JSOI 冬令营 2022

图论问题选讲

授课人：薛志坚

目录

01 经典算法回顾

最小生成树：Prim 算法，Kruskal 算法
最短路：Floyd 算法，Dijkstra 算法，Bellman-ford/SPFA 算法
连通性：DFS，BFS，并查集，Tarjan
拓扑排序

最小生成树算法简单回顾
Kruskal 算法：把边按权值从小到大排序，然后一个一个试着往里加边，若这条边的两个端点原来不在同一个连通块内则加边，时间复杂度 O(ElogE)
Prim 算法：从一个点出发维护一个集合，每次选择一条连接集合内某点和集合外某点的最小权值边加入，并把集合外的那个点加入集合，时间复杂度：暴力找最小边 O(V^2)，堆优化 O(ElogV)

最短路算法：
Floyd：dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j])
可以预处理出来多源多汇的最短路
Dijkstra：每次找 dis 最小的点，用来松弛其他点单源最短路，无负权边
SPFA：开始把源点入队，然后每次取队头的点松弛，如果能成功松弛并且不在队列里则入队
可以有负权边，且能判负环。
SPFA 的时间复杂度理论上最大 O(nm)，期望时间复杂度 O(km)，k 为一个比较小的常数。
对于网格图，SPFA 效率比较低

连通性相关的基本算法
DFS,BFS
无向图 **Tarjan** 找 BCC(点双/边双),割点,桥
有向图 **Tarjan** 找 SCC（本质上 Tarjan 用的还是 DFS 的性质)
拓扑排序

02 常见算法的相关应用

例1、 最小生成树计数[JSOI2008]
【题目描述】 给定 n 个点、m 条边的一个简单无向带权图。请你统计这个图中有多少个不同的最小生成树。
（如果两颗最小生成树中至少有一条边不同，则这两个最小生成树就是不同的）。由于不同的 最小生成树可能很多，所以你只需要输出方案数对 31011 的模就可以了。
1<=n<=100; 1<=m<=1000，保证不会出现自环和重边。
注意：具有相同权值的边不会超过 10 条。

【问题分析】 先需要一个结论，对于一个图的不同最小生成树，每种方案所包含的每种权值的边的数量一定一致。
换句话说，把每种方案包含的所有边的边权都写下来，写出来的序列一定都一样。
这样的话，可以先做一遍 kruskal，记下每种边权的使用次数，然后对于每种边权进行 dfs，判断有多 少种合法的组合方式。
一种方案合法意味着：
1.加入每条边时，边的两端点一定属于不同的并查集，也就是仍然要符合 kruskal 的要求。
2.加入的总边数等于开始统计的使用次数。
第一条要求也就决定了不能用组合数进行计算，只能 dfs，而因为相同边权的边不超过 10，所 以 dfs 可行。
然后用乘法原理即可。
注意：
1.dfs 的时候要回溯，所以这里的并查集不能进行路径压缩。
2.处理完每种边之后要把这些边加上去，这才是 kruskal 的过程。
3.考虑图不连通，即不存在最小生成树的情况。

例2、 严格次小生成树[BJWC 2010]
【问题描述】
给定 n 个点、m 条边的一个无向带权图。请你求出该无向图的严格次小生成树。
也就是说： 如果最小生成树选择的边集是 E(M)，严格次小生成树选择的边集是 E(S)，那么需要满足

$$\sum_{e\in E(M)} value(e) < \sum_{e\in E(M)} value(e)$$

$n<=10^5, m<=3*10^5$

【问题分析】
先考虑非严格的次小生成树
先求出最小生成树，然后我们枚举非树边，如果我们加入的非树边是(u,v)，那么一定会在最 小生成树中 u->v 的路径上断掉一条边
我们肯定是断掉最大边，然后把(u,v)加进去
那么什么情况会出现相等呢？
相等的情况是树上路径最大值等于加入的边
我们这时候同时记录次大值，断掉次大边然后连上新边就可以得到严格次小了
找最大值次大值的过程都可以通过倍增维护

例3、 最优比率生成树（poj2728）

【问题描述】 给定 n 个点一个无向图，图中每对顶点间的边 e 有一个收益 C_e 和一个成本 R_e。求该图的一个生成树 T，使树中各边的收益之和除以成本之和最大，即

$$\sum_{e\in T}C(e)/\sum_{e\in T}R(e) \quad (n <= 100)$$

最大

【问题分析】

这个题目是要求一棵最优比率生成树。显然是一个 01 分数规划问题，我们可以用二分答案来解决。

设 x[i]等于 1 或 0，表示边 e[i]是否属于生成树。

则我们所求的比率

$$r=\sum(C[i]*x[i])/\sum(R[i]*x[i]) \quad (0\le i=mid)$$

$$r=\frac{\sum(C[i]*x[i])}{\sum(R[i]*x[I])} \quad (0\le i=mid)$$

如果存在也就是说 mid 比我们求的最大值要小。反之，如果对于任意一组解都有

$$r=\sum(C[i]*x[i])/\sum(R[i]*x[i]) < mid$$

$$r=\frac{\sum(C[i]*x[i])}{\sum(R[i]*x[I])} < mid$$

则说明 mid 比求的最大值要大。

所以我们可以二分 mid 的值，然后建一个新的无向图，设每条边的权值为 C_e-mid*R_e．然后在图中求最大生成树，若最大生成树边权之和非负，则令 l=mid，否则令 r=mid。

当二分结束时，(l+r)/2 即为答案。

例4、 最小树形图（luogu4716）

【问题描述】

给定一个带权有向图和一个 root，求以该 root 为根的一个有向生成树，

且边权和最小

n<=200

n<=5000

众所周知的朱刘算法：

考虑对除了根的每个点选一个入边，我们选最小权值的入边

然后如果出现环，那么我们考虑环外连向环上点的边(u,v,w)，把这些边的边权-=in[v](v 已 经选的入边的权值),然后把环缩点

实现的时候可以用并查集，如果一个点和根没打通就沿最小权入边跳，跳到一个已经访问过

的点则发现有环，对环进行处理并缩点，然后再继续沿着最小边权跳……

这个算法每次修改环上的边权是 O(E)，最多修改 O(V)次复杂度 O(VE)

考虑改进这个算法，我们考虑优化缩环的过程

我们对每个点维护一个堆，那么减权值是堆整体减去某个权值，这个只要打个标记就行了；

然后缩点是堆的合并：标记就是整体偏移量，那么我们算一下对应的偏移量即可完成合并

用优先队列+启发式合并实现是 O(Elog^2V)的，用左偏树实现是 O(ElogV)的

例5、 电话线（poj3662）

【问题描述】

给你 n 个点，它们之间有 p 条边，起点是 1，终点是 n，让你用电话线从 1 连接到 n，其中的 k 条线已经提供给你了，无限长，剩下的得自己铺，代价就是铺的所有电话线中最长的那一根， 求最短的代价。

0 <= k < n <= 1000，1 <= p <=10000

【问题分析】

分析：二分枚举最大边长 mid，如果图中的边大于 mid,则将图中的边当作 1，表示免费使用一次，否则就当作 0，然后求从 1 到 n 的最短路 dist[n]，判断 dist[n]与 k 的大小即可。

可以用分层图最短路。

所谓分层图最短路，就是有一维限制的最短路

比如本题每走一条边免费电缆的数量可能会加 1。

那么问免费电缆数量不超过 k 的情况下，经过的路径上最贵的电缆的花费最小是多少。

那么我们把这维限制加入状态 dp[u][k]，表示到点 u,有 k 条电缆免费时，经过的路径上最贵的 电缆的花费最小是多少。假设现在有一条 u-->v 权值为 w 的边，则有：

当边 u-->v 不免费时,dp[v][k] = min(dp[v][k], max(dp[u][k], w)

当边 u-->v 免费时，dp[v][k+1] = min(dp[v][k+1], dp[u][k])

因为是无向图，所以这样的状态有后效性。

从最短路角度来理解，我们可以把图中原来表示结点编号的一维 u 扩展到二维(u,k)， 然后从(u,k)到(v,k)有长度为 w 的边，从(u,k)到(v,k+1)有长度为 0 的边。然后可以用 SPFA 的方法来求图上的 dp[u][k]表示到点(u,k)路径上最长的边最短是多少。

例6、 Legacy（CF786B）

【题目描述】

有 n 个点，q 个询问，每次询问给出一个操作：

操作 1：1 u v w，从 u 向 v 连一条权值为 w 的有向边

操作 2：2 u l r w，从 u 向区间[l,r]的所有点连一条权值为 w 的有向边

操作 3：3 u l r w，从区间[l,r]的所有点向 u 连一条权值为 w 的有向边

求 q 次询问后，源点 s 到每个点的最短距离。（ 1 <= n,q <= 10^5, 1 <= s <= n)

【问题分析】

线段树优化建图

有一类最短路问题，需要区间[l_1,r_1]向区间[l_2,r_2]连边

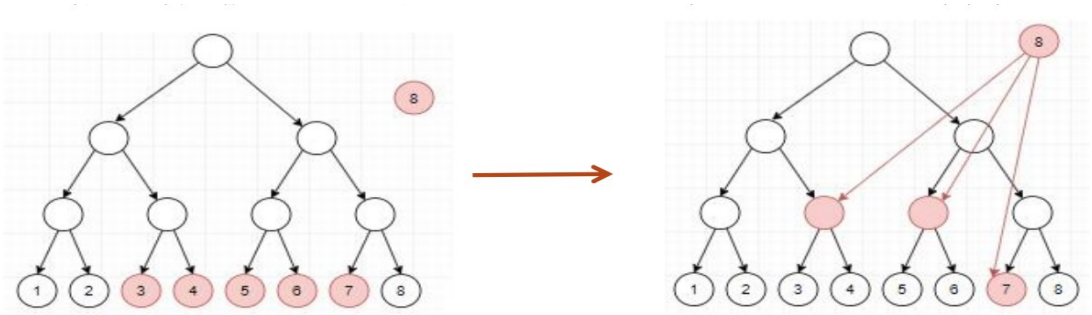
这类问题规模通常无法支持暴力连边，于是有一个优化：线段树加速

考虑两棵线段树表示入和出

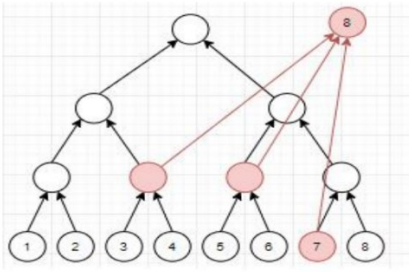
入树从每个点向左右子节点连边，边权为 0 出树从每个点向父节点连边，边权为 0 由于两棵线段树的叶子节点实际上是同一个点，因此要在它们互相之间连边权为 0 的边。

先建一棵线段树。

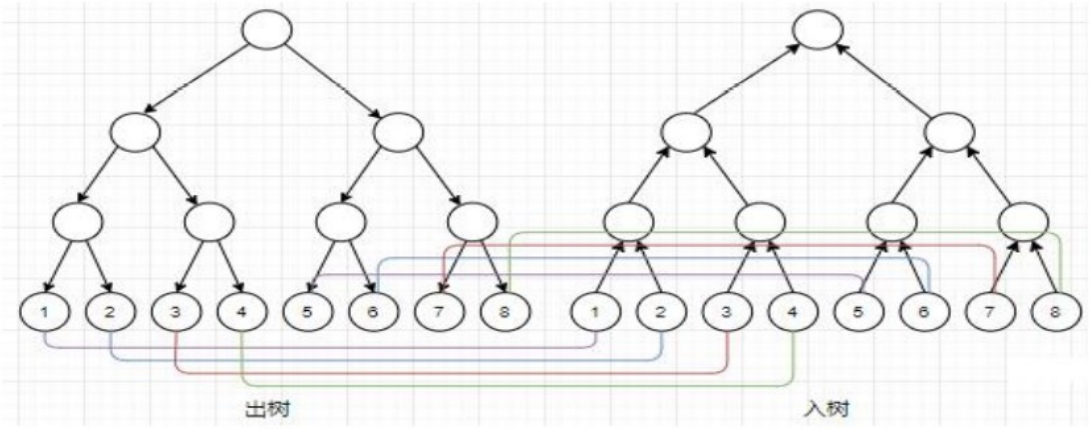
假如现在我们要从 8 号点向区间[3,7]的所有点连一条权值为 w 有向边



操作三用和操作二类似的方法连边。从区间 [3,7]的所有点向 8 号点连一条权值为 w



以上是操作二与操作三分开来考虑的情形，那么操作二与操作三相结合该怎么办呢?
考虑建两棵线段树，第一棵只连自上而下的边，第二棵只连自下而上的边。方便起见，我们把第一棵树称作“出树”，第二棵树称作“入树”。
初始时自上而下或自下而上地在每个节点与它的父亲之间连边。由于两棵线段树的叶子节点实际上是同一个点，因此要在它们互相之间连边权为 0 的边。初始时是这样的



例7、 Warfare And Logistics（LA 4080）
【题目描述】
给定 n 个点、m 条边的带权无向图，先求任意两点间的最短路径累加和，其中不连通的边权为 L 求全局最短路径和（所有点间的最短路径之和）删除任意一条边，求删除后全局最短路径和的最大值。 N<=100,M<=1000
【问题分析】
直接暴力枚举删边跑最短路复杂度爆炸
考虑哪些边删掉是影响答案的：在最短路树上的边
所以我们对每个源点记录哪些边在最短路树上，然后枚举删除的边的时候，只有删掉这些边才需要重新跑最短路 O(M * N^2 logN)

树结构问题选讲

授课人：佟海轩

前言

与树相关的问题在 OI 的各个层次中都经常出现，是比较重要的内容
本节课讲的内容可能比较杂，但（大概）都是 CSP/NOIP 中会见到，能用上的东西，也都比较基本，难度不大
希望各位大佬轻喷 QAQ
一些不讲的基础内容
树的定义和一些等价定义
树的存储方式
先序/中序/后序遍历
……

目录

01 树的直径/重心

树的直径
树的直径：树上最长的简单路径（可能不唯一）
求法一：**两次 DFS**
- 第一次 DFS 找到直径的一个端点（直径的性质）
- 第二次 DFS 求出直径
- 不支持负边权

求法二：**树形 DP**
- DP 求出子树高度
- 在从孩子节点合并 DP 值的时候更新答案
- 边权可为负

直径的性质
以下仅讨论**正边权**情况（可以将边权都视为 1 来考虑）
距离树上任意点最远的点一定是直径的一个端点

若一棵树存在多条直径，那么这些路径的中点（可能在边上）是同一个点，这个点一般称为**树的中心**
若一棵树有直径(u,v)，另一颗树有直径(x,y)，用一条边将这两棵树连起来成为一棵新树，那么新树**一定有一条直径以 u,v,x,y 中两个点为端点**

P1099 [NOIP2007 提高组] 树网的核

题意：给定一棵 n 个点的正边权无根树，在树上求出一段长度不超过 s 的路径，使得离路径距离最远的点到路径的距离最短。输出这个最短距离。
可以证明，可以在一条直径上选取一段路径得到答案。
于是只要先求出树的直径，预处理直径上点不经过直径所能到达的最远距离，然后在直径上滑动窗口扫一遍，每次统计的答案为 路径两端点到直径两端点距离 和 路径上每个点预处理出的距离（统计这部分可以用单调队列优化）取最大值。
复杂度 O(n)

树的重心
树的重心：对于树上的每一个点，计算其所有子树中最大的子树节点数，这个值最小的点就是这棵树的**重心**

求法：DFS 中对于每个点求出以它为根的子树大小，然后用总点数减去以该点为根子树大小得到“向上的”子树大小，再结合以它的儿子为根的子树大小，根据定义判断得到重心

重心的性质
重心若不唯一，则至多两个，且这两个重心相邻
一个点是重心等价于若以它为根，那么每个子树的大小都不超过整个树大小的一半
对于树中所有点到某个点的距离和，到重心的距离和最小。若有两个重心，那么到它们的距离和一样。更进一步，距离和最小与是重心等价
如果一个树增或删一个叶子，则整个树的同一个重心最多移动一个节点
将两棵树通过连一条边合成一棵树，那么新的重心在原来这以这两个树的重心为端点的路径上
证明详见：https://www.cnblogs.com/suxxsfe/p/13543253.html

CodeForces 1406C Link Cut Centroids

题意：给定一棵节点数为 n 的树，删一条边然后加上一条边，使得该树的重心唯一 。（删掉的边和加上的边可以是同一条）
若原树的重心只有一个，那么随便删一条之后加回来就行
若原树的重心有两个，记为 u 和 v，那么这两个重心相邻，且互为对方的最大子树，且作为子树时大小相同，只需要在 v 的子树中拿一个叶子节点接到 u 上，u 就成为了唯一的重心

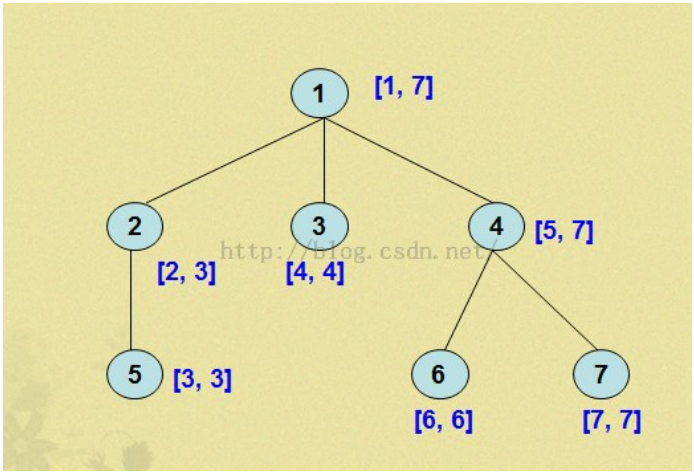
02 DFS 序

DFS 序

树的 DFS 序即为在对树做 DFS 时的节点入栈序列，在 DFS 时对于每个节点记录进入和离开时的**时间戳**即可

DFS 序建立起了树中的子树和序列上的区间之间的对应关系，子树间的包含关系也对应着区间的包含关系

对子树的操作转化为了区间操作，于是可以通过支持区间操作的数据结构（如线段树）维护



DFS 序的应用

直接应用—子树相关：树上单点加/子树加，单点查询/子树求和

差分思想—路径相关：（这些基本可以直接树剖）

路径修改，单点查询

路径修改，子树查询

单点修改，路径查询

子树修改，路径查询

感兴趣可以了解一下：https://www.cnblogs.com/bytebull/p/5929137.html

03 LCA

最近公共祖先简称 LCA（Lowest Common Ancestor）

两个节点的最近公共祖先，即为这两个点的公共祖先里离根最远的

求 LCA

倍增算法：O(nlogn)预处理，在线 O(logn)

$fa[u][i]$ 表示节点 u 的第 2^i 个祖先

一开始通过一遍 DFS 可以得到 $fa[u][0]$ 即节点 u 的父亲，然后通过

$$fa[u][i] = fa\big[fa[u][i-1]\big][i-1]$$

转移在 O(nlogn)复杂度内预处理出 fa 数组

求 lca(u,v)，可以先通过 fa 数组在 O(logn) 的时间内将 u 和 v 跳转至同一深度，若此时相同，则已得到 lca(u,v)

否则 i 从最大开始尝试，依次递减至 0（包括 0），若 fa[u][i]!=fa[v][i]则跳转，这之后 fa[u][0]即为 lca(u,v)

Tarjan 算法：离线算法，n 个点 m 个询问，复杂度 O(n+m)
初始化并查集，每个点为并查集中单独的点；将询问以无向边的形式离线下来
从根开始 DFS ，DFS 到的节点标记为已访问
当 DFS 完 u 的子树后：
先处理询问，对于所有询问 lca(u,v)若 v 已访问过，则 LCA 就是 v 所在并查集的根（即对应联通块中最高节点）
然后在并查集中将 u 合并至它的父亲
可以证明使用路径压缩的并查集在这种情况下复杂度为均摊 O(1)

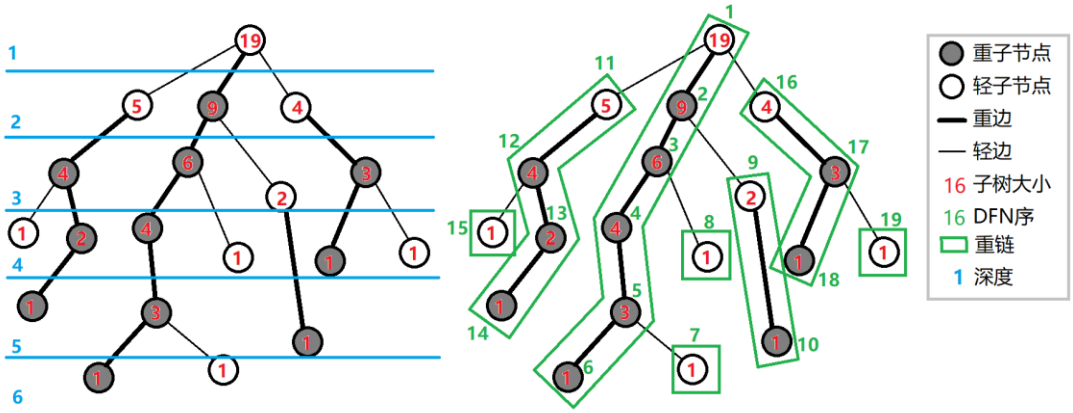
转化为 RMQ：O(nlogn)或 O(n)预处理，在线 O(1)查询
在求 DFS 序的时候，改为无论第一次访问还是回溯访问都将编号记录下来，得到长度为 2n-1 的序列，成为欧拉序列
记 u 在序列中第一次出现的位置为 pos(u)那么 lca(u,v)即为欧拉序列的区间[pos(u),pos(v)]中深度最浅的节点
问题转化为区间最小值询问，可以使用 RMQ 处理
由于序列中相邻两数相差 1 ，于是可以使用 加减 1RMQ 达到 $O(n)$ 预处理
使用树链剖分：O(n)预处理，在线 O(log)
常数很小

04 树链剖分

树链剖分将树分割成若干条链，以类似 DFS 序的形式铺放到线性结构上，从而使用线性数据结构维护子树/路径信息
链的剖分方式有重链剖分（按子树大小）、长链剖分（按子树深度）等等，不加说明时，树链剖分一般指重链剖分，也是这里要介绍的内容

重链剖分
定义：
- **重儿子**：该节点所有儿子中子树最大的儿子
- **重边**：该节点到重儿子的边
- **轻儿子**：该节点剩余的其他所有儿子
- **轻边**：该节点到轻儿子的边
- 若干条首尾衔接的重边构成 **重链**

实现：
- 第一遍 DFS 求出每个点的重儿子和其他基本信息
- 第二遍 DFS 时优先进入重儿子，求出 DFS 序和每条重链的顶部节点



重链剖分的性质

将没有重边相连的单独一个点也看作重链，那么树上每个节点都属于且仅属于一条重链

一条重链上的点对应 DFS 序中一段区间

向下经过一条轻边，子树大小至少除以二

重链剖分的应用

单点/子树维护：同 DFS 序

求 LCA：

若在同一条重链上，则深度较小的点为 LCA

若不在，则不断让重链顶端节点深度大的点向上跳整条重链

路径维护：

路径可以转化为两个点到 LCA 的路径

类似求 LCA 那样向上跳重链，这样路径可以被分为 log 个重链的一部分。每条重链在 DFS 序中是连续的，可以用维护区间的数据结构维护

比如 q 次路径加/查询，用线段树维护，复杂度 $O(q\log^2 n)$。实际中重链个数很难达到 logn 的上界，常数较小

P1967 [NOIP2013 提高组] 货车运输

题意：n 个点 m 条边的无向图，每条边有载重限制。q 条询问，每次询问在不超重的情况下从 u 到 v 最多运送多重货物。无解输出 -1。

边权比较小的边永远不会被用到，于是可以先求出最大生成树来，运送路径在最大生成树上一定最优。

于是问题转化为了求路径最小值，可以使用倍增求 LCA，额外求出一个数组 w[u][i] 表示 u 向上 2^i 长度的路径上的最小边权；也可以直接树剖。

P2486 [SDOI2011]染色

题意：一棵树，每个点有个颜色，需要支持路径染色操作，查询路径上颜色段数量。颜色段定义是极长的连续相同颜色，例如 112221 由三段组成：11、222、1。

考虑在序列上的做法：线段树维护区间颜色段数、左端点颜色、右端点颜色，即可完成区间的合并，从而支持区间染色和查询。

放到树上可以先树剖，然后路径染色/查询转化为 log 个区间染色/查询。

P4211 [LNOI2014]LCA

题意：n 个点的有根树，定义深度为到根距离加一。q 次询问给出 l_i,r_i,z_i，每次询问节点编号在区间$[l_i,r_i]$中所有点与 z_i 的 LCA 的深度之和。可离线。

考虑两点 LCA 的深度可以转化为两点到根节点的路径交的长度（点的个数），于是可以继续转化为将一个点到根路径的点权+1，查询另一个点到根路径的点权和。

对$[l_i,r_i]$的询问可以转化为$[1,r_i]$的询问减去对$[1,l_i-1]$的询问
于是可以将询问离线，按照 r_i 从小到大排序，然后对 r 从 1 到 n 处理，每次将 r 到根路径 +1，然后处理询问。树链剖分维护路径操作

05 树哈希

树哈希即对树结构进行哈希，是判断树同构的一种十分重要的手段
树哈希实际没有什么标准的方式，可以搞出各种奇奇怪怪的哈希方式
但是需要避免一些典型错误，如：
异或导致的问题：比如直接对所有儿子节点的哈希值操作操作之后直接异或起来
多项式导致的问题：比如

$$f[u]=size[u]*\sum f[son[u][i]]*seed^{i-1}$$

$$f_u = size_u \times \sum f_{son(u,i)} \times seed^{i-1}$$

可以构造小数据造成冲突，见 https://oi.wiki/graph/tree-hash/
一种树哈希的方式是：

$$f[u]=1+\sum_{v\in son[u]} f[u]*prime(size[v])$$

$$f_u = 1 + \sum_{v \in son_u} f_v \times prime(size_v)$$

判断同构时同时判断树大小和哈希值是否相同
P4323 [JSOI2016]独特的树叶
题意：n 个点的 A 树添加一个叶节点并打乱节点编号得到 n+1 个点的 B 树，现给出 A 树和 B 树，问 B 树中哪个是新加的叶节点，若有多种可能答案输出最小编号。
考虑对于 A 树，如何快速求出以每个点为根的哈希值
只需要换根 DP 一下即可，先随便选定根，求出每个子树的哈希值，然后再 DFS 一遍，DFS 时维护当前节点向上部分的哈希值
对于 B 树，也用同样的方法求出每个点为根的哈希值，然后枚举叶节点删去，求出相邻点为根的删去单个点的哈希值，然后判断在 A 树中是否出现过

06 结语

练习题（基本都是树剖习题）
P2590 [ZJOI2008]树的统计
P2486 [SDOI2011]染色
P2146 [NOI2015] 软件包管理器
P3313 [SDOI2014]旅行
P2680 [NOIP2015 提高组] 运输计划
P4211 [LNOI2014]LCA
P5024 [NOIP2018 提高组] 保卫王国
P3979 遥远的国度
本节课讲了许多树上算法，但很多实现细节没有展开细讲，需要大家通过做题来掌握
祝大家下午模拟赛顺利~

动态规划问题选讲

授课人：赵伯阳

前言

动态规划问题在 OI 的各个层次中都经常出现，是比较重要的内容。
动态规划问题通常代码难度较低，而思维含量较高，灵活多变，通常需要根据深入分析问题本身的性质。
考虑到大家是提高一班的学生，各种简单的动态规划应该都掌握了。所以本节课主要是挑一些 interesting 的 DP 题来讲解，希望你们能从中了解动态规划的解题思想。
一些需要知道的概念
动态规划的两个要求：最优子结构和无后效性。
最优子结构，指的是大问题可以分解为小问题，且大问题的（最优）解可以由小问题的（最优）解推出。并且问题分解到最后一定是我们可以直接得出答案的极小的子问题。
无后效性，简单来说，就是过去和未来无关。具体来说，如果已经得出了小问题的解，那么如何得到它的解的过程不影响大问题的求解。我们只关心小问题的解是什么，而不需要知道它的解是怎么来的。
动态规划的两个元素：状态和转移。
状态，可以理解为子问题的描述。状态的表示不是唯一的，但是我们通常用一些整数来表示它。例如背包问题，f(i, j) 表示考虑了前 i 个物品，总体积为 j 的最大价值，此处的状态就是由两个整数 i,j 来描述的。状态数，就是我们需要求解问题的所有本质不同的子问题数量。
转移，从一个状态的解推出另一个状态的解的过程。例如背包问题,f(i,j)可以根据 i 这个物品选和不选，从 f(i-1,j),f(i-1, j-vi)转移过来。
对于状态数和转移的优化是 DP 复杂度优化的基本出发点。
Gym102441A Template for Search
热身题
给定由小写字母、字符* 和 ? 组成的字符串 s，其中 * 可以被替换成任意串（包括空串），? 可以被替换成任意字符，求 s 可以表示成的最短回文串。
|S| <= 500。

比较简单的区间 dp。
设 dp[l][r]表示区间[l,r]能组成的最短回文串长度
那么就有下面几种转移:
1. Sl 匹配 Sr 要么 Sl == Sr，要么 Sl 或者 Sr 是问号。dp[l][r] = dp[l+1][r-1] + cnt * 2
2. Sl = *，匹配右边一段。 dp[l][r] = dp[l+1][k] + cnt * 2
3. Sr = *，匹配左边一段。dp[l][r] = dp[k][r-1] + cnt * 2
cnt 表示实际长度，就是因为*可以是空的，所以非*字符算 1 个长度。
输出方案的话，记录一下转移路径就行了。

LOJ6177 送外卖 2

一张 n 个点 m 条有向边的图上，有 q 个配送需求，需求的描述形式为 (s_i , t_i , l_i , r_i)，即需要从点 s_i 送到 t_i， 在时刻 l_i 之后（包括 l_i ）可以在 s_i 领取货物，需要在时刻 r_i 之前（包括 r_i）送达 t_i ，每个任务只需完成一次。

图上的每一条边均有边权，权值代表通过这条边消耗的时间。在时刻 0 有一个工作人员在点 1 上，求他最多能完成多少个配送任务。

在整个过程中，可以认为领货跟交货都是不消耗时间的，时间只花费在路程上。当然在一个点逗留也是允许的。

2 <= n <= 20, 1 <= m <= 400, 1 <= q <= 10

根据实际经验，外卖员可以先领取多个外卖再依次送。

状态压缩，设 0 表示这个外卖没领取，1 表示已领取未送达，2 表示已送达，dp(S) 表示状态 S 需要的最少时间，由于 q 只有 10，所以 S 可以用三进制整数来表示。用类似最短路的方式进行状态转移。

使用最短路的 dijkstra 算法，复杂度约为 O(q^2*3^q)

CF1627E Not Escaping

有一栋 n 层高的楼着火了，每层楼有 m 个房间，第 i 层第 j 个房间用(i, j)表示(1<=i<=n, 1<=j<=m)，你现在在 (1, 1)，你需要跑到 (n, m) 坐直升飞机逃走。

这栋楼有 k 个楼梯，每个楼梯连接(a_i, b_i)和(c_i, d_i)，保证 a_i < c_i，也就是说你只会往高处爬。

同层可以直接来回跑，从(i, j) 跑到(i, k)会消耗|j - k| * x_i 点血（因为火灾）不同的楼层之间必须爬楼梯，爬第 i 个楼梯可以补充 hi 点血。

问最少消耗多少点血才能到(n, m)，答案可以是负数，因为可以一直加血。

n, m, k <= 10^5, x_i, h_i <= 10^6

设 dp(i, j)表示跑到第 i 层第 j 个房间最少耗血。

直接通过枚举楼梯口转移复杂度是 O(k^2)的，但是同层之间乱跑的耗血是一次函数，而且斜率是相同的，所以可以通过正反跑两次来变成 O(k)，也可以维护凸包来变成 O(klogk)，具体通过画图解释。

CF573D Bear and Cavalry

有 n 个人和 n 匹马，第 i 个人对应第 i 匹马。第 i 个人能力值 wi;，第 i 匹马能力值 hi，第 i 个人骑第 j 匹马的总能力值为 wi*hj，整个军队的总能力值为 Σwi*hj (一个人只能骑一匹马，一匹马只能被一个人骑)。有一个要求:每个人都不能骑自己对应的马。让你制定骑马方案，使得整个军队的总能力值最大。现在有 q 个操作，每次给出 a,b，交换 α 和 b 对应的马。每次操作后你都需要输出最大的总能力值。

n, q <= 100000

如果没有那个不能骑自己马的要求？

简单的贪心，大的跟大的匹配比较赚，所以 w 和 h 分别排一下序然后对应即可。

现在加上这个限制，考虑如下的 4 个人，w 和 h 都是排好序的

```
w1 w2 w3 w4
h1 h2 h3 h4
```

如果 w4 可以骑 h4，那 w4 一定会骑 h4（理由同上贪心）

如果 w4 不能骑 h4，那么 w4 会骑谁呢？

如果骑 h1:

```
w1 w2 w3 w4
h? h? h? h1
```

把 h1 和左边某个交换都不会变劣，找一个不是 h4 且能骑 h1 的交换就行。
所以 w4 骑 h1 不可能是最优方案。
由此得到一个结论：排序后一个人骑的马只会在他排名的 3 个以内。
设 dp[i] 表示考虑了前 i 个人的答案，每次 3! 枚举所有骑马情况，从 dp[i-1], dp[i-2], dp[i-3] 转移过来。
这个 dp 式子可以写成矩阵的形式。
考虑修改：每次交换两个数，只会影响到 6 个位置的矩阵，每次重新计算矩阵，用线段树维护矩阵即可。

CF1097G Vladislav and a Great Legend

（原题需要用斯特林数转化，与讲课主题无关，此处给出转化后的题意）
给定一棵 n 个点的无根树，对于每个非空节点集合 X，令 f(X) 表示包含 X 每个节点的最小连通子树的最小边数，即虚树的大小。
求 Sum C(f(X), k)
n <= 100000, k <= 200

树上背包
设 f[0/1][x][i] 表示 x 子树内（不）包含 x 的所有点集构成的生成树边上选 i 条边的方案数之和
访问到 x 的儿子 y 时，如果我选择 x-y 这条边，说明它在生成树上，说明 x 其他的子树（包括 x）选了一些点，和 y 子树中的点勾连，生成树上才能包含 x-y 这条边。

```
f[0][x][i+j+1] += (f[1][x][i] - (i == 0)) * (f[1][y][j] * 2 - (j == 0))
```

如果不选 x 这个点，由于 x-y 在生成树上，所以 x 在生成树上，相当于先选 x 点，算出答案后把 x 去掉，空集不算，所以 C(0, 0)的 1 个方案要去掉。
y 在生成树上，要算上 y 选和 y 不选答案相加，y 选的话就是 f[1][y][j]，y 不选的话就是 f[1][y][j] - (j == 0)

```
f[1][x][i+j+1] += f[1][x][i] * (f[1][y][j] * 2 - (j == 0))
```

选 x 这个点，同上。
如果不选择 x-y 这条边，但是 x-y 在生成树上，那么和上面一样，只需要把 i+j+1 换成 i+j 就行。
如果 x-y 不在生成树上，就是把 y 子树的答案直接累加到 f[0][x] 上

```
f[0][x][i] += f[0][y][i] + f[1][y][i];
```

这样复杂度看似要在每个点枚举 i, j，是 O(nk^2)的。
实际上，将上面两个循环看成枚举原 x 子树内 dfs 序后 k 个点和 y 子树内 dfs 序前 k 个点，那么每个点只会和它 dfs 序前 2k 个点和后 2k 个点被枚举到，所以复杂度是 O(nk)的。

AGC036D Negative Cycle

有一个 n 个点的有向图，节点标号为 0~(N-1)。
这张图初始时只有 N - 1 条边，每条边从 i 指向 i + 1，边权为 0。
对于每一对 i, j(0<=i,j<=N-1,i!=j),Snuke 会加入新边 i->j,如果 i<j 则边权为-1，否则边权为 1。
Ringo 不喜欢图中的负环，所以他想要删掉一些 Snuke 加入的边，使得最终得到的图没有负环。

但是删掉每一条边是有代价的，具体地说，删掉 i→j 这条边，要花费 A(i,j)的代价。请问满足图中不存在负环的最小删边代价是多少？
3 <= n <= 500, 1 <= A(i, j) <= 10^9

无负环，等价于差分约束有解
每个点设变量 xi，一条 i->j 的边权为 w 的边代表 xj - xi <= w.
一开始 i->i+1, 0 代表 xi >= xi+1
设 di = xi-1 - xi >= 0
i->j,-1 的边就表示 d_{i+1} + d_{i+2} + ... + d_j >= 1，区间和为 0 的话这种边就不符合条件，要删掉
i->j,1 的边就表示 d_{j+1} + d_{j+2} + ... + d_i <= 1，区间和大于 1 的话这种边就不符合条件，要删掉
设 dp(i, j) 表示考虑了前 i 个变量，最后两个 1 分别在 i, j 的最小代价。
dp(i, j) 从 dp(j, k)转移过来，那么 j+1...i-1 这段都是 0，所以这段边权为-1 的边都要删掉，这个可以用二维前缀和计算。
从区间[j, i) 指向区间[0, k)的边区间和大于 1，要删掉，也可以用二位前缀和计算。
时间复杂度 O(n^3)

AGC056B Range Argmax
给定整数 n，有 m 个 pair(Li, Ri)，计算序列 x = (x1, x2, ..., xm)的数量，满足下面的条件:
存在 (1, 2, ..., n) 的一个排列 (p1, p2, ..., pn)，满足对于任意的 i， 区间[Li, Ri] 的最大值位置是 xi，即 max(p_{Li}, p_{Li+1}, ..., p_{Ri}) = p_{xi}
答案对 998244353 取模。
n <= 300, m <= n(n-1)/2

看到题，一脸懵，这也能做？
尝试一下用 dp 计算。
设 dp[l][r] 表示在区间[l,r]满足条件的排列个数。
然后枚举最大值的位置 m，插入这个最大值之后，包含这个位置的区间的 xi 就会变成 m，然后删除。
然后分左右两边计算？
注意到这样会出现问题
比如只有两个区间
[1, 2] [3, 4]
排列 1 2 3 4 和排列 3 4 1 2 是一样的，x1 = 2, x2 = 4
多个 p 排列对应同一个 x 序列，为我们计数增添了很多麻烦，所以考虑让一个合法的 x 序列只对应唯一的一个 p 序列。
从大到小插入 n, n - 1, ..., 1 这些数，使符合 xi 是区间最大值位置这个条件。
如果有很多合法位置，就插入在最左边的位置，这样可以通过 x 构造出唯一的 p 序列，x 和 p 形成了一一对应的双射关系。
重新考虑上面的 dp，枚举最大值 Max 的位置 m 后，考虑当前位置 m 作为最左边合法位置的条件。
如果左边还有个合法的最大值位置，那么这个位置也是[l, m - 1]最大值的位置，所以只

要看 Max 能不能放到[l, m - 1]最大值位置 k 上即可。
如果不存在一个[l,r]的子区间完全包含 m, k 的话，我们把 m 和 k 上的数交换，x 序列不变，说明 k 是合法的。
如果存在某个[l,r]的子区间，完全包含 m, k，把 Max 放到 k 这个位置之后，该子区间的 x 值由 m 变成了 k，不合法。
所以说，m 位置是最左边的合法位置，当且仅当不存在某个[l, r]的子区间，完全包含 m, k。
为上面的 dp 增加一维表示最大值的位置，dp[l][r][k]表示区间 l, r，最大值位置在 k 的方案数。
设 Left[m] 表示[l, r]所有子区间中包含 m 的左端点的最小值，
那么 dp[l][r][m]就要从 dp[l][m-1][k], k >= Left[m]转移过来，而右半部分的最大值位置无所谓，所以是 dp[l][r][l...r]
将 dp 数组用后缀和优化一下就可以做到 O(n^3) 的复杂度。

结语

本节课讲了几道 DP 题，但很多实现细节没有展开细讲，建议大家还是把题目自己实现一遍。
虽然上午讲了一堆 DP，但是下午的模拟赛不全是 DP。
祝大家下午模拟赛顺利~

高级数据结构基础

JSOI2022 冬令营 A 班
李争彦
复旦大学
2022 年 1 月 27 日

引子

「NOIP2021」棋局
原题题意非常复杂，正解需要维护数据结构支持:

1. 连通块中棋子删除
2. 对连通块中黑/白子按等级求和
3. 连通块的拆分（离线后转化为连通块合并）

可用多个线段树 + 并查集实现，包括线段树单点修改，区间查询，线段树合并。

NOI 考纲中的数据结构部分

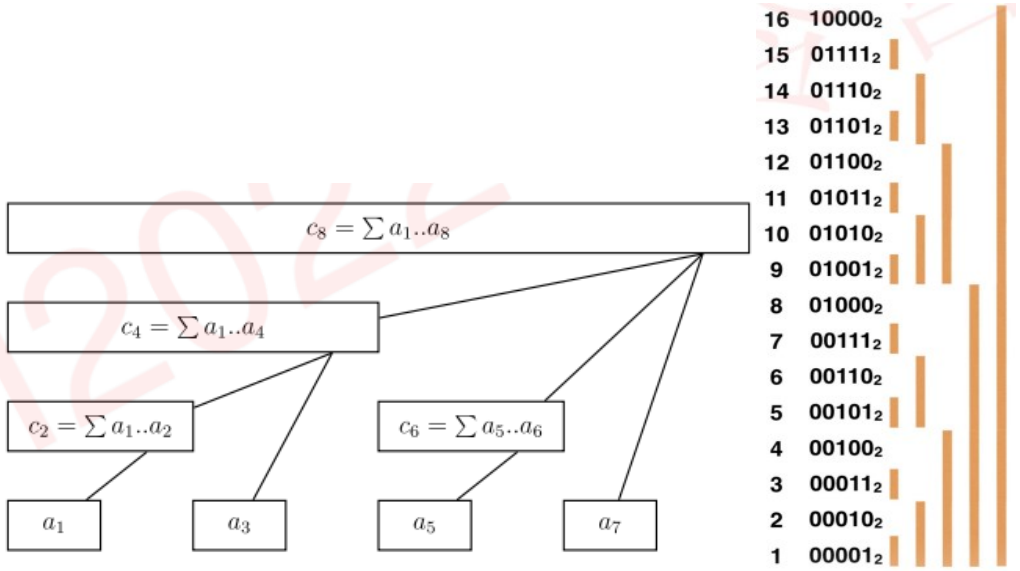
1. 线性结构 【5】双端栈、【5】双端队列、【5】有序队列、【5】优先队列、【6】倍 增表（ST 表）
2. 集合与森林 【6】等价类、【6】并查集、【6】树与二叉树的转化——孩子兄弟表示法
3. 特殊树 【6】线段树和树状数组、【6】字典树（trie 树）、【7】笛卡尔树、【8】二叉平衡树、【8】基环树
4. 常见图 【5】稀疏图、【6】偶图（二分图）、【6】欧拉图、【6】有向无环图、【7】连通图和强连通图、【7】重连通图

目录

树状数组

树状数组被称为 Binary Indexed Tree (BIT) 或 Fenwick Tree，其支持的基本操作有：

- 构造, O(n)
- 前缀和查询, O(log(n))
- 单点更新, O(log(n))



树状数组原理

树状数组本质上是原数组区间和 a[l, r] 和 BIT 数组元素 c[i] 之间的关系。

- c[i] → a[l, r]
  - 观察如图中 c[8], c[12] 位置，分别包含哪些元素的和？

    c[810] = c[010002] = a[000012, 010002] = a[110, 810]

    c[1210] = c[011002] = a[010012, 011002] = a[910, 1210]
  - 推广一下，若设 `lowbit(x)` 为 `x` 二进制中最低的 `1` 的值，

    则 `c[x] = a(x - lowbit(x), x] = a[x - lowbit(x) + 1, x]`， 注意到这个区间内的元素也恰好为 `lowbit(x)` 个。
- `a[l, r]` → `c[i]`
  - 首先对于区间和 `l ≠ 1` 的情况，可将 `a[l, r]` 转化为 `a[1, r] - a[1, l - 1]`。因此只需研究 `a[1, x]` 如何使用 `c[i]` 表示。
  - 观察如图中 `a[1, 6]`, `a[1, 13]`，分别可以由哪些 `c[i]` 的值 组合得到？

    `a[1, 610] = a[1, 001102] = c[001102] + c[001002] = c[610] + c[410]`

    `a[1, 1310] = a[1, 011012]`

    `= c[011012] + c[011002] + c[010002]`

    `= c[1310] + c[1210] + c[810]`
  - 推广一下，设

    $f^1(x) = x - lowbit(x)$, $f^k(x) = f^{k-1}(x) - lowbit(f^{k-1}(x))$, 则

    $a[1, x] = c[x] + c[f^1(x)] + \cdots + c[f^{n-1}(x)]$

    其中 `n` 是 `x` 二进制中 `1` 的数量。

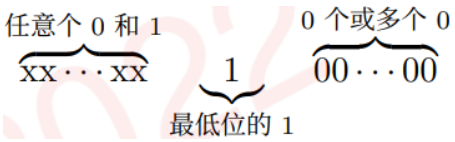
- c[i] → a[l, r]
  c[x] = a(x − lowbit(x), x] = a[x − lowbit(x) + 1, x]
- a[l, r]
  c[i] $f^1(x)$ = x − lowbit(x),
  $f^k(x)$ = $f^{k-1}(x)$ − lowbit($f^{k-1}(x)$),
  a[1, x] = c[x] + c[$f^1(x)$] + · · · + c[$f^{n-1}(x)$]

树状数组利用了一个数字在二进制下能被拆成 log n 个数字的原理。

lowbit 的实现

```
1   int lowbit1(int x) {return x & (x ^ (x - 1));}
2   int lowbit2(int x) {return x & -x;}
```

设有 x≠ 0，其二进制表示必然可表示为



分别考虑两种实现：

- 对于 lowbit1
  - x − 1 = xx···xx011···11
  - x ⊕ (x − 1) = 00···00111···11
  - x ∧ (x ⊕ (x − 1)) = 00···00100···00
- 对于 lowbit2
- 对于补码 −x = ¬(x − 1) = (1-x)(1-x)···(1-x)(1-x)100···00
- x ∧ ¬(x − 1) = 00···00100···00

树状数组基本操作

由树状数组基本原理一节中已经证明：如果对 x 迭代减去 lowbit(x) 得到 f k (x)，并在 c 数组上求和得到 a[1, x] = Pc[f k (x)]。

其对应于树状数组的前缀和查询部分。

为了实现树状数组的基本功能，还需要研究单点更新 a[x] 时，对 c 数组的影响。

设 $g^1(x)$ = x + lowbit(x), $g^k(x)$ = $g^{k-1}(x)$ + lowbit($g^{k-1}(x)$)

定理 1

c[$g^k(x)$] 中必然包含 a[x]，即

x ∈ ($g^k(x)$ − lowbit($g^k(x)$), $g^k(x)$]

定理 2

若 x ∈ (p − lowbit(p), p]，必有 p ∈ {$g^k(x)$}

单点更新区间查询

```
1   const int N = 5e5+5;
2   int c[N];
3   void update(int x, int v) {
4       for(; x < N; x += x & -x) c[x] += v;
5   }
```

```
6   int query(int x) {
7       int ret = 0;
8       for(; x; x -= x & -x) ret += c[x];
9       return ret;
10  }
11  int query(int a, int b) {
12      return query(b) - query(a - 1);
13  }
```

注意：在树状数组更新时，要避免 update(0, 1) 即更新 0 位置值的操作出现。

逆序对 I (Luogu P1908)

题意

给定长度为 $n(n \le 5 \times 10^5)$ 的序列 $a(1 \le a_i < 109)$，求逆序对 (i, j), i < j 且 $a_i > a_j$ 的数量。

思路

树状数组的基本应用。若 ai 的大小在数组可表示的范围内，则从后向前地 ans += query(a[i] - 1) 后更新树状数组，update(a[i], 1) 。显然在逆序对问题中只需要相对关系的大小，因此直接使用离散化减小 a 的大小范围。

逆序对 II

题意

给定长度为 $n(n \le 5 \times 10^5)$ 的序列 $a(|a_i| < 10^9)$。现在可将任意数字变为其相反数，求该序列最小的逆序对数量。

思路

思考修改 a_i 符号对逆序对数量造成的影响：

1. 对于其前后绝对值 $\ge |a_i|$ 的元素，修改 a_i 符号不影响
2. 对于绝对值 $< |a_i|$ 的元素，若在 i 后有 x 个，在 i 前有 y 个。若 a_i 符号为正， 则逆序对增加 x 个；若 a_i 符号为负，则逆序对增加 y 个。

将 a_i 离散后使用树状数组维护 x, y 数量并统计答案，时间复杂度 O(nlogn)。

树状数组处理区间最值问题

在上述的例题中，使用的是树状数组处理具有相加性质运算的能力，如加法、减法以及异或等。那么树状数组能否处理区间最值查询呢？

update

假定处理最大值问题，如果将原先的修改代码中的 + 修改为 max 操作是有问题的。

例子是将原先的区间最大值修改为较小值，按此操作 c 数组仍在记录原先的最大值。

退一步，如果能够将给定的 $c[g^k(x)]$ 包含的区间内的最大值得到再去更新其值，则能够保证正确性。进一步观察，能够发现任何一个 c 的区间能够被拆成 log(n) 个子区间。更新操作单次时间复杂度 $O(\log^2(n))$，代码为：

```
1   void update(int x, int v) {
2       for(; x < N; x += x & -x) {
3           c[x] = v;
4           for (int i = 1, lowbit = x & -x; i < lowbit; i <<= 1)
5               c[x] = max(c[x], c[x - i]);
6       }
7   }
```

Query
显然区间最值是没有区间相加性质的。参照 update 的写法，在 $O(\log^2(n))$ 时间内，从 y 逼近 x：

```
int query(int x, int y) {
    int ans = 0;
    while(y >= x) {
        ans = max(ans, a[y]); y--;
        for(; y - (y & -y) >= x; y -= y & -y)
            ans = max(ans, c[y]);
    }
}
```

树状数组二分优化
对于二分题，使用朴素的实现将在 O(log n) 区间内使用树状数组检查是否可行。
总时间复杂度为 $O(\log^2 n)$。
根据树状数组的性质，是否能够将查询优化到 O(log n)?
以权值树状数组 + 二分求动态第 k 小问题为例，在 N 的空间中，使用树状数组依次从高到低确定二进制的 1 是否可以取到：

```
int kth(int k) {
    int cnt = 0, ret = 0;
    for (int i = lg2[N]; ~i; i--) {
        ret += 1 << i;
        if (ret >= N || cnt + c[ret] >= k)
            ret -= 1 << i;
        else
            cnt += c[ret];
    }
    return ret + 1;
}
```

区间修改[单点/区间]查询
区间修改单点查询
将树状数组维护的 c 数组视为差分数组，对区间 [l, r] 的修改 v 相当于单点修改 a[l] + v 和 a[r + 1] - v。而该数组的前缀和可以看作之前操作对查询位置的影响。
区间修改区间查询
继续利用差分的原理，考虑区间修改区间查询的实现。进行以下变化：

$$\sum_{i=1}^{x} a[i] = \sum_{i=1}^{x}\sum_{j=1}^{i} d[j]$$

$$= \sum_{j=1}^{x} (x - j + 1) d[j]$$

$$= (x+1)\sum_{j=1}^{x} d[j] - \sum_{j=1}^{x} j d[j]$$

使用两个树状数组分别维护 d_i 和 id_i

树状数组区间修改区间查询实现

```
const int N = 5e5+5;
int c1[N], c2[N];
void update(int x, int v) {
    for(int i = x; i < N; i += i & -i) {
        c1[i] += v;
        c2[i] += x * v;
    }
}
void update(int l, int r, int v) {
    update(l, v);
    update(r + 1, -v);
}
int query(int x) {
    int ret = 0;
    for(int i = x; i; i -= i & -i) {
        ret += (x + 1) * c1[i] - c2[i];
    }
    return ret;
}
int query(int l, int r) {
    return query(r) - query(l - 1);
}
```

「SHOI2007」园丁的烦恼

题意

给定 $n(0 \le n \le 5 \times 10^5)$ 个点的坐标 (x, y)，有 m 次询问，查询以 (a, b) 为左下角，(c, d) 为右下角的矩阵内部有多少点。$(0 \le x, y, a, b, c, d \le 10^7)$

思路 1 二维树状数组

问题为无修改的二维树状数组区间查询，对于区间 (a, b) 到 (c, d) 的计数，可转化为 $S(c,d) - S(c,b) - S(a,d) + S(a,b)$。

而此方法需要的树状数组空间为离散化后的 $O(N^2)$。

思路 2 二维偏序

虽然不能在数据结构直接查询 $S(c,d) - S(c,b) - S(a,d) + S(a,b)$ 的结果，但启发能够将查询四个基本查询。进一步将问题转化为二维偏序问题，将点的插入和查询都转化为 $(x, y,task\_type)$。然后将所有点按 x 轴排序，对 y 离散化后，如果是插入点则更新一维树状数组，若求和则查询一维树状数组。时间复杂度为 $O((n + q)\log(n + q))$。

二维区间修改区间查询 (Luogu P4514)

题意

初始为空的 $n \times m$ 矩阵，有两种操作，

`L a b c d delta` 将 $(a, b),(c, d)$ 区域内的数字加上 `delta`；

`k a b c d` 求 $(a, b),(c, d)$ 区域内的数字之和。

$1 \le n, m \le 2048, -500 \le delta \le 500$。总操作次数不超过 5×10^5 次。

二维树状数组

$$\sum_{i=1}^{n}\sum_{j=1}^{m} a[i][j]$$

$$=\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{i}\sum_{l=1}^{j} d[k][l]$$

$$=\sum_{i=1}^{n}\sum_{j=1}^{m} d[i][j]\times(n-i+1)\times(m-i+1)$$

$$=(n+1)(m+1)\sum_{i=1}^{n}\sum_{j=1}^{m} d[i][j]-(m+1)\sum_{i=1}^{n}\sum_{j=1}^{m} d[i][j]\times i-(n+1)\sum_{i=1}^{n}\sum_{j=1}^{m} d[i][j]\times j$$

$$+\sum_{i=1}^{n}\sum_{j=1}^{m} d[i][j]\times i\times j$$

「NOI Online #1 提高组」冒泡排序

题意

给定一个 1 ~ n 的排列 pi，m 次操作 (2 ≤ n, m ≤ 2 × 10^5)：

- 交换第 x 和第 x + 1 个数的位置；
- 询问当前序列经过 k 轮冒泡排序后的逆序对个数。

思路

观察在一轮冒泡排序后，与逆序对数量相关的数值会发生什么变化。

对于逆序对数量为

$$\sum_{i,j}^{i<j}[\,a[i]>a[j]\,]=\sum_{j}\sum_{i}^{i<j}[\,a[i]>a[j]\,]$$

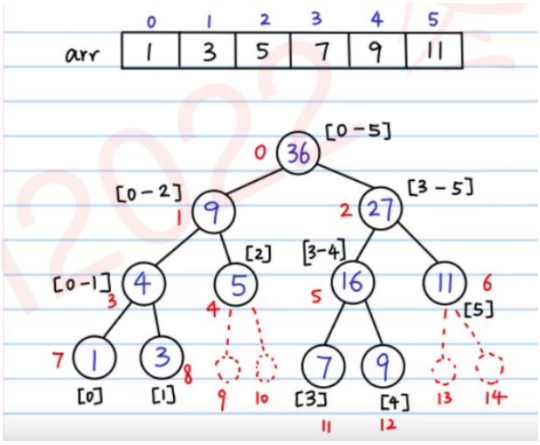
设$s[j]=\sum_{i,j}^{i<j}[\,a[i]>a[j]\,]$，则在一轮冒泡排序后，

对于任意 j 有 s[j] = max(s[j] − 1, 0)。

- 使用树状数组维护 s[j]，并支持对 s[j] 的后缀和查询。
- 发生交换操作时维护逆序对总数、交换的两数 s[j] 变化。

线段树

线段树被称为 Segment Tree，是一种常用于维护区间信息的数据结构。线段树支持单点修改、区间修改、区间查询等操作，支持和值、最值等信息的统计。线段树及其变型应该是信息学竞赛中应用最为广泛的一类数据结构。



线段树与树状数组的比较

线段树的优点

- 线段树能自然地支持更多的运算，基本覆盖树状数组的所有应用；
- 能够在区间并时，维护一些复杂的操作；
- 在同一个线段树中，能够在单个节点同时维护多个信息，有利于复杂场景处理。

线段树的缺点

- 线段树常数较大，虽然均在 O(n log n) 时间复杂度，但运行效率低于树状数组；
- 线段树空间占用较大，虽然均在 O(n) 空间复杂度，但常数至少为四倍；
- 线段树的实现相较于树状数组代码更长。

线段树的性质

线段树本质上是一个 `Full/Strictly Binary Tree`，即每个节点有 `0` 个或 `2` 个孩子节点。

- 一般将根标记为 `1` 号节点，遵从 `Complete Binary Tree` 的标号方法（或称堆式存储），对于节点 `i`，左孩子的下标为 `2i`，右孩子的下标为 `2i + 1`，父亲节点的 下标为 `i/2`；
- 每个节点存储一个区间的信息。每一个非叶节点，表示两个孩子节点的并；每 一个叶子节点，从左到右表示单位长度的区间；
- 一般从上向下构建线段树时，若父亲节点的区间范围为 `[l, r]`，则左孩子表示区间 `[l, (l+r)/2]`，右孩子表示区间 `[(l+r)/2 + 1, r]`。
- 线段树的深度为 `h = [log n]`，因此存储线段树的空间至少为 $2^{h+1} - 1 \le 4n$。

问：在 `n` 等于何值时，线段树数组浪费最大，此时完全二叉树的节点总数为？

线段树的区间查询时间复杂度

线段树在进行区间查询时，往往是从根位置向下不断选入与当前区间有交集的部分。 为何该方式的复杂度为 `O(n log n)`，试分析如下：

假设当前查询为 `[L, R]`，当前考虑的区间为 `[l, r]`，

则有左区间 `[l, (l+r)/2]`和右区间 `[(l+r)/2+ 1, r]`，细分到左右区间各有 `3` 种情况：

1. [l, l+r 2] ⊆ [L, R] 或 [l+r 2 + 1, r] ⊆ [L, R] 包含，此时直接取该区间的统计信息；

2. [l, l+r 2] ∩ [L, R] = ∅ 或 [l+r 2 + 1, r] ∩ [L, R] = ∅ 无交集，此时不再进入子区间；

3. [l, l+r 2] ⊈ [L, R]且[l, l+r 2] ∩ [L, R] ≠ ∅ 或

[l+r 2 + 1, r] ⊈ [L, R]且[l+r 2 + 1, r] ∩ [L, R] ≠ ∅

有交集但不包含，此时需要递归的 向下确定区间边界。

易知对于 [L, R] 在线段树上确定边界时，至多每层有两个 3 类型边界（左右边界各 一个）。而线段树是一棵平衡二叉树，深度为 O(log n) 层。因此区间查询的时间复杂 度为 O(log n)。后面讲到的带懒惰标记的区间更新亦是如此。

单点更新区间查询

```
const int N = 1e5+5;
int data[N << 2];

void build(int rt, int l, int r) {
   if (l == r) {
       scanf("%d", &data[rt]);
       return;
   }
   int mid = (l + r) >> 1;
   build(rt << 1, l, mid);
   build(rt << 1 | 1, mid + 1, r);
}

void update(int rt, int l, int r, int pos, int v) {
   if(l == r) {
       data[rt] += v;
       return;
   }
   int mid = (l + r) >> 1;
   if (pos <= mid) update(rt << 1, l, mid, pos, v);
   else update(rt << 1 | 1, mid + 1, r, pos, v);
}

int query(int rt, int l, int r, int L, int R) {
   if (L <= l && r <= R) return data[rt];
   int mid = (l + r) >> 1, ans = 0;
   if (L <= mid) ans += query(rt << 1, l, mid, L, R);
   if (R > mid) ans += query(rt << 1 | 1, mid + 1, r, L, R);
   return ans;
}
```

注意：在 update 中可以根据题意直接修改统计数组，或进行常规的 pushup 操作。

「TJOI2018」数学计算

题意 初始有一个数 x 值为 1。共有 Q 次操作，共两种类型：

1. 1 m，将 x 变为 x × m，并输出 x mod M;
2. 2 pos，将 x 除以第 pos 次操作所乘的数，并输出 x mod M。
$1 \le Q \le 10^5$, $0 < m, M \le 10^9$
思路
由于次数和乘数较多，且有除法的存在，直接用高精度模拟非常困难;
由于模数不一定是质数，因此除法操作难以转化为乘法逆元。
用另一个角度观察，每次操作为把一个新的数字放在序列的最后，或者将序列中的某个数字变为 1（或将该数字去除）。然后输出序列所有元素的乘积。尝试考虑使用线段树处理序列，1 操作将下一个位置变为 m，2 操作将 pos 变为 1。

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
const int N = 1e5+5;
ll proc[N << 2];
int M;
void update(int rt, int l, int r, int pos, int v) {
    if (l == r) {
        proc[rt] = v;
        return;
    }
    int mid = (l + r) / 2;
    if (pos <= mid) update(rt<<1, l, mid, pos, v);
    else update(rt<<1|1, mid+1, r, pos, v);
    proc[rt] = (proc[rt<<1] * proc[rt<<1|1]) % M;
}
int main(){
    int T;
    scanf(“%d”,&T);
    while(T--) {
        int n, cnt = 0;
        scanf(“%d%lld”, &n, &M);
        fill(proc,proc + (N<<2), 1);
        for(int i = 1; i <= n; i++) {
            int opt; ll m;
            scanf("%d%lld", &opt, &m);
            if (opt == 1) {
                update(1, 1, n, i, m % M);
            } else {
                update(1, 1, n, m, 1);
            }
            printf("%lld\n", proc[1]);
        }
    }
    return 0;
```

}

带懒惰标记线段树

lazy

在线段树的区间查询时间复杂度中讨论了区间查询为何时间复杂度为 O(n log n)。 现在考虑区间更新，显然不能对每个区间内的点进行单点更新。

由之前的讨论得到启发，如果对于包含的区间，能够在 O(1) 的时间进行 “修改”，则时间复杂度是与区间查询一致的。此处将被包含的区间放置修改的标记，待下次进入区间内部时再将修改的影响下传，而此类下传在每次操作中均只会发生 O(log n) 次。

此处的标记被记为 lazy 数组，而该标记往往是一个数值表示修改的具体操作。

区间修改区间查询

```
1   const int N = 1e5+5;
2   int data[N << 2], lazy[N << 2];
3   void build(int rt, int l, int r) {/*...*/}
4   int mark(int rt, int l, int r, int v) {
5       lazy[rt] += v; 6 data[rt] += v * (r - l + 1);
7   }
8   void push_down(int rt, int l, int r) {
9       if(lazy[rt]) {
10          int mid = (l + r) / 2;
11          mark(rt << 1, l, mid, lazy[rt]);
12          mark(rt << 1 | 1, mid + 1, r, lazy[rt]);
13          lazy[rt] = 0;
14      }
15  }
16  void update(int rt, int l, int r, int L, int R, int v) {
17      if(l == r) {
18          mark(rt, l, r, v);
19          return;
20      }
21      int mid = (l + r) >> 1;
22      push_down(rt, l, r);
23      if (mid >= L) update(rt << 1, l, mid, L, R, v);
24      if (mid < R) update(rt << 1 | 1, mid + 1, r, L, R, v);
25      data[rt] = data[rt << 1] + data[rt << 1 | 1];
26  }
27  int query(int rt, int l, int r, int L, int R) {
28      if (L <= l && r <= R) return data[rt];
29      push_down(rt, l, r);
30      int mid = (l + r) >> 1, ans = 0;
31      if (L <= mid) ans += query(rt << 1, l, mid, L, R);
32      if (R > mid) ans += query(rt << 1 | 1, mid + 1, r, L, R);
33      return ans;
34  }
```

彩色弹珠 (Loj P6234)

题意

有一个 n 个正整数组成的序列 a 。选择其中一个区间，得分为该区间内仅出现一次 的数字数量。求得分的最高值。

思路

计算得分最大值，暴力需要计算每个区间，时间复杂度为 $O(n^2)$。 思考当第 i 个数字加入序列后，其对序列的贡献。

若当前第 i 个数字作为区间的右端点 R，设与其数值相同的上一个位置为 pre[i]，设 区间左端点 L，

- 当 L ∈ (pre[i], i] 区间时，cnt[L, R] = cnt[L, R − 1] + 1 (cnt[R, R] = 1)；
- 当 L ∈ (pre[pre[i]], pre[i]] 区间时，cnt[L, R] = cnt[L, R − 1] − 1；
- 当 L ∈ [1, pre[pre[i]]] 区间时，cnt[L, R] = cnt[L, R − 1]

使用线段树维护该数组的最大值，从左到右遍历序列。到达 i 位置时，线段树的根位置值表示 $\max_{j,j<i}\{cnt[j,i]\}$. 对所有位置取最大值即可。

```
#include<bits/stdc++.h>
using namespace std;
const int N = 3e5+5;
int mx[N<<2],tag[N<<2];
int pre[N],pos[N];
inline void mark(int rt, int v){
    mx[rt] += v;
    tag[rt] += v;
}
inline void push_down(int rt){
    if(tag[rt]){
        mark(rt<<1, tag[rt]);
        mark(rt<<1|1, tag[rt]);
        tag[rt] = 0;
    }
}
void add(int rt, int l, int r, int L, int R, int v){
    if(L <= l && r <= R){
        mark(rt, v);
        return;
    }
    int m = (l+r)>>1;
    push_down(rt);
    if(m >= L) add(rt<<1, l, m, L, R, v);
    if(m <= R) add(rt<<1|1, m+1, r, L, R, v);
    mx[rt] = max(mx[rt<<1], mx[rt<<1|1]);
}
```

```
int main(){
    int n, x;
    scanf("%d", &n);
    int ans = 0;
    for(int i = 1; i <= n; i++){
        scanf("%d", &x);
        add(1, 1, n, pos[x]+1, i, 1);
        if(pos[x])
            add(1, 1, n, pre[x]+1, pos[x], -1);
        pre[x] = pos[x];
        pos[x] = i;
        ans = max(ans, mx[1]);
    }
    printf("%d\n", ans);
    return 0;
}
```

多个懒惰标记

当线段树有多种类型的区间修改操作时，会使用到多个懒惰标记，在大部分情况下它们会互相影响，因此需要合理安排多种懒惰标记的计算顺序。

线段树 2 (Luogu P3373)

已知长度为 n 的数列和 p，进行下面三种操作 m 次，$n, m \leqslant 10^5$:

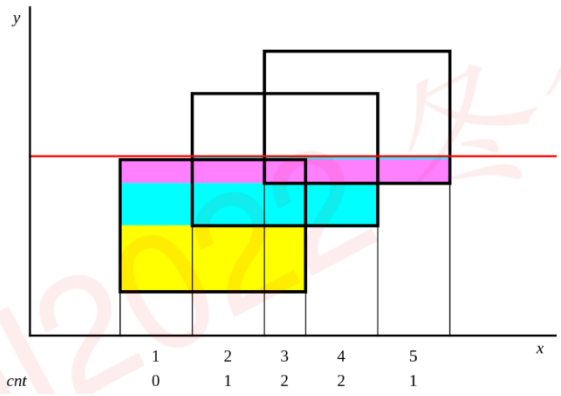
- 将某区间每一个数乘上 x
- 将某区间每一个数加上 x
- 求区间和取模 p 的结果 显然使用乘法优先在修改时更简便，具体而言:

$data[son]=((data[son] * lazy_{mul}[root])+lazy_{add}[root] * (r - l + 1)) \bmod p$

线段树 3 (Luogu P6242)

支持区间加、区间取 max 更新，支持区间和、区间最大值、历史最大值的区间最大值查询。详见吉如一老师的《区间最值操作与历史最值问题》及其讲义。

扫描线算法



原图

扫描线算法用于在二维或多维问题上，使用一条法线扫过图形，统计图形面积、周长等问题。

题意（Luogu P5490）

求 n 个矩形的面积并。($n \le 10^5$, $0 \le x1, y1, x2, y2 \le 10^9$)

思路

以上图为例，将面积并划分为多个小矩形，易知小矩形的数量级别为 O(n)：

- 对于每一个矩形，高度为扫过的高度，将 x 轴离散化后使用线段树统计区间非 零值的区间长度；
- 将矩阵拆分成入边和出边，记 y 轴位置低者为入边，高者为出边，对于线段树 为区间的 +1 和 −1；
- 将所有拆出的边排序，对于每次扫描线移动，累加小矩形的面积。

最大区间子段和（Luogu P4513）

题意

给定长度为 n 的序列 a，有 m 个操作：

1. 求 [L, R] 区间内的最大字段和
2. 将 a[p] 的值改为 s

$1 \le n \le 5 \times 10^5$, $1 \le m \le 10^5$, $|a_i| \le 1000$

思路

使用动态规划能够在 O(n) 时间内求出一段区间的最大字段和。考虑使用线段树，

思考一个区间的最大子段和与子区间统计信息的关系：

1. 左区间最大子段和 max、右区间最大子段和 max
2. 左区间包含右端点的最大子段和 maxr + 右区间包含左端点的最大子段和 maxl

注意：在维护当前区间的 maxl 时，有两种来源：lson.maxl 和 lson.sum + rson.maxl。因此还需要维护区间的和。

环上动态最大子段和 (POJ-2750)
在上一题的基础上，子段和可以跨越区间首尾计算。
思路
环状的最大子段和可分为两个情况计算：

1. 不跨越首尾，与上一题计算方式相同；
2. 跨越首尾，转化为区间和减去区间不跨越首尾的最小子段和。

线段树中额外维护区间最小子段和 min，包含左右端点的最小子段和 minl 和 minr。

动态开点
例题 (HDU-6183)
给定一个矩阵，支持三种类型的操作：】
1. 清空矩阵
2. 1 x y c，将 (x, y)($1 \le x, y \le 10^6$) 加上颜色 c ($0 \neq c \le 50$) (不覆盖之前的颜色)
3. 2 x y_1 y_2，查询左上角 (1, y_1)，右下角为 x, y_2 矩阵中的颜色个数。
清空操作不超过 10 次，1、2 操作不超过 150, 000 次。
思路
按普通线段树思考，为每个颜色建立一棵线段树。
1. 对某一种颜色更新时，对应线段树的 x 位置对 y 更新最小值
2. 查询时，对于每种颜色，查询 [y_1, y_2] 区间的最小值，若 ≤ x 则计入该颜色
动态开点
即使使用离散化，50 棵线段树的空间开销仍无法承受。
动态开点初始时只有根节点，当具体的修改操作时，再创建覆盖的区间与边界。

```
1   void update(int &rt, int l, int r, int pos, int v) {
2       if(!rt) data[rt = ++cnt] = v;
3       data[rt] = min(data[rt], v);
4       if(l == r) return;
5       int mid = (l + r) / 2;
6       if(pos <= mid) update(lson[rt], l, mid, pos, v);
7       else update(rson[rt], mid + 1, r, pos, v);
8   }
```

空间复杂度优化到 O(n log n)

标记永久化
lazy 标记在树套树和持久化数据结构中无法下传。 标记永久化指的是不再寻求下传 lazy 标记，而是在查询时考虑当前标记的影响。

```
void update(int rt, int l, int r, int L, int R, int v) {
    if (L <= l && r <= R) {
        tag[rt] += v;
        sum[rt] += v * (r - l + 1);
        return;
    }
    int mid = (l + r) / 2;
    if (mid >= L) update(rt << 1, l, mid, L, R, v);
```

```
    if (mid < R) update(rt << 1 | 1, mid + 1, r, L, R, v);
    sum[rt] = sum[rt << 1] + sum[rt << 1 | 1] + tag[rt] * (r - l + 1);
}
int query(int rt, int l, int r, int L, int R) {
    if (L <= l && r <= R) return sum[rt];
    int mid = (l + r) / 2;
    int ans = tag[rt] * (min(r, R) - max(l, L) + 1);
    if (mid <= L) ans += query(rt << 1, l, mid, L, R);
    if (mid < R) ans += query(rt << 1 | 1, mid + 1, r, L, R);
    return ans;
}
```

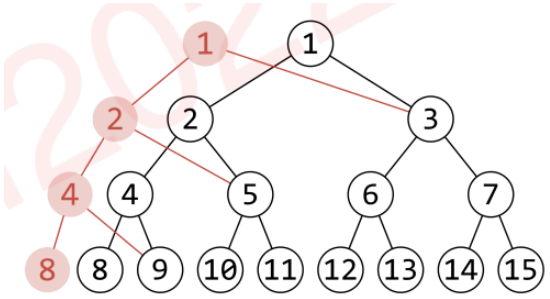
可持久化线段树

在树状数组的部分，介绍了使用权值树状数组解决动态整体第 k 小问题。

接下来尝试解决静态区间第 k 小和动态区间第 k 小问题

题意 (Luogu P3834)

给定 n 个整数构成的序列 a，对于指定的闭区间 [l, r] 查询其区间内的第 k 小值。 尝试在保留旧版本的基础上更新线段树，按之前的分析最多修改 log n 个节点，且 修改的节点必然为一条链并产生新的根。将未修改的子区间仍连到过去版本上：



对于区间查询，使用前缀和确认进入左右区间。注意更新时使用动态开点。

```
int build(int l, int r) {
    int rt = ++tot;
    if (l == r) return;
    int mid = (l + r) / 2;
    lson[rt] = build(l, mid);
    rson[rt] = build(mid + 1, r);
}
int update(int pre, int l, int r, int pos) {
    int rt = ++tot;
    lson[rt] = lson[pre];
    rson[rt] = rson[pre];
    sum[rt] = sum[pre] + 1;
    if (l == r) return; int mid = (l + r) / 2;
    if (pos <= mid) lson[rt] = update(lson[rt], l, mid, pos);
    else rson[rt] = update(rson[rt], mid + 1, r, pos);
    return rt;
}
```

```
int query(int u, int v, int l, int r, int k) {
    int mid = l + r >> 1, cnt = sum[lson[v]] - sum[lson[u]];
    if (l == r) return l;
    if (k <= cnt) return query(lson[u], lson[v], l, mid, k);
    else return query(rson[u], rson[v], mid + 1, r, k - cnt);
}
```

可持久化线段树 + 树状数组

在静态区间第 k 小上进一步考虑动态区间第 k 小，思考如何实现修改操作：

1. 对于修改前该位置的值 w，在 $[1, i](x \le i \le n)$ 的线段树权值 -1；
2. 在 $[1, i](x \le i \le n)$ 的线段树都把权值为 v 的点 $+1$。

修改涉及的线段树为 $O(n)$ 级别无法实现。

考虑使用树状数组的思想，用线段树 $T[i]$ 表示 $[i - lowbit(i) + 1, i]$ 区间的权值信息。 这样对于任何的修改和查询操作均能够在 $O(\log^2 n)$ 时间内完成。总时间复杂度为 $O((n + m)\log^2 n)$。

网络流基础

授课人：单启程
授课日期：2022.1.28

目 录

最大流问题

最大流问题定义：
一个**网络**可以描述为：
给定一个有源汇有向图 G=(V,E)，源点为 S，汇点为 T.
对于每条边形如(u,v,c)，表示一条从 u 到 v 的边，容量为 c.
一个**流**(flow)可以描述为：
对于每条边有一个流量 f(u,v)，

$$0 \le f(u,v) \le c(u,v)$$

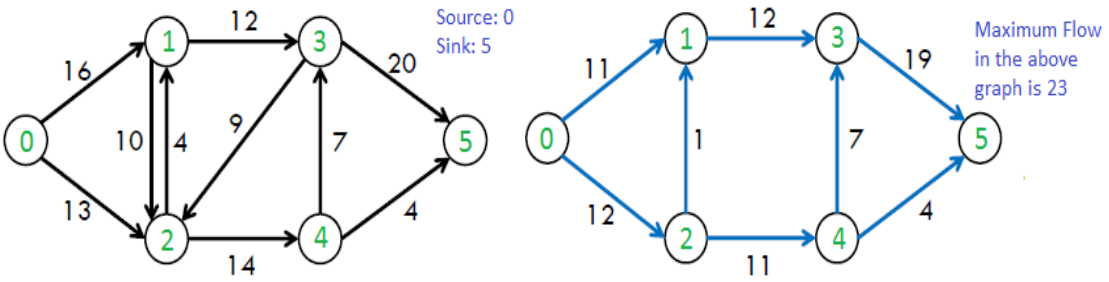
对于每个非源汇结点$u \in V-S-T$，都有

$$\sum_{v \in V} f(v,u) = \sum_{v \in V} f(u,v)$$

流量：

$$V(f) = \sum_{v} f(S,v) = \sum_{v} f(v,T)$$

满足以上条件的就是一个**可行流**，而**所有可行流中流量最大**的就是一个**最大流**
上面说的是数学上的定义，下面来看看直观理解
可以想象成管道运输水流的过程：
每个管道有一个容量，流量不能超过容量
每个非源汇结点不能储存东西，因此流入多少必须
流出多少
最大流就是最大化管道网络的利用效率，使得从 S 到
T 的总流量最大



解决最大流问题有哪些算法呢?

基于增广路的算法：FF,EK,Dinic,ISAP,带 LCT 优化的 Dinic

预流推进：HLPP

……

这次课程重点介绍较为常用的 FF,EK 和 Dinic 算法，掌握这三种算法，足以应对绝大多数的最大流应用场景

介绍 FF,EK 和 Dinic 之前，我们需要介绍几个概念

残量网络和**反向弧**:

对于一条边(u,v,c,f)，假设其已经用了f,那么接下来最多还可以用的量为$c-f$，称之为**残量**

这时候图中需要有一条反向边来表示可以把已经用的流量倒退回去(可以看作**反悔**)，那么就需要一条反向边$(v,u,c,c-f)$来进行**退流操作**

实际上，代码实现的时候可以通过一些小技巧来方便实现：

我们不记录容量，只需要记录残量是多少，初始建图时加入正向边(u,v,c)和反向边$(v,u,0)$*(这里第三个值表示残量是多少)*

反向边可以通过邻接表记录反向边标号或者从 0 开始标号，然后利用 xor 1 来寻找反向边标号

注意：我们下面每对边的残量做一次+x 的修改，那么反向边对应需要-x，反之同理

增广路:

FF,EK 和 Dinic 这三个算法的核心都是寻找增广路

什么是增广路呢?

从 S 到 T，只要能在残量网络上找到一条路径，这条路径上边的最小残量不为 0，那么我们就可以沿着这条路径流一个流量过去，那这条路径就是**增广路**

可以证明，**一个找不到增广路的残量网络即达到最大流**

那么，FF,EK 和 Dinic 便不难理解了

FF 的过程为：

每次 dfs 寻找增广路，找到就沿着这条路增广，时间复杂度$O(nmU)$

EK 的过程为：

每次 bfs 找到一条最短增广路，沿着这条增广路增广，时间复杂度$O(nm^2)$

Dinic 算法：

Dinic 是在 EK 基础上加入了分层图优化和多路增广

Dinic 的过程为:每轮 bfs 建立分层网络，bfs 只能走那些有残量的边，只保留那些距离为 i 的点到距离为 i+1 的点之间的边

然后再 dfs 进行多路增广，沿着分层网络贪心地增广流量(能流多少就流多少)

直到 bfs 建立分层网络时到不了 T，算法结束，时间复杂度$O(mn^2)$
可以证明，在二部图上时间复杂度为$O(m\sqrt{n})$，在边容量均为 1 的图上时间复杂度为

$$O(m * min(n^{\frac{2}{3}}, m^{\frac{1}{2}}))$$

最小割问题:

给定一个有向图$G = (V, E)$，有两个点S和T，每条边有一个边权

现在需要割断一些边，使得 S 和 T 不连通，这称为 **S-T 的一个割(cut)**
边权和最小的割称为**最小割**

最大流-最小割定理:
把有向图中边的边权看作容量建立网络，那么该网络的最大流等于原问题中的最小割
那么最小割问题可以转化为最大流问题，使用 Dinic 算法求解
最小割问题的难点不在于算法，往往在于模型的建立，我们会在最后详细介绍相关建模

费用流问题:

最常见的费用流应该是最小费用最大流，其余大部分费用流模型都可以在这个问题上修改得到
最小费用最大流问题可以描述为:
在最大流问题的基础上，每条边有一个费用 w
现在在满足增广到最大流的条件下，最小化总代价
总代价定义为：**所有边的流量乘以费用之和，**即:

$$\sum_{(u,v)\in E} f(u,v) * w(u,v)$$

解决费用流问题的算法很简单:
考虑修改 EK 算法
每次寻找最短路时，改为边权为费用做最短路
反向边也需要做相应修改：反向边的边权为-w，这样反悔时候增广反向边即可
每次增广就沿着带权图的最短路增广
注意这边的最短路因为存在负权边，因此需要使用支持负权边的最短路算法:

SPFA 或者 Johnson 算法

Johnson 算法可以只需要一次 SPFA(如果初始网络是 DAG 可以 DP)，之后通过顶标重赋边权转为正权图跑 Dijkstra，效率高于每次跑 SPFA，这种方法也被称为**原始对偶**
但是大多数情况下，SPFA 已经足够

网络流建模选讲：

网络流问题的核心主要在于建模上，将实际问题抽象成网络流图，然后通过网络流算法进行解决
下面看一些经典的网络流建模示例

二部图最大匹配：
有 n 个人，m 个工作，每个人可以做某一些工作，每个工作最多一个人来做，问最多给多少人安排工作
连边：(S,人,1)，(人,工作,1),(工作,T,1)，最大流
多重匹配：
有 n 个人，m 个工作，每个人可以做某一些工作，每个工作可以最多由k_i个人做，问最多给多少人安排工作
上面和 T 连边改为(工作，T，k_i)
最小不相交路径覆盖：
给定一个 DAG，求最小的不相交简单路径数量，能覆盖这个 DAG 上所有点
路径数量=出度为 0 的点数
每个点拆成 in 和 out
如果有边(x,y),那么连边(out(x),in(y))
二部图最大匹配
out(x)如果连不出去就是一条路径结束
答案=n-最大匹配

[JSOI 2016] 反质数序列
给定一个长度为 n 的序列，你需要选择一个子序列，使得$\forall i,j,i\neq j,\ x_i+x_j$不为质数。最大化子序列元素个数。
数据范围:$n\leq 3000,a_i\leq 100000$
考虑两个数字和是质数，那么除了 1+1 的情况以外，必须和为奇数
这也意味着两个数必须是一奇一偶
因此之保留一个 1，问题就转化为了二部图最大独立集
二部图最大独立集=顶点数-二部图最大匹配
(考虑每个匹配删掉一个顶点就是方案)

[2021 ICPC 沈阳 D]　Cross the Maze
给定一个 r*c 的网格迷宫，有 n 个人和 n 个出口位置，每个出口在走掉一个人后就会关闭。每个人的每一步可以朝上下左右移动或者不移动，到了出口位置可以选择出去或者不出去
任意两个人不能在同一步待在同一格
问最短多少步，所有人都能出去
数据范围:$n\leq 100,r*c\leq 100$
考虑已知步数 t，怎么去判定可不可行：
设 P(i,x,y)为第 i 个时刻，(x,y)位置所代表的点
(sx,sy)为人初始所在位置,(ex,ey)为出口位置

考虑怎么表示每步每个位置只能有一个人：
拆点，每步拆成 i 和 i’，入边全从 i 入，出边全从 i’出
然后 i 和 i’连容量为 1 的边，限制每步每个位置只能有一个人
那么连边：
(S,P(1,sx,sy),1)
(P(i’,x,y),P(i+1,x,y),1)
(P(i’,x,y),P(i+1,x+1,y),1)
(P(i’,x,y),P(i+1,x-1,y),1)
(P(i’,x,y),P(i+1,x,y+1),1)
(P(i’,x,y),P(i+1,x,y-1),1)
(P(i,x,y),P(i’,x,y),1)
(P(i’,ex,ey),Q(ex,ey),1), (Q(ex,ey),T,1)
这个图能满流就是可行
然后考虑步数只有 O(n)级别，可以二分或者从小到大枚举，每次在已有的网络上加一层继续增广

最大权闭合子图：
n 个物品，每个物品有一个权值，可正可负
有一些限制(a,b)表示选了 a 必须选 b
选出一些物品最大化权值和，并满足所有限制
没有限制的情况：一定选所有正权，舍弃所有负权
限制可以用最小割模型刻画：
对于限制(a,b)，a>0,b<0，有：
(S,a,val[a])
(a,b,inf)
(b,T,-val[b])
这样一个割的方案中，如果割(S,a,val[a])则表示 a 不选，割(b,T,-val[b])表示选 b
那么答案=正权和-最小割
最小割建模的关键点在于将一个割的方式和一种实际方案对应

最大密度子图：
定义一个图的密度=边数/点数
求最大密度子图
二分答案，设答案为 x，
那就需要判定是否存在一个子图满足$|E| - x * |V| \geq 0$
只需要将边看作正权物品，权值为 1，点看作负权物品，权值为$-x$
选一个边必须选其两个端点
最大权闭合子图模型

方格取数：
给定一个 n*m 的网格
每个格子有一个数字
如果选了一个数字那么就不能选择相邻数字
最大化选择的数字和

考虑网格图可以黑白染色
一个冲突一定是不同色格子之间的，冲突可以用最小割描述
那么连边：(S,黑格子,val(黑)),(黑格子,相邻白格子,inf),(白格子,T,val(白))
一个方案要么不选这个黑格子，要么不选这个白格子
答案=总权值和-最小割

Dinic 最大流.cpp

```
int head[maxn],p;
struct edge{int to,next;ll f;}e[maxn*2];
void addedge(int u,int v,int f)
{
    e[p].to=v;e[p].f=f;e[p].next=head[u];head[u]=p++;
    e[p].to=u;e[p].f=0;e[p].next=head[v];head[v]=p++;
}
ll dis[maxn];
bool bfs(int s,int t)
{
    memset(dis,0,sizeof(dis));
    queue<int> q;q.push(s);dis[s]=1;
    while(!q.empty())
    {
        int u=q.front();q.pop();
        for(int i=head[u];i!=-1;i=e[i].next)
        {
            int v=e[i].to;
            ll f=e[i].f;
            if(f&&!dis[v])dis[v]=dis[u]+1,q.push(v);
        }
    }
    return dis[t]!=0;
}
ll dfs(int u,ll maxf,int t)
{
    if(u==t)return maxf;
    ll tmp=0;
    for(int i=head[u];i!=-1&&tmp<maxf;i=e[i].next)
    {
        int v=e[i].to;
        ll f=e[i].f;
        if(f&&dis[v]==dis[u]+1)
```

```
        {
            ll minn=min(maxf-tmp,f);
            f=dfs(v,minn,t);
            tmp+=f;
            e[i].f-=f;e[i^1].f+=f;
        }
    }
    if(!tmp)dis[u]=inf;
    return tmp;
}
ll dinic(int s,int t)
{
    ll ans=0;
    while(bfs(s,t))ans+=dfs(s,inf,t);
    return ans;
}
```

MCMF_多路增广.cpp

```
#include<bits/stdc++.h>
#define inf 1000000000
#define maxn 805
using namespace std;
int T;
int n,m,k;
int a[maxn],b[maxn],w[maxn];
int c[maxn];
int head[maxn],p,s,t;
struct edge
{
    int from,to,next,f,c;
}e[maxn*10];
void addedge(int u,int v,int f,int c)
{

e[p].from=u;e[p].to=v;e[p].f=f;e[p].c=c;e[p].next=head[u];head[u]=p++;
    e[p].from=v;e[p].to=u;e[p].f=0;e[p].c=-
c;e[p].next=head[v];head[v]=p++;
}
int dis[maxn],inq[maxn],pre[maxn];
bool vis[maxn];
bool spfa(int s,int t)
{
    memset(inq,0,sizeof(inq));
```

```
    memset(pre,-1,sizeof(pre));
    queue<int> q;
    for(int i=s;i<=t;i++)dis[i]=inf;dis[s]=0;
    q.push(s);inq[s]=1;
    while(!q.empty())
    {
        int u=q.front();q.pop();inq[u]=0;
        for(int i=head[u];i!=-1;i=e[i].next)
        {
            int v=e[i].to,f=e[i].f,c=e[i].c;
            if(f&&dis[v]>dis[u]+c)
            {
                dis[v]=dis[u]+c;pre[v]=i;
                if(!inq[v])inq[v]=1,q.push(v);
            }
        }
    }
    return dis[t]!=inf;
}
int dfs(int u,int maxf,int t)
{
    if(u==t)return maxf;int tmp=0;
    vis[u]=1;
    for(int i=head[u];i!=-1&&tmp<maxf;i=e[i].next)if(!vis[e[i].to])
    {
        int v=e[i].to,f=e[i].f,c=e[i].c;
        if(f&&dis[v]==dis[u]+c)
        {
            int minn=min(maxf-tmp,f);
            f=dfs(v,minn,t);tmp+=f;e[i].f-=f;e[i^1].f+=f;
        }
    }
    return tmp;
}
int mcmf(int s,int t)
{
    int ans=0;
    while(spfa(s,t))
    {
        memset(vis,0,sizeof(vis));
        ans+=dfs(s,inf,t)*dis[t];
    }
    return ans;
}
```

```
int main()
{
    scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d%d",&n,&k,&m);
        for(int i=1;i<=m;++i)
        {
            scanf("%d%d%d",&a[i],&b[i],&w[i]);
            c[2*i-1]=a[i];c[2*i]=b[i];
        }
        sort(c+1,c+2*m+1);
        n=unique(c+1,c+2*m+1)-c-1;
        p=0;
        memset(head,-1,sizeof(head));
        s=0;t=n+2;
        for(int i=1;i<=m;++i)
        {
            a[i]=lower_bound(c+1,c+n+1,a[i])-c;
            b[i]=lower_bound(c+1,c+n+1,b[i])-c;
        }
        for(int i=1;i<=n;++i)addedge(i,i+1,k,0);
        for(int i=1;i<=m;++i)addedge(a[i],b[i]+1,1,-w[i]);
        addedge(s,1,k,0);
        addedge(n+1,t,k,0);
        int ans=abs(mcmf(s,t));
        printf("%d\n",ans);
    }
    return 0;
}
```

容斥原理与组合计数

开场

普林斯顿数学指南

枚举基本定理与计数

$$|A| = \sum_{a \in A} 1$$

- 解释了加法和乘法的本质
- ∑是求和程序
- 提示我们寻找 A->B 的一一映射，那么|A| = |B|

计数的本质

扩展到两个集合

- 加法原理：|A∪B| = |A| + |B|(当 A∩B = ∅时成立)
- 乘法原理：|A×B| = |A| · |B|

为什么是原理?

因为通用

- ~~凡乎所有的计数问题都在用乘法原理~~
- 不仅是”原理”，而且是加法和乘法的”定义”

```
Inductive nat : Type :=
    | O
    | S (n : nat).
Fixpoint plus (n : nat) (m : nat) : nat :=
    Match n with
    | O => m
    | S n’ => S (plus n’ m)
    end.
```

二进制串计数(0)

n-bit 二进制串一共有多少个?

- 乘法原理-> 2^n个
- 另一种理解方法：
  - 我们可以把集合分成两个部分，“0”开头的和“1”开头的
  - 剩下的部分存在一个一一对应
  - f[n] = 2*f[n-1], f[0] = 1

二进制串计数(1)

n-bit 二进制串中“0”比“1”多的一共有多少个?

“加法原理”

$$(n, n) + (n, n-1)+\ldots\ldots+(n, \lfloor n/2 \rfloor + 1)$$

二进制串计数(2)

如果把 0 看作“(”，1 看作“)“，配对的括号序列有多少个?

- (()) – 0011;)()()(- 101010
- (……k……)……n-k……

$$f[n] = \sum_k f[k-2] \cdot f[n-k]$$

排列计数(0)

求 1，2,…,n 所有排列的数量

- 是 n!
- 其实这个构造不算太显然
  - 例子：可以把这 n 个数分成 k 和 n-k 两组

排列计数(1)

错位排列：求所有 1,2,…,n 的排列中，每个数字都不在原位的排列的数量

还记得“集合”吗?

- 把集合写出来呀!

```
from itertools import permutations
n=4
for p in permutations(list(range(n)),n):
    if True not in [p[i] == i for i in range(n)]:
        print([x+1 for x in p])
```

错位排列递推公式：

$$f[n] = (n-1)(f[n-1] + f[n-2])$$

容斥原理(PIE)

集合视角的计数

加法原理：

$$|A \cup B| = |A| + |B|$$

成立条件：$A \cap B = \varnothing$

“互相存在关联“的|A∪B|能不能求解?

- 1,2,…,100 中能被 2 或 3 整除的

  A={2, 4, 6, 8, 10…}

  B={3, 6, 9, 12, 15…}

  $|A| + |B| \neq |A \cup B|$

求交容易，求并难?

$$|A \cup B| = |A| + |B| - |A \cap B|$$

- 许多问题都有“交比并简单“的性质
  - 凸多边形的交 v.s.并

能不能“用交求并“?
这样很多麻烦的计数问题就可以求解了!

$$(x_0\,x_1\,x_2\,x_3\,x_4\,x_5\,x_6\,x_7)\cdot\begin{pmatrix}|\varnothing|=0\\|A|\\|B|\\|C|\\|A\cap B|\\|A\cap C|\\|B\cap C|\\|A\cap B\cap C|\end{pmatrix}=|A\cup B\cup C|$$

这个“方程可以求解吗?
考虑“单个元素“的集合 e

$$e\in A,\ e\notin B,\ e\notin C \Rightarrow x_1 = 1$$
$$e\in A,\ e\notin B,\ e\notin C \Rightarrow x_1 + x_3 + x_5 = 1$$
$$e\in A,\ e\in B,\ e\in C \Rightarrow x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = 1$$

$$x1 = 1$$
$$x2 = 1$$
$$x3 = 1$$
$$x1 + x2 + x4 = 1$$
$$x1 + x3 + x5 = 1$$
$$x2 + x3 + x6 = 1$$
$$x1 + x2 + x3 + x4 + x5 + x6 + x7 = 1$$

$$|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$$

容斥原理(Principle of Inclusion and Exclusion)
若用若干个集合“覆盖“一个不规则的集合，即可化并为交!

$$\left|\bigcup_{i=1}^{n} Ai\right| = \sum_{\emptyset\neq I\subseteq[n]} (-1)^{|I|+1}\left|\bigcap_{i\in I} A_i\right|$$

容(Inclusion)
$(-1)^{|I|+1} = 1 \Rightarrow |I| \in 1, 3, 5, 7, \ldots$
斥(Exclusion)
$(-1)^{|I|+1} = -1 \Rightarrow |I| \in 2, 4, 6, 8, \ldots$

什么问题求交容易，求并难?
错位排列(容斥原理)
求所有 1，2,…,n 排列中，每个数字都不在原位的排列数量
例子：n=4

- A,B,C,D 表示 1,2,3,4 恰好在第 1,2,3,4 个位置的排列
- n!-|A∪B∪C∪D|就是答案
  - |A| = (n-1)!
  - |A∩C∩D| = (n-3)!

容斥原理：为什么是“原理“？

“集合“和”并“是非常具有一般性的结构

- 例子：$\phi(n)$是 1,2,…,n 中与 n 互质的数量
  - 如何求 $\phi(n)$?
  - ϕ 和 μ 的关系
    - $\mu(n) = (-1)^k$ （n 是 k 个素数的乘积）
    - $\mu(n) = 0$ （如果 p^2 整除 n）
    - 例子：$n = 5 \times 7 \times 13$

$$\varphi(n) = \sum_{d|n} \mu(d) \cdot \frac{n}{d}$$

$$g(A) = \sum_{S \subseteq A} f(S) \Rightarrow \sum_{S \subseteq A} \mu(A - S) g(s)$$

容斥原理：

偏序集上 Mobius 反演在 Boolean Lattic 上的应用(Gian-Carlo Rota,1964)

例子：

XVIII Open Cup named after E.V.Pankratiev.Grand Prix of Gomel,Problem K

给定正整数 $m<=10^{18}$,统计非空集合 S 的数量，其中 S 满足：

$$\gcd(S) = 1 \ \wedge \ \mathrm{lcm}(S) = m$$

- 首先，S 中的数必然是 m 的约数
- 然后，我们应该找什么样“集合的并“

总结

计数与容斥原理

枚举(计数)的是**集合**

$$|A| = \sum_{a \in A} 1$$

简化问题的钥匙：**化并为交**

$$\left| \bigcup_{i=1}^{n} Ai \right| = \sum_{\emptyset \neq I \subseteq [n]} (-1)^{|I|+1} \left| \bigcap_{i \in I} A_i \right|$$