Kung Fu Training Problem

Problem Statement

A Ninja has an 'N' Day training schedule. He has to perform one of these three activities (Running, Fighting Practice, or Learning New Moves) each day. There are merit points associated with performing an activity each day. The same activity can't be performed on two consecutive days. We need to find the maximum merit points the ninja can attain in N Days.

We are given a 2D Array POINTS of size 'N*3' which tells us the merit point of specific activity on that particular day. Our task is to calculate the maximum number of merit points that the ninja can earn.

Editorial

- Greedy fails. Hence, try all possible cases i.e., recursion.
- 3 steps: interms of indices, do stuff, return the final value(max or min).
- There are n days, so indices from 0 to n-1.
- At a random index, in order to choose an action, we need to know what action is done in the prev day.
- If we write a function, we need `day` (the index as parameter) and the `last` as the yesterdays action. Note: we can also represent `last` as index.
- Start from any side, top-down or bottum-up.
- If we start from top:
    - So far, function: `f(day, last)`
    - Everytime we need the max merit from index `0` till index `day` provided the action done on the prev day is `last`.
    - If we start from day `n-1` and go down:
        - `f(2, 1)` represents max merit from index 0 till index 2 provided task with index 1 is done on **day 3. Note it is day 3.**
        - But for `f(n-1, ?)` : We need to have something that represents no task is done the prev day that is n th day. One way to represent no task as index 3 (which is not 0, 1, and 2).
    - We will be going all the way down till index 0.
- **Base case:** when day index is 0:
    - We need to perform a task that is not done on day 1 but is maximum among the rest 2.

  Code:

```
if(day == 0){
    int maxi = 0;
    for(int i = 0; i <= 2; i++){
        if(i != last) maxi = max(maxi, arr[0][i]);
    }
    return maxi;
}
```

- Normal case:
    - We need to find from index day till index 0 and add **all** merit points.

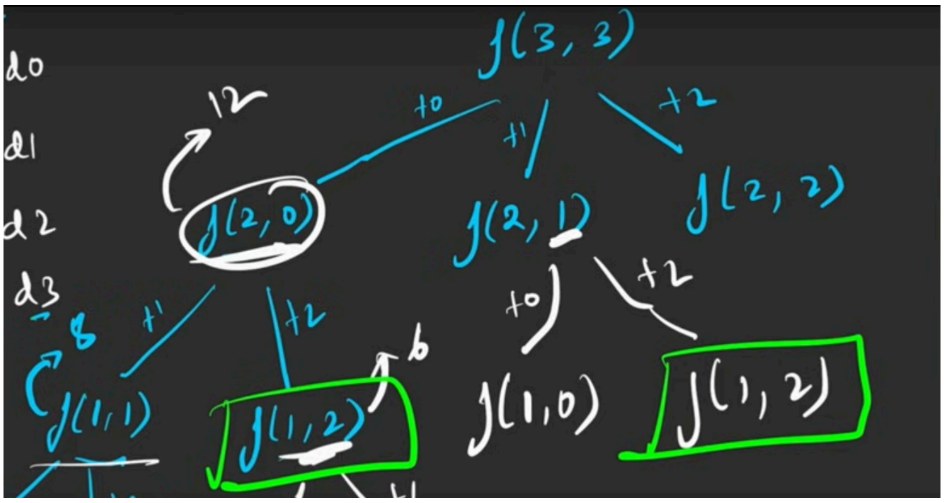
- At current index:
  - Perform a task making sure conditions met. Add the points. It is done for that index.
  - We need to do the same for all other prev days i.e., Call the function again for prev day and task `i` done on present day. Recursion will take care.

  ```
  maxi = 0;
  for(int i = 0; i <= 2; i++){
          if(i != last)
                  points = arr[day][i] + f(day - 1, i);
  }
  ```

- For all possible tasks done from that day till 0, we need to return the maximum points. Hence:

  ```
  maxi = max(maxi, points);
  ```

- Draw the recursion tree to understand overlapping subproblems:



Memoization:

- Changing parameters: day and last. Possible values for day: N; last: 4 (0, 1, 2, 3)
- Create a 2D dp array of size Nx4.
- Add the dp conditon that if it is solved, return the value.

Code:

```
int f(int day, int last, vector<vector<int>>& points, vector<vector<int>>& dp){
        if(day == 0){
                int maxi = 0;
                for(int task = 0; task <= 2; task++){
                        if(task != last) maxi = max(maxi, points[0][i]);
                }
                return maxi;
        }
```

```
        if(dp[day][last] != -1) return dp[day][last];

        maxi = 0;
        for(int task = 0; task <= 2; task++){
                if(task != last)
                        int result = points[day][task] + f(day-1, task, points, dp);
                        maxi = max(maxi, result);
        }
        return dp[day][last] = maxi;
}

int training(int n, vector<vector<int>>& points){
        vector<vector<int>> dp(n, vector<int>(4, -1));
        return f(n-1, 3, points, dp);
}
```

Tabulation

Same dp array size.

Base case:

```
dp[0][0] = max(points[0][1], points[0][2])
dp[0][1] = max(points[0][0], points[0][2])
dp[0][2] = max(points[0][0], points[0][1])
dp[0][3] = max(points[0][0], points[0][1] and points[0][2])
```

Final code:

```
int ninjaTraining(int n, vector<vector<int>>& points) {
  // Create a 2D DP (Dynamic Programming) table to store the maximum points
  // dp[i][j] represents the maximum points at day i, considering the last activity as j
  vector<vector<int>> dp(n, vector<int>(4, 0));

  // Initialize the DP table for the first day (day 0)
  dp[0][0] = max(points[0][1], points[0][2]);
  dp[0][1] = max(points[0][0], points[0][2]);
  dp[0][2] = max(points[0][0], points[0][1]);
  dp[0][3] = max(points[0][0], max(points[0][1], points[0][2]));

  // Iterate through the days starting from day 1
  for (int day = 1; day < n; day++) {
    for (int last = 0; last < 4; last++) {
      dp[day][last] = 0;
      // Iterate through the tasks for the current day
      for (int task = 0; task <= 2; task++) {
        if (task != last) {
          // Calculate the points for the current activity and add it to the
          // maximum points obtained on the previous day (recursively calculated)
          int activity = points[day][task] + dp[day - 1][task];
          // Update the maximum points for the current day and last activity
          dp[day][last] = max(dp[day][last], activity);
        }
      }
    }
  }

  // The maximum points for the last day with any activity can be found in dp[n-1][3]
  return dp[n - 1][3];
}
```

Space optimisation:

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

int ninjaTraining(int n, vector<vector<int>>& points) {
  vector<int> prev(4, 0);

  prev[0] = max(points[0][1], points[0][2]);
  prev[1] = max(points[0][0], points[0][2]);
  prev[2] = max(points[0][0], points[0][1]);
  prev[3] = max(points[0][0], max(points[0][1], points[0][2]));

  for (int day = 1; day < n; day++) {

    vector<int> temp(4, 0);
    for (int last = 0; last < 4; last++) {

      temp[last] = 0;
      for (int task = 0; task <= 2; task++) {

        if (task != last) {
          temp[last] = max(temp[last], points[day][task] + prev[task]);
        }
      }
    }
    prev = temp;
  }

  return prev[3];
}

int main() {
  vector<vector<int>> points = {{10, 40, 70},
                                {20, 50, 80},
                                {30, 60, 90}};
  int n = points.size();
  cout << ninjaTraining(n, points);
}
```