Dynamic Programming

fircone.

大量内容从我的博客复制而来。

备注：记得补“阶段”这个概念。

一、DP 的本质

一般 DP 的本质

- 状态：点。（带了值）
- 转移：边。
- DP：在 DAG 上推。（得到 / 更新 点的值）

特殊（类似 DP）

图不是 DAG。有两种思路：

- 解方程
  - 简单的：直接解（比如只有一个环）。
  - 复杂的：高斯消元。
    - 高斯消元。
    - 高斯-约旦消元。
  - 环 + DAG：解方程 + DP。
- 图论
  - 类似最短路：
    - Dijkstra 算法 / 类似 Dijkstra 的算法。
    - SPFA 算法 / 类似 SPFA 的算法。（迭代）
  - 其他的我可能还不知道。

二、DP 的要素

（我目前想起来的。）

- **状态**。无后效性，设计方式：[把之后要用的全部塞进状态里]（更新了，是全部塞到 状态或值里）。
- **转移方程**。就近转移，不重不漏。最值（min、max）、其他运算。
- **初值**。按定义 / 为了转移后得到的状态的值正确。[也许要为了之后正确用奇怪的定义。]（？）（好像是 重返现世 那题）。[注意特殊情况。]（之前写的，现在不知道例子是什么）
- **最终答案**。可能是一个状态，可能是多个状态运算起来。

可能一般按 状态 -> 转移方程 -> 初值、最终答案 的顺序思考。

三、DP 的转移方式

之前听说叫“填表法”和“刷表法”。

推到其他状态

在当前状态上做改动。个人认为更符合正向思维。

似乎转移对应关系复杂的时候用这种方式比较清晰方便。

从其他状态转移来

找之前的状态。个人认为更符合逆向思维。

似乎更常用。

似乎在决策方面优化 DP 的时候常用。比如 单调栈、单调队列、斜率、决策单调性、线段树之类的 DS 大力找决策点 优化 DP。

四、DP 的转移顺序

按拓扑序

（我知识水平非常有限，不太了解拓扑序，这里说拓扑序可能不止一种是不严谨的或错误的，但是就是这个意思 qwq。）

一般的转移顺序。本质上是按照 DAG 的拓扑序。

注意拓扑序可能不止一种，有时换一种顺序可能可以优化 DP，换顺序如：二维状态的 DP，先枚举哪一维后枚举哪一维可能可以颠倒。（如：把序列分割成给定个数个区间的题。）

记忆化搜索

避免了转移顺序的问题。

注意初始化 vis 数组（或者直接把 f 初始化成 -1 之类的）。

时间复杂度[一般要]（？）结合状态数来分析，但不一定等于状态数，可能还要算上转移和其他的代价。（如二维区间 DP。）

典型例子是数位 DP、二维区间 DP。

五、DP 的优化

- 空间优化
- 时间优化

- 状态方面的优化
- 转移方面的优化
    - 快速寻找决策点

- 批量转移
  - 从一批状态转移来
  - 转移到一批状态

1. 纯粹的空间优化

滚动数组 & 类似滚动数组的

本质：某个时刻，某些状态现在和以后都不会再被用到了，于是让新的状态覆盖掉这些状态。

滚动数组常用取模（特殊的：奇偶）来实现。

2. 时间优化 & 时间和空间一起优化的（综合优化）

1). 单调队列优化 DP

（它强于单调栈优化 DP，所以不写单调栈优化 DP 了，单调栈优化 DP 就是不用移动左端点的单调队列优化 DP，即转移范围的左端点不变）

作用：去除多余点 来 快速寻找决策点。

状态转移方程：

$$f_i = \min_{l_i \le j \le r_i} f_j$$

- 可能还有 只与 i 有关的数值、其他只与 j 有关的数值 和 常数 参与运算。总之这些都是定值（j 走过了，f_j 就成定值了）。
- $\max$ 和 $\min$ 同理，这里以 $\min$ 为例。
- **l_i 和 r_i 都随 i 增大而不降（这里假定是 f_1 到 f_n，且 $r_i < i$）。**

发现 l_i 和 r_i 都是单调不降的，l_i 右移相当于进队，而 r_i 右移相当于出队。于是直接用队列来维护。

前面的点显然会比后面的先出队，那么如果后面的点已经进队了且比前面的点优，这个前面的点就永远不会作为决策点了，于是直接让它出队即可。发现这样的过程形成了一个从前往后单调递增的队列，它就是单调队列。出队的过程其实就是在队尾踢点直到踢不动。队头的点即是决策点。

在线。

2). 斜率优化 DP

作用：去除多余点 来 快速寻找决策点。

单调队列优化 DP 无法处理 $c_i \times c_j$ 这种与 i 相关的和与 j 相关的乘起来的情况。这时就需要斜率优化 DP。

状态转移方程：

$$f_i = \min_{1 \le j \le i-1} a_i + b_j + c_i c_j$$

- $\max$ 和 $\min$ 同理。

下面两个方向都是在线（上面方程里和 j 有关的可以换成 f_j，总之走过了就变成定值了）。

有两个方向：

1°). 李超线段树（维护直线）

应用范围更广，但速度稍慢。

去 min 推式子。

$$f_i - a_i = c_j c_i + b_j$$

$$y = kx + b$$

把与 j 相关的作为直线信息，把与 i 相关的作为点的信息（直线 $x = \ldots$）

应用范围更广，但修改 2 只 log，查询 1 只 log。

可以放到树上，李超线段树合并。

2°). 队列（维护凸壳）

速度较快，但限制更多。

一些细节可能搞忘了，哪天复习来补。

$$b_j = -c_i c_j + f_i - a_i$$

$$y = kx + b$$

把与 j 相关的作为点的信息，把与 i 相关的作为直线的信息。

相当于拿一根已知斜率的直线去过每个点，看截距的最小值。那么只需要从下往上移动这个直线，看第一个过的点是哪个。

[斜率 单增 / 单减 时，可以用单调队列来维护凸壳。斜率单增且取 $\min$ 是维护下凸壳，斜率单减且取 $\max$ 是维护上凸壳。总之就是考虑斜率单增还是单减，取 $\max$ 还是取 $\min$，四种情况，**画图（也许也可以在脑子里想象出图）分析。**]（？）

[斜率没有单调性的时候也可能可以，只要点的横坐标有单调性就可以用队列维护凸壳，找决策点时在凸壳上二分。也要注意思考是上凸壳还是下凸壳。]（？）

[实现细节上，队列要保证一定有一个点在里面。]（？）

[要注意开始的时候那个位置 0 的取值，不一定直接是 0，要为了之后得到正确的值而取值。]（？）

还有关于精度、[0]（？？？？？；斜率？？？？？）的问题，待补。

听 xjy 说推出来的式子可以不止一种，只要斜率单调即可。

QwQ). 单调队列优化 DP、斜率优化 DP 的观察技巧

- 拆式子，变成能进行这两种优化的形式。
- [换 维度枚举顺序。]（？）

之前写的内容到此为止qwq。

接下来的内容会很杂。之后再整理吧。

说明：

1. 这一行以后的内容里，用 [ ] 框起来的是我不确定的（不确定正确性 / 不确定能不能这样称呼 / ...）。
2. 这一行以后的内容里，$\oplus$ 表示异或。

Sum Over Subset DP

- 好像是叫这个名字；缩写：SOSDP；中文名：[子集和 DP]。
- 作用：求高维前缀和（高维偏序）。
  - 具体地，它用于求解每一维上只有 $0/1$，且点分布密集的 [带权]（点有权值）高维偏序问题。
  - 用集合的思想理解，就是每个集合有一个权值，对每个集合求它的子集的权值之和。
  - 总结：从前缀和、偏序、集合三个角度理解。
- 思路：
  - 一位一位考虑。这里从低位到高位，但是显然考虑位的顺序没有影响。
  - 设 $f_{i,j}$ 表示子集的前 i 位（维）可以与 j 不同，其他位必须与 j 相同，当前集合是 j 时，j 的子集权值之和。
    - 当 j 的第 i 位为 0 时：$f_{i,j}=f_{i-1,j}$。（这一位只能选 0）
    - 当 j 的第 i 位为 1 时：$f_{i,j}=f_{i-1,j}+f_{i-1,j\oplus 2^i}$。（这一位可选 1（$f_{i-1,j}$）也可选 0（$f_{i-1,j\oplus 2^i}$））
  - 正确性说明：
    - 带入定义即可。显然不重不漏。
  - 空间优化：把 i 那一维滚掉。
    - 写法：类似 01 背包。
    - 事实上，j 那一维顺序枚举也不妨。因为在空间优化的写法下，只有 j 的第 i 位为 0 时 j 才可以被拿去更新其他状态的值，而第 i 位为 0 的 j 的值其实没变，即 $f_{i,j}=f_{i-1,j}$。
- 写法：
  - 这里采用顺序枚举 j 的写法。
  - 这里的 i,j 和前面说的 i,j 是反着的。
  - ```
    for(int i = 0; i < (1 << k); ++ i) f[i] = a[i];
    for(int j = 0; j < k; ++ j){
        for(int i = 0; i < (1 << k); ++ i){
            if(((i >> j) & 1) == 0) f[i | (1 << j)] += f[i];
        }
    }
    ```
  - 说明：代码未经过验证，可能有误，抱歉。
- 拓展：
  - 超集和 DP：反过来即可。不放代码了。
    ```

    - 更广义的高维前缀和：每一维不止 $0,1$。可以理解为可重集的子集和 DP。容易拓展得到做法；似乎可以写个前缀和优化；注意空间优化的写法不能顺序枚举集合了，而是要按照 01 背包的写法来写。
    - 高维差分：我还没学，咕咕咕。应该可以看这部分参考的第二篇文章来学。
- 和数位 DP（这里只讨论计算子集权值和的数位 DP）的关系：
    - 子集和 DP 能处理的（代表集合的数的）值域相对较小，而数位 DP 能处理的（代表集合的数的）值域可以很大。
    - 子集和 DP 能处理集合带任意权值的情况，而数位 DP 只能处理权值特殊的情况（权值与集合相关）。
- 参考：
    - （%%%，让我看懂子集和 DP 的一篇）「学习笔记」SOS DP - cyl06 - 博客园 (cnblogs.com)
    - [dp 小计] SOSdp - g1ove - 博客园 (cnblogs.com)
    - 【DP】解析 SOSdp（子集和 dp） - HinanawiTenshi - 博客园 (cnblogs.com)
    - 卢卡斯定理 - OI Wiki (oi-wiki.org)
- 例题：
    - Subset - HackerRank subset - Virtual Judge (vjudge.net)
        - sol 1:
            - 值域较小，于是考虑子集和 DP。
            - 子集和 DP 可以一次计算很多数的贡献，但现在的问题是每加入一个数都有新的贡献。
            - 于是我们考虑把一些数的贡献一起算。
                - 思路 1：CDQ 分治 / 二进制分组合并：发现不行，因为每次子集和 DP 的时间复杂度都是值域大小，而非元素个数。
                - 思路 2：思路 1 不行的原因启发我们采用更暴力的方式。于是采用根号 [重构]，即每根号次就重新算一次当前所有数的贡献。
            - 总结：子集和 DP + 根号 [重构]。
            - 不保证这个做法能过。我还没写代码也还没仔细算时间复杂度。
        - sol 2:
            - 总结：按位折半，一半修改，一半查询。这应该也算一种根号做法吧（按位折半是把指数变成了一半，相当于 [开了个根]）。
            - 这部分参考的第二篇文章写的是这个做法。我不知道能不能过。
    - Problem - B - Codeforces
        - 题面见网页右下方的 Statements。
        - sol:
            - 奇偶 $\to \mod 2$。
            - 组合数模较小 **质数** $\to$ 用 Lucas 定理。
                - 一直拆下去，发现 $\binom{x}{y} \mod 2 = 1$ 相当于（充要）二进制下，每一位都满足 x 的这一位 $\geq y$ 的这一位。
                - $\binom{x}{y} \equiv 1 \pmod 2$

                  $\Leftrightarrow$ 2进制下，每一位都满足x的这一位 $\geq y$的这一位

$$\Leftrightarrow x \text{ or } y = x$$

$$\Leftrightarrow x \text{ and } y = y$$

$$\Leftrightarrow X是Y的超集$$

$$\Leftrightarrow Y是X的子集$$

        - （代表集合的数的）值域较小，权值又没什么特殊性。于是不用数位 DP，用子集和 DP 来做。
    - kor
        - 如果想听我再扯一通绕了个大弯的莫反做法的话就讲一下。
        - 不完整的总结：此题中子集和 DP 作为工具，推着推着式子发现要用它了就用。
    - 上面这几道应该算是比较板，只是把子集和 DP 作为工具。只是大概让大家见一下子集和 DP 相关做法的一些套路，加深印象和理解。
    - 更多的题看这里吧：[SOS Dynamic Programming Tutorial] - Codeforces
        - 我还没怎么看，可能这篇文章里给的题我都还没做过。不知道质量如何，可能很厉害（我猜）。

[自动机 DP]

- 自动机作为一类重要的 [字符串数据结构]，能够保证 **不重不漏地 在限制内** 填信息，使得 DP 能够 **不重不漏地** 统计所有 **合法** 情况。
- 两种形式：
    - 自动机作为一种思维模型，实际不必真正建出自动机。
        - 适用于较简单的自动机。
        - 如：子序列自动机。
    - 真正建出自动机，在自动机及相关结构（[伴生的] [那棵] 树）上 DP。
        - 有的问题既可以在自动机上 DP，也可以在 [伴生的] [那棵] 树上 DP。
        - 适用于较复杂的自动机。
        - 如：AC 自动机、后缀自动机。
- 例题：
    - Erase Subsequences - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
        - 感觉是好题啊。
        - CF1303E序列自动机+dp-CSDN博客
        - 题解 CF1303E 【Erase Subsequences】 - 洛谷专栏 (luogu.com.cn)
        - 题解 CF1303E 【Erase Subsequences】 - 洛谷专栏 (luogu.com.cn)
        - 这样 DP 会不会取重、如果不会是为什么：我还没弄懂。
        - 数据范围 **较** 小（也可能比较大）使时间 **较** 宽裕时，不妨先枚举使得问题更容易 DP。
    - [P3041 USACO12JAN] Video Game G - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
        - AC 自动机（要在 [fail 树] 上推一推） + 简单最值 DP。（注意滚动数组实现的细节。）
    - Anthem of Berland - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
        - KMP 自动机 + 简单最值 DP。
        - [[时间复杂度里乘了 [个] 字符集大小。]] 如果把字符集变得很大？

        - 一个 trick（对 KMP 自动机和 AC 自动机都有效，对于其他自动机我还没怎么想）：因为填字符的时候是填成一个串，所以不用预处理出压缩路径的转移边，可以直接把跳 fail 的过程融入 DP。但具体怎么融入我还不会。
        - 我从这里看到的，可以去学：经典AC自动机DP - autoint - 博客园 (cnblogs.com)
        - 这个 trick **好像** P3041（上一道）也能用。我还没想清楚。
    - [P4052 JSOI2007] 文本生成器 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
        - AC 自动机 + [（广义）数位 DP]（应该算是吧）（连实现方式都是数位 DP 的记忆化搜索）。
    - Wireless Password - HDU 2825 - Virtual Judge (vjudge.net)
        - 状压（据一篇博客）。
    - chino with string - Gym 103115A - Virtual Judge (vjudge.net)
        - 矩阵加速（据一篇博客）。
    - 上面这两道的做法见（我还没仔细看）：在AC自动机上dp - 樱与梅子 - 博客园 (cnblogs.com)
    - [P2292 HNOI2004] L 语言 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
        - 我对这道题的印象有点深。
        - $1 \le |s| \le 20$
        - 状压。
    - **（提醒自己）**这里可能会有一道我中考前出（[口胡]）的题。现在（2024.11.10）我几乎忘了。如果我想起来了这道题并且想起了这回事就可能会写。
    - 还有一个我随便出的题：给一个序列，求本质不同上升子序列的个数。
        - **好像** 有这题：[蓝桥杯国赛真题]：本质上升序列_本质不同的最长上升子序列-CSDN博客
- 参考：
    - CF1303E序列自动机+dp-CSDN博客
    - 题解 CF1303E 【Erase Subsequences】 - 洛谷专栏 (luogu.com.cn)
    - 题解 CF1303E 【Erase Subsequences】 - 洛谷专栏 (luogu.com.cn)
    - 在AC自动机上dp - 樱与梅子 - 博客园 (cnblogs.com)
    - 经典AC自动机DP - autoint - 博客园 (cnblogs.com)
    - 深度理解AC自动机与dp - 铃狐sama - 博客园 (cnblogs.com)
    - [蓝桥杯国赛真题]：本质上升序列_本质不同的最长上升子序列-CSDN博客
- 进阶：
    - DP 套 DP：内层 求值 / 判定 DP 建出 DFA（确定性有限状态自动机），外层 计数 DP 在这个 DFA 上进行。
        - 参考：【学习笔记】DP 套 DP - SoyTony - 博客园 (cnblogs.com)

DP 求本质不同子序列的个数

- 写 [自动机 DP] 的例题时突然想起来了。
- 发现现在自己只会 [[子序列自动机 + 统计路径数]]（应该可以这样做吧？）这一种方法了。
- 放篇在网上找到的博客：对求本质不同子序列个数的一点理解 - hzy1 - 博客园 (cnblogs.com)
- 参考：对求本质不同子序列个数的一点理解 - hzy1 - 博客园 (cnblogs.com)

拉格朗日插值优化 DP

- ~~夹带私货。~~之前总结过了，这次就不怎么写了。见我之前的博客：[做题笔记] 拉格朗日插值 - huangkxQwQ - 博客园 (cnblogs.com)
- 分两种：
  - 先 DP，最后统一拉插优化。
  - 分段优化，DP 的每一个阶段都要拉插。

Slope Trick

- 我很喜欢！但似乎比较冷门。
- 一类数据结构优化 DP：
  - 状态很多，于是每个阶段的状态用数据结构一起维护。
  - 转移时：继承 & 批量修改。继承也许需要合并（树上）。
- 但是如果修改用常规的数据结构很难实现呢？此时我们考虑利用维护的东西的特殊性质。
- Slope Trick 的作用：
  - 优化 $f(x)=dp_x$ 是分段一次凸函数（连续）的最值 DP。
- Slope Trick 的本质：
  - 用特殊方式和数据结构来优化具有特殊性质的 DP。
  - 维护特殊点（关键点）来快速修改查询。
- 分段一次凸函数的[本质]：类似 [离散微积分的求导]。斜率（[一阶导]）有单调性。
- 分段一次凸函数的性质：
  - 两个分段一次凸函数相加还是分段一次凸函数。新函数的分段点集合就是原来两个函数的分段点集合的并集。（从斜率的角度思考。）
  - 凸函数加一次函数还是凸函数。（因为一次函数的斜率是常数，而凸函数斜率单调。）
- 分段一次凸函数的维护方式：
  - 如果 斜率都是整数且斜率变化量很小（比如相邻两段斜率的差是 $1,2$ 之类的）：
    - 不带权。把斜率变化（加或减）1 的横坐标丢到一个 **可重集** 里。注意一个横坐标上斜率变化了多少就要把几个它丢进去。本质：差分。
    - 另外还要维护[最左一段或最右一段]的函数信息。
    - 容易发现这样就能确定整个函数。
  - [否则：
    - 点带权。] 我还不会。
  - 下面都以这里的第一种（不带权）为例。
  - 这里的“不带权”其实是权值为 1。
- 凸性质和最值 DP 的联系：
  - [凸函数] 的斜率具有单调性。那么我们只需要找到斜率为 0 的某个点就可以确定一个最值（试想二次函数的 [顶点]），而找到开头、末尾的那两个点就可以确定另一个最值。

- 用数据结构来维护那个可重集：
  - 这种数据结构要能快速地找到斜率为 0 的点（实际上可能并不存在斜率为 0 的一段，但我们把斜率每次变化 1 的过程中会使得一个点的斜率在某时为 0（只要斜率有正有负的话））。
  - 于是我们需要把可重集分成两段，又发现每次向可重集中加入一个点只会造成两段的分割线进行很少步数的移动，即只会造成很少的点从一段进入另一段。这启发我们尝试 [对顶数据结构]。
  - 不需要使两段内部有序，只需要知道最值；加入的是一个较自由的值而不是最值。对顶堆完美地契合了我们的要求，而对顶栈不行。
  - 记左边的堆为 L，右边的堆为 R。
- 维护各种修改（函数）：
  - 为了方便，这里我们称 [左 / 右] 的那段函数为“段”，称分段点的集合为“点”。
  - 相加：段相加（k, b 分别相加），点合并。
  - 加一次函数：段相加，把 R 中的 k（一次函数的斜率）个点丢到 L 里。
  - 前缀 / 后缀取最值：直接清空一个集合。画图理解。
  - 平移：直接改段，点打平移标记。
  - 翻转：直接改段，点打翻转标记。
- 求最值：[L 的堆顶和 R 的堆顶就是最值横坐标的范围。]
- 统计答案（两种做法）：
  - 记录决策点（[一般是] 最值处的横坐标）。
  - 还原图像。
- 证明是分段一次凸函数的方法：
  - 初始。
  - 分段一次凸函数相加还是分段一次凸函数。
  - 分段一次凸函数加一次函数还是分段一次凸函数。
- 显然 Slope Trick 属于整体 DP。其实它也可以上树（树上合并），用 [可并堆] 来实现（如：[APIO] 烟火表演 那题）。
- 参考：
  - （有些几乎是抄这篇）（%%%）【学习笔记】Slope Trick - CCComfy - 博客园 (cnblogs.com)
  - Slope Trick 总结 - rui_er - 博客园 (cnblogs.com)
- 例题：
  - P4597 序列 sequence - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)（后面跟着的题单里也有）
  - 想讲上面这道和烟火表演，但我还不会烟火表演。/dk /dk /dk
  - LCP 24. 数字游戏 - 力扣（LeetCode）
  - slope trick - 题单 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
  - Slope Trick：一种让人意想不到的优化方式 - 题单 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

WQS 二分优化（不止 DP）

- [WQS 二分又名凸优化、带权二分。]
- 解决的问题：
  - 一类特殊的最优化问题：
    - 给出 k，要求恰好选 k 个 或 恰好选 k 个……（比如有黑边和白边，要求恰好选 k 条白边）；$f(x)$ 有凹凸性（$f(x)$ 指的是选择恰好 x 个时的最优值，而不是其他的 DP 状态或其他的什么东西）。
- 作用：使用二分去掉“恰好选 k 个”的限制。
- 本质：凹凸性保证了斜率单调，利用这个单调性来二分求斜率，用一条直线去 [切] 这个凸壳，使得能 [切] 到横坐标为 k 的点，此时得到的 $f(k)$ 即为答案。（我们不知道也并不需要知道所有 $f(x)$，而是每次根据斜率算一个 $f(x)$，由算 $f(x)$ 的过程得到这次 [切] 到的 x，利用 x 是多少来决定斜率应该多还是少，最终使得恰好切到 k）
- 实现上，把 **每个** 受限制物品（如：有黑边和白边，要求恰好选 k 条白边，那么白边受限制）的价值都加上斜率。
  - 为什么？具体怎么做？
    - $f(x)=kx+b$。注意这里的 k 是斜率而不是恰好选 k 个的 k。
    - 我们通过 DP 或其他手段，不管限制，求最值，得到的是 b 的最值和此时的 x。
    - 那么用 $kx+b$ 还原出 $f(x)$ 即可。
- 问题：有可能一段点都可以用某个斜率 [“切”] 到。
  - 解决方法：优先选更多的点，即优先选这一段点里最靠右的一个（可能横坐标大于 k）。如果题目保证有解，就一定可以用恰好 k 个限制物品拼出来。
    - 注意：最后还原出答案的时候要用二分到的斜率、截距（b）和 **限制** k（不是二分时求出的 x）。如果用二分时求出的 x 来还原，就是 $f(x)$ 而不是 $f(k)$ 了。
- 画图理解。
- 例题：
  - （记得找一道最小生成树、找一道 DP。）
  - [P2619 国家集训队] Tree I - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)
  - nkp 提的 [P4767 IOI2000] 邮局 加强版 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)。我还不会。
  - [[[把规定区间个数的区间划分问题转化为任意区间个数的区间划分问题。]]]
- 参考：
  - [总结] wqs 二分 - ¶凉笙 - 博客园 (cnblogs.com)
  - 感觉这一篇里“算法思想”中对 WQS 二分的讲述比我写的自然：[总结] wqs二分学习笔记 - YoungNeal - 博客园 (cnblogs.com)
  - 看了看“细节处理”和“例题”：wqs 二分浅谈 - hyl天梦 - 博客园 (cnblogs.com)
  - 关于 WQS 二分优化费用流：wqs二分 学习笔记 - 洛谷专栏 (luogu.com.cn)
  - wqs 二分练习题 - 题单 - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)

这里本来有点分树优化动态换根 DP 的，但是那是我暑假想的了，当时没记，现在搞忘了。/dk /dk /dk

矩阵快速幂优化（不止 DP）& 动态 DP

- 注：这里的 DP 包括简单递推。
- 矩阵乘法优化 DP：
  - 用一个向量表示状态和 DP 值，用矩阵表示转移。
  - 本质：把转移变成实在的、可维护的东西。
  - 利用矩阵乘法的 **结合律** 加速转移：
    - 当转移矩阵只有一个或不同的个数非常少时，用矩阵快速幂算，再乘初值向量。
    - 当转移矩阵不同的个数很多，而 有修改操作 或 要取出某一部分来 DP 时：用数据结构来维护转移矩阵的乘积（动态 DP）。
  - **注意** [矩阵乘法是从右往左算的]、[矩阵乘法没有交换律]。
  - 适用范围：计数 DP。
- 广义矩阵乘法。
  - 懒得写了，放篇讲得特别好的博客：「笔记」广义矩阵乘法与 DP - Luckyblock - 博客园
  - 广义矩阵乘法允许使用其他的运算组合，从而处理其他类型的 DP（如：最值 DP）。
  - 使用限制（使新运算满足结合律）：$(\otimes, \oplus)$ 满足 $\oplus$ 有交换律，$\otimes$ 有 [[交换律]]、结合律，且 $\otimes$ 对 $\oplus$ 有分配率。
    - 矩阵乘法：$(\times, +)$。
  - 可以用到的运算组合（我目前见过的）：$(+, \max)$。
- 关于图论：
  - 邻接矩阵（无边权）矩阵快速幂：走 k 步，从 u 到 v 的路径数。（定长路径方案数）
  - [邻接矩阵]（有边权）广义矩阵快速幂（用 $(+, \min)$）：走 k 步，从 u 到 v 的最短路长度。类似 [Floyd] 算法，但是 [这个相当于一步步走，再用快速幂优化]。（定长路径最短路）
  - 常用一些建图方法来实现一些功能，比如建一个终止结点来把定长的限制变成长度不超过……的限制。
- 例题：
  - CF718C Sasha and Array
    - 题解 CF718C Sasha and Array - 洛谷专栏
  - [P3597 POI2015] WYC - 洛谷 | 计算机科学教育新生态
    - sol:
      - 边权很小，n 很小，k 巨大。于是我们考虑二分答案，拆边后跑邻接矩阵快速幂统计路径数来验证。
      - 拆边方式：
        - 每个点 u 建两个虚点 u', u''。连边：$u'' \to u', u' \to u$。当一条边 (u, v) 边权为 1 时，连边 $u \to v$；边权为 2 时，连边 $u \to v'$；边权为 3 时，连边 $u \to v''$。
        - 不能对于每个点只建一个虚点，既管入也管出。否则就可能导致这个虚点使 [并不存在的路径存在]。

        - 矩阵快速幂的一个细节：
            - 此题中 **统计的是走 $\le k$ 步的路径数而不是走 $= k$ 步的路径数**，于是我们添加一个虚点 0 作为 **终止结点**，每个点 u（**不包含 u' 和 u''**，因为显然不能没走完某条边就不走了）都连向虚点 0，**虚点 0 自己连向自己。**
            - 不知道能不能从矩阵的角度推式子来做。
        - 关于二分和倍增：
            - 矩阵快速幂本质就是倍增。用二分会多一个 $\log$，而用倍增可以边算矩阵快速幂边验证，于是可以没有这个 $\log$。
            - 此题中倍增的具体写法：边预处理 2 的幂边找倍增范围，再从高位到低位填（倍增），填的时候要更新矩阵，要拿矩阵来验证。
    - [P2151 SDOI2009] HH去散步 - 洛谷 | 计算机科学教育新生态
        - 题目里的一些意思：
            1. 有重边。
            2. 不允许 **连续** 重复经过某一条边。
            - 应该是无向图。
        - sol:
            - 有重边：似乎仍可以用邻接矩阵（两点之间有几条边就是几）。
            - 不允许 **连续** 重复经过某一条边：[不是很会直接做]。
            - （先是 **无向边拆成有向边**）于是 **化边为点。**
                - 把一条无向边拆成两条有向边，把每条有向边当成一个点，把原图的点作为边拆成的点之间的联系，依据原图的点来在新图中连边。特殊地，同一条无向边拆成的两条有向边对应的点之间不连边。
        - 参考：[题解 P2151 【SDOI2009]HH去散步】 - 洛谷专栏
    - [P2886 USACO07NOV] Cow Relays G - 洛谷 | 计算机科学教育新生态
        - 前面提到的 [广义邻接矩阵快速幂] 求固定步数最短路。
    - [P6569 NOI Online #3 提高组] 魔法值 - 洛谷 | 计算机科学教育新生态
        - 参考：[题解 P6569 【NOI Online #3 提高组]魔法值（民间数据）】 - 洛谷专栏
        - sol:
            - 把每个点的魔法值作为向量，把邻接矩阵作为转移矩阵。
            - 考虑如何 [刻画] 条件。用乘法和 [01 矩阵]（邻接矩阵）来限定拿来更新某个点的点。
            - 于是定义广义矩乘 $(\times, \oplus)$。（此处 $\oplus$ 表示异或。）
            - 注意到一般情况下 $\times$ 对 $\oplus$ 并没有分配率。但是这里转移矩阵是 01 矩阵，而 01 矩阵经过这样的广义矩乘仍然是 01 矩阵。如果参与运算的只有 $0, 1$，$\times$ 对 $\oplus$ 就是有分配率的。因此这里转移矩阵的广义乘法仍然具有结合律。
            - 矩阵快速幂题多询问：
                - 指的是：转移矩阵不变，每次询问初始向量可以不同。
                - 每个询问直接跑矩阵快速幂时间复杂度可能大了。
                - 时间复杂度更优秀的做法：预处理出转移矩阵的 2 的幂 次方，每次询问用初始向量去乘。

            - 这样时间复杂度更小的原因：向量乘矩阵是 $O(n^2)$ 的，而矩阵乘矩阵是 $O(n^3)$ 的。
        - 有个疑问：此题中向量并不只含有 $0,1$，又要用向量和矩阵相乘多次，不会出问题吗？
            - 不会，因为证了转移矩阵相乘是有结合律的，而向量和转移矩阵的幂相乘的过程并不依赖结合律。
            - 其实相当于把式子拉通写，在后面那一堆转移矩阵里加了一些括号，但向量[还是在外面的]。
        - 似乎还有 bitset 做法，我还不会。
    - P4719 【模板】"动态 DP"&动态树分治 - 洛谷 | 计算机科学教育新生态
        - 我刚刚学会（2024.11.9）。放一篇讲得特别好的题解，我就暂时不写了。
        - 题解 P4719 【模板】动态dp - 洛谷专栏
- 参考：
    - 「笔记」广义矩阵乘法与 DP - Luckyblock - 博客园
    - 动态DP - NuclearReactor - 博客园
    - 学习笔记：矩阵快速幂与图论 - f2021ljh - 博客园
    - [题解 P2151 【SDOI2009]HH去散步】 - 洛谷专栏
    - [题解 P6569 【NOI Online #3 提高组]魔法值（民间数据）】 - 洛谷专栏
    - 题解 P4719 【模板】动态dp - 洛谷专栏

分治 FFT

- 不一定是优化 DP，其实是优化某种式子的计算，但是如果 DP 式子长成这个样子就可以优化 DP 了。
- 本质：CDQ 分治（每次把范围切成两半，用左边的信息更新右边的信息，此时左边相当于已知的静态信息，可以利用 CDQ 分治把较在线的东西用这种离线的方式来求）。
- 分治 FFT 解决的问题：已知 $g_1,g_2,\dots,g_n$ 和 f_0，$\forall 1\le i\le n,f_i=\sum_{j=1}^{i}f_{i-j}g_j$，求 $f_1,f_2,\dots,f_n$。
- 做法：
    - 注意到一个个依次计算 f_i 不好优化，而这个式子很像 [卷积]，于是我们考虑把一些东西合到一起转移。
    - 于是用 CDQ 分治 左对右算贡献 的思想：
        - 当前处理的区间是 $[l,r]$，即当前要计算 f_l 到 f_r。
        - 如果 $l=r$，直接返回即可。
        - 取 $mid=\lfloor\frac{l+r}{2}\rfloor$。
        - 递归处理 $[l,mid]$ 和 $[mid+1,r]$。
        - 现在我们知道了 f_l 到 f_{mid}，现在要计算它们对 f_{mid+1} 到 f_r 的贡献。
        - 设它们对 f_x 的贡献为 w_x，那么有 $w_x=\sum_{i=l}^{mid}f_ig_{x-i}$。
        - 为了变成卷积的形式，要使 i 的范围上限到一个和 x 有关的值（这样说不准确，可能是 $x+C$ 这种形式，C 在这一次贡献计算中为定值），于是我们令 f_{mid+1} 到 f_r 都为 0。现在就有 $w_x=\sum_{i=l}^{x-1}f_ig_{x-i}$。

        - 为了变成卷积的形式，我们把 i 变得从 0 开始，即把新的 i 设置为原来的 $i-l$，于是有 $f_x=\sum_{i=0}^{x-1-l}f_{i+l}g_{x-i-l}$。
        - 现在 f 和 g 的 [底数] 并不是很好看，但 i 的系数一个是 1 一个是 -1，是可以通过平移变成卷积形式的。于是我们令 $f'_i=f_{i+l},g'_{x-1-l-i}=g_{x-i-l}$。那么就有 $f_x=\sum_{i=0}^{x-1-l}f'_ig'_{x-1-l-i}$。
        - 再定义一个 f''_x 来 [算卷积]，算完之后平移回 f_x。定义、平移的方式和上一条类似：令 $f''_{x-1-l}=f_x$。
        - 卷积用 FFT 系列来加速即可。
        - 把分治每一层的时间复杂度 [加起来]，得到总的时间复杂度是 $O(n\log^2 n)$。
    - 一个小优化（参考博客如是说，但我没验证）：

> 对于递归到范围较小的序列，直接暴力卷积，常数更小。

- 参考：[多项式算法](Part 5)分治FFT 学习笔记 - LanrTabe - 博客园 (cnblogs.com)

FFT 优化生成函数合并（乘积）

- 合并有几种（不止这几种）：
    - 线段树合并：合并时走几步就会删除几个（大概）。
    - 启发式合并：单次合并的时间复杂度是 **小** 的结构的大小 乘 单点插入 / 单点修改的时间复杂度。
    - [FFT 合并]：单次合并的时间复杂度 [大概] 是 **大** 的结构的大小 乘 FFT 的一只 $\log$。
        - 其实就是生成函数的乘积。可能也可以从 DP 的角度理解。
        - [例：] 方案数背包合并。
            - 当然最优化背包也可以转成方案数背包，但我不知道是所有时候都能转还是只有一些时候能转。
- 线段树合并和启发式合并只需要合并就行了，而 FFT 合并要考虑的就多了。
- 把 FFT 合并的过程看成一棵（二叉）树，那么合并的总时间复杂度就是树上每个结点的长度之和，也即树上每一层的总长度之和。上面那些层是满的，也就是这些（一）层的总长度是 n。
- 如果我们要合并的每个点一开始的长度都是 1，那么直觉告诉我们 让树尽量平衡、树高尽量低 就可以实现很快速的合并。
- 实现上，我们用一个小根堆来维护，每次取出长度最短的两个点，从堆中删除它们，把它们合并，再把合并的结果丢进堆里。
- 时间复杂度：$O(n\log^2 n)$。因为合并的那棵树共有 $\log$ 层，每层长度为 n，还带 FFT 的一只 $\log$（深度加 1 这只 $\log$ 只是减 1）；而堆里的结点不超过 n 个，因此堆的时间复杂度被覆盖了。
- 另外还有一个树上合并的版本，要用点分治（或许边分治更可以？）来保证复杂度。如果有时间且我想得起来这事我就写。
- 参考：快速傅里叶变换 - OI Wiki (oi-wiki.org)

整体 DP

- 稍微提一下。

- 感谢 xwb 告诉我这个东西。
- [其实是个很宽泛的概念]。
- 就是用数据结构维护有共性的一批批状态，通过数据结构进行继承和单点、批量修改。
  - 线段树、平衡树、线段树合并（支持上树）、平衡树合并（支持上树）、堆（[维护最值来转移]）。
- 参考（%%%）（感觉这篇博客里的题可以去练，但是可能需要一些 DS 方便的能力储备）：高级算法指北——浅谈整体dp - 烟山嘉鸿 - 博客园
- 例题：
  - Minimax，就是线段树合并那道。
  - 其实 Slope Trick 的题应该也算。

把斜率优化 DP 和决策单调性优化 DP 的原因：

- 它们都很重要。（个人感受）
- 它们都可能需要数形结合。

斜率优化 DP（再提）

- 据 决策单调性 - JueFan - 博客园，斜率优化也属于决策单调性，但这里单独拿出来说。我还不懂它为什么属于决策单调性。
- 数形结合[、线性规划]。
- 很多都 [需要] 画图分析！可以动笔也可以在脑子里画。
- 前面写了一些，但有些地方不清楚。这里对于它的几种实现方式和使用条件进行说明。
- 两种推式子思路：
  - 可能决策点作为直线，当前状态作为竖直直线，求交点（由当前状态的信息得出的竖直直线 和其他直线 的交点，而不是所有交点）纵坐标最值。
  - 可能决策点作为点，当前状态作为斜率固定的直线，[[切]]，求截距最值。
-
  - 对于第一种思路，直接用李超线段树做即可。
  - 现在我们只讨论第二种思路的做法。
- 维护什么：
  - 求 $\max$：维护上凸壳。
  - 求 $\min$：维护下凸壳。
  - 关于上面两条的正确性：思考斜率为不同值的时候从上往下（求截距 $\max$） / 从下往上（求截距 $\min$）碰到的第一个点是什么。
- 如何维护凸壳：
  - 点的横坐标有单调性：用 单调栈。[单调栈的“单调”在于斜率单调。]
    - 通过比较斜率来判断是否删除一个点。
      - **注意** 精度问题。遇精度问题可以尝试转除法为乘法，[注意负数对不等号的影响]。
      - [[**注意** [斜率] 为 0 的时候。]]
    - 另一个需要 **注意** 的地方是任意时刻单调栈里必须至少有一个点。

        - [这是因为需要算斜率。]
        - [一般] 一开始时要把横坐标为 0 那个点加进去。注意这个点的值（初值）应该和后面的点的各个值的意义相契合。
    - 顺带一提：按 x 坐标排序后用这个方法跑一遍上凸壳跑一遍下凸壳 **好像** 就是求二维凸包的算法之一。
  - 点的横坐标没有单调性：我还不会。因此后面的内容暂时（2024.11.9）都默认点的横坐标有单调性。
- 怎么求最值：
  - 假设我们现在已经得到了凸壳。具体地，假设我们已经知道了凸壳的每一个 [转折点]，并且它们已经按照 x 坐标排序。
  - [最值点就是那个两边线段斜率夹着当前斜率的点。]
    - 如果没有这样的点：[只需要找最左或最右的点即可，[也就是找斜率最接近当前斜率的地方]]。
  - 斜率有单调性：
    - 取 $\max$（上凸壳），且 斜率单调不升。
    - 取 $\min$（下凸壳），且 斜率单调不降。
    - 都是用单调队列（拓展原来的单调栈）：
      - 左端点处删除：为了查询时直接用左端点作为答案。
      - 右端点处删除：为了维护凸壳。
      - 右端点处添加：为了 [向凸壳中] 加入新的点。
  - 斜率没有单调性：在单调栈上二分。
- 参考：
  - 辰星凌的博客QAQ
  - 决策单调性 - JueFan - 博客园

决策单调性优化 DP

- 放一篇讲得特别好的博客（%%%%%%）：决策单调性 - JueFan - 博客园
- 四边形不等式：
  - 若 $\forall a \le b \le c \le d, w(a,c)+w(b,d) < w(a,d)+w(b,c)$，则称 w 满足 四边形不等式。
  - 简单记：交叉优于包含。
- 区间包含单调性：
  - 若 $\forall a \le b \le c \le d, w(b,c) \le w(a,d)$，则称 w 满足 区间包含单调性。
  - 简单记：小区间优于大区间。
  - 实质：$w(a,b)$ 关于 a 单减，关于 b 单增。
- 决策单调性：
  - 令 $p(i)$ 表示 $f_{\dots,i}$ 的最优决策点在 i 这一维上的 最小 / 最大 数值，若 $\forall i < j, p(i) \le p(j)$，则 [$f_{\dots}$]（我不确定是不是 f 而非 $f_{\dots}$）满足决策单调性。（这里 $\dots$ 表示的是状态的其他部分。）
    - 决策单调性通常体现在一维上，但也有多维的。

- 最小后缀问题：$f_j = \min_{i \le j} w(i, j)$。
  - 定理：若 w 满足四边形不等式，则 f 满足决策单调性。
- 区间拆分问题（区间拆分 DP）：
  - $f_j = \min f_i + w(i, j)$。（区间个数任意，在线，二分队列）
    - 优化方法：二分队列。放在后面。
  - $f_j = \min g_i + w(i, j)$。（规定区间个数，离线，分治）
    - 优化方法：分治。放在后面。
  - 定理：若 w 满足四边形不等式，则 f 满足决策单调性。
  - 可以忽略只和 i 有关、只和 j 有关的项。决策单调性不依赖 f，只依赖 w 的四边形不等式性质。
- 区间合并问题（区间 DP）：$f_{l,r} = \min f_{l,k} + f_{k,r} + w(l, r)$，此时记 $p(l, r)$ 为最小 / 最大的 k。
  - 引理：若 w 同时满足 区间包含单调性 和 四边形不等式，则 $f_{l,r}$ 也满足四边形不等式。
  - 定理：若 f 满足四边形不等式，则 $p(l, r-1) \le p(l, r) \le p(l+1, r)$。
    - 优化方法：$[l, r-1]$ 和 $[l+1, r]$ 的长度都比 $[l, r]$ 的长度小 1，因此可以在 DP 时记录 p，用 $p(l, r-1)$ 和 $p(l+1, r)$ 来夹出 $p(l, r)$ 的范围，在这个范围内枚举。
    - 时间复杂度分析：区间长度相同的每层的总枚举次数是 n。
      - 我还不知道为什么。
- 二分队列（在线）：
  - 考虑每个 i 会成为哪些 j 的决策点。由决策单调性可知，在每个时刻，可能有一些 i 对应 j 是一段区间，可能有另一些 i 不会再作为决策点了；对于可以作为决策点的 i，递增的 i 对应着从左到右的一段段区间，这些区间不重不漏地覆盖了所有还没求出 DP 值的点。
  - 我们尝试使用 [单调] 队列来维护：
    - 每个元素都是一个三元组 (l, r, p)，表示 $[l, r]$ 目前的最优决策点都是 p。
    - 初始时：队列中只有 $(1, n, 1)$。
    - 我们枚举 $k = (2 \to n)$ 作为决策点插入进队列中。插入决策点 k 时：
      - 与队尾 (l, n, p) 比较，如果 k 作为 l 的决策点比 p 还要优秀，那 p 就不会再用到了，于是弹出队尾；重复此过程直到弹不出队尾。
      - 如果有队尾，找到队尾和 k 的决策分界点；在 w 好求的前提下，这个分界点可以用二分求出。
  - 在线在于计算中需要用到 f。
  - 适用于：（可用决策单调性优化的）区间个数随意的区间划分问题。
- 分治（离线）：
  - 大力找中间点的决策点，通过它把可能的决策点也分成两半，分治下去即可。注意分治下去的可能决策点要包含找到的这个点。
  - 因为序列分治是取 [中点] 分治的，即使决策点分治的那棵树长得很神秘也不妨。显然每一层的决策点个数是 n [这个级别的]，而一共有 [大概] $\log$ 层，因此总的时间复杂度是 $O(n \log n)$。
  - 离线在于计算中不需要用到 f，需要用到的是 g，而此时整个 g [都] 是已知的。
  - trick: 移动端点计算 w：
    - $w(l, r)$，l 是可能的决策点，r 是分治时的 mid。

        - 如果并不能较快地（如：$O(1)$）计算 $w(l,r)$，但是能较快地（这里以 $O(1)$ 为例）移动 l,r 来求出 $w(l,r)$，我们仍可以保证 $O(n\log n)$ 的时间复杂度。
            - 整个分治过程统一用一组 l,r。
            - 让每轮移动（处理一对区间）之后都还原。
            - 就按分治的顺序来，不用像莫队那样先排序。
        - 时间复杂度分析：
            - 左右端点分开考虑。
                - [[左端点每次从 $p(l)$ 或 $p(r)$ 跳到 $p(mid)$。]]
                - [[右端点每次从 l 或 r 跳到 mid。]]
            - [左右端点的移动长度都是区间长级别的。]
            - 事实上，不还原时间复杂度并不会变大，而且可能会更快。因为不还原的话原来还原走的就变成有需要再走。
        - 该 trick 可以支持端点单调变化，只需要支持撤销。类似回滚莫队。但它和回滚莫队我都还不会。
    - 适用于：（可用决策单调性优化的）规定区间个数（为某个数值）的区间划分问题。
- 参考（几乎整个都是抄）：[决策单调性 - JueFan - 博客园]

2024.11.8

2024.11.9

2024.11.10