`#include <senior week 11>`

DP Extra

Today, we'll go over some extra DP topics.

Game Theory

In game theory, given some current state of the game, we want to figure out who wins. In a two-player game, there are two positions: a *winning* position or a *losing* position.
A winning position means a position where the player who goes first wins.
A losing position means a position where the player who goes first loses, i.e. the person who goes later wins.

After handling bases cases, when we want to transition there are 2 possibilities:

1. If I can make a move and go to a losing position, I am currently in a winning position
2. If no matter what move I make, the game transitions to a winning position, I am currently in a losing position.

Because one must consider all possible next positions, generally we loop based on position rather than move. So it should look something like this:

for (position):
  for (move):
    consider move

Try this: [Educational DP Contest AtCoder K - Stones - DMOJ: Modern Online Judge]

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    int n, k; cin >> n >> k;
    bool dp[k+1];

    memset(dp,false,sizeof(dp));

    int stones[n];
    for (int i = 0; i < n; i++) cin >> stones[i];

    for (int i = 1; i <= k; i++) {
        for (int j = 0; j < n; j ++){
            int s = stones[j];
            if (i-s >= 0 && dp[i-s] == false) {
```

```
                dp[i] = true;
            }
        }
    }

    if (dp[k] == true) {
        cout << "First";
    } else {
        cout << "Second";
    }
    return 0;
}
```

DP w/ states

In DP, we don't know what our last move was, which is a problem when encountering situations where we have restrictions on possible moves. For example, not being able to use the same move twice in a row.

In this case, we want to assign specific states to a DP position. In general, we'll be filling out a DP[position][state].

The idea of states is to somehow update them to make sure we never reach an illegal position.

Question: Educational DP Contest AtCoder C - Vacation - DMOJ: Modern Online Judge

How can we assign states to make sure we never violate the rule of 'don't do the same activity twice in a row?'
ANS: have states 1,2,3 which represent activity A,B,C. Then we don't want the same activity twice, so transition looks like this:

DP[d][1] = max(DP[d-1][2],DP[d-1][3]) + activity A today
DP[d][2] = max(DP[d-1][1],DP[d-1][3]) + activity B today
DP[d][3] = max(DP[d-1][1],DP[d-1][2]) + activity C today

```
#include <bits/stdc++.h>

using namespace std;
```

```
int n,a,b,c;
int dp[100000][3];

int main() {

    cin >> n >> a >> b >> c;
    dp[0][0] = a;
    dp[0][1] = b;
    dp[0][2] = c;
    for (int i = 1; i < n; i ++){
        cin >> a >> b >> c;
        dp[i][0] = a + max(dp[i-1][1],dp[i-1][2]);
        dp[i][1] = b + max(dp[i-1][0],dp[i-1][2]);
        dp[i][2] = c + max(dp[i-1][0],dp[i-1][1]);
    }
    cout << max(dp[n-1][0],max(dp[n-1][1],dp[n-1][2]));

    return 0;
}
```

Interval DP

This is another popular type of DP. DP[left][right] = ans.

It's common in this type of problem to loop by range, so it looks like this:
for (range):
    for ([i to i+range]):
        consider range

When there are 2 lists, a common way to solve it is to use DP[a][b] = answer for list1[0…a] + list2[0…b].

[Educational DP Contest AtCoder L - Deque - DMOJ: Modern Online Judge]()

We can set DP[L][R] = (First player - Second player), or (Player about to move - other player). Then for each transition, we either take the left or the right most values.
So [L…R] ← [L+1…R] or [L…R-1].

If the first player (say A) takes from the left, we are considering [L+1…R]. Now it is the second player (say B)'s turn to move. Now it is B's turn for the array [L+1…R]. From our DP, we know the answer to this subproblem.So B-A = DP[L+1][R]. We want to solve for A-B though, so -DP[L+1][R] = A-B. Adding on the value taken, we have:

DP[L][R] = arr[L] - DP[L+1][R].

Similarly, the other case is DP[L][R] = arr[R] - DP[L][R-1].
So our transition is:

DP[L][R] = max(arr[L]-DP[L+1][R],arr[R]-DP[L][R-1]).

```
#include <bits/stdc++.h>

using namespace std;

int n, arr[3000];
long long dp[3000][3000];

int main() {

    cin >> n;
    for (int i = 0; i < n; i ++) {
        cin >> arr[i];
        dp[i][i] = arr[i];
    }

    for (int len = 1; len < n; len++){
        for (int i = 0; i < n-len; i++){
            dp[i][i+len] =
max(arr[i]-dp[i+1][i+len],arr[i+len]-dp[i][i+len-1]);
        }
    }

    cout << dp[0][n-1];

    return 0;
}
```

HW:
[DMOPC '18 Contest 3 P2 - Bob and French Class - DMOJ: Modern Online Judge]

[A Game - DMOJ: Modern Online Judge]
[IOI '96 P1 - A Game - DMOJ: Modern Online Judge] (Very similar to above, for the point farmers)

[COCI '21 Contest 1 #2 Kamenčići - DMOJ: Modern Online Judge] (may want to use the data struct `pair`)

Bonus: Longest Common Subsequence - DMOJ: Modern Online Judge