Algorithms and Data Structures: Homework #11

Due on April 27, 2020 at 23:00

Henri Sota

Problem 11.3

A scuba diver uses a special equipment for diving. He has a cylinder with two containers: one with oxygen and the other with nitrogen. Depending on the time he wants to stay under water and the depth of diving the scuba diver needs various amount of oxygen and nitrogen. The scuba diver has at his disposal a certain number of cylinders. Each cylinder can be described by its weight and the volume of gas it contains. In order to complete his task the scuba diver needs specific amounts of oxygen and nitrogen. Theoretically, the diver can take as many cylinders as he wants/needs. Use dynamic programming to find the minimal total weight of cylinders he has to take to complete the task and which those cylinders are. In case of several acceptable solutions, printing just one of them is enough.
Dynamic Programming solution for this problem is given below and it is found inside of file "scubaDiver.cpp". Solution is based on implementation given here and here.

```
int INF = 1061109567;   // 0x3f3f3f3f

// Integer vectors to hold information of up to 1000 possible cylinders
std::vector<int> oxygen(1000, 0);
std::vector<int> nitrogen(1000, 0);
std::vector<int> weight(1000, 0);

// Dynamic Programming table (3D Matrix) with maximum number of cylinders
// (items) and all possible oxygen and nitrogen values
std::vector<std::vector<std::vector<int>>> dp(
    1000, std::vector<std::vector<int>>(22, std::vector<int>(80, 0))
);

int main() {
    int c;          // Test cases
    int t;          // Minimum oxygen needed
    int a;          // Minimum nitrogen needed
    int n;          // Number of cylinders

    // Input number of test cases
    std::cin >> c;

    // Run algorithm for each test case
    while (c--) {
        std::vector<int> solution;
        // Input requirements of test case and number of cylinders offered
        std::cin >> t >> a >> n;

        // Set (Reset) each value in 3D matrix to a significantly large value
        for (int i = 0; i < 1000; ++i)
            for (int j = 0; j < 22; ++j)
                for (int k = 0; k < 80; ++k)
                    dp[i][j][k] = INF;

        // Set (Reset) each cylinder's oxygen, nitrogen and weight status
        for (int i = 0; i < 1000; ++i) {
            oxygen[i] = 0;
            nitrogen[i] = 0;
            weight[i] = 0;
        }

        // Input information for each cylinder
        for (int i = 0; i < n; ++i)
            std::cin >> oxygen[i] >> nitrogen[i] >> weight[i];

        // Set base values when oxygen and nitrogen is 0 in supply
        for (int i = 0; i < n; ++i)
```

```
            dp[i][0][0] = weight[i - 1];

        // Traverse the vector of cylinders and check if we can meet the
        // minimum requirement with cylinders from 1 to i
        for (int i = 1; i <= n; ++i)
            for (int j = 0; j <= t; ++j)
                for (int k = 0; k <= a; ++k) {
                    // Check if the current cylinder can fulfill the
                    // oxygen and nitrogen requirements
                    if ((j <= oxygen[i - 1]) && (k <= nitrogen[i - 1]))
                        // Save minimum of the weight of taking this current
                        // cylinder or the weight of the previous cylinders
                        dp[i][j][k] = std::min(dp[i - 1][j][k], weight[i - 1]);
                    else
                        // In case the current cylinder does not fulfill the
                        // requirements
                        // Save the minimum between the weight of the previous
                        // cylinders or taking this cylinder and a combination
                        // of the previous cylinders
                        dp[i][j][k] = std::min(
                            dp[i - 1][j][k],
                            dp[i - 1][std::max(0, j - oxygen[i - 1])][std::max(0, k -
    nitrogen[i - 1])] + weight[i - 1]
                        );
                }

        // Reconstruct solution to retrieve the combination of cylinders used
        int cylinder = n;
        int oxy = t;
        int nitro = a;
        // Traverse 3D Table until all cylinders have been checked and oxygen
        // and nitrogen supply has been fulfilled
        while (cylinder >= 0 && (oxy > 0 || nitro > 0)) {
            // Add the first cylinder at the end in case requirement has not yet
            // been fulfilled
            if (cylinder == 0) {
                solution.push_back(cylinder);
                break;
            }
            // Check if current cylinder has been used in optimal solution
            if (dp[cylinder][oxy][nitro] != dp[cylinder - 1][oxy][nitro]) {
                oxy = std::max(0, oxy - oxygen[cylinder - 1]);
                nitro = std::max(0, nitro - nitrogen[cylinder - 1]);
                solution.push_back(cylinder);
            }
            // Move onto the solution set for the cylinder before current one to
            // find out if is part of solution
            cylinder--;
        }

        // Output optimal weight of the combination of cylinders that meet
        // requirements
        std::cout << dp[n][t][a] << std::endl;

        // Output the indices of the cylinders in the optimal solution
        for (int i = solution.size() - 1; i >= 0; --i)
            std::cout << solution[i] << " ";
        std::cout << std::endl;
    }

    return 0;
```

```
}
```

Listing 1: Dynamic Programming Solution for Scuba Diver in C++