COMP3121 21T2 Assignment 3 Q4

Written by Zheng Luo (z5206267)

In order to determine the maximum total enjoyment over the entire day of N days and the sequence of activities choice each day, both days N and all 3 activities need to be considered as variables, hence solution will be constituted by using two while loops for specified day i and specified activity j. Consequently, the corresponding choice is choosing an activity between activity 1, 2, or 3 at each day. However there is one extra restriction for this question, that is same activities cannot be chosen for two days in a row. Assuming function $dp[i][j]$ can be defined as the maximum enjoyment by playing activity j at day i. The maximum total enjoyment can be determined by choosing the maximum activity enjoyment so far at each day in a double while loop. The pseudo code below can explain the algorithm in more detailed manner.

```
int calculateMaxEnjoyment(int[n][3] e) {
    int[n][3] dp = int[n][3];
    int prevChoice = 0;
    // Base case, when 0 days no activity yet.
    dp[0][j] = 0;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= 3; j++) {
            // The maximum amount of enjoyment TODAY,
            // if I played activity 1 yesterday,
            // and hence I will not able to play activity 1 today
            int activityEnjoy1 = dp[i-1][1] + e[i][j];
            // Same logic as above
            int activityEnjoy2 = dp[i-1][2] + e[i][j];
            int activityEnjoy3 = dp[i-1][3] + e[i][j];
            // Avaiable choices based on previous choice.
            if (prevChoice == 1) {
                // I can only choose activity 2 or 3 if I chose activity 1 yesterday.
                dp[i][j] = Math.max(activityEnjoy2, activityEnjoy3);
            }
            else if (prevChoice == 2) {
                dp[i][j] = Math.max(activityEnjoy1, activityEnjoy3);
            }
            else if (prevChoice == 3) {
                dp[i][j] = Math.max(activityEnjoy1, activityEnjoy2);
            }
            else {
                // There is no prev choice == 1st day.
                // I can choose any activity I want.
                dp[i][j] = Math.max(activityEnjoy1, activityEnjoy2, activityEnjoy3);
            }
        }
    }
    return Math.max(dp[n][1], dp[n][2], dp[n][3]);
}
```

Hence the function can calculate the maximal total enjoyment during the trip, the actual pattern of choosing each trip can be obtained by observing and recording the position of maximal dp value in last day, then move one day backward, until the first day has been reached. The overall time complexity for this algorithm is $O(n^2)$ since two loops have been used to produce the dp table.