DP 及其优化

Wearry

Aug 9, 2018

Table of Contents

常见 DP 模型及其构造

- 序列 DP
- 树型 DP
- 数位 DP
- 状压 DP

Table of Contents

常见 DP 模型及其构造

- 序列 DP
- 树型 DP
- 数位 DP
- 状压 DP

DP 优化的方法

- 基本优化
- 斜率优化，凸优化
- 分治与数据结构优化
- 决策单调性优化，四边形不等式优化

Table of Contents

常见 DP 模型及其构造

- 序列 DP
- 树型 DP
- 数位 DP
- 状压 DP

DP 优化的方法

- 基本优化
- 斜率优化，凸优化
- 分治与数据结构优化
- 决策单调性优化，四边形不等式优化

题目选讲

常见 DP 模型及其构造

序列 DP

序列 DP 的常见模型有以下几种

序列 DP

序列 DP 的常见模型有以下几种

- 线性齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_{n-k}$ 推出, $O(nk)$.

序列 DP

序列 DP 的常见模型有以下几种

- **线性齐次常系数递推**, f_n **由** $f_{n-1}, f_{n-2}, \ldots, f_{n-k}$ **推出**, $O(nk)$.
- **线性非齐次常系数递推**, f_n **由** $f_{n-1}, f_{n-2}, \ldots, f_1$ **推出**, $O(n^2)$.

序列 DP

序列 DP 的常见模型有以下几种

- 线性齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_{n-k}$ 推出, $O(nk)$.
- 线性非齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_1$ 推出, $O(n^2)$.
- 区间问题的 DP, $f_{l,r}$ 与 $f_{l,k}, f_{k+1,r} | k \in [l, r]$ 有关, $O(n^3)$.

序列 DP

序列 DP 的常见模型有以下几种

- 线性齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_{n-k}$ 推出, $O(nk)$.
- 线性非齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_1$ 推出, $O(n^2)$.
- 区间问题的 DP, $f_{l,r}$ 与 $f_{l,k}, f_{k+1,r} | k \in [l, r]$ 有关, $O(n^3)$.
- 高维 DP, 一般要同时考虑多个序列, 还要在序列中各个位置移动.

序列 DP

序列 DP 的常见模型有以下几种

- 线性齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_{n-k}$ 推出, $O(nk)$.
- 线性非齐次常系数递推, f_n 由 $f_{n-1}, f_{n-2}, \ldots, f_1$ 推出, $O(n^2)$.
- 区间问题的 DP, $f_{l,r}$ 与 $f_{l,k}, f_{k+1,r} | k \in [l, r]$ 有关, $O(n^3)$.
- 高维 DP, 一般要同时考虑多个序列, 还要在序列中各个位置移动.
- 特殊区间问题模型, 一般情况下数据范围比较小, 只需要要设计出满足最优子结构的 DP 状态就行了.

ARC074 E RGB Sequence

给你一个长度为 N 的序列和 M 组约束条件, 每组条件形如 L_i, R_i, X_i, 表示序列上的 $[L_i, R_i]$ 中恰好有 X_i 种颜色, 现在要你用三种颜色给这个序列染色, 求满足所有约束的方案数.

$N, M \leq 300$

ARC074 E RGB Sequence

如果给定序列求区间中的颜色数，可以对每个右端点，记录它向左第一次出现某种颜色的位置. 按照类似的方式定义状态 $f_{i,r,g,b}$ 表示当前考虑到序列的第 i 个位置，每种颜色最后一次出现位置分别是 r, g, b 的方案数，然后每个把约束挂在右端点上，就可以删去不合法的状态.

ARC074 E RGB Sequence

如果给定序列求区间中的颜色数，可以对每个右端点，记录它向左第一次出现某种颜色的位置．按照类似的方式定义状态 $f_{i,r,g,b}$ 表示当前考虑到序列的第 i 个位置，每种颜色最后一次出现位置分别是 r,g,b 的方案数，然后每个把约束挂在右端点上，就可以删去不合法的状态．

不难发现对于一个合法的状态，一定有 $i=\max\{r,g,b\}$，可以省掉一维，复杂度 $O(N^3+N^2M)$．

树形 DP

树上的 DP 一般基于树的递归结构，常见模型相信大家都会，有几点需要注意:

树形 DP

树上的 DP 一般基于树的递归结构，常见模型相信大家都会，有几点需要注意:

- 图上的问题可以转化到生成树上考虑.

树形 DP

树上的 DP 一般基于树的递归结构，常见模型相信大家都会，有几点需要注意：

- 图上的问题可以转化到生成树上考虑.
- DFS 序或者 BFS 序可能是解决问题的关键.

树形 DP

树上的 DP 一般基于树的递归结构，常见模型相信大家都会，有几点需要注意：

- 图上的问题可以转化到生成树上考虑.
- DFS 序或者 BFS 序可能是解决问题的关键.
- 优化起来比较套路，一般可以考虑点分治，链剖和启发式合并.

EX 仙人掌 DP

- 仙人掌上的 DP 转化之后可以分成两个部分:
  1. 树边上的转移类似树形 DP.
  2. 环上的转移类似序列 DP.

EX 仙人掌 DP

- 仙人掌上的 DP 转化之后可以分成两个部分:
  1. 树边上的转移类似树形 DP.
  2. 环上的转移类似序列 DP.
- 两部分分开转移, 处理一些细节即可解决多数仙人掌 DP.

ARC086 E Smuggling Marbles

给出一棵 N 个点的有根树，初始时其中一些点上有一个石子，每次同时将所有石子从所在的点移动到父亲上，根节点上的石子移动到篮子里.
如果有一个点上的石子数大于 1 则移除所有石子，树上没有石子时结束.
求所有 2^N 种初始局面经过操作后篮子里石子的期望数量.

$N \leq 2 \times 10^5$

ARC086 E Smuggling Marbles

显然的一点是深度不同的点之间不会产生影响, 可以分开考虑.

ARC086 E Smuggling Marbles

显然的一点是深度不同的点之间不会产生影响，可以分开考虑.

解法一

一条从上到下的链不会影响石子的状态，所以可以对于每一个深度的点建虚树然后在虚树上 DP 即可，这样虚树的总点数是 $O(N)$ 的.

ARC086 E Smuggling Marbles

显然的一点是深度不同的点之间不会产生影响，可以分开考虑.

解法一

一条从上到下的链不会影响石子的状态，所以可以对于每一个深度的点建虚树然后在虚树上 DP 即可，这样虚树的总点数是 $O(N)$ 的.

解法二

可以维护子树内每个深度的信息，然后相同深度的可以合并，看起来是 $O(N^2)$ 的，可以使用长链剖分的技巧优化到 $O(N)$.

数位 DP

- 相对来说没有那么多的套路, 形式也比较简单易懂.

数位 DP

- 相对来说没有那么多的套路, 形式也比较简单易懂.
- 重点在于对题目性质的分析和模型的转化.

数位 DP

- 相对来说没有那么多的套路, 形式也比较简单易懂.
- 重点在于对题目性质的分析和模型的转化.
- 题面与数位没有直接关联的题目也可能用到数位 DP.

数位 DP

- 相对来说没有那么多的套路，形式也比较简单易懂.
- 重点在于对题目性质的分析和模型的转化.
- 题面与数位没有直接关联的题目也可能用到数位 DP.
- 数位 DP 的适用性不受进制影响，但如果存在进位要选择合适的 DP 方向.

Original Order

定义 f_n 表示将 n 的各个数位上的数排序后形成的数，例 $f_{50394} = 3459$. 给定 N，求 $f_1 + f_2 + \cdots + f_N$.

$N \leq 10^{700}$

Original Order

- f_n 中每一位的贡献是可以分开统计的.

Original Order

- f_n 中每一位的贡献是可以分开统计的.
- 每一位恰好等于多少的方案数并不是很好计算.

Original Order

- f_n 中每一位的贡献是可以分开统计的.
- 每一位恰好等于多少的方案数并不是很好计算.
- 考虑计算至少, 满足第 i 位 $\geq j$ 的方案, 即 $\geq j$ 的数字有 $\geq i$ 个.

Original Order

- f_n 中每一位的贡献是可以分开统计的.
- 每一位恰好等于多少的方案数并不是很好计算.
- 考虑计算至少, 满足第 i 位 $\geq j$ 的方案, 即 $\geq j$ 的数字有 $\geq i$ 个.
- $g_{i,j,k,0/1}$ 表示前 i 位有 j 个数位 $\geq k$, 是否大于 N 的方案数.

LIS

定义 f_n 表示将 n 的各个数位拆开形成序列的最长上升子序列, 给定 L, R, K, 求满足 $L \leq n \leq R$ 且 $f_n = k$ 的 n 的数量.

$1 \leq L \leq R \leq 10^{18}, K \leq 10$

LIS

考虑求 LIS 的经典二分法, 维护一个单调栈并每次换掉大于当前元素的最小元素.

LIS

考虑求 LIS 的经典二分法, 维护一个单调栈并每次换掉大于当前元素的最小元素.

在数位 DP 上处理可以用类似 DP of DP 的思路, 使用状压表示这个单调栈, 栈的操作直接在二进制位上操作就好了.

状压 DP

和数位 DP 类似，形式相对单一，重点在模型转换，不过既然讲了状压，顺便讲一些位运算的技巧：

状压 DP

和数位 DP 类似，形式相对单一，重点在模型转换，不过既然讲了状压，顺便讲一些位运算的技巧：

- `__builtin_popcount` 统计二进制下 1 的个数.

状压 DP

和数位 DP 类似，形式相对单一，重点在模型转换，不过既然讲了状压，顺便讲一些位运算的技巧：

- `__builtin_popcount` 统计二进制下 1 的个数.
- `__builtin_ctz`，`__builtin_clz` 统计二进制尾和头连续 0 的个数.

状压 DP

和数位 DP 类似，形式相对单一，重点在模型转换，不过既然讲了状压，顺便讲一些位运算的技巧：

- `__builtin_popcount` 统计二进制下 1 的个数.
- `__builtin_ctz`, `__builtin_clz` 统计二进制尾和头连续 0 的个数.
- `for(int i = all; i; i = (i - 1) & all) { ... }` 枚举子集.

AGC016 F Games on DAG

给出一个 N 个点，M 条边的 DAG，每条边的起点编号均小于终点编号. 两个人在图上博弈，先在 1 号点和 2 号点上分别放一个棋子，每次操作可以将一个棋子沿一条边移动。两人轮流操作，直到不能操作的人输.

求有多少边的子集，满足在只保留这个子集的图上先手必胜.

$1 \leq N \leq 15, 1 \leq M \leq \frac{N(N-1)}{2}$

AGC016 F Games on DAG

首先分析对于一个已知的图如何判定其是否合法.

AGC016 F Games on DAG

首先分析对于一个已知的图如何判定其是否合法.

那么显然这样的两个石子的游戏可以拆分成两个游戏的组合, 对于其中单个石子的情况我们可以用 SG 函数解决, 两个石子的情况就是 1 号点和 2 号点 SG 值的异或.

AGC016 F Games on DAG

首先分析对于一个已知的图如何判定其是否合法.

那么显然这样的两个石子的游戏可以拆分成两个游戏的组合, 对于其中单个石子的情况我们可以用 SG 函数解决, 两个石子的情况就是 1 号点和 2 号点 SG 值的异或.

这样满足条件的图中 1 号点和 2 号点 SG 值不同, 这个问题不是那么好计算, 考虑计算问题的反面, 即 1 号点和 2 号点 SG 值相同的图.

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

考虑这样的 x 和 y 之间的连边方案, 不难发现有以下几个限制:

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

考虑这样的 x 和 y 之间的连边方案, 不难发现有以下几个限制:

1. x 内部无连边, 即不存在 SG 值为 0 的点之间相连.

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

考虑这样的 x 和 y 之间的连边方案, 不难发现有以下几个限制:

1. x 内部无连边, 即不存在 SG 值为 0 的点之间相连.
2. x 连向 y 的边可以任意确定连或者不连.

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

考虑这样的 x 和 y 之间的连边方案, 不难发现有以下几个限制:

1. x 内部无连边, 即不存在 SG 值为 0 的点之间相连.
2. x 连向 y 的边可以任意确定连或者不连.
3. y 中的每一个点至少向 x 集合中的一个点连边.

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

考虑这样的 x 和 y 之间的连边方案, 不难发现有以下几个限制:

1. x 内部无连边, 即不存在 SG 值为 0 的点之间相连.
2. x 连向 y 的边可以任意确定连或者不连.
3. y 中的每一个点至少向 x 集合中的一个点连边.
4. y 集合内部的连边方案数等于 f_y .

AGC016 F Games on DAG

记 f_s 表示考虑了 s 中的点之后, 1 号和 2 号点 SG 值相同的方案数. 计算 f_s 时可以枚举它的 SG 值为 0 的子集 x, 并记 $y = s \setminus x$, 表示 SG 值不为 0 的子集.

考虑这样的 x 和 y 之间的连边方案, 不难发现有以下几个限制:

1. x 内部无连边, 即不存在 SG 值为 0 的点之间相连.
2. x 连向 y 的边可以任意确定连或者不连.
3. y 中的每一个点至少向 x 集合中的一个点连边.
4. y 集合内部的连边方案数等于 f_y .
5. 1 号点和 2 号点同时在 x 中或者同时在 y 中.

DP 优化的方法

基本优化

- 设计状态和转移的时候尽量剔除不必要的部分.

基本优化

- 设计状态和转移的时候尽量剔除不必要的部分.
- 冷静分析复杂度, 特别是有关树上 DP 的题目.

基本优化

- 设计状态和转移的时候尽量剔除不必要的部分.
- 冷静分析复杂度, 特别是有关树上 DP 的题目.
- 状压 DP 中的常数优化往往有意想不到的效果.

基本优化

- 设计状态和转移的时候尽量剔除不必要的部分.
- 冷静分析复杂度, 特别是有关树上 DP 的题目.
- 状压 DP 中的常数优化往往有意想不到的效果.
- 对于状态直接存储存不下的题目, 利用 `std::map` 或者哈希表.

例题

给出一棵 N 个点的树，求树上背包，每个点的体积是 1.

$N \leq 5000$

- 直接暴力做看起来复杂度是 $O(N^3)$ 的.

- 直接暴力做看起来复杂度是 $O(N^3)$ 的.
- 冷静分析后会发现复杂度其实可以做到 $O(N^2)$.

- 直接暴力做看起来复杂度是 $O(N^3)$ 的.
- 冷静分析后会发现复杂度其实可以做到 $O(N^2)$.
- 因为任意两个点恰好会在 LCA 处产生一次贡献.

斜率优化

- 用于最优化题目中快速找到最优解.

斜率优化

- 用于最优化题目中快速找到最优解.
- 可以通过推式子分析题目最优转移的凸壳形式.

斜率优化

- 用于最优化题目中快速找到最优解.
- 可以通过推式子分析题目最优转移的凸壳形式.
- 维护凸壳一般要分析凸壳上点的加入顺序选择合适的数据结构.

斜率优化

- 用于最优化题目中快速找到最优解.
- 可以通过推式子分析题目最优转移的凸壳形式.
- 维护凸壳一般要分析凸壳上点的加入顺序选择合适的数据结构.
- 跟区间问题结合起来可能涉及到区间凸壳.

斜率优化

- 用于最优化题目中快速找到最优解.
- 可以通过推式子分析题目最优转移的凸壳形式.
- 维护凸壳一般要分析凸壳上点的加入顺序选择合适的数据结构.
- 跟区间问题结合起来可能涉及到区间凸壳.
- 如果题目允许离线, 也可以用 CDQ 分治避免写动态凸包.

APIO 2014 Split the sequence

给出一个长度为 N 的非负整数序列, 要重复下面的操作 K 次: 选择一个序列, 然后把它分成两个新的非空序列.

每次操作后将获得那两个新序列和的乘积的得分. 求最大总得分.

$N \leq 10^5, K \leq \min(N-1, 200)$

APIO 2014 Split the sequence

首先转化以下要求的东西，发现就是对于每一对 i, j，如果它们不在同一个块中，就会产生 $a_i \times a_j$ 的贡献.

APIO 2014 Split the sequence

首先转化以下要求的东西，发现就是对于每一对 i, j，如果它们不在同一个块中，就会产生 $a_i \times a_j$ 的贡献.

于是可以设计出 $O(N^2K)$ 的算法，然后推一下式子发现，可以用单调队列优化做到 $O(NK)$.

凸优化

凸优化解决的是一类选择恰好 K 个某种物品的最优化问题，一般来说这样的题目在不考虑物品数量限制的条件下会有一个隐性的图像，表示选择的物品数量与问题最优解之间的关系.

凸优化

凸优化解决的是一类选择恰好 K 个某种物品的最优化问题，一般来说这样的题目在不考虑物品数量限制的条件下会有一个隐性的图像，表示选择的物品数量与问题最优解之间的关系.

问题能够用凸优化解决还需要满足图像是凸的，直观地理解就是选的物品越多的情况下多选一个物品，最优解的增长速度会变慢.

凸优化

解决凸优化类型的题目可以采用二分的方法，即二分隐性凸壳上最优值所在点的斜率，然后忽略恰好 K 个的限制做一次原问题.

凸优化

解决凸优化类型的题目可以采用二分的方法，即二分隐性凸壳上最优值所在点的斜率，然后忽略恰好 K 个的限制做一次原问题.

这样每次选择一个物品的时候要多付出斜率大小的代价，就能够根据最优情况下选择的物品数量来判断二分的斜率与实际最优值的斜率的大小关系.

凸优化

解决凸优化类型的题目可以采用二分的方法，即二分隐性凸壳上最优值所在点的斜率，然后忽略恰好 K 个的限制做一次原问题.

这样每次选择一个物品的时候要多付出斜率大小的代价，就能够根据最优情况下选择的物品数量来判断二分的斜率与实际最优值的斜率的大小关系.

理论上这个斜率一定是整数，由于题目性质可能会出现二分不出这个数的情况，这时就需要一些实现上的技巧保证能够找到这个最优解.

例题

将上一道例题中的 K 的范围改成 $\leq N$, 其它条件不变.

首先不难发现，分的段数越多分数会更大，所以我们可以二分分数的增长率.

首先不难发现, 分的段数越多分数会更大, 所以我们可以二分分数的增长率.

假设二分的值是 mid, 相当于转化成没有分段次数的限制, 但是每次分段都要额外付出 mid 的代价, 求最大化收益的前提下分段数是多少.

首先不难发现，分的段数越多分数会更大，所以我们可以二分分数的增长率.

假设二分的值是 mid，相当于转化成没有分段次数的限制，但是每次分段都要额外付出 mid 的代价，求最大化收益的前提下分段数是多少.

当然这个问题依然可以斜率优化，这样就可以在 $O(N \log W)$ 的复杂度内解决这个问题.

分治与数据结构优化

分治与数据结构优化

- 这一类题比较多，一般也比较套路.

分治与数据结构优化

- 这一类题比较多, 一般也比较套路.
- 大部分严格说来难点是数据结构而不是 DP 本身.

分治与数据结构优化

- 这一类题比较多，一般也比较套路.
- 大部分严格说来难点是数据结构而不是 DP 本身.
- 分治优化并不是简单地用分治优化某个值的计算，而是利用分治处理一些比较棘手的限制.

Myjnie

N 家洗车店从左往右排成一排. 有 M 个人要来消费, 第 i 个人会驶过 $[a_i, b_i]$ 之间的洗车店, 并选择这些店中最便宜的一个进行一次消费, 但是如果这个价格大于 c_i, 这个人就不会消费.

请给每家店指定一个价格, 使得所有人花的钱的总和最大.

$N \leq 50, M \leq 1000, 1 \leq a_i \leq b_i \leq N, c_i \leq 5 \times 10^5$

Myjnie

显然每家店的价格肯定等于某个 c_i，否则一定存在更优解.

考虑分治去掉最小值的限制，定义 $f_{i,j,k}$ 表示区间 $[i,j]$ 中的洗车店，最小值为 k 的最优解，转移的时候枚举最小值的位置，并计算跨越最小值的人会产生的消费即可.

决策单调性优化与四边形不等式优化

决策单调性优化与四边形不等式优化

- 严格来说两者都属于决策单调性优化，不过一个是一维的一个是二维的而已.

决策单调性优化与四边形不等式优化

- 严格来说两者都属于决策单调性优化，不过一个是一维的一个是二维的而已.
- 四边形不等式应该大部分人都是背代码的吧...

决策单调性优化与四边形不等式优化

- 严格来说两者都属于决策单调性优化，不过一个是一维的一个是二维的而已.
- 四边形不等式应该大部分人都是背代码的吧...
- 一维决策单调性优化的通用解法有两种，单调栈和分治.

决策单调性优化与四边形不等式优化

- 严格来说两者都属于决策单调性优化，不过一个是一维的一个是二维的而已.
- 四边形不等式应该大部分人都是背代码的吧...
- 一维决策单调性优化的通用解法有两种，单调栈和分治.
- 两种方法复杂度是一样的，但是感觉分治更加简单无脑一点.

珠宝

有 N 个珠宝，每个珠宝价值 C_i，能产生 V_i 的愉悦度，现在你有 M 元，问你最多能获得多大的愉悦度，对于 $M \in [1, K]$ 回答问题.

$N \leq 10^5, C_i \leq 300, V_i \leq 10^9, K \leq 5 \times 10^4$

珠宝

注意到 C_i 的取值比较小，可以按照 C_i 对珠宝进行分类，考虑同一类中的转移，显然可以得到两个结论：

珠宝

注意到 C_i 的取值比较小，可以按照 C_i 对珠宝进行分类，考虑同一类中的转移，显然可以得到两个结论：

1. 每次选择的一定是同一类中愉悦度从大到小排序后的一个前缀.

珠宝

注意到 C_i 的取值比较小，可以按照 C_i 对珠宝进行分类，考虑同一类中的转移，显然可以得到两个结论：

1. 每次选择的一定是同一类中愉悦度从大到小排序后的一个前缀.
2. 转移的过程中涉及的状态模 C_i 是同余的.

珠宝

注意到 C_i 的取值比较小，可以按照 C_i 对珠宝进行分类，考虑同一类中的转移，显然可以得到两个结论：

1. 每次选择的一定是同一类中愉悦度从大到小排序后的一个前缀.
2. 转移的过程中涉及的状态模 C_i 是同余的.

进一步地观察发现前缀和的增长率是单调的，同时在这些状态中的决策也是单调的，于是直接上优化就好了.

题目选讲

LOJ Round10 YQL 的生成树

对于可重实数集 $\{a_i\}$, 定义离差为: 对于任意实数 d, $\sum_i |a_i - d|$ 的最小值.

现在给出一个 N 个点, M 条边的无向联通图, 求最大离差生成树.

$N, M \leq 2 \times 10^5$

LOJ Round10 YQL 的生成树

首先对于已知的实数集计算离差就只要找中位数就好了.

LOJ Round10 YQL 的生成树

首先对于已知的实数集计算离差就只要找中位数就好了.

考虑对于一个给定的中位数 mid, 将大于这个中位数的边标为黑色, 其它标为白色. 这样我们可以枚举中位数 mid, 然后套用凸优化求恰好有 $\frac{n-1}{2}$ 条白边的图的答案.

LOJ Round10 YQL 的生成树

首先对于已知的实数集计算离差就只要找中位数就好了.

考虑对于一个给定的中位数 mid, 将大于这个中位数的边标为黑色, 其它标为白色. 这样我们可以枚举中位数 mid, 然后套用凸优化求恰好有 $\frac{n-1}{2}$ 条白边的图的答案.

实际上可以继续优化, 发现中位数与答案也是单调的, 可以直接二分中位数加上斜率的值, 然后判断白边的数量与 $\frac{n-1}{2}$ 的关系即可.

UNR2 梦中的题面

给定 M, N, C, B, 求满足以下条件的 M 元组 $\{x_1, x_2, \ldots, x_M\}$ 的数量:

- $0 \leq x_i \leq B^i - C$
- $\sum_i x_i < N$

$C \in \{0, 1\}, N \leq B^M, B, M \leq 50$

UNR2 梦中的题面

题目中的限制提示了要在 B 进制下思考. 可以先观察一下 B 进制意义下 $C = \{0, 1\}$ 时, x 的取值形式分别是什么.

UNR2 梦中的题面

题目中的限制提示了要在 B 进制下思考. 可以先观察一下 B 进制意义下 $C = \{0, 1\}$ 时, x 的取值形式分别是什么.

- $C = 1$ 的时候, x_i 的上限在 B 进制下是一排 $B - 1$.

UNR2 梦中的题面

题目中的限制提示了要在 B 进制下思考. 可以先观察一下 B 进制意义下 $C = \{0, 1\}$ 时, x 的取值形式分别是什么.

- $C = 1$ 的时候, x_i 的上限在 B 进制下是一排 $B - 1$.
- $C = 0$ 的时候, x_i 的上限在第 $i + 1$ 位是 1 然后接下来全是 0.

UNR2 梦中的题面

先思考 $C = 1$ 的情况怎么处理，发现对于 x_i 来说，B 进制下每个位置都是独立的，所以在每个位置上都有一些数可以取 $[0, B-1]$，直接数位 DP 即可.

UNR2 梦中的题面

先思考 $C = 1$ 的情况怎么处理，发现对于 x_i 来说，B 进制下每个位置都是独立的，所以在每个位置上都有一些数可以取 $[0, B-1]$，直接数位 DP 即可.

考虑 $C = 0$ 的情况，相比 $C = 1$ 时 x_i 多了选择 B^i 的方案. 这样的方案，必须满足前 i 位均为 0，第 $i+1$ 位为 1. 可以从高位向低位 DP，并且维护有多少个数已经选择了上限，转移强制这些数在当前位上为 0 即可，注意高位向低位 DP 的实现细节.

Druzyny

有 N 个人排成一排，现在要将它们按顺序分成若干组，对于第 i 个人，他希望自己所在的组的人数在 $[c_i, d_i]$ 之间，现在你想知道在分的组数尽可能多的情况下，有多少不同的分组方案.

$N \leq 10^6$

Druzyny

有两个限制的情况有一点不是很好处理，先考虑去掉其中一个限制.

Druzyny

有两个限制的情况有一点不是很好处理，先考虑去掉其中一个限制.

首先考虑对于每一个 i 找到一个最小的 g_i 满足:

$$i - g_i \leq \min\{d_j \mid j \in [i, g_i]\}$$

这样的 g_i 一定是单调的，所以可以维护一个指针与线段树预处理.

Druzyny

对于其中 c_i 的限制我们用分治去处理, 记当前区间为 $[l, r]$. 首先找到 k 满足:

$$c_k = \max\{c_i \mid i \in [l, r]\}$$

Druzyny

对于其中 c_i 的限制我们用分治去处理，记当前区间为 $[l, r]$. 首先找到 k 满足:

$$c_k = \max\{c_i \mid i \in [l, r]\}$$

接下来先处理左边区间的 DP 值，然后考虑左边对右边的贡献. 由于 c_k 最大，所以和 c 有关的限制现在只要考虑 c_k 就好了.

Druzyny

由于不是按照中点分治的，所以我们要保证复杂度的下界与较短的区间相关长度. 首先枚举右区间的每一个状态 i，这时候左边能够的决策一定是一段连续的区间，当 $g_i \leq l$ 时，每次右移相当于增加了决策区间的上限，可以 $O(1)$ 维护，显然上限增加的次数不大于较短区间的长度.

Druzyny

由于不是按照中点分治的，所以我们要保证复杂度的下界与较短的区间相关长度. 首先枚举右区间的每一个状态 i，这时候左边能够的决策一定是一段连续的区间，当 $g_i \leq l$ 时，每次右移相当于增加了决策区间的上限，可以 $O(1)$ 维护，显然上限增加的次数不大于较短区间的长度.

当这个上限达到 $k-1$ 后下限相同的 i 也是连续的，可以同时转移，这样的下限种类数也不会大于较短区间的长度. 看起来这样的复杂度是 $\log^2$ 的，因为每层分治还要在线段树上操作，但实际上仔细思考发现满足 $l < g_i$ 的区间最多只有一个，所以线段树的操作次数是均摊 $O(N)$ 的，总复杂度 $O(N \log N)$.

51nod 排列合并机

对于两个长度相等的排列 A, B, 定义这样两个排列的分数为将 A 中的元素按顺序插入 B 的空位中, 能够形成的本质不同的序列的数量.

现在给出 N, 求在所有 $N!$ 种排列中任选两个排列 (可以相同) 的分数期望.

$N \leq 100$

51nod 排列合并机

首先考虑这个问题的子问题，即如果给出了 A，B 两个排列，如何求出得分.

51nod 排列合并机

首先考虑这个问题的子问题，即如果给出了 A，B 两个排列，如何求出得分.

比较直观的一个想法是定义 $f_{i,j}$ 表示当前 A 选到了第 i 个数，B 选到了第 j 个数的本质不同前缀串数量，转移的时候选择 A 的开头或者选择 B 的开头，但是显然这样的状态定义会计算重复.

51nod 排列合并机

首先考虑这个问题的子问题，即如果给出了 A，B 两个排列，如何求出得分.

比较直观的一个想法是定义 $f_{i,j}$ 表示当前 A 选到了第 i 个数，B 选到了第 j 个数的本质不同前缀串数量，转移的时候选择 A 的开头或者选择 B 的开头，但是显然这样的状态定义会计算重复.

这样我们需要去重，而去重最重要的是想清楚每种序列被计算的次数.

51nod 排列合并机

定义一个序列为好的，当且仅当其由两个完全相同的排列构造而成，且没有任何一个前缀满足条件. 那么显然这样的好的序列会被计算两次(交换两次构造的顺序可以得到一样的结果).

51nod 排列合并机

定义一个序列为好的，当且仅当其由两个完全相同的排列构造而成，且没有任何一个前缀满足条件. 那么显然这样的好的序列会被计算两次(交换两次构造的顺序可以得到一样的结果).

对于转移过程中的一个中间状态，我们可以枚举它的一个好的后缀序列的长度，然后删除这样的贡献即可. 这样需要预处理 g_d 表示长度为 d 的好的序列的长度，可以先想一想这个子问题怎么做.

51nod 排列合并机

定义一个序列为好的，当且仅当其由两个完全相同的排列构造而成，且没有任何一个前缀满足条件. 那么显然这样的好的序列会被计算两次(交换两次构造的顺序可以得到一样的结果).

对于转移过程中的一个中间状态，我们可以枚举它的一个好的后缀序列的长度，然后删除这样的贡献即可. 这样需要预处理 g_d 表示长度为 d 的好的序列的长度，可以先想一想这个子问题怎么做.

因为这样的序列一定可以按顺序拆成两个排列，我们可以假定第一个排列已经固定下来，然后把第二个排列一个个拼上去，可以做到 N^2 的复杂度，也许有更优的做法.

51nod 排列合并机

现在回到原问题，根据刚才的分析我们发现，通过 $DPofDP$ 的技巧，我们只需要考虑一个状态再往后添加一个好的后缀能够转移到哪些状态即可去重.

51nod 排列合并机

现在回到原问题，根据刚才的分析我们发现，通过 $DPofDP$ 的技巧，我们只需要考虑一个状态再往后添加一个好的后缀能够转移到哪些状态即可去重.

记 $dp_{i,j,k}$ 表示当前在 A 中选到第 i 个位置，B 中选到第 j 个位置，并且这些位置中有 k 个相同的方案数.

那么转移显然：

$$dp(i,j,k) \times (j-k) \to dp(i+1,j,k+1)$$
$$dp(i,j,k) \times (i-k) \to dp(i,j+1,k+1)$$
$$dp(i,j,k) \times (n-i-j+k) \to dp(i+1,j,k)$$
$$dp(i,j,k) \times (n-i-j+k) \to dp(i,j+1,k)$$
$$-dp(i,j,k) \times g(d) \binom{n-i-j+k}{d} d! \to dp(i+d,j+d,k+d)$$