DP 优化

n + e

Tsinghua University

2016 年 7 月 15 日

DP 状态怎么设计？

- 很多同学在学 DP 的时候都有这样一个疑问，包括我
- 如果可以归类，那么有它具体的设计状态方式（树型 DP、状压 DP、区间 DP⋯⋯）
- 在通常情况下（指 NOIP），只要把题目中所给的所有状态揉在一起，好像就好了？
- 阶段的划分：经典模型

DP 状态怎么设计？

- 很多同学在学 DP 的时候都有这样一个疑问，包括我
- 如果可以归类，那么有它具体的设计状态方式（树型 DP、状压 DP、区间 DP……）
- 在通常情况下（指 NOIP），只要把题目中所给的所有状态揉在一起，好像就好了？
- 阶段的划分：经典模型
- 以下题目若未特殊声明，时限 1s，内存 128M。

1. 减少状态总数
   - 跳过无用状态
   - 变量相互制约
2. 减少状态转移数
3. 内存优化
4. 其他

- 我们知道，动态规划的求解过程实际上就是计算所有状态值的过程，因此状态的规模直接影响到算法的时间效率。所以，减少状态总数是动态规划优化的重要部分

1. 减少状态总数
   - 跳过无用状态
     - [NOIP2005] 青蛙过河
     - 总结
   - 变量相互制约
2. 减少状态转移数
3. 内存优化
4. 其他

[NOIP2005] 青蛙过河

- x 轴上分布着 M 个石子，在坐标为 0 的点上有一只青蛙，每一次它向 x 轴正方向跳跃的距离是 S 到 T 之间的任意正整数（包括 S,T）。当青蛙跳到或跳过坐标为 L 的点时，就算青蛙已经过了河。
- 你的任务是确定青蛙要想过河，最少需要踩到的石子数。$L \leq 10^9, 1 \leq S \leq T \leq 10, M \leq 100$

```
f[0]=0;
for(int i=1,j;i<=l+t;i++){
    for(f[i]=1<<30,j=max(0,i-t);j<=i-s;j++)
        f[i]=min(f[i],f[j]);
    if(stone_is_in[i])f[i]++;
}
```

```
f[0]=0;
for(int i=1,j;i<=l+t;i++){
    for(f[i]=1<<30,j=max(0,i-t);j<=i-s;j++)
        f[i]=min(f[i],f[j]);
    if(stone_is_in[i])f[i]++;
}
```

- L 很大，不能一个个算；石子间隔很远，有大量无用状态
- 当 S 与 T 不相等时，因为对于任意 d>ST，从 x 总能跳到 x+d，所以对石子排序，把距离大于 ST 的缩到 ST，缩完以后 L 不超过 10000

总结

- 其实是我找不到类似的题目了……都是要套个数据结构……
- 不过，通常的思路是，把 10^9 级别的数排序完离散化，然后该干嘛干嘛。刚才那道题类似离散化。

1. 减少状态总数
   - 跳过无用状态
   - 变量相互制约
     - [NOIP2000] 方格取数
     - 练习：[NOIP2010] 乌龟棋
2. 减少状态转移数
3. 内存优化
4. 其他

[NOIP2000] 方格取数

- 有一个 N×N 的方格图，在某些方格中填入正整数，其它方格中填入数字 0。如下图所示：

A

0	0	0	0	0	0	0	0
0	0	13	0	0	6	0	0
0	0	0	0	7	0	0	0
0	0	0	14	0	0	0	0
0	21	0	0	0	4	0	0
0	0	15	0	0	0	0	0
0	14	0	0	0	0	0	0
0	0	0	0	0	0	0	0

B

- 某人从 A(1,1) 出发，可以向下、向右行走，到达 B(N,N)。在走过的路上，他可以取走方格中的数，取走后的方格变为 0。
- 此人从 A 点到 B 点共走了两次，试找出两条这样的路径，使得取得的数字和为最大。$N \leq 100$

改进状态表示

- 我们用 $f(X_1, Y_1, X_2, Y_2)$ 表示第一个点走到 (X_1, Y_1)，第二个点走到 (X_2, Y_2) 时的取数的最大和

改进状态表示

- 我们用 $f(X_1, Y_1, X_2, Y_2)$ 表示第一个点走到 (X_1, Y_1)，第二个点走到 (X_2, Y_2) 时的取数的最大和
- 同一时刻这两个点的到原点的距离相等，不满足这个方程的状态是不合法的：$X_1 + Y_1 = X_2 + Y_2$
- $f(x_1, y_1, x_2)$，四方变三方

改进状态表示

- 我们用 $f(X_1, Y_1, X_2, Y_2)$ 表示第一个点走到 (X_1, Y_1)，第二个点走到 (X_2, Y_2) 时的取数的最大和
- 同一时刻这两个点的到原点的距离相等，不满足这个方程的状态是不合法的：$X_1 + Y_1 = X_2 + Y_2$
- $f(x_1, y_1, x_2)$，四方变三方
- 把两条改成三条？四条？N 条？

练习：[NOIP2010] 乌龟棋

- 乌龟棋的棋盘是一行 N 个格子，每个格子上一个分数（非负整数）。游戏要求玩家控制一个乌龟棋子，从起点第 1 格出发走到终点第 N 格。

- 乌龟棋中有 M 张卡片，卡片有四种花色，分别对应 1,2,3,4 四个数字。每次使用一张卡片，棋子就可以向前移动这张卡片所对应数字的格数。比如用一张第三种花色的卡片，乌龟棋就向前移动三格。每张卡片在一次游戏中只能使用一次。

练习：[NOIP2010] 乌龟棋

- 玩家在本次游戏中的得分，就是移动乌龟棋从第一格到最后一格的过程中，经过的所有的格子上的分值的和。
- 很明显，用不同的卡片使用顺序会使得最终游戏的得分不同，你的任务是要找到一种卡片使用顺序使得最终游戏得分最多。
- 数据保证到达终点时刚好用光 M 张爬行卡片。
  $N \le 350, M \le 120,$ 每种卡片张数 ≤ 40

- $f(i, a_1, a_2, a_3, a_4)$ 表示玩到第 i 格，用了 a_1 张 1,a_2 张 2,a_3 张 3,a_4 张 4 的最大得分。
- 从 $f(i-1, a_1-1, a_2, a_3, a_4), f(i-2, a_1, a_2-1, a_3, a_4), f(i-3, a_1, a_2, a_3-1, a_4), f(i-4, a_1, a_2, a_3, a_4-1)$ 转移
- $i = a_1 + 2a_2 + 3a_3 + 4a_4 + 1$，直接把第一维扔掉就 AC 了。

1. 减少状态总数
2. 减少状态转移数
   - 预处理/前缀和
   - 部分和优化
   - 四边形不等式
   - 决策单调性
   - 数据结构优化
3. 内存优化
4. 其他

- 有的时候 dp 复杂度太高，往往是一直在重复计算某些东西。那么我们可以将这些重复计算的东西在一开始就算好，来降低 dp 的复杂度。

最大子段和

- 给定一个长度为 N 的序列 $A_{1..N}$，请找出一段区间 $a_L, a_{L+1}, \cdots, a_R$，使得 $a_L + a_{L+1} + \cdots + a_R$ 最大，即最大化

$$\sum_{i=L}^{R} a_i$$

- $N \le 100000$，你的答案要保证 $L \le R$

暴力怎么写

- 枚举区间左端点 L 和右端点 R

```
for(int l=1;l<=n;l++)
for(int r=l;r<=n;r++){
    int sum=0;
    for(int i=l;i<=r;i++)sum+=a[i];
    if(ans<sum)ans=sum;
}
```

- 这样做的复杂度是 $O(N^3)$ 的：因为第三个 for 语句是在第二个 for 和第一个 for 语句控制之下，所以算复杂度的时候是乘起来算的

暴力进化版：前缀和优化

- 一直在重复计算 $[L, R]$ 的和，有点浪费

```
for(int i=1;i<=n;i++)sum[i]=sum[i-1]+a[i];
for(int l=1;l<=n;l++)
for(int r=l;r<=n;r++){
    int tmp=sum[r]-sum[l-1];
    if(ans<tmp)ans=tmp;
}
```

- 这样做的复杂度是 $O(N^2)$ 的：因为第一个 for 跟后面的两个 for 是分开来的。

dp 怎么做

- 原来是枚举左右端点
- 如果固定右端点 R，能快速求出 $\max(sum[R] - sum[L-1])$ 就能解决这个问题。
- 由于 R 已经固定，因此我们只需找出 $\min(sum[L-1])$ 即可。
- 写成 dp 方程就是

$$f[r] = \max(sum[r] - sum[l-1]) = sum[r] - \min(sum[l-1])$$

其中 $f[r]$ 表示 $[1, r]$ 中的最大子段和，并且强制要取 $A[r]$。

- 把前缀和拆掉就是

$$f[r] = \max(f[r-1], 0) + A[r]$$

dp 怎么做

- 怎么找：有一堆数，可以往里面添加一个数，要求查询最小值：保留最小值即可。

```
for(int i=1;i<=n;i++)sum[i]=sum[i-1]+a[i];
int min=0;
for(int r=1;r<=n;r++){
    if(ans<sum[r]-min)ans=sum[r]-min;
    if(min>sum[r])min=sum[r];
}
```

- 这样做的复杂度是 $O(N)$ 的：因为第一个 for 跟后面一个 for 是分开来的。
- 这样我们运用预处理的方法，在线性时间内完成了求一段序列的最大子段和的任务。

环状最大子段和

- 给定一段环状序列，即认为 $A[1]$ 和 $A[N]$ 是相邻的，选出其中连续且非空的一段使得这一段和的最大。
- $N \leq 100000$

环状最大子段和

- 给定一段环状序列，即认为 $A[1]$ 和 $A[N]$ 是相邻的，选出其中连续且非空的一段使得这一段和的最大。
- $N \leq 100000$
- 由于环是一条链的两端黏在一起的，因此可以枚举分界点，让一把剪刀从这个分界点把环剪开来变成链。然后就是上面那个问题加上一个限制条件：区间长度不能超过 N。运用单调队列即可维护。
- 实际编程过程中，常常是把这段序列复制一遍，接在后面，新的序列长度为 $2N$，查询答案时只要访问 $Ans[1, N], Ans[2, N+1], \cdots, Ans[N, 2N-1]$ 即可

另一种做法

- 就在原序列上做，不复制。
- 先求常规的序列上的最大子段和，那么我们就求出了未跨过 $N \to 1$ 的答案
- 跨过的怎么办？考虑它的反面：剩下没有被选的那段一定没有跨过 $N \to 1$，并且如果跨过的那段和最大，那么剩下的一段和一定最小，即：最小子段和
- 因此求一遍最大子段和、最小子段和即可。

两段最大子段和

- 给定一段长度为 N 的序列 $A_{1..N}$，选出其中连续不重叠且非空的两段使得这两段和最大。$N \leq 100000$

两段最大子段和

- 给定一段长度为 N 的序列 $A_{1..N}$，选出其中连续不重叠且非空的两段使得这两段和最大。$N \leq 100000$
- 由于是两段，因此必然会有一个分界点，可以从这里把序列切开。
- 那么，我们只要 $O(N)$ 枚举分界点 i，如果能快速求出 $[1, i]$ 和 $[i+1, N]$ 中的最大子段和，那么本题就能成功解决。
- $f[i]$ 表示 $[1, i]$ 中的最大子段和，$g[i]$ 表示 $[i+1, N]$ 中的最大子段和，预处理 $f[i], g[i]$，即正着一遍、反着一遍做最大子段和即可。

环状两段最大子段和

- 把上面那题的序列变成环。

环状两段最大子段和

- 把上面那题的序列变成环。
- 分割情况无非两种：
  1. 两段最大子段和完整，两段最小子段和的其中一段被分割
  2. 两段最小子段和完整，两段最大子段和的其中一段被分割
- 所以我们就先求两段最大/小子段和，然后分类讨论，比比哪个结果大就好了。
- 当然这题还有很多做法。

1 减少状态总数

2 减少状态转移数
- 预处理/前缀和
- 部分和优化
  - [NOIP2015PJ] 求和
  - 练习：区间 GCD
- 四边形不等式
- 决策单调性
- 数据结构优化

3 内存优化

4 其他

- 预处理表达式中的某些元素，这个大家应该都能看得出来
- 通常都要拆式子

[NOIP2015PJ] 求和

- 一条狭长的纸带被均匀划分出了 n 个格子，格子编号从 1 到 n。每个格子上都染了一种颜色 col_i（用 $[1, m]$ 当中的一个整数表示），并且写了一个数字 num_i。

- 定义一种特殊的三元组：(x, y, z)，其中 x, y, z 都代表纸带上格子的编号，这里的三元组要求满足以下两个条件：
  1. x, y, z 都是整数，$x < y < z, y - x = z - y$
  2. $col_x = col_z$
- 满足上述条件的三元组的分数规定为 $(x + z)(num_x + num_z)$。整个纸带的分数规定为所有满足条件的三元组的分数的和。输出整个纸带的分数。$n, m \leq 100000$

- 注意到 y 基本没啥用。枚举 z，统计 x 和 z 同奇偶、同颜色的对数 (x,z) 的答案。

$$(x+z)(num_x+num_z) = x\cdot num_x + x\cdot num_z + z\cdot num_x + z\cdot num_z$$

$$(x_1+z)(num_{x_1}+num_z) = x_1\cdot num_{x_1} + x_1\cdot num_z + z\cdot num_{x_1} + z\cdot num_z$$

$$(x_2+z)(num_{x_2}+num_z) = x_2\cdot num_{x_2} + x_2\cdot num_z + z\cdot num_{x_2} + z\cdot num_z$$

- 对于同一种颜色，奇偶分类后，维护 $\sum x\cdot num_x$, $\sum x$, $\sum num_x$, $\sum 1$

练习：区间 GCD

- 给定一个长度为 n 的字符串 $s_{1..n}$，串仅包含小写字母。
- 有 q 组询问，每组询问对于区间 $[l, r]$，你需要回答 $s_{l..r}$ 中有多少个长度为 3 的子序列组成了"gcd"，即有多少组 (i, j, k) 满足 $l \leq i < j < k \leq r$，$s_i =$'g'，$s_j =$'c'，$s_k =$'d'.
- $1 \leq n, q \leq 100000$，串仅包含小写字母，$1 \leq l_i \leq r_i \leq n$。

- 枚举 c 在哪里，统计左边有多少个 g，右边有多少个 d

$$ans = \sum_{j=l}^{r} [s_j =' c'] \cdot \left(\sum_{i=l}^{j-1} [s_i =' g'] \right) \cdot \left(\sum_{k=j+1}^{r} [s_k =' d'] \right)$$

$$= \sum_{j=l}^{r} [s_j =' c'] \cdot (sum_g[j-1] - sum_g[l-1]) \cdot (sum_d[r] - sum_d[j])$$

- 剩下的都是套路，应该不用我说了吧

1. 减少状态总数
2. 减少状态转移数
   - 预处理/前缀和
   - 部分和优化
   - 四边形不等式
     - 石子合并
     - 练习：[FJOI2004] 达尔文芯片问题
   - 决策单调性
   - 数据结构优化
3. 内存优化
4. 其他

- 当函数 $f(i,j)$ 满足 $f(a,c)+f(b,d) \le f(b,c)+f(a,d)$，且 $a \le b < c \le d$ 时，称 $f(i,j)$ 满足四边形不等式.
- 当函数 $f(i,j)$ 满足 $f(i',j') \le f(i,j)$，且 $i \le i' < j' \le j$ 时，称 f 关于区间包含关系单调.
- $s(i,j)=k$ 是指 $f(i,j)$ 这个状态的最优决策
- 如果一个区间 DP 方程满足四边行不等式，那么求 k 值的时候 $s(i,j)$ 只和 $s(i+1,j)$ 和 $s(i,j-1)$ 有关，所以可以以 $i-j$ 递增为顺序递推各个状态值最终求得结果，将 $O(n^3)$ 降为 $O(n^2)$
- 证明我放出来肯定没人看的。包括我。

```
for(int k=i;k<j;k++)...
for(int k=s[i][j-1];k<=s[i+1][j];k++)...
```

石子合并

- 在操场上沿一直线排列着 n 堆石子。现要将石子有次序地合并成一堆。规定每次只能选相邻的两堆石子合并成新的一堆，并将新的一堆石子数计为该次合并的得分。
- 我们希望这 n-1 次合并后得到的得分总和最小。$n \leq 300$

$$f(i,j) = min\left\{ f(i,k) + f(k+1,j) | i \leq k < j \right\} + sum_j - sum_{i-1}$$

- 通常如果没有乘号在里面一般就可以四边行不等式优化？
- 打表大法好！

$$f(i,j) = min\left\{ f(i,k) + f(k+1,j) | i \le k < j \right\} + sum_j - sum_{i-1}$$

- 通常如果没有乘号在里面一般就可以四边行不等式优化？
- 打表大法好！
- 如果不相邻？

$$f(i,j) = min\left\{ f(i,k) + f(k+1,j) | i \le k < j \right\} + sum_j - sum_{i-1}$$

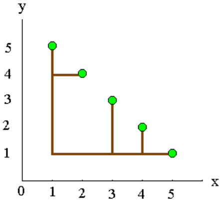
- 通常如果没有乘号在里面一般就可以四边行不等式优化？
- 打表大法好!
- 如果不相邻？哈夫曼树。如果是三堆，四堆，n 堆？

$$f(i,j) = min\left\{ f(i,k) + f(k+1,j) | i \leq k < j \right\} + sum_j - sum_{i-1}$$

- 通常如果没有乘号在里面一般就可以四边行不等式优化？
- 打表大法好！
- 如果不相邻？哈夫曼树。如果是三堆，四堆，n 堆？我过于傻逼考场上不会做结果就滚粗了 QaQ → [NOI2015] 荷马史诗

练习：[FJOI2004] 达尔文芯片问题

- 最近科学家们发现，将若干关键逻辑元按照电路板平面坐标系 2 维降序排列，经过演化，这些逻辑元自动按照 x, y 坐标方向联接成一棵树，树的每条边都平行于坐标轴。关键逻辑元构成这棵树的全部叶结点。这类树称为正交树。
- 有趣的是，达尔文芯片自动产生的正交树的总边长是所有这种正交树中总边长最小的。例如，将 5 个关键逻辑元分别置于电路板 xOy 坐标系中，则达尔文芯片自动产生的一棵正交树如下图所示，它的总边长为 12。



- 求总边长最小的正交树的总边长值。

$50\% : n \le 600;\ 100\% : n \le 3000$

- $f[i][j]$ 表示 $[i,j]$ 所有点连起来的最小代价。

$$f[i][j] = \min\left\{ f[i][k] + f[k+1][j] + y[k] - y[j] + x[k+1] - x[i] \mid i \le k < j \right\}$$

- 当你看到部分分是 $O(n^3)$ 暴力，100 分是 $O(n^2)$ 可过时，对，不要怀疑，就是四边形不等式！

- $f[i][j]$ 表示 $[i,j]$ 所有点连起来的最小代价。

$$f[i][j]=\min\left\{f[i][k]+f[k+1][j]+y[k]-y[j]+x[k+1]-x[i] \mid i \leq k < j\right\}$$

- 当你看到部分分是 $O(n^3)$ 暴力，100 分是 $O(n^2)$ 可过时，对，不要怀疑，就是四边形不等式!
- 小心发现，大胆猜想，不用证明!
- 实践是检验真理的唯一标准

1. 减少状态总数
2. 减少状态转移数
   - 预处理/前缀和
   - 部分和优化
   - 四边形不等式
   - 决策单调性
     - [POJ2823]Sliding Window
     - 练习：M 最大和
   - 数据结构优化
3. 内存优化
4. 其他

- 做动态规划时常常会见到形如这样的转移方程：

$$f[i] = optimize\left\{g(j) | L[i] \leq j < i\right\} + w[i]$$

其中 $L[1] \leq L[2] \leq \cdots \leq L[n]$

- 有这样一个性质：如果存在两个数 j,k，使得 $k \leq j$，而且 $g(k) \leq g(j)$ $(opt = max)$ 或 $g(j) \leq g(k)(opt = min)$，则决策 k 是毫无用处的。
- 根据 L[i] 单调的特性，如果 k 可以作为合法决策，那么 j 一定可以作为合法决策，又因为 j 比 k 要优 (注意：在这个经典模型中，“优”是绝对的，与当前正在计算的状态无关)，因此如果把表中的决策按照 j 排序的话，则 g(j) 必然不升 (opt=max) 或必然不降 (opt=min)。
- 因此使用单调队列即可将原本 $O(N^2)$ 的复杂度降至 $O(N)$

[POJ2823]Sliding Window

- 给你一个长度为 N 的数组，一个长为 K 的滑动的窗体从最左移至最右端，你只能见到窗口的 K 个数，每次窗体向右移动一位，如下表：

Window position	Min value	Max value
[1 3 -1] -3 5 3 6 7	-1	3
1 [3 -1 -3] 5 3 6 7	-3	3
1 3 [-1 -3 5] 3 6 7	-3	5
1 3 -1 [-3 5 3] 6 7	-3	5
1 3 -1 -3 [5 3 6] 7	3	6
1 3 -1 -3 5 [3 6 7]	3	7

- 你的任务是找出窗口在各位置时的 max value 和 min value。$N \leq 10^6$

- 我当然知道用 RMQ、线段树之类的东西直接上更无脑。
- $f[i]$ 表示 $(i-k,i]$ 的答案 $\rightarrow$ 以 i 结尾
- $fmin[i]=\min\left\{a[j] \mid i-k<j\leq i\right\}$，维护递增序列
- $fmax[i]=\max\left\{a[j] \mid i-k<j\leq i\right\}$，维护递减序列
- 插入 i 时把 $i-k$ 踢掉。

练习：M 最大和

- 输入一个长度为 n 的整数序列 $A_{1..n}$，从中找出一段连续的长度不超过 m 的连续子序列，使得这个序列的和最大。
  $n, m \le 100000$

练习：M 最大和

- 输入一个长度为 n 的整数序列 $A_{1..n}$，从中找出一段连续的长度不超过 m 的连续子序列，使得这个序列的和最大。
  $n, m \leq 100000$
- $sum_i = \sum_{j=1}^{i} A_j$
- $f[i] = \max\left\{sum_i - sum_{j-1} \mid i - m < j \leq i\right\}$，维护递减序列
- 如果长度还不能小于 k？

练习：M 最大和

- 输入一个长度为 n 的整数序列 $A_{1..n}$，从中找出一段连续的长度不超过 m 的连续子序列，使得这个序列的和最大。
  $n, m \leq 100000$
- $sum_i = \sum_{j=1}^{i} A_j$
- $f[i] = \max\left\{sum_i - sum_{j-1} \mid i - m < j \leq i\right\}$，维护递减序列
- 如果长度还不能小于 k？上面那个最小长度是 1，把 1 改成 k 就好了，注意入队出队顺序。

1 减少状态总数

2 减少状态转移数
- 预处理/前缀和
- 部分和优化
- 四边形不等式
- 决策单调性
- 数据结构优化
  - 和谐序列
  - 练习：[BZOJ2259] 新型计算机
  - 练习：[SJTU1123] 折线统计

3 内存优化

4 其他

- 听说 *coolinging* 没给你们讲树状数组？

- 听说 *coolinging* 没给你们讲树状数组？
- 就是一个能用 $O(logN)$ 的时间效率在序列上进行：修改某个元素的值、查询前缀和 的数据结构。注意，前缀和在这里可以指广义上的前缀和，比如前缀 max、前缀 min……
- 插入：

```
void add(int t,int x){for(;t<=n;t+=t&-t)s[t]+=x;}
```

- 查询：

```
int query(int t,int f=0){for(;t;t-=t&-t)f+=s[t];return
```

- 由于今天不是数据结构专场，因此不予展开解释，只要知道它能干嘛、记好代码就好了。

- 听说 *coolinging* 没给你们讲树状数组？
- 就是一个能用 $O(logN)$ 的时间效率在序列上进行：修改某个元素的值、查询前缀和 的数据结构。注意，前缀和在这里可以指广义上的前缀和，比如前缀 max、前缀 min……
- 插入：

```
void add(int t,int x){for(;t<=n;t+=t&-t)s[t]+=x;}
```

- 查询：

```
int query(int t,int f=0){for(;t;t-=t&-t)f+=s[t];return
```

- 由于今天不是数据结构专场，因此不予展开解释，只要知道它能干嘛、记好代码就好了。
- 如何优化：$f[i] = opt\left\{ f[j] + |g(j)| \right\}$ 把绝对值拆开，分类讨论求极值。

和谐序列

- 给定一个 n 个元素的序列 a[i]，定义“和谐序列”为序列的任何两个相邻元素相差不超过 K，求 a[i] 的子序列中“和谐序列”的个数。$n \leq 100000$

和谐序列

- 给定一个 n 个元素的序列 a[i]，定义“和谐序列”为序列的任何两个相邻元素相差不超过 K，求 a[i] 的子序列中“和谐序列”的个数。$n \le 100000$
- f[i] 表示以第 i 个元素结尾的“和谐序列”的个数。则：

$$f[i] = 1 + \sum_{j=1}^{i-1} [\, |a[i] - a[j]| \le K \,] \cdot f[j]$$

绝对值拆开：

$-K \le a[i] - a[j] \le K \ \rightarrow \ a[i] - K \le a[j] \le a[i] + K$

- 按顺序 1-n for 过去，每次以 a[i] 为下标，插入 f[i]；求 f[i] 时用树状数组求前缀和。注意需排序 + 离散 + 二分

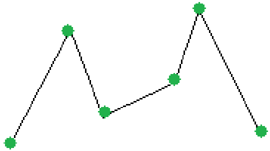
练习：[BZOJ2259] 新型计算机

- 新型计算机的输入很独特。假设输入序列中有一些数字 (都是自然数，包括 0)，计算机先读取第一个数字 s_1，然后顺序向后读入 s_1 个数字；接着再读一个数字 s_2，顺序向后读入 s_2 个数字……依此类推。不过只有计算机正好将输入序列中的数字读完，它才能正确处理数据，否则计算机就会进行自毁性操作！
- Tim 现在有一串输入序列。但可能不是合法的，也就是可能会对计算机造成破坏。于是他想对序列中的每一个数字做一些更改，加上一个数或者减去一个数，当然，仍然保持其为自然数。使得更改后的序列为一个新型计算机可以接受的合法序列。不过 Tim 还希望更改的总代价最小，所谓总代价，就是对序列中每一个数改变的绝对值之和。
- $N \leq 10^6$

- 子结构： $k \to (j \to (i \to N))$
- 记 f[i] 为从 i 到 N 的答案 $i \to (j \to N)$
- $f[i] = \min \left\{ f[j] + abs(a[i] - (j - i - 1)) \mid i < j \le N \right\}$
  1. $a[i] > j - i - 1$: $j < a[i] + i + 1$
     $f[i] = min(f[j] + a[i] - j + i + 1) = min(f[j] - j) + (a[i] + i + 1)$
  2. $a[i] < j - i - 1$: $j > a[i] + i + 1$
     $f[i] = min(f[j] - a[i] + j - i - 1) = min(f[j] + j) - (a[i] + i + 1)$
- 以 j 为下标，用树状数组分别维护 $f[j] - j$ 的前缀 min 和 $f[j] + j$ 的后缀 min
- 网络上一坨最短路的题解表示没看懂 @*hzwer*

练习：[SJTU1123] 折线统计

- 二维平面上有 n 个点 (x_i, y_i)，现在这些点中取若干点构成一个集合 S，对它们按照 x 坐标排序，顺次连接，将会构成一些连续上升、下降的折线，设其数量为 $f(S)$。如下图中，$1 \to 2, 2 \to 3, 3 \to 5, 5 \to 6$（数字为下图中从左到右的点编号），将折线分为了 4 部分，每部分连续上升、下降。



- 如图共 6 个点，$f(S) = 4$。现给定 k，求满足 $f(S) = k$ 的 S 集合个数。$n \le 50000$，$k \le 10$，$1 \le x_i, y_i \le 100000$

- 拐点是很明显的状态分割标志。按 x-y 排序
- f[k][i][0/1] 表示 [1,i] 中的某些点组成的折线有 k 部分，并且最后一部分以 i 为结尾，是上升 (1) 还是下降 (0)

$$f[k][i][0] = \sum_{j=1}^{i-1} [Y_j > Y_i] \Big(f[k][j][0] + f[k-1][j][1] \Big)$$

$$f[k][i][1] = \sum_{j=1}^{i-1} [Y_j < Y_i] \Big(f[k][j][1] + f[k-1][j][0] \Big)$$

- 剩下的都是套路，应该不用我说了吧

- C/C++ 内存换算：1M=262144(2^{18}) 个 int，1 个 long long=2 个 int，1 个 char/bool=0.25 个 int
- Pascal 不是太懂虽然我以前就是 Pas 选手……好像差不多？
- 听说三年以后 Pascal 就要退出 OI 了？

- C/C++ 内存换算：1M=262144(2^{18}) 个 int，1 个 long long=2 个 int，1 个 char/bool=0.25 个 int
- Pascal 不是太懂虽然我以前就是 Pas 选手……好像差不多？
- 听说三年以后 Pascal 就要退出 OI 了？
- 有的时候常常写完了 dp 方程，发现数组开不下，怎么办？

1. 减少状态总数
2. 减少状态转移数
3. 内存优化
   - 滚动数组
     - 经典模型：LCS
   - 减少状态总数
4. 其他

- 如果 f[i] 只和 f[i-1] 有关，那么根据 i 的奇偶性，用 f[2] 来存就好了。只要访问 f[i&1] 和 f[~i&1] 即可。

- 如果 f[i] 只和 f[i-1] 有关，那么根据 i 的奇偶性，用 f[2] 来存就好了。只要访问 f[i&1] 和 f[~i&1] 即可。
- 如果 f[i][j] 只和 f[i-1][..] 有关，那么用 f[2][..] 来存就好了。

- 如果 f[i] 只和 f[i-1] 有关，那么根据 i 的奇偶性，用 f[2] 来存就好了。只要访问 f[i&1] 和 f[~i&1] 即可。
- 如果 f[i][j] 只和 f[i-1][..] 有关，那么用 f[2][..] 来存就好了。
- 如果 f[i][j][k] 只和 f[i-1][..][..] 有关，那么用 f[2][..][..] 来存就好了。

- 如果 f[i] 只和 f[i-1] 有关，那么根据 i 的奇偶性，用 f[2] 来存就好了。只要访问 f[i&1] 和 f[~i&1] 即可。
- 如果 f[i][j] 只和 f[i-1][..] 有关，那么用 f[2][..] 来存就好了。
- 如果 f[i][j][k] 只和 f[i-1][..][..] 有关，那么用 f[2][..][..] 来存就好了。
- 如果 f[i][j][k] 只和 f[i-1][..][..] 和 f[i-2][..][..] 有关，那么用 f[3][..][..] 来存就好了。

- 如果 f[i] 只和 f[i-1] 有关，那么根据 i 的奇偶性，用 f[2] 来存就好了。只要访问 f[i&1] 和 f[~i&1] 即可。
- 如果 f[i][j] 只和 f[i-1][..] 有关，那么用 f[2][..] 来存就好了。
- 如果 f[i][j][k] 只和 f[i-1][..][..] 有关，那么用 f[2][..][..] 来存就好了。
- 如果 f[i][j][k] 只和 f[i-1][..][..] 和 f[i-2][..][..] 有关，那么用 f[3][..][..] 来存就好了。
- ……

经典模型：LCS

- 昨天 GTC 讲过了

```
f[i][j]=max(f[i-1][j],f[i][j-1]);
if(x[i]==y[j])f[i][j]=max(f[i][j],f[i-1][j-1]+1);
```

- 第一维可以滚动掉变成 f[2][..]

- 请看 ppt 最前面 已经讲过

1. 减少状态总数
2. 减少状态转移数
3. 内存优化
4. 其他

- DP 的优化还有很多，比如斜率优化、线段树优化、Hash 优化（插头 DP）……
- 也许过几年又会冒出其他奇奇怪怪的优化？