Dynamic Programming
Class 4

- Priyansh Agarwal

Problem 1: Link

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

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

Problem 2: Link

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

DP with Bitmasking

- Bitmasks
- Basic operations on Bitmasks
- Limitations on “N” (You will need 2^N integers to represent all the subsets)

Problem 1:

Given a list of points on a 2D plane, rearrange these points in any way such that in the final permutation of points, the sum of distances of the adjacent elements is minimized.

Constraints: [N <= 15], [-1e9 <= Xi, Yi <=1e9]

Points : [{0, 0}, {5, 6}, {1, 2}]

Best permutation -> [{0, 0}, {1, 2}, {5, 6}]]

Ans = Dist(P1, P3) + Dist(P3, P2)

Problem 1: TC: $O(n^3 2^n)$, SC: $O(n^2 2^n)$

```
state:
    dp[i][bitmask][last element] = minimum sum of distances in the suffix [i... n - 1]
    such tha bitmask represents the elements in the first i - 1 elements and last elements
    represents the last point

transition:
    check for jth point from (0 to n - 1)
    can you pick the jth point as the ith element in the final array or not

    if(bitmask & (1 << j)){ whether jth bit is set or not
        continue;
    }else{
        dp[i][bitmask][last element] = min(dp[i][bitmask][last element],
        (bitmask != 0 ? dist(j, last element) : 0) + dp[i + 1][bitmask | (1 << j)][j]
    }

base case:
    dp[n][(1 << n) - 1][anything] = 0

final subproblem
    dp[0][0][anything]
```

Problem 1: Code

```
int n;
int dp[n][1 << n][n]; // stored -1 everywhere initially
vector<Point> points(n);

int f(int i, int mask, int last){
    if(i == n)
        return 0;
    if(dp[i][mask][last] != -1)
        return dp[i][mask][last];
    int ans = INF;
    for(int j = 0; j < n; j++){
        if(mask & (1 << j))
            continue;
        ans = min(ans, f(i + 1, mask | (1 << j), j) + (i > 0 ? dist(j, last) : 0));
    }
    return dp[i][mask][last] = ans;
}
```

Problem 1: TC: $O(n^2 2^n)$, SC: $O(n*2^n)$

```
state:
    dp[bitmask][last element]
    i = set_bits(bitmask)
    = minimum sum of distances in the suffix [i... n - 1] such tha bitmask represents the
    elements in the first i - 1 elements and last elements represents the last point

transition:
    check for jth point from (0 to n - 1)
    can you pick the jth point as the ith element in the final array or not

    if(bitmask & (1 << j)){ whether jth bit is set or not
        continue;
    }else{
        dp[bitmask][last element] = min(dp[bitmask][last element],
        (bitmask != 0 ? dist(j, last element) : 0) + dp[bitmask | (1 << j)][j]
    }

base case:
    dp[(1 << n) - 1][anything] = 0

final subproblem
    dp[0][anything]
```

Problem 1: Optimized Code

```
int n;
int dp[1 << n][n]; // stored -1 everywhere initially
vector<Point> points(n);

int f(int mask, int last){
    int i = set_bits(mask);
    if(i == n)
        return 0;
    if(dp[mask][last] != -1)
        return dp[mask][last];
    int ans = INF;
    for(int j = 0; j < n; j++){
        if(mask & (1 << j))
            continue;
        ans = min(ans, f(mask | (1 << j), j) + (i > 0 ? dist(j, last) : 0));
    }
    return dp[mask][last] = ans;
}
```

Problem 2: Link (Homework)

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