Dynamic Programming
Day 4

- Priyansh Agarwal

Solving Homework Problems

Problem 1: Link

- State:
  - dp[i][j] = edit distance b/w string A[i:n] and B[j:m]
- Transition:
  - dp[i][j] = (a[i] == b[j]) ? dp[i + 1][j + 1] : 1 + min(dp[i][j + 1], dp[i + 1][j], dp[i + 1][j + 1])
- Base Case:
  - dp[n][j] = m - j, dp[i][m] = n - i
- Final Subproblem:
  - dp[0][0]

Problem 2: Link

- State:
  - dp[i][j] = min cuts to make rectangle of (i * j) into squares
- Transition:
  - dp[i][j] = min(1 + min(dp[i][k] + dp[i][j - k]), 1 + min(dp[k][j] + dp[i - k][j]))
- Base Case:
  - dp[i][i] = 0
- Final Subproblem:
  - dp[n][m]

Problem 3: Link

- State:
  - dp[i] = set of sums possible with coins from 1 to i^{th} coin
- Transition:
  - dp[i] => add or not add the current coin to all the sums possible in the previous state
- Base Case:
  - dp[0] = {0}
- Final Subproblem:
  - dp[n]

Problem 3: Link

```
void solve(){
    vector<bool> possible(1e5 + 1, false);
    possible[0] = true;
    for(int i = 1; i <= n; i++){
        vector<bool> current(1e5 + 1, false);
        int currentDenomination = coins[i - 1];
        for(int j = 0; j <= 1e5; j++){
            if(possible[j]){
                current[j + currentDenomination] = true;
                current[j] = true;
            }
        }
        possible = current;
    }
    for(int i = 1; i <= 1e5; i++){
        if(possible[i])
            cout << i << " ";
    }
    cout << endl;
}
```

Answer Construction

- Grid problem: Find the actual path with the minimum sum.
- Minimizing coins problem: Find the actual choice of coins.
- At every state we are making some optimal choice.
    - If we store this choice, we can be sure that if we are at any state we know what is the best choice.
    - Start from the state that contains your final subproblem and keep making the best choice (which was already stored) until you reach the end.

Answer Construction - Grid Problem

```cpp
int n = 3, m = 3;
vector<vector<int>> grid(3, vector<int>(3));
vector<vector<pair<int, int>>> dp(n, vector<pair<int, int>>(m, {-1, 0}));
// 0 -> take a down direction
// 1 -> take a right direction
int f(int i, int j){
    if(i == n || j == m)
        return 1e9;
    if(i == n - 1 && j == m - 1)
        return grid[n - 1][m - 1];
    if(dp[i][j].first != -1)
        return dp[i][j].first;

    int ans1 = f(i + 1, j);
    int ans2 = f(i, j + 1);
    if(ans1 < ans2){
        dp[i][j].second = 0;
    }else{
        dp[i][j].second = 1;
    }
    return dp[i][j].first = grid[i][j] + min(ans1, ans2);
}
```

Answer Construction - Grid Problem

```
void solve(){
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            cin >> grid[i][j];
        }
    }
    cout << f(0, 0) << nline;
    pair<int, int> current = {0, 0};
    while(current != mp(n - 1, m - 1)){
        cout << current.first << " " << current.second << nline;
        if(dp[current.first][current.second].second == 0)
            current.first++;
        else
            current.second++;
    }
    cout << current.first << " " << current.second << nline;
}
```

Non Standard Problem 1: Link

```
State:
dp[i][x].first = is it possible to make a xor of x from the first i
rows

dp[i][x].second = choice of column that you made

Transition:
dp[i][x].first = true if dp[i - 1][x ^ arr[i][j]].first is true for
any j from 1 to m

dp[i][x].second = that particular j for which dp[i - 1][x ^ arr[
i][j]].first = true

Base Case:
dp[0][0].first = true, dp[0][i].first = false
```

Non Standard Problem 1: Link

```
Final Subproblem:
if(dp[n][any value > 0].first is true):
    ans = true
else
    ans = false

Ans construction:
    if(dp[n][k].first is true){
        vector<int> choices(n, -1);
        int row = n, xor_value = k;
        while(i != 0){
            int choice = dp[row][xor_value].second;
            int new_xor = xor_value ^ arr[row][choice];
            int new_row = row - 1;
            choices[row - 1] = choice + 1;
            xor = new_xor;
            row = new_row;
        }
        for(auto i : choices)
            cout << i << " ";
    }
```

Representing non-integer parameters

- How will you store the dp states if instead of integer parameters you had a string or a vector or a map or any complex data type?
- Use a map instead of an array.
- Tradeoff map<pair<int, string>> DP or vector<map<string>> DP

Non Standard Problem 2: Link (HW)

- State:
  -
- Transition:
  -
- Base Case:
  -
- Final Subproblem:
  -