倍增专题

广附信奥

广州大学附属中学

2023 年 8 月

- 有 n 个点，升序给出每个点到起点的距离。
- 有编号为 $1 \sim n$ 的 n 只青蛙分别在第 $1 \sim n$ 个点上，每次它们会跳到距离自己第 k 近的点上。
- 如果有相同距离的点，就跳到下标更小的点上。
- 求跳 m 次之后，依次给出 $1 \sim n$ n 只青蛙所在的点编号。
- $1 \leq k < n \leq 10^6$,$1 \leq m \leq 10^{18}$，
- $1 \leq p_1 < p_2 < \ldots < p_n \leq 10^{18}$

- m 非常大，暴力模拟时间复杂度为 $m * n$ 肯定会超时，如何优化？

- m 非常大，暴力模拟时间复杂度为 $m*n$ 肯定会超时，如何优化?
- 我们可以预处理一个倍增数组 $step[i][j]$，表示从 i 点出发跳了 2^j 步所到的点的编号
- 通过倍增数组，我们从每个点出发跑 m 次总的时间复杂度 $nlog(m)$

- m 非常大，暴力模拟时间复杂度为 $m * n$ 肯定会超时，如何优化?
- 我们可以预处理一个倍增数组 $step[i][j]$，表示从 i 点出发跳了 2^j 步所到的点的编号
- 通过倍增数组，我们从每个点出发跑 m 次总的时间复杂度 $nlog(m)$
- 如何预处理 $step[i][0]$，即距离每个点的第 k 近点的编号?

- 先考虑最一般的情况，对于 1 号点，第 k 近一定是第 $k+1$ 号点

- 先考虑最一般的情况, 对于 1 号点, 第 k 近一定是第 $k+1$ 号点
- 对于 2 号点呢?

- 先考虑最一般的情况, 对于 1 号点，第 k 近一定是第 $k+1$ 号点
- 对于 2 号点呢? 对于 2 号点，我们发现，可能第 k 近是第 $k+2$ 号点，也可能是第 $k+1$ 号点

- 先考虑最一般的情况，对于 1 号点，第 k 近一定是第 $k+1$ 号点
- 对于 2 号点呢？对于 2 号点，我们发现，可能第 k 近是第 $k+2$ 号点，也可能是第 $k+1$ 号点
- 我们可以把算法抽象出来：

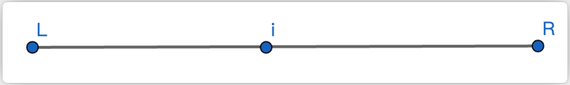


图 1: [L,R] 区间共有 k+1 个点

POI2010 Frog
————如何快速预处理倍增数组

- 先考虑最一般的情况，对于 1 号点，第 k 近一定是第 $k+1$ 号点
- 对于 2 号点呢？对于 2 号点，我们发现，可能第 k 近是第 $k+2$ 号点，也可能是第 $k+1$ 号点
- 我们可以把算法抽象出来：

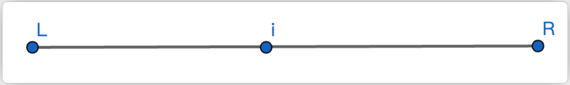


图 1: [L,R] 区间共有 k+1 个点

- 当 dis(R+1,i+1)<dis(i+1,L) 时，说明点 L 是 i+1 的第 k+1 近，区间整体右移，否则不动

- 时间复杂度?

- 时间复杂度？时间复杂度 $O(n)$

- 时间复杂度？时间复杂度 $O(n)$
- 我们发现 $log(m) * n \approx 60 * 10^6$
- 假如我们开的是 int 数组，那么需要多少字节？

- 时间复杂度？时间复杂度 $O(n)$
- 我们发现 $log(m) * n \approx 60 * 10^6$
- 假如我们开的是 int 数组，那么需要多少字节？
- 我们需要开一个 int 数组，所需的字节就是 $60 * 10^6 * 4$
- $60 * 10^6 * 4/1024/1024 \approx 228MB$

- 时间复杂度？时间复杂度 $O(n)$
- 我们发现 $log(m) * n \approx 60 * 10^6$
- 假如我们开的是 int 数组，那么需要多少字节？
- 我们需要开一个 int 数组，所需的字节就是 $60 * 10^6 * 4$
- $60 * 10^6 * 4/1024/1024 \approx 228MB$
- 如何优化内存空间？

- 时间复杂度？时间复杂度 $O(n)$
- 我们发现 $log(m) * n \approx 60 * 10^6$
- 假如我们开的是 int 数组，那么需要多少字节？
- 我们需要开一个 int 数组，所需的字节就是 $60 * 10^6 * 4$
- $60 * 10^6 * 4/1024/1024 \approx 228MB$
- 如何优化内存空间？ $step[i][j] = step[step[i][j-1]][j-1]$，我们每次倍增当前 i，只用到了上一层的数据，所以可以用滚动数组优化
- 优化后的空间复杂度为 $O(n)$

- 有一个大小为 n 的等边三角形，第 i 行包含 i 个数字
- 对这个等边三角形，求对于每个大小为 k 的子三角形，子三角形内数字的最大值之和。
- $1 \leq k < n \leq 3 * 10^3$

- 尝试用 DP 的方法
- n 最大为 $3 * 10^3$ 推测状态可能是 2 维或 3 维

- 尝试用 DP 的方法
- n 最大为 $3 * 10^3$ 推测状态可能是 2 维或 3 维
- 定义 $dp[i][j][p]$ 以（i，j）为顶点，边长为 p 的三角形的数字中的最大值

- 尝试用 DP 的方法
- n 最大为 $3 * 10^3$ 推测状态可能是 2 维或 3 维
- 定义 $dp[i][j][p]$ 以（i，j）为顶点，边长为 p 的三角形的数字中的最大值
- 枚举状态的时间复杂度为 n^3，空间复杂度 n^3

- 我们发现可以通过倍增的方式枚举 p

[CCC2019] Triangle: The Data Structure ————优化

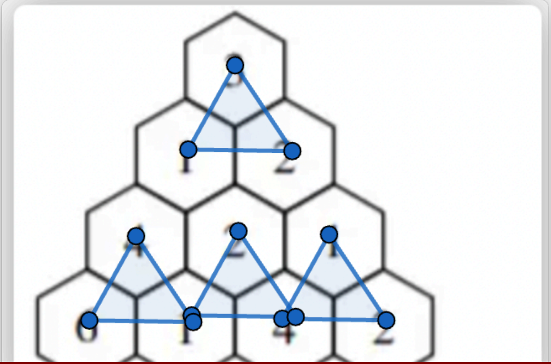
- 我们发现可以通过倍增的方式枚举 p 这时状态定义是?

[CCC2019] Triangle: The Data Structure ————优化

- 我们发现可以通过倍增的方式枚举 p 这时状态定义是?
- $dp[i][j][p]$ 以（i，j）为顶点，边长为 2^p 的三角形的数字中的最大值
- 枚举状态的时间复杂度和空间复杂度均为 $n^2 log(n)$

- 我们发现可以通过倍增的方式枚举 p 这时状态定义是?
- $dp[i][j][p]$ 以（i，j）为顶点，边长为 2^p 的三角形的数字中的最大值
- 枚举状态的时间复杂度和空间复杂度均为 $n^2 log(n)$
- 如何转移?

- 我们发现可以通过倍增的方式枚举 p 这时状态定义是?
- $dp[i][j][p]$ 以（i，j）为顶点，边长为 2^p 的三角形的数字中的最大值
- 枚举状态的时间复杂度和空间复杂度均为 $n^2 log(n)$
- 如何转移?



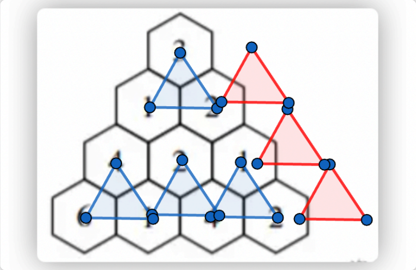
[CCC2019] Triangle: The Data Structure ————优化

- 枚举到当前 (i,j,p), 状态转移就是把三角形分成上下两半，上面一个边长为 2^{p-1} 的三角形，下面 $2^{p-1}+1$ 个边长为 2^{p-1} 的三角形，从中求出最大值

[CCC2019] Triangle: The Data Structure
————优化

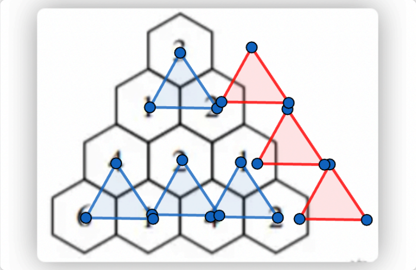
- 枚举到当前 (i,j,p)，状态转移就是把三角形分成上下两半，上面一个边长为 2^{p-1} 的三角形，下面 $2^{p-1}+1$ 个边长为 2^{p-1} 的三角形，从中求出最大值
- 对于每一个状态，都需要这样枚举吗？

- 枚举到当前 (i,j,p)，状态转移就是把三角形分成上下两半，上面一个边长为 2^{p-1} 的三角形，下面 $2^{p-1}+1$ 个边长为 2^{p-1} 的三角形，从中求出最大值
- 对于每一个状态，都需要这样枚举吗？



[CCC2019] Triangle: The Data Structure ——————优化

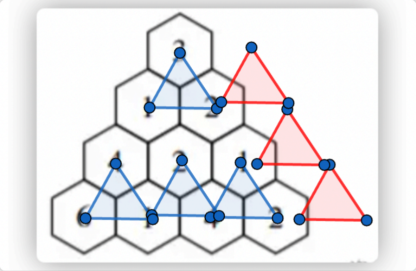
- 枚举到当前 (i,j,p)，状态转移就是把三角形分成上下两半，上面一个边长为 2^{p-1} 的三角形，下面 $2^{p-1}+1$ 个边长为 2^{p-1} 的三角形，从中求出最大值
- 对于每一个状态，都需要这样枚举吗？



- 状态转移可以用滑动窗口优化，时间复杂度均摊为 $O(1)$

[CCC2019] Triangle: The Data Structure
————优化

- 枚举到当前 (i,j,p)，状态转移就是把三角形分成上下两半，上面一个边长为 2^{p-1} 的三角形， 下面 $2^{p-1}+1$ 个边长为 2^{p-1} 的三角形，从中求出最大值
- 对于每一个状态， 都需要这样枚举吗？



- 状态转移可以用滑动窗口优化， 时间复杂度均摊为 $O(1)$
- 最后统计答案时也是类似的，总的时间复杂度 $n^2log(n)$ 同样可以用滚动数组将空间优化为 n^2

[CTSC2011] 幸福路径

————问题描述

- 有向图,n 个点 m 条边，每个点有个权值，蚂蚁从某个点开始，初始体力为 1，每经过一条边，体力会变为原来的 p(0 < p < 1) 倍
- 每经过一个点，会得到一个幸福值，为它当时的体力与该点权值 $w[i]$ 的乘积。
- 问你对走过的路径中，幸福值之和最大是多少
- $n \leq 100$，$m \leq 1000$, $p \leq$, $w[i] \leq 100$

- 因为只要走过一个点就会有幸福值，那么肯定是走的点越多越好。

- 因为只要走过一个点就会有幸福值，那么肯定是走的点越多越好。
- 假如定义 $step[i][j][k]$ 表示才能够 i 走到 j，走了 k 步的最大幸福值之和。然后用类似于 floyd 算法的方式转移状态

[CTSC2011] 幸福路径
————思路

- 因为只要走过一个点就会有幸福值，那么肯定是走的点越多越好。
- 假如定义 $step[i][j][k]$ 表示才能够 i 走到 j，走了 k 步的最大幸福值之和。然后用类似于 floyd 算法的方式转移状态
- 什么时候结束状态转移？

[CTSC2011] 幸福路径
————思路

- 因为只要走过一个点就会有幸福值，那么肯定是走的点越多越好。
- 假如定义 $step[i][j][k]$ 表示才能够 i 走到 j，走了 k 步的最大幸福值之和。然后用类似于 floyd 算法的方式转移状态
- 什么时候结束状态转移？当 p 小于某个精度时，因为不会产生贡献，所以结束

- 因为只要走过一个点就会有幸福值，那么肯定是走的点越多越好。
- 假如定义 $step[i][j][k]$ 表示才能够 i 走到 j，走了 k 步的最大幸福值之和。然后用类似于 floyd 算法的方式转移状态
- 什么时候结束状态转移？当 p 小于某个精度时，因为不会产生贡献，所以结束
- 我们发现这样子枚举很容易很慢，如何优化？

- 因为只要走过一个点就会有幸福值，那么肯定是走的点越多越好。
- 假如定义 $step[i][j][k]$ 表示才能够 i 走到 j，走了 k 步的最大幸福值之和。然后用类似于 floyd 算法的方式转移状态
- 什么时候结束状态转移？当 p 小于某个精度时，因为不会产生贡献，所以结束
- 我们发现这样子枚举很容易很慢，如何优化？
- 定义 $step[i][j][k]$ 为从 i 走到 j，期间走了 2^k 步点最大幸福值
- 因为我们每次枚举一次 k，p 都会变成 p*p，缩小的非常快的，很快就会到达不会产生贡献的大小。

- 有 n 座山峰，每座山峰有他的高度 h_i。有些山峰之间有双向道路相连，共 m 条路径，每条路径有一个困难值，这个值越大表示越难走。
- q 组询问，每组询问询问从点 v 开始只经过困难值小于等于 x 的路径所能到达的山峰中第 k 高的山峰，如果无解输出 -1。
- 对于 100% 的数据，$n \leq 10^5$，$0 \leq m, q \leq 5 \times 10^5$，$h_i, c, x \leq 10^9$。

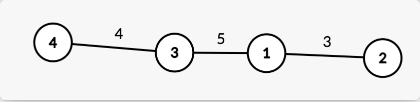
- 如何快速找到一个点步经过边权为 k 的边能走到的点的集合?

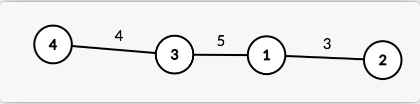
- 如何快速找到一个点步经过边权为 k 的边能走到的点的集合？对原本的点建一棵 kruskal 重构树

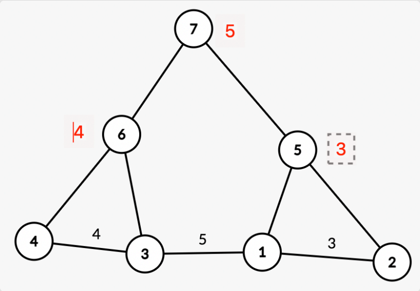
- 如何快速找到一个点步经过边权为 k 的边能走到的点的集合？对原本的点建一棵 kruskal 重构树
- kruskal 重构树算法:
- 将边按照边权从大到小排序。
- 从小到大遍历边，如果边权为 w 的边连接两点 (u,v)，则求出并查集中 (u,v) 的代表元素 (a,b) 。
- 新建一个结点 f，其点权为 w，并从 f 建两条边连向 (a,b)，在并查集中将 (a,b) 的代表元素都置为 f

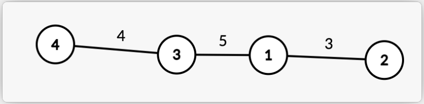
[ONTAK2010]Peaks

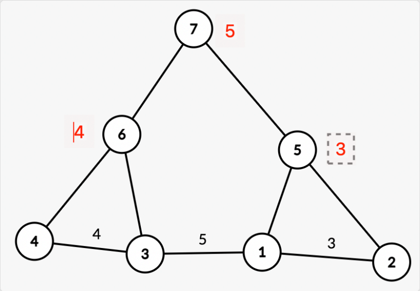
————kruskal 重构树

- 

- 
4
4
3
5
1
3
2


7
5
6
4
5
3
4
4
3
5
1
3
2

- 



- 如何快速找到一个点步经过边权为 k 的边能走到的点的集合?

- 对于每组询问，从 v 点在不经过点权超过 x 苦难值的情况下尽可能往上走，假设 e 是最后走到的点，那么就有，e 为根的子树中，叶子都是可以走到的，且叶子编号的范围 [L,R] 可以提前用数组记录下来

- 对于每组询问，从 v 点在不经过点权超过 x 苦难值的情况下尽可能往上走，假设 e 是最后走到的点，那么就有，e 为根的子树中，叶子都是可以走到的，且叶子编号的范围 [L,R] 可以提前用数组记录下来
- 需要暴力走吗？

- 对于每组询问，从 v 点在不经过点权超过 x 苦难值的情况下尽可能往上走，假设 e 是最后走到的点，那么就有，e 为根的子树中，叶子都是可以走到的，且叶子编号的范围 [L,R] 可以提前用数组记录下来
- 需要暴力走吗？
- 如何查询区间第 k 大？

- 对于每组询问，从 v 点在不经过点权超过 x 苦难值的情况下尽可能往上走，假设 e 是最后走到的点，那么就有，e 为根的子树中，叶子都是可以走到的，且叶子编号的范围 [L,R] 可以提前用数组记录下来
- 需要暴力走吗?
- 如何查询区间第 k 大?
- 可以对 kruskal 树中的叶子建可持久化线段树，我们要查询 [L,R] 区间的第 k 大，也就是查询可持久化线段树中第 R 个版本到第 L 个版本的第 k 大的。

- 输入一个 $r*c$ 的循环矩阵（即第一行的上一行是最后一行，最后一行的下一行是第一行，最后一列的下一列是第一列）。你的初始位置在 $(1,1)$，每一步会走向右上、右、右下三个格子中权值最大的一个格子。
- 两种操作：1 修改一个格子的权值，2 从当前位置走 k 步，并输出目标位置
- m 组操作，对询问操作输出答案
- $(3 \leq r, c \leq 2000)$,$(1 \leq m \leq 5000)$,$1 \leq k \leq 10^9$

- 我们可以先暴力跳到第一列，再从第一列倍增 k 能跳多少圈，再从第一列开始暴力跳不够一圈步数，时间复杂度 $log(k) + O(C)$

- 我们可以先暴力跳到第一列，再从第一列倍增 k 能跳多少圈，再从第一列开始暴力跳不够一圈步数，时间复杂度 $log(k) + O(C)$
- 需要同时维护跳向下一个位置的行号和列号吗？

- 我们可以先暴力跳到第一列，再从第一列倍增 k 能跳多少圈，再从第一列开始暴力跳不够一圈步数，时间复杂度 $log(k) + O(C)$
- 需要同时维护跳向下一个位置的行号和列号吗?
- 维护一个 $ne[i][j]$ 表示第 1 列的第 i 行，跳了 2^j 圈后回到第一列的行号
- 我们可以借助 ne 数组来倍增 k 能跳多少圈

- 如何维护某一行跳过一个区间后的行号?

- 如何维护某一行跳过一个区间后的行号?
- 对列号建一棵线段树 $tr[p][i]$ 记录的是线段树节点 p 对应的区间 $[l, r]$, 从 l 的第 i 行跳到 r+1 列的行号

- 如何维护某一行跳过一个区间后的行号?
- 对列号建一棵线段树 $tr[p][i]$ 记录的是线段树节点 p 对应的区间 $[l, r]$, 从 l 的第 i 行跳到 r+1 列的行号
- 如何初始化 $ne[i][0]$?

- 如何维护某一行跳过一个区间后的行号?
- 对列号建一棵线段树 $tr[p][i]$ 记录的是线段树节点 p 对应的区间 $[l, r]$, 从 l 的第 i 行跳到 r+1 列的行号
- 如何初始化 $ne[i][0]$?
- $ne[i][0] = tr[1][i]$; 因为 1 号节点维护的区间是 $[1, C]$

- 如何维护某一行跳过一个区间后的行号?
- 对列号建一棵线段树 $tr[p][i]$ 记录的是线段树节点 p 对应的区间 $[l, r]$, 从 l 的第 i 行跳到 r+1 列的行号
- 如何初始化 $ne[i][0]$?
- $ne[i][0] = tr[1][i]$; 因为 1 号节点维护的区间是 $[1, C]$
- 如何单点修改?

[CERC2017] Donut Drone
————维护 ne 数组

- 如何维护某一行跳过一个区间后的行号?
- 对列号建一棵线段树 $tr[p][i]$ 记录的是线段树节点 p 对应的区间 $[l, r]$, 从 l 的第 i 行跳到 r+1 列的行号
- 如何初始化 $ne[i][0]$?
- $ne[i][0] = tr[1][i]$; 因为 1 号节点维护的区间是 $[1, C]$
- 如何单点修改?
- 修改操作即对应了线段树的单点修改操作：修改当前列第 i 行跳到下一列的行号，也就是 $tr[p][i]$
- 再通过 pushUP 来维护区间修改后的信息：
$tr[p][i] = tr[p << 1|1][tr[p << 1][i]]$

[CERC2017] Donut Drone
————维护 ne 数组

- 如何维护某一行跳过一个区间后的行号?
- 对列号建一棵线段树 $tr[p][i]$ 记录的是线段树节点 p 对应的区间 $[l, r]$, 从 l 的第 i 行跳到 r+1 列的行号
- 如何初始化 $ne[i][0]$?
- $ne[i][0] = tr[1][i]$; 因为 1 号节点维护的区间是 $[1, C]$
- 如何单点修改?
- 修改操作即对应了线段树的单点修改操作：修改当前列第 i 行跳到下一列的行号，也就是 $tr[p][i]$
- 再通过 pushUP 来维护区间修改后的信息：
  $tr[p][i] = tr[p << 1|1][tr[p << 1][i]]$
- p 维护的区间列号是 $[l, r]$,$p << 1$ 的区间是 $[l, mid]$,$p << 1|1$ 的区间是 $[mid + 1, r]$
- $tr[p << 1][i]$ 也就是从第 l 列跳过左儿子的区间 $[l, mid]$ 到达右儿子第一列 $mid + 1$ 的行号
- 然后再把从右儿子第一列跳出后的行号转移给父亲节点

CF1707E Replace
————问题描述

- 给定一个长为 n 的序列 $a_1, \ldots, a_n$，其中对于任意的 i 满足 $1 \le a_i \le n$。定义一个二元组函数如下：

$$f((l,r)) = (\min\{a_l, \ldots, a_r\}, \max\{a_l, \ldots, a_r\})(l \le r)$$

- 你需要回答 q 次询问，每次给定 (l_i, r_i)，问其最少经过多少次 f 的调用（即 $(l,r) \to f((l,r))$）使得 (l_i, r_i) 变成 $(1,n)$，若无解请输出 '-1'。
- $(1 \le n, q \le 10^5)$,$a_i(1 \le a_i \le n)$,$l_i, r_i(1 \le l_i \le r_i \le n)$

CF1707E Replace
————重要性质

- $f(l,r)$ 是区间 $[l,r]$ 调用一次函数的结果，如果存在 $[l_1,r_1]\cap[l_2,r_2]!=\varnothing,[l_1,r_1]\cup[l_2,r_2]=[l,r]$
- 那么就有 $f(l,r)=f(l_1,r_1)\cup f(l_2,r_2)$

- $f(l,r)$ 是区间 $[l,r]$ 调用一次函数的结果，如果存在 $[l_1,r_1]\cap[l_2,r_2]!=\varnothing,[l_1,r_1]\cup[l_2,r_2]=[l,r]$
- 那么就有 $f(l,r)=f(l_1,r_1)\cup f(l_2,r_2)$
- 可以感性理解一下：假如 $[l_1,r_1]\cap[l_2,r_2]!=\varnothing$ 成立，说明 $f([l_1,r_1])$ $f([l_2,r_2])$ 这两个区间至少有一个交点，$[l_1,r_1]\cup[l_2,r_2]=[l,r]$ 成立，说明 $[l_1,r_1]$ 和 $[l_2,r_2]$ 这两个区间能包含 $[l,r]$ 中的所有点，不会遗漏也不会有区间 $[l,r]$ 外的点，那么必然有 $f(l,r)=f(l_1,r_1)\cup f(l_2,r_2)$

- $f(l,r)$ 是区间 $[l,r]$ 调用一次函数的结果，如果存在 $[l_1,r_1]\cap[l_2,r_2]!=\varnothing,[l_1,r_1]\cup[l_2,r_2]=[l,r]$
- 那么就有 $f(l,r)=f(l_1,r_1)\cup f(l_2,r_2)$
- 可以感性理解一下：假如 $[l_1,r_1]\cap[l_2,r_2]!=\varnothing$ 成立，说明 $f([l_1,r_1])$ $f([l_2,r_2])$ 这两个区间至少有一个交点，$[l_1,r_1]\cup[l_2,r_2]=[l,r]$ 成立，说明 $[l_1,r_1]$ 和 $[l_2,r_2]$ 这两个区间能包含 $[l,r]$ 中的所有点，不会遗漏也不会有区间 $[l,r]$ 外的点，那么必然有 $f(l,r)=f(l_1,r_1)\cup f(l_2,r_2)$
- 同理，此性质也可以推广到 $f^k(l,r)$ 是区间 $[l,r]$ 调用 k 次函数的结果，如果存在 $[l_1,r_1]\cap[l_2,r_2]!=\varnothing,[l_1,r_1]\cup[l_2,r_2]=[l,r]$，那么就有 $f^k(l,r)=f^k(l_1,r_1)\cup f^k(l_2,r_2)$

- 前面的性质给了倍增的正确性前提，如何倍增呢?

- 前面的性质给了倍增的正确性前提，如何倍增呢?
- 定义 $L[i][j][k]$ 为区间 $f^i[k, k+(1<<j)-1]$ 的左端点，$R[i][j][k]$ 为区间 $f^i[k, k+(1<<j)-1]$ 的右端点，
- 那么预处理 $L[0][0][i]$ 和 $L[0][1][i]$ 非常的方便，R 同理

- 前面的性质给了倍增的正确性前提，如何倍增呢?
- 定义 $L[i][j][k]$ 为区间 $f^i[k, k+(1<<j)-1]$ 的左端点，$R[i][j][k]$ 为区间 $f^i[k, k+(1<<j)-1]$ 的右端点，
- 那么预处理 $L[0][0][i]$ 和 $L[0][1][i]$ 非常的方便，R 同理
- 然后我们可以枚举倍增次数 i，再枚举区间的长度 j 和区间的起点 k，时间复杂度 $log(n)*log(n)*n$
- 预处理出两个数组之后，就可以倍增枚举跳多少次能到达 [1,n]

- 有一个括号串 S，其中括号串定义为仅由 '(' 和 ')' 组成的字符串。现在可以进行任意次如下操作：
- 将 S 分为三个可以为空的连续子串 A, B, C，接着将 S 变为一个新串 $A(B)C$（即在 B 前加上左括号，在 B 后加上右括号）。
- 给出了经过任意次操作后的括号串 S，你需要还原出原串 S'，使得 S' 的字典序最小。
- 对于两个字符串 a 和 b，若 a 字典序比 b 小，那么满足:
- 1 a 为 b 的前缀，且 $a \neq b$
- 2 令 i 为最小的数使得 $a_i \neq b_i$，那么 a_i 为左括号。
- $1 \leq |S| \leq 3 \times 10^5$

CF1610G AmShZ Wins a Bet

————性质

- 观察发现，假如要删除 S_i, S_j 两个括号，那么区间 $[i, j]$ 内的括号一定是已经删干净了的
- 因为假如区间 $[i, j]$ 内还有括号，那么删除 S_i, S_j 两个括号后，会出现右括号删之后比删之前更早出现，那么字典序更大，所以不能删

- 有了上面的性质，其实我们已经知道了如果一对括号能删，那么他们其实是一一匹配的，

- 有了上面的性质，其实我们已经知道了如果一对括号能删，那么他们其实是一一匹配的，
- 定义 (i, j) 是一对匹配的左右括号，假如从后往前 DP，定义 $dp[i]$ 表示区间 [i,n] 删除一些括号后变成的字典序最小的字符串，那么就有一个非常好想到的转移方程：

- 有了上面的性质，其实我们已经知道了如果一对括号能删，那么他们其实是一一匹配的，
- 定义 (i, j) 是一对匹配的左右括号，假如从后往前 DP，定义 $dp[i]$ 表示区间 [i,n] 删除一些括号后变成的字典序最小的字符串，那么就有一个非常好想到的转移方程：
- $dp[i] = min(S_i + dp[i+1], dp[j+1])$ 时间复杂度 n^2

- 状态转移的瓶颈在于比较 $S_i + dp[i+1]$ $dp[j+1]$ 两个字符串的字典序大小
- 我们需要找到两个串中第一个不同位置的字符，从而比较字典序大小

- 状态转移的瓶颈在于比较 $S_i + dp[i+1]$ $dp[j+1]$ 两个字符串的字典序大小
- 我们需要找到两个串中第一个不同位置的字符，从而比较字典序大小
- 我们可以将后面已经转移过的字符加入 trie 树中，那么每一次转移，最多会在 tire 树的叶子后面添加一个字符
- 我们可以倍增预处理 trie 树上每一个点跳 2^k 次所到的点，同时预处理这一段的字符串的哈希值

- 状态转移的瓶颈在于比较 $S_i + dp[i+1]$ $dp[j+1]$ 两个字符串的字典序大小
- 我们需要找到两个串中第一个不同位置的字符，从而比较字典序大小
- 我们可以将后面已经转移过的字符加入 trie 树中，那么每一次转移，最多会在 tire 树的叶子后面添加一个字符
- 我们可以倍增预处理 trie 树上每一个点跳 2^k 次所到的点，同时预处理这一段的字符串的哈希值
- 那么我们只需要倍增的跳到第一个 hash 值不同的位置即可判断大小，时间复杂度 $log(n)$
- 在添加一个字符后，我们也可以倍增更新，时间复杂度同样是 $log(n)$

- n 个线段，编号 1 到 n。
- 求出这些线段在两两不相交的情况下在数轴上最多能放多少个，
- 同时输出具体的最优方案，且要求方案的字典序最小
- $n \leq 2 \times 10^5$

- 假如我们需要整个区间最优，那么肯定是所有的子区间也要满足最优的，不然可以通过优化某个子区间的线段数，使得整个区间更优

[APIO2009] 会议中心

———思路

- 假如我们需要整个区间最优，那么肯定是所有的子区间也要满足最优的，不然可以通过优化某个子区间的线段数，使得整个区间更优
- 有一种贪心的做法：先求出最优的总数 m，再贪心的按序号从小到大加线段
- 对于当前要加的线段 i，求出 i 在已经添加的线段里的不相交的前驱 *pre* 和后继 *next*
- 那么线段 i 会影响到区间就是 $[pre.R+1, next.L-1]$
- 假如假如线段 i，区间 $[pre.R+1, next.L-1]$ 仍是最优的，那么我们就加入线段 i

- 维护一个数组 $ne[i][k]$ 表示从 i 号线段往右选 2^k 个线段的最小右端点对应线段的编号

[APIO2009] 会议中心

- 维护一个数组 $ne[i][k]$ 表示从 i 号线段往右选 2^k 个线段的最小右端点对应线段的编号
- 如何初始化?

- 维护一个数组 $ne[i][k]$ 表示从 i 号线段往右选 2^k 个线段的最小右端点对应线段的编号
- 如何初始化？
- Right[i] 后面距离 i 最近的右端点，可以通过枚举每个线段更新：$Right[Line[i].l] = Line[i].r$
- 然后从后往前更新 $Right[i] = min(Right[i], Right[i+1])$
- ID[i] 后面距离 i 最近的右端点所对应线段的编号，这个可以在求 Right 数组时同步求出来

[APIO2009] 会议中心

- 维护一个数组 $ne[i][k]$ 表示从 i 号线段往右选 2^k 个线段的最小右端点对应线段的编号
- 如何初始化？
- Right[i] 后面距离 i 最近的右端点，可以通过枚举每个线段更新：$Right[Line[i].l] = Line[i].r$
- 然后从后往前更新 $Right[i] = min(Right[i], Right[i+1])$
- ID[i] 后面距离 i 最近的右端点所对应线段的编号，这个可以在求 Right 数组时同步求出来
- 如何倍增?
- 求出上面两个数组后，就可以完成对 $ne[i][0]$ 的初始化：$ne[lines[i].id][0] = ID[lines[i].r+1]$
- lines[i].r+1 是因为不能包含当前第 i 个线段，所以从 i 的右端点 +1 开始算
- 初始化完之后即可倍增:$ne[u][i] = ne[ne[u][i-1]][i-1]$

```
//从第s个编号的线段(不包括s)到r点之间最多有多少条线段
    int MaxSeg(int s,int r)
    {
        int ans=0;
        for(int i=31;i>=0;i--)
        {
            if(lines[ne[s][i]].r>r) continue;
            s=ne[s][i];
            ans+=(1<<i);
        }
        return ans;
    }
```

[APIO2009] 会议中心
————快速求出区间最多线段数

- 假如我们要求区间 $[L,R]$ 最多有多少个线段，那么就是找到右端点为 $L-1$ 的线段 s，然后调用 $MaxSeg(s,R)$
- 最后贪心的模拟，用 set 维护已经加入的线段集合，按序号从小到达遍历线段，对于当前线段 $Line[i]$ 通过 $MaxSeg$ 函数判断加入 $Line[i]$ 后，会不会更劣，从而判断 $Line[i]$ 能不能加入 set