

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6
«Динамическое программирование»

Выполнил работу

Карташов Игорь

Академическая группа №J3111

Принято

Вершинин Владислав

Санкт-Петербург

2024

Введение

Цель работы: Определить минимальное начальное здоровье, необходимое рыцарю для того, чтобы пройти через все комнаты подземелья и спасти принцессу, сохранив здоровье на протяжении всего пути.

Теоретическая подготовка

Динамическое программирование используется, потому что:

1. Задача обладает оптимальной структурой подзадач.
2. Проблема содержит подзадачи.
3. Жадные и прямые методы не дают оптимального решения.
4. ДП эффективно сокращает время вычислений и сохраняет промежуточные результаты, избегая повторных вычислений.

Реализация

Используется метод динамического программирования для вычисления минимального здоровья, необходимого рыцарю для входа в каждую комнату, начиная с конца подземелья (клетка с принцессой). Для любой комнаты (i,j) , минимальное здоровье рассчитывается на основе комнат справа и снизу. Начальная комната $(0,0)$ содержит итоговый результат: минимальное начальное здоровье рыцаря.

Тестовая часть

Problem List

Run

Submit

Accepted

DescriptionEditorialSolutionsSubmissionsAccepted

All Submissions

Accepted

user2222Qx submitted at Nov 27, 2024 00:34

EditorialSolution

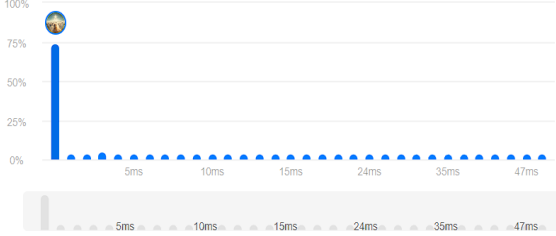
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

13.08 MB | Beats 7.53%



Code | C++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
    int calculateMinimumHP(vector<vector<int>>& dungeon) {
        int m = dungeon.size(); // 4 байта
        int n = dungeon[0].size(); // 4 байта

        // dp массив для минимального здоровья рыцаря
        vector<vector<int>> dp(m, vector<int>(n, INT_MAX)); // O(m * n) m*n*4 байт

        // заполняем клетку с принцессой
        dp[m - 1][n - 1] = max(1, 1 - dungeon[m - 1][n - 1]);

        // Заполняем таблицу снизу вверх
        for (int i = m - 1; i >= 0; --i) { // Сложность O(m * n) // 4 байт + 4 байт для i j
            for (int j = n - 1; j >= 0; --j) {
                if (i < m - 1) {
                    dp[i][j] = min(dp[i][j], max(1, dp[i + 1][j] - dungeon[i][j]));
                }
                if (j < n - 1) {
                    dp[i][j] = min(dp[i][j], max(1, dp[i][j + 1] - dungeon[i][j]));
                }
            }
        }
    }
};
```

TestcaseTest Result

Заключение

Временная сложность $O(m*n)$ позволяет обрабатывать большие размеры матрицы за разумное время. Всего памяти $4*m*n + 4*m*n + 16 = 8 * m * n + 16$ байт.

Приложения

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:

int calculateMinimumHP(vector<vector<int>>& dungeon) {
    int m = dungeon.size(); // 4 байта
    int n = dungeon[0].size(); // 4 байта

    // dp массив для минимального здоровья рыцаря
    vector<vector<int>> dp(m, vector<int>(n, INT_MAX)); // O(m * n) m*n*4 байт

    // заполняем клетку с принцессой
    dp[m - 1][n - 1] = max(1, 1 - dungeon[m - 1][n - 1]);

    // Заполняем таблицу снизу вверх
    for (int i = m - 1; i >= 0; --i) { // Сложность O(m * n) // 4 байт + 4 байт для i j
        for (int j = n - 1; j >= 0; --j) {
            if (i < m - 1) {
                dp[i][j] = min(dp[i][j], max(1, dp[i + 1][j] - dungeon[i][j]));
            }
            if (j < n - 1) {
                dp[i][j] = min(dp[i][j], max(1, dp[i][j + 1] - dungeon[i][j]));
            }
        }
    }

    return dp[0][0];
}

// Всего памяти 4*m*n + 4*m*n + 16 = 8 * m * n + 16 байт
int main() {
    vector<vector<int>> dungeon = { // 4*m*n байт
        {-2, -3, 3},
        {-5, -10, 1},
        {10, 30, -5}
    };

    cout << calculateMinimumHP(dungeon) << endl;
    return 0;
}

};
```