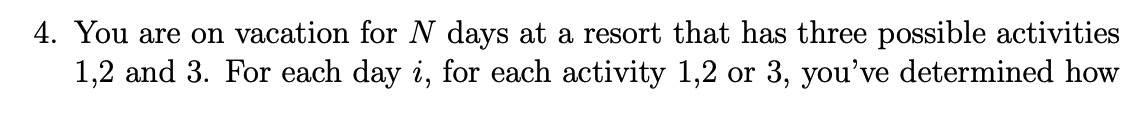
**COMP9101 Ass03**

**Name: Zanning Wang**

**zID: z5224151**



文本, 信件

描述已自动生成

**Q4.**

This problem similar to the Q1, we can also use dynamic programming to solve the question.

We use dp[i][j] to store the most enjoyment we get till the day i if we do activity j;

i∈[0, n], j∈[1, 3]

We know that the e(i, j) to represent current day of the enjoyment we get if we do activity j.

Therefore, the base case is:

dp[0][1] = e(0, 1)

dp[0][2] = e(0, 2)

dp[0][3] = e(0, 3)

we know that we should not do the same activity two days in a row, therefore there are three situations for the day i:

Suppose we do activity 1 on day i:

dp[i][1] = max (dp[i-1][2] + e(i, 1), dp[i-1][3] + e(i, 1))

Suppose we do activity 2 on day i:

dp[i][2] = max (dp[i-1][1] + e(i, 2), dp[i-1][3] + e(i, 2))

Suppose we do activity 3 on day i:

dp[i][3] = max (dp[i-1][1] + e(i, 3), dp[i-1][2] + e(i, 3))

for each algorithm, we should store each activity in a list called “res”, if the result of enjoyment is the highest, the res should be the sequence of activity we should do at each day.

the recursion should be

therefore for day N, the max enjoyment we may get should be max(dp[N][1], dp[N][2], dp[N][3])

the time complexity for this problem should be O(3N), which is O(N) at last.