Slope Trick：一种特殊的动态规划优化方式

张瑞珉

2024.11.21

摘要

动态规划算法是一种通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法 [1]。有一类特殊的可以用动态规划算法解决的问题，其分解后的子问题的解呈一个凸包且斜率均为整数。传统动态规划算法需要直接求解其所有子问题的值，效果并不理想。本文将介绍一种特殊的优化方法 `Slope Trick` [2]，通过记录斜率的变化位置从而较快的解决这一类问题。

1 引言

动态规划的一个重要思想是重复利用子问题的解从而避免复杂计算，所以往往需要一个数组去辅助存储，我们称之为“dp 数组”。同时它还需要从一个子问题的解推出另一个子问题的解，这部分所需要的步骤称之为“转移方程”。

有一类特殊的动态规划问题，其子问题的解呈一个凸包且斜率均为整数。它的 dp 数组一般为二维，其中一维的定义类似于时间轴，且如果我们把该问题 dp 数组的第一维（时间轴维）固定，以第二维的下标为 x 轴、值为 y 轴，将其抽象为一个二维平面上的折线，那么这个折线就是一个斜率较为连续的凸包。同时，我们可以以将这个折线抽象为一个定义域为 $(-\infty,+\infty)$ 的分段一次函数。

如果这类问题的转移方程在二维平面上的刻画为随着时间的变化，对全局（$x\in(-\infty,+\infty)$）加上一个形如 $|x-a|$ 的绝对值函数，然后进行平移或者前/后缀取 min 等操作，那么它们就可能用 `Slope Trick` 来进行优化。

2 核心思想与常用实现

我们将先介绍 `Slope Trick` 的核心思想，然后介绍它的常用实现方式。

我们记斜率突变的位置为“转折点”。很多的绝对值函数的和会形成一个凸包，每段的斜率递增且为较为连续的整数。`Slope Trick` 的核心思想在于存储所有的转折点的位置，因为只需要记录转折点就可以很好的刻画这个折线。

我们通常用 `priority_queue` 来存储这些转折点的位置，这是为了保证这些位置的单调性。假设在某个子问题中，我们当前的折线由 n 个绝对值函数叠加，那么我们的斜率范围即为 $[-n, n]$。我们将维护所有的 $2n$ 个转折点（包括不存在的转折点），这样会更方便的维护这条折线。对于每个整数 $i \in [-n, n-1]$ 我们都记录斜率由 i 突变至 $i+1$ 的位置。特殊的，如果某个斜率 x 不存在，那么我们假设最大的比 x 小且存在的斜率为 p，最小的比 x 大且存在的斜率为 q，那么我们令 $x-1$ 突变为 x 的位置和 x 突变为 $x+1$ 的位置均为 p 突变为 q 的位置。

我们通过维护转折点的位置，就可以方便的维护整个图像的最小值等关键信息。

如函数 $f(x) = |x-2| + |x-2| + |x+1| + |x+2|$（图 1），它由 4 个绝对值函数叠加而成。我们便记录对于所有 $i \in [-4, 3]$，斜率由 i 变为 $i+1$ 的位置，如表 1 所示。

i	-4	-3	-2	-1	0	1	2	3
转折点位置	-2	-2	-1	-1	2	2	2	2

表 1: 斜率变化的位置

3 例题

3.1 Codeforces 713C Sonya and Problem Wihtout a Legend

题目大意 给定一个长度为 n 的数组 $a_{1\ldots n}$。每次你可以花费 1 的代价将 a 中的某个元素加 1 或者减 1。你需要让数组 a 严格单调递增，并求出最少花费的代价。

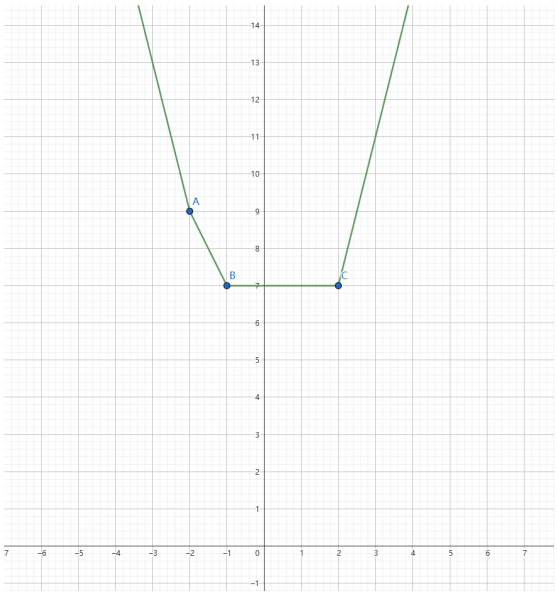

图 1: f(x) 的函数图像

解题思路 首先对于每个 $i = 1, 2, \ldots, n$ 将 a_i 减去 i。这样做就可以将题目改为使得 a 数组不严格单调增了。

然后就可以得到一个基础的动态规划：令 $dp_{i,j}$ 表示考虑到第 i 个元素，最终将 a_i 改变为 j 且 $a_{1,\ldots,j}$ 非严格单调增的最小代价。那么我们的转移方程为 `dp[i][j]=min{dp[i-1][j]}+|j-a[i]|,j<=i`。将 a 数组离散化之后即可以 $O(n^2)$ 的时间复杂度通过本题。

接下来我们试着将原问题转化为可以利用 `Slope Trick` 解决的问题。

令 $f_{i,j}$ 为 $dp_{i,j}$，$g_{i,j}$ 为考虑到第 i 位，将 a_i 最终改为的值 $\leq j$ 时最小的代价。那么我们最终要求的答案为 $g_{n,+\infty}$，且 $g_{i,j}$ 为 $f_{i,j}$ 的前缀最小值。此时我们的转移方程就变为 `f[i][j]=g[i-1][j]+|j-a[i]|`。

如果将 i 固定，以 j 为 x 轴，$f_{i,j}, g_{i,j}$ 为 y 轴，那么就可以观察到由 f 计算出 g 的过程相当于将斜率为 0 之后的部分抹平（即弹出所有 $i \geq 0$ 的转折点）；由 g 计算出 f 的过程相当于将函数图像叠加上一个绝对值函数 $|x - a_i|$。

此时的问题已经符合了 `Slope Trick` 的解决范围。我们考虑利用 `Slope Trick` 进行优化。

我们记当前所有转折点的最大值（即斜率由 -1 转为 0 的位置）为 s。那么最终的答案就是 $x = s$ 时的函数值。

如果 $s \le a_i$，那么加入函数 $|x - a_i|$ 只会产生一个新的转折点 a_i。我们只需要向记录转折点的堆中压入 a_i 即可。注意到这一操作并不会改变当前的答案值。

如果 $a_i < s$，那么我们会发现这一操作将所有小于 a_i 的转折点对应的斜率 -1，大于 a_i 的转折点的斜率 $+1$。这意味着我们需要连着向记录转折点的堆中压入两次 a_i。这一操作之后，我们会得到一个斜率由 0 转为 1 的转折点。而我们由 f 计算 g 的过程需要抹去所有 $i \ge 0$ 的转折点，所以我们需要弹出这一个转折点（即 s）。但是此时我们的答案会发生改变。通过图像可以得出，答案加上了 $|a_i - s|$。

最终，我们可以通过使用堆存储所有的转折点，在 $O(n \log n)$ 的时间复杂度内解决本题。

代码

Listing 1: Codeforces 713C

```cpp
#include<bits/stdc++.h>
using namespace std;
priority_queue<int>q;//记录转折点位置
int main(){
        int n;
        long long ans=0;
        cin>>n;
        for(int i=1;i<=n;++i){
                int x;
                cin>>x;
                x-=i;
                q.push(x);
                if(x<q.top()){
                        q.push(x);//再压入一个
                        ans+=q.top()-x;//更新答案
                        q.pop();//弹出斜率由0转1的点
                }
```

```
        }
        cout<<ans<<endl;
        return 0;
}
```

4 总结

Slope Trick 是一种不常见的技巧，然而它在解决这一类特殊的动态规划问题时能够起到奇效。它的代码量很短，运行效率也很高，无法被一般动态规划做法所取代。

参考文献

[1] CBW2007 等. 动态规划部分简介. https://oi-wiki.org/dp/, 2021.

[2] Doan Duc Trung Dang. Slope trick explained. https://codeforces.com/blog/entry/77298, 2019.