Динамическое программирование. Часть 2

Основная концепция динамическое программирования была изложена в 12_lesson. Далее рассмотрим особенности псевдодвумерного динамического программирования.

Псевдодвумерное динамическое программирование.

Напомним, что у всякой подзадачи есть некоторое количество параметров, которые её определяют. В задачах на псевдодвумерное динамическое программирование таких параметров 2, причём второй параметр изменяется от 0 до 1. Промежуточные результаты вычисления хранятся в двумерном векторе n*2 - vector<vector<int> > dp(n, vector<int>(2)).

Структура решения задач псевдодвумерного динамического программирования практически совпадает со струрой решения задач одномерного динамического программирования. Наша задача состоит в нахождении двух чисел - dp[n][0] и dp[n][1]. Мы знаем, что dp[i][0] и dp[i][1] связаны с меньшими dp некоторые образом, например dp[i][0] = dp[i-1][1] + dp[i-1][0], dp[i][1] = -dp[i-1][0]. При этом нам известно dp[0][0] и dp[0][1] - например dp[0][0] = 1 и dp[0][1] = -1. Тогда мы будем последовательно заполнять массив dp, начиная от меньших i к большим и, таким образом, в итоге мы найдём dp[n][0] и dp[n][1]. Далее разберём типичную задачу псевдодвумерного динамического программирования.

Задача 1. В дощечке в один ряд вбиты гвоздики. Любые два гвоздика можно соединить ниточкой. Требуется соединить некоторые пары гвоздиков ниточками так, чтобы к каждому гвоздику была привязана хотя бы одна ниточка, а суммарная длина всех ниточек была минимальна.

Для решения этой задачи методом псевдодинамического программирования сведем исходную задачу к подзадачам. То есть подзадачей здесь является нахождение минимальной суммарной длины для всех гвоздиков от первого до i-ого, где i <= n. Однако в этой задаче важно различать 2 состояния - связан ли последний гвоздик с предпоследним или же он ни с кем не связан. На первый взгляд может показаться, что второе состояние противоречит условию задачи, так как все гвоздики должны быть с кем-то связаны. Однако введение такого "противоречивого"состояния облегчит наши вычисления. Таким образом, пусть dp[i][0] - суммарная минимальная длина всех ниточек, такая что к каждому из первых i гвоздиков привязана хотя бы одна ниточка и i-ый гвоздик связан с i-1-ым, dp[i][1] это суммарная минимальная длина всех ниточек, такая что к каждому из первых i-1 гвоздиков привязана хотя бы одна ниточка и i-ый гвоздик ни с кем не связан. В итоге, ответ на вопрос задачи это dp[n-1][0].

Пусть функция int dist(int i, int j) возвращает расстояние между гвоздиками i и j. При i = 1 ответ очевиден. То есть dp[1][0] = dist(0, 1), dp[1][1] = -1 (пусть -1 обозначает, что нельзя связать гвоздики указанным образом). Теперь заметим, что dp[i][0] = min(dist(i, i-1) + dp[i-1][1], dist(i, i-1) +

dp[i-1][0]), dp[i][1] = dp[i-1][0].

Таким образом, последовательно заполняя массив dp мы найдём dp[n-1][0]. Ниже приведёна программа, представляющая собой решение данной задачи.

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

vector<int> d;

inline int dist(int i, int j)
{
        return d[max(i, j)] - d[min(i, j)];
}

int main()
{
    int n;
    cin >> n;
    vector<vector<int> > dp(n, vector<int>(2));
    d.resize(n); // koordinati gvozdikov

    for(int i = 0; i < n; i++)
        cin >> d[i];

    sort(d.begin(), d.end());
    dp[1][0] = dist(0, 1);
    dp[1][1] = -1;

    for(int i = 2; i < n; i++)
        {
                        dp[i][0] = 1000000;
                        if (dp[i-1][0] != -1)
                dp[i][0] = min(dp[i][0], dist(i, i-1) + dp[i-1][0]);
            if (dp[i-1][1] != -1)
                dp[i][0] = min(dp[i][0], dist(i, i-1) + dp[i-1][1]);

            dp[i][1] = dp[i-1][0];
        }

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