
        - = 2 * dp[i + 1][1] + dp[i + 1][0]

# Problem 1: Link

- Base case:
    - dp[n - 1][0] = 1
    - dp[n - 1][1] = 1

- Final subproblem:
    - dp[0][1] + dp[0][0]