Problem in Dynamic Programming

A. Madhur

1 Introduction

In what follows, several problems related to "Dynamic Programming" (`DP`) and their solutions are discussed. These problems have been taken from several websites and are their property.

1.1 *368.* Largest Divisible Subset

Source: *https://leetcode.com/problems/largest-divisible-subset*
Problem Statement: Given a set of distinct positive integers `nums`, return the largest subset answer such that every pair (`answer[i]`, `answer[j]`) of elements in this subset satisfies: `answer[i] % answer[j] == 0`, or `answer[j] % answer[i] == 0`. If there are multiple solutions, return any of them.
Solution:

```
    std::vector<int> largestDivisibleSubset(std::vector<int> &nums) {
        int const n = nums.size();
        std::sort(nums.begin(), nums.end());
        std::vector<int> ans;

        std::vector<int> dp(n, 1), ix(n);
        int max_len = 1;
        int max_idx = 0;

        for(int i = 0; i < n; ++i) {
            ix[i] = i;
            for(int j = 0; j < i; ++j) {
                if(nums[i] % nums[j] == 0 and 1 + dp[j] > dp[i]) {
                    dp[i] = 1 + dp[j];
                    ix[i] = j;
                }
            }
            max_idx = (dp[i] > max_len) ?    i   : max_idx;
            max_len = (dp[i] > max_len) ? dp[i] : max_len;
        }
        ans.push_back(nums[max_idx]);

        while(ix[max_idx] != max_idx) {
            max_idx = ix[max_idx];
            ans.push_back(nums[max_idx]);
        }

        return ans;
    }
```

Discussion:

1.2 *279.* Perfect Squares

Source: *https://leetcode.com/problems/perfect-squares*
Problem Statement: Given an integer `n`, return the least number of perfect square numbers that sum to `n`.
A perfect square is an integer that is the square of an integer; in other words, it is the product of some integer with itself. For example, `1`, `4`, `9`, and `16` are perfect squares while `3` and `11` are not.
Solution:

```
  int numSquares(int n) {
        int const sqr = std::sqrt(n);
        int dp[n+1];
        memset(dp, 0x3f, sizeof(dp));
        dp[0] = 0;
        for(int i = 1; i <= sqr; ++i) {
            for(int j = i*i; j <= n; ++j) {
                dp[j] = std::min(dp[j], dp[j - i*i] + 1);
            }
        }
        return dp[n];
    }
```

Discussion:

1.3 *1463.* Cherry Pickup II

Source: *https://leetcode.com/problems/cherry-pickup-ii*
Problem Statement: You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

You have two robots that can collect cherries for you:

Robot #1 is located at the top-left corner `(0, 0)`, and Robot #2 is located at the top-right corner `(0, cols - 1)`. Return the maximum number of cherries collection using both robots by following the rules below:

(a) From a cell `(i, j)`, robots can move to cell `(i + 1, j - 1)`, `(i + 1, j)`, or `(i + 1, j + 1)`.

(b) When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.

(c) When both robots stay in the same cell, only one takes the cherries.

(d) Both robots cannot move outside of the grid at any moment.

(e) Both robots should reach the bottom row in grid.

Solution I:

```
int cherryPickup(vector<vector<int>>& grid) {
    std::ios_base::sync_with_stdio(false);
    std::cin.tie(nullptr);
    int const n = grid.size(), m = grid[0].size();
    int dp[n][m][m];
    memset(dp, 0, sizeof(dp));
    int max_cherries = 0;
    dp[0][0][m-1] = grid[0][0] + grid[0][m-1];
    for(int i = 1; i < n; ++i) {
        for(int j = 0; j < m; ++j) {
            for(int k = 0; k < m; ++k) {
                if(j > k || j > i || k < m-i-1) { continue; }
                dp[i][j][k] = dp[i-1][j][k];
                if(j-1 >= 0) {
                    dp[i][j][k] = std::max(dp[i][j][k], dp[i-1][j-1][k]);
                    dp[i][j][k] = (k-1 >= 0) ? std::max(dp[i][j][k], dp[i-1][j-1][k-1]) : dp[i][j][k];
                    dp[i][j][k] = (k+1  < m) ? std::max(dp[i][j][k], dp[i-1][j-1][k+1]) : dp[i][j][k];
                }
                if(j+1 < m) {
                    dp[i][j][k] = std::max(dp[i][j][k], dp[i-1][j+1][k]);
                    dp[i][j][k] = (k-1 >= 0) ? std::max(dp[i][j][k], dp[i-1][j+1][k-1]) : dp[i][j][k];
                    dp[i][j][k] = (k+1  < m) ? std::max(dp[i][j][k], dp[i-1][j+1][k+1]) : dp[i][j][k];

                }
                dp[i][j][k] = (k-1 >= 0) ? std::max(dp[i][j][k], dp[i-1][j][k-1]) : dp[i][j][k];
                dp[i][j][k] = (k+1  < m) ? std::max(dp[i][j][k], dp[i-1][j][k+1]) : dp[i][j][k];

                dp[i][j][k] += (j != k) ? (grid[i][j] + grid[i][k]) : grid[i][j];
                max_cherries = std::max(max_cherries, dp[i][j][k]) ;
            }
        }
    }
    return max_cherries;
}
```

There is another way to solve this; though not as efficient as the previous one, however it is more intutive.

Solution II (DFS with memoization):

```
class Solution {
private:
    int row, col;
public:
    int dirs[3] = {-1, 0, 1};
    int dp[100][100][100];
public:
    int dfs(std::vector<std::vector<int>> const &grid, int const &i,
                                                       int const &j,
                                                       int const &k) {

        if(j < 0 or j >= col or k < 0 or k >= col) return INT_MIN;
        if(i == row-1) {
            return (j == k) ? grid[i][j] : grid[i][j] + grid[i][k];
        }
        if(dp[i][j][k] != -1) { return dp[i][j][k]; }
        int max_cherries = 0, cherries = 0;
        for(int x = 0; x < 3; ++x) {
            for(int y = 0; y < 3; ++y) {
                cherries  = (j == k) ? grid[i][j] : grid[i][j] + grid[i][k];
                cherries += dfs(grid, i+1, j+dirs[x], k+dirs[y]);
                max_cherries = std::max(max_cherries, cherries);
            }
        }
        return dp[i][j][k] = max_cherries;
    }

    int cherryPickup(vector<vector<int>>& grid) {
        this->row = grid.size(); this->col = grid[0].size();
        memset(dp, -1, sizeof(dp));
        return dfs(grid, 0, 0, col-1);
    }
};
```

Discussion:

1.4 *390.* Lorem Ipsum

Source: *https://leetcode.com/problems/*
Problem Statement:
Solution:

```

```

Discussion:

1.5 *390.* Lorem Ipsum

Source: *https://leetcode.com/problems/*
Problem Statement:
Solution:

```

```

Discussion:

2 Acknowledgement