Динамическое программирование. Часть 3

Основная концепция динамическое программирования была изложена в 12_lesson и 13_lesson. Далее рассмотрим особенности двумерного динамического программирования.

Двумерное динамическое программирование.

Напомним, что у всякой подзадачи есть некоторое количество параметров, которые её определяют. В задачах на двумерное динамическое программирование таких параметров 2, то есть промежуточные результаты вычисления хранятся в двумерном векторе - vector<vector<int> > dp.

Структуру решения задач двумерного динамического программирования будет несколько сложнее, чем структура задач одномерного ДП. Наша задача состоит в нахождении dp[n][m]. Мы знаем, что dp[i][j] связано с меньшими dp некоторые образом, например dp[i][j] = dp[i][j-1] + dp[i-1][j] + 2*dp[i-1][j-1]. При этом нам известно несколько первых dp - например dp[0][0] = 1 и dp[0][1] = 2 dp[1][0] = 1, dp[1][1] = 3. Тогда мы будем последовательно заполнять массив dp, начиная от меньших i, j к большим и, таким образом, в итоге мы найдём dp[n][m]. Далее разберём типичную задачу двумерного динамического программирования.

Задача 1. Дано прямоугольное поле M размером n*m клеток. Можно совершать шаги длиной в одну клетку вправо, вниз или по диагонали вправо-вниз. В каждой клетке записано некоторое натуральное число M[i][j]. Необходимо попасть из верхней левой клетки в правую нижнюю. Цена маршрута вычисляется как сумма чисел со всех посещенных клеток. Необходимо найти цену самого дешёвого маршрута.

Для решения этой задачи методом динамического программирования сведем исходную задачу к подзадачам. То есть подзадачей здесь является нахождение цены оптимального маршрута для прямоугольного поля меньшего размера, то есть dp[i][j], где 0 <= i <= n-1, 0 <= j <= m-1.

При i = 0, j = 0 ответ очевиден. То есть dp[0][0] = M[0][0], Допустим, что мы уже нашли dp[i – 1][j], dp[i][j-1] и dp[i-1][j-1]. Тогда dp[i][j] = min(min(dp[i-1][j], dp[i][j-1]), dp[i-1][j-1]) + M[i][j].

Таким образом, последовательно заполняя массив dp мы найдём dp[n-1][m-1]. Ниже приведёна программа, представляющая собой решение данной задачи.

```
#include<iostream>
#include<vector>
using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;

    vector<vector<int> > dp(n, vector<int>(m, 0));
    vector<vector<int> > M(n, vector<int>(m, 0));


    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
            cin >> M[i][j];


    for(int i = 0; i < n; i++)
        for(int j = 0; j < m; j++)
        {
            if (i == 0 && j == 0)
                dp[i][j] = M[i][j];
            else if (i == 0)
                dp[i][j] = dp[i][j-1] + M[i][j];
            else if (j == 0)
                dp[i][j] = dp[i-1][j] + M[i][j];
            else
                dp[i][j] = min(min(dp[i-1][j], dp[i][j-1]), dp[i-1][j-1])
                + M[i][j];
        }

    cout << dp[n-1][m-1] << endl;
        return 0;
}
```