



WIDA's XCPC

Algorithm Template (I)

v1.8.8 2024.11.19
WIDA

目录

基础算法	
头文件	1
常用函数	1
最大公约数 gcd	2
整数域二分	2
旧版（无法处理负数情况）	2
树上问题	
树的直径	5
树论大封装（直径+重心+中心）	5
点分治 / 树的重心	7
最近公共祖先 LCA	8
图论	
常见概念	13
单源最短路径（SSSP 问题）	13
（正权稀疏图）动态数组存图+Djikstra	13
（负权图、判负环）Bellman-ford 算法	14
（负权图）SPFA 算法	14
多源汇最短路（APSP 问题）	15
平面图最短路（对偶图）	16
最小生成树（MST 问题）	17
（稀疏图）Prim 算法	17
（稠密图）Kruskal 算法	17
缩点（Tarjan 算法）	18
（有向图）强连通分量缩点	18
（无向图）割边缩点	19
（无向图）割点缩点	20
染色法判定二分图（dfs 算法）	22
链式前向星建图与搜索	22
一般图最大匹配（带花树算法）	23
一般图最大权匹配（带权带花树算法）	25
二分图最大匹配	29
匈牙利算法（KM 算法）解	29
HopcroftKarp 算法（基于最大流）解	30
二分图最大权匹配（二分图完美匹配）	32
二分图最大独立点集（Konig 定理）	33
最长路（topsort+DP 算法）	34
最短路径树（SPT 问题）	35
网络流	
最大流	52
最小割	53
数论	
常见数列	57
调和级数	57
素数密度与分布	57
因数最多数字与其因数数量	57
快速幂	57
新版	2
实数域二分	3
整数域三分	3
实数域三分	3
树链剖分解法	8
树上倍增解法	9
树上路径交	11
树上启发式合并 (DSU on tree)	11
无源汇点的最小割问题 Stoer - Wagner	36
欧拉路径/欧拉回路 Hierholzers	37
有向图欧拉路径存在判定	37
无向图欧拉路径存在判定	37
有向图欧拉路径求解（字典序最小）	38
无向图欧拉路径求解	38
差分约束	39
2-Sat	39
基础封装	39
答案不唯一时不输出	40
常见结论	41
常见例题	41
杂	41
Prüfer 序列：凯莱公式	43
Prüfer 序列	44
单源最短/次短路计数	44
输出任意一个三元环	45
带权最小环大小与计数	45
最小环大小	46
本质不同简单环计数	47
输出任意一个非二元简单环	47
有向图环计数	48
有向图简单环检查、输出	49
无向图简单环检查、输出	50
判定带环图是否是平面图	50
最小割树 Gomory-Hu Tree	53
费用流	55
常规	57
长整型 with 防爆乘法	58
质数判定	58
试除法	58
筛法	58

Miller-Rabin.....	59	康拓展开	70
质因子分解.....	59	正向展开普通解法	70
筛法.....	59	正向展开树状数组解	70
Pollard-Rho.....	60	逆向还原	71
裴蜀定理.....	60	Min25 筛	71
逆元.....	61	矩阵四则运算	72
费马小定理解（借助快速幂）	61	矩阵快速幂	74
扩展欧几里得解.....	61	矩阵加速	74
离线求解：线性递推解.....	61	莫比乌斯函数/反演	75
扩展欧几里得 exgcd.....	61	整除（数论）分块	76
离散对数 bsgs 与 exbsgs.....	62	常见结论	76
欧拉函数.....	63	球盒模型	76
直接求解单个数的欧拉函数.....	63	麦乐鸡定理	79
求解 1 到 N 所有数的欧拉函数.....	64	抽屉原理（鸽巢原理）	79
使用莫比乌斯反演求解欧拉函数.....	64	哥德巴赫猜想	79
组合数.....	64	除法、取模运算的本质	79
debug.....	64	与、或、异或	79
逆元+卢卡斯定理（质数取模）	65	调和级数近似公式	79
质因数分解.....	65	欧拉函数常见性质	79
杨辉三角（精确计算）	66	组合数学常见性质	80
求解连续数字的正约数集合——倍数法.....	66	范德蒙德卷积公式	80
容斥原理.....	67	卡特兰数	80
二进制枚举解.....	67	狄利克雷卷积	81
dfs 解.....	67	斐波那契数列	81
同余方程组、拓展中国剩余定理 excrt	68	杂	81
求解连续按位异或.....	68	常见例题	82
高斯消元求解线性方程组.....	69		

几何

库实数类实现（双精度）	85	两点是否在直线同侧/异侧	89
平面几何必要初始化	85	两直线相交交点	89
字符串读入浮点数	85	两直线是否平行/垂直/相同	89
预置函数	85	点到直线的最近距离与最近点	90
点线封装	86	点是否在线段上	90
叉乘	86	点到线段的最近距离与最近点	90
点乘	87	点在直线上的投影点（垂足）	90
欧几里得距离公式	87	线段的中垂线	90
曼哈顿距离公式	87	两线段是否相交及交点	91
将向量转换为单位向量	87	平面圆相关（浮点数处理）	92
向量旋转	87	点到圆的最近点	92
平面角度与弧度	87	根据圆心角获取圆上某点	92
弧度角度相互转换	87	直线是否与圆相交及交点	92
正弦定理	88	线段是否与圆相交及交点	92
余弦定理（已知三角形三边，求角）	88	两圆是否相交及交点	93
求两向量的夹角	88	两圆相交面积	93
向量旋转任意角度	88	三点确定一圆	94
点绕点旋转任意角度	88	求解点到圆的切线数量与切点	94
平面点线相关	89	求解两圆的内公、外公切线数量与切点	94
点是否在直线上（三点是否共线）	89	平面三角形相关（浮点数处理）	95
点是否在向量（直线）左侧	89	三角形面积	95

三角形外心.....	95	四点是否共面	105
三角形内心.....	95	空间点是否在线段上	106
三角形垂心.....	96	空间两点是否在线段同侧	106
平面直线方程转换.....	96	两点是否在平面同侧	106
浮点数计算直线的斜率.....	96	空间两直线是否平行/垂直	106
分数精确计算直线的斜率.....	96	两平面是否平行/垂直	106
两点式转一般式.....	96	空间两直线是否是同一条	107
一般式转两点式.....	97	两平面是否是同一个	107
抛物线与 x 轴是否相交及交点.....	97	直线是否与平面平行	107
平面多边形.....	98	空间两线段是否相交	107
两向量构成的平面四边形有向面积.....	98	空间两直线是否相交及交点	107
判断四个点能否组成矩形/正方形.....	98	直线与平面是否相交及交点	108
点是否在任意多边形内.....	98	两平面是否相交及交线	108
线段是否在任意多边形内部.....	99	点到直线的最近点与最近距离	108
任意多边形的面积.....	100	点到平面的最近点与最近距离	108
皮克定理.....	100	空间两直线的最近距离与最近点对	109
任意多边形上/内的网格点个数（仅能处理整数）.....	100	三维角度与弧度	109
二维凸包.....	100	空间两直线夹角的 cos 值	109
获取二维静态凸包（Andrew 算法）	100	空间两平面夹角的 cos 值	109
二维动态凸包.....	101	直线与平面夹角的 sin 值	109
点与凸包的位置关系.....	102	空间多边形	109
闵可夫斯基和.....	103	正 N 棱锥体积公式	109
半平面交.....	103	四面体体积	110
三维几何必要初始化.....	104	点是否在空间三角形上	110
点线面封装.....	104	线段是否与空间三角形相交及交点	110
其他函数.....	105	空间三角形是否相交	110
三维点线面相关.....	105	常用结论	111
空间三点是否共线.....	105	平面几何结论归档	111
常用例题.....		立体几何结论归档	112
将平面某点旋转任意角度			123
平面最近点对（set 解）			123
平面若干点能构成的最大四边形的面积（简单版，暴力枚举）			124
平面若干点能构成的最大四边形的面积（困难版，分类讨论+旋转卡壳）			125
线段将多边形切割为几个部分			125
平面若干点能否构成凸包（暴力枚举）			127
凸包上的点能构成的最大三角形（暴力枚举）			127
凸包上的点能构成的最大四角形的面积（旋转卡壳）			129
判断一个凸包是否完全在另一个凸包内			129
多项式			
多项式封装.....	118	拉格朗日插值	126
离散傅里叶变换 dft 与其逆变换 idft	121	常用结论	127
Berlekamp-Massey 算法（杜教筛）	122	杂	127
Linear-Recurrence 算法.....	123	普通生成函数 / OGF	127
快速傅里叶变换 FFT.....	124	指数生成函数 / EGF	127
快速数论变换 NTT.....	125		
数据结构			
并查集（全功能）	129	半动态区间和（单点修改）	129
树状数组.....	129	动态区间和（区间修改）	130

静态区间最值（单点修改）	130	轻重链剖分/树链剖分	141
逆序对扩展	131	小波矩阵树：高效静态区间第 K 大查询	144
前驱后继扩展（常规+第 k 小值查询+元素排名查询+元素前驱后继查询）	131	普通莫队	145
二维树状数组	132	带修改的莫队（带时间维度的莫队）	146
半动态矩阵和（单点修改）	132	分数运算类	147
动态矩阵和（区间修改）	133	主席树（可持久化线段树）	148
线段树	133	KD Tree	149
区间加法修改、区间最小值查询	133	ST 表	151
同时需要处理区间加法与乘法修改	135	基于状压的线性 RMQ 算法	152
区间赋值/推平	137	pbds 扩展库实现平衡二叉树	153
区间取模	137	vector 模拟实现平衡二叉树	154
区间异或修改	137	线性基（高斯消元法）	154
拆位运算	139	珂朵莉树（OD Tree）	155
坐标压缩与离散化	140	取模运算类	156
简单版本	140	大整数类（高精度计算）	158
封装	140	常见结论	163
动态规划		常见例题	164
01 背包	166	有依赖的背包	169
完全背包	166	背包问题求方案数	170
多重背包	167	背包问题求具体方案	170
混合背包	168	数位 DP	171
二维费用的背包	168	状压 DP	172
分组背包	169	常用例题	172
串			
子串与子序列	175	字典树 trie	179
字符串模式匹配算法 KMP	175	基础封装	179
Z 函数（扩展 KMP）	175	01 字典树	179
最长公共子序列 LCS	176	后缀数组 SA	180
小数据解	176	AC 自动机	181
大数据解	176	回文自动机 PAM（回文树）	182
字符串哈希	177	后缀自动机 SAM	182
双哈希封装	177	子序列自动机	183
前后缀去重	178	自动离散化、自动类型匹配封装	183
马拉车	178	朴素封装	184
博弈论			
巴什博奕	185	Anti-SG 游戏（反 SG 游戏）	187
扩展巴什博奕	185	Lasker's Nim 游戏（Multi-SG 游戏）	187
Nim 博奕	185	Every-SG 游戏	188
Nim 游戏具体取法	186	威佐夫博奕	188
Moore's Nim 游戏（Nim-K 游戏）	186	斐波那契博奕	188
Anti-Nim 游戏（反 Nim 游戏）	186	树上删边游戏	189
阶梯 - Nim 博奕	186	无向图删边游戏（Fusion Principle 定理）	189
SG 游戏（有向图游戏）	186		
STL			
库函数	190	二分搜索 binary_search	190
pb_ds 库	190	批量递增赋值函数 iota	190
查找后继 lower_bound、upper_bound	190	数组去重函数 unique	190
数组打乱 shuffle	190	bit 库与位运算函数 __builtin_	191

数字转字符串函数.....	191	队列 queue	194
字符串转数字.....	191	双向队列 deque	194
全排列算法 next_permutation 、 prev_permutation.....	192	优先队列 priority_queue	194
字符串转换为数值函数 sto.....	192	字符串 string	195
数值转换为字符串函数 to_string.....	193	有序、多重有序集合 set、multiset	195
判断非递减 is_sorted.....	193	map、multimap	196
累加 accumulate.....	193	bitset	196
迭代器 iterator.....	193	哈希系列 unordered	197
其他函数.....	193	对 pair、tuple 定义哈希	197
容器与成员函数.....	193	对结构体定义哈希	198
元组 tuple.....	193	对 vector 定义哈希	198
数组 array.....	193	程序标准化	198
变长数组 vector.....	194	使用 Lambda 函数	198
栈 stack.....	194	使用构造函数	199
卡常			
基础算法 最大公约数 gcd 位运算加速	200	数论 取模运算类 蒙哥马利模乘	202
数论 质数判定 预分类讨论加速.....	200	网络流 最大流 预流推进 HLPP	203
数论 质数判定 Miller-Rabin.....	200	快读	204
数论 质因数分解 Pollard-Rho.....	201		
杂类			
统计区间不同数字的数量（离线查询）	206	魔改十进制快速幂（暴力计算）	219
选数（DFS 解）	206	扩展欧拉定理（欧拉降幂公式）	219
选数（位运算状压）	207	int128 输入输出流控制	220
网格路径计数.....	207	对拍版子	221
德州扑克.....	207	随机数生成与样例构造	221
N*M 数独字典序最小方案.....	211	手工哈希	222
高精度进制转换.....	212	Python 常用语法	222
物品装箱.....	212	读入与定义	222
从前往后装（线段树解）	212	格式化输出	222
选择最优的箱子装（multiset 解）	213	排序	222
浮点数比较.....	213	文件 I/O	223
阿达马矩阵 (Hadamard matrix)	215	增加输出流长度、递归深度	223
幻方.....	215	自定义结构体	223
最长严格/非严格递增子序列 (LIS)	216	数据结构	223
一维.....	216	其他	223
二维+输出方案.....	216	OJ 测试	223
cout 输出流控制.....	217	GNU C++ 版本测试	223
读取一行数字，个数未知.....	217	编译器位数测试	224
约瑟夫问题.....	218	评测器环境测试	224
日期换算（基姆拉尔森公式）	218	运算速度测试	224
单调队列.....	218	编译器设置	225
高精度快速幂.....	218		

内容并不完美，仅供参考。如有需要，您可以通过以下渠道获取最新版本或与我取得联系

www.github.com/hh2048

WIDA, 2024.11.19

1 基础算法

1.1 头文件

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define int long long
4 template<class... Args> void _(Args... args) {
5     auto _ = [&](auto x) { cout << x << " "; };
6     cout << "--->";
7     int arr[] = {(_(args), 0)...};
8     cout << "\n";
9 }
10
11 signed main() {
12     int Task = 1;
13     for (cin >> Task; Task; Task--) {}
14 }
15 int __OI_INIT__ = []() {
16     ios::sync_with_stdio(0), cin.tie(0);
17     cout.tie(0);
18     cout << fixed << setprecision(12);
19     return 0;
20 }()

```

1.2 常用函数

```

1 i64 mysqrt(i64 n) { // 针对 sqrt 无法精确计算 ll 型
2     i64 ans = sqrt(n);
3     while ((ans + 1) * (ans + 1) <= n) ans++;
4     while (ans * ans > n) ans--;
5     return ans;
6 }
7 int mylcm(int x, int y) {
8     return x / gcd(x, y) * y;
9 }

```

```

1 template<class T> int log2floor(T n) { // 针对 log2 无法精确计算 ll 型；向下取整
2     assert(n > 0);
3     for (T i = 0, chk = 1;; i++, chk *= 2) {
4         if (chk <= n && n < chk * 2) {
5             return i;
6         }
7     }
8 }
9 template<class T> int log2ceil(T n) { // 向上取整
10    assert(n > 0);
11    for (T i = 0, chk = 1;; i++, chk *= 2) {
12        if (n <= chk) {
13            return i;
14        }
15    }
16 }
17 int log2floor(int x) {
18     return 31 - __builtin_clz(x);
19 }
20 int log2ceil(int x) { // 向上取整
21     return log2floor(x) + (__builtin_popcount(x) != 1);
22 }

```

```

1 template <class T> T sign(const T &a) {
2     return a == 0 ? 0 : (a < 0 ? -1 : 1);
3 }
4 template <class T> T floor(const T &a, const T &b) { // 注意大数据计算时会丢失精度
5     T A = abs(a), B = abs(b);
6     assert(B != 0);
7     return sign(a) * sign(b) > 0 ? A / B : -(A + B - 1) / B;
8 }
9 template <class T> T ceil(const T &a, const T &b) { // 注意大数据计算时会丢失精度
10    T A = abs(a), B = abs(b);
11    assert(b != 0);
12    return sign(a) * sign(b) > 0 ? (A + B - 1) / B : -A / B;
13 }

```

1.3 最大公约数 gcd

速度不如内置函数！ 以 $\mathcal{O}(\log(a+b))$ 的复杂度求解最大公约数。与内置函数 `__gcd` 功能基本相同（支持 $a, b \leq 0$ ）。有使用位运算的常数优化版本。

```
1 | inline int mygcd(int a, int b) { return b ? gcd(b, a % b) : a; }
```

1.4 整数域二分

1.4.1 旧版（无法处理负数情况）

- 在递增序列 a 中查找 $\geq x$ 数中最小的一个（即 x 或 x 的后继）

```

1 while (l < r) {
2     int mid = (l + r) / 2;
3     if (a[mid] >= x) {
4         r = mid;
5     } else {
6         l = mid + 1;
7     }
8 }
9 return a[l];

```

- 在递增序列 a 中查找 $\leq x$ 数中最大的一个（即 x 或 x 的前驱）

```

1 while (l < r) {
2     int mid = (l + r + 1) / 2;
3     if (a[mid] <= x) {
4         l = mid;
5     } else {
6         r = mid - 1;
7     }
8 }
9 return a[l];

```

1.4.2 新版

- x 或 x 的后继

```

1 int l = 0, r = 1E8, ans = r;
2 while (l <= r) {
3     int mid = (l + r) / 2;
4     if (judge(mid)) {
5         r = mid - 1;
6         ans = mid;
7     } else {
8         l = mid + 1;
9     }
10 }
11 return ans;

```

- x 或 x 的前驱

```

1 int l = 0, r = 1E8, ans = l;
2 while (l <= r) {
3     int mid = (l + r) / 2;
4     if (judge(mid)) {
5         l = mid + 1;
6         ans = mid;
7     } else {
8         r = mid - 1;
9     }
10 }
11 return ans;

```

1.5 实数域二分

目前主流的写法是限制二分次数。

```

1 for (int t = 1; t <= 100; t++) {
2     ld mid = (l + r) / 2;
3     if (judge(mid)) r = mid;
4     else l = mid;
5 }
6 cout << l << endl;

```

1.6 整数域三分

```

1 while (l < r) {
2     int mid = (l + r) / 2;
3     if (check(mid) <= check(mid + 1)) r = mid;
4     else l = mid + 1;
5 }
6 cout << check(l) << endl;

```

1.7 实数域三分

限制次数实现。

```
1 ld l = -1E9, r = 1E9;
2 for (int t = 1; t <= 100; t++) {
3     ld mid1 = (l * 2 + r) / 3;
4     ld mid2 = (l + r * 2) / 3;
5     if (judge(mid1) < judge(mid2)) {
6         r = mid2;
7     } else {
8         l = mid1;
9     }
10}
11 cout << l << endl;
```

/END/

2 树上问题

2.1 树的直径

```

1 struct Tree {
2     int n;
3     vector<vector<int>> ver;
4     Tree(int n) {
5         this->n = n;
6         ver.resize(n + 1);
7     }
8     void add(int x, int y) {
9         ver[x].push_back(y);
10        ver[y].push_back(x);
11    }
12    int getlen(int root) { // 获取x所在树的直径
13        map<int, int> dep; // map用于优化输入为森林时的深度计算，亦可用vector
14        function<void(int, int)> dfs = [&](int x, int fa) -> void {
15            for (auto y : ver[x]) {
16                if (y == fa) continue;
17                dep[y] = dep[x] + 1;
18                dfs(y, x);
19            }
20            if (dep[x] > dep[root]) {
21                root = x;
22            }
23        };
24        dfs(root, 0);
25        int st = root; // 记录直径端点
26
27        dep.clear();
28        dfs(root, 0);
29        int ed = root; // 记录直径另一端点
30
31        return dep[root];
32    }
33};

```

2.2 树论大封装（直径+重心+中心）

```

1 struct Tree {
2     int n;
3     vector<vector<pair<int, int>>> e;
4     vector<int> dep, parent, maxdep, d1, d2, s1, s2, up;
5     Tree(int n) {
6         this->n = n;
7         e.resize(n + 1);
8         dep.resize(n + 1);
9         parent.resize(n + 1);
10        maxdep.resize(n + 1);
11        d1.resize(n + 1);
12        d2.resize(n + 1);
13        s1.resize(n + 1);
14        s2.resize(n + 1);
15        up.resize(n + 1);
16    }
17    void add(int u, int v, int w) {
18        e[u].push_back({w, v});
19        e[v].push_back({w, u});
20    }
21    void dfs(int u, int fa) {

```

```

22     maxdep[u] = dep[u];
23     for (auto [w, v] : e[u]) {
24         if (v == fa) continue;
25         dep[v] = dep[u] + 1;
26         parent[v] = u;
27         dfs(v, u);
28         maxdep[u] = max(maxdep[u], maxdep[v]);
29     }
30 }
31
32 void dfs1(int u, int fa) {
33     for (auto [w, v] : e[u]) {
34         if (v == fa) continue;
35         dfs1(v, u);
36         int x = d1[v] + w;
37         if (x > d1[u]) {
38             d2[u] = d1[u], s2[u] = s1[u];
39             d1[u] = x, s1[u] = v;
40         } else if (x > d2[u]) {
41             d2[u] = x, s2[u] = v;
42         }
43     }
44 }
45 void dfs2(int u, int fa) {
46     for (auto [w, v] : e[u]) {
47         if (v == fa) continue;
48         if (s1[u] == v) {
49             up[v] = max(up[u], d2[u]) + w;
50         } else {
51             up[v] = max(up[u], d1[u]) + w;
52         }
53         dfs2(v, u);
54     }
55 }
56
57 int radius, center, diam;
58 void getCenter() {
59     center = 1; //中心
60     for (int i = 1; i <= n; i++) {
61         if (max(d1[i], up[i]) < max(d1[center], up[center])) {
62             center = i;
63         }
64     }
65     radius = max(d1[center], up[center]); //距离最远点的距离的最小值
66     diam = d1[center] + up[center] + 1; //直径
67 }
68
69 int rem; //删除重心后剩余连通块体积的最小值
70 int cog; //重心
71 vector<bool> vis;
72 void getCog() {
73     vis.resize(n);
74     rem = INT_MAX;
75     cog = 1;
76     dfsCog(1);
77 }
78 int dfsCog(int u) {
79     vis[u] = true;
80     int s = 1, res = 0;
81     for (auto [w, v] : e[u]) {
82         if (vis[v]) continue;
83         int t = dfsCog(v);
84         res = max(res, t);
85         s += t;
}

```

```

86     }
87     res = max(res, n - s);
88     if (res < rem) {
89         rem = res;
90         cog = u;
91     }
92     return s;
93 }
94 };

```

2.3 点分治 / 树的重心

重心的定义：删除树上的某一个点，会得到若干棵子树；删除某点后，得到的最大子树最小，这个点称为重心。我们假设某个点是重心，记录此时最大子树的最小值，遍历完所有点后取最大值即可。

重心的性质：重心最多可能会有两个，且此时两个重心相邻。

点分治的一般过程是：取重心为新树的根，随后使用 `dfs` 处理当前这棵树，灵活运用 `child` 和 `pre` 两个数组分别计算通过根节点、不通过根节点的路径信息，根据需要进行答案的更新；再对子树分治，寻找子树的重心，……。时间复杂度降至 $O(N \log N)$ 。

```

1 int root = 0, MaxTree = 1e18; // 分别代表重心下标、最大子树大小
2 vector<int> vis(n + 1), siz(n + 1);
3 auto get = [&](auto self, int x, int fa, int n) -> void { // 获取树的重心
4     siz[x] = 1;
5     int val = 0;
6     for (auto [y, w] : ver[x]) {
7         if (y == fa || vis[y]) continue;
8         self(self, y, x, n);
9         siz[x] += siz[y];
10        val = max(val, siz[y]);
11    }
12    val = max(val, n - siz[x]);
13    if (val < MaxTree) {
14        MaxTree = val;
15        root = x;
16    }
17};
18
19 auto clac = [&](int x) -> void { // 以 x 为新的根，维护询问
20     set<int> pre = {0}; // 记录到根节点 x 距离为 i 的路径是否存在
21     vector<int> dis(n + 1);
22     for (auto [y, w] : ver[x]) {
23         if (vis[y]) continue;
24         vector<int> child; // 记录 x 的子树节点的深度信息
25         auto dfs = [&](auto self, int x, int fa) -> void {
26             child.push_back(dis[x]);
27             for (auto [y, w] : ver[x]) {
28                 if (y == fa || vis[y]) continue;
29                 dis[y] = dis[x] + w;
30                 self(self, y, x);
31             }
32         };
33         dis[y] = w;
34         dfs(dfs, y, x);
35
36         for (auto it : child) {
37             for (int i = 1; i <= m; i++) { // 根据询问更新值
38                 if (q[i] < it || !pre.count(q[i] - it)) continue;
39                 ans[i] = 1;
40             }
41         }
42     }
43 };

```

```

42     pre.insert(child.begin(), child.end());
43 }
44 };
45
46 auto dfz = [&](auto self, int x, int fa) -> void { // 点分治
47     vis[x] = 1; // 标记已经被更新过的旧重心，确保只对子树分治
48     clac(x);
49     for (auto [y, w] : ver[x]) {
50         if (y == fa || vis[y]) continue;
51         MaxTree = 1e18;
52         get(get, y, x, siz[y]);
53         self(self, root, x);
54     }
55 };
56
57 get(get, 1, 0, n);
58 dfz(dfz, root, 0);

```

2.4 最近公共祖先 LCA

2.4.1 树链剖分解法

预处理时间复杂度 $\mathcal{O}(N)$ ；单次查询 $\mathcal{O}(\log N)$ ，常数较小。

```

1 struct HLD {
2     int n, idx;
3     vector<vector<int>> ver;
4     vector<int> siz, dep;
5     vector<int> top, son, parent;
6
7     HLD(int n) {
8         this->n = n;
9         ver.resize(n + 1);
10        siz.resize(n + 1);
11        dep.resize(n + 1);
12
13        top.resize(n + 1);
14        son.resize(n + 1);
15        parent.resize(n + 1);
16    }
17    void add(int x, int y) { // 建立双向边
18        ver[x].push_back(y);
19        ver[y].push_back(x);
20    }
21    void dfs1(int x) {
22        siz[x] = 1;
23        dep[x] = dep[parent[x]] + 1;
24        for (auto y : ver[x]) {
25            if (y == parent[x]) continue;
26            parent[y] = x;
27            dfs1(y);
28            siz[x] += siz[y];
29            if (siz[y] > siz[son[x]]) {
30                son[x] = y;
31            }
32        }
33    }
34    void dfs2(int x, int up) {
35        top[x] = up;
36        if (son[x]) dfs2(son[x], up);
37        for (auto y : ver[x]) {
38            if (y == parent[x] || y == son[x]) continue;

```

```

39         dfs2(y, y);
40     }
41 }
42 int lca(int x, int y) {
43     while (top[x] != top[y]) {
44         if (dep[top[x]] > dep[top[y]]) {
45             x = parent[top[x]];
46         } else {
47             y = parent[top[y]];
48         }
49     }
50     return dep[x] < dep[y] ? x : y;
51 }
52 int clac(int x, int y) { // 查询两点间距离
53     return dep[x] + dep[y] - 2 * dep[lca(x, y)];
54 }
55 void work(int root = 1) { // 在此初始化
56     dfs1(root);
57     dfs2(root, root);
58 }
59 };

```

2.4.2 树上倍增解法

预处理时间复杂度 $\mathcal{O}(N \log N)$ ；单次查询 $\mathcal{O}(\log N)$ ，但是常数比树链剖分解法更大。

封装一：基础封装，针对无权图。

```

1 struct Tree {
2     int n;
3     vector<vector<int>> ver, val;
4     vector<int> lg, dep;
5     Tree(int n) {
6         this->n = n;
7         ver.resize(n + 1);
8         val.resize(n + 1, vector<int>(30));
9         lg.resize(n + 1);
10        dep.resize(n + 1);
11        for (int i = 1; i <= n; i++) { // 预处理 log
12            lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
13        }
14    }
15    void add(int x, int y) { // 建立双向边
16        ver[x].push_back(y);
17        ver[y].push_back(x);
18    }
19    void dfs(int x, int fa) {
20        val[x][0] = fa; // 储存 x 的父节点
21        dep[x] = dep[fa] + 1;
22        for (int i = 1; i <= lg[dep[x]]; i++) {
23            val[x][i] = val[val[x][i - 1]][i - 1];
24        }
25        for (auto y : ver[x]) {
26            if (y == fa) continue;
27            dfs(y, x);
28        }
29    }
30    int lca(int x, int y) {
31        if (dep[x] < dep[y]) swap(x, y);
32        while (dep[x] > dep[y]) {
33            x = val[x][lg[dep[x]] - dep[y]] - 1;
34        }

```

```

35     if (x == y) return x;
36     for (int k = lg[dep[x]] - 1; k >= 0; k--) {
37         if (val[x][k] == val[y][k]) continue;
38         x = val[x][k];
39         y = val[y][k];
40     }
41     return val[x][0];
42 }
43 int clac(int x, int y) { // 倍增查询两点间距离
44     return dep[x] + dep[y] - 2 * dep[lca(x, y)];
45 }
46 void work(int root = 1) { // 在此初始化
47     dfs(root, 0);
48 }
49 };

```

封装二：扩展封装，针对有权图，支持“倍增查询两点路径上的最大边权”功能。

```

1 struct Tree {
2     int n;
3     vector<vector<int>> val, Max;
4     vector<vector<pair<int, int>>> ver;
5     vector<int> lg, dep;
6     Tree(int n) {
7         this->n = n;
8         ver.resize(n + 1);
9         val.resize(n + 1, vector<int>(30));
10        Max.resize(n + 1, vector<int>(30));
11        lg.resize(n + 1);
12        dep.resize(n + 1);
13        for (int i = 1; i <= n; i++) { // 预处理 log
14            lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);
15        }
16    }
17    void add(int x, int y, int w) { // 建立双向边
18        ver[x].push_back({y, w});
19        ver[y].push_back({x, w});
20    }
21    void dfs(int x, int fa) {
22        val[x][0] = fa;
23        dep[x] = dep[fa] + 1;
24        for (int i = 1; i <= lg[dep[x]]; i++) {
25            val[x][i] = val[val[x][i - 1]][i - 1];
26            Max[x][i] = max(Max[x][i - 1], Max[val[x][i - 1]][i - 1]);
27        }
28        for (auto [y, w] : ver[x]) {
29            if (y == fa) continue;
30            Max[y][0] = w;
31            dfs(y, x);
32        }
33    }
34    int lca(int x, int y) {
35        if (dep[x] < dep[y]) swap(x, y);
36        while (dep[x] > dep[y]) {
37            x = val[x][lg[dep[x]] - dep[y]];
38        }
39        if (x == y) return x;
40        for (int k = lg[dep[x]] - 1; k >= 0; k--) {
41            if (val[x][k] == val[y][k]) continue;
42            x = val[x][k];
43            y = val[y][k];
44        }
45        return val[x][0];

```

```

46 }
47 int clac(int x, int y) { // 倍增查询两点间距离
48     return dep[x] + dep[y] - 2 * dep[lca(x, y)];
49 }
50 int query(int x, int y) { // 倍增查询两点路径上的最大边权 (带权图)
51     auto get = [&](int x, int y) -> int {
52         int ans = 0;
53         if (x == y) return ans;
54         for (int i = lg[dep[x]]; i >= 0; i--) {
55             if (dep[val[x][i]] > dep[y]) {
56                 ans = max(ans, Max[x][i]);
57                 x = val[x][i];
58             }
59         }
60         ans = max(ans, Max[x][0]);
61         return ans;
62     };
63     int fa = lca(x, y);
64     return max(get(x, fa), get(y, fa));
65 }
66 void work(int root = 1) { // 在此初始化
67     dfs(root, 0);
68 }
69 };

```

2.5 树上路径交

计算两条路径的交点数量，直接载入任意 LCA 封装即可。

```

1 int intersection(int x, int y, int X, int Y) {
2     vector<int> t = {lca(x, X), lca(x, Y), lca(y, X), lca(y, Y)};
3     sort(t.begin(), t.end());
4     int r = lca(x, y), R = lca(X, Y);
5     if (dep[t[0]] < min(dep[r], dep[R]) || dep[t[2]] < max(dep[r], dep[R])) {
6         return 0;
7     }
8     return 1 + clac(t[2], t[3]);
9 }

```

2.6 树上启发式合并 (DSU on tree)

$\mathcal{O}(N \log N)$ 。

```

1 struct HLD {
2     vector<vector<int>> e;
3     vector<int> siz, son, cnt;
4     vector<LL> ans;
5     LL sum, Max;
6     int hson;
7     HLD(int n) {
8         e.resize(n + 1);
9         siz.resize(n + 1);
10        son.resize(n + 1);
11        ans.resize(n + 1);
12        cnt.resize(n + 1);
13        hson = 0;
14        sum = 0;
15        Max = 0;
16    }
17    void add(int u, int v) {
18        e[u].push_back(v);

```

```

19     e[v].push_back(u);
20 }
21 void dfs1(int u, int fa) {
22     siz[u] = 1;
23     for (auto v : e[u]) {
24         if (v == fa) continue;
25         dfs1(v, u);
26         siz[u] += siz[v];
27         if (siz[v] > siz[son[u]]) son[u] = v;
28     }
29 }
30 void calc(int u, int fa, int val) {
31     cnt[color[u]] += val;
32     if (cnt[color[u]] > Max) {
33         Max = cnt[color[u]];
34         sum = color[u];
35     } else if (cnt[color[u]] == Max) {
36         sum += color[u];
37     }
38     for (auto v : e[u]) {
39         if (v == fa || v == hson) continue;
40         calc(v, u, val);
41     }
42 }
43 void dfs2(int u, int fa, int opt) {
44     for (auto v : e[u]) {
45         if (v == fa || v == son[u]) continue;
46         dfs2(v, u, 0);
47     }
48     if (son[u]) {
49         dfs2(son[u], u, 1);
50         hson = son[u]; //记录重链编号，计算的时候跳过
51     }
52     calc(u, fa, 1);
53     hson = 0; //消除的时候所有儿子都清除
54     ans[u] = sum;
55     if (!opt) {
56         calc(u, fa, -1);
57         sum = 0;
58         Max = 0;
59     }
60 }
61 };

```

/END/

3 图论

3.1 常见概念

oriented graph : 有向图

bidirectional edges : 双向边

平面图：若能将无向图 $G = (V, E)$ 画在平面上使得任意两条无重合顶点的边不相交，则称 G 是平面图。

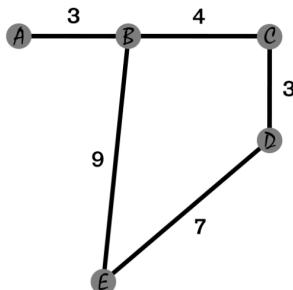
无向正权图上某一点的偏心距：记为 $ecc(u) = \max \{ dist(u, v) \}$ ，即以这个点为源，到其他点的所有最短路的最大值。如下图 A 点， $ecc(A)$ 即为 12。

图的直径：定义为 $d = \max \{ ecc(u) \}$ ，即最大的偏心距，亦可以简化为图中最远的一对点的距离。

图的中心：定义为 $arg = \min \{ ecc(u) \}$ ，即偏心距最小的点。如下图，图的中心即为 B 点。

图的绝对中心：可以定义在边上的图的中心。

图的半径：图的半径不同于圆的半径，其不等于直径的一半（但对于绝对中心定义上的直径而言是一半）。定义为 $r = \min \{ ecc(u) \}$ ，即中心的偏心距。计算方式：使用全源最短路，计算出所有点的偏心距，再加以计算。



3.2 单源最短路径 (SSSP问题)

3.2.1 (正权稀疏图) 动态数组存图+Dijkstra算法

使用优先队列优化，以 $\mathcal{O}(M \log N)$ 的复杂度计算。

```

1 | vector<int> dis(n + 1, 1E18);
2 | auto djikstra = [&](int s = 1) -> void {
3 |     using PII = pair<int, int>;
4 |     priority_queue<PII, vector<PII>, greater<PII>> q;
5 |     q.emplace(0, s);
6 |     dis[s] = 0;
7 |     vector<int> vis(n + 1);
8 |     while (!q.empty()) {
9 |         int x = q.top().second;
10 |        q.pop();
11 |        if (vis[x]) continue;
12 |        vis[x] = 1;
13 |        for (auto [y, w] : ver[x]) {
14 |            if (dis[y] > dis[x] + w) {
15 |                dis[y] = dis[x] + w;
16 |                q.emplace(dis[y], y);
  
```

```

17     }
18 }
19 }
20 };

```

3.2.2 (负权图、判负环) Bellman-ford 算法

使用结构体存边（该算法无需存图），以 $\mathcal{O}(NM)$ 的复杂度计算。

```

1 int n, m, s;
2 cin >> n >> m >> s;
3
4 vector<tuple<int, int, i64>> ver(m + 1);
5 for (int i = 1; i <= m; ++i) {
6     int x, y;
7     i64 w;
8     cin >> x >> y >> w;
9     ver[i] = {x, y, w};
10}
11
12 vector<i64> dis(n + 1, inf), chk(n + 1);
13 dis[s] = 0;
14 for (int i = 1; i <= 2 * n; ++i) { // 双倍松弛，获取负环信息
15     vector<i64> backup = dis;
16     for (int j = 1; j <= m; ++j) {
17         auto [x, y, w] = ver[j];
18         chk[y] |= (i > n && backup[x] + w < dis[y]);
19         dis[y] = min(dis[y], backup[x] + w);
20     }
21 }
22
23 for (int i = 1; i <= n; ++i) {
24     if (i == s) {
25         cout << 0 << " ";
26     } else if (dis[i] >= inf / 2) {
27         cout << "no ";
28     } else if (chk[i]) {
29         cout << "inf ";
30     } else {
31         cout << dis[i] << " ";
32     }
33 }

```

3.2.3 (负权图) SPFA 算法

以 $\mathcal{O}(KM)$ 的复杂度计算，其中 K 虽然为常数，但是可以通过特殊的构造退化成接近 N ，需要注意被卡。

```

1 const int N = 1e5 + 7, M = 1e6 + 7;
2 int n, m;
3 int ver[M], ne[M], h[N], edge[M], tot;
4 int d[N], v[N];
5
6 void add(int x, int y, int w) {
7     ver[++tot] = y, ne[tot] = h[x], h[x] = tot;
8     edge[tot] = w;
9 }
10 void spfa() {
11     ms(d, 0x3f); d[1] = 0;
12     queue<int> q; q.push(1);
13     v[1] = 1;

```

```

14     while(!q.empty()) {
15         int x = q.front(); q.pop(); v[x] = 0;
16         for (int i = h[x]; i; i = ne[i]) {
17             int y = ver[i];
18             if(d[y] > d[x] + edge[i]) {
19                 d[y] = d[x] + edge[i];
20                 if(v[y] == 0) q.push(y), v[y] = 1;
21             }
22         }
23     }
24 }
25 int main() {
26     cin >> n >> m;
27     for (int i = 1; i <= m; ++ i) {
28         int x, y, w; cin >> x >> y >> w;
29         add(x, y, w);
30     }
31     spfa();
32     for (int i = 1; i <= n; ++ i) {
33         if (d[i] == INF) cout << "N" << endl;
34         else cout << d[n] << endl;
35     }
36 }
```

3.3 多源汇最短路 (APSP问题)

使用邻接矩阵存图，可以处理负权边，以 $\mathcal{O}(N^3)$ 的复杂度计算。注意，这里建立的是单向边，计算双向边需要额外加边。

```

1 const int N = 210;
2 int n, m, d[N][N];
3
4 void floyd() {
5     for (int k = 1; k <= n; k++)
6         for (int i = 1; i <= n; i++)
7             for (int j = 1; j <= n; j++)
8                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
9 }
10 int main() {
11     cin >> n >> m;
12     for (int i = 1; i <= n; i++)
13         for (int j = 1; j <= n; j++)
14             if (i == j) d[i][j] = 0;
15             else d[i][j] = INF;
16     while (m--) {
17         int x, y, w; cin >> x >> y >> w;
18         d[x][y] = min(d[x][y], w);
19     }
20     floyd();
21     for (int i = 1; i <= n; ++ i) {
22         for (int j = 1; j <= n; ++ j) {
23             if (d[i][j] > INF / 2) cout << "N" << endl;
24             else cout << d[i][j] << endl;
25         }
26     }
27 }
```

3.4 平面图最短路（对偶图）

对于矩阵图，建立对偶图的过程如下（注释部分为建立原图），其中数据的给出顺序依次为：各 $n(n+1)$ 个数字分别代表从左向右、从上向下、从右向左、从下向上的边。

```

1  for (int i = 1; i <= n + 1; i++) {
2      for (int j = 1, w; j <= n; j++) {
3          cin >> w;
4          int pre = Hash(i - 1, j), now = Hash(i, j);
5          if (i == 1) {
6              add(s, now, w);
7          } else if (i == n + 1) {
8              add(pre, t, w);
9          } else {
10             add(pre, now, w);
11         }
12         // flow.add(Hash(i, j), Hash(i, j + 1), w);
13     }
14 }
15 for (int i = 1; i <= n; i++) {
16     for (int j = 1, w; j <= n + 1; j++) {
17         cin >> w;
18         int now = Hash(i, j), net = Hash(i, j - 1);
19         if (j == 1) {
20             add(now, t, w);
21         } else if (j == n + 1) {
22             add(s, net, w);
23         } else {
24             add(now, net, w);
25         }
26         // flow.add(Hash(i, j), Hash(i + 1, j), w);
27     }
28 }
29 for (int i = 1; i <= n + 1; i++) {
30     for (int j = 1, w; j <= n; j++) {
31         cin >> w;
32         int now = Hash(i, j), net = Hash(i - 1, j);
33         if (i == 1) {
34             add(now, s, w);
35         } else if (i == n + 1) {
36             add(t, net, w);
37         } else {
38             add(now, net, w);
39         }
40         // flow.add(Hash(i, j), Hash(i, j - 1), w);
41     }
42 }
43 for (int i = 1; i <= n; i++) {
44     for (int j = 1, w; j <= n + 1; j++) {
45         cin >> w;
46         int pre = Hash(i, j - 1), now = Hash(i, j);
47         if (j == 1) {
48             add(t, now, w);
49         } else if (j == n + 1) {
50             add(pre, s, w);
51         } else {
52             add(pre, now, w);
53         }
54         // flow.add(Hash(i, j), Hash(i - 1, j), w);
55     }
56 }
```

3.5 最小生成树 (MST问题)

3.5.1 (稀疏图) Prim算法

使用邻接矩阵存图，以 $\mathcal{O}(N^2 + M)$ 的复杂度计算，思想与 djikstra 基本一致。

```

1 const int N = 550, INF = 0x3f3f3f3f;
2 int n, m, g[N][N];
3 int d[N], v[N];
4 int prim() {
5     ms(d, 0x3f); //这里的d表示到“最小生成树集合”的距离
6     int ans = 0;
7     for (int i = 0; i < n; ++ i) { //遍历 n 轮
8         int t = -1;
9         for (int j = 1; j <= n; ++ j)
10            if (v[j] == 0 && (t == -1 || d[j] < d[t])) //如果这个点不在集合内且当前距离
集合最近
11            t = j;
12            v[t] = 1; //将t加入“最小生成树集合”
13            if (i && d[t] == INF) return INF; //如果发现不连通，直接返回
14            if (i) ans += d[t];
15            for (int j = 1; j <= n; ++ j) d[j] = min(d[j], g[t][j]); //用t更新其他点到集合
的距离
16        }
17    return ans;
18}
19 int main() {
20     ms(g, 0x3f); cin >> n >> m;
21     while (m -- ) {
22         int x, y, w; cin >> x >> y >> w;
23         g[x][y] = g[y][x] = min(g[x][y], w);
24     }
25     int t = prim();
26     if (t == INF) cout << "impossible" << endl;
27     else cout << t << endl;
28 } //22.03.19已测试

```

3.5.2 (稠密图) Kruskal算法

平均时间复杂度为 $\mathcal{O}(M \log M)$ ，简化了并查集。

```

1 struct DSU {
2     vector<int> fa;
3     DSU(int n) : fa(n + 1) {
4         iota(fa.begin(), fa.end(), 0);
5     }
6     int get(int x) {
7         while (x != fa[x]) {
8             x = fa[x] = fa[fa[x]];
9         }
10        return x;
11    }
12    bool merge(int x, int y) { // 设x是y的祖先
13        x = get(x), y = get(y);
14        if (x == y) return false;
15        fa[y] = x;
16        return true;
17    }
18    bool same(int x, int y) {
19        return get(x) == get(y);
20    }
21};

```

```

22 struct Tree {
23     using TII = tuple<int, int, int>;
24     int n;
25     priority_queue<TII, vector<TII>, greater<TII>> ver;
26
27     Tree(int n) {
28         this->n = n;
29     }
30     void add(int x, int y, int w) {
31         ver.emplace(w, x, y); // 注意顺序
32     }
33     int kruskal() {
34         DSU dsu(n);
35         int ans = 0, cnt = 0;
36         while (ver.size()) {
37             auto [w, x, y] = ver.top();
38             ver.pop();
39             if (dsu.same(x, y)) continue;
40             dsu.merge(x, y);
41             ans += w;
42             cnt++;
43         }
44         assert(cnt < n - 1); // 输入有误，建树失败
45         return ans;
46     }
47 };

```

3.6 缩点 (Tarjan 算法)

3.6.1 (有向图) 强连通分量缩点

强连通分量缩点后的图称为 SCC。以 $\mathcal{O}(N + M)$ 的复杂度完成上述全部操作。

性质：缩点后的图拥有拓扑序 $color_{cnt}, color_{cnt-1}, \dots, 1$ ，可以不再另跑一遍 topsort；缩点后的图是一张有向无环图（DAG、拓扑图）。

```

1 struct SCC {
2     int n, now, cnt;
3     vector<vector<int>> ver;
4     vector<int> dfn, low, col, S;
5
6     SCC(int n) : n(n), ver(n + 1), low(n + 1) {
7         dfn.resize(n + 1, -1);
8         col.resize(n + 1, -1);
9         now = cnt = 0;
10    }
11    void add(int x, int y) {
12        ver[x].push_back(y);
13    }
14    void tarjan(int x) {
15        dfn[x] = low[x] = now++;
16        S.push_back(x);
17        for (auto y : ver[x]) {
18            if (dfn[y] == -1) {
19                tarjan(y);
20                low[x] = min(low[x], low[y]);
21            } else if (col[y] == -1) {
22                low[x] = min(low[x], dfn[y]);
23            }
24        }
25        if (dfn[x] == low[x]) {
26            int pre;

```

```

27     cnt++;
28     do {
29         pre = S.back();
30         col[pre] = cnt;
31         S.pop_back();
32     } while (pre != x);
33 }
34 }
35 auto work() { // [cnt 新图的顶点数量]
36     for (int i = 1; i <= n; i++) { // 避免图不连通
37         if (dfn[i] == -1) {
38             tarjan(i);
39         }
40     }
41
42     vector<int> siz(cnt + 1); // siz 每个 scc 中点的数量
43     vector<vector<int>> adj(cnt + 1);
44     for (int i = 1; i <= n; i++) {
45         siz[col[i]]++;
46         for (auto j : ver[i]) {
47             int x = col[i], y = col[j];
48             if (x != y) {
49                 adj[x].push_back(y);
50             }
51         }
52     }
53     return {cnt, adj, col, siz};
54 }
55 };

```

3.6.2 (无向图) 割边缩点

割边缩点后的图称为边双连通图 (E-DCC)，该模板可以在 $\mathcal{O}(N + M)$ 复杂度内求解图中全部割边、划分边双（颜色相同的点位于同一个边双连通分量中）。

割边（桥）：将某边 e 删去后，原图分成两个以上不相连的子图，称 e 为图的割边。

边双连通：在一张连通的无向图中，对于两个点 u 和 v ，删去任何一条边（只能删去一条）它们依旧连通，则称 u 和 v 边双连通。一个图如果不存在割边，则它是一个边双连通图。

性质补充：对于一个边双，删去任意边后依旧联通；对于边双中的任意两点，一定存在两条不相交的路径连接这两个点（路径上可以有公共点，但是没有公共边）。

```

1 struct EDCC {
2     int n, m, now, cnt;
3     vector<vector<array<int, 2>>> ver;
4     vector<int> dfn, low, col, S;
5     set<array<int, 2>> bridge, direct; // 如果不需要，删除这一部分可以得到一些时间上的优化
6
7     EDCC(int n) : n(n), low(n + 1), ver(n + 1), dfn(n + 1), col(n + 1) {
8         m = now = cnt = 0;
9     }
10    void add(int x, int y) { // 和 scc 相比多了一条连边
11        ver[x].push_back({y, m});
12        ver[y].push_back({x, m++});
13    }
14    void tarjan(int x, int fa) {
15        dfn[x] = low[x] = ++now;
16        S.push_back(x);
17        for (auto &[y, id] : ver[x]) {
18            if (!dfn[y]) {

```

```

19         direct.insert({x, y});
20         tarjan(y, id);
21         low[x] = min(low[x], low[y]);
22         if (dfn[x] < low[y]) {
23             bridge.insert({x, y});
24         }
25     } else if (id != fa && dfn[y] < dfn[x]) {
26         direct.insert({x, y});
27         low[x] = min(low[x], dfn[y]);
28     }
29 }
30 if (dfn[x] == low[x]) {
31     int pre;
32     cnt++;
33     do {
34         pre = S.back();
35         col[pre] = cnt;
36         S.pop_back();
37     } while (pre != x);
38 }
39 }
40 auto work() {
41     for (int i = 1; i <= n; i++) { // 避免图不连通
42         if (!dfn[i]) {
43             tarjan(i, 0);
44         }
45     }
46 /**
47 * @param cnt 新图的顶点数量，adj 新图，col 旧图节点对应的新图节点
48 * @param siz 旧图每一个边双中点的数量
49 * @param bridge 全部割边，direct 非割边定向
50 */
51 vector<int> siz(cnt + 1);
52 vector<vector<int>> adj(cnt + 1);
53 for (int i = 1; i <= n; i++) {
54     siz[col[i]]++;
55     for (auto &[j, id] : ver[i]) {
56         int x = col[i], y = col[j];
57         if (x != y) {
58             adj[x].push_back(y);
59         }
60     }
61 }
62 return tuple{cnt, adj, col, siz};
63 }
64 };

```

3.6.3 (无向图) 割点缩点

割点缩点后的图称为点双连通图 (V-DCC)，该模板可以在 $\mathcal{O}(N + M)$ 复杂度内求解图中全部割点、划分点双（颜色相同的点位于同一个点双连通分量中）。

割点（割顶）：将与某点 i 连接的所有边删去后，原图分成两个以上不相连的子图，称 i 为图的割点。

点双连通：在一张连通的无向图中，对于两个点 u 和 v ，删去任何一个点（只能删去一个，且不能删 u 和 v 自己）它们依旧连通，则称 u 和 v 边双连通。如果一个图不存在割点，那么它是一个点双连通图。

性质补充：每一个割点至少属于两个点双。


```

66         }
67     }
68     return {cnt, adj};
69 }
70 void work() {
71     for (int i = 1; i <= n; ++i) { // 避免图不连通
72         if (!dfn[i]) {
73             tarjan(i, i);
74         }
75     }
76 }
77 }
78 };

```

3.7 染色法判定二分图 (dfs算法)

判断一张图能否被二分染色。

```

1 vector<int> vis(n + 1);
2 auto dfs = [&](auto self, int x, int type) -> void {
3     vis[x] = type;
4     for (auto y : ver[x]) {
5         if (vis[y] == type) {
6             cout << "NO\n";
7             exit(0);
8         }
9         if (vis[y]) continue;
10        self(self, y, 3 - type);
11    }
12 };
13 for (int i = 1; i <= n; ++i) {
14     if (vis[i]) {
15         dfs(dfs, i, 1);
16     }
17 }
18 cout << "Yes\n";

```

3.8 链式前向星建图与搜索

很少使用这种建图法。dfs：标准复杂度为 $\mathcal{O}(N + M)$ 。节点子节点的数量包含它自己（至少为 1），深度从 0 开始（根节点深度为 0）。bfs：深度从 1 开始（根节点深度为 1）。topsort：有向无环图（包括非联通）才拥有完整的拓扑序列（故该算法也可用于判断图中是否存在环）。每次找到入度为 0 的点并将其放入待查找队列。

```

1 namespace Graph {
2     const int N = 1e5 + 7;
3     const int M = 1e6 + 7;
4     int tot, h[N], ver[M], ne[M];
5     int deg[N], vis[M];
6
7     void clear(int n) {
8         tot = 0; // 多组样例清空
9         for (int i = 1; i <= n; ++i) {
10            h[i] = 0;
11            deg[i] = vis[i] = 0;
12        }
13    }
14    void add(int x, int y) {
15        ver[++tot] = y, ne[tot] = h[x], h[x] = tot;
16        ++deg[y];

```

```

17 }
18 void dfs(int x) {
19     a.push_back(x); // DFS序
20     siz[x] = vis[x] = 1;
21     for (int i = h[x]; i; i = ne[i]) {
22         int y = ver[i];
23         if (vis[y]) continue;
24         dis[y] = dis[x] + 1;
25         dfs(y);
26         siz[x] += siz[y];
27     }
28     a.push_back(x);
29 }
30 void bfs(int s) {
31     queue<int> q;
32     q.push(s);
33     dis[s] = 1;
34     while (!q.empty()) {
35         int x = q.front();
36         q.pop();
37         for (int i = h[x]; i; i = ne[i]) {
38             int y = ver[i];
39             if (dis[y]) continue;
40             d[y] = d[x] + 1;
41             q.push(y);
42         }
43     }
44 }
45 bool topsort() {
46     queue<int> q;
47     vector<int> ans;
48     for (int i = 1; i <= n; ++i)
49         if (deg[i] == 0) q.push(i);
50     while (!q.empty()) {
51         int x = q.front();
52         q.pop();
53         ans.push_back(x);
54         for (int i = h[x]; i; i = ne[i]) {
55             int y = ver[i];
56             --deg[y];
57             if (deg[y] == 0) q.push(y);
58         }
59     }
60     return ans.size() == n; // 判断是否存在拓扑排序
61 }
62 } // namespace Graph

```

3.9 一般图最大匹配 (带花树算法)

与二分图匹配的差别在于图中可能存在奇环，时间复杂度与边的数量无关，为 $\mathcal{O}(N^3)$ 。下方模板编号从 0 开始，例题为 [UOJ #79. 一般图最大匹配](#)。

```

1 struct Graph {
2     int n;
3     vector<vector<int>> e;
4     Graph(int n) : n(n), e(n) {}
5     void add(int u, int v) {
6         e[u].push_back(v);
7         e[v].push_back(u);
8     }
9     pair<int, vector<int>> work() {
10        vector<int> match(n, -1), vis(n), link(n), f(n), dep(n);

```

```

11     auto find = [&](int u) {
12         while (f[u] != u) u = f[u] = f[f[u]];
13         return u;
14     };
15     auto lca = [&](int u, int v) {
16         u = find(u), v = find(v);
17         while (u != v) {
18             if (dep[u] < dep[v]) swap(u, v);
19             u = find(link[match[u]]);
20         }
21         return u;
22     };
23     queue<int> q;
24     auto blossom = [&](int u, int v, int p) {
25         while (find(u) != p) {
26             link[u] = v;
27             v = match[u];
28             if (vis[v] == 0) {
29                 vis[v] = 1;
30                 q.push(v);
31             }
32             f[u] = f[v] = p;
33             u = link[v];
34         }
35     };
36     auto augment = [&](int u) {
37         while (!q.empty()) q.pop();
38         iota(f.begin(), f.end(), 0);
39         fill(vis.begin(), vis.end(), -1);
40         q.push(u);
41         vis[u] = 1;
42         dep[u] = 0;
43         while (!q.empty()) {
44             int u = q.front();
45             q.pop();
46             for (auto v : e[u]) {
47                 if (vis[v] == -1) {
48                     vis[v] = 0;
49                     link[v] = u;
50                     dep[v] = dep[u] + 1;
51                     if (match[v] == -1) {
52                         for (int x = v, y = u, temp; y != -1;
53                             x = temp, y = x == -1 ? -1 : link[x]) {
54                             temp = match[y];
55                             match[x] = y;
56                             match[y] = x;
57                         }
58                     }
59                 }
60                 vis[match[v]] = 1;
61                 dep[match[v]] = dep[u] + 2;
62                 q.push(match[v]);
63             } else if (vis[v] == 1 && find(v) != find(u)) {
64                 int p = lca(u, v);
65                 blossom(u, v, p);
66                 blossom(v, u, p);
67             }
68         }
69     };
70 };
71     auto greedy = [&]() {
72         for (int u = 0; u < n; ++u) {
73             if (match[u] != -1) continue;
74             for (auto v : e[u]) {

```

```

75         if (match[v] == -1) {
76             match[u] = v;
77             match[v] = u;
78             break;
79         }
80     }
81 }
82 greedy();
83 for (int u = 0; u < n; u++) {
84     if (match[u] == -1) {
85         augment(u);
86     }
87 }
88 int ans = 0;
89 for (int u = 0; u < n; u++) {
90     if (match[u] != -1) {
91         ans++;
92     }
93 }
94 return {ans / 2, match};
95 }
96 };
97
98 signed main() {
99     int n, m;
100    cin >> n >> m;
101
102    Graph graph(n);
103    for (int i = 1; i <= m; i++) {
104        int x, y;
105        cin >> x >> y;
106        graph.add(x - 1, y - 1);
107    }
108    auto [ans, match] = graph.work();
109    cout << ans << endl;
110    for (auto it : match) {
111        cout << it + 1 << " ";
112    }
113 }
114 }
```

3.10 一般图最大权匹配（带权带花树算法）

下方模板编号从 1 开始，复杂度为 $\mathcal{O}(N^3)$ 。

```

1 namespace Graph {
2     const int N = 403 * 2; //两倍点数
3     typedef int T; //权值大小
4     const T inf = numeric_limits<int>::max() >> 1;
5     struct Q { int u, v; T w; } e[N][N];
6     T lab[N];
7     int n, m = 0, id, h, t, lk[N], sl[N], st[N], f[N], b[N][N], s[N], ed[N], q[N];
8     vector<int> p[N];
9 #define dvd(x) (lab[x.u] + lab[x.v] - e[x.u][x.v].w * 2)
10 #define FOR(i, b) for (int i = 1; i <= (int)(b); i++)
11 #define ALL(x) (x).begin(), (x).end()
12 #define ms(x, i) memset(x + 1, i, m * sizeof x[0])
13 void upd(int u, int v) {
14     if (!sl[v] || dvd(e[u][v]) < dvd(e[sl[v]][v])) {
15         sl[v] = u;
16     }
17 }
```

```

18 void ss(int v) {
19     sl[v] = 0;
20     FOR(u, n) {
21         if (e[u][v].w > 0 && st[u] != v && !s[st[u]]) {
22             upd(u, v);
23         }
24     }
25 }
26 void ins(int u) {
27     if (u <= n) { q[+t] = u; }
28     else {
29         for (int v : p[u]) ins(v);
30     }
31 }
32 void mdf(int u, int w) {
33     st[u] = w;
34     if (u > n) {
35         for (int v : p[u]) mdf(v, w);
36     }
37 }
38 int gr(int u, int v) {
39     v = find(ALL(p[u]), v) - p[u].begin();
40     if (v & 1) {
41         reverse(1 + ALL(p[u]));
42         return (int)p[u].size() - v;
43     }
44     return v;
45 }
46 void stm(int u, int v) {
47     lk[u] = e[u][v].v;
48     if (u <= n) return;
49     Q w = e[u][v];
50     int x = b[u][w.u], y = gr(u, x);
51     for (int i = 0; i < y; i++) {
52         stm(p[u][i], p[u][i ^ 1]);
53     }
54     stm(x, v);
55     rotate(p[u].begin(), y + ALL(p[u]));
56 }
57 void aug(int u, int v) {
58     int w = st[lk[u]];
59     stm(u, v);
60     if (!w) return;
61     stm(w, st[f[w]]), aug(st[f[w]], w);
62 }
63 int lca(int u, int v) {
64     for (++id; u | v; swap(u, v)) {
65         if (!u) continue;
66         if (ed[u] == id) return u;
67         ed[u] = id;
68         if (u == st[lk[u]]) u = st[f[u]];
69     }
70     return 0;
71 }
72 void add(int u, int a, int v) {
73     int x = n + 1, i, j;
74     while (x <= m && st[x]) ++x;
75     if (x > m) ++m;
76     lab[x] = s[x] = st[x] = 0;
77     lk[x] = lk[a];
78     p[x].clear();
79     p[x].push_back(a);
80     for (i = u; i != a; i = st[f[j]]) {
81         p[x].push_back(i);

```

```

82         p[x].push_back(j = st[lk[i]]);
83         ins(j);
84     }
85     reverse(1 + ALL(p[x]));
86     for (i = v; i != a; i = st[f[j]]) { // 复制，只需改循环
87         p[x].push_back(i);
88         p[x].push_back(j = st[lk[i]]);
89         ins(j);
90     }
91     mdf(x, x);
92     FOR(i, m) {
93         e[x][i].w = e[i][x].w = 0;
94     }
95     memset(b[x] + 1, 0, n * sizeof b[0][0]);
96     for (int u : p[x]) {
97         FOR(v, m) {
98             if (!e[x][v].w || dvd(e[u][v]) < dvd(e[x][v])) {
99                 e[x][v] = e[u][v], e[v][x] = e[v][u];
100            }
101        }
102        FOR(v, n) {
103            if (b[u][v]) { b[x][v] = u; }
104        }
105    }
106    ss(x);
107 }
108 void ex(int u) {
109     for (int x : p[u]) mdf(x, x);
110     int a = b[u][e[u][f[u]].u], r = gr(u, a);
111     for (int i = 0; i < r; i += 2) {
112         int x = p[u][i], y = p[u][i + 1];
113         f[x] = e[y][x].u;
114         s[x] = 1;
115         s[y] = sl[x] = 0;
116         ss(y), ins(y);
117     }
118     s[a] = 1, f[a] = f[u];
119     for (int i = r + 1; i < p[u].size(); i++) {
120         s[p[u][i]] = -1;
121         ss(p[u][i]);
122     }
123     st[u] = 0;
124 }
125 bool on(const Q &e) {
126     int u = st[e.u], v = st[e.v];
127     if (s[v] == -1) {
128         f[v] = e.u, s[v] = 1;
129         int a = st[lk[v]];
130         sl[v] = sl[a] = s[a] = 0;
131         ins(a);
132     } else if (!s[v]) {
133         int a = lca(u, v);
134         if (!a) {
135             return aug(u, v), aug(v, u), 1;
136         } else {
137             add(u, a, v);
138         }
139     }
140     return 0;
141 }
142 bool bfs() {
143     ms(s, -1), ms(sl, 0);
144     h = 1, t = 0;
145     FOR(i, m) {

```

```

146         if (st[i] == i && !lk[i]) {
147             f[i] = s[i] = 0;
148             ins(i);
149         }
150     }
151     if (h > t) return 0;
152     while (1) {
153         while (h <= t) {
154             int u = q[h++];
155             if (s[st[u]] == 1) continue;
156             FOR(v, n) {
157                 if (e[u][v].w > 0 && st[u] != st[v]) {
158                     if (dvd(e[u][v])) upd(u, st[v]);
159                     else if (on(e[u][v])) return 1;
160                 }
161             }
162         }
163         T x = inf;
164         for (int i = n + 1; i <= m; i++) {
165             if (st[i] == i && s[i] == 1) {
166                 x = min(x, lab[i] >> 1);
167             }
168         }
169         FOR(i, m) {
170             if (st[i] == i && sl[i] && s[i] != 1) {
171                 x = min(x, dvd(e[sl[i]][i]) >> s[i] + 1);
172             }
173         }
174         FOR(i, n) {
175             if (~s[st[i]]) {
176                 if ((lab[i] += (s[st[i]] * 2 - 1) * x) <= 0) return 0;
177             }
178         }
179         for (int i = n + 1; i <= m; i++) {
180             if (st[i] == i && ~s[st[i]]) {
181                 lab[i] += (2 - s[st[i]] * 4) * x;
182             }
183         }
184         h = 1, t = 0;
185         FOR(i, m) {
186             if (st[i] == i && sl[i] && st[sl[i]] != i && !dvd(e[sl[i]][i]) &&
187             on(e[sl[i]][i])) {
188                 return 1;
189             }
190         }
191         for (int i = n + 1; i <= m; i++) {
192             if (st[i] == i && s[i] == 1 && !lab[i]) ex(i);
193         }
194     }
195     return 0;
196 }
197 template<typename TT> i64 work(int N, const vector<tuple<int, int, TT>> &edges)
{
198     ms(ed, 0), ms(lk, 0);
199     n = m = N; id = 0;
200     iota(st + 1, st + n + 1, 1);
201     T wm = 0; i64 r = 0;
202     FOR(i, n) FOR(j, n) {
203         e[i][j] = {i, j, 0};
204     }
205     for (auto [u, v, w] : edges) {
206         wm = max(wm, e[v][u].w = e[u][v].w = max(e[u][v].w, (T)w));
207     }
208     FOR(i, n) { p[i].clear(); }

```

```

208     FOR(i, n) FOR(j, n) {
209         b[i][j] = i * (i == j);
210     }
211     fill_n(lab + 1, n, w);
212     while (bfs()) {};
213     FOR(i, n) if (lk[i]) {
214         r += e[i][lk[i]].w;
215     }
216     return r / 2;
217 }
218 auto match() {
219     vector<array<int, 2>> ans;
220     FOR(i, n) if (lk[i]) {
221         ans.push_back({i, lk[i]});
222     }
223     return ans;
224 }
225 } // namespace Graph
226 using Graph::work, Graph::match;
227
228 signed main() {
229     int n, m;
230     cin >> n >> m;
231     vector<tuple<int, int, i64>> ver(m);
232     for (auto &[u, v, w] : ver) {
233         cin >> u >> v >> w;
234     }
235     cout << work(n, ver) << "\n";
236     auto ans = match();
237 }
```

3.11 二分图最大匹配

二分图：一个图能被分为左右两部分，任何一条边的两个端点都不在同一部分中。

匹配（独立边集）：一个边的集合，这些边没有公共顶点。

二分图最大匹配即找到边的数量最多那个匹配。

一般我们规定，左半部包含 n_1 个点（编号 $1 - n_1$ ），右半部包含 n_2 个点（编号 $1 - n_2$ ），保证任意一条边的两个端点都不可能在同一部分中。

3.11.1 匈牙利算法（KM算法）解

$\mathcal{O}(NM)$ 。

```

1 signed main() {
2     int n1, n2, m;
3     cin >> n1 >> n2 >> m;
4
5     vector<vector<int>> ver(n1 + 1);
6     for (int i = 1; i <= m; ++i) {
7         int x, y;
8         cin >> x >> y;
9         ver[x].push_back(y); // 只需要建立单向边
10    }
11
12    int ans = 0;
13    vector<int> match(n2 + 1);
14    for (int i = 1; i <= n1; ++i) {
15        vector<int> vis(n2 + 1);
16        auto dfs = [&](auto self, int x) -> bool {

```

```

17     for (auto y : ver[x]) {
18         if (vis[y]) continue;
19         vis[y] = 1;
20         if (!match[y] || self(self, match[y])) {
21             match[y] = x;
22             return true;
23         }
24     }
25     return false;
26 };
27 if (dfs(dfs, i)) {
28     ans++;
29 }
30 }
31 cout << ans << endl;
32 }
```

3.11.2 HopcroftKarp算法（基于最大流）解

该算法基于最大流，常数极小，且引入随机化，几乎卡不掉。最坏时间复杂度为 $\mathcal{O}(\sqrt{NM})$ ，经测试，在 N, M 均为 2×10^5 的情况下能在 60ms 内跑完。

```

1 struct HopcroftKarp {
2     int n, m;
3     vector<array<int, 2>> ver;
4     vector<int> l, r;
5
6     HopcroftKarp(int n, int m) : n(n), m(m) { // 左右半部
7         l.assign(n, -1);
8         r.assign(m, -1);
9     }
10    void add(int x, int y) {
11        x--, y--; // 这个板子是 0-idx 的
12        ver.push_back({x, y});
13    }
14    int work() {
15        vector<int> adj(ver.size());
16
17        mt19937 rgen(chrono::steady_clock::now().time_since_epoch().count());
18        shuffle(ver.begin(), ver.end(), rgen); // 随机化防卡
19
20        vector<int> deg(n + 1);
21        for (auto &[u, v] : ver) {
22            deg[u]++;
23        }
24        for (int i = 1; i <= n; i++) {
25            deg[i] += deg[i - 1];
26        }
27        for (auto &[u, v] : ver) {
28            adj[-deg[u]] = v;
29        }
30
31        int ans = 0;
32        vector<int> a, p, q(n);
33        while (true) {
34            a.assign(n, -1), p.assign(n, -1);
35
36            int t = 0;
37            for (int i = 0; i < n; i++) {
38                if (l[i] == -1) {
39                    q[t++] = a[i] = p[i] = i;
40                }
41            }
42
43            for (int i = 0; i < n; i++) {
44                if (p[i] == -1) {
45                    int j = a[p[i]];
46                    if (r[j] == -1) {
47                        r[j] = i;
48                    } else {
49                        int k = l[r[j]];
50                        if (k == -1) {
51                            l[r[j]] = i;
52                        } else {
53                            a[k] = p[i];
54                            p[i] = k;
55                            t++;
56                        }
57                    }
58                }
59            }
60
61            for (int i = 0; i < n; i++) {
62                if (p[i] == -1) {
63                    break;
64                }
65            }
66            if (t == n) {
67                break;
68            }
69        }
70    }
71
72    void print() {
73        for (int i = 0; i < n; i++) {
74            cout << l[i] << " ";
75        }
76        cout << endl;
77    }
78
79    void print() {
80        for (int i = 0; i < n; i++) {
81            cout << r[i] << " ";
82        }
83        cout << endl;
84    }
85
86    void print() {
87        for (int i = 0; i < n; i++) {
88            cout << a[i] << " ";
89        }
90        cout << endl;
91    }
92
93    void print() {
94        for (int i = 0; i < n; i++) {
95            cout << p[i] << " ";
96        }
97        cout << endl;
98    }
99
100   void print() {
101        for (int i = 0; i < n; i++) {
102            cout << q[i] << " ";
103        }
104        cout << endl;
105    }
106
107    void print() {
108        for (int i = 0; i < n; i++) {
109            cout << adj[i] << " ";
110        }
111        cout << endl;
112    }
113
114    void print() {
115        for (int i = 0; i < n; i++) {
116            cout << deg[i] << " ";
117        }
118        cout << endl;
119    }
120
121    void print() {
122        for (int i = 0; i < n; i++) {
123            cout << l[i] << " ";
124        }
125        cout << endl;
126    }
127
128    void print() {
129        for (int i = 0; i < n; i++) {
130            cout << r[i] << " ";
131        }
132        cout << endl;
133    }
134
135    void print() {
136        for (int i = 0; i < n; i++) {
137            cout << a[i] << " ";
138        }
139        cout << endl;
140    }
141
142    void print() {
143        for (int i = 0; i < n; i++) {
144            cout << p[i] << " ";
145        }
146        cout << endl;
147    }
148
149    void print() {
150        for (int i = 0; i < n; i++) {
151            cout << q[i] << " ";
152        }
153        cout << endl;
154    }
155
156    void print() {
157        for (int i = 0; i < n; i++) {
158            cout << adj[i] << " ";
159        }
160        cout << endl;
161    }
162
163    void print() {
164        for (int i = 0; i < n; i++) {
165            cout << deg[i] << " ";
166        }
167        cout << endl;
168    }
169
170    void print() {
171        for (int i = 0; i < n; i++) {
172            cout << l[i] << " ";
173        }
174        cout << endl;
175    }
176
177    void print() {
178        for (int i = 0; i < n; i++) {
179            cout << r[i] << " ";
180        }
181        cout << endl;
182    }
183
184    void print() {
185        for (int i = 0; i < n; i++) {
186            cout << a[i] << " ";
187        }
188        cout << endl;
189    }
190
191    void print() {
192        for (int i = 0; i < n; i++) {
193            cout << p[i] << " ";
194        }
195        cout << endl;
196    }
197
198    void print() {
199        for (int i = 0; i < n; i++) {
200            cout << q[i] << " ";
201        }
202        cout << endl;
203    }
204
205    void print() {
206        for (int i = 0; i < n; i++) {
207            cout << adj[i] << " ";
208        }
209        cout << endl;
210    }
211
212    void print() {
213        for (int i = 0; i < n; i++) {
214            cout << deg[i] << " ";
215        }
216        cout << endl;
217    }
218
219    void print() {
220        for (int i = 0; i < n; i++) {
221            cout << l[i] << " ";
222        }
223        cout << endl;
224    }
225
226    void print() {
227        for (int i = 0; i < n; i++) {
228            cout << r[i] << " ";
229        }
230        cout << endl;
231    }
232
233    void print() {
234        for (int i = 0; i < n; i++) {
235            cout << a[i] << " ";
236        }
237        cout << endl;
238    }
239
240    void print() {
241        for (int i = 0; i < n; i++) {
242            cout << p[i] << " ";
243        }
244        cout << endl;
245    }
246
247    void print() {
248        for (int i = 0; i < n; i++) {
249            cout << q[i] << " ";
250        }
251        cout << endl;
252    }
253
254    void print() {
255        for (int i = 0; i < n; i++) {
256            cout << adj[i] << " ";
257        }
258        cout << endl;
259    }
260
261    void print() {
262        for (int i = 0; i < n; i++) {
263            cout << deg[i] << " ";
264        }
265        cout << endl;
266    }
267
268    void print() {
269        for (int i = 0; i < n; i++) {
270            cout << l[i] << " ";
271        }
272        cout << endl;
273    }
274
275    void print() {
276        for (int i = 0; i < n; i++) {
277            cout << r[i] << " ";
278        }
279        cout << endl;
280    }
281
282    void print() {
283        for (int i = 0; i < n; i++) {
284            cout << a[i] << " ";
285        }
286        cout << endl;
287    }
288
289    void print() {
290        for (int i = 0; i < n; i++) {
291            cout << p[i] << " ";
292        }
293        cout << endl;
294    }
295
296    void print() {
297        for (int i = 0; i < n; i++) {
298            cout << q[i] << " ";
299        }
300        cout << endl;
301    }
302
303    void print() {
304        for (int i = 0; i < n; i++) {
305            cout << adj[i] << " ";
306        }
307        cout << endl;
308    }
309
310    void print() {
311        for (int i = 0; i < n; i++) {
312            cout << deg[i] << " ";
313        }
314        cout << endl;
315    }
316
317    void print() {
318        for (int i = 0; i < n; i++) {
319            cout << l[i] << " ";
320        }
321        cout << endl;
322    }
323
324    void print() {
325        for (int i = 0; i < n; i++) {
326            cout << r[i] << " ";
327        }
328        cout << endl;
329    }
330
331    void print() {
332        for (int i = 0; i < n; i++) {
333            cout << a[i] << " ";
334        }
335        cout << endl;
336    }
337
338    void print() {
339        for (int i = 0; i < n; i++) {
340            cout << p[i] << " ";
341        }
342        cout << endl;
343    }
344
345    void print() {
346        for (int i = 0; i < n; i++) {
347            cout << q[i] << " ";
348        }
349        cout << endl;
350    }
351
352    void print() {
353        for (int i = 0; i < n; i++) {
354            cout << adj[i] << " ";
355        }
356        cout << endl;
357    }
358
359    void print() {
360        for (int i = 0; i < n; i++) {
361            cout << deg[i] << " ";
362        }
363        cout << endl;
364    }
365
366    void print() {
367        for (int i = 0; i < n; i++) {
368            cout << l[i] << " ";
369        }
370        cout << endl;
371    }
372
373    void print() {
374        for (int i = 0; i < n; i++) {
375            cout << r[i] << " ";
376        }
377        cout << endl;
378    }
379
380    void print() {
381        for (int i = 0; i < n; i++) {
382            cout << a[i] << " ";
383        }
384        cout << endl;
385    }
386
387    void print() {
388        for (int i = 0; i < n; i++) {
389            cout << p[i] << " ";
390        }
391        cout << endl;
392    }
393
394    void print() {
395        for (int i = 0; i < n; i++) {
396            cout << q[i] << " ";
397        }
398        cout << endl;
399    }
400
401    void print() {
402        for (int i = 0; i < n; i++) {
403            cout << adj[i] << " ";
404        }
405        cout << endl;
406    }
407
408    void print() {
409        for (int i = 0; i < n; i++) {
410            cout << deg[i] << " ";
411        }
412        cout << endl;
413    }
414
415    void print() {
416        for (int i = 0; i < n; i++) {
417            cout << l[i] << " ";
418        }
419        cout << endl;
420    }
421
422    void print() {
423        for (int i = 0; i < n; i++) {
424            cout << r[i] << " ";
425        }
426        cout << endl;
427    }
428
429    void print() {
430        for (int i = 0; i < n; i++) {
431            cout << a[i] << " ";
432        }
433        cout << endl;
434    }
435
436    void print() {
437        for (int i = 0; i < n; i++) {
438            cout << p[i] << " ";
439        }
440        cout << endl;
441    }
442
443    void print() {
444        for (int i = 0; i < n; i++) {
445            cout << q[i] << " ";
446        }
447        cout << endl;
448    }
449
450    void print() {
451        for (int i = 0; i < n; i++) {
452            cout << adj[i] << " ";
453        }
454        cout << endl;
455    }
456
457    void print() {
458        for (int i = 0; i < n; i++) {
459            cout << deg[i] << " ";
460        }
461        cout << endl;
462    }
463
464    void print() {
465        for (int i = 0; i < n; i++) {
466            cout << l[i] << " ";
467        }
468        cout << endl;
469    }
470
471    void print() {
472        for (int i = 0; i < n; i++) {
473            cout << r[i] << " ";
474        }
475        cout << endl;
476    }
477
478    void print() {
479        for (int i = 0; i < n; i++) {
480            cout << a[i] << " ";
481        }
482        cout << endl;
483    }
484
485    void print() {
486        for (int i = 0; i < n; i++) {
487            cout << p[i] << " ";
488        }
489        cout << endl;
490    }
491
492    void print() {
493        for (int i = 0; i < n; i++) {
494            cout << q[i] << " ";
495        }
496        cout << endl;
497    }
498
499    void print() {
500        for (int i = 0; i < n; i++) {
501            cout << adj[i] << " ";
502        }
503        cout << endl;
504    }
505
506    void print() {
507        for (int i = 0; i < n; i++) {
508            cout << deg[i] << " ";
509        }
510        cout << endl;
511    }
512
513    void print() {
514        for (int i = 0; i < n; i++) {
515            cout << l[i] << " ";
516        }
517        cout << endl;
518    }
519
520    void print() {
521        for (int i = 0; i < n; i++) {
522            cout << r[i] << " ";
523        }
524        cout << endl;
525    }
526
527    void print() {
528        for (int i = 0; i < n; i++) {
529            cout << a[i] << " ";
530        }
531        cout << endl;
532    }
533
534    void print() {
535        for (int i = 0; i < n; i++) {
536            cout << p[i] << " ";
537        }
538        cout << endl;
539    }
540
541    void print() {
542        for (int i = 0; i < n; i++) {
543            cout << q[i] << " ";
544        }
545        cout << endl;
546    }
547
548    void print() {
549        for (int i = 0; i < n; i++) {
550            cout << adj[i] << " ";
551        }
552        cout << endl;
553    }
554
555    void print() {
556        for (int i = 0; i < n; i++) {
557            cout << deg[i] << " ";
558        }
559        cout << endl;
560    }
561
562    void print() {
563        for (int i = 0; i < n; i++) {
564            cout << l[i] << " ";
565        }
566        cout << endl;
567    }
568
569    void print() {
570        for (int i = 0; i < n; i++) {
571            cout << r[i] << " ";
572        }
573        cout << endl;
574    }
575
576    void print() {
577        for (int i = 0; i < n; i++) {
578            cout << a[i] << " ";
579        }
580        cout << endl;
581    }
582
583    void print() {
584        for (int i = 0; i < n; i++) {
585            cout << p[i] << " ";
586        }
587        cout << endl;
588    }
589
590    void print() {
591        for (int i = 0; i < n; i++) {
592            cout << q[i] << " ";
593        }
594        cout << endl;
595    }
596
597    void print() {
598        for (int i = 0; i < n; i++) {
599            cout << adj[i] << " ";
600        }
601        cout << endl;
602    }
603
604    void print() {
605        for (int i = 0; i < n; i++) {
606            cout << deg[i] << " ";
607        }
608        cout << endl;
609    }
610
611    void print() {
612        for (int i = 0; i < n; i++) {
613            cout << l[i] << " ";
614        }
615        cout << endl;
616    }
617
618    void print() {
619        for (int i = 0; i < n; i++) {
620            cout << r[i] << " ";
621        }
622        cout << endl;
623    }
624
625    void print() {
626        for (int i = 0; i < n; i++) {
627            cout << a[i] << " ";
628        }
629        cout << endl;
630    }
631
632    void print() {
633        for (int i = 0; i < n; i++) {
634            cout << p[i] << " ";
635        }
636        cout << endl;
637    }
638
639    void print() {
640        for (int i = 0; i < n; i++) {
641            cout << q[i] << " ";
642        }
643        cout << endl;
644    }
645
646    void print() {
647        for (int i = 0; i < n; i++) {
648            cout << adj[i] << " ";
649        }
650        cout << endl;
651    }
652
653    void print() {
654        for (int i = 0; i < n; i++) {
655            cout << deg[i] << " ";
656        }
657        cout << endl;
658    }
659
660    void print() {
661        for (int i = 0; i < n; i++) {
662            cout << l[i] << " ";
663        }
664        cout << endl;
665    }
666
667    void print() {
668        for (int i = 0; i < n; i++) {
669            cout << r[i] << " ";
670        }
671        cout << endl;
672    }
673
674    void print() {
675        for (int i = 0; i < n; i++) {
676            cout << a[i] << " ";
677        }
678        cout << endl;
679    }
680
681    void print() {
682        for (int i = 0; i < n; i++) {
683            cout << p[i] << " ";
684        }
685        cout << endl;
686    }
687
688    void print() {
689        for (int i = 0; i < n; i++) {
690            cout << q[i] << " ";
691        }
692        cout << endl;
693    }
694
695    void print() {
696        for (int i = 0; i < n; i++) {
697            cout << adj[i] << " ";
698        }
699        cout << endl;
700    }
701
702    void print() {
703        for (int i = 0; i < n; i++) {
704            cout << deg[i] << " ";
705        }
706        cout << endl;
707    }
708
709    void print() {
710        for (int i = 0; i < n; i++) {
711            cout << l[i] << " ";
712        }
713        cout << endl;
714    }
715
716    void print() {
717        for (int i = 0; i < n; i++) {
718            cout << r[i] << " ";
719        }
720        cout << endl;
721    }
722
723    void print() {
724        for (int i = 0; i < n; i++) {
725            cout << a[i] << " ";
726        }
727        cout << endl;
728    }
729
730    void print() {
731        for (int i = 0; i < n; i++) {
732            cout << p[i] << " ";
733        }
734        cout << endl;
735    }
736
737    void print() {
738        for (int i = 0; i < n; i++) {
739            cout << q[i] << " ";
740        }
741        cout << endl;
742    }
743
744    void print() {
745        for (int i = 0; i < n; i++) {
746            cout << adj[i] << " ";
747        }
748        cout << endl;
749    }
750
751    void print() {
752        for (int i = 0; i < n; i++) {
753            cout << deg[i] << " ";
754        }
755        cout << endl;
756    }
757
758    void print() {
759        for (int i = 0; i < n; i++) {
760            cout << l[i] << " ";
761        }
762        cout << endl;
763    }
764
765    void print() {
766        for (int i = 0; i < n; i++) {
767            cout << r[i] << " ";
768        }
769        cout << endl;
770    }
771
772    void print() {
773        for (int i = 0; i < n; i++) {
774            cout << a[i] << " ";
775        }
776        cout << endl;
777    }
778
779    void print() {
780        for (int i = 0; i < n; i++) {
781            cout << p[i] << " ";
782        }
783        cout << endl;
784    }
785
786    void print() {
787        for (int i = 0; i < n; i++) {
788            cout << q[i] << " ";
789        }
790        cout << endl;
791    }
792
793    void print() {
794        for (int i = 0; i < n; i++) {
795            cout << adj[i] << " ";
796        }
797        cout << endl;
798    }
799
800    void print() {
801        for (int i = 0; i < n; i++) {
802            cout << deg[i] << " ";
803        }
804        cout << endl;
805    }
806
807    void print() {
808        for (int i = 0; i < n; i++) {
809            cout << l[i] << " ";
810        }
811        cout << endl;
812    }
813
814    void print() {
815        for (int i = 0; i < n; i++) {
816            cout << r[i] << " ";
817        }
818        cout << endl;
819    }
820
821    void print() {
822        for (int i = 0; i < n; i++) {
823            cout << a[i] << " ";
824        }
825        cout << endl;
826    }
827
828    void print() {
829        for (int i = 0; i < n; i++) {
830            cout << p[i] << " ";
831        }
832        cout << endl;
833    }
834
835    void print() {
836        for (int i = 0; i < n; i++) {
837            cout << q[i] << " ";
838        }
839        cout << endl;
840    }
841
842    void print() {
843        for (int i = 0; i < n; i++) {
844            cout << adj[i] << " ";
845        }
846        cout << endl;
847    }
848
849    void print() {
850        for (int i = 0; i < n; i++) {
851            cout << deg[i] << " ";
852        }
853        cout << endl;
854    }
855
856    void print() {
857        for (int i = 0; i < n; i++) {
858            cout << l[i] << " ";
859        }
860        cout << endl;
861    }
862
863    void print() {
864        for (int i = 0; i < n; i++) {
865            cout << r[i] << " ";
866        }
867        cout << endl;
868    }
869
870    void print() {
871        for (int i = 0; i < n; i++) {
872            cout << a[i] << " ";
873        }
874        cout << endl;
875    }
876
877    void print() {
878        for (int i = 0; i < n; i++) {
879            cout << p[i] << " ";
880        }
881        cout << endl;
882    }
883
884    void print() {
885        for (int i = 0; i < n; i++) {
886            cout << q[i] << " ";
887        }
888        cout << endl;
889    }
890
891    void print() {
892        for (int i = 0; i < n; i++) {
893            cout << adj[i] << " ";
894        }
895        cout << endl;
896    }
897
898    void print() {
899        for (int i = 0; i < n; i++) {
900            cout << deg[i] << " ";
901        }
902        cout << endl;
903    }
904
905    void print() {
906        for (int i = 0; i < n; i++) {
907            cout << l[i] << " ";
908        }
909        cout << endl;
910    }
911
912    void print() {
913        for (int i = 0; i < n; i++) {
914            cout << r[i] << " ";
915        }
916        cout << endl;
917    }
918
919    void print() {
920        for (int i = 0; i < n; i++) {
921            cout << a[i] << " ";
922        }
923        cout << endl;
924    }
925
926    void print() {
927        for (int i = 0; i < n; i++) {
928            cout << p[i] << " ";
929        }
930        cout << endl;
931    }
932
933    void print() {
934        for (int i = 0; i < n; i++) {
935            cout << q[i] << " ";
936        }
937        cout << endl;
938    }
939
940    void print() {
941        for (int i = 0; i < n; i++) {
942            cout << adj[i] << " ";
943        }
944        cout << endl;
945    }
946
947    void print() {
948        for (int i = 0; i < n; i++) {
949            cout << deg[i] << " ";
950        }
951        cout << endl;
952    }
953
954    void print() {
955        for (int i = 0; i < n; i++) {
956            cout << l[i] << " ";
957        }
958        cout << endl;
959    }
960
961    void print() {
962        for (int i = 0; i < n; i++) {
963            cout << r[i] << " ";
964        }
965        cout << endl;
966    }
967
968    void print() {
969        for (int i = 0; i < n; i++) {
970            cout << a[i] << " ";
971        }
972        cout << endl;
973    }
974
975    void print() {
976        for (int i = 0; i < n; i++) {
977            cout << p[i] << " ";
978        }
979        cout << endl;
980    }
981
982    void print() {
983        for (int i = 0; i < n; i++) {
984            cout << q[i] << " ";
985        }
986        cout << endl;
987    }
988
989    void print() {
990        for (int i = 0; i < n; i++) {
991            cout << adj[i] << " ";
992        }
993        cout << endl;
994    }
995
996    void print() {
997        for (int i = 0; i < n; i++) {
998            cout << deg[i] << " ";
999        }
1000        cout << endl;
1001    }
1002
1003    void print() {
1004        for (int i = 0; i < n; i++) {
1005            cout << l[i] << " ";
1006        }
1007        cout << endl;
1008    }
1009
1010    void print() {
1011        for (int i = 0; i < n; i++) {
1012            cout << r[i] << " ";
1013        }
1014        cout << endl;
1015    }
1016
1
```

```

41     }
42
43     bool match = false;
44     for (int i = 0; i < t; i++) {
45         int x = q[i];
46         if (~l[a[x]]) continue;
47
48         for (int j = deg[x]; j < deg[x + 1]; j++) {
49             int y = adj[j];
50             if (r[y] == -1) {
51                 while (~y) {
52                     r[y] = x;
53                     swap(l[x], y);
54                     x = p[x];
55                 }
56                 match = true;
57                 ++ans;
58                 break;
59             }
60             if (p[r[y]] == -1) {
61                 q[t++] = y = r[y];
62                 p[y] = x;
63                 a[y] = a[x];
64             }
65         }
66     }
67     if (!match) break;
68 }
69 return ans;
70 }
71 vector<array<int, 2>> answer() {
72     vector<array<int, 2>> ans;
73     for (int i = 0; i < n; i++) {
74         if (~l[i]) {
75             ans.push_back({i, l[i]});
76         }
77     }
78     return ans;
79 }
80 };
81
82 signed main() {
83     int n1, n2, m;
84     cin >> n1 >> n2 >> m;
85     HopcroftKarp flow(n1, n2);
86     while (m--) {
87         int x, y;
88         cin >> x >> y;
89         flow.add(x, y);
90     }
91
92     cout << flow.work() << "\n";
93
94     auto match = flow.answer();
95     for (auto [u, v] : match) {
96         cout << u << " " << v << "\n";
97     }
98 }
```

3.12 二分图最大权匹配（二分图完美匹配）

定义：找到边权和最大的那个匹配。

一般我们规定，左半部包含 n_1 个点（编号 $1 - n_1$ ），右半部包含 n_2 个点（编号 $1 - n_2$ ）。

使用匈牙利算法（KM算法）解，时间复杂度为 $\mathcal{O}(N^3)$ 。下方模板用于求解最大权值、且可以输出其中一种可行方案，例题为 [UOJ #80. 二分图最大权匹配](#)。

```

1 struct MaxCostMatch {
2     vector<int> ansL, ansR, pre;
3     vector<int> lx, ly;
4     vector<vector<int>> ver;
5     int n;
6
7     MaxCostMatch(int n) : n(n) {
8         ver.resize(n + 1, vector<int>(n + 1));
9         ansL.resize(n + 1, -1);
10        ansR.resize(n + 1, -1);
11        lx.resize(n + 1);
12        ly.resize(n + 1, -1E18);
13        pre.resize(n + 1);
14    }
15    void add(int x, int y, int w) {
16        ver[x][y] = w;
17    }
18    void bfs(int x) {
19        vector<bool> visL(n + 1), visR(n + 1);
20        vector<int> slack(n + 1, 1E18);
21        queue<int> q;
22        function<bool(int)> check = [&](int x) {
23            visR[x] = 1;
24            if (~ansR[x]) {
25                q.push(ansR[x]);
26                visL[ansR[x]] = 1;
27                return false;
28            }
29            while (~x) {
30                ansR[x] = pre[x];
31                swap(x, ansL[pre[x]]);
32            }
33            return true;
34        };
35        q.push(x);
36        visL[x] = 1;
37        while (1) {
38            while (!q.empty()) {
39                int x = q.front();
40                q.pop();
41                for (int y = 1; y <= n; ++y) {
42                    if (visR[y]) continue;
43                    int del = lx[x] + ly[y] - ver[x][y];
44                    if (del < slack[y]) {
45                        pre[y] = x;
46                        slack[y] = del;
47                        if (!slack[y] && check(y)) return;
48                    }
49                }
50            }
51            int val = 1E18;
52            for (int i = 1; i <= n; ++i) {
53                if (!visR[i]) {
54                    val = min(val, slack[i]);

```

```

55         }
56     }
57     for (int i = 1; i <= n; ++i) {
58         if (visl[i]) lx[i] -= val;
59         if (visr[i]) {
60             ly[i] += val;
61         } else {
62             slack[i] -= val;
63         }
64     }
65     for (int i = 1; i <= n; ++i) {
66         if (!visr[i] && !slack[i] && check(i)) {
67             return;
68         }
69     }
70 }
71 }
72 int work() {
73     for (int i = 1; i <= n; ++i) {
74         for (int j = 1; j <= n; ++j) {
75             ly[i] = max(ly[i], ver[j][i]);
76         }
77     }
78     for (int i = 1; i <= n; ++i) bfs(i);
79     int res = 0;
80     for (int i = 1; i <= n; ++i) {
81         res += ver[i][ansl[i]];
82     }
83     return res;
84 }
85 void getMatch(int x, int y) { // 获取方案 (0代表无匹配)
86     for (int i = 1; i <= x; ++i) {
87         cout << (ver[i][ansl[i]] ? ansl[i] : 0) << " ";
88     }
89     cout << endl;
90     for (int i = 1; i <= y; ++i) {
91         cout << (ver[i][ansr[i]] ? ansr[i] : 0) << " ";
92     }
93     cout << endl;
94 }
95 };
96
97 signed main() {
98     int n1, n2, m;
99     cin >> n1 >> n2 >> m;
100
101     MaxCostMatch match(max(n1, n2));
102     for (int i = 1; i <= m; i++) {
103         int x, y, w;
104         cin >> x >> y >> w;
105         match.add(x, y, w);
106     }
107     cout << match.work() << '\n';
108     match.getMatch(n1, n2);
109 }

```

3.13 二分图最大独立点集 (Konig 定理)

给出一张二分图，要求选择一些点使得它们两两没有边直接连接。最小点覆盖等价于最大匹配数，转换为最小割模板，答案即为总点数减去最大流得到的值。

```
1 | cout << n - flow.work(s, t) << endl;
```

3.14 最长路 (topsort+DP算法)

计算一张 DAG 中的最长路径，在执行前可能需要使用 tarjan 重构一张正确的 DAG，复杂度 $\mathcal{O}(N + M)$ 。

```

1 struct DAG {
2     int n;
3     vector<vector<pair<int, int>>> ver;
4     vector<int> deg, dis;
5     DAG(int n) : n(n) {
6         ver.resize(n + 1);
7         deg.resize(n + 1);
8         dis.assign(n + 1, -1E18);
9     }
10    void add(int x, int y, int w) {
11        ver[x].push_back({y, w});
12        ++deg[y];
13    }
14    int topsort(int s, int t) {
15        queue<int> q;
16        for (int i = 1; i <= n; i++) {
17            if (deg[i] == 0) {
18                q.push(i);
19            }
20        }
21        dis[s] = 0;
22        while (!q.empty()) {
23            int x = q.front();
24            q.pop();
25            for (auto [y, w] : ver[x]) {
26                dis[y] = max(dis[y], dis[x] + w);
27                --deg[y];
28                if (deg[y] == 0) {
29                    q.push(y);
30                }
31            }
32        }
33        return dis[t];
34    }
35 };
36
37 signed main() {
38     int n, m;
39     cin >> n >> m;
40     DAG dag(n);
41     for (int i = 1; i <= m; i++) {
42         int x, y, w;
43         cin >> x >> y >> w;
44         dag.add(x, y, w);
45     }
46
47     int s, t;
48     cin >> s >> t;
49     cout << dag.topsort(s, t) << "\n";
50 }
```

3.15 最短路径树 (SPT问题)

定义：在一张无向带权联通图中，有这样一棵**生成树**：满足从根节点到任意点的路径都为原图中根到任意点的最短路径。

性质：记根节点 $Root$ 到某一结点 x 的最短距离 $dis_{Root,x}$ ，在 SPT 上这两点之间的距离为 $len_{Root,x}$ ——则两者长度相等。

该算法与最小生成树无关，基于最短路 Djikstra 算法完成（但多了个等于号）。下方代码实现的功能为：读入图后，输出以 1 为根的 SPT 所使用的各条边的编号、边权和。

```

1 map<pair<int, int>, int> id;
2 namespace G {
3     vector<pair<int, int>> ver[N];
4     map<pair<int, int>, int> edge;
5     int v[N], d[N], pre[N], vis[N];
6     int ans = 0;
7
8     void add(int x, int y, int w) {
9         ver[x].push_back({y, w});
10        edge[{x, y}] = edge[{y, x}] = w;
11    }
12    void djikstra(int s) { // !注意，该 djikstra 并非原版，多加了一个等于号
13        priority_queue<PII, vector<PII>, greater<PII>> q; q.push({0, s});
14        memset(d, 0x3f, sizeof d); d[s] = 0;
15        while (!q.empty()) {
16            int x = q.top().second; q.pop();
17            if (v[x]) continue; v[x] = 1;
18            for (auto [y, w] : ver[x]) {
19                if (d[y] >= d[x] + w) { // !注意，SPT 这里修改为>=号
20                    d[y] = d[x] + w;
21                    pre[y] = x; // 记录前驱结点
22                    q.push({d[y], y});
23                }
24            }
25        }
26    }
27    void dfs(int x) {
28        vis[x] = 1;
29        for (auto [y, w] : ver[x]) {
30            if (vis[y]) continue;
31            if (pre[y] == x) {
32                cout << id[{x, y}] << " "; // 输出SPT所使用的边编号
33                ans += edge[{x, y}];
34                dfs(y);
35            }
36        }
37    }
38    void solve(int n) {
39        djikstra(1); // 以 1 为根
40        dfs(1); // 以 1 为根
41        cout << endl << ans; // 输出SPT的边权和
42    }
43}
44 bool Solve() {
45     int n, m; cin >> n >> m;
46     for (int i = 1; i <= m; ++ i) {
47         int x, y, w; cin >> x >> y >> w;
48         G::add(x, y, w), G::add(y, x, w);
49         id[{x, y}] = id[{y, x}] = i;
50     }
51     G::solve(n);

```

```

52     return 0;
53 }
```

3.16 无源汇点的最小割问题 Stoer-Wagner

也称为全局最小割。定义补充（与《网络流》中的定义不同）：

割：是一个边集，去掉其中所有边能使一张网络流图不再连通（即分成两个子图）。

通过递归的方式来解决无向正权图上的全局最小割问题，算法复杂度 $\mathcal{O}(VE + V^2 \log V)$ ，一般可近似看作 $\mathcal{O}(V^3)$ 。

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
4
5     DSU dsu(n); // 这里引入DSU判断图是否联通，如题目有保证，则不需要此步骤
6     vector<vector<int>> edge(n + 1, vector<int>(n + 1));
7     for (int i = 1; i <= m; i++) {
8         int x, y, w;
9         cin >> x >> y >> w;
10        dsu.merge(x, y);
11        edge[x][y] += w;
12        edge[y][x] += w;
13    }
14
15    if (dsu.Poi(1) != n || m < n - 1) { // 图不联通
16        cout << 0 << endl;
17        return 0;
18    }
19
20    int MinCut = INF, S = 1, T = 1; // 虚拟源汇点
21    vector<int> bin(n + 1);
22    auto contract = [&]() -> int { // 求解S到T的最小割，定义为 cut of phase
23        vector<int> dis(n + 1), vis(n + 1);
24        int Min = 0;
25        for (int i = 1; i <= n; i++) {
26            int k = -1, maxc = -1;
27            for (int j = 1; j <= n; j++) {
28                if (!bin[j] && !vis[j] && dis[j] > maxc) {
29                    k = j;
30                    maxc = dis[j];
31                }
32            }
33            if (k == -1) return Min;
34            S = T, T = k, Min = maxc;
35            vis[k] = 1;
36            for (int j = 1; j <= n; j++) {
37                if (!bin[j] && !vis[j]) {
38                    dis[j] += edge[k][j];
39                }
40            }
41        }
42        return Min;
43    };
44    for (int i = 1; i < n; i++) { // 这里取不到等号
45        int val = contract();
46        bin[T] = 1;
47        MinCut = min(MinCut, val);
48        if (!MinCut) {
49            cout << 0 << endl;
50            return 0;
51        }
52    }
53 }
```

```

51     }
52     for (int j = 1; j <= n; j++) {
53         if (!bin[j]) {
54             edge[S][j] += edge[j][T];
55             edge[j][S] += edge[j][T];
56         }
57     }
58 }
59 cout << MinCut << endl;
60 }
```

3.17 欧拉路径/欧拉回路 Hierholzers

欧拉路径：一笔画完图中全部边，画的顺序就是一个可行解；当起点终点相同时称欧拉回路。

3.17.1 有向图欧拉路径存在判定

有向图欧拉路径存在：¹ 恰有一个点出度比入度多 1（为起点）；² 恰有一个点入度比出度多 1（为终点）；³ 恰有 $N - 2$ 个点入度均等于出度。如果是欧拉回路，则上方起点与终点的条件不存在，全部点均要满足最后一个条件。

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
4
5     DSU dsu(n + 1); // 如果保证连通，则不需要 DSU
6     vector<unordered_multiset<int>> ver(n + 1); // 如果对于字典序有要求，则不能使用
7     unordered
8     vector<int> degI(n + 1), degO(n + 1);
9     for (int i = 1; i <= m; i++) {
10         int x, y;
11         cin >> x >> y;
12         ver[x].insert(y);
13         degI[y]++;
14         degO[x]++;
15         dsu.merge(x, y); // 直接当无向图
16     }
17     int s = 1, t = 1, cnt = 0;
18     for (int i = 1; i <= n; i++) {
19         if (degI[i] == degO[i])
20             cnt++;
21         } else if (degI[i] + 1 == degO[i]) {
22             s = i;
23         } else if (degI[i] == degO[i] + 1) {
24             t = i;
25         }
26     if (dsu.size(1) != n || (cnt != n - 2 && cnt != n)) {
27         cout << "No\n";
28     } else {
29         cout << "Yes\n";
30     }
31 }
```

3.17.2 无向图欧拉路径存在判定

无向图欧拉路径存在：¹ 恰有两个点度数为奇数（为起点与终点）；² 恰有 $N - 2$ 个点度数为偶数。

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
```

```

4     DSU dsu(n + 1); // 如果保证连通，则不需要 DSU
5     vector<unordered_multiset<int>> ver(n + 1); // 如果对于字典序有要求，则不能使用
6     unordered
7     vector<int> deg(n + 1);
8     for (int i = 1; i <= m; i++) {
9         int x, y;
10        cin >> x >> y;
11        ver[x].insert(y);
12        ver[y].insert(x);
13        deg[y]++;
14        deg[x]++;
15        dsu.merge(x, y); // 直接当无向图
16    }
17    int s = -1, t = -1, cnt = 0;
18    for (int i = 1; i <= n; i++) {
19        if (deg[i] % 2 == 0) {
20            cnt++;
21        } else if (s == -1) {
22            s = i;
23        } else {
24            t = i;
25        }
26    }
27    if (dsu.size(1) != n || (cnt != n - 2 && cnt != n)) {
28        cout << "No\n";
29    } else {
30        cout << "Yes\n";
31    }
32 }
```

3.17.3 有向图欧拉路径求解（字典序最小）

```

1 vector<int> ans;
2 auto dfs = [&](auto self, int x) -> void {
3     while (ver[x].size()) {
4         int net = *ver[x].begin();
5         ver[x].erase(ver[x].begin());
6         self(self, net);
7     }
8     ans.push_back(x);
9 };
10 dfs(dfs, s);
11 reverse(ans.begin(), ans.end());
12 for (auto it : ans) {
13     cout << it << " ";
14 }
```

3.17.4 无向图欧拉路径求解

```

1 auto dfs = [&](auto self, int x) -> void {
2     while (ver[x].size()) {
3         int net = *ver[x].begin();
4         ver[x].erase(ver[x].find(net));
5         ver[net].erase(ver[net].find(x));
6         cout << x << " " << net << endl;
7         self(self, net);
8     }
9 };
10 dfs(dfs, s);
```

3.18 差分约束

给出一组包含 m 个不等式，有 n 个未知数的形如： $\begin{cases} u_1 - v_1 \leq w_1 \\ u_2 - v_2 \leq w_2 \\ \dots \\ u_m - v_m \leq w_m \end{cases}$ 的不等式组，求任意一组满足这个不等式组的解。SPFA 解， $\mathcal{O}(nm)$ 。[参考](#)

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
4
5     vector<array<int, 3>> e(m + 1);
6     for (int i = 1; i <= m; i++) {
7         int u, v, w;
8         cin >> u >> v >> w;
9         e[i] = {v, u, w};
10    }
11
12    vector<int> d(n + 1, 1E9);
13    d[1] = 0;
14    for (int i = 1; i < n; i++) {
15        for (int j = 1; j <= m; j++) {
16            auto [u, v, w] = e[j];
17            d[v] = min(d[v], d[u] + w);
18        }
19    }
20    for (int i = 1; i <= m; i++) {
21        auto [u, v, w] = e[i];
22        if (d[v] > d[u] + w) {
23            cout << "NO\n";
24            return 0;
25        }
26    }
27    for (int i = 1; i <= n; i++) {
28        cout << d[i] << " \n"[i == n];
29    }
30    return 0;
31 }
```

3.19 2-Sat

3.19.1 基础封装

基于 tarjan 缩点，时间复杂度为 $\mathcal{O}(N + M)$ 。注意下标从 0 开始，答案输出为字典序最小的一个可行解。

```

1 struct TwoSat {
2     int n;
3     vector<vector<int>> e;
4     vector<bool> ans;
5     TwoSat(int n) : n(n), e(2 * n), ans(n) {}
6     void add(int u, bool f, int v, bool g) {
7         e[2 * u + !f].push_back(2 * v + g);
8         e[2 * v + !g].push_back(2 * u + f);
9     }
10    bool work() {
11        vector<int> id(2 * n, -1), dfn(2 * n, -1), low(2 * n, -1);
12        vector<int> stk;
13        int now = 0, cnt = 0;
```

```

14     auto tarjan = [&](auto self, int u) -> void {
15         stk.push_back(u);
16         dfn[u] = low[u] = now++;
17         for (auto v : e[u]) {
18             if (dfn[v] == -1) {
19                 self(self, v);
20                 low[u] = min(low[u], low[v]);
21             } else if (id[v] == -1) {
22                 low[u] = min(low[u], dfn[v]);
23             }
24         }
25         if (dfn[u] == low[u]) {
26             int v;
27             do {
28                 v = stk.back();
29                 stk.pop_back();
30                 id[v] = cnt;
31             } while (v != u);
32             ++cnt;
33         }
34     };
35     for (int i = 0; i < 2 * n; ++i) {
36         if (dfn[i] == -1) {
37             tarjan(tarjan, i);
38         }
39     }
40     for (int i = 0; i < n; ++i) {
41         if (id[2 * i] == id[2 * i + 1]) return false;
42         ans[i] = id[2 * i] > id[2 * i + 1];
43     }
44     return true;
45 }
46 vector<bool> answer() {
47     return ans;
48 }
49 };

```

3.19.2 答案不唯一时不输出

在运行后针对每一个点进行一次 dfs，时间复杂度为 $\mathcal{O}(N^2)$ ，当且仅当答案唯一时才输出，否则输出 ? 替代。

```

1 // 结构体中增加
2 int check(int x, int y) {
3     vector<int> vis(2 * n);
4     auto dfs = [&](auto self, int x) -> void {
5         vis[x] = 1;
6         for (auto y : e[x]) {
7             if (vis[y]) continue;
8             self(self, y);
9         }
10    };
11    dfs(dfs, x);
12    return vis[y];
13 }
14 // 主函数中增加
15 for (int i = 0; i < n; i++) {
16     if (sat.check(2 * i, 2 * i + 1)) {
17         cout << 1 << " ";
18     } else if (sat.check(2 * i + 1, 2 * i)) {
19         cout << 0 << " ";
20     } else {

```

```

21     cout << "?" << " ";
22 }
23 }
```

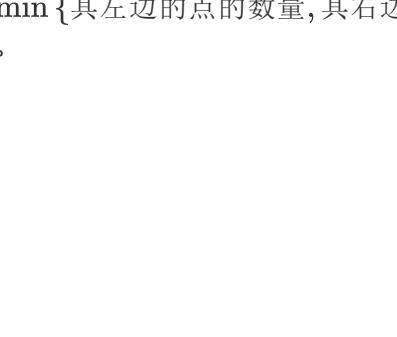
3.20 常见结论

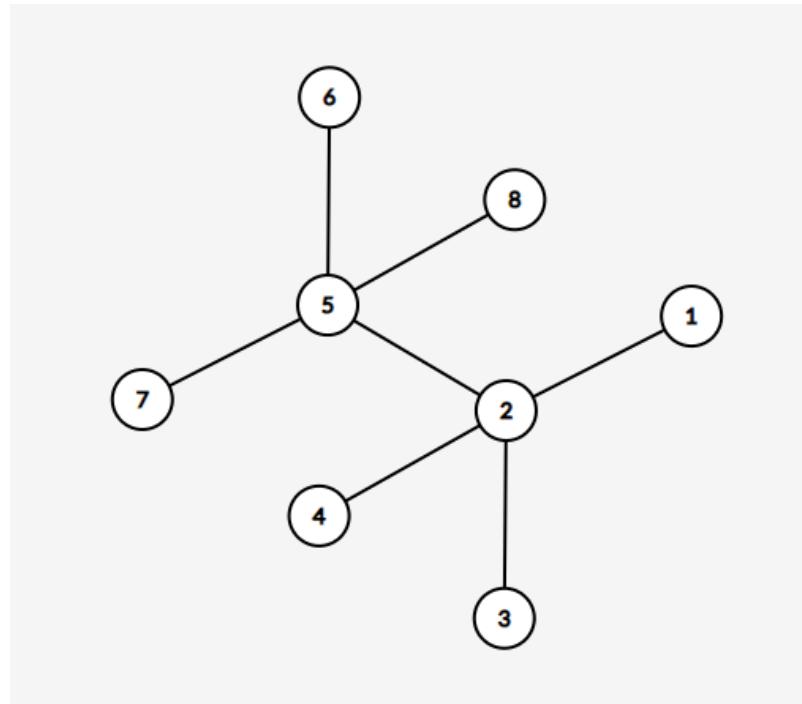
1. 要在有向图上求一个最大点集，使得任意两个点 (i, j) 之间至少存在一条路径（可以是从 i 到 j ，也可以反过来，这两种有一个就行），即求解最长路；
2. 要求出连通图上的任意一棵生成树，只需要跑一遍 **bfs**；
3. 给出一棵树，要求添加尽可能多的边，使得其是二分图：对树进行二分染色，显然，相同颜色的点之间连边不会破坏二分图的性质，故可添加的最多的边数即为 $cnt_{Black} * cnt_{White} - (n - 1)$ ；
4. 当一棵树可以被黑白染色时，所有染黑节点的度之和等于所有染白节点的度之和；
5. 在竞赛图中，入度小的点，必定能到达出度小（入度大）的点 [See](#)。
6. 在竞赛图中，将所有点按入度从小到大排序，随后依次遍历，若对于某一点 i 满足前 i 个点的入度之和恰好等于 $\left\lfloor \frac{n \cdot (n + 1)}{2} \right\rfloor$ ，那么对于上一次满足这一条件的点 p ， $p + 1$ 到 i 点构成一个新的强连通分量 [See](#)。
 举例说明，设满足上方条件的点为 p_1, p_2 ($p_1 + 1 < p_2$)，那么点 1 到 p_1 构成一个强连通分量、点 $p_1 + 1$ 到 p_2 构成一个强连通分量。
7. 选择图中最少数量的边删除，使得图不连通，即求最小割；如果是删除点，那么拆点后求最小割 [See](#)。
8. 如果一张图是平面图，那么其边数一定小于等于 $3n - 6$ [See](#)。
9. 若一张有向完全图存在环，则一定存在三元环。
10. 竞赛图三元环计数：[See](#)。
11. 有向图判是否存在环直接用 topsort；无向图判是否存在环直接用 dsu，也可以使用 topsort，条件变为 $deg[i] \leq 1$ 时入队。

3.21 常见例题

3.21.1 杂

题意：给出一棵节点数为 $2n$ 的树，要求将点分割为 n 个点对，使得点对的点之间的距离和最大。

可以转化为边上问题：对于每一条边，其被利用的次数即为 $\min\{\text{其左边的点的数量, 其右边的点的数量}\}$ ，使用树形 dp 计算一遍即可。如下图样例，答案为 10。




```

1 vector<int> val(n + 1, 1);
2 int ans = 0;
3 function<void(int, int)> dfs = [&](int x, int fa) {
4     for (auto y : ver[x]) {
5         if (y == fa) continue;
6         dfs(y, x);
7         val[x] += val[y];
8         ans += min(val[y], k - val[y]);
9     }
10 };
11 dfs(1, 0);
12 cout << ans << endl;

```

题意：以哪些点为起点可以无限的在有向图上绕

概括一下这些点可以发现，一类是环上的点，另一类是可以到达环的点。建反图跑一遍 topsort 板子，根据容斥，未被移除的点都是答案 [See](#)。

题意：添加最少的边，使得有向图变成一个 SCC

将原图的 SCC 缩点，统计缩点后的新图上入度为 0 和出度为 0 的点的数量 cnt_{in} 、 cnt_{out} ，答案即为 $\max(cnt_{in}, cnt_{out})$ 。过程大致是先将一个出度为 0 的点和一个入度为 0 的点相连，剩下的点随便连 [See](#)。

题意：添加最少的边，使得无向图变成一个 E-DCC

将原图的 E-DCC 缩点，统计缩点后的新图上入度为 1 的点（叶子结点）的数量 cnt ，答案即为 $\lceil \frac{cnt}{2} \rceil$ 。过程大致是每次找两个叶子结点（但是还有一些条件限制）相连，若最后余下一个点随便连 [See](#)。

题意：在树上找到一个最大的连通块，使得这个联通内点权和边权之和最大，输出这个值，数据中存在负数的情况。

使用 dfs 即可解决。

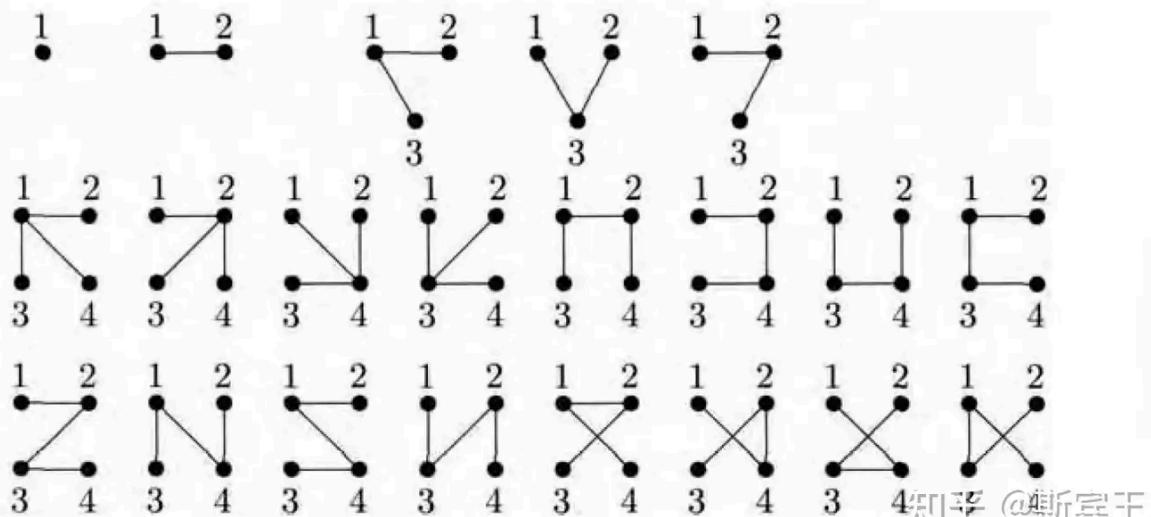
```

1 LL n, point[N];
2 LL ver[N], head[N], nex[N], tot; bool v[N];
3 map<pair<LL, LL>, LL> edge;
4 // void add(LL x, LL y) {}
5 void dfs(LL x) {
6     for (LL i = head[x]; i; i = nex[i]) {
7         LL y = ver[i];
8         if (v[y]) continue;
9         v[y] = true; dfs(y); v[y] = false;
10    }
11    for (LL i = head[x]; i; i = nex[i]) {
12        LL y = ver[i];
13        if (v[y]) continue;
14        point[x] += max(point[y] + edge[{x, y}], 0LL);
15    }
16}
17 void Solve() {
18     cin >> n;
19     FOR(i, 1, n) cin >> point[i];
20     FOR(i, 2, n) {
21         LL x, y, w; cin >> x >> y >> w;
22         edge[{x, y}] = edge[{y, x}] = w;
23         add(x, y), add(y, x);
24     }
25     v[1] = true; dfs(1); LL ans = -MAX18;
26     FOR(i, 1, n) ans = max(ans, point[i]);
27     cout << ans << endl;
28 }

```

3.21.2 Prüfer 序列：凯莱公式

题意：给定 n 个顶点，可以构建出多少棵标记树？



$n \leq 4$ 时的样例如上，通项公式为 n^{n-2} 。

3.21.3 Prüfer 序列

一个 n 个点 m 条边的带标号无向图有 k 个连通块。我们希望添加 $k - 1$ 条边使得整个图连通，求方案数量 [See](#)。

设 s_i 表示每个连通块的数量，通项公式为 $n^{k-2} \cdot \prod_{i=1}^k s_i$ ，当 $k < 2$ 时答案为 1。

3.21.4 单源最短/次短路计数

```

1 const int N = 2e5 + 7, M = 1e6 + 7;
2 int n, m, s, e; int d[N][2], v[N][2]; // 0 代表最短路， 1 代表次短路
3 Z num[N][2];
4
5 void Clear() {
6     for (int i = 1; i <= n; ++ i) h[i] = edge[i] = 0;
7     tot = 0;
8     for (int i = 1; i <= n; ++ i) num[i][0] = num[i][1] = v[i][0] = v[i][1] = 0;
9     for (int i = 1; i <= n; ++ i) d[i][0] = d[i][1] = INF;
10 }
11
12 int ver[M], ne[M], h[N], edge[M], tot;
13 void add(int x, int y, int w) {
14     ver[++ tot] = y, ne[tot] = h[x], h[x] = tot;
15     edge[tot] = w;
16 }
17
18 void dji() {
19     priority_queue<PIII, vector<PIII>, greater<PIII> > q; q.push({0, s, 0});
20     num[s][0] = 1; d[s][0] = 0;
21     while (!q.empty()) {
22         auto [dis, x, type] = q.top(); q.pop();
23         if (v[x][type]) continue; v[x][type] = 1;
24         for (int i = h[x]; i; i = ne[i]) {
25             int y = ver[i], w = dis + edge[i];
26             if (d[y][0] > w) {
27                 d[y][1] = d[y][0], num[y][1] = num[y][0];
28                 // 如果找到新的最短路，将原有的最短路数据转化为次短路
29                 q.push({d[y][1], y, 1});
30                 d[y][0] = w, num[y][0] = num[x][type];
31                 q.push({d[y][0], y, 0});
32             }
33             else if (d[y][0] == w) num[y][0] += num[x][type];
34             else if (d[y][1] > w) {
35                 d[y][1] = w, num[y][1] = num[x][type];
36                 q.push({d[y][1], y, 1});
37             }
38             else if (d[y][1] == w) num[y][1] += num[x][type];
39         }
40     }
41 }
42 void Solve() {
43     cin >> n >> m >> s >> e;
44     Clear(); // 多组样例务必完全清空
45     for (int i = 1; i <= m; ++ i) {
46         int x, y, w; cin >> x >> y; w = 1;
47         add(x, y, w), add(y, x, w);
48     }
49     dji();
50     Z ans = num[e][0];
51     if (d[e][1] == d[e][0] + 1) {
52         ans += num[e][1]; // 只有在次短路满足条件时才计算（距离恰好比最短路大1）
53     }
54 }
```

```

53     }
54     cout << ans.val() << endl;
55 }
```

3.21.5 输出任意一个三元环

原题：给出一张有向完全图，输出任意一个三元环上的全部元素 [See](#)。使用 dfs，复杂度 $\mathcal{O}(N + M)$ ，可以扩展到非完全图和无向图。

```

1 int n;
2 cin >> n;
3 vector<vector<int>> a(n + 1, vector<int>(n + 1));
4 for (int i = 1; i <= n; ++i) {
5     for (int j = 1; j <= n; ++j) {
6         char x;
7         cin >> x;
8         if (x == '1') a[i][j] = 1;
9     }
10 }
11
12 vector<int> vis(n + 1);
13 function<void(int, int)> dfs = [&](int x, int fa) {
14     vis[x] = 1;
15     for (int y = 1; y <= n; ++y) {
16         if (a[x][y] == 0) continue;
17         if (a[y][fa] == 1) {
18             cout << fa << " " << x << " " << y;
19             exit(0);
20         }
21         if (!vis[y]) dfs(y, x); // 这一步的if判断很关键
22     }
23 };
24 for (int i = 1; i <= n; ++i) {
25     if (!vis[i]) dfs(i, -1);
26 }
27 cout << -1;
```

3.21.6 带权最小环大小与计数

原题：给出一张有向带权图，求解图上最小环的长度、有多少个这样的最小环 [See](#)。使用 floyd，复杂度为 $\mathcal{O}(N^3)$ ，可以扩展到无向图。

```

1 LL Min = 1e18, ans = 0;
2 for (int k = 1; k <= n; k++) {
3     for (int i = 1; i <= n; i++) {
4         for (int j = 1; j <= n; j++) {
5             if (dis[i][j] > dis[i][k] + dis[k][j]) {
6                 dis[i][j] = dis[i][k] + dis[k][j];
7                 cnt[i][j] = cnt[i][k] * cnt[k][j] % mod;
8             } else if (dis[i][j] == dis[i][k] + dis[k][j]) {
9                 cnt[i][j] = (cnt[i][j] + cnt[i][k] * cnt[k][j] % mod) % mod;
10            }
11        }
12    }
13    for (int i = 1; i < k; i++) {
14        if (a[k][i]) {
15            if (a[k][i] + dis[i][k] < Min) {
16                Min = a[k][i] + dis[i][k];
17                ans = cnt[i][k];
18            } else if (a[k][i] + dis[i][k] == Min) {
19                ans = (ans + cnt[i][k]) % mod;
20            }
21        }
22    }
23 }
```

```

20         }
21     }
22 }
23 }
```

3.21.7 最小环大小

原题：给出一张无向图，求解图上最小环的长度、有多少个这样的最小环 [See](#)。使用 floyd，可以扩展到有向图。

```

1 int floyd(int n) {
2     for (int i = 1; i <= n; ++ i) {
3         for (int j = 1; j <= n; ++ j) {
4             val[i][j] = dis[i][j]; // 记录最初的边权值
5         }
6     }
7     int ans = 0x3f3f3f3f;
8     for (int k = 1; k <= n; ++ k) {
9         for (int i = 1; i < k; ++ i) { // 注意这里是沒有等于号的
10            for (int j = 1; j < i; ++ j) {
11                ans = min(ans, dis[i][j] + val[i][k] + val[k][j]);
12            }
13        }
14        for (int i = 1; i <= n; ++ i) { // 往下是标准的floyd
15            for (int j = 1; j <= n; ++ j) {
16                dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
17            }
18        }
19    }
20    return ans;
21 }
```

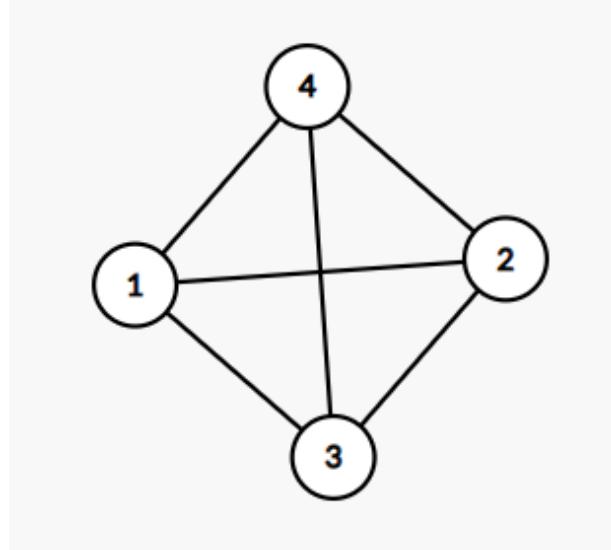
使用 bfs，复杂度为 $\mathcal{O}(N^2)$ 。

```

1 auto bfs = [&] (int s) {
2     queue<int> q; q.push(s);
3     dis[s] = 0;
4     fa[s] = -1;
5     while (q.size()) {
6         auto x = q.front(); q.pop();
7         for (auto y : ver[x]) {
8             if (y == fa[x]) continue;
9             if (dis[y] == -1) {
10                 dis[y] = dis[x] + 1;
11                 fa[y] = x;
12                 q.push(y);
13             }
14             else ans = min(ans, dis[x] + dis[y] + 1);
15         }
16     }
17 };
18 for (int i = 1; i <= n; ++ i) {
19     fill(dis + 1, dis + 1 + n, -1);
20     bfs(i);
21 }
22 cout << ans;
```

3.21.8 本质不同简单环计数

原题：给出一张无向图，输出简单环的数量 [See](#)。注意这里环套环需要分别多次统计，下图答案应当为 7。使用状压 dp，复杂度为 $\mathcal{O}(M \cdot 2^N)$ ，可以扩展到有向图。



```

1 int n, m;
2 cin >> n >> m;
3 vector<vector<int>> G(n);
4 for (int i = 0; i < m; i++) {
5     int u, v;
6     cin >> u >> v;
7     u--;
8     v--;
9     G[u].push_back(v);
10    G[v].push_back(u);
11}
12vector<vector<LL>> dp(1 << n, vector<LL>(n));
13for (int i = 0; i < n; i++) dp[1 << i][i] = 1;
14LL ans = 0;
15for (int st = 1; st < (1 << n); st++) {
16    for (int u = 0; u < n; u++) {
17        if (!dp[st][u]) continue;
18        int start = st & -st;
19        for (auto v : G[u]) {
20            if ((1 << v) < start) continue;
21            if ((1 << v) & st) {
22                if ((1 << v) == start) {
23                    ans += dp[st][u];
24                }
25            } else {
26                dp[st | (1 << v)][v] += dp[st][u];
27            }
28        }
29    }
30}
31cout << (ans - m) / 2 << "\n";

```

3.21.9 输出任意一个非二元简单环

原题：给出一张无向图，不含自环与重边，输出任意一个简单环的大小以及其上面的全部元素 [See](#)。注意输出的环的大小是随机的，**不等价于最小环**。

由于不含重边与自环，所以环的大小至少为 3，使用 dfs 处理出 dfs 序，复杂度为 $\mathcal{O}(N + M)$ ，可以扩展到有向图；如果有向图中二元环也允许计入答案，则需要删除下方标注行。

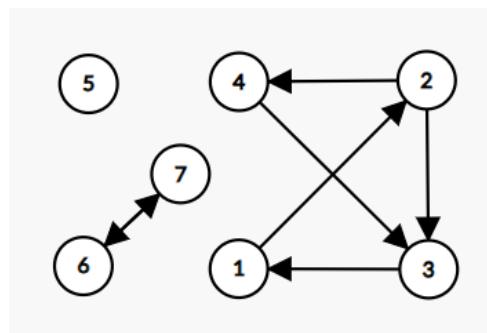
```

1 vector<int> dis(n + 1, -1), fa(n + 1);
2 auto dfs = [&](auto self, int x) -> void {
3     for (auto y : ver[x]) {
4         if (y == fa[x]) continue; // 二元环需删去该行
5         if (dis[y] == -1) {
6             dis[y] = dis[x] + 1;
7             fa[y] = x;
8             self(self, y);
9         } else if (dis[y] < dis[x]) {
10            cout << dis[x] - dis[y] + 1 << endl;
11            int pre = x;
12            cout << pre << " ";
13            while (pre != y) {
14                pre = fa[pre];
15                cout << pre << " ";
16            }
17            cout << endl;
18            exit(0);
19        }
20    }
21 };
22 for (int i = 1; i <= n; i++) {
23     if (dis[i] == -1) {
24         dis[i] = 0;
25         dfs(dfs, 1);
26     }
27 }

```

3.21.10 有向图环计数

原题：给出一张有向图，输出环的数量。注意这里环套环仅需要计算一次，数据包括二元环和自环，下图例应当输出 3 个环。使用 dfs 染色法，复杂度为 $\mathcal{O}(N + M)$ 。



```

1 int ans = 0;
2 vector<int> vis(n + 1);
3 auto dfs = [&](auto self, int x) -> void {
4     vis[x] = 1;
5     for (auto y : ver[x]) {
6         if (vis[y] == 0) {
7             self(self, y);
8         } else if (vis[y] == 1) {
9             ans++;
10        }
11    }
12    vis[x] = 2;
13 };
14 for (int i = 1; i <= n; i++) {
15     if (!vis[i]) {
16         dfs(dfs, i);
17     }

```

```

18 }
19 cout << ans << endl;

```

3.21.11 有向图简单环检查、输出

NowCoder，带自环重边、不连通。

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
4
5     vector<vector<array<int, 2>>> ver(n + 1);
6     for (int i = 1; i <= m; i++) {
7         int x, y;
8         cin >> x >> y;
9         ver[x].push_back({y, i});
10    }
11
12    vector<int> vis(n + 1);
13    vector<array<int, 2>> fa(n + 1);
14    auto dfs = [&](auto self, int x, int from) -> void {
15        vis[x] = 1;
16        for (auto [y, id] : ver[x]) {
17            if (id == from) continue;
18            if (!vis[y]) {
19                fa[y] = {x, id};
20                self(self, y, id);
21            } else if (vis[y] == 1) {
22                vector<int> V = {y}, E = {id};
23                for (int pre = x; pre != y; pre = fa[pre][0]) {
24                    V.push_back(pre);
25                    E.push_back(fa[pre][1]);
26                }
27
28                int l = V.size();
29                cout << l << "\n";
30                reverse(V.begin(), V.end());
31                reverse(E.begin(), E.end());
32                rotate(E.begin(), E.begin() + 1, E.end());
33                for (int i = 0; i < l; i++) {
34                    cout << V[i] << " \n"[i == l - 1];
35                }
36                for (int i = 0; i < l; i++) {
37                    cout << E[i] << " \n"[i == l - 1];
38                }
39                exit(0);
40            }
41        }
42        vis[x] = 2;
43    };
44    for (int i = 1; i <= n; i++) {
45        if (!vis[i]) {
46            dfs(dfs, i, -1);
47        }
48    }
49
50    cout << -1 << "\n";
51 }

```

3.21.12 无向图简单环检查、输出

[NowCoder](#)，带自环重边、不连通。

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
4
5     vector<vector<array<int, 2>>> ver(n + 1);
6     for (int i = 1; i <= m; i++) {
7         int x, y;
8         cin >> x >> y;
9         ver[x].push_back({y, i});
10        ver[y].push_back({x, i});
11    }
12
13    vector<int> vis(n + 1);
14    vector<array<int, 2>> fa(n + 1);
15    auto dfs = [&](auto self, int x, int from) -> void {
16        vis[x] = 1;
17        for (auto [y, id] : ver[x]) {
18            if (id == from) continue;
19            if (!vis[y]) {
20                fa[y] = {x, id};
21                self(self, y, id);
22            } else if (vis[y] == 1) {
23                vector<int> ans1 = {y}, ans2 = {id};
24                for (int pre = x; pre != y; pre = fa[pre][0]) {
25                    ans1.push_back(pre);
26                    ans2.push_back(fa[pre][1]);
27                }
28
29                int l = ans1.size();
30                cout << l << "\n";
31                for (int i = 0; i < l; i++) {
32                    cout << ans1[i] << " \n"[i == l - 1];
33                }
34                for (int i = 0; i < l; i++) {
35                    cout << ans2[i] << " \n"[i == l - 1];
36                }
37                exit(0);
38            }
39        }
40        vis[x] = 2;
41    };
42    for (int i = 1; i <= n; i++) {
43        if (!vis[i]) {
44            dfs(dfs, i, -1);
45        }
46    }
47
48    cout << -1 << "\n";
49 }
```

3.21.13 判定带环图是否是平面图

原题：给定一个环以一些额外边，对于每一条额外边判定其位于环外还是环内，使得任意两条无重合顶点的额外边都不相交（即这张图构成平面图）[See1](#), [See2](#)。

使用 2-sat。考虑全部边都位于环内，那么“一条边完全包含另一条边”、“两条边完全没有交集”这两种情况都不会相交，可以直接跳过这两种情况的讨论。

```
1 signed main() {
2     int n, m;
3     cin >> n >> m;
4     vector<pair<int, int>> in(m);
5     for (int i = 0, x, y; i < m; i++) {
6         cin >> x >> y;
7         in[i] = minmax(x, y);
8     }
9     TwoSat sat(m);
10    for (int i = 0; i < m; i++) {
11        auto [s, e] = in[i];
12        for (int j = i + 1; j < m; j++) {
13            auto [S, E] = in[j];
14            if (s < S && S < e && e < E || S < s && s < E && E < e) {
15                sat.add(i, 0, j, 0);
16                sat.add(i, 1, j, 1);
17            }
18        }
19    }
20    if (!sat.work()) {
21        cout << "Impossible\n";
22        return 0;
23    }
24    auto ans = sat.answer();
25    for (auto it : ans) {
26        cout << (it ? "out" : "in") << " ";
27    }
28 }
```

/END/

4 网络流

4.1 最大流

使用 Dinic 算法，理论最坏复杂度为 $\mathcal{O}(N^2M)$ ，例题范围： $N = 1200$, $m = 5 \times 10^3$ 。一般步骤：BFS 建立分层图，无回溯 DFS 寻找所有可行的增广路径。封装：求从点 S 到点 T 的最大流。预流推进见常数优化章节。

```

1 template<typename T> struct Flow_ {
2     const int n;
3     const T inf = numeric_limits<T>::max();
4     struct Edge {
5         int to;
6         T w;
7         Edge(int to, T w) : to(to), w(w) {}
8     };
9     vector<Edge> ver;
10    vector<vector<int>> h;
11    vector<int> cur, d;
12
13    Flow_(int n) : n(n + 1), h(n + 1) {}
14    void add(int u, int v, T c) {
15        h[u].push_back(ver.size());
16        ver.emplace_back(v, c);
17        h[v].push_back(ver.size());
18        ver.emplace_back(u, 0);
19    }
20    bool bfs(int s, int t) {
21        d.assign(n, -1);
22        d[s] = 0;
23        queue<int> q;
24        q.push(s);
25        while (!q.empty()) {
26            auto x = q.front();
27            q.pop();
28            for (auto it : h[x]) {
29                auto [y, w] = ver[it];
30                if (w && d[y] == -1) {
31                    d[y] = d[x] + 1;
32                    if (y == t) return true;
33                    q.push(y);
34                }
35            }
36        }
37        return false;
38    }
39    T dfs(int u, int t, T f) {
40        if (u == t) return f;
41        auto r = f;
42        for (int &i = cur[u]; i < h[u].size(); i++) {
43            auto j = h[u][i];
44            auto &[v, c] = ver[j];
45            auto &[u, rc] = ver[j ^ 1];
46            if (c && d[v] == d[u] + 1) {
47                auto a = dfs(v, t, std::min(r, c));
48                c -= a;
49                rc += a;
50                r -= a;
51                if (!r) return f;
52            }
53        }
54    }
55}
```

```

54     return f - r;
55 }
56 T work(int s, int t) {
57     T ans = 0;
58     while (bfs(s, t)) {
59         cur.assign(n, 0);
60         ans += dfs(s, t, inf);
61     }
62     return ans;
63 }
64 };
65 using Flow = Flow_<int>;

```

4.2 最小割

基础模型：构筑二分图，左半部 n 个点代表盈利项目，右半部 m 个点代表材料成本，收益为盈利之和减去成本之和，求最大收益。

建图：建立源点 S 向左半部连边，建立汇点 T 向右半部连边，如果某个项目需要某个材料，则新增一条容量 $+\infty$ 的跨部边。

割边：放弃某个项目则断开 S 至该项目的边，购买某个原料则断开该原料至 T 的边，最终的图一定不存在从 S 到 T 的路径，此时我们得到二分图的一个 $S-T$ 割。此时最小割即为求解最大流，边权之和减去最大流即为最大收益。

```

1 signed main() {
2     int n, m;
3     cin >> n >> m;
4
5     int S = n + m + 1, T = n + m + 2;
6     Flow flow(T);
7     for (int i = 1; i <= n; i++) {
8         int w;
9         cin >> w;
10        flow.add(S, i, w);
11    }
12
13    int sum = 0;
14    for (int i = 1; i <= m; i++) {
15        int x, y, w;
16        cin >> x >> y >> w;
17        flow.add(x, n + i, 1E18);
18        flow.add(y, n + i, 1E18);
19        flow.add(n + i, T, w);
20        sum += w;
21    }
22    cout << sum - flow.work(S, T) << endl;
23 }

```

4.3 最小割树 Gomory-Hu Tree

无向连通图抽象出的一棵树，满足任意两点间的距离是他们的最小割。一共需要跑 n 轮最小割，总复杂度 $\mathcal{O}(N^3M)$ ，预处理最小割树上任意两点的距离 $\mathcal{O}(N^2)$ 。

过程：分治 n 轮，每一轮在图上随机选点，跑一轮最小割后连接树边；这一网络的残留网络会将剩余的点分为两组，根据分组分治。

```

1 void reset() { // struct需要额外封装退流
2     for (int i = 0; i < ver.size(); i += 2) {
3         ver[i].w += ver[i ^ 1].w;

```

```

4     ver[i ^ 1].w = 0;
5   }
6 }
7
8 signed main() { // Gomory-Hu Tree
9   int n, m;
10  cin >> n >> m;
11
12  Flow<int> flow(n);
13  for (int i = 1; i <= m; i++) {
14    int u, v, w;
15    cin >> u >> v >> w;
16    flow.add(u, v, w);
17    flow.add(v, u, w);
18  }
19
20  vector<int> vis(n + 1), fa(n + 1);
21  vector ans(n + 1, vector<int>(n + 1, 1E9)); // N^2 枚举出全部答案
22  vector<vector<pair<int, int>>> adj(n + 1);
23  for (int i = 1; i <= n; i++) { // 分治 n 轮
24    int s = 0; // 本质是在树上随机选点、跑最小割后连边
25    for (; s <= n; s++) {
26      if (fa[s] != s) break;
27    }
28    int t = fa[s];
29
30    int ans = flow.work(s, t); // 残留网络将点集分为两组，分治
31    adj[s].push_back({t, ans});
32    adj[t].push_back({s, ans});
33
34    vis.assign(n + 1, 0);
35    auto dfs = [&](auto dfs, int u) -> void {
36      vis[u] = 1;
37      for (auto it : flow.h[u]) {
38        auto [v, c] = flow.ver[it];
39        if (c && !vis[v]) {
40          dfs(dfs, v);
41        }
42      }
43    };
44    dfs(dfs, s);
45    for (int j = 0; j <= n; j++) {
46      if (vis[j] && fa[j] == t) {
47        fa[j] = s;
48      }
49    }
50  }
51
52  for (int i = 0; i <= n; i++) {
53    auto dfs = [&](auto dfs, int u, int fa, int c) -> void {
54      ans[i][u] = c;
55      for (auto [v, w] : adj[u]) {
56        if (v == fa) continue;
57        dfs(dfs, v, u, min(c, w));
58      }
59    };
59    dfs(dfs, i, -1, 1E9);
60  }
61
62  int q;
63  cin >> q;
64  while (q--) {
65    int u, v;
66    cin >> u >> v;
67

```

```

68     cout << ans[u][v] << "\n"; // 预处理答数组
69 }
70 }
```

4.4 费用流

给定一个带费用的网络，规定 (u, v) 间的费用为 $f(u, v) \times w(u, v)$ ，求解该网络中总花费最小的最大流称之为**最小费用最大流**。总时间复杂度为 $\mathcal{O}(NMf)$ ，其中 f 代表最大流。

```

1 struct MinCostFlow {
2     using LL = long long;
3     using PII = pair<LL, int>;
4     const LL INF = numeric_limits<LL>::max();
5     struct Edge {
6         int v, c, f;
7         Edge(int v, int c, int f) : v(v), c(c), f(f) {}
8     };
9     const int n;
10    vector<Edge> e;
11    vector<vector<int>> g;
12    vector<LL> h, dis;
13    vector<int> pre;
14
15    MinCostFlow(int n) : n(n), g(n) {}
16    void add(int u, int v, int c, int f) { // c 流量, f 费用
17        // if (f < 0) {
18        //     g[u].push_back(e.size());
19        //     e.emplace_back(v, 0, f);
20        //     g[v].push_back(e.size());
21        //     e.emplace_back(u, c, -f);
22        // } else {
23            g[u].push_back(e.size());
24            e.emplace_back(v, c, f);
25            g[v].push_back(e.size());
26            e.emplace_back(u, 0, -f);
27        }
28    }
29    bool dijkstra(int s, int t) {
30        dis.assign(n, INF);
31        pre.assign(n, -1);
32        priority_queue<PII, vector<PII>, greater<PII>> que;
33        dis[s] = 0;
34        que.emplace(0, s);
35        while (!que.empty()) {
36            auto [d, u] = que.top();
37            que.pop();
38            if (dis[u] < d) continue;
39            for (int i : g[u]) {
40                auto [v, c, f] = e[i];
41                if (c > 0 && dis[v] > d + h[u] - h[v] + f) {
42                    dis[v] = d + h[u] - h[v] + f;
43                    pre[v] = i;
44                    que.emplace(dis[v], v);
45                }
46            }
47        }
48        return dis[t] != INF;
49    }
50    pair<int, LL> flow(int s, int t) {
51        int flow = 0;
52        LL cost = 0;
53        h.assign(n, 0);
```

```
54     while (dijkstra(s, t)) {
55         for (int i = 0; i < n; ++i) h[i] += dis[i];
56         int aug = numeric_limits<int>::max();
57         for (int i = t; i != s; i = e[pre[i] ^ 1].v) aug = min(aug,
e[pre[i]].c);
58             for (int i = t; i != s; i = e[pre[i] ^ 1].v) {
59                 e[pre[i]].c -= aug;
60                 e[pre[i] ^ 1].c += aug;
61             }
62             flow += aug;
63             cost += LL(aug) * h[t];
64         }
65         return {flow, cost};
66     }
67 }
```

/END/

5 数论

5.1 常见数列

5.1.1 调和级数

满足调和级数 $\mathcal{O}\left(\frac{N}{1} + \frac{N}{2} + \frac{N}{3} + \dots + \frac{N}{N}\right)$ ，可以用 $\approx N \ln N$ 来拟合，但是会略小，误差量级在 10% 左右。本地可以在 500ms 内完成 10^8 量级的预处理计算。

N的量级	1	2	3	4	5	6	7	8	9
累加和	27	482	7'069	93'668	1'166'750	13'970'034	162'725'364	1'857'511'568	20'877'697'634

下方示例为求解 1 到 N 中各个数字的因数值。

```

1 const int N = 1E5;
2 vector<vector<int>> dic(N + 1);
3 for (int i = 1; i <= N; i++) {
4     for (int j = i; j <= N; j += i) {
5         dic[j].push_back(i);
6     }
7 }
```

5.1.2 素数密度与分布

N的量级	1	2	3	4	5	6	7	8	9
素数数量	4	25	168	1'229	9'592	78'498	664'579	5'761'455	50'847'534

除此之外，对于任意两个相邻的素数 $p_1, p_2 \leq 10^9$ ，有 $|p_1 - p_2| < 300$ 成立，更具体的说，最大的差值为 282。

5.1.3 因数最多数字与其因数数量

N的量级	1	2	3	4	5	6	7
因数最多数字的因数数量	4	25	32	64	128	240	448
因数最多的数字	-	-	-	7560, 9240	83160, 98280	720720, 831600, 942480, 982800, 997920	-

5.2 快速幂

5.2.1 常规

```

1 i64 pow(i64 a, int b, int m) { // 复杂度是 log N
2     i64 r = 1 % m; /**! 这里的取模容易遗漏 */
3     for (; b; b >= 1, a = a * a % m) {
4         if (b & 1) r = r * a % m;
5     }
6     return r;
7 }
```

5.2.2 长整型 with 防爆乘法

```

1 i64 mul(i64 a, i64 b, i64 m) { // 由于不取模，运行速度非常快
2     a %= m, b %= m; /**! 这里的取模容易遗漏 */
3     i64 r = a * b - m * i64(1.L / m * a * b);
4     return r - m * (r >= m) + m * (r < 0);
5 }
6
7 i64 mul(i64 a, i64 b, i64 m) { // 比较慢
8     return (__int128)a * b % m;
9 }
10
11 i64 pow(i64 a, i64 b, i64 m) {
12     i64 res = 1 % m; /**! 这里的取模容易遗漏 */
13     for (; b; b >>= 1, a = mul(a, a, m)) { // 配合下方手写乘法
14         if (b & 1) res = mul(res, a, m);
15     }
16     return res;
17 }
```

5.3 质数判定

5.3.1 试除法

标准 $\mathcal{O}(\sqrt{N})$ ，有常数优化版本可达到 $\mathcal{O}(\frac{\sqrt{N}}{3})$ 。

```

1 bool isprime(int n) {
2     if (n < 2) return false;
3     for (int i = 2; i <= n / i; i++) {
4         if (n % i == 0) return false;
5     }
6     return true;
7 }
```

5.3.2 筛法

使用欧拉筛（线性筛），预期时间复杂度为 $\mathcal{O}(N)$ 。

```

1 vector<int> prime, minp;
2
3 void sieve(int n = 1e7) {
4     minp.resize(n + 1);
5     for (int i = 2; i <= n; i++) {
6         if (!minp[i]) {
7             minp[i] = i;
8             prime.push_back(i);
9         }
10        for (auto j : prime) {
11            if (j > minp[i] || j > n / i) break;
12            minp[i * j] = j;
13        }
14    }
15 }
16
17 bool isprime(int n) {
18     return minp[n] == n;
19 }
```

5.3.3 Miller-Rabin

随机化验证，非严谨计算的平均复杂度约为 $\mathcal{O}(3.5 \times \log X)$ 。对于某些强力质数，可能会退化至约 $\mathcal{O}(35 \times \log X)$ 。有常数优化版本可以再快五倍。

```

1 i64 mul(i64 a, i64 b, i64 m) { // 快速乘提速，约四倍效果
2     i64 r = a * b - m * i64(1.L / m * a * b);
3     return r - m * (r >= m) + m * (r < 0);
4 }
5
6 i64 pow(i64 a, i64 b, i64 m) {
7     i64 res = 1 % m;
8     for (; b; b >= 1, a = mul(a, a, m)) {
9         if (b & 1) res = mul(res, a, m);
10    }
11    return res;
12 }
13
14 bool isprime(i64 n) {
15     if (n < 2 || n % 6 % 4 != 1) {
16         return (n | 1) == 3;
17     }
18     i64 s = __builtin_ctzll(n - 1), d = n >> s;
19     for (i64 a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
20         i64 p = pow(a % n, d, n), i = s;
21         while (p != 1 && p != n - 1 && a % n && i--) {
22             p = mul(p, p, n);
23         }
24         if (p != n - 1 && i != s) return false;
25     }
26     return true;
27 }
```

5.4 质因子分解

5.4.1 筛法

使用欧拉筛（线性筛），预处理时间复杂度 $\mathcal{O}(N)$ ，单次查询 $\mathcal{O}(\text{Prime Numer})$ 。

```

1 vector<int> prime, minp, maxp;
2
3 void sieve(int n = 1e7) {
4     minp.resize(n + 1);
5     maxp.resize(n + 1);
6     for (int i = 2; i <= n; i++) {
7         if (!minp[i]) {
8             minp[i] = maxp[i] = i;
9             prime.push_back(i);
10        }
11        for (auto j : prime) {
12            if (j > minp[i] || j > n / i) break;
13            minp[i * j] = j;
14            maxp[i * j] = maxp[i];
15        }
16    }
17 }
18
19 vector<array<int, 2>> factorize(int n) {
20     vector<array<int, 2>> ans;
21     while (n > 1) {
22         int now = minp[n], cnt = 0;
```

```

23     while (n % now == 0) {
24         n /= now;
25         cnt++;
26     }
27     ans.push_back({now, cnt});
28 }
29
30 }
```

5.4.2 Pollard-Rho

以单个因子 $\mathcal{O}(\log X)$ 的复杂度输出数字 X 的全部质因数，由于需要结合素数测试，总复杂度会略高一些。如果遇到超时的情况，可能需要考虑进一步优化，例如检查题目是否强制要求枚举全部质因数等等。有常数优化版本可以再快五倍。

```

1 i64 rho(i64 n) {
2     if (!(n & 1)) return 2;
3     i64 x = 0, y = 0, prod = 1;
4     auto f = [&](i64 x) -> i64 {
5         return mul(x, x, n) + 5; // 这里的种子为 1 时能被 hack，取 5 到目前为止没有什么
问题
6     };
7     for (int t = 30, z = 0; t % 64 || gcd(prod, n) == 1; ++t) {
8         if (x == y) x = ++z, y = f(x);
9         if (i64 q = mul(prod, x + n - y, n)) prod = q;
10        x = f(x), y = f(f(y));
11    }
12    return gcd(prod, n);
13}
14
15 vector<i64> factorize(i64 x) {
16     vector<i64> res;
17     auto f = [&](auto f, i64 x) {
18         if (x == 1) return;
19         if (isprime(x)) return res.push_back(x);
20         i64 y = rho(x);
21         f(f, y), f(f, x / y);
22     };
23     f(f, x), sort(res.begin(), res.end());
24     return res;
25 }
```

5.5 裴蜀定理

$ax + by = c$ ($x \in Z^*, y \in Z^*$) 成立的充要条件是 $\gcd(a, b) | c$ (Z^* 表示正整数集)。

例题：给定一个序列 a ，找到一个序列 x ，使得 $\sum_{i=1}^n a_i x_i$ 最小。

```

1 LL n, a, ans;
2 LL gcd(LL a, LL b){
3     return b ? gcd(b, a % b) : a;
4 }
5 int main(){
6     cin >> n;
7     for (int i = 0; i < n; i ++ ){
8         cin >> a;
9         if (a < 0) a = -a;
10        ans = gcd(ans, a);
11    }
12    cout << ans << "\n";
13    return 0;
```

14 | }

5.6 逆元

5.6.1 费马小定理解（借助快速幂）

单次计算的复杂度即为快速幂的复杂度 $\mathcal{O}(\log X)$ 。限制： MOD 必须是质数，且需要满足 x 与 MOD 互质。

1 | LL inv(LL x) { return mypow(x, mod - 2, mod); }

5.6.2 扩展欧几里得解

此方法的 MOD 没有限制，复杂度为 $\mathcal{O}(\log X)$ ，但是比快速幂法常数大一些。

```

1 int x, y;
2 int exgcd(int a, int b, int &x, int &y) { //扩展欧几里得算法
3     if (b == 0) {
4         x = 1, y = 0;
5         return a; //到达递归边界开始向上一层返回
6     }
7     int r = exgcd(b, a % b, x, y);
8     int temp = y; //把x y变成上一层的
9     y = x - (a / b) * y;
10    x = temp;
11    return r; //得到a b的最大公因数
12 }
13 LL getInv(int a, int mod) { //求a在mod下的逆元，不存在逆元返回-1
14     LL x, y, d = exgcd(a, mod, x, y);
15     return d == 1 ? (x % mod + mod) % mod : -1;
16 }
```

5.6.3 离线求解：线性递推解

以 $\mathcal{O}(N)$ 的复杂度完成 $1 - N$ 中全部逆元的计算。

```

1 inv[1] = 1;
2 for (int i = 2; i <= n; i++) {
3     inv[i] = (p - p / i) * inv[p % i] % p;
```

5.7 扩展欧几里得 exgcd

求解形如 $a \cdot x + b \cdot y = \gcd(a, b)$ 的不定方程的任意一组解。

```

1 int exgcd(int a, int b, int &x, int &y) {
2     if (!b) {
3         x = 1, y = 0;
4         return a;
5     }
6     int d = exgcd(b, a % b, y, x);
7     y -= a / b * x;
8     return d;
9 }
```

例题：求解二元一次不定方程 $A \cdot x + B \cdot y = C$ 。

1 | auto clac = [&](int a, int b, int c) {

```

2     int u = 1, v = 1;
3     if (a < 0) { // 负数特判，但是没用经过例题测试
4         a = -a;
5         u = -1;
6     }
7     if (b < 0) {
8         b = -b;
9         v = -1;
10    }
11
12    int x, y, d = exgcd(a, b, x, y), ans;
13    if (c % d != 0) { // 无整数解
14        cout << -1 << "\n";
15        return;
16    }
17    a /= d, b /= d, c /= d;
18    x *= c, y *= c; // 得到可行解
19
20    ans = (x % b + b - 1) % b + 1;
21    auto [A, B] = pair{u * ans, v * (c - ans * a) / b}; // x最小正整数 特解
22
23    ans = (y % a + a - 1) % a + 1;
24    auto [C, D] = pair{u * (c - ans * b) / a, v * ans}; // y最小正整数 特解
25
26    int num = (C - A) / b + 1; // xy均为正整数 的 解的组数
27};

```

5.8 离散对数 bsgs 与 exbsgs

以 $\mathcal{O}(\sqrt{P})$ 的复杂度求解 $a^x \equiv b \pmod{P}$ 。其中标准 BSGS 算法不能计算 a 与 MOD 互质的情况，而 exbsgs 则可以。

```

1 namespace BSGS {
2     LL a, b, p;
3     map<LL, LL> f;
4     inline LL gcd(LL a, LL b) { return b > 0 ? gcd(b, a % b) : a; }
5     inline LL ps(LL n, LL k, int p) {
6         LL r = 1;
7         for (; k; k >= 1) {
8             if (k & 1) r = r * n % p;
9             n = n * n % p;
10        }
11        return r;
12    }
13    void exgcd(LL a, LL b, LL &x, LL &y) {
14        if (!b)
15            x = 1, y = 0;
16        else {
17            exgcd(b, a % b, x, y);
18            LL t = x;
19            x = y;
20            y = t - a / b * y;
21        }
22    }
23    LL inv(LL a, LL b) {
24        LL x, y;
25        exgcd(a, b, x, y);
26        return (x % b + b) % b;
27    }
28    LL bsgs(LL a, LL b, LL p) {
29        f.clear();
30        int m = ceil(sqrt(p));

```

```

31     b %= p;
32     for (int i = 1; i <= m; i++) {
33         b = b * a % p;
34         f[b] = i;
35     }
36     LL tmp = ps(a, m, p);
37     b = 1;
38     for (int i = 1; i <= m; i++) {
39         b = b * tmp % p;
40         if (f.count(b) > 0) return (i * m - f[b] + p) % p;
41     }
42     return -1;
43 }
44 LL exbsgs(LL a, LL b, LL p) {
45     if (b == 1 || p == 1) return 0;
46     LL g = gcd(a, p), k = 0, na = 1;
47     while (g > 1) {
48         if (b % g != 0) return -1;
49         k++;
50         b /= g;
51         p /= g;
52         na = na * (a / g) % p;
53         if (na == b) return k;
54         g = gcd(a, p);
55     }
56     LL f = bsgs(a, b * inv(na, p) % p, p);
57     if (f == -1) return -1;
58     return f + k;
59 }
60 } // namespace BSGS
61
62 using namespace BSGS;
63
64 int main() {
65     IOS;
66     cin >> p >> a >> b;
67     a %= p, b %= p;
68     LL ans = exbsgs(a, b, p);
69     if (ans == -1) cout << "no solution\n";
70     else cout << ans << "\n";
71     return 0;
72 }
```

5.9 欧拉函数

5.9.1 直接求解单个数的欧拉函数

1 到 N 中与 N 互质数的个数称为欧拉函数，记作 $\varphi(N)$ 。求解欧拉函数的过程即为分解质因数的过程，复杂度 $\mathcal{O}(\sqrt{n})$ 。

```

1 int phi(int n) { //求解 phi(n)
2     int ans = n;
3     for(int i = 2; i <= n / i; i++) { //注意，这里要写 n / i，以防止 int 型溢出风险和
4         if(n % i == 0) {
5             ans = ans / i * (i - 1);
6             while(n % i == 0) n /= i;
7         }
8     }
9     if(n > 1) ans = ans / n * (n - 1); //特判 n 为质数的情况
10    return ans;
11 }
```

5.9.2 求解 1 到 N 所有数的欧拉函数

利用上述第四条性质，我们可以快速递推出 $2 - N$ 中每个数的欧拉函数，复杂度 $\mathcal{O}(N)$ ，而该算法即是线性筛的算法。

```

1 const int N = 1e5 + 7;
2 int v[N], prime[N], phi[N];
3 void euler(int n) {
4     ms(v, 0); //最小质因子
5     int m = 0; //质数数量
6     for (int i = 2; i <= n; ++ i) {
7         if (v[i] == 0) { // i 是质数
8             v[i] = i, prime[++ m] = i;
9             phi[i] = i - 1;
10        }
11        //为当前的数 i 乘上一个质因子
12        for (int j = 1; j <= m; ++ j) {
13            //如 i 有比 prime[j] 更小的质因子，或超出 n，停止
14            if(prime[j] > v[i] || prime[j] > n / i) break;
15            // prime[j] 是合数 i * prime[j] 的最小质因子
16            v[i * prime[j]] = prime[j];
17            phi[i * prime[j]] = phi[i] * (i % prime[j] ? prime[j] - 1 : prime[j]);
18        }
19    }
20 }
21 int main() {
22     int n; cin >> n; euler(n);
23     for (int i = 1; i <= n; ++ i) cout << phi[i] << endl;
24     return 0;
25 }
```

5.9.3 使用莫比乌斯反演求解欧拉函数

```

1 int phi[N];
2 vector<int> fac[N];
3 void get_eulers() {
4     for (int i = 1; i <= N - 10; i++) {
5         for (int j = i; j <= N - 10; j += i) {
6             fac[j].push_back(i);
7         }
8     }
9     phi[1] = 1;
10    for (int i = 2; i <= N - 10; i++) {
11        phi[i] = i;
12        for (auto j : fac[i]) {
13            if (j == i) continue;
14            phi[i] -= phi[j];
15        }
16    }
17 }
```

5.10 组合数

5.10.1 debug

提供一组测试数据： $\binom{132}{66} = 377'389'666'165'540'953'244'592'352'291'892'721'700$ ，模数为 998244353 时为 241'200'029； $10^9 + 7$ 时为 598375978。

5.10.2 逆元+卢卡斯定理（质数取模）

$\mathcal{O}(N)$ ，模数必须为质数。

```

1 struct Comb {
2     int n;
3     vector<Z> _fac, _inv;
4
5     Comb() : _fac{1}, _inv{0} {}
6     Comb(int n) : Comb() {
7         init(n);
8     }
9     void init(int m) {
10        if (m <= n) return;
11        _fac.resize(m + 1);
12        _inv.resize(m + 1);
13        for (int i = n + 1; i <= m; i++) {
14            _fac[i] = _fac[i - 1] * i;
15        }
16        _inv[m] = _fac[m].inv();
17        for (int i = m; i > n; i--) {
18            _inv[i - 1] = _inv[i] * i;
19        }
20        n = m;
21    }
22    Z fac(int x) {
23        if (x > n) init(x);
24        return _fac[x];
25    }
26    Z inv(int x) {
27        if (x > n) init(x);
28        return _inv[x];
29    }
30    Z C(int x, int y) {
31        if (x < 0 || y < 0 || x < y) return 0;
32        return fac(x) * inv(y) * inv(x - y);
33    }
34    Z P(int x, int y) {
35        if (x < 0 || y < 0 || x < y) return 0;
36        return fac(x) * inv(x - y);
37    }
38 } comb(1 << 21);

```

5.10.3 质因数分解

此法适用于： $1 < n, m, MOD < 10^7$ 的情况。

```

1 int n,m,p,b[10000005],prime[1000005],t,min_prime[1000005];
2 void euler_Prime(int n){//用欧拉筛求出1~n中每个数的最小质因数的编号是多少，保存在
min_prime中
3     for(int i=2;i<n;i++){
4         if(b[i]==0){
5             prime[++t]=i;
6             min_prime[i]=t;
7         }
8         for(int j=1;j<=t&&i*prime[j]<=n;j++){
9             b[prime[j]*i]=1;
10            min_prime[prime[j]*i]=j;
11            if(i%prime[j]==0) break;
12        }
13    }
14 }

```

```

15 long long c(int n,int m,int p){//计算C(n,m)%p的值
16     euler_Prime(n);
17     int a[t+5];//t代表1~n中质数的个数，a[i]代表编号为i的质数在答案中出现的次数
18     for(int i=1;i<=t;i++) a[i]=0;//注意清0，一开始是随机数
19     for(int i=n;i>=n-m+1;i--){//处理分子
20         int x=i;
21         while (x!=1){
22             a[min_prime[x]]++; //注意min_prime中保存的是这个数的最小质因数的编号 ( 1~t )
23             x/=prime[min_prime[x]];
24         }
25     }
26     for(int i=1;i<=m;i++){//处理分母
27         int x=i;
28         while (x!=1){
29             a[min_prime[x]]--;
30             x/=prime[min_prime[x]];
31         }
32     }
33     long long ans=1;
34     for(int i=1;i<=t;i++){//枚举质数的编号，看它出现了几次
35         while(a[i]>0){
36             ans=ans*prime[i]%p;
37             a[i]--;
38         }
39     }
40     return ans;
41 }
42 int main(){
43     cin>>n>>m;
44     m=min(m,n-m); //小优化
45     cout<<c(n,m,MOD);
46 }
```

5.10.4 杨辉三角（精确计算）

60 以内 `long long` 可解，130 以内 `__int128` 可解。

```

1 vector<vector<int>> C(n + 1, vector<int>(n + 1));
2 C[0][0] = 1;
3 for (int i = 1; i <= n; i++) {
4     C[i][0] = 1;
5     for (int j = 1; j <= n; j++) {
6         C[i][j] = C[i - 1][j] + C[i - 1][j - 1];
7     }
8 }
9 cout << C[n][m] << endl;
```

5.11 求解连续数字的正约数集合—倍数法

使用规律递推优化，时间复杂度为 $\mathcal{O}(N \log N)$ ，如果不需要详细的输出集合，则直接将 `vector` 换为普通数组即可（时间更快）。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e5 + 7;
4 vector<int> f[N];
5
6 void divide(int n) {
7     for (int i = 1; i <= n; ++ i)
8         for (int j = 1; j <= n / i; ++ j)
9             f[i * j].push_back(i);
```

```

10     for (int i = 1; i <= n; ++ i) {
11         for (auto it : f[i]) cout << it << " ";
12         cout << endl;
13     }
14 }
15 int main() {
16     int x; cin >> x; divide(x);
17     return 0;
18 }
```

5.12 容斥原理

定义：

$$|S_1 \cup S_2 \cup S_3 \cup \dots \cup S_n| = \sum_{i=1}^N |S_i| - \sum_{i,j=1}^N |S_i \cap S_j| + \sum_{i,j,k=1}^N |S_i \cap S_j \cap S_k| - \dots$$

例题：给定一个整数 n 和 m 个不同的质数 p_1, p_2, \dots, p_m ，请你求出 $1 \sim n$ 中能被 p_1, p_2, \dots, p_m 中的至少一个数整除的整数有多少个。

5.12.1 二进制枚举解

```

1 int main(){
2     ios::sync_with_stdio(false);cin.tie(0);
3     LL n, m;
4     cin >> n >> m;
5     vector<LL> p(m);
6     for (int i = 0; i < m; i ++ )
7         cin >> p[i];
8     LL ans = 0;
9     for (int i = 1; i < (1 << m); i ++ ){
10         LL t = 1, cnt = 0;
11         for (int j = 0; j < m; j ++ ){
12             if (i >> j & 1){
13                 cnt++;
14                 t *= p[j];
15                 if (t > n){
16                     t = -1;
17                     break;
18                 }
19             }
20         }
21         if (t != -1){
22             if (cnt & 1) ans += n / t;
23             else ans -= n / t;
24         }
25     }
26     cout << ans << "\n";
27     return 0;
28 }
```

5.12.2 dfs 解

```

1 int main(){
2     ios::sync_with_stdio(false);cin.tie(0);
3     LL n, m;
4     cin >> n >> m;
5     vector<LL> p(m);
6     for (int i = 0; i < m; i ++ )
7         cin >> p[i];
8     LL ans = 0;
9     function<void(LL, LL, LL)> dfs = [&](LL x, LL s, LL odd){
10         if (x == m){
```

```

11         if (s == 1) return;
12         ans += odd * (n / s);
13         return;
14     }
15     dfs(x + 1, s, odd);
16     if (s <= n / p[x]) dfs(x + 1, s * p[x], -odd);
17 }
18 dfs(0, 1, -1);
19 cout << ans << "\n";
20 return 0;
21 }
```

5.13 同余方程组、拓展中国剩余定理 exCRT

公式： $x \equiv b_i \pmod{a_i}$ ，即 $(x - b_i) \mid a_i$ 。

```

1 int n; LL ai[maxn], bi[maxn];
2 inline int mypow(int n, int k, int p) {
3     int r = 1;
4     for (; k; k >= 1, n = n * n % p)
5         if (k & 1) r = r * n % p;
6     return r;
7 }
8 LL exgcd(LL a, LL b, LL &x, LL &y) {
9     if (b == 0) { x = 1, y = 0; return a; }
10    LL gcd = exgcd(b, a % b, x, y), tp = x;
11    x = y, y = tp - a / b * y;
12    return gcd;
13 }
14 LL excrt() {
15     LL x, y, k;
16     LL M = bi[1], ans = ai[1];
17     for (int i = 2; i <= n; ++ i) {
18         LL a = M, b = bi[i], c = (ai[i] - ans % b + b) % b;
19         LL gcd = exgcd(a, b, x, y), bg = b / gcd;
20         if (c % gcd != 0) return -1;
21         x = mul(x, c / gcd, bg);
22         ans += x * M;
23         M *= bg;
24         ans = (ans % M + M) % M;
25     }
26     return (ans % M + M) % M;
27 }
28 int main() {
29     cin >> n;
30     for (int i = 1; i <= n; ++ i) cin >> bi[i] >> ai[i];
31     cout << excrt() << endl;
32     return 0;
33 }
```

5.14 求解连续按位异或

以 $\mathcal{O}(1)$ 复杂度计算 $0 \oplus 1 \oplus \dots \oplus n$ 。

```

1 unsigned xor_n(unsigned n) {
2     unsigned t = n & 3;
3     if (t & 1) return t / 2u ^ 1;
4     return t / 2u ^ n;
5 }
```

```

1 i64 xor_n(i64 n) {
2     if (n % 4 == 1) return 1;
3     else if (n % 4 == 2) return n + 1;
4     else if (n % 4 == 3) return 0;
5     else return n;
6 }

```

5.15 高斯消元求解线性方程组

题目大意：输入一个包含 N 个方程 N 个未知数的线性方程组，系数与常数均为实数（两位小数）。求解这个方程组。如果存在唯一解，则输出所有 N 个未知数的解，结果保留两位小数。如果无数解，则输出 X，如果无解，则输出 N。

```

1 const int N = 110;
2 const double eps = 1e-8;
3 LL n;
4 double a[N][N];
5 LL gauss(){
6     LL c, r;
7     for (c = 0, r = 0; c < n; c ++ ){
8         LL t = r;
9         for (int i = r; i < n; i ++ ) //找到绝对值最大的行
10            if (fabs(a[i][c]) > fabs(a[t][c]))
11                t = i;
12            if (fabs(a[t][c]) < eps) continue;
13            for (int j = c; j < n + 1; j ++ ) swap(a[t][j], a[r][j]); //将绝对值最大的一行换到最顶端
14            for (int j = n; j >= c; j -- ) a[r][j] /= a[r][c]; //将当前行首位变成 1
15            for (int i = r + 1; i < n; i ++ ) //将下面列消成 0
16                if (fabs(a[i][c]) > eps)
17                    for (int j = n; j >= c; j -- )
18                        a[i][j] -= a[r][j] * a[i][c];
19            r ++ ;
20        }
21        if (r < n){
22            for (int i = r; i < n; i ++ )
23                if (fabs(a[i][n]) > eps)
24                    return 2;
25            return 1;
26        }
27        for (int i = n - 1; i >= 0; i -- )
28            for (int j = i + 1; j < n; j ++ )
29                a[i][n] -= a[i][j] * a[j][n];
30        return 0;
31    }
32    int main(){
33        cin >> n;
34        for (int i = 0; i < n; i ++ )
35            for (int j = 0; j < n + 1; j ++ )
36                cin >> a[i][j];
37        LL t = gauss();
38        if (t == 0){
39            for (int i = 0; i < n; i ++ ){
40                if (fabs(a[i][n]) < eps) a[i][n] = abs(a[i][n]);
41                printf("%.2lf\n", a[i][n]);
42            }
43        }
44        else if (t == 1) cout << "Infinite group solutions\n";
45        else cout << "No solution\n";
46        return 0;
47    }

```

5.16 康拓展开

5.16.1 正向展开普通解法

将一个字典序排列转换成序号。例如：12345->1，12354->2。

```

1 int f[20];
2 void jie_cheng(int n) { // 打出1-n的阶乘表
3     f[0] = f[1] = 1; // 0的阶乘为1
4     for (int i = 2; i <= n; i++) f[i] = f[i - 1] * i;
5 }
6 string str;
7 int kangtuo() {
8     int ans = 1; // 注意，因为 12345 是算作0开始计算的，最后结果要把12345看作是第一个
9     int len = str.length();
10    for (int i = 0; i < len; i++) {
11        int tmp = 0; // 用来计数的
12        // 计算str[i]是第几大的数，或者说计算有几个比他小的数
13        for (int j = i + 1; j < len; j++)
14            if (str[i] > str[j]) tmp++;
15        ans += tmp * f[len - i - 1];
16    }
17    return ans;
18 }
19 int main() {
20     jie_cheng(10);
21     string str = "52413";
22     cout << kangtuo() << endl;
23 }
```

5.16.2 正向展开树状数组解

给定一个全排列，求出它是 $1 \sim n$ 所有全排列的第几个，答案对 998244353 取模。

答案就是 $\sum_{i=1}^n res_{a_i} (n-i)!$ 。 res_x 表示剩下的比 x 小的数字的数量，通过**树状数组**处理。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define LL long long
4 const int mod = 998244353, N = 1e6 + 10;
5 LL fact[N];
6 struct fwt{
7     LL n;
8     vector<LL> a;
9     fwt(LL n) : n(n), a(n + 1) {}
10    LL sum(LL x){
11        LL res = 0;
12        for (; x; x -= x & -x)
13            res += a[x];
14        return res;
15    }
16    void add(LL x, LL k){
17        for (; x <= n; x += x & -x)
18            a[x] += k;
19    }
20    LL query(LL x, LL y){
21        return sum(y) - sum(x - 1);
22    }
23};
```

```

24 int main(){
25     ios::sync_with_stdio(false);cin.tie(0);
26     LL n;
27     cin >> n;
28     fwt a(n);
29     fact[0] = 1;
30     for (int i = 1; i <= n; i ++ ){
31         fact[i] = fact[i - 1] * i % mod;
32         a.add(i, 1);
33     }
34     LL ans = 0;
35     for (int i = 1; i <= n; i ++ ){
36         LL x;
37         cin >> x;
38         ans = (ans + a.query(1, x - 1) * fact[n - i] % mod ) % mod;
39         a.add(x, -1);
40     }
41     cout << (ans + 1) % mod << "\n";
42     return 0;
43 }
```

5.16.3 逆向还原

```

1 string str;
2 int kangtuo(){
3     int ans = 1; //注意，因为 12345 是算作0开始计算的，最后结果要把12345看作是第一个
4     int len = str.length();
5     for(int i = 0; i < len; i++){
6         int tmp = 0;//用来计数的
7         for(int j = i + 1; j < len; j++){
8             if(str[i] > str[j]) tmp++;
9             //计算str[i]是第几大的数，或者说计算有几个比他小的数
10        }
11        ans += tmp * f[len - i - 1];
12    }
13    return ans;
14 }
15 int main(){
16     jie_cheng(10);
17     string str = "52413";
18     cout<<kangtuo()<<endl;
19 }
```

5.17 Min25 筛

求解 $1 - N$ 的质数和，其中 $N \leq 10^{10}$ 。

```

1 namespace min25{
2     const int N = 1000000 + 10;
3     int prime[N], id1[N], id2[N], flag[N], ncnt, m;
4     LL g[N], sum[N], a[N], T;
5     LL n;
6     LL mod;
7     inline LL ps(LL n,LL k) {LL r=1;for(;k;k>>=1){if(k&1)r=r*n%mod;n=n*n%mod;}return
8     r;}
9     void finit(){ // 最开始清0
10         memset(g, 0, sizeof(g));
11         memset(a, 0, sizeof(a));
12         memset(sum, 0, sizeof(sum));
13         memset(prime, 0, sizeof(prime));
14         memset(id1, 0, sizeof(id1));
```

```

14     memset(id2, 0, sizeof(id2));
15     memset(flag, 0, sizeof(flag));
16     ncnt = m = 0;
17 }
18 int ID(LL x) {
19     return x <= T ? id1[x] : id2[n / x];
20 }
21
22 LL calc(LL x) {
23     return x * (x + 1) / 2 - 1;
24 }
25
26 LL init(LL x) {
27     T = sqrt(x + 0.5);
28     for (int i = 2; i <= T; i++) {
29         if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt - 1] + i;
30         for (int j = 1; j <= ncnt && i * prime[j] <= T; j++) {
31             flag[i * prime[j]] = 1;
32             if (i % prime[j] == 0) break;
33         }
34     }
35     for (LL l = 1; l <= x; l = x / (x / l) + 1) {
36         a[++m] = x / l;
37         if (a[m] <= T) id1[a[m]] = m; else id2[x / a[m]] = m;
38         g[m] = calc(a[m]);
39     }
40     for (int i = 1; i <= ncnt; i++)
41         for (int j = 1; j <= m && (LL) prime[i] * prime[i] <= a[j]; j++)
42             g[j] = g[j] - (LL) prime[i] * (g[ID(a[j] / prime[i])] - sum[i - 1]);
43 }
44 LL solve(LL x) {
45     if (x <= 1) return x;
46     return n = x, init(n), g[ID(n)];
47 }
48 }
49
50 using namespace min25;
51
52 int main() {
53     // while (1) {
54     int tt;
55     scanf("%d", &tt);
56     while (tt--) {
57         finit();
58         scanf("%lld%lld", &n, &mod);
59         LL ans = (n + 3) % mod * n % mod * ps(2, mod - 2) % mod + solve(n + 1) -
60         4;
61         // cout << solve(n) << endl;
62         // ans = (ans + mod) % mod;
63         ans = (ans + mod) % mod;
64         printf("%lld\n", ans);
65     }
66     // }
67 }

```

5.18 矩阵四则运算

封装来自。矩阵乘法复杂度 $\mathcal{O}(N^3)$ 。

```

1 const int SIZE = 2;
2 struct Matrix {

```

```

3  ll M[SIZE + 5][SIZE + 5];
4  void clear() { memset(M, 0, sizeof(M)); }
5  void reset() { //初始化
6      clear();
7      for (int i = 1; i <= SIZE; ++i) M[i][i] = 1;
8  }
9  Matrix friend operator*(const Matrix &A, const Matrix &B) {
10     Matrix Ans;
11     Ans.clear();
12     for (int i = 1; i <= SIZE; ++i)
13         for (int j = 1; j <= SIZE; ++j)
14             for (int k = 1; k <= SIZE; ++k)
15                 Ans.M[i][j] = (Ans.M[i][j] + A.M[i][k] * B.M[k][j]) % mod;
16     return Ans;
17 }
18 Matrix friend operator+(const Matrix &A, const Matrix &B) {
19     Matrix Ans;
20     Ans.clear();
21     for (int i = 1; i <= SIZE; ++i)
22         for (int j = 1; j <= SIZE; ++j)
23             Ans.M[i][j] = (A.M[i][j] + B.M[i][j]) % mod;
24     return Ans;
25 }
26 };
27
28 inline int mypow(LL n, LL k, int p = MOD) {
29     LL r = 1;
30     for (; k; k >>= 1, n = n * n % p) {
31         if (k & 1) r = r * n % p;
32     }
33     return r;
34 }
35 bool ok = 1;
36 Matrix getinv(Matrix a) { //矩阵求逆
37     int n = SIZE, m = SIZE * 2;
38     for (int i = 1; i <= n; i++) a.M[i][i + n] = 1;
39     for (int i = 1; i <= n; i++) {
40         int pos = i;
41         for (int j = i + 1; j <= n; j++)
42             if (abs(a.M[j][i]) > abs(a.M[pos][i])) pos = j;
43         if (i != pos) swap(a.M[i], a.M[pos]);
44         if (!a.M[i][i]) {
45             puts("No Solution");
46             ok = 0;
47         }
48         ll inv = q_pow(a.M[i][i], mod - 2);
49         for (int j = 1; j <= n; j++)
50             if (j != i) {
51                 ll mul = a.M[j][i] * inv % mod;
52                 for (int k = i; k <= m; k++)
53                     a.M[j][k] = ((a.M[j][k] - a.M[i][k] * mul) % mod + mod) % mod;
54             }
55         for (int j = 1; j <= m; j++) a.M[i][j] = a.M[i][j] * inv % mod;
56     }
57     Matrix res;
58     res.clear();
59     for (int i = 1; i <= n; i++)
60         for (int j = 1; j <= m; j++)
61             res.M[i][j] = a.M[i][n + j];
62     return res;
63 }

```

5.19 矩阵快速幂

以 $\mathcal{O}(N^3 \log M)$ 的复杂度计算。

```

1 const int N = 110, mod = 1e9 + 7;
2 LL n, k, a[N][N], b[N][N], t[N][N];
3 void matrixOp(LL y){
4     while (y){
5         if (y & 1){
6             memset(t, 0, sizeof t);
7             for (int i = 1; i <= n; i++)
8                 for (int j = 1; j <= n; j++)
9                     for (int k = 1; k <= n; k++)
10                        t[i][j] = (t[i][j] + (a[i][k] * b[k][j]) % mod) % mod;
11             memcpy(b, t, sizeof t);
12         }
13         y >>= 1;
14         memset(t, 0, sizeof t);
15         for (int i = 1; i <= n; i++)
16             for (int j = 1; j <= n; j++)
17                 for (int k = 1; k <= n; k++)
18                     t[i][j] = (t[i][j] + (a[i][k] * a[k][j]) % mod) % mod;
19         memcpy(a, t, sizeof t);
20     }
21 }
22 int main(){
23     cin >> n >> k;
24     for (int i = 1; i <= n; i++)
25         for (int j = 1; j <= n; j++){
26             cin >> b[i][j];
27             a[i][j] = b[i][j];
28         }
29     matrixOp(k - 1);
30     for (int i = 1; i <= n; i++)
31         for (int j = 1; j <= n; j++)
32             cout << b[i][j] << " \n"[j == n];
33     return 0;
34 }
```

5.20 矩阵加速

```

1 const int mod = 1e9 + 7;
2 LL T, n, t[5][5], a[5][5], b[5][5];
3 void matrixOp(LL y){
4     while (y){
5         if (y & 1){
6             memset(t, 0, sizeof t);
7             for (int i = 1; i <= 3; i++)
8                 for (int j = 1; j <= 1; j++)
9                     for (int k = 1; k <= 3; k++)
10                        t[i][j] = (t[i][j] + (a[i][k] * b[k][j]) % mod) % mod;
11             memcpy(b, t, sizeof t);
12         }
13         y >>= 1;
14         memset(t, 0, sizeof t);
15         for (int i = 1; i <= 3; i++)
16             for (int j = 1; j <= 3; j++)
17                 for (int k = 1; k <= 3; k++)
18                     t[i][j] = (t[i][j] + (a[i][k] * a[k][j]) % mod) % mod;
19         memcpy(a, t, sizeof t);
20     }
}
```

```

21 }
22 void init(){
23     b[1][1] = b[2][1] = b[3][1] = 1;
24     memset(a, 0, sizeof a);
25     a[1][1] = a[2][1] = a[1][3] = a[3][2] = 1;
26 }
27 void solve(){
28     cin >> n;
29     if (n <= 3) cout << "1\n";
30     else{
31         init();
32         matrixQp(n - 3);
33         cout << b[1][1] << "\n";
34     }
35 }
36 int main(){
37     cin >> T;
38     while (T --)
39         solve();
40     return 0;
41 }
42

```

5.21 莫比乌斯函数/反演

$$\text{莫比乌斯函数定义} : \mu(n) = \begin{cases} 1 & n = 1 \\ (-1)^k & n = \prod_{i=1}^k p_i \text{ 且 } p_i \text{ 互质} \\ 0 & \text{else} \end{cases}.$$

莫比乌斯函数性质：对于任意正整数 n 满足 $\sum_{d|n} \mu(d) = \begin{cases} 1 & n = 1 \\ 0 & n \neq 1 \end{cases}$; $\sum_{d|n} \frac{\mu(d)}{d} = \frac{\varphi(n)}{n}$ 。

莫比乌斯反演定义：定义： $F(n)$ 和 $f(n)$ 是定义在非负整数集合上的两个函数，并且满足 $F(n) = \sum_{d|n} f(d)$ ，可得 $f(n) = \sum_{d|n} \mu(d)F(\left\lfloor \frac{n}{d} \right\rfloor)$ 。

```

1 const int N = 5e4 + 10;
2 bool st[N];
3 int mu[N], prime[N], cnt, sum[N];
4 void getMu() {
5     mu[1] = 1;
6     for (int i = 2; i <= N - 10; i++) {
7         if (!st[i]) {
8             prime[++cnt] = i;
9             mu[i] = -1;
10        }
11        for (int j = 1; j <= cnt && i * prime[j] <= N - 10; j++) {
12            st[i * prime[j]] = true;
13            if (i % prime[j] == 0) {
14                mu[i * prime[j]] = 0;
15                break;
16            }
17            mu[i * prime[j]] = -mu[i];
18        }
19    }
20    for (int i = 1; i <= N - 10; i++) {
21        sum[i] = sum[i - 1] + mu[i];
22    }
23 }
24 void solve() {

```

```

25     int n, m, k; cin >> n >> m >> k;
26     n = n / k, m = m / k;
27     if (n < m) swap(n, m);
28     LL ans = 0;
29     for (int i = 1, j = 0; i <= m; i = j + 1) {
30         j = min(n / (n / i), m / (m / i));
31         ans += (LL)(sum[j] - sum[i - 1]) * (n / i) * (m / i);
32     }
33     cout << ans << "\n";
34 }
35 int main() {
36     getMu();
37     int T; cin >> T;
38     while (T--) solve();
39 }
```

5.22 整除（数论）分块

$\left\lfloor \frac{n}{l} \right\rfloor = \left\lfloor \frac{n}{l+1} \right\rfloor = \dots = \left\lfloor \frac{n}{r} \right\rfloor \iff \left\lfloor \frac{n}{l} \right\rfloor \leq \frac{n}{r} < \left\lfloor \frac{n}{l} \right\rfloor + 1$ ，根据不等式左侧，得到
 $r \leq \left\lfloor \frac{n}{\lfloor \frac{n}{l} \rfloor} \right\rfloor$ 。

```

1 void solve() {
2     LL n; cin >> n;
3     LL ans = 0;
4     for (LL i = 1, j; i <= n; i = j + 1) {
5         j = n / (n / i);
6         ans += (LL)(j - i + 1) * (n / i);
7     }
8     cout << ans << "\n";
9 }
10 int main() {
11     int T; cin >> T;
12     while (T--) solve();
13 }
```

5.23 常见结论

5.23.1 球盒模型

[参考链接](#)。给定 n 个小球 m 个盒子。

- 球同，盒不同、不能空

隔板法： N 个小球即一共 $N - 1$ 个空，分成 M 堆即 $M - 1$ 个隔板，答案为 $\binom{n-1}{m-1}$ 。

- 球同，盒不同、能空

隔板法：多出 $M - 1$ 个虚空球，答案为 $\binom{m-1+n}{n}$ 。

- 球同，盒同、能空

$\frac{1}{(1-x)(1-x^2)\dots(1-x^m)}$ 的 x^n 项的系数。动态规划，答案为

$$dp[i][j] = \begin{cases} dp[i][j-1] + dp[i-j][j] & i \geq j \\ dp[i][j-1] & i < j \\ 1 & j == 1 \text{ || } i \leq 1 \end{cases}$$

- 球同，盒同、不能空

$\frac{x^m}{(1-x)(1-x^2)\dots(1-x^m)}$ 的 x^n 项的系数。动态规划，答案为

$$dp[n][m] = \begin{cases} dp[n-m][m] & n \geq m \\ 0 & n < m \end{cases}$$

- 球不同，盒同、不能空

第二类斯特林数 $\text{Stirling2}(n, m)$ ，答案为

$$dp[n][m] = \begin{cases} m * dp[n-1][m] + dp[n-1][m-1] & 1 \leq m < n \\ 1 & 0 \leq n == m \\ 0 & m == 0 \text{ 且 } 1 \leq n \end{cases}$$

- 球不同，盒同、能空

第二类斯特林数之和 $\sum_{i=1}^m \text{Stirling2}(n, m)$ ，答案为 $\sum_{i=0}^m dp[n][i]$ 。

- 球不同，盒不同、不能空

第二类斯特林数乘上 m 的阶乘 $m! \cdot \text{Stirling2}(n, m)$ ，答案为 $dp[n][m] * m!$ 。

- 球不同，盒不同、能空

答案为 m^n 。

```

1 i64 mypow(i64 n, i64 k) { // 复杂度是 log N
2     i64 r = 1;
3     for (; k; k >>= 1, n *= n) {
4         if (k & 1) r *= n;
5     }
6     return r;
7 }
8
9 vector<vector<i64>> comb;
10 void YangHuiTriangle(int n = 60) {
11     comb.resize(n + 1, vector<i64>(n + 1));
12     comb[0][0] = 1;
13     for (int i = 1; i <= n; i++) {
14         comb[i][0] = 1;
15         for (int j = 1; j <= n; j++) {
16             comb[i][j] = comb[i - 1][j] + comb[i - 1][j - 1];
17         }
18     }
19 }
20
21 vector<vector<i64>> S;
22 void Stirling2(int n = 15) {
23     S.resize(n + 1, vector<i64>(n + 1));
24     S[1][1] = 1;
25     for (int i = 2; i <= 15; i++) {
26         for (int j = 1; j <= i; j++) {
27             S[i][j] = S[i - 1][j - 1] + S[i - 1][j] * j;
28         }
    }
}

```

```

29     }
30 }
31
32 vector<vector<i64>> dp;
33 void GeneratingFunction(int n = 15) {
34     dp.resize(n + 1, vector<i64>(n + 1));
35     for (int i = 0; i <= n; i++) {
36         dp[i][1] = 1;
37         for (int j = 2; j <= n; j++) {
38             dp[i][j] = dp[i][j - 1];
39             if (i >= j) dp[i][j] += dp[i - j][j];
40         }
41     }
42 }
43
44 vector<i64> fac;
45 void Fac(int n = 30) {
46     fac.resize(n + 1);
47     fac[0] = 1;
48     for (int i = 1; i <= n; i++) {
49         fac[i] = fac[i - 1] * i;
50     }
51 }
52
53 i64 A(int n, int m) {
54     if (n < 0 || m < 0 || n < m) return 0;
55     return fac[n] / fac[n - m];
56 }
57
58 i64 C(int n, int m) {
59     if (n < 0 || m < 0 || n < m) return 0;
60     return comb[n][m];
61 }
62
63 signed main() {
64     int Task = 1;
65     for (cin >> Task; Task; Task--) {
66         int op, n, m;
67         cin >> op >> n >> m;
68
69         i64 ans = -1;
70         if (op == 1) { // 球同，盒同、能空
71             ans = dp[n][m];
72         } else if (op == 2) { // 球同，盒同、至多放一个
73             ans = (n <= m);
74         } else if (op == 3) { // 球同，盒同、至少放一个
75             ans = (n < m ? 0 : dp[n - m][m]);
76         } else if (op == 4) { // 球同，盒不同、能空
77             ans = C(m - 1 + n, n);
78         } else if (op == 5) { // 球同，盒不同、至多放一个
79             ans = C(m, n);
80         } else if (op == 6) { // 球同，盒不同、至少放一个
81             ans = C(n - 1, m - 1);
82         } else if (op == 7) { // 球不同，盒同、能空
83             ans = accumulate(S[n].begin() + 1, S[n].begin() + m + 1, 0LL);
84         } else if (op == 8) { // 球不同，盒同、至多放一个
85             ans = (n <= m);
86         } else if (op == 9) { // 球不同，盒同、至少放一个
87             ans = S[n][m];
88         } else if (op == 10) { // 球不同，盒不同、能空
89             ans = mypow(m, n);
90         } else if (op == 11) { // 球不同，盒不同、至多放一个
91             ans = A(m, n);
92         } else if (op == 12) { // 球不同，盒不同、至少放一个

```

```

93         ans = fac[m] * S[n][m];
94     }
95     cout << ans << "\n";
96 }
97 }
```

5.23.2 麦乐鸡定理

给定两个互质的数 n, m ，定义 $x = a * n + b * m$ ($a \geq 0, b \geq 0$)，当 $x > n * m - n - m$ 时，该式子恒成立。

5.23.3 抽屉原理（鸽巢原理）

将 $n + 1$ 个物体，划分为 n 组，那么有至少一组有两个（或以上）的物体。

5.23.4 哥德巴赫猜想

任何一个大于 5 的整数都可写成三个质数之和；任何一个大于 2 的偶数都可写成两个素数之和。

5.23.5 除法、取模运算的本质

有公式： $x \div i = \left\lfloor \frac{x}{i} \right\rfloor + x - i \cdot \left\lfloor \frac{x}{i} \right\rfloor$ ， $x \mod i = x - i \cdot \left\lfloor \frac{x}{i} \right\rfloor$ 。

5.23.6 与、或、异或

运算	运算符、数学符号表示	解释
与	&、and	同1出1
或	、or	有1出1
异或	^、⊕、xor	不同出1

一些结论：

对于给定的 X 和序列 $[a_1, a_2, \dots, a_n]$ ，有： $X = (X \& a_1) or (X \& a_2) or \dots or (X \& a_n)$ 。
原理是 *and* 意味着取交集，*or* 意味着取子集。[来源 - 牛客小白月赛49C](#)

5.23.7 调和级数近似公式

```
1 | log(n) + 0.5772156649 + 1.0 / (2 * n)
```

5.23.8 欧拉函数常见性质

- $1 - n$ 中与 n 互质的数之和为 $n * \varphi(n)/2$ 。
- 若 a, b 互质，则 $\varphi(a * b) = \varphi(a) * \varphi(b)$ 。实际上，所有满足这一条件的函数统称为积性函数。
- 若 f 是积性函数，且有 $n = \prod_{i=1}^m p_i^{c_i}$ ，那么 $f(n) = \prod_{i=1}^m f(p_i^{c_i})$ 。
- 若 p 为质数，且满足 $p \mid n$ ，
 - $p^2 \mid n$ ，那么 $\varphi(n) = \varphi(n/p) * p$ 。
 - $p^2 \nmid n$ ，那么 $\varphi(n) = \varphi(n/p) * (p - 1)$ 。
- $\sum_{d|n} \varphi(d) = n$ 。

| 如 $n = 10$ ，则 $d = 10/5/2/1$ ，那么 $10 = \varphi(10) + \varphi(5) + \varphi(2) + \varphi(1)$ 。

$$\bullet \sum_{i=1}^n \gcd(i, n) = \sum_{d|n} \left\lfloor \frac{n}{d} \right\rfloor \varphi(d) \text{ (欧拉反演).}$$

5.23.9 组合数学常见性质

- $k * C_n^k = n * C_{n-1}^{k-1}$;
- $C_k^n * C_m^k = C_m^n * C_{m-n}^{m-k}$;
- $C_n^k + C_n^{k+1} = C_{n+1}^{k+1}$;
- $\sum_{i=0}^n C_n^i = 2^n$;
- $\sum_{k=0}^n (-1)^k * C_n^k = 0$.
- 二项式反演 :
$$\begin{cases} f_n = \sum_{i=0}^n \binom{n}{i} g_i \Leftrightarrow g_n = \sum_{i=0}^n (-1)^{n-i} \binom{n}{i} f_i \\ f_k = \sum_{i=k}^n \binom{i}{k} g_i \Leftrightarrow g_k = \sum_{i=k}^n (-1)^{i-k} \binom{i}{k} f_i \end{cases} ;$$
- $\sum_{i=1}^n i \binom{n}{i} = n * 2^{n-1}$;
- $\sum_{i=1}^n i^2 \binom{n}{i} = n * (n+1) * 2^{n-2}$;
- $\sum_{i=1}^n \frac{1}{i} \binom{n}{i} = \sum_{i=1}^n \frac{1}{i}$;
- $\sum_{i=0}^n \binom{n}{i}^2 = \binom{2n}{n}$;
- 拉格朗日恒等式 : $\sum_{i=1}^n \sum_{j=i+1}^n (a_i b_j - a_j b_i)^2 = (\sum_{i=1}^n a_i)^2 (\sum_{i=1}^n b_i)^2 - (\sum_{i=1}^n a_i b_i)^2$.

5.23.10 范德蒙德卷积公式

在数量为 $n+m$ 的堆中选 k 个元素，和分别在数量为 n 、 m 的堆中选 i 、 $k-i$ 个元素的方案数是相同的，即 $\sum_{i=0}^k \binom{n}{i} \binom{m}{k-i} = \binom{n+m}{k}$;

变体：

- $\sum_{i=0}^k C_{i+n}^i = C_{k+n+1}^k$;
- $\sum_{i=0}^k C_n^i * C_m^i = \sum_{i=0}^k C_n^i * C_m^{m-i} = C_{n+m}^n$.

5.23.11 卡特兰数

是一类奇特的组合数，前几项为 $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862$ 。如遇到以下问题，则直接套用即可。

- 【括号匹配问题】 n 个左括号和 n 个右括号组成的合法括号序列的数量，为 Cat_n 。
- 【进出栈问题】 $1, 2, \dots, n$ 经过一个栈，形成的合法出栈序列的数量，为 Cat_n 。
- 【二叉树生成问题】 n 个节点构成的不同二叉树的数量，为 Cat_n 。
- 【路径数量问题】在平面直角坐标系上，每一步只能向上或向右走，从 $(0, 0)$ 走到 (n, n) ，并且除两个端点外不接触直线 $y = x$ 的路线数量，为 $2Cat_{n-1}$ 。

计算公式： $Cat_n = \frac{C_{2n}^n}{n+1}$ ， $C_n = \frac{C_{n-1} * (4n-2)}{n+1}$ 。

5.23.12 狄利克雷卷积

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

5.23.13 斐波那契数列

通项公式： $F_n = \frac{1}{\sqrt{5}} * \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$ 。

直接结论：

- 卡西尼性质： $F_{n-1} * F_{n+1} - F_n^2 = (-1)^n$ ；
- $F_n^2 + F_{n+1}^2 = F_{2n+1}$ ；
- $F_{n+1}^2 - F_{n-1}^2 = F_{2n}$ （由上一条写两遍相减得到）；
- 若存在序列 $a_0 = 1, a_n = a_{n-1} + a_{n-3} + a_{n-5} + \dots (n \geq 1)$ 则 $a_n = F_n (n \geq 1)$ ；
- 齐肯多夫定理：任何正整数都可以表示成若干个不连续的斐波那契数（ F_2 开始）可以用贪心实现。

求和公式结论：

- 奇数项求和： $F_1 + F_3 + F_5 + \dots + F_{2n-1} = F_{2n}$ ；
- 偶数项求和： $F_2 + F_4 + F_6 + \dots + F_{2n} = F_{2n+1} - 1$ ；
- 平方和： $F_1^2 + F_2^2 + F_3^2 + \dots + F_n^2 = F_n * F_{n+1}$ ；
- $F_1 + 2F_2 + 3F_3 + \dots + nF_n = nF_{n+2} - F_{n+3} + 2$ ；
- $-F_1 + F_2 - F_3 + \dots + (-1)^n F_n = (-1)^n (F_{n+1} - F_n) + 1$ ；
- $F_{2n-2m-2}(F_{2n} + F_{2n+2}) = F_{2m+2} + F_{4n-2m}$ 。

数论结论：

- $F_a | F_b \Leftrightarrow a | b$ ；
- $\gcd(F_a, F_b) = F_{\gcd(a,b)}$ ；
- 当 p 为 $5k \pm 1$ 型素数时， $\begin{cases} F_{p-1} \equiv 0 \pmod{p} \\ F_p \equiv 1 \pmod{p} \\ F_{p+1} \equiv 1 \pmod{p} \end{cases}$ ；
- 当 p 为 $5k \pm 2$ 型素数时， $\begin{cases} F_{p-1} \equiv 1 \pmod{p} \\ F_p \equiv -1 \pmod{p} \\ F_{p+1} \equiv 0 \pmod{p} \end{cases}$ ；
- $F(n) \% m$ 的周期 $\leq 6m$ （ $m = 2 \times 5^k$ 时取到等号）；
- 既是斐波那契数又是平方数的有且仅有 1, 144。

5.23.14 杂

- 负数取模得到的是负数，如果要用 0/1 判断的话请取绝对值；
- 辗转相除法原式为 $\gcd(x, y) = \gcd(x, y - x)$ ，推广到 N 项为 $\gcd(a_1, a_2, \dots, a_N) = \gcd(a_1, a_2 - a_1, \dots, a_N - a_{N-1})$ ，
该推论在“四则运算后 gcd”这类题中有特殊意义，如求解 $\gcd(a_1 + X, a_2 + X, \dots, a_N + X)$ 时 See；

- 以下式子成立： $\gcd(a, m) = \frac{\gcd(a+x, m)}{m} \Leftrightarrow \gcd(a, m) = \gcd(x, m)$ 。求解上式满足条件的 x 的数量即为求比 $\frac{m}{\gcd(a, m)}$ 小且与其互质的数的个数，即用欧拉函数求解 $\varphi\left(\frac{m}{\gcd(a, m)}\right)$ 。
- 已知序列 a ，定义集合 $S = \{a_i \cdot a_j \mid i < j\}$ ，现在要求解 $\gcd(S)$ ，即为求解 $\gcd(a_j, \gcd(a_i \mid i < j))$ ，换句话说，即为求解后缀 gcd。
- 连续四个数互质的情况如下，当 n 为奇数时， $n, n-1, n-2$ 一定互质；而当 n 为偶数时，
 $\begin{cases} n, n-1, n-3 \text{互质} & \gcd(n, n-3) = 1 \text{ 时} \\ n-1, n-2, n-3 \text{互质} & \gcd(n, n-3) \neq 1 \text{ 时} \end{cases}$ [See](#)；
- 由 $a \bmod b = (b+a) \bmod b = (2 \cdot b + a) \bmod b = \dots = (K \cdot b + a) \bmod b$ 可以推广得到 $(a \bmod b) \bmod c = ((K \cdot bc + a) \bmod b) \bmod c$ ，由此可以得到一个 bc 的答案周期[See](#)；
- 对于长度为 $2 \cdot N$ 的数列 a ，将其任意均分为两个长度为 N 的数列 p, q ，随后对 p 非递减排序、对 q 非递增排序，定义 $f(p, q) = \sum_{i=1}^n |p_i - q_i|$ ，那么答案为 a 数列前 N 大的数之和减去前 N 小的数之和[See](#)。
- 令 $\begin{cases} X = a + b \\ Y = a \oplus b \end{cases}$ ，如果该式子有解，那么存在前提条件 $\begin{cases} X \geq Y \\ X, Y \text{ 同奇偶} \end{cases}$ ；进一步，此时最小的 a 的取值为 $\frac{X - Y}{2}$ [See](#)。
 然而，上方方程并不总是有解的，只有当变量增加到三个时，才一定有解，即：在保证上方前提条件成立的情况下，求解 $\begin{cases} X = a + b + c \\ Y = a \oplus b \oplus c \end{cases}$ ，则一定存在一组解 $\{\frac{X - Y}{2}, \frac{X - Y}{2}, Y\}$ [See](#)。
- 已知序列 p 是由序列 a_1 、序列 a_2 、……、序列 a_n 合并而成，且合并过程中各序列内元素相对顺序不变，记 $T(p)$ 是 p 序列的最大前缀和，则 $T(p) = \sum_{i=1}^n T(a_i)$ [See](#)。
 • $x + y = x|y + x\&y$ ，对于两个数字 x 和 y ，如果将 x 变为 $x|y$ ，同时将 y 变为 $x\&y$ ，那么在本质上即将 x 二进制模式下的全部 1 移动到了 y 的对应的位置上[See](#)。
 • 一个正整数 x 异或、加上另一个正整数 y 后奇偶性不发生变化： $a + b \equiv a \oplus b \pmod{2}$ [See](#)。

5.24 常见例题

题意：将 1 至 N 的每个数字分组，使得每一组的数字之和均为质数。输出每一个数字所在的组别，且要求分出的组数最少 [See](#)。

考察哥德巴赫猜想，记全部数字之和为 S ，分类讨论如下：

- 为 S 质数时，只需要分入同一组；
- 当 S 为偶数时，由猜想可知一定能分成两个质数，可以证明其中较小的那个一定小于 N ，暴力枚举分组；
- 当 $S - 2$ 为质数时，特殊判断出答案；
- 其余情况一定能被分成三组，其中 3 单独成组， $S - 3$ 后成为偶数，重复讨论二的过程即可。

题意：给定一个长度为 n 的数组，定义这个数组是 BAD 的，当且仅当可以把数组分成两个子序列，这两个子序列的元素之和相等。现在你需要删除最少的元素，使得删除后的数组不是 BAD 的。

最少删除一个元素——如果原数组存在奇数，则直接删除这个奇数即可；反之，我们发现，对数列同除以一个数不影响计算，故我们只需要找到最大的满足 $2^k \mid a_i$ 成立的 2^k ，随后将全部的 a_i 变为 $\frac{a_i}{2^k}$ ，此时一定有一个奇数（换句话说，我们可以对原数列的每一个元素不断的除以 2 直到出现奇数为止），删除这个奇数即可 [See](#)。

题意：设当前有一个数字为 x ，减去、加上最少的数字使得其能被 k 整除。

最少减去 $x \bmod k$ 这个很好想；最少加上 $\left(\left\lceil \frac{x}{k} \right\rceil * k\right) \bmod k$ 也比较好想，但是更简便的方法为加上 $k - x \bmod k$ ，这个式子等价于前面这一坨。

题意：给定一个整数 n ，用恰好 k 个 2 的幂次数之和表示它。例如： $n = 9, k = 4$ ，答案为 $1 + 2 + 2 + 4$ 。

结论1： k 合法当且仅当 `__builtin_popcountll(n) <= k && k <= n`，显然。

结论2： $2^{k+1} = 2 \cdot 2^k$ ，所以我们可以将二进制位看作是数组，然后从高位向低位推，一个高位等于两个低位，直到数组之和恰好等于 k ，随后依次输出即可。举例说明， $\{1, 0, 0, 1\} \rightarrow \{0, 2, 0, 1\} \rightarrow \{0, 1, 2, 1\}$ ，即答案为 0 个 2^3 、1 个 2^2 、……。

```

1 signed main() {
2     int n, k;
3     cin >> n >> k;
4
5     int cnt = __builtin_popcountll(n);
6
7     if (k < cnt || n < k) {
8         cout << "NO\n";
9         return 0;
10    }
11    cout << "YES\n";
12
13    vector<int> num;
14    while (n) {
15        num.push_back(n % 2);
16        n /= 2;
17    }
18
19    for (int i = num.size() - 1; i > 0; i--) {
20        int p = min(k - cnt, num[i]);
21        num[i] -= p;
22        num[i - 1] += 2 * p;
23        cnt += p;
24    }
25
26    for (int i = 0; i < num.size(); i++) {
27        for (int j = 1; j <= num[i]; j++) {
28            cout << (1LL << i) << " ";
29        }
30    }
31 }
```

题意： n 个取值在 $[0, k)$ 之间的数之和为 m 的方案数

答案为 $\sum_{i=0}^n -1^i \cdot \binom{n}{i} \cdot \binom{m - i \cdot k + n - 1}{n - 1}$ [See1](#) [See2](#)。

```

1 | Z clac(int n, int k, int m) {
2 |     Z ans = 0;
3 |     ans += C(n, i) * C(m - i * k + n - 1, n - 1) * pow(-1, i);
4 |
5 |     return ans;
6 |

```

¹ 先考虑没有 k 的限制，那么即球盒模型： m 个球放入 n 个盒子，球同、盒子不同、能空。使用隔板法得到公式： $C(m+n-1, n-1)$ ；² 下面加上取值范围后进一步考虑：假设现在 n 个数之和为 $m-k$ ，运用上述隔板法可得公式： $C(m-k+n-1, n-1)$ ；³ 随后，选择任意一个数字，将其加上 k ，这样，这个数字一定不满足条件，选法为： $C(n, 1)$ ；⁴ 此时，至少有一个数字是不满足条件的，按照一般流程，到这里， $C(m+n-1, n-1) - C(n, 1) * C(m-k+n-1, n-1)$ 即是答案；但是，这样的操作会导致重复的部分，所以这里要使用容斥原理将重复部分去除（关于为什么会重复，试比较概率论中的加法公式）。

/END/

6 几何

6.1 库实数类实现（双精度）

```

1 using Real = int;
2 using Point = complex<Real>;
3
4 Real cross(const Point &a, const Point &b) {
5     return (conj(a) * b).imag();
6 }
7 Real dot(const Point &a, const Point &b) {
8     return (conj(a) * b).real();
9 }
```

6.2 平面几何必要初始化

6.2.1 字符串读入浮点数

```

1 const int Knum = 4;
2 int read(int k = Knum) {
3     string s;
4     cin >> s;
5
6     int num = 0;
7     int it = s.find('.');
8     if (it != -1) { // 存在小数点
9         num = s.size() - it - 1; // 计算小数位数
10        s.erase(s.begin() + it); // 删除小数点
11    }
12    for (int i = 1; i <= k - num; i++) { // 补全小数位数
13        s += '0';
14    }
15    return stoi(s);
16 }
```

6.2.2 预置函数

```

1 using ld = long double;
2 const ld PI = acos(-1);
3 const ld EPS = 1e-7;
4 const ld INF = numeric_limits<ld>::max();
5 #define cc(x) cout << fixed << setprecision(x);
6
7 ld fgcd(ld x, ld y) { // 实数域gcd
8     return abs(y) < EPS ? abs(x) : fgcd(y, fmod(x, y));
9 }
10 template<class T, class S> bool equal(T x, S y) {
11     return -EPS < x - y && x - y < EPS;
12 }
13 template<class T> int sign(T x) {
14     if (-EPS < x && x < EPS) return 0;
15     return x < 0 ? -1 : 1;
16 }
```

6.2.3 点线封装

```

1 template<class T> struct Point { // 在C++17下使用 emplace_back 绑定可能会导致CE !
2     T x, y;
3     Point(T x_ = 0, T y_ = 0) : x(x_), y(y_) {} // 初始化
4     template<class U> operator Point<U>() { // 自动类型匹配
5         return Point<U>(U(x), U(y));
6     }
7     Point &operator+=(Point p) & { return x += p.x, y += p.y, *this; }
8     Point &operator+=(T t) & { return x += t, y += t, *this; }
9     Point &operator-=(Point p) & { return x -= p.x, y -= p.y, *this; }
10    Point &operator-=(T t) & { return x -= t, y -= t, *this; }
11    Point &operator*=(T t) & { return x *= t, y *= t, *this; }
12    Point &operator/=(T t) & { return x /= t, y /= t, *this; }
13    Point operator-() const { return Point(-x, -y); }
14    friend Point operator+(Point a, Point b) { return a += b; }
15    friend Point operator+(Point a, T b) { return a += b; }
16    friend Point operator-(Point a, Point b) { return a -= b; }
17    friend Point operator-(Point a, T b) { return a -= b; }
18    friend Point operator*(Point a, T b) { return a *= b; }
19    friend Point operator*(T a, Point b) { return b *= a; }
20    friend Point operator/(Point a, T b) { return a /= b; }
21    friend bool operator<(Point a, Point b) {
22        return equal(a.x, b.x) ? a.y < b.y - EPS : a.x < b.x - EPS;
23    }
24    friend bool operator>(Point a, Point b) { return b < a; }
25    friend bool operator==(Point a, Point b) { return !(a < b) && !(b < a); }
26    friend bool operator!=(Point a, Point b) { return a < b || b < a; }
27    friend auto &operator>>(istream &is, Point &p) {
28        return is >> p.x >> p.y;
29    }
30    friend auto &operator<<(ostream &os, Point p) {
31        return os << "(" << p.x << ", " << p.y << ")";
32    }
33};
34 template<class T> struct Line {
35     Point<T> a, b;
36     Line(Point<T> a_ = Point<T>(), Point<T> b_ = Point<T>()) : a(a_), b(b_) {}
37     template<class U> operator Line<U>() { // 自动类型匹配
38         return Line<U>(Point<U>(a), Point<U>(b));
39     }
40     friend auto &operator<<(ostream &os, Line l) {
41         return os << "<" << l.a << ", " << l.b << ">";
42     }
43 };

```

6.2.4 叉乘

定义公式 $a \times b = |a||b| \sin \theta$ 。

```

1 template<class T> T cross(Point<T> a, Point<T> b) { // 叉乘
2     return a.x * b.y - a.y * b.x;
3 }
4 template<class T> T cross(Point<T> p1, Point<T> p2, Point<T> p0) { // 叉乘 (p1 - p0) x
5     (p2 - p0);
6     return cross(p1 - p0, p2 - p0);
7 }

```

6.2.5 点乘

定义公式 $a \times b = |a||b| \cos \theta$ 。

```

1 template<class T> T dot(Point<T> a, Point<T> b) { // 点乘
2     return a.x * b.x + a.y * b.y;
3 }
4 template<class T> T dot(Point<T> p1, Point<T> p2, Point<T> p0) { // 点乘 (p1 - p0) *
(p2 - p0);
5     return dot(p1 - p0, p2 - p0);
6 }
```

6.2.6 欧几里得距离公式

最常用的距离公式。需要注意，开根号会丢失精度，如无强制要求，先不要开根号，留到最后一步一起开。

```

1 template <class T> ld dis(T x1, T y1, T x2, T y2) {
2     ld val = (x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2);
3     return sqrt(val);
4 }
5 template <class T> ld dis(Point<T> a, Point<T> b) {
6     return dis(a.x, a.y, b.x, b.y);
7 }
```

6.2.7 曼哈顿距离公式

```

1 template <class T> T dis1(Point<T> p1, Point<T> p2) { // 曼哈顿距离公式
2     return abs(p1.x - p2.x) + abs(p1.y - p2.y);
3 }
```

6.2.8 将向量转换为单位向量

```

1 Point<ld> standardize(Point<ld> vec) { // 转换为单位向量
2     return vec / sqrt(vec.x * vec.x + vec.y * vec.y);
3 }
```

6.2.9 向量旋转

将当前向量移动至原点后顺时针旋转 90° ，即获取垂直于当前向量的、起点为原点的向量。在计算垂线时非常有用。例如，要想获取点 a 绕点 o 顺时针旋转 90° 后的点，可以这样书写代码：`auto ans = o + rotate(o, a);`；如果是逆时针旋转，那么只需更改符号即可：`auto ans = o - rotate(o, a);`。

```

1 template<class T> Point<T> rotate(Point<T> p1, Point<T> p2) { // 旋转
2     Point<T> vec = p1 - p2;
3     return {-vec.y, vec.x};
4 }
```

6.3 平面角度与弧度

6.3.1 弧度角度相互转换

```

1 ld toDeg(ld x) { // 弧度转角度
2     return x * 180 / PI;
3 }
4 ld toArc(ld x) { // 角度转弧度
```

```

5 |     return PI / 180 * x;
6 }

```

6.3.2 正弦定理

$$\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R, \text{ 其中 } R \text{ 为三角形外接圆半径};$$

6.3.3 余弦定理 (已知三角形三边, 求角)

$$\cos C = \frac{a^2 + b^2 - c^2}{2ab}, \cos B = \frac{a^2 + c^2 - b^2}{2ac}, \cos A = \frac{b^2 + c^2 - a^2}{2bc}。可以借此推导出三角形$$

面积公式 $S_{\triangle ABC} = \frac{ab \cdot \sin C}{2} = \frac{bc \cdot \sin A}{2} = \frac{ac \cdot \sin B}{2}$ 。

注意，计算格式是：由 b, c, a 三边求 $\angle A$ ；由 a, c, b 三边求 $\angle B$ ；由 a, b, c 三边求 $\angle C$ 。

```

1 | ld angle(ld a, ld b, ld c) { // 余弦定理
2 |     ld val = acos((a * a + b * b - c * c) / (2.0 * a * b)); // 计算弧度
3 |     return val;
4 }

```

6.3.4 求两向量的夹角

能够计算 $[0^\circ, 180^\circ]$ 区间的角度。

```

1 | ld angle(Point<ld> a, Point<ld> b) {
2 |     ld val = abs(cross(a, b));
3 |     return abs(atan2(val, a.x * b.x + a.y * b.y));
4 }

```

6.3.5 向量旋转任意角度

$$\text{逆时针旋转, 转换公式: } \begin{cases} x' = x \cos \theta - y \sin \theta \\ y' = x \sin \theta + y \cos \theta \end{cases}$$

```

1 | Point<ld> rotate(Point<ld> p, ld rad) {
2 |     return {p.x * cos(rad) - p.y * sin(rad), p.x * sin(rad) + p.y * cos(rad)};
3 }

```

6.3.6 点绕点旋转任意角度

$$\text{逆时针旋转, 转换公式: } \begin{cases} x' = (x_0 - x_1) \cos \theta + (y_0 - y_1) \sin \theta + x_1 \\ y' = (x_1 - x_0) \sin \theta + (y_0 - y_1) \cos \theta + y_1 \end{cases}$$

```

1 | Point<ld> rotate(Point<ld> a, Point<ld> b, ld rad) {
2 |     ld x = (a.x - b.x) * cos(rad) + (a.y - b.y) * sin(rad) + b.x;
3 |     ld y = (b.x - a.x) * sin(rad) + (a.y - b.y) * cos(rad) + b.y;
4 |     return {x, y};
5 }

```

6.4 平面点线相关

6.4.1 点是否在直线上（三点是否共线）

```

1 template<class T> bool onLine(Point<T> a, Point<T> b, Point<T> c) {
2     return sign(cross(b, a, c)) == 0;
3 }
4 template<class T> bool onLine(Point<T> p, Line<T> l) {
5     return onLine(p, l.a, l.b);
6 }
```

6.4.2 点是否在向量（直线）左侧

需要注意，向量的方向会影响答案；点在向量上时不视为在左侧。

```

1 template<class T> bool pointOnLineLeft(Pt p, Lt l) {
2     return cross(l.b, p, l.a) > 0;
3 }
```

6.4.3 两点是否在直线同侧/异侧

```

1 template<class T> bool pointOnLineSide(Pt p1, Pt p2, Lt vec) {
2     T val = cross(p1, vec.a, vec.b) * cross(p2, vec.a, vec.b);
3     return sign(val) == 1;
4 }
5 template<class T> bool pointNotOnLineSide(Pt p1, Pt p2, Lt vec) {
6     T val = cross(p1, vec.a, vec.b) * cross(p2, vec.a, vec.b);
7     return sign(val) == -1;
8 }
```

6.4.4 两直线相交交点

在使用前需要先判断直线是否平行。

```

1 Pd lineIntersection(Ld l1, Ld l2) {
2     ld val = cross(l2.b - l2.a, l1.a - l2.a) / cross(l2.b - l2.a, l1.a - l1.b);
3     return l1.a + (l1.b - l1.a) * val;
4 }
```

6.4.5 两直线是否平行/垂直/相同

```

1 template<class T> bool lineParallel(Lt p1, Lt p2) {
2     return sign(cross(p1.a - p1.b, p2.a - p2.b)) == 0;
3 }
4 template<class T> bool lineVertical(Lt p1, Lt p2) {
5     return sign(dot(p1.a - p1.b, p2.a - p2.b)) == 0;
6 }
7 template<class T> bool same(Line<T> l1, Line<T> l2) {
8     return lineParallel(Line{l1.a, l2.b}, {l1.b, l2.a}) &&
9         lineParallel(Line{l1.a, l2.a}, {l1.b, l2.b}) && lineParallel(l1, l2);
10 }
```

6.4.6 点到直线的最近距离与最近点

```

1 pair<Pd, ld> pointToLine(Pd p, Ld l) {
2     Pd ans = lineIntersection({p, p + rotate(l.a, l.b)}, l);
3     return {ans, dis(p, ans)};
4 }
```

如果只需要计算最近距离，下方的写法可以减少书写的代码量，效果一致。

```

1 template<class T> ld disPointToLine(Pt p, Lt l) {
2     ld ans = cross(p, l.a, l.b);
3     return abs(ans) / dis(l.a, l.b); // 面积除以底边长
4 }
```

6.4.7 点是否在线段上

```

1 template<class T> bool pointOnSegment(Pt p, Lt l) { // 端点也算作在直线上
2     return sign(cross(p, l.a, l.b)) == 0 && min(l.a.x, l.b.x) <= p.x && p.x <=
3         max(l.a.x, l.b.x) &&
4             min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y, l.b.y);
5 }
6 template<class T> bool pointOnSegment(Pt p, Lt l) { // 端点不算
7     return pointOnSegment(p, l) && min(l.a.x, l.b.x) < p.x && p.x < max(l.a.x, l.b.x)
8     &&
9         min(l.a.y, l.b.y) < p.y && p.y < max(l.a.y, l.b.y);
10 }
```

6.4.8 点到线段的最近距离与最近点

```

1 pair<Pd, ld> pointToSegment(Pd p, Ld l) {
2     if (sign(dot(p, l.b, l.a)) == -1) { // 特判到两端点的距离
3         return {l.a, dis(p, l.a)};
4     } else if (sign(dot(p, l.a, l.b)) == -1) {
5         return {l.b, dis(p, l.b)};
6     }
7     return pointToLine(p, l);
8 }
```

6.4.9 点在直线上的投影点（垂足）

```

1 Pd project(Pd p, Ld l) { // 投影
2     Pd vec = l.b - l.a;
3     ld r = dot(vec, p - l.a) / (vec.x * vec.x + vec.y * vec.y);
4     return l.a + vec * r;
5 }
```

6.4.10 线段的中垂线

```

1 template<class T> Lt midSegment(Lt l) {
2     Pt mid = (l.a + l.b) / 2; // 线段中点
3     return {mid, mid + rotate(l.a, l.b)};
4 }
```

6.4.11 两线段是否相交及交点

该扩展版可以同时返回相交状态和交点，分为四种情况：0 代表不相交；1 代表普通相交；2 代表重叠（交于两个点）；3 代表相交于端点。需要注意，部分运算可能会使用到直线求交点，此时务必保证变量类型为浮点数！

```

1 template<class T> tuple<int, Pt, Pt> segmentIntersection(Lt l1, Lt l2) {
2     auto [s1, e1] = l1;
3     auto [s2, e2] = l2;
4     auto A = max(s1.x, e1.x), AA = min(s1.x, e1.x);
5     auto B = max(s1.y, e1.y), BB = min(s1.y, e1.y);
6     auto C = max(s2.x, e2.x), CC = min(s2.x, e2.x);
7     auto D = max(s2.y, e2.y), DD = min(s2.y, e2.y);
8     if (A < CC || C < AA || B < DD || D < BB) {
9         return {0, {}, {}};
10    }
11    if (sign(cross(e1 - s1, e2 - s2)) == 0) {
12        if (sign(cross(s2, e1, s1)) != 0) {
13            return {0, {}, {}};
14        }
15        Pt p1(max(AA, CC), max(BB, DD));
16        Pt p2(min(A, C), min(B, D));
17        if (!pointOnSegment(p1, l1)) {
18            swap(p1.y, p2.y);
19        }
20        if (p1 == p2) {
21            return {3, p1, p2};
22        } else {
23            return {2, p1, p2};
24        }
25    }
26    auto cp1 = cross(s2 - s1, e2 - s1);
27    auto cp2 = cross(s2 - e1, e2 - e1);
28    auto cp3 = cross(s1 - s2, e1 - s2);
29    auto cp4 = cross(s1 - e2, e1 - e2);
30    if (sign(cp1 * cp2) == 1 || sign(cp3 * cp4) == 1) {
31        return {0, {}, {}};
32    }
33    // 使用下方函数时请使用浮点数
34    Pd p = lineIntersection(l1, l2);
35    if (sign(cp1) != 0 && sign(cp2) != 0 && sign(cp3) != 0 && sign(cp4) != 0) {
36        return {1, p, p};
37    } else {
38        return {3, p, p};
39    }
40}

```

如果不要求交点，那么使用快速排斥+跨立实验即可，其中重叠、相交于端点均视为相交。

```

1 template<class T> bool segmentIntersection(Lt l1, Lt l2) {
2     auto [s1, e1] = l1;
3     auto [s2, e2] = l2;
4     auto A = max(s1.x, e1.x), AA = min(s1.x, e1.x);
5     auto B = max(s1.y, e1.y), BB = min(s1.y, e1.y);
6     auto C = max(s2.x, e2.x), CC = min(s2.x, e2.x);
7     auto D = max(s2.y, e2.y), DD = min(s2.y, e2.y);
8     return A >= CC && B >= DD && C >= AA && D >= BB &&
9         sign(cross(s1, s2, e1) * cross(s1, e1, e2)) == 1 &&
10        sign(cross(s2, s1, e2) * cross(s2, e2, e1)) == 1;
11}

```

6.5 平面圆相关（浮点数处理）

6.5.1 点到圆的最近点

同时返回最近点与最近距离。需要注意，当点为圆心时，这样的点有无数个，此时我们视作输入错误，直接返回圆心。

```

1 | pair<Pd, ld> pointToCircle(Pd p, Pd o, ld r) {
2 |     Pd U = o, V = o;
3 |     ld d = dis(p, o);
4 |     if (sign(d) == 0) { // p 为圆心时返回圆心本身
5 |         return {o, 0};
6 |     }
7 |     ld val1 = r * abs(o.x - p.x) / d;
8 |     ld val2 = r * abs(o.y - p.y) / d * ((o.x - p.x) * (o.y - p.y) < 0 ? -1 : 1);
9 |     U.x += val1, U.y += val2;
10 |    V.x -= val1, V.y -= val2;
11 |    if (dis(U, p) < dis(V, p)) {
12 |        return {U, dis(U, p)};
13 |    } else {
14 |        return {V, dis(V, p)};
15 |    }
16 |}
```

6.5.2 根据圆心角获取圆上某点

将圆上最右侧的点以圆心为旋转中心，逆时针旋转 `rad` 度。

```

1 | Point<ld> getPoint(Point<ld> p, ld r, ld rad) {
2 |     return {p.x + cos(rad) * r, p.y + sin(rad) * r};
3 | }
```

6.5.3 直线是否与圆相交及交点

0 代表不相交；1 代表相切；2 代表相交。

```

1 | tuple<int, Pd, Pd> lineCircleCross(Ld l, Pd o, ld r) {
2 |     Pd P = project(o, l);
3 |     ld d = dis(P, o), tmp = r * r - d * d;
4 |     if (sign(tmp) == -1) {
5 |         return {0, {}, {}};
6 |     } else if (sign(tmp) == 0) {
7 |         return {1, P, {}};
8 |     }
9 |     Pd vec = standardize(l.b - l.a) * sqrt(tmp);
10 |    return {2, P + vec, P - vec};
11 |}
```

6.5.4 线段是否与圆相交及交点

0 代表不相交；1 代表相切；2 代表相交于一个点；3 代表相交于两个点。

```

1 | tuple<int, Pd, Pd> segmentCircleCross(Ld l, Pd o, ld r) {
2 |     auto [type, U, V] = lineCircleCross(l, o, r);
3 |     bool f1 = pointOnSegment(U, l), f2 = pointOnSegment(V, l);
4 |     if (type == 1 && f1) {
5 |         return {1, U, {}};
6 |     } else if (type == 2 && f1 && f2) {
7 |         return {3, U, V};
```

```

8 } else if (type == 2 && f1) {
9     return {2, U, {}};
10 } else if (type == 2 && f2) {
11     return {2, V, {}};
12 } else {
13     return {0, {}, {}};
14 }
15 }
```

6.5.5 两圆是否相交及交点

0 代表内含；1 代表相离；2 代表相切；3 代表相交。

```

1 tuple<int, Pd, Pd> circleIntersection(Pd p1, ld r1, Pd p2, ld r2) {
2     ld x1 = p1.x, x2 = p2.x, y1 = p1.y, y2 = p2.y, d = dis(p1, p2);
3     if (sign(abs(r1 - r2) - d) == 1) {
4         return {0, {}, {}};
5     } else if (sign(r1 + r2 - d) == -1) {
6         return {1, {}, {}};
7     }
8     ld a = r1 * (x1 - x2) * 2, b = r1 * (y1 - y2) * 2, c = r2 * r2 - r1 * r1 - d * d;
9     ld p = a * a + b * b, q = -a * c * 2, r = c * c - b * b;
10    ld cosa, sina, cosb, sinb;
11    if (sign(d - (r1 + r2)) == 0 || sign(d - abs(r1 - r2)) == 0) {
12        cosa = -q / p / 2;
13        sina = sqrt(1 - cosa * cosa);
14        Point<ld> p0 = {x1 + r1 * cosa, y1 + r1 * sina};
15        if (sign(dis(p0, p2)) - r2) {
16            p0.y = y1 - r1 * sina;
17        }
18        return {2, p0, p0};
19    } else {
20        ld delta = sqrt(q * q - p * r * 4);
21        cosa = (delta - q) / p / 2;
22        cosb = (-delta - q) / p / 2;
23        sina = sqrt(1 - cosa * cosa);
24        sinb = sqrt(1 - cosb * cosb);
25        Pd ans1 = {x1 + r1 * cosa, y1 + r1 * sina};
26        Pd ans2 = {x1 + r1 * cosb, y1 + r1 * sinb};
27        if (sign(dis(ans1, p1)) - r2) ans1.y = y1 - r1 * sina;
28        if (sign(dis(ans2, p2)) - r2) ans2.y = y1 - r1 * sinb;
29        if (ans1 == ans2) ans1.y = y1 - r1 * sina;
30        return {3, ans1, ans2};
31    }
32 }
```

6.5.6 两圆相交面积

上述所言四种相交情况均可计算，之所以不使用三角形面积计算公式是因为在计算过程中会出现“负数”面积（扇形面积与三角形面积的符号关系会随圆的位置关系发生变化），故公式全部重新推导，这里采用的是扇形面积减去扇形内部的那个三角形的面积。

```

1 ld circleIntersectionArea(Pd p1, ld r1, Pd p2, ld r2) {
2     ld x1 = p1.x, x2 = p2.x, y1 = p1.y, y2 = p2.y, d = dis(p1, p2);
3     if (sign(abs(r1 - r2) - d) >= 0) {
4         return PI * min(r1 * r1, r2 * r2);
5     } else if (sign(r1 + r2 - d) == -1) {
6         return 0;
7     }
8     ld theta1 = angle(r1, dis(p1, p2), r2);
9     ld area1 = r1 * r1 * (theta1 - sin(theta1 * 2) / 2);
10    ld theta2 = angle(r2, dis(p1, p2), r1);
11    ld area2 = r2 * r2 * (theta2 - sin(theta2 * 2) / 2);
12    return area1 + area2;
13 }
```

6.5.7 三点确定一圆

```

1 tuple<int, Pd, ld> getCircle(Pd A, Pd B, Pd C) {
2     if (onLine(A, B, C)) { // 特判三点共线
3         return {0, {}, 0};
4     }
5     Ld l1 = midSegment(Line{A, B});
6     Ld l2 = midSegment(Line{A, C});
7     Pd O = lineIntersection(l1, l2);
8     return {1, O, dis(A, O)};
9 }
```

6.5.8 求解点到圆的切线数量与切点

```

1 pair<int, vector<Point<ld>>> tangent(Point<ld> p, Point<ld> A, ld r) {
2     vector<Point<ld>> ans; // 储存切点
3     Point<ld> u = A - p;
4     ld d = sqrt(dot(u, u));
5     if (d < r) {
6         return {0, {}};
7     } else if (sign(d - r) == 0) { // 点在圆上
8         ans.push_back(u);
9         return {1, ans};
10    } else {
11        ld ang = asin(r / d);
12        ans.push_back(getPoint(A, r, -ang));
13        ans.push_back(getPoint(A, r, ang));
14        return {2, ans};
15    }
16 }
```

6.5.9 求解两圆的内公、外公切线数量与切点

同时返回公切线数量以及每个圆的切点。

```

1 tuple<int, vector<Point<ld>>, vector<Point<ld>>> tangent(Point<ld> A, ld Ar,
2 Point<ld> B, ld Br) {
3     vector<Point<ld>> a, b; // 储存切点
4     if (Ar < Br) {
5         swap(Ar, Br);
6         swap(A, B);
7         swap(a, b);
8     }
9     int d = disEx(A, B), dif = Ar - Br, sum = Ar + Br;
10    if (d < dif * dif) { // 内含，无
```

```

10     return {0, {}, {}};
11 }
12 ld base = atan2(B.y - A.y, B.x - A.x);
13 if (d == 0 && Ar == Br) { // 完全重合，无数条外公切线
14     return {-1, {}, {}};
15 }
16 if (d == dif * dif) { // 内切，1条外公切线
17     a.push_back(getPoint(A, Ar, base));
18     b.push_back(getPoint(B, Br, base));
19     return {1, a, b};
20 }
21 ld ang = acos(dif / sqrt(d));
22 a.push_back(getPoint(A, Ar, base + ang)); // 保底2条外公切线
23 a.push_back(getPoint(A, Ar, base - ang));
24 b.push_back(getPoint(B, Br, base + ang));
25 b.push_back(getPoint(B, Br, base - ang));
26 if (d == sum * sum) { // 外切，多1条内公切线
27     a.push_back(getPoint(A, Ar, base));
28     b.push_back(getPoint(B, Br, base + PI));
29 } else if (d > sum * sum) { // 相离，多2条内公切线
30     ang = acos(sum / sqrt(d));
31     a.push_back(getPoint(A, Ar, base + ang));
32     a.push_back(getPoint(A, Ar, base - ang));
33     b.push_back(getPoint(B, Br, base + ang + PI));
34     b.push_back(getPoint(B, Br, base - ang + PI));
35 }
36 return {a.size(), a, b};
37 }

```

6.6 平面三角形相关（浮点数处理）

6.6.1 三角形面积

```

1 ld area(Point<ld> a, Point<ld> b, Point<ld> c) {
2     return abs(cross(b, c, a)) / 2;
3 }

```

6.6.2 三角形外心

三角形外接圆的圆心，即三角形三边垂直平分线的交点。

```

1 template<class T> Pt center1(Pt p1, Pt p2, Pt p3) { // 外心
2     return lineIntersection(midSegment({p1, p2}), midSegment({p2, p3}));
3 }

```

6.6.3 三角形内心

三角形内切圆的圆心，也是三角形三个内角的角平分线的交点。其到三角形三边的距离相等。

```

1 Pd center2(Pd p1, Pd p2, Pd p3) { // 内心
2 #define atan2(p) atan2(p.y, p.x) // 注意先后顺序
3 Line<ld> U = {p1, {}}, V = {p2, {}};
4 ld m, n, alpha;
5 m = atan2((p2 - p1));
6 n = atan2((p3 - p1));
7 alpha = (m + n) / 2;
8 U.b = {p1.x + cos(alpha), p1.y + sin(alpha)};
9 m = atan2((p1 - p2));
10 n = atan2((p3 - p2));
11 alpha = (m + n) / 2;

```

```

12     V.b = {p2.x + cos(alpha), p2.y + sin(alpha)};
13     return lineIntersection(U, V);
14 }
```

6.6.4 三角形垂心

三角形的三条高线所在直线的交点。锐角三角形的垂心在三角形内；直角三角形的垂心在直角顶点上；钝角三角形的垂心在三角形外。

```

1 Pd center3(Pd p1, Pd p2, Pd p3) { // 垂心
2     Ld U = {p1, p1 + rotate(p2, p3)}; // 垂线
3     Ld V = {p2, p2 + rotate(p1, p3)};
4     return lineIntersection(U, V);
5 }
```

6.7 平面直线方程转换

6.7.1 浮点数计算直线的斜率

一般很少使用到这个函数，因为斜率的取值不可控（例如接近平行于 x, y 轴时）。需要注意，当直线平行于 y 轴时斜率为 `inf`。

```

1 template <class T> ld slope(Pt p1, Pt p2) { // 斜率，注意 inf 的情况
2     return (p1.y - p2.y) / (p1.x - p2.x);
3 }
4 template <class T> ld slope(Lt l) {
5     return slope(l.a, l.b);
6 }
```

6.7.2 分数精确计算直线的斜率

调用分数四则运算精确计算斜率，返回最简分数，只适用于整数计算。

```

1 template<class T> Frac<T> slopeEx(Pt p1, Pt p2) {
2     Frac<T> U = p1.y - p2.y;
3     Frac<T> V = p1.x - p2.x;
4     return U / V; // 调用分数精确计算
5 }
```

6.7.3 两点式转一般式

返回由三个整数构成的方程，在输入较大时可能找不到较小的满足题意的一组整数解。可以处理平行于 x, y 轴、两点共点的情况。

```

1 template<class T> tuple<T, T, T> getfun(Lt p) {
2     T A = p.a.y - p.b.y, B = p.b.x - p.a.x, C = p.a.x * A + p.a.y * B;
3     if (A < 0) { // 符号调整
4         A = -A, B = -B, C = -C;
5     } else if (A == 0) {
6         if (B < 0) {
7             B = -B, C = -C;
8         } else if (B == 0 && C < 0) {
9             C = -C;
10        }
11    }
12    if (A == 0) { // 数值计算
13        if (B == 0) {
14            C = 0; // 共点特判
15        }
16    }
17 }
```

```

15     } else {
16         T g = fgcd(abs(B), abs(C));
17         B /= g, C /= g;
18     }
19 } else if (B == 0) {
20     T g = fgcd(abs(A), abs(C));
21     A /= g, C /= g;
22 } else {
23     T g = fgcd(fgcd(abs(A), abs(B)), abs(C));
24     A /= g, B /= g, C /= g;
25 }
26 return tuple{A, B, C}; // Ax + By = C
27 }
```

6.7.4 一般式转两点式

由于整数点可能很大或者不存在，故直接采用浮点数；如果与 x, y 轴有交点则取交点。可以处理平行于 x, y 轴的情况。

```

1 Line<ld> getfun(int A, int B, int C) { // Ax + By = C
2     ld x1 = 0, y1 = 0, x2 = 0, y2 = 0;
3     if (A && B) { // 正常
4         if (C) {
5             x1 = 0, y1 = 1. * C / B;
6             y2 = 0, x2 = 1. * C / A;
7         } else { // 过原点
8             x1 = 1, y1 = 1. * -A / B;
9             x2 = 0, y2 = 0;
10        }
11    } else if (A && !B) { // 垂直
12        if (C) {
13            y1 = 0, x1 = 1. * C / A;
14            y2 = 1, x2 = x1;
15        } else {
16            x1 = 0, y1 = 1;
17            x2 = 0, y2 = 0;
18        }
19    } else if (!A && B) { // 水平
20        if (C) {
21            x1 = 0, y1 = 1. * C / B;
22            x2 = 1, y2 = y1;
23        } else {
24            x1 = 1, y1 = 0;
25            x2 = 0, y2 = 0;
26        }
27    } else { // 不合法，请特判
28        assert(false);
29    }
30    return {{x1, y1}, {x2, y2}};
31 }
```

6.7.5 抛物线与 x 轴是否相交及交点

0 代表没有交点；1 代表相切；2 代表有两个交点。

```

1 tuple<int, ld, ld> getAns(ld a, ld b, ld c) {
2     ld delta = b * b - a * c * 4;
3     if (delta < 0.) {
4         return {0, 0, 0};
5     }
6     delta = sqrt(delta);
```

```

7  ld ans1 = -(delta + b) / 2 / a;
8  ld ans2 = (delta - b) / 2 / a;
9  if (ans1 > ans2) {
10     swap(ans1, ans2);
11 }
12 if (sign(delta) == 0) {
13     return {1, ans2, 0};
14 }
15 return {2, ans1, ans2};
16 }
```

6.8 平面多边形

6.8.1 两向量构成的平面四边形有向面积

```

1 template<class T> T areaEx(Point<T> p1, Point<T> p2, Point<T> p3) {
2     return cross(b, c, a);
3 }
```

6.8.2 判断四个点能否组成矩形/正方形

可以处理浮点数、共点的情况。返回分为三种情况：2 代表构成正方形；1 代表构成矩形；0 代表其他情况。

```

1 template<class T> int isSquare(vector<Pt> x) {
2     sort(x.begin(), x.end());
3     if (equal(dis(x[0], x[1]), dis(x[2], x[3])) && sign(dis(x[0], x[1])) &&
4         equal(dis(x[0], x[2]), dis(x[1], x[3])) && sign(dis(x[0], x[2])) &&
5         lineParallel(Lt{x[0], x[1]}, Lt{x[2], x[3]}) &&
6         lineParallel(Lt{x[0], x[2]}, Lt{x[1], x[3]}) &&
7         lineVertical(Lt{x[0], x[1]}, Lt{x[0], x[2]})) {
8             return equal(dis(x[0], x[1]), dis(x[0], x[2])) ? 2 : 1;
9         }
10    return 0;
11 }
```

6.8.3 点是否在任意多边形内

射线法判定， t 为穿越次数，当其为奇数时即代表点在多边形内部；返回 2 代表点在多边形边界上。

```

1 template<class T> int pointInPolygon(Point<T> a, vector<Point<T>> p) {
2     int n = p.size();
3     for (int i = 0; i < n; i++) {
4         if (pointOnSegment(a, Line{p[i], p[(i + 1) % n]})) {
5             return 2;
6         }
7     }
8     int t = 0;
9     for (int i = 0; i < n; i++) {
10        auto u = p[i], v = p[(i + 1) % n];
11        if (u.x < a.x && v.x >= a.x && pointOnLineLeft(a, Line{v, u})) {
12            t ^= 1;
13        }
14        if (u.x >= a.x && v.x < a.x && pointOnLineLeft(a, Line{u, v})) {
15            t ^= 1;
16        }
17    }
18    return t == 1;
19 }
```

6.8.4 线段是否在任意多边形内部

```

1 template<class T>
2 bool segmentInPolygon(Line<T> l, vector<Point<T>> p) {
3 // 线段与多边形边界不相交且两端点都在多边形内部
4 #define L(x, y) pointOnLineLeft(x, y)
5     int n = p.size();
6     if (!pointInPolygon(l.a, p)) return false;
7     if (!pointInPolygon(l.b, p)) return false;
8     for (int i = 0; i < n; i++) {
9         auto u = p[i];
10        auto v = p[(i + 1) % n];
11        auto w = p[(i + 2) % n];
12        auto [t, p1, p2] = segmentIntersection(l, Line(u, v));
13        if (t == 1) return false;
14        if (t == 0) continue;
15        if (t == 2) {
16            if (pointOnSegment(v, l) && v != l.a && v != l.b) {
17                if (cross(v - u, w - v) > 0) {
18                    return false;
19                }
20            }
21        } else {
22            if (p1 != u && p1 != v) {
23                if (L(l.a, Line(v, u)) || L(l.b, Line(v, u))) {
24                    return false;
25                }
26            } else if (p1 == v) {
27                if (l.a == v) {
28                    if (L(u, l)) {
29                        if (L(w, l) && L(w, Line(u, v))) {
30                            return false;
31                        }
32                    } else {
33                        if (L(w, l) || L(w, Line(u, v))) {
34                            return false;
35                        }
36                    }
37                } else if (l.b == v) {
38                    if (L(u, Line(l.b, l.a))) {
39                        if (L(w, Line(l.b, l.a)) && L(w, Line(u, v))) {
40                            return false;
41                        }
42                    } else {
43                        if (L(w, Line(l.b, l.a)) || L(w, Line(u, v))) {
44                            return false;
45                        }
46                    }
47                } else {
48                    if (L(u, l)) {
49                        if (L(w, Line(l.b, l.a)) || L(w, Line(u, v))) {
50                            return false;
51                        }
52                    } else {
53                        if (L(w, l) || L(w, Line(u, v))) {
54                            return false;
55                        }
56                    }
57                }
58            }
59        }
60    }
61    return true;

```

62 | }

6.8.5 任意多边形的面积

```

1 template<class T> ld area(vector<Point<T>> P) {
2     int n = P.size();
3     ld ans = 0;
4     for (int i = 0; i < n; i++) {
5         ans += cross(P[i], P[(i + 1) % n]);
6     }
7     return ans / 2.0;
8 }
```

6.8.6 皮克定理

绘制在方格纸上的多边形面积公式可以表示为 $S = n + \frac{s}{2} - 1$ ，其中 n 表示多边形内部的点数、 s 表示多边形边界上的点数。一条线段上的点数为 $\gcd(|x_1 - x_2|, |y_1 - y_2|) + 1$ 。

6.8.7 任意多边形上/内的网格点个数（仅能处理整数）

皮克定理用。

```

1 int onPolygonGrid(vector<Point<int>> p) { // 多边上
2     int n = p.size(), ans = 0;
3     for (int i = 0; i < n; i++) {
4         auto a = p[i], b = p[(i + 1) % n];
5         ans += gcd(abs(a.x - b.x), abs(a.y - b.y));
6     }
7     return ans;
8 }
9 int inPolygonGrid(vector<Point<int>> p) { // 多边形内
10    int n = p.size(), ans = 0;
11    for (int i = 0; i < n; i++) {
12        auto a = p[i], b = p[(i + 1) % n], c = p[(i + 2) % n];
13        ans += b.y * (a.x - c.x);
14    }
15    ans = abs(ans);
16    return (ans - onPolygonGrid(p)) / 2 + 1;
17 }
```

6.9 二维凸包

6.9.1 获取二维静态凸包（Andrew算法）

`flag` 用于判定凸包边上的点、重复的顶点是否要加入到凸包中，为 0 时代表加入凸包（不严格）；为 1 时不加入凸包（严格）。时间复杂度为 $\mathcal{O}(N \log N)$ 。

```

1 template<class T> vector<Point<T>> staticConvexHull(vector<Point<T>> A, int flag =
2 1) {
3     int n = A.size();
4     if (n <= 2) { // 特判
5         return A;
6     }
7     vector<Point<T>> ans(n * 2);
8     sort(A.begin(), A.end());
9     int now = -1;
10    for (int i = 0; i < n; i++) { // 维护下凸包
11        while (now > 0 && cross(A[i], ans[now], ans[now - 1]) <= 0) {
12            now--;
13        }
14    }
15    now++;
16    for (int i = n - 1; i >= 0; i--) {
17        while (now > 0 && cross(A[i], ans[now], ans[now - 1]) <= 0) {
18            now--;
19        }
20    }
21    now--;
22    ans.pop_back();
23    ans.pop_back();
24    return ans;
25 }
```

```

12     }
13     ans[++now] = A[i];
14 }
15 int pre = now;
16 for (int i = n - 2; i >= 0; i--) { // 维护上凸包
17     while (now > pre && cross(A[i], ans[now], ans[now - 1]) <= 0) {
18         now--;
19     }
20     ans[++now] = A[i];
21 }
22 ans.resize(now);
23 return ans;
24 }
```

6.9.2 二维动态凸包

固定为 `int` 型，需要重新书写 `Line` 函数，`cmp` 用于判定边界情况。可以处理如下两个要求：

- 动态插入点 (x, y) 到当前凸包中；
- 判断点 (x, y) 是否在凸包上或是在内部（包括边界）。

```

1 template<class T> bool turnRight(Pt a, Pt b) {
2     return cross(a, b) < 0 || (cross(a, b) == 0 && dot(a, b) < 0);
3 }
4 struct Line {
5     static int cmp;
6     mutable Point<int> a, b;
7     friend bool operator<(Line x, Line y) {
8         return cmp ? x.a < y.a : turnRight(x.b, y.b);
9     }
10    friend auto &operator<<(ostream &os, Line l) {
11        return os << "(" << l.a << ", " << l.b << ")";
12    }
13 };
14
15 int Line::cmp = 1;
16 struct UpperConvexHull : set<Line> {
17     bool contains(const Point<int> &p) const {
18         auto it = lower_bound({p, 0});
19         if (it != end() && it->a == p) return true;
20         if (it != begin() && it != end() && cross(prev(it)->b, p - prev(it)->a) <= 0) {
21             return true;
22         }
23         return false;
24     }
25     void add(const Point<int> &p) {
26         if (contains(p)) return;
27         auto it = lower_bound({p, 0});
28         for (; it != end(); it = erase(it)) {
29             if (turnRight(it->a - p, it->b)) {
30                 break;
31             }
32         }
33         for (; it != begin() && prev(it) != begin(); erase(prev(it))) {
34             if (turnRight(prev(prev(it))->b, p - prev(prev(it))->a)) {
35                 break;
36             }
37         }
38         if (it != begin()) {
39             prev(it)->b = p - prev(it)->a;
40         }
41     }
42 }
```

```

41     if (it == end()) {
42         insert({p, {0, -1}});
43     } else {
44         insert({p, it->a - p});
45     }
46 }
47 };
48 struct ConvexHull {
49     UpperConvexHull up, low;
50     bool empty() const {
51         return up.empty();
52     }
53     bool contains(const Point<int> &p) const {
54         Line::cmp = 1;
55         return up.contains(p) && low.contains(-p);
56     }
57     void add(const Point<int> &p) {
58         Line::cmp = 1;
59         up.add(p);
60         low.add(-p);
61     }
62     bool isIntersect(int A, int B, int C) const {
63         Line::cmp = 0;
64         if (empty()) return false;
65         Point<int> k = {-B, A};
66         if (k.x < 0) k = -k;
67         if (k.x == 0 && k.y < 0) k.y = -k.y;
68         Point<int> P = up.upper_bound({{0, 0}, k})->a;
69         Point<int> Q = -low.upper_bound({{0, 0}, k})->a;
70         return sign(A * P.x + B * P.y - C) * sign(A * Q.x + B * Q.y - C) > 0;
71     }
72     friend ostream &operator<<(ostream &out, const ConvexHull &ch) {
73         for (const auto &line : ch.up) out << "(" << line.a.x << "," << line.a.y << ")";
74         cout << "/";
75         for (const auto &line : ch.low) out << "(" << -line.a.x << "," << -line.a.y << ")";
76         return out;
77     }
78 };

```

6.9.3 点与凸包的位置关系

0 代表点在凸包外面；1 代表在凸壳上；2 代表在凸包内部。

```

1 template<class T> int contains(Point<T> p, vector<Point<T>> A) {
2     int n = A.size();
3     bool in = false;
4     for (int i = 0; i < n; i++) {
5         Point<T> a = A[i] - p, b = A[(i + 1) % n] - p;
6         if (a.y > b.y) {
7             swap(a, b);
8         }
9         if (a.y <= 0 && 0 < b.y && cross(a, b) < 0) {
10            in = !in;
11        }
12        if (cross(a, b) == 0 && dot(a, b) <= 0) {
13            return 1;
14        }
15    }
16    return in ? 2 : 0;
17 }

```

6.9.4 阁可夫斯基和

计算两个凸包合成的大凸包。

```

1 template<class T> vector<Point<T>> minkowski(vector<Point<T>> P1, vector<Point<T>>
2 P2) {
3     int n = P1.size(), m = P2.size();
4     vector<Point<T>> V1(n), V2(m);
5     for (int i = 0; i < n; i++) {
6         V1[i] = P1[(i + 1) % n] - P1[i];
7     }
8     for (int i = 0; i < m; i++) {
9         V2[i] = P2[(i + 1) % m] - P2[i];
10    }
11    vector<Point<T>> ans = {P1.front() + P2.front()};
12    int t = 0, i = 0, j = 0;
13    while (i < n && j < m) {
14        Point<T> val = sign(cross(V1[i], V2[j])) > 0 ? V1[i++] : V2[j++];
15        ans.push_back(ans.back() + val);
16    }
17    while (i < n) ans.push_back(ans.back() + V1[i++]);
18    while (j < m) ans.push_back(ans.back() + V2[j++]);
19    return ans;
}

```

6.9.5 半平面交

计算多条直线左边平面部分的交集。

```

1 template<class T> vector<Point<T>> halfcut(vector<Line<T>> lines) {
2     sort(lines.begin(), lines.end(), [&](auto l1, auto l2) {
3         auto d1 = l1.b - l1.a;
4         auto d2 = l2.b - l2.a;
5         if (sign(d1) != sign(d2)) {
6             return sign(d1) == 1;
7         }
8         return cross(d1, d2) > 0;
9     });
10    deque<Line<T>> ls;
11    deque<Point<T>> ps;
12    for (auto l : lines) {
13        if (ls.empty()) {
14            ls.push_back(l);
15            continue;
16        }
17        while (!ps.empty() && !pointOnLineLeft(ps.back(), l)) {
18            ps.pop_back();
19            ls.pop_back();
20        }
21        while (!ps.empty() && !pointOnLineLeft(ps[0], l)) {
22            ps.pop_front();
23            ls.pop_front();
24        }
25        if (cross(l.b - l.a, ls.back().b - ls.back().a) == 0) {
26            if (dot(l.b - l.a, ls.back().b - ls.back().a) > 0) {
27                if (!pointOnLineLeft(ls.back().a, l)) {
28                    assert(ls.size() == 1);
29                    ls[0] = l;
30                }
31                continue;
32            }
33        }
}

```

```

34     }
35     ps.push_back(lineIntersection(ls.back(), l));
36     ls.push_back(l);
37 }
38 while (!ps.empty() && !pointOnLineLeft(ps.back(), ls[0])) {
39     ps.pop_back();
40     ls.pop_back();
41 }
42 if (ls.size() <= 2) {
43     return {};
44 }
45 ps.push_back(lineIntersection(ls[0], ls.back()));
46 return vector(ps.begin(), ps.end());
47 }

```

6.10 三维几何必要初始化

6.10.1 点线面封装

```

1 struct Point3 {
2     ld x, y, z;
3     Point3(ld x_ = 0, ld y_ = 0, ld z_ = 0) : x(x_), y(y_), z(z_) {}
4     Point3 &operator+=(Point3 p) & {
5         return x += p.x, y += p.y, z += p.z, *this;
6     }
7     Point3 &operator-=(Point3 p) & {
8         return x -= p.x, y -= p.y, z -= p.z, *this;
9     }
10    Point3 &operator*=(Point3 p) & {
11        return x *= p.x, y *= p.y, z *= p.z, *this;
12    }
13    Point3 &operator*=(ld t) & {
14        return x *= t, y *= t, z *= t, *this;
15    }
16    Point3 &operator/=(ld t) & {
17        return x /= t, y /= t, z /= t, *this;
18    }
19    friend Point3 operator+(Point3 a, Point3 b) { return a += b; }
20    friend Point3 operator-(Point3 a, Point3 b) { return a -= b; }
21    friend Point3 operator*(Point3 a, Point3 b) { return a *= b; }
22    friend Point3 operator*(Point3 a, ld b) { return a *= b; }
23    friend Point3 operator*(ld a, Point3 b) { return b *= a; }
24    friend Point3 operator/(Point3 a, ld b) { return a /= b; }
25    friend auto &operator>>(istream &is, Point3 &p) {
26        return is >> p.x >> p.y >> p.z;
27    }
28    friend auto &operator<<(ostream &os, Point3 p) {
29        return os << "(" << p.x << ", " << p.y << ", " << p.z << ")";
30    }
31 };
32 struct Line3 {
33     Point3 a, b;
34 };
35 struct Plane {
36     Point3 u, v, w;
37 };

```

6.10.2 其他函数

```

1 ld len(P3 p) { // 原点到当前点的距离计算
2     return sqrt(p.x * p.x + p.y * p.y + p.z * p.z);
3 }
4 P3 crossEx(P3 a, P3 b) { // 叉乘
5     P3 ans;
6     ans.x = a.y * b.z - a.z * b.y;
7     ans.y = a.z * b.x - a.x * b.z;
8     ans.z = a.x * b.y - a.y * b.x;
9     return ans;
10 }
11 ld cross(P3 a, P3 b) {
12     return len(crossEx(a, b));
13 }
14 ld dot(P3 a, P3 b) { // 点乘
15     return a.x * b.x + a.y * b.y + a.z * b.z;
16 }
17 P3 getVec(Plane s) { // 获取平面法向量
18     return crossEx(s.u - s.v, s.v - s.w);
19 }
20 ld dis(P3 a, P3 b) { // 三维欧几里得距离公式
21     ld val = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + (a.z - b.z) *
22         (a.z - b.z);
23     return sqrt(val);
24 }
25 P3 standardize(P3 vec) { // 将三维向量转换为单位向量
26     return vec / len(vec);
}

```

6.11 三维点线面相关

6.11.1 空间三点是否共线

其中第二个函数是专门用来判断给定的三个点能否构成平面的，因为不共线的三点才能构成平面。

```

1 bool onLine(P3 p1, P3 p2, P3 p3) { // 三点是否共线
2     return sign(cross(p1 - p2, p3 - p2)) == 0;
3 }
4 bool onLine(Plane s) {
5     return onLine(s.u, s.v, s.w);
6 }

```

6.11.2 四点是否共面

```

1 bool onPlane(P3 p1, P3 p2, P3 p3, P3 p4) { // 四点是否共面
2     ld val = dot(getVec({p1, p2, p3}), p4 - p1);
3     return sign(val) == 0;
4 }

```

6.11.3 空间点是否在线段上

```

1 bool pointOnSegment(P3 p, L3 l) {
2     return sign(cross(p - l.a, p - l.b)) == 0 && min(l.a.x, l.b.x) <= p.x &&
3         p.x <= max(l.a.x, l.b.x) && min(l.a.y, l.b.y) <= p.y && p.y <= max(l.a.y,
4             l.b.y) &&
5                 min(l.a.z, l.b.z) <= p.z && p.z <= max(l.a.z, l.b.z);
6 }
7 bool pointOnSegmentEx(P3 p, L3 l) { // pointOnSegment去除端点版
8     return sign(cross(p - l.a, p - l.b)) == 0 && min(l.a.x, l.b.x) < p.x &&
9         p.x < max(l.a.x, l.b.x) && min(l.a.y, l.b.y) < p.y && p.y < max(l.a.y,
10            l.b.y) &&
11                min(l.a.z, l.b.z) < p.z && p.z < max(l.a.z, l.b.z);
12 }
```

6.11.4 空间两点是否在线段同侧

当给定的两点与线段不共面、点在线段上时返回 *false*。

```

1 bool pointOnSegmentSide(P3 p1, P3 p2, L3 l) {
2     if (!onPlane(p1, p2, l.a, l.b)) { // 特判不共面
3         return 0;
4     }
5     ld val = dot(crossEx(l.a - l.b, p1 - l.b), crossEx(l.a - l.b, p2 - l.b));
6     return sign(val) == 1;
7 }
```

6.11.5 两点是否在平面同侧

点在平面上时返回 *false*。

```

1 bool pointOnPlaneSide(P3 p1, P3 p2, Plane s) {
2     ld val = dot(getVec(s), p1 - s.u) * dot(getVec(s), p2 - s.u);
3     return sign(val) == 1;
4 }
```

6.11.6 空间两直线是否平行/垂直

```

1 bool lineParallel(L3 l1, L3 l2) {
2     return sign(cross(l1.a - l1.b, l2.a - l2.b)) == 0;
3 }
4 bool lineVertical(L3 l1, L3 l2) {
5     return sign(dot(l1.a - l1.b, l2.a - l2.b)) == 0;
6 }
```

6.11.7 两平面是否平行/垂直

```

1 bool planeParallel(Plane s1, Plane s2) {
2     ld val = cross(getVec(s1), getVec(s2));
3     return sign(val) == 0;
4 }
5 bool planeVertical(Plane s1, Plane s2) {
6     ld val = dot(getVec(s1), getVec(s2));
7     return sign(val) == 0;
8 }
```

6.11.8 空间两直线是否是同一条

```

1 | bool same(L3 l1, L3 l2) {
2 |     return lineParallel(l1, l2) && lineParallel({l1.a, l2.b}, {l1.b, l2.a});
3 |

```

6.11.9 两平面是否是同一个

```

1 | bool same(Plane s1, Plane s2) {
2 |     return onPlane(s1.u, s2.u, s2.v, s2.w) && onPlane(s1.v, s2.u, s2.v, s2.w) &&
3 |         onPlane(s1.w, s2.u, s2.v, s2.w);
4 |

```

6.11.10 直线是否与平面平行

```

1 | bool linePlaneParallel(L3 l, Plane s) {
2 |     ld val = dot(l.a - l.b, getVec(s));
3 |     return sign(val) == 0;
4 |

```

6.11.11 空间两线段是否相交

```

1 | bool segmentIntersection(L3 l1, L3 l2) { // 重叠、相交于端点均视为相交
2 |     if (!onPlane(l1.a, l1.b, l2.a, l2.b)) { // 特判不共面
3 |         return 0;
4 |     }
5 |     if (!onLine(l1.a, l1.b, l2.a) || !onLine(l1.a, l1.b, l2.b)) {
6 |         return !pointOnSegmentSide(l1.a, l1.b, l2) && !pointOnSegmentSide(l2.a,
7 |             l1);
8 |     }
9 |     return pointOnSegment(l1.a, l2) || pointOnSegment(l1.b, l2) ||
10 |            pointOnSegment(l2.a, l1) ||
11 |            pointOnSegment(l2.b, l2);
12 |
13 | bool segmentIntersection1(L3 l1, L3 l2) { // 重叠、相交于端点不视为相交
14 |     return onPlane(l1.a, l1.b, l2.a, l2.b) && !pointOnSegmentSide(l1.a, l1.b, l2) &&
15 |           !pointOnSegmentSide(l2.a, l2.b, l1);
16 |

```

6.11.12 空间两直线是否相交及交点

当两直线不共面、两直线平行时返回 *false*。

```

1 | pair<bool, P3> lineIntersection(L3 l1, L3 l2) {
2 |     if (!onPlane(l1.a, l1.b, l2.a, l2.b) || lineParallel(l1, l2)) {
3 |         return {0, {}};
4 |     }
5 |     auto [s1, e1] = l1;
6 |     auto [s2, e2] = l2;
7 |     ld val = 0;
8 |     if (!onPlane(l1.a, l1.b, {0, 0, 0}, {0, 0, 1})) {
9 |         val = ((s1.x - s2.x) * (s2.y - e2.y) - (s1.y - s2.y) * (s2.x - e2.x)) /
10 |               ((s1.x - e1.x) * (s2.y - e2.y) - (s1.y - e1.y) * (s2.x - e2.x));
11 |     } else if (!onPlane(l1.a, l1.b, {0, 0, 0}, {0, 1, 0})) {
12 |         val = ((s1.x - s2.x) * (s2.z - e2.z) - (s1.z - s2.z) * (s2.x - e2.x)) /
13 |               ((s1.x - e1.x) * (s2.z - e2.z) - (s1.z - e1.z) * (s2.x - e2.x));
14 |     } else {
15 |         val = ((s1.y - s2.y) * (s2.z - e2.z) - (s1.z - s2.z) * (s2.y - e2.y)) /

```

```

16         ((s1.y - e1.y) * (s2.z - e2.z) - (s1.z - e1.z) * (s2.y - e2.y));
17     }
18     return {1, s1 + (e1 - s1) * val};
19 }
```

6.11.13 直线与平面是否相交及交点

当直线与平面平行、给定的点构不成平面时返回 *false*。

```

1 pair<bool, P3> linePlaneCross(L3 l, Plane s) {
2     if (linePlaneParallel(l, s)) {
3         return {0, {}};
4     }
5     P3 vec = getVec(s);
6     P3 U = vec * (s.u - l.a), V = vec * (l.b - l.a);
7     ld val = (U.x + U.y + U.z) / (V.x + V.y + V.z);
8     return {1, l.a + (l.b - l.a) * val};
9 }
```

6.11.14 两平面是否相交及交线

当两平面平行、两平面为同一个时返回 *false*。

```

1 pair<bool, L3> planeIntersection(Plane s1, Plane s2) {
2     if (planeParallel(s1, s2) || same(s1, s2)) {
3         return {0, {}};
4     }
5     P3 U = linePlaneParallel({s2.u, s2.v}, s1) ? linePlaneCross({s2.v, s2.w},
6                     s1).second
7                         : linePlaneCross({s2.u, s2.v},
8                     s1).second;
9     P3 V = linePlaneParallel({s2.w, s2.u}, s1) ? linePlaneCross({s2.v, s2.w},
10                s1).second
11                   : linePlaneCross({s2.w, s2.u},
12                 s1).second;
13     return {1, {U, V}};
14 }
```

6.11.15 点到直线的最近点与最近距离

```

1 pair<ld, P3> pointToLine(P3 p, L3 l) {
2     ld val = cross(p - l.a, l.a - l.b) / dis(l.a, l.b); // 面积除以底边长
3     ld val1 = dot(p - l.a, l.a - l.b) / dis(l.a, l.b);
4     return {val, l.a + val1 * standardize(l.a - l.b)};
5 }
```

6.11.16 点到平面的最近点与最近距离

```

1 pair<ld, P3> pointToPlane(P3 p, Plane s) {
2     P3 vec = getVec(s);
3     ld val = dot(vec, p - s.u);
4     val = abs(val) / len(vec); // 面积除以底边长
5     return {val, p - val * standardize(vec)};
6 }
```

6.11.17 空间两直线的最近距离与最近点对

```

1  tuple<ld, P3, P3> lineToLine(L3 l1, L3 l2) {
2      P3 vec = crossEx(l1.a - l1.b, l2.a - l2.b); // 计算同时垂直于两直线的向量
3      ld val = abs(dot(l1.a - l2.a, vec)) / len(vec);
4      P3 U = l1.b - l1.a, V = l2.b - l2.a;
5      vec = crossEx(U, V);
6      ld p = dot(vec, vec);
7      ld t1 = dot(crossEx(l2.a - l1.a, V), vec) / p;
8      ld t2 = dot(crossEx(l2.a - l1.a, U), vec) / p;
9      return {val, l1.a + (l1.b - l1.a) * t1, l2.a + (l2.b - l2.a) * t2};
10 }

```

6.12 三维角度与弧度

6.12.1 空间两直线夹角的 cos 值

任意位置的空间两直线。

```

1  ld lineCos(L3 l1, L3 l2) {
2      return dot(l1.a - l1.b, l2.a - l2.b) / len(l1.a - l1.b) / len(l2.a - l2.b);
3 }

```

6.12.2 空间两平面夹角的 cos 值

```

1  ld planeCos(Plane s1, Plane s2) {
2      P3 U = getVec(s1), V = getVec(s2);
3      return dot(U, V) / len(U) / len(V);
4 }

```

6.12.3 直线与平面夹角的 sin 值

```

1  ld linePlaneSin(L3 l, Plane s) {
2      P3 vec = getVec(s);
3      return dot(l.a - l.b, vec) / len(l.a - l.b) / len(vec);
4 }

```

6.13 空间多边形

6.13.1 正N棱锥体积公式

棱锥通用体积公式 $V = \frac{1}{3}Sh$ ，当其恰好是棱长为 l 的正 n 棱锥时，有公式

$$V = \frac{l^3 \cdot n}{12 \tan \frac{\pi}{n}} \cdot \sqrt{1 - \frac{1}{4 \cdot \sin^2 \frac{\pi}{n}}}.$$

```

1  ld V(ld l, int n) { // 正n棱锥体积公式
2      return l * l * l * n / (12 * tan(PI / n)) * sqrt(1 - 1 / (4 * sin(PI / n) *
3      sin(PI / n)));
}

```

6.13.2 四面体体积

```

1 | ld V(P3 a, P3 b, P3 c, P3 d) {
2 |     return abs(dot(d - a, crossEx(b - a, c - a))) / 6;
3 |

```

6.13.3 点是否在空间三角形上

点位于边界上时返回 *false*。

```

1 | bool pointOnTriangle(P3 p, P3 p1, P3 p2, P3 p3) {
2 |     return pointOnSegmentSide(p, p1, {p2, p3}) && pointOnSegmentSide(p, p2, {p1, p3})
3 |     && pointOnSegmentSide(p, p3, {p1, p2});
4 |

```

6.13.4 线段是否与空间三角形相交及交点

只有交点在空间三角形内部时才视作相交。

```

1 | pair<bool, P3> segmentOnTriangle(P3 l, P3 r, P3 p1, P3 p2, P3 p3) {
2 |     P3 x = crossEx(p2 - p1, p3 - p1);
3 |     if (sign(dot(x, r - l)) == 0) {
4 |         return {0, {}};
5 |     }
6 |     ld t = dot(x, p1 - l) / dot(x, r - l);
7 |     if (t < 0 || t - 1 > 0) { // 不在线段上
8 |         return {0, {}};
9 |     }
10 |    bool type = pointOnTriangle(l + (r - l) * t, p1, p2, p3);
11 |    if (type) {
12 |        return {1, l + (r - l) * t};
13 |    } else {
14 |        return {0, {}};
15 |    }
16 |

```

6.13.5 空间三角形是否相交

相交线段在空间三角形内部时才视作相交。

```

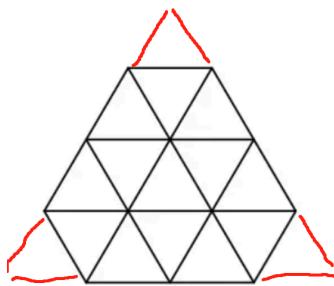
1 | bool triangleIntersection(vector<P3> a, vector<P3> b) {
2 |     for (int i = 0; i < 3; i++) {
3 |         if (segmentOnTriangle(b[i], b[(i + 1) % 3], a[0], a[1], a[2]).first) {
4 |             return 1;
5 |         }
6 |         if (segmentOnTriangle(a[i], a[(i + 1) % 3], b[0], b[1], b[2]).first) {
7 |             return 1;
8 |         }
9 |     }
10 |    return 0;
11 |

```

6.14 常用结论

6.14.1 平面几何结论归档

- `hypot` 函数可以直接计算直角三角形的斜边长；
- **边心距**是指正多边形的外接圆圆心到正多边形某一边的距离，边长为 s 的正 n 角形的边心距公式为 $a = \frac{t}{2 \cdot \tan \frac{\pi}{n}}$ ，外接圆半径为 R 的正 n 角形的边心距公式为 $a = R \cdot \cos \frac{\pi}{n}$ ；
- **三角形外接圆半径**为 $\frac{a}{2 \sin A} = \frac{abc}{4S}$ ，其中 S 为三角形面积，内切圆半径为 $\frac{2S}{a+b+c}$ ；
- 由小正三角形拼成的大正三角形，耗费的小三角形数量即为构成一条边的小三角形数量的平方。如下图，总数量即为 4^2 [See](#)。



- 正 n 边形圆心角为 $\frac{360^\circ}{n}$ ，圆周角为 $\frac{180^\circ}{n}$ 。定义正 n 边形上的三个顶点 A, B 和 C （可以不相邻），使得 $\angle ABC = \theta$ ，当 $n \leq 360$ 时， θ 可以取 1° 到 179° 间的任何一个整数 [See](#)。
- 某一点 B 到直线 AC 的距离公式为 $\frac{|\vec{BA} \times \vec{BC}|}{|AC|}$ ，等价于 $\frac{|aX + bY + c|}{\sqrt{a^2 + b^2}}$ 。
- `atan(y / x)` 函数仅用于计算第一、四象限的值，而 `atan2(y, x)` 则允许计算所有四个象限的正反切，在使用这个函数时，需要尽量保证 x 和 y 的类型为整数型，如果使用浮点数，实测会慢十倍。
- 在平面上有奇数个点 A_0, A_1, \dots, A_n 以及一个点 X_0 ，构造 X_1 使得 X_0, X_1 关于 A_0 对称、构造 X_2 使得 X_1, X_2 关于 A_1 对称、……、构造 X_j 使得 X_{j-1}, X_j 关于 $A_{(j-1) \bmod n}$ 对称。那么周期为 $2n$ ，即 A_0 与 A_{2n} 共点、 A_1 与 A_{2n+1} 共点 [See](#)。
- 已知 $A(x_A, y_A)$ 和 $X(x_X, y_X)$ 两点及这两点的坐标，构造 Y 使得 X, Y 关于 A 对称，那么 Y 的坐标为 $(2 \cdot x_A - x_X, 2 \cdot y_A - y_X)$ 。
- **海伦公式**：已知三角形三边长 a, b 和 c ，定义 $p = \frac{a+b+c}{2}$ ，则 $S_\Delta = \sqrt{p(p-a)(p-b)(p-c)}$ ，在使用时需要注意越界问题，本质是铅锤定理，一般多使用叉乘计算三角形面积而不使用该公式。
- 棱台体积 $V = \frac{1}{3}(S_1 + S_2 + \sqrt{S_1 S_2}) \cdot h$ ，其中 S_1, S_2 为上下底面积。
- 正棱台侧面积 $\frac{1}{2}(C_1 + C_2) \cdot L$ ，其中 C_1, C_2 为上下底周长， L 为斜高（上下底对应的平行边的距离）。
- 球面积 $4\pi r^2$ ，体积 $\frac{4}{3}\pi r^3$ 。
- 正三角形面积 $\frac{\sqrt{3}a^2}{4}$ ，正四面体面积 $\frac{\sqrt{2}a^3}{12}$ 。
- 设扇形对应的圆心角弧度为 θ ，则面积为 $S = \frac{\theta}{2} \cdot R^2$ 。

6.14.2 立体几何结论归档

- 已知向量 $\vec{r} = \{x, y, z\}$, 则该向量的三个方向余弦为
 $\cos \alpha = \frac{x}{|\vec{r}|} = \frac{x}{\sqrt{x^2 + y^2 + z^2}}$; $\cos \beta = \frac{y}{|\vec{r}|}$; $\cos \gamma = \frac{z}{|\vec{r}|}$ 。其中 $\alpha, \beta, \gamma \in [0, \pi]$,
 $\cos^2 \alpha + \cos^2 \beta + \cos^2 \gamma = 1$ 。

6.15 常用例题

6.15.1 将平面某点旋转任意角度

题意：给定平面上一点 (a, b) , 输出将其逆时针旋转 d 度之后的坐标。

```

1 | signed main() {
2 |     int a, b, d;
3 |     cin >> a >> b >> d;
4 |
5 |     ld l = hypot(a, b); // 库函数，求直角三角形的斜边
6 |     ld alpha = atan2(b, a) + toArc(d);
7 |
8 |     cout << l * cos(alpha) << " " << l * sin(alpha) << endl;
9 |

```

6.15.2 平面最近点对 (set解)

借助 `set` , 在严格 $\mathcal{O}(N \log N)$ 复杂度内求解 , 比常见的分治法稍快。

```

1 | template<class T> T sqr(T x) {
2 |     return x * x;
3 | }
4 |
5 | using V = Point<int>;
6 | signed main() {
7 |     int n;
8 |     cin >> n;
9 |
10 |     vector<V> in(n);
11 |     for (auto &it : in) {
12 |         cin >> it;
13 |     }
14 |
15 |     int dis = disEx(in[0], in[1]); // 设定阈值
16 |     sort(in.begin(), in.end());
17 |
18 |     set<V> S;
19 |     for (int i = 0, h = 0; i < n; i++) {
20 |         V now = {in[i].y, in[i].x};
21 |         while (dis && dis <= sqr(in[i].x - in[h].x)) { // 删除超过阈值的点
22 |             S.erase({in[h].y, in[h].x});
23 |             h++;
24 |         }
25 |         auto it = S.lower_bound(now);
26 |         for (auto k = it; k != S.end() && sqr(k->x - now.x) < dis; k++) {
27 |             dis = min(dis, disEx(*k, now));
28 |         }
29 |         if (it != S.begin()) {
30 |             for (auto k = prev(it); sqr(k->x - now.x) < dis; k--) {
31 |                 dis = min(dis, disEx(*k, now));
32 |                 if (k == S.begin()) break;
33 |             }
34 |         }
35 |     }
36 |
37 |     cout << dis << endl;
38 |
39 | }
```

```

35     S.insert(now);
36 }
37 cout << sqrt(dis) << endl;
38 }
```

6.15.3 平面若干点能构成的最大四边形的面积（简单版，暴力枚举）

题意：平面上存在若干个点，保证没有两点重合、没有三点共线，你需要从中选出四个点，使得它们构成的四边形面积是最大的，注意这里能组成的四边形可以不是凸四边形。

暴力枚举其中一条对角线后枚举剩余两个点， $\mathcal{O}(N^3)$ 。

```

1 signed main() {
2     int n;
3     cin >> n;
4     vector<Pi> in(n);
5     for (auto &it : in) {
6         cin >> it;
7     }
8     ld ans = 0;
9     for (int i = 0; i < n; i++) {
10        for (int j = i + 1; j < n; j++) { // 枚举对角线
11            ld l = 0, r = 0;
12            for (int k = 0; k < n; k++) { // 枚举第三点
13                if (k == i || k == j) continue;
14                if (pointOnLineLeft(in[k], {in[i], in[j]})) {
15                    l = max(l, triangleS(in[k], in[j], in[i]));
16                } else {
17                    r = max(r, triangleS(in[k], in[j], in[i]));
18                }
19            }
20            if (l * r != 0) { // 确保构成的是四边形
21                ans = max(ans, l + r);
22            }
23        }
24    }
25    cout << ans << endl;
26 }
```

6.15.4 平面若干点能构成的最大四边形的面积（困难版，分类讨论+旋转卡壳）

题意：平面上存在若干个点，可能存在多点重合、共线的情况，你需要从中选出四个点，使得它们构成的四边形面积是最大的，注意这里能组成的四边形可以不是凸四边形、可以是退化的四边形。

当凸包大小 ≤ 2 时，说明是退化的四边形，答案直接为 0；大小恰好为 3 时，说明是凹四边形，我们枚举不在凸包上的那一点，将两个三角形面积相减既可得到答案；大小恰好为 4 时，说明是凸四边形，使用旋转卡壳求解。

```

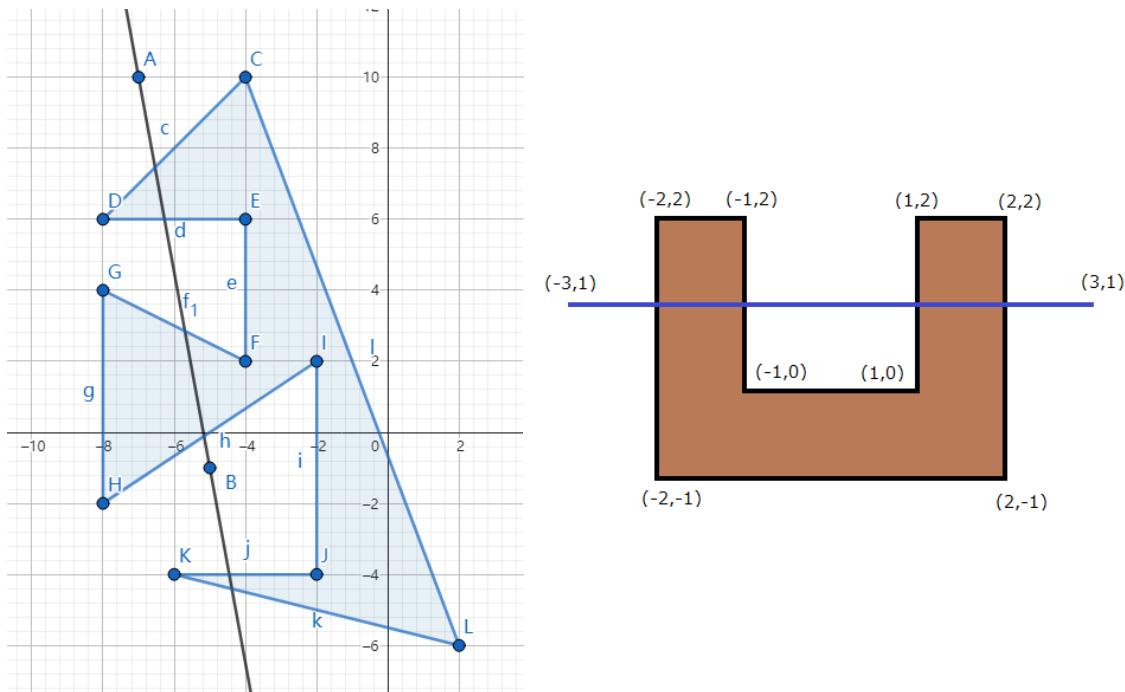
1 using V = Point<int>;
2 signed main() {
3     int Task = 1;
4     for (cin >> Task; Task; Task--) {
5         int n;
6         cin >> n;
7
8         vector<V> in_(n);
9         for (auto &it : in_) {
10             cin >> it;
11         }
12     }
13 }
```

```

12     auto in = staticConvexHull(in_, 0);
13     n = in.size();
14
15     int ans = 0;
16     if (n > 3) {
17         ans = rotatingCalipers(in);
18     } else if (n == 3) {
19         int area = triangleAreaEx(in[0], in[1], in[2]);
20         for (auto it : in_) {
21             if (it == in[0] || it == in[1] || it == in[2]) continue;
22             int Min = min({triangleAreaEx(it, in[0], in[1]), triangleAreaEx(it,
in[0], in[2]), triangleAreaEx(it, in[1], in[2])});
23             ans = max(ans, area - Min);
24         }
25     }
26
27     cout << ans / 2;
28     if (ans % 2) {
29         cout << ".5";
30     }
31     cout << endl;
32 }
33 }
```

6.15.5 线段将多边形切割为几个部分

题意：给定平面上一线段与一个任意多边形，求解线段将多边形切割为几个部分；保证线段的端点不在多边形内、多边形边上，多边形顶点不位于线段上，多边形的边不与线段重叠；多边形端点按逆时针顺序给出。下方的几个样例均合法，答案均为 3。



当线段切割多边形时，本质是与多边形的边交于两个点、或者说是与多边形的两条边相交，设交点数目为 x ，那么答案即为 $\frac{x}{2} + 1$ 。于是，我们只需要计算交点数量即可，先判断某一条边是否与线段相交，再判断边的两个端点是否位于线段两侧。

```

1 signed main() {
2     Pi s, e;
3     cin >> s >> e; // 读入线段
4
5     int n;
```

```

6   cin >> n;
7   vector<Pi> in(n);
8   for (auto &it : in) {
9       cin >> it; // 读入多边形端点
10  }
11
12  int cnt = 0;
13  for (int i = 0; i < n; i++) {
14      Pi x = in[i], y = in[(i + 1) % n];
15      cnt += (pointNotOnLineSide(x, y, {s, e}) && segmentIntersection(Line{x, y},
16 {s, e}));
17  }
18  cout << cnt / 2 + 1 << endl;
19 }
```

6.15.6 平面若干点能否构成凸包（暴力枚举）

题意：给定平面上若干个点，判断其是否构成凸包 [See](#)。

可以直接使用凸包模板，但是代码较长；在这里我们使用暴力枚举试点，也能以 $\mathcal{O}(N)$ 的复杂度通过。当两个向量的叉乘 ≤ 0 时说明其夹角大于等于 180° ，使用这一点即可判定。

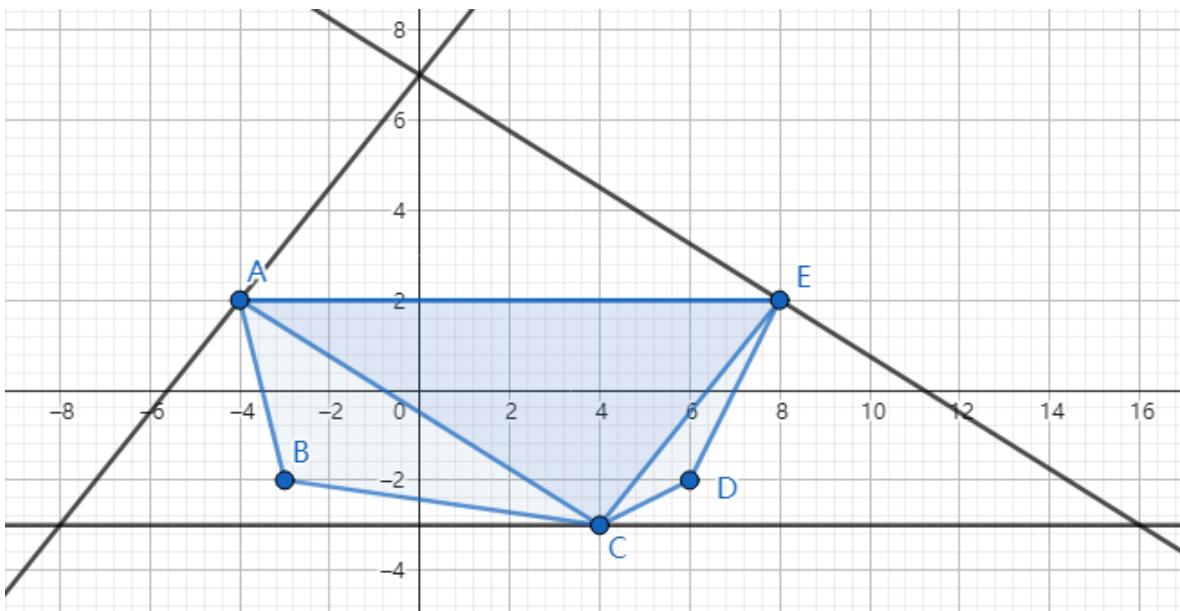
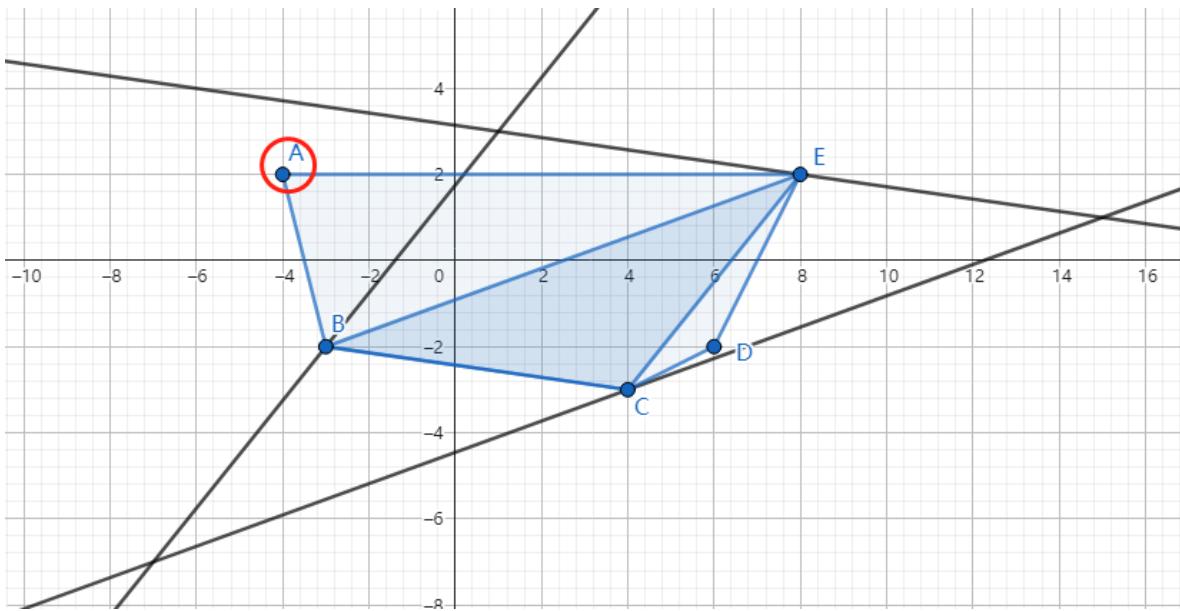
```

1 signed main() {
2     int n;
3     cin >> n;
4
5     vector<Point<ld>> in(n);
6     for (auto &it : in) {
7         cin >> it;
8     }
9
10    for (int i = 0; i < n; i++) {
11        auto A = in[(i - 1 + n) % n];
12        auto B = in[i];
13        auto C = in[(i + 1) % n];
14        if (cross(A - B, C - B) > 0) {
15            cout << "No\n";
16            return 0;
17        }
18    }
19    cout << "Yes\n";
20 }
```

6.15.7 凸包上的点能构成的最大三角形（暴力枚举）

可以直接使用凸包模板，但是代码较长；在这里我们使用暴力枚举试点，也能以 $\mathcal{O}(N)$ 的复杂度通过。

另外补充一点性质：所求三角形的反互补三角形一定包含了凸包上的所有点（可以在边界）。通俗的说，构成的三角形是这个反互补三角形的中点三角形。如下图所示，点 A 不在 $\triangle BCE$ 的反互补三角形内部，故 $\triangle BCE$ 不是最大三角形； $\triangle ACE$ 才是。



```

1 signed main() {
2     int n;
3     cin >> n;
4
5     vector<Point<int>> in(n);
6     for (auto &it : in) {
7         cin >> it;
8     }
9
10 #define S(x, y, z) triangleAreaEx(in[x], in[y], in[z])
11
12     int i = 0, j = 1, k = 2;
13     while (true) {
14         int val = S(i, j, k);
15         if (S((i + 1) % n, j, k) > val) {
16             i = (i + 1) % n;
17         } else if (S((i - 1 + n) % n, j, k) > val) {
18             i = (i - 1 + n) % n;
19         } else if (S(i, (j + 1) % n, k) > val) {
20             j = (j + 1) % n;
21         } else if (S(i, (j - 1 + n) % n, k) > val) {
22             j = (j - 1 + n) % n;
23         } else if (S(i, j, (k + 1) % n) > val) {

```

```

24         k = (k + 1) % n;
25     } else if (S(i, j, (k - 1 + n) % n) > val) {
26         k = (k - 1 + n) % n;
27     } else {
28         break;
29     }
30 }
31 cout << i + 1 << " " << j + 1 << " " << k + 1 << endl;
32 }
```

6.15.8 凸包上的点能构成的最大四角形的面积（旋转卡壳）

由于是凸包上的点，所以保证了四边形一定是凸四边形，时间复杂度 $\mathcal{O}(N^2)$ 。

```

1 template<class T> T rotatingCalipers(vector<Point<T>> &p) {
2     #define S(x, y, z) triangleAreaEx(p[x], p[y], p[z])
3     int n = p.size();
4     T ans = 0;
5     auto nxt = [&](int i) -> int {
6         return i == n - 1 ? 0 : i + 1;
7     };
8     for (int i = 0; i < n; i++) {
9         int p1 = nxt(i), p2 = nxt(nxt(nxt(i)));
10        for (int j = nxt(nxt(i)); nxt(j) != i; j = nxt(j)) {
11            while (nxt(p1) != j && S(i, j, nxt(p1)) > S(i, j, p1)) {
12                p1 = nxt(p1);
13            }
14            if (p2 == j) {
15                p2 = nxt(p2);
16            }
17            while (nxt(p2) != i && S(i, j, nxt(p2)) > S(i, j, p2)) {
18                p2 = nxt(p2);
19            }
20            ans = max(ans, S(i, j, p1) + S(i, j, p2));
21        }
22    }
23    return ans;
24 }
25 #undef S
```

6.15.9 判断一个凸包是否完全在另一个凸包内

题意：给定一个凸多边形 A 和一个凸多边形 B ，询问 B 是否被 A 包含，分别判断严格/不严格包含。[例题](#)。

考虑严格包含，使用 A 点集计算出凸包 T_1 ，使用 A, B 两个点集计算出不严格凸包 T_2 ，如果包含，那么 T_1 应该与 T_2 完全相等；考虑不严格包含，在计算凸包 T_2 时严格即可。最终以 $\mathcal{O}(N)$ 复杂度求解，且代码不算很长。

/END/

7 多项式

7.1 多项式封装

```

1 template<int P = 998244353> struct Poly : public vector<MInt<P>> {
2     using Value = MInt<P>;
3
4     Poly() : vector<Value>() {}
5     explicit constexpr Poly(int n) : vector<Value>(n) {}
6
7     explicit constexpr Poly(const vector<Value> &a) : vector<Value>(a) {}
8     constexpr Poly(const initializer_list<Value> &a) : vector<Value>(a) {}
9
10    template<class InputIt, class = _RequireInputIter<InputIt>>
11        explicit constexpr Poly(InputIt first, InputIt last) : vector<Value>(first,
12        last) {}
13
14    template<class F> explicit constexpr Poly(int n, F f) : vector<Value>(n) {
15        for (int i = 0; i < n; i++) {
16            (*this)[i] = f(i);
17        }
18    }
19
20    constexpr Poly shift(int k) const {
21        if (k >= 0) {
22            auto b = *this;
23            b.insert(b.begin(), k, 0);
24            return b;
25        } else if (this->size() <= -k) {
26            return Poly();
27        } else {
28            return Poly(this->begin() + (-k), this->end());
29        }
30    }
31    constexpr Poly trunc(int k) const {
32        Poly f = *this;
33        f.resize(k);
34        return f;
35    }
36    constexpr friend Poly operator+(const Poly &a, const Poly &b) {
37        Poly res(max(a.size(), b.size()));
38        for (int i = 0; i < a.size(); i++) {
39            res[i] += a[i];
40        }
41        for (int i = 0; i < b.size(); i++) {
42            res[i] += b[i];
43        }
44        return res;
45    }
46    constexpr friend Poly operator-(const Poly &a, const Poly &b) {
47        Poly res(max(a.size(), b.size()));
48        for (int i = 0; i < a.size(); i++) {
49            res[i] += a[i];
50        }
51        for (int i = 0; i < b.size(); i++) {
52            res[i] -= b[i];
53        }
54        return res;
55    }
56    constexpr friend Poly operator-(const Poly &a) {
57        vector<Value> res(a.size());

```

```

58         res[i] = -a[i];
59     }
60     return Poly(res);
61 }
62 constexpr friend Poly operator*(Poly a, Poly b) {
63     if (a.size() == 0 || b.size() == 0) {
64         return Poly();
65     }
66     if (a.size() < b.size()) {
67         swap(a, b);
68     }
69     int n = 1, tot = a.size() + b.size() - 1;
70     while (n < tot) {
71         n *= 2;
72     }
73     if (((P - 1) & (n - 1)) != 0 || b.size() < 128) {
74         Poly c(a.size() + b.size() - 1);
75         for (int i = 0; i < a.size(); i++) {
76             for (int j = 0; j < b.size(); j++) {
77                 c[i + j] += a[i] * b[j];
78             }
79         }
80         return c;
81     }
82     a.resize(n);
83     b.resize(n);
84     dft(a);
85     dft(b);
86     for (int i = 0; i < n; ++i) {
87         a[i] *= b[i];
88     }
89     idft(a);
90     a.resize(tot);
91     return a;
92 }
93 constexpr friend Poly operator*(Value a, Poly b) {
94     for (int i = 0; i < int(b.size()); i++) {
95         b[i] *= a;
96     }
97     return b;
98 }
99 constexpr friend Poly operator*(Poly a, Value b) {
100    for (int i = 0; i < int(a.size()); i++) {
101        a[i] *= b;
102    }
103    return a;
104 }
105 constexpr friend Poly operator/(Poly a, Value b) {
106     for (int i = 0; i < int(a.size()); i++) {
107         a[i] /= b;
108     }
109     return a;
110 }
111 constexpr Poly &operator+=(Poly b) {
112     return (*this) = (*this) + b;
113 }
114 constexpr Poly &operator-=(Poly b) {
115     return (*this) = (*this) - b;
116 }
117 constexpr Poly &operator*=(Poly b) {
118     return (*this) = (*this) * b;
119 }
120 constexpr Poly &operator*=(Value b) {
121     return (*this) = (*this) * b;

```

```

122 }
123 constexpr Poly &operator/=(Value b) {
124     return (*this) = (*this) / b;
125 }
126 constexpr Poly deriv() const {
127     if (this->empty()) {
128         return Poly();
129     }
130     Poly res(this->size() - 1);
131     for (int i = 0; i < this->size() - 1; ++i) {
132         res[i] = (i + 1) * (*this)[i + 1];
133     }
134     return res;
135 }
136 constexpr Poly integr() const {
137     Poly res(this->size() + 1);
138     for (int i = 0; i < this->size(); ++i) {
139         res[i + 1] = (*this)[i] / (i + 1);
140     }
141     return res;
142 }
143 constexpr Poly inv(int m) const {
144     Poly x{(*this)[0].inv()};
145     int k = 1;
146     while (k < m) {
147         k *= 2;
148         x = (x * (Poly{2} - trunc(k) * x)).trunc(k);
149     }
150     return x.trunc(m);
151 }
152 constexpr Poly log(int m) const {
153     return (deriv() * inv(m)).integr().trunc(m);
154 }
155 constexpr Poly exp(int m) const {
156     Poly x{1};
157     int k = 1;
158     while (k < m) {
159         k *= 2;
160         x = (x * (Poly{1} - x.log(k) + trunc(k))).trunc(k);
161     }
162     return x.trunc(m);
163 }
164 constexpr Poly pow(int k, int m) const {
165     int i = 0;
166     while (i < this->size() && (*this)[i] == 0) {
167         i++;
168     }
169     if (i == this->size() || 1LL * i * k >= m) {
170         return Poly(m);
171     }
172     Value v = (*this)[i];
173     auto f = shift(-i) * v.inv();
174     return (f.log(m - i * k) * k).exp(m - i * k).shift(i * k) * power(v, k);
175 }
176 constexpr Poly sqrt(int m) const {
177     Poly x{1};
178     int k = 1;
179     while (k < m) {
180         k *= 2;
181         x = (x + (trunc(k) * x.inv(k)).trunc(k)) * CInv<2, P>;
182     }
183     return x.trunc(m);
184 }
185 constexpr Poly mult(Poly b) const {

```

```

186     if (b.size() == 0) {
187         return Poly();
188     }
189     int n = b.size();
190     reverse(b.begin(), b.end());
191     return ((*this) * b).shift(-(n - 1));
192 }
193 constexpr vector<Value> eval(vector<Value> x) const {
194     if (this->size() == 0) {
195         return vector<Value>(x.size(), 0);
196     }
197     const int n = max(x.size(), this->size());
198     vector<Poly> q(4 * n);
199     vector<Value> ans(x.size());
200     x.resize(n);
201     function<void(int, int, int)> build = [&](int p, int l, int r) {
202         if (r - l == 1) {
203             q[p] = Poly{1, -x[l]};
204         } else {
205             int m = (l + r) / 2;
206             build(2 * p, l, m);
207             build(2 * p + 1, m, r);
208             q[p] = q[2 * p] * q[2 * p + 1];
209         }
210     };
211     build(1, 0, n);
212     function<void(int, int, int, const Poly &)> work = [&](int p, int l, int r,
213                                         const Poly
214                                         &num) {
215         if (r - l == 1) {
216             if (l < int(ans.size())) {
217                 ans[l] = num[0];
218             }
219         } else {
220             int m = (l + r) / 2;
221             work(2 * p, l, m, num.mult(q[2 * p + 1]).resize(m - 1));
222             work(2 * p + 1, m, r, num.mult(q[2 * p]).resize(r - m));
223         }
224     };
225     work(1, 0, n, mult(q[1].inv(n)));
226     return ans;
227 }

```

7.2 离散傅里叶变换 dft 与其逆变换 idft

```

1 vector<int> rev;
2 template<int P> vector<MIInt<P>> roots{0, 1};
3
4 template<int P> constexpr MIInt<P> findPrimitiveRoot() {
5     MIInt<P> i = 2;
6     int k = __builtin_ctz(P - 1);
7     while (true) {
8         if (power(i, (P - 1) / 2) != 1) {
9             break;
10        }
11        i += 1;
12    }
13    return power(i, (P - 1) >> k);
14 }
15
16 template<int P> constexpr MIInt<P> primitiveRoot = findPrimitiveRoot<P>();

```

```

17 template<> constexpr MInt<998244353> primitiveRoot<998244353>{31};
18
19 template<int P> constexpr void dft(vector<MInt<P>> &a) { // 离散傅里叶变换
20     int n = a.size();
21
22     if (int(rev.size()) != n) {
23         int k = __builtin_ctz(n) - 1;
24         rev.resize(n);
25         for (int i = 0; i < n; i++) {
26             rev[i] = rev[i >> 1] >> 1 | (i & 1) << k;
27         }
28     }
29
30     for (int i = 0; i < n; i++) {
31         if (rev[i] < i) {
32             swap(a[i], a[rev[i]]);
33         }
34     }
35     if (roots<P>.size() < n) {
36         int k = __builtin_ctz(roots<P>.size());
37         roots<P>.resize(n);
38         while ((1 << k) < n) {
39             auto e = power(primitiveRoot<P>, 1 << (__builtin_ctz(P - 1) - k - 1));
40             for (int i = 1 << (k - 1); i < (1 << k); i++) {
41                 roots<P>[2 * i] = roots<P>[i];
42                 roots<P>[2 * i + 1] = roots<P>[i] * e;
43             }
44             k++;
45         }
46     }
47     for (int k = 1; k < n; k *= 2) {
48         for (int i = 0; i < n; i += 2 * k) {
49             for (int j = 0; j < k; j++) {
50                 MInt<P> u = a[i + j];
51                 MInt<P> v = a[i + j + k] * roots<P>[k + j];
52                 a[i + j] = u + v;
53                 a[i + j + k] = u - v;
54             }
55         }
56     }
57 }
58 template<int P> constexpr void idft(vector<MInt<P>> &a) { // 逆变换
59     int n = a.size();
60     reverse(a.begin() + 1, a.end());
61     dft(a);
62     MInt<P> inv = (1 - P) / n;
63     for (int i = 0; i < n; i++) {
64         a[i] *= inv;
65     }
66 }
```

7.3 Berlekamp-Massey 算法 (杜教筛)

求解数列的最短线性递推式，最坏复杂度为 $\mathcal{O}(NM)$ ，其中 N 为数列长度， M 为它的最短递推式的阶数。

```

1 template<int P = 998244353> Poly<P> berlekampMassey(const Poly<P> &s) {
2     Poly<P> c;
3     Poly<P> oldC;
4     int f = -1;
5     for (int i = 0; i < s.size(); i++) {
6         auto delta = s[i];
```

```

7   for (int j = 1; j <= c.size(); j++) {
8     delta -= c[j - 1] * s[i - j];
9   }
10  if (delta == 0) {
11    continue;
12  }
13  if (f == -1) {
14    c.resize(i + 1);
15    f = i;
16  } else {
17    auto d = oldC;
18    d *= -1;
19    d.insert(d.begin(), 1);
20    MInt<P> df1 = 0;
21    for (int j = 1; j <= d.size(); j++) {
22      df1 += d[j - 1] * s[f + 1 - j];
23    }
24    assert(df1 != 0);
25    auto coef = delta / df1;
26    d *= coef;
27    Poly<P> zeros(i - f - 1);
28    zeros.insert(zeros.end(), d.begin(), d.end());
29    d = zeros;
30    auto temp = c;
31    c += d;
32    if (i - temp.size() > f - oldC.size()) {
33      oldC = temp;
34      f = i;
35    }
36  }
37 }
38 c *= -1;
39 c.insert(c.begin(), 1);
40 return c;
41 }
```

7.4 Linear-Recurrence 算法

```

1 template<int P = 998244353> MInt<P> linearRecurrence(Poly<P> p, Poly<P> q, i64 n) {
2   int m = q.size() - 1;
3   while (n > 0) {
4     auto newq = q;
5     for (int i = 1; i <= m; i += 2) {
6       newq[i] *= -1;
7     }
8     auto newp = p * newq;
9     newq = q * newq;
10    for (int i = 0; i < m; i++) {
11      p[i] = newp[i * 2 + n % 2];
12    }
13    for (int i = 0; i <= m; i++) {
14      q[i] = newq[i * 2];
15    }
16    n /= 2;
17  }
18  return p[0] / q[0];
19 }
```

7.5 快速傅里叶变换 FFT

$\mathcal{O}(N \log N)$ 。

```

1 struct Polynomial {
2     constexpr static double PI = acos(-1);
3     struct Complex {
4         double x, y;
5         Complex(double _x = 0.0, double _y = 0.0) {
6             x = _x;
7             y = _y;
8         }
9         Complex operator-(const Complex &rhs) const {
10             return Complex(x - rhs.x, y - rhs.y);
11         }
12         Complex operator+(const Complex &rhs) const {
13             return Complex(x + rhs.x, y + rhs.y);
14         }
15         Complex operator*(const Complex &rhs) const {
16             return Complex(x * rhs.x - y * rhs.y, x * rhs.y + y * rhs.x);
17         }
18     };
19     vector<Complex> c;
20     Polynomial(vector<int> &a) {
21         int n = a.size();
22         c.resize(n);
23         for (int i = 0; i < n; i++) {
24             c[i] = Complex(a[i], 0);
25         }
26         fft(c, n, 1);
27     }
28     void change(vector<Complex> &a, int n) {
29         for (int i = 1, j = n / 2; i < n - 1; i++) {
30             if (i < j) swap(a[i], a[j]);
31             int k = n / 2;
32             while (j >= k) {
33                 j -= k;
34                 k /= 2;
35             }
36             if (j < k) j += k;
37         }
38     }
39     void fft(vector<Complex> &a, int n, int opt) {
40         change(a, n);
41         for (int h = 2; h <= n; h *= 2) {
42             Complex wn(cos(2 * PI / h), sin(opt * 2 * PI / h));
43             for (int j = 0; j < n; j += h) {
44                 Complex w(1, 0);
45                 for (int k = j; k < j + h / 2; k++) {
46                     Complex u = a[k];
47                     Complex t = w * a[k + h / 2];
48                     a[k] = u + t;
49                     a[k + h / 2] = u - t;
50                     w = w * wn;
51                 }
52             }
53         }
54         if (opt == -1) {
55             for (int i = 0; i < n; i++) {
56                 a[i].x /= n;
57             }
58         }
59     }
}

```

60 | };

7.6 快速数论变换 NTT

$\mathcal{O}(N \log N)$ 。

```

1 struct Polynomial {
2     vector<Z> z;
3     vector<int> r;
4     Polynomial(vector<int> &a) {
5         int n = a.size();
6         z.resize(n);
7         r.resize(n);
8         for (int i = 0; i < n; i++) {
9             z[i] = a[i];
10            r[i] = (i & 1) * (n / 2) + r[i / 2] / 2;
11        }
12        ntt(z, n, 1);
13    }
14    LL power(LL a, int b) {
15        LL res = 1;
16        for (; b; b /= 2, a = a * a % mod) {
17            if (b % 2) {
18                res = res * a % mod;
19            }
20        }
21        return res;
22    }
23    void ntt(vector<Z> &a, int n, int opt) {
24        for (int i = 0; i < n; i++) {
25            if (r[i] < i) {
26                swap(a[i], a[r[i]]);
27            }
28        }
29        for (int k = 2; k <= n; k *= 2) {
30            Z gn = power(3, (mod - 1) / k);
31            for (int i = 0; i < n; i += k) {
32                Z g = 1;
33                for (int j = 0; j < k / 2; j++, g *= gn) {
34                    Z t = a[i + j + k / 2] * g;
35                    a[i + j + k / 2] = a[i + j] - t;
36                    a[i + j] = a[i + j] + t;
37                }
38            }
39        }
40        if (opt == -1) {
41            reverse(a.begin() + 1, a.end());
42            Z inv = power(n, mod - 2);
43            for (int i = 0; i < n; i++) {
44                a[i] *= inv;
45            }
46        }
47    }
48 };

```

7.7 拉格朗日插值

$n + 1$ 个点可以唯一确定一个最高为 n 次的多项式。普通情况： $f(k) = \sum_{i=1}^{n+1} y_i \prod_{i \neq j} \frac{k - x[j]}{x[i] - x[j]}$ 。

```

1 struct Lagrange {
2     int n;
3     vector<Z> x, y, fac, invfac;
4     Lagrange(int n) {
5         this->n = n;
6         x.resize(n + 3);
7         y.resize(n + 3);
8         fac.resize(n + 3);
9         invfac.resize(n + 3);
10        init(n);
11    }
12    void init(int n) {
13        iota(x.begin(), x.end(), 0);
14        for (int i = 1; i <= n + 2; i++) {
15            Z t;
16            y[i] = y[i - 1] + t.power(i, n);
17        }
18        fac[0] = 1;
19        for (int i = 1; i <= n + 2; i++) {
20            fac[i] = fac[i - 1] * i;
21        }
22        invfac[n + 2] = fac[n + 2].inv();
23        for (int i = n + 1; i >= 0; i--) {
24            invfac[i] = invfac[i + 1] * (i + 1);
25        }
26    }
27    Z solve(LL k) {
28        if (k <= n + 2) {
29            return y[k];
30        }
31        vector<Z> sub(n + 3);
32        for (int i = 1; i <= n + 2; i++) {
33            sub[i] = k - x[i];
34        }
35        vector<Z> mul(n + 3);
36        mul[0] = 1;
37        for (int i = 1; i <= n + 2; i++) {
38            mul[i] = mul[i - 1] * sub[i];
39        }
40        Z ans = 0;
41        for (int i = 1; i <= n + 2; i++) {
42            ans = ans + y[i] * mul[n + 2] * sub[i].inv() * pow(-1, n + 2 - i) *
43                invfac[i - 1] *
44                invfac[n + 2 - i];
45        }
46    }
47 };

```

7.8 常用结论

7.8.1 杂

- 求 $B_i = \sum_{k=i}^n C_k^i A_k$, 即 $B_i = \frac{1}{i!} \sum_{k=i}^n \frac{1}{(k-i)!} \cdot k! A_k$, 反转后卷积。
- NTT中, $\omega_n = \text{qpow}(G, (\text{mod}-1)/n)$ 。
- 遇到 $\sum_{i=0}^n [i \% k = 0] f(i)$ 可以转换为 $\sum_{i=0}^n \frac{1}{k} \sum_{j=0}^{k-1} (\omega_k^i)^j f(i)$ 。（单位根卷积）
- 广义二项式定理 $(1+x)^\alpha = \sum_{i=0}^{\infty} \binom{n}{\alpha} x^i$ 。

7.8.2 普通生成函数 / OGF

- 普通生成函数: $A(x) = a_0 + a_1 x + a_2 x^2 + \dots = \langle a_0, a_1, a_2, \dots \rangle$;
- $1 + x^k + x^{2k} + \dots = \frac{1}{1 - x^k}$;
- 取对数后 $= -\ln(1 - x^k) = \sum_{i=1}^{\infty} \frac{1}{i} x^{ki}$ 即 $\sum_{i=1}^{\infty} \frac{1}{i} x^i \otimes x^k$ (polymul_special);
- $x + \frac{x^2}{2} + \frac{x^3}{3} + \dots = -\ln(1 - x)$;
- $1 + x + x^2 + \dots + x^{m-1} = \frac{1 - x^m}{1 - x}$;
- $1 + 2x + 3x^2 + \dots = \frac{1}{(1-x)^2}$ (借用导数, $nx^{n-1} = (x^n)'$);
- $C_m^0 + C_m^1 x + C_m^2 x^2 + \dots + C_m^m x^m = (1+x)^m$ (二项式定理);
- $C_m^0 + C_{m+1}^1 x^1 + C_{m+2}^2 x^2 + \dots = \frac{1}{(1-x)^{m+1}}$ (归纳法证明);
- $\sum_{n=0}^{\infty} F_n x^n = \frac{(F_1 - F_0)x + F_0}{1 - x - x^2}$ (F 为斐波那契数列, 列方程
 $G(x) = xG(x) + x^2 G(x) + (F_1 - F_0)x + F_0$);
- $\sum_{n=0}^{\infty} H_n x^n = \frac{1 - \sqrt{n - 4x}}{2x}$ (H 为卡特兰数);
- 前缀和 $\sum_{n=0}^{\infty} s_n x^n = \frac{1}{1-x} f(x)$;
- 五边形数定理: $\prod_{i=1}^{\infty} (1 - x^i) = \sum_{k=0}^{\infty} (-1)^k x^{\frac{1}{2}k(3k \pm 1)}$ 。

7.8.3 指数生成函数 / EGF

- 指数生成函数: $A(x) = a_0 + a_1 x + a_2 \frac{x^2}{2!} + a_3 \frac{x^3}{3!} + \dots = \langle a_0, a_1, a_2, a_3, \dots \rangle$;
- 普通生成函数转换为指数生成函数: 系数乘以 $n!$;
- $1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \exp x$;

- 长度为 n 的循环置换数为 $P(x) = -\ln(1-x)$, 长度为 n 的置换数为 $\exp P(x) = \frac{1}{1-x}$ (注意是**指数生成函数**)

- n 个点的生成树个数是 $P(x) = \sum_{n=1}^{\infty} n^{n-2} \frac{x^n}{n!}$, n 个点的生成森林个数是 $\exp P(x)$;

- n 个点的无向连通图个数是 $P(x)$, n 个点的无向图个数是 $\exp P(x) = \sum_{n=0}^{\infty} 2^{\frac{1}{2}n(n-1)} \frac{x^n}{n!}$;

- 长度为 $n (n \geq 2)$ 的循环置换数是 $P(x) = -\ln(1-x) - x$, 长度为 n 的错排数是 $\exp P(x)$ 。

/END/

8 数据结构

8.1 并查集（全功能）

```

1 struct DSU {
2     vector<int> fa, p, e, f;
3
4     DSU(int n) {
5         fa.resize(n + 1);
6         iota(fa.begin(), fa.end(), 0);
7         p.resize(n + 1, 1);
8         e.resize(n + 1);
9         f.resize(n + 1);
10    }
11    int get(int x) {
12        while (x != fa[x]) {
13            x = fa[x] = fa[fa[x]];
14        }
15        return x;
16    }
17    bool merge(int x, int y) { // 设x是y的祖先
18        if (x == y) f[get(x)] = 1;
19        x = get(x), y = get(y);
20        e[x]++;
21        if (x == y) return false;
22        if (x < y) swap(x, y); // 将编号小的合并到大的上
23        fa[y] = x;
24        f[x] |= f[y], p[x] += p[y], e[x] += e[y];
25        return true;
26    }
27    bool same(int x, int y) {
28        return get(x) == get(y);
29    }
30    bool F(int x) { // 判断连通块内是否存在自环
31        return f[get(x)];
32    }
33    int size(int x) { // 输出连通块中点的数量
34        return p[get(x)];
35    }
36    int E(int x) { // 输出连通块中边的数量
37        return e[get(x)];
38    }
39 };
40

```

8.2 树状数组

8.2.1 半动态区间和（单点修改）

```

1 struct BIT {
2     vector<i64> w;
3     int n;
4     BIT(int n) : n(n), w(n + 1) {}
5     void add(int x, i64 v) {
6         for (; x <= n; x += x & -x) {
7             w[x] += v;
8         }
9     }
10    i64 rangeAsk(int l, int r) { // 差分实现区间和查询
11        auto ask = [&](int x) {
12            i64 ans = 0;
13        }
14    }
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40

```

```

13         for ( ; x; x -= x & -x) {
14             ans += w[x];
15         }
16         return ans;
17     };
18     return ask(r) - ask(l - 1);
19 }
20 };

```

8.2.2 动态区间和（区间修改）

```

1 struct BIT {
2     vector<i64> a, b;
3     int n;
4
5     BIT(int n) : n(n), a(n + 1), b(n + 1) {}
6     void rangeAdd(int l, int r, i64 val) { // 区间修改
7         auto add = [&](int pos, i64 val) {
8             for (int i = pos; i <= n; i += i & -i) {
9                 a[i] += val;
10                b[i] += pos * val;
11            }
12        };
13        add(l, val), add(r + 1, -val);
14    }
15    i64 rangeSum(int l, int r) { // 区间和查询
16        auto sum = [&](int x) {
17            i64 ans = 0;
18            for (int i = x; i; i -= i & -i) {
19                ans += (x + 1) * a[i] - b[i];
20            }
21        };
22        return sum(r) - sum(l - 1);
23    }
24 }
25 };

```

8.2.3 静态区间最值（单点修改）

```

1 struct BIT {
2     vector<i64> w;
3     int n;
4     BIT(int n) : n(n), w(2 * n + 1) {}
5
6     void modify(int pos, i64 val) {
7         for (w[pos += n] = val; pos > 1; pos /= 2) {
8             w[pos / 2] = max(w[pos], w[pos ^ 1]);
9         }
10    }
11
12    i64 rangeMax(int l, int r) {
13        r++; // 使用开区间实现
14        i64 res = -1e18;
15        for (l += n, r += n; l < r; l /= 2, r /= 2) {
16            if (l % 2) res = max(res, w[l++]);
17            if (r % 2) res = max(res, w[--r]);
18        }
19        return res;
20    }
21 };

```

8.2.4 逆序对扩展

性质：交换序列的任意两元素，序列的逆序数的奇偶性必定发生改变。

```

1 struct BIT {
2     int n;
3     vector<int> w, chk; // chk 为传入的待处理数组
4     BIT(auto &in) : n(in.size() - 1), w(in.size()), chk(in) {}
5     void add(int x, i64 v) {...}
6     i64 rangeAsk(int l, int r) {...}
7     i64 get() {
8         vector<array<int, 2>> alls;
9         for (int i = 1; i <= n; i++) {
10            alls.push_back({chk[i], i});
11        }
12        sort(alls.begin(), alls.end());
13        i64 ans = 0;
14        for (auto [val, idx] : alls) {
15            ans += ask(idx + 1, n);
16            add(idx, 1);
17        }
18        return ans;
19    }
20};

```

8.2.5 前驱后继扩展（常规+第 k 小值查询+元素排名查询+元素前驱后继查询）

注意，被查询的值都应该小于等于 N ，否则会越界；如果离散化不可使用，则需要使用平衡树替代。

```

1 struct BIT {
2     int n;
3     vector<int> w;
4     BIT(int n) : n(n), w(n + 1) {}
5     void add(int x, int v) {
6         for (; x <= n; x += x & -x) {
7             w[x] += v;
8         }
9     }
10    int kth(int x) { // 查找第 k 小的值
11        int ans = 0;
12        for (int i = __lg(n); i >= 0; i--) {
13            int val = ans + (1 << i);
14            if (val < n && w[val] < x) {
15                x -= w[val];
16                ans = val;
17            }
18        }
19        return ans + 1;
20    }
21    int get(int x) { // 查找 x 的排名
22        int ans = 1;
23        for (x--; x; x -= x & -x) {
24            ans += w[x];
25        }
26        return ans;
27    }
28    int pre(int x) { return kth(get(x) - 1); } // 查找 x 的前驱
29    int suf(int x) { return kth(get(x + 1)); } // 查找 x 的后继
30};

```

```

31 const int N = 10000000; // 可以用于在线处理平衡二叉树的全部要求
32 signed main() {
33     BIT bit(N + 1); // 在线处理不能够离散化，一定要开到比最大值更大
34     int n;
35     cin >> n;
36     for (int i = 1; i <= n; i++) {
37         int op, x;
38         cin >> op >> x;
39         if (op == 1) bit.add(x, 1); // 插入 x
40         else if (op == 2) bit.add(x, -1); // 删除任意一个 x
41         else if (op == 3) cout << bit.get(x) << "\n"; // 查询 x 的排名
42         else if (op == 4) cout << bit.kth(x) << "\n"; // 查询排名为 x 的数
43         else if (op == 5) cout << bit.pre(x) << "\n"; // 求小于 x 的最大值（前驱）
44         else if (op == 6) cout << bit.suf(x) << "\n"; // 求大于 x 的最小值（后继）
45     }
46 }
```

8.3 二维树状数组

8.3.1 半动态矩阵和（单点修改）

封装一：该版本不能同时进行区间修改+区间求和。空间占用为 $\mathcal{O}(NM)$ 、建树复杂度为 $\mathcal{O}(NM)$ 、单次查询复杂度为 $\mathcal{O}(\log N \cdot \log M)$ 。

```

1 template<class T> struct BIT_2D {
2     int n, m;
3     vector<vector<T>> w;
4
5     BIT_2D(int n, int m) : n(n), m(m) {
6         w.resize(n + 1, vector<T>(m + 1));
7     }
8     void pointAdd(int x, int y, T k) {
9         for (int i = x; i <= n; i += i & -i) {
10             for (int j = y; j <= m; j += j & -j) {
11                 w[i][j] += k;
12             }
13         }
14     }
15     void matrixAdd(int x, int y, int X, int Y, T k) { // 区块修改：二维差分
16         X++, Y++;
17         pointAdd(x, y, k), pointAdd(X, y, -k);
18         pointAdd(X, Y, k), pointAdd(x, Y, -k);
19     }
20     T pointSum(int x, int y) {
21         T ans = T();
22         for (int i = x; i -= i & -i) {
23             for (int j = y; j; j -= j & -j) {
24                 ans += w[i][j];
25             }
26         }
27         return ans;
28     }
29     T matrixSum(int x, int y, int X, int Y) { // 区块查询：二维前缀和
30         X--, Y--;
31         return pointSum(X, Y) - pointSum(x, Y) - pointSum(X, y) + pointSum(x, y);
32     }
33 };
```

8.3.2 动态矩阵和（区间修改）

封装二：仅支持区间修改+区间求和。但是时空复杂度均比上一个版本多 4 倍。

```

1 template<class T> struct BIT_2D {
2     int n, m;
3     vector<vector<T>> b1, b2, b3, b4;
4
5     BIT_2D(int n, int m) : n(n), m(m) {
6         b1.resize(n + 1, vector<T>(m + 1));
7         b2.resize(n + 1, vector<T>(m + 1));
8         b3.resize(n + 1, vector<T>(m + 1));
9         b4.resize(n + 1, vector<T>(m + 1));
10    }
11    void matrixAdd(int x, int y, int X, int Y, T k) { // 区块修改：二维差分
12        X++, Y++;
13        auto add = [&](int x, int y, T k) {
14            for (int i = x; i <= n; i += i & -i) {
15                for (int j = y; j <= m; j += j & -j) {
16                    b1[i][j] += k;
17                    b2[i][j] += k * (x - 1);
18                    b3[i][j] += k * (y - 1);
19                    b4[i][j] += k * (x - 1) * (y - 1);
20                }
21            }
22        };
23        add(x, y, k), add(X, y, -k);
24        add(X, Y, k), add(x, Y, -k);
25    }
26    T matrixSum(int x, int y, int X, int Y) { // 区块查询：二维前缀和
27        x--, y--;
28        auto ask = [&](int x, int y) {
29            T ans = T();
30            for (int i = x; i; i -= i & -i) {
31                for (int j = y; j; j -= j & -j) {
32                    ans += x * y * b1[i][j];
33                    ans -= y * b2[i][j];
34                    ans -= x * b3[i][j];
35                    ans += b4[i][j];
36                }
37            }
38            return ans;
39        };
40        return ask(X, Y) - ask(x, Y) - ask(X, y) + ask(x, y);
41    }
42};
```

8.4 线段树

8.4.1 区间加法修改、区间最小值查询

```

1 template<class T> struct Segt {
2     struct node {
3         int l, r;
4         T w, rmq, lazy;
5     };
6     vector<T> w;
7     vector<node> t;
8
9     Segt() {}
10    Segt(int n) { init(n); }
11    Segt(vector<int> in) {
```

```

12     int n = in.size() - 1;
13     w.resize(n + 1);
14     for (int i = 1; i <= n; i++) {
15         w[i] = in[i];
16     }
17     init(in.size() - 1);
18 }
19
20 #define GL (k << 1)
21 #define GR (k << 1 | 1)
22
23 void init(int n) {
24     w.resize(n + 1);
25     t.resize(n * 4 + 1);
26     auto build = [&](auto self, int l, int r, int k = 1) {
27         if (l == r) {
28             t[k] = {l, r, w[l], w[l], -1}; // 如果有赋值为 0 的操作，则懒标记必须要
29             -1
30             return;
31         }
32         t[k] = {l, r, 0, 0, -1};
33         int mid = (l + r) / 2;
34         self(self, l, mid, GL);
35         self(self, mid + 1, r, GR);
36         pushup(k);
37     };
38     build(build, 1, n);
39 }
40 void pushdown(node &p, T lazy) { /* 【在此更新下递函数】 */
41     p.w += (p.r - p.l + 1) * lazy;
42     p.rmq += lazy;
43     p.lazy += lazy;
44 }
45 void pushdown(int k) {
46     if (t[k].lazy == -1) return;
47     pushdown(t[GL], t[k].lazy);
48     pushdown(t[GR], t[k].lazy);
49     t[k].lazy = -1;
50 }
51 void pushup(int k) {
52     auto pushup = [&](node &p, node &l, node &r) { /* 【在此更新上传函数】 */
53         p.w = l.w + r.w;
54         p.rmq = min(l.rmq, r.rmq); // RMQ -> min/max
55     };
56     pushup(t[k], t[GL], t[GR]);
57 }
58 void modify(int l, int r, T val, int k = 1) { // 区间修改
59     if (l <= t[k].l && t[k].r <= r) {
60         pushdown(t[k], val);
61         return;
62     }
63     pushdown(k);
64     int mid = (t[k].l + t[k].r) / 2;
65     if (l <= mid) modify(l, r, val, GL);
66     if (mid < r) modify(l, r, val, GR);
67     pushup(k);
68 }
69 T rmq(int l, int r, int k = 1) { // 区间询问最小值
70     if (l <= t[k].l && t[k].r <= r) {
71         return t[k].rmq;
72     }
73     pushdown(k);
74     int mid = (t[k].l + t[k].r) / 2;
75     T ans = numeric_limits<T>::max(); // RMQ -> 为 max 时需要修改为 ::lowest()

```

```

75     if (l <= mid) ans = min(ans, rmq(l, r, GL)); // RMQ -> min/max
76     if (mid < r) ans = min(ans, rmq(l, r, GR)); // RMQ -> min/max
77     return ans;
78 }
79 T ask(int l, int r, int k = 1) { // 区间询问
80     if (l <= t[k].l && t[k].r <= r) {
81         return t[k].w;
82     }
83     pushdown(k);
84     int mid = (t[k].l + t[k].r) / 2;
85     T ans = 0;
86     if (l <= mid) ans += ask(l, r, GL);
87     if (mid < r) ans += ask(l, r, GR);
88     return ans;
89 }
90 void debug(int k = 1) {
91     cout << "[" << t[k].l << ", " << t[k].r << "]: ";
92     cout << "w = " << t[k].w << ", ";
93     cout << "Min = " << t[k].rmq << ", ";
94     cout << "lazy = " << t[k].lazy << ", ";
95     cout << endl;
96     if (t[k].l == t[k].r) return;
97     debug(GL), debug(GR);
98 }
99 };

```

8.4.2 同时需要处理区间加法与乘法修改

```

1 template <class T> struct Segt_ {
2     struct node {
3         int l, r;
4         T w, add, mul = 1; // 注意初始赋值
5     };
6     vector<T> w;
7     vector<node> t;
8
9     Segt_(int n) {
10        w.resize(n + 1);
11        t.resize((n << 2) + 1);
12        build(1, n);
13    }
14    Segt_(vector<int> in) {
15        int n = in.size() - 1;
16        w.resize(n + 1);
17        for (int i = 1; i <= n; i++) {
18            w[i] = in[i];
19        }
20        t.resize((n << 2) + 1);
21        build(1, n);
22    }
23    void pushdown(node &p, T add, T mul) { // 在此更新下递函数
24        p.w = p.w * mul + (p.r - p.l + 1) * add;
25        p.add = p.add * mul + add;
26        p.mul *= mul;
27    }
28    void pushup(node &p, node &l, node &r) { // 在此更新上传函数
29        p.w = l.w + r.w;
30    }
31 #define GL (k << 1)
32 #define GR (k << 1 | 1)
33 void pushdown(int k) { // 不需要动
34     pushdown(t[GL], t[k].add, t[k].mul);
35     pushdown(t[GR], t[k].add, t[k].mul);

```

```

36     t[k].add = 0, t[k].mul = 1;
37 }
38 void pushup(int k) { // 不需要动
39     pushup(t[k], t[GL], t[GR]);
40 }
41 void build(int l, int r, int k = 1) {
42     if (l == r) {
43         t[k] = {l, r, w[l]};
44         return;
45     }
46     t[k] = {l, r};
47     int mid = (l + r) / 2;
48     build(l, mid, GL);
49     build(mid + 1, r, GR);
50     pushup(k);
51 }
52 void modify(int l, int r, T val, int k = 1) { // 区间修改
53     if (l <= t[k].l && t[k].r <= r) {
54         t[k].w += (t[k].r - t[k].l + 1) * val;
55         t[k].add += val;
56         return;
57     }
58     pushdown(k);
59     int mid = (t[k].l + t[k].r) / 2;
60     if (l <= mid) modify(l, r, val, GL);
61     if (mid < r) modify(l, r, val, GR);
62     pushup(k);
63 }
64 void modify2(int l, int r, T val, int k = 1) { // 区间修改
65     if (l <= t[k].l && t[k].r <= r) {
66         t[k].w *= val;
67         t[k].add *= val;
68         t[k].mul *= val;
69         return;
70     }
71     pushdown(k);
72     int mid = (t[k].l + t[k].r) / 2;
73     if (l <= mid) modify2(l, r, val, GL);
74     if (mid < r) modify2(l, r, val, GR);
75     pushup(k);
76 }
77 T ask(int l, int r, int k = 1) { // 区间询问，不合并
78     if (l <= t[k].l && t[k].r <= r) {
79         return t[k].w;
80     }
81     pushdown(k);
82     int mid = (t[k].l + t[k].r) / 2;
83     T ans = 0;
84     if (l <= mid) ans += ask(l, r, GL);
85     if (mid < r) ans += ask(l, r, GR);
86     return ans;
87 }
88 void debug(int k = 1) {
89     cout << "[" << t[k].l << ", " << t[k].r << "]: ";
90     cout << "w = " << t[k].w << ", ";
91     cout << "add = " << t[k].add << ", ";
92     cout << "mul = " << t[k].mul << ", ";
93     cout << endl;
94     if (t[k].l == t[k].r) return;
95     debug(GL), debug(GR);
96 }
97 };

```

8.4.3 区间赋值/推平

如果存在推平为 0 的操作，那么需要将 `lazy` 初始赋值为 -1 。

```

1 void pushdown(node &p, T lazy) { /* 【在此更新下递函数】 */
2     p.w = (p.r - p.l + 1) * lazy;
3     p.lazy = lazy;
4 }
5 void modify(int l, int r, T val, int k = 1) {
6     if (l <= t[k].l && t[k].r <= r) {
7         t[k].w = val;
8         t[k].lazy = val;
9         return;
10    }
11    // 剩余部分不变
12 }
```

8.4.4 区间取模

原题需要进行“单点赋值+区间取模+区间求和” [See](#)。该操作不需要懒标记。

需要额外维护一个区间最大值，当模数大于区间最大值时剪枝，否则进行单点取模。由于单点 $MOD < x$ 时 $x \bmod MOD < \frac{x}{2}$ ，故单点取模至 0 最劣只需要 $\log x$ 次。

```

1 void modifyMod(int l, int r, T val, int k = 1) {
2     if (l <= t[k].l && t[k].r <= r) {
3         if (t[k].rmq < val) return; // 重要剪枝
4     }
5     if (t[k].l == t[k].r) {
6         t[k].w %= val;
7         t[k].rmq %= val;
8         return;
9     }
10    int mid = (t[k].l + t[k].r) / 2;
11    if (l <= mid) modifyMod(l, r, val, GL);
12    if (mid < r) modifyMod(l, r, val, GR);
13    pushup(k);
14 }
```

8.4.5 区间异或修改

原题需要维护“区间异或修改+区间求和” [See](#)。

```

1 struct Segt { // #define GL (k << 1) // #define GR (k << 1 | 1)
2     struct node {
3         int l, r;
4         int w[N], lazy; // 注意这里为了方便计算，w 只需要存位
5     };
6     vector<int> base;
7     vector<node> t;
8
9     Segt(vector<int> in) : base(in) {
10         int n = in.size() - 1;
11         t.resize(n * 4 + 1);
12         auto build = [&](auto self, int l, int r, int k = 1) {
13             t[k] = {l, r}; // 前置赋值
14             if (l == r) {
15                 for (int i = 0; i < N; i++) {
16                     t[k].w[i] = base[l] >> i & 1;
17                 }
18             }
19         };
20         build();
21     }
22     void modify(int l, int r, int val, int k = 1) {
23         if (l <= t[k].l && t[k].r <= r) {
24             t[k].w ^= val;
25             t[k].lazy ^= val;
26         } else {
27             if (l < t[k].l) modify(l, t[k].l - 1, val, k * 2);
28             if (r > t[k].r) modify(t[k].r + 1, r, val, k * 2 + 1);
29             if (t[k].l != t[k].r) build();
30         }
31     }
32     int query(int l, int r, int k = 1) {
33         if (l <= t[k].l && t[k].r <= r) {
34             return t[k].w;
35         } else {
36             if (l < t[k].l) return query(l, t[k].l - 1, k * 2);
37             if (r > t[k].r) return query(t[k].r + 1, r, k * 2 + 1);
38             if (t[k].l != t[k].r) build();
39             return t[k].w;
40         }
41     }
42 }
```

```

18         return;
19     }
20     int mid = (l + r) / 2;
21     self(self, l, mid, GL);
22     self(self, mid + 1, r, GR);
23     pushup(k);
24 };
25 build(build, 1, n);
26 }
27 void pushdown(node &p, int lazy) { /* 【在此更新下递函数】 */
28     int len = p.r - p.l + 1;
29     for (int i = 0; i < N; i++) {
30         if (lazy >> i & 1) { // 即 p.w = (p.r - p.l + 1) - p.w;
31             p.w[i] = len - p.w[i];
32         }
33     }
34     p.lazy ^= lazy;
35 }
36 void pushdown(int k) { // 【不需要动】
37     if (t[k].lazy == 0) return;
38     pushdown(t[GL], t[k].lazy);
39     pushdown(t[GR], t[k].lazy);
40     t[k].lazy = 0;
41 }
42 void pushup(int k) {
43     auto pushup = [&](node &p, node &l, node &r) { /* 【在此更新上传函数】 */
44         for (int i = 0; i < N; i++) {
45             p.w[i] = l.w[i] + r.w[i]; // 即 p.w = l.w + r.w;
46         }
47     };
48     pushup(t[k], t[GL], t[GR]);
49 }
50 void modify(int l, int r, int val, int k = 1) { // 区间修改
51     if (l <= t[k].l && t[k].r <= r) {
52         pushdown(t[k], val);
53         return;
54     }
55     pushdown(k);
56     int mid = (t[k].l + t[k].r) / 2;
57     if (l <= mid) modify(l, r, val, GL);
58     if (mid < r) modify(l, r, val, GR);
59     pushup(k);
60 }
61 i64 ask(int l, int r, int k = 1) { // 区间求和
62     if (l <= t[k].l && t[k].r <= r) {
63         i64 ans = 0;
64         for (int i = 0; i < N; i++) {
65             ans += t[k].w[i] * (1LL << i);
66         }
67         return ans;
68     }
69     pushdown(k);
70     int mid = (t[k].l + t[k].r) / 2;
71     i64 ans = 0;
72     if (l <= mid) ans += ask(l, r, GL);
73     if (mid < r) ans += ask(l, r, GR);
74     return ans;
75 }
76 };

```

8.4.6 拆位运算

原题同上。使用若干棵线段树维护每一位的值，区间异或转变为区间翻转。

```

1 template<class T> struct Segt_ { // GL 为 (k << 1), GR 为 (k << 1 | 1)
2     struct node {
3         int l, r;
4         T w;
5         bool lazy; // 注意懒标记用布尔型足以
6     };
7     vector<T> w;
8     vector<node> t;
9
10    Segt_() {}
11    void init(vector<int> in) {
12        int n = in.size() - 1;
13        w.resize(n * 4 + 1);
14        for (int i = 0; i <= n; i++) { w[i] = in[i]; }
15        t.resize(n * 4 + 1);
16        build(1, n);
17    }
18    void pushdown(node &p, bool lazy = 1) { // 【在此更新下递函数】
19        p.w = (p.r - p.l + 1) - p.w;
20        p.lazy ^= lazy;
21    }
22    void pushup(node &p, node &l, node &r) { // 【在此更新上传函数】
23        p.w = l.w + r.w;
24    }
25    void pushdown(int k) { // 【不需要动】
26        if (t[k].lazy == 0) return;
27        pushdown(t[GL]), pushdown(t[GR]); // 注意这里不再需要传入第二个参数
28        t[k].lazy = 0;
29    }
30    void pushup(int k) { pushup(t[k], t[GL], t[GR]); } // 【不需要动】
31    void build(int l, int r, int k = 1) {
32        if (l == r) {
33            t[k] = {l, r, w[l], 0}; // 注意懒标记初始为 0
34            return;
35        }
36        t[k] = {l, r};
37        int mid = (l + r) / 2;
38        build(l, mid, GL);
39        build(mid + 1, r, GR);
40        pushup(k);
41    }
42    void reverse(int l, int r, int k = 1) { // 区间翻转
43        if (l <= t[k].l && t[k].r <= r) {
44            pushdown(t[k], 1);
45            return;
46        }
47        pushdown(k);
48        int mid = (t[k].l + t[k].r) / 2;
49        if (l <= mid) reverse(l, r, GL);
50        if (mid < r) reverse(l, r, GR);
51        pushup(k);
52    }
53    T ask(int l, int r, int k = 1) { // 区间求和
54        if (l <= t[k].l && t[k].r <= r) {
55            return t[k].w;
56        }
57        pushdown(k);
58        int mid = (t[k].l + t[k].r) / 2;
59        T ans = 0;

```

```

60     if (l <= mid) ans += ask(l, r, GL);
61     if (mid < r) ans += ask(l, r, GR);
62     return ans;
63 }
64 };
65 signed main() {
66     int n; cin >> n;
67     vector<vector<int>> in(20, vector<int>(n + 1));
68     Segt_<i64> segt[20]; // 拆位建线段树
69     for (int i = 1, x; i <= n; i++) { cin >> x;
70         for (int bit = 0; bit < 20; bit++) {
71             in[bit][i] = x >> bit & 1;
72         }
73     }
74     for (int i = 0; i < 20; i++) {
75         segt[i].init(in[i]);
76     }
77
78     int m, op;
79     for (cin >> m; m--) { cin >> op;
80         if (op == 1) {
81             int l, r; i64 ans = 0; cin >> l >> r;
82             for (int i = 0; i < 20; i++) {
83                 ans += segt[i].ask(l, r) * (1LL << i);
84             }
85             cout << ans << "\n";
86         } else {
87             int l, r, val; cin >> l >> r >> val;
88             for (int i = 0; i < 20; i++) {
89                 if (val >> i & 1) { segt[i].reverse(l, r); }
90             }
91         }
92     }
93 }

```

8.5 坐标压缩与离散化

8.5.1 简单版本

```

1 sort(alls.begin(), alls.end());
2 alls.erase(unique(alls.begin(), alls.end()), alls.end());
3 auto get = [&](int x) {
4     return lower_bound(alls.begin(), alls.end(), x) - alls.begin();
5 };

```

8.5.2 封装

```

1 template <typename T> struct Compress_ {
2     int n, shift = 0; // shift 用于标记下标偏移量
3     vector<T> alls;
4
5     Compress_() {}
6     Compress_(auto in) : alls(in) {
7         init();
8     }
9     void add(T x) {
10         alls.emplace_back(x);
11     }
12     template <typename... Args> void add(T x, Args... args) {
13         add(x), add(args...);
14     }

```

```

15 void init() {
16     alls.emplace_back(numeric_limits<T>::max());
17     sort(alls.begin(), alls.end());
18     alls.erase(unique(alls.begin(), alls.end()), alls.end());
19     this->n = alls.size();
20 }
21 int size() {
22     return n;
23 }
24 int operator[](T x) { // 返回 x 元素的新下标
25     return upper_bound(alls.begin(), alls.end(), x) - alls.begin() + shift;
26 }
27 T Get(int x) { // 根据新下标返回原来元素
28     assert(x - shift < n);
29     return x - shift < n ? alls[x - shift] : -1;
30 }
31 bool count(T x) { // 查找元素 x 是否存在
32     return binary_search(alls.begin(), alls.end(), x);
33 }
34 friend auto &operator<< (ostream &o, const auto &j) {
35     cout << "{";
36     for (auto it : j.all) {
37         o << it << " ";
38     }
39     return o << "}";
40 }
41 };
42 using Compress = Compress_<int>;

```

8.6 轻重链剖分/树链剖分

将线段树处理的部分分离，方便修改。支持链上查询/修改、子树查询/修改，建树时间复杂度 $\mathcal{O}(N \log N)$ ，单次查询时间复杂度 $\mathcal{O}(\log^2 N)$ 。

```

1 struct Segt {
2     struct node {
3         int l, r, w, lazy;
4     };
5     vector<int> w;
6     vector<node> t;
7
8     Segt() {}
9     #define GL (k << 1)
10    #define GR (k << 1 | 1)
11
12    void init(vector<int> in) {
13        int n = in.size() - 1;
14        w.resize(n + 1);
15        for (int i = 1; i <= n; i++) {
16            w[i] = in[i];
17        }
18        t.resize(n * 4 + 1);
19        auto build = [&](auto self, int l, int r, int k = 1) {
20            if (l == r) {
21                t[k] = {l, r, w[l], 0}; // 如果有赋值为 0 的操作，则懒标记必须要 -1
22                return;
23            }
24            t[k] = {l, r};
25            int mid = (l + r) / 2;
26            self(self, l, mid, GL);
27            self(self, mid + 1, r, GR);
28            pushup(k);
29        };
30        build(0, 0, n);
31    }
32
33    void update(int l, int r, int v) {
34        update(0, 0, n, l, r, v);
35    }
36
37    int query(int l, int r) {
38        return query(0, 0, n, l, r);
39    }
40
41    void pushup(int k) {
42        t[k].w = t[t[k].l].w + t[t[k].r].w;
43    }
44
45    void update(int k, int l, int r, int v) {
46        if (t[k].l == t[k].r) {
47            t[k].w = v;
48            t[k].lazy = -1;
49            return;
50        }
51        if (t[k].lazy != -1) {
52            t[t[k].l].w += t[k].lazy;
53            t[t[k].r].w += t[k].lazy;
54            t[k].lazy = -1;
55        }
56        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
57            t[t[k].l].w += v;
58            t[t[k].r].w += v;
59        } else {
60            update(t[k].l, l, r, v);
61            update(t[k].r, l, r, v);
62        }
63    }
64
65    int query(int k, int l, int r) {
66        if (t[k].l == t[k].r) {
67            return t[k].w;
68        }
69        if (t[k].lazy != -1) {
70            t[t[k].l].w += t[k].lazy;
71            t[t[k].r].w += t[k].lazy;
72            t[k].lazy = -1;
73        }
74        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
75            return t[t[k].l].w;
76        } else {
77            return query(t[k].l, l, r) + query(t[k].r, l, r);
78        }
79    }
80
81    void self_update(int k, int l, int r) {
82        if (t[k].l == t[k].r) {
83            t[k].w = t[t[k].l].w;
84            t[k].lazy = -1;
85            return;
86        }
87        if (t[k].lazy != -1) {
88            t[t[k].l].w += t[k].lazy;
89            t[t[k].r].w += t[k].lazy;
90            t[k].lazy = -1;
91        }
92        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
93            t[t[k].l].w = t[t[k].l].w;
94            t[t[k].r].w = t[t[k].r].w;
95        } else {
96            self_update(t[k].l, l, r);
97            self_update(t[k].r, l, r);
98        }
99    }
100
101    void self_query(int k, int l, int r) {
102        if (t[k].l == t[k].r) {
103            return t[t[k].l].w;
104        }
105        if (t[k].lazy != -1) {
106            t[t[k].l].w += t[k].lazy;
107            t[t[k].r].w += t[k].lazy;
108            t[k].lazy = -1;
109        }
110        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
111            return t[t[k].l].w;
112        } else {
113            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
114        }
115    }
116
117    void pushdown(int k) {
118        if (t[k].lazy == -1) {
119            return;
120        }
121        if (t[t[k].l].l == t[t[k].l].r) {
122            t[t[k].l].w += t[k].lazy;
123            t[t[k].l].lazy = -1;
124        } else {
125            t[t[k].l].w += t[k].lazy;
126            t[t[k].r].w += t[k].lazy;
127            t[t[k].l].lazy = -1;
128            t[t[k].r].lazy = -1;
129        }
130        t[k].lazy = -1;
131    }
132
133    void pushup(int k) {
134        t[k].w = t[t[k].l].w + t[t[k].r].w;
135    }
136
137    void self(int k, int l, int r) {
138        if (t[k].l == t[k].r) {
139            cout << t[t[k].l].w;
140            return;
141        }
142        if (t[k].lazy != -1) {
143            t[t[k].l].w += t[k].lazy;
144            t[t[k].r].w += t[k].lazy;
145            t[k].lazy = -1;
146        }
147        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
148            cout << " ";
149            self(t[k].l, l, r);
150        } else {
151            self(t[k].l, l, r);
152            self(t[k].r, l, r);
153        }
154    }
155
156    void print() {
157        self(0, 0, n);
158    }
159
160    void pushup(int k) {
161        t[k].w = t[t[k].l].w + t[t[k].r].w;
162    }
163
164    void update(int k, int l, int r, int v) {
165        if (t[k].l == t[k].r) {
166            t[k].w = v;
167            t[k].lazy = -1;
168            return;
169        }
170        if (t[k].lazy != -1) {
171            t[t[k].l].w += t[k].lazy;
172            t[t[k].r].w += t[k].lazy;
173            t[k].lazy = -1;
174        }
175        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
176            t[t[k].l].w += v;
177            t[t[k].r].w += v;
178        } else {
179            update(t[k].l, l, r, v);
180            update(t[k].r, l, r, v);
181        }
182    }
183
184    int query(int k, int l, int r) {
185        if (t[k].l == t[k].r) {
186            return t[t[k].l].w;
187        }
188        if (t[k].lazy != -1) {
189            t[t[k].l].w += t[k].lazy;
190            t[t[k].r].w += t[k].lazy;
191            t[k].lazy = -1;
192        }
193        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
194            return t[t[k].l].w;
195        } else {
196            return query(t[k].l, l, r) + query(t[k].r, l, r);
197        }
198    }
199
200    void self_update(int k, int l, int r) {
201        if (t[k].l == t[k].r) {
202            t[k].w = t[t[k].l].w;
203            t[k].lazy = -1;
204            return;
205        }
206        if (t[k].lazy != -1) {
207            t[t[k].l].w += t[k].lazy;
208            t[t[k].r].w += t[k].lazy;
209            t[k].lazy = -1;
210        }
211        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
212            t[t[k].l].w = t[t[k].l].w;
213            t[t[k].r].w = t[t[k].r].w;
214        } else {
215            self_update(t[k].l, l, r);
216            self_update(t[k].r, l, r);
217        }
218    }
219
220    void self_query(int k, int l, int r) {
221        if (t[k].l == t[k].r) {
222            return t[t[k].l].w;
223        }
224        if (t[k].lazy != -1) {
225            t[t[k].l].w += t[k].lazy;
226            t[t[k].r].w += t[k].lazy;
227            t[k].lazy = -1;
228        }
229        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
230            return t[t[k].l].w;
231        } else {
232            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
233        }
234    }
235
236    void pushdown(int k) {
237        if (t[k].lazy == -1) {
238            return;
239        }
240        if (t[t[k].l].l == t[t[k].l].r) {
241            t[t[k].l].w += t[k].lazy;
242            t[t[k].l].lazy = -1;
243        } else {
244            t[t[k].l].w += t[k].lazy;
245            t[t[k].r].w += t[k].lazy;
246            t[t[k].l].lazy = -1;
247            t[t[k].r].lazy = -1;
248        }
249        t[k].lazy = -1;
250    }
251
252    void pushup(int k) {
253        t[k].w = t[t[k].l].w + t[t[k].r].w;
254    }
255
256    void self(int k, int l, int r) {
257        if (t[k].l == t[k].r) {
258            cout << t[t[k].l].w;
259            return;
260        }
261        if (t[k].lazy != -1) {
262            t[t[k].l].w += t[k].lazy;
263            t[t[k].r].w += t[k].lazy;
264            t[k].lazy = -1;
265        }
266        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
267            cout << " ";
268            self(t[k].l, l, r);
269        } else {
270            self(t[k].l, l, r);
271            self(t[k].r, l, r);
272        }
273    }
274
275    void print() {
276        self(0, 0, n);
277    }
278
279    void update(int k, int l, int r, int v) {
280        if (t[k].l == t[k].r) {
281            t[k].w = v;
282            t[k].lazy = -1;
283            return;
284        }
285        if (t[k].lazy != -1) {
286            t[t[k].l].w += t[k].lazy;
287            t[t[k].r].w += t[k].lazy;
288            t[k].lazy = -1;
289        }
290        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
291            t[t[k].l].w += v;
292            t[t[k].r].w += v;
293        } else {
294            update(t[k].l, l, r, v);
295            update(t[k].r, l, r, v);
296        }
297    }
298
299    int query(int k, int l, int r) {
300        if (t[k].l == t[k].r) {
301            return t[t[k].l].w;
302        }
303        if (t[k].lazy != -1) {
304            t[t[k].l].w += t[k].lazy;
305            t[t[k].r].w += t[k].lazy;
306            t[k].lazy = -1;
307        }
308        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
309            return t[t[k].l].w;
310        } else {
311            return query(t[k].l, l, r) + query(t[k].r, l, r);
312        }
313    }
314
315    void self_update(int k, int l, int r) {
316        if (t[k].l == t[k].r) {
317            t[k].w = t[t[k].l].w;
318            t[k].lazy = -1;
319            return;
320        }
321        if (t[k].lazy != -1) {
322            t[t[k].l].w += t[k].lazy;
323            t[t[k].r].w += t[k].lazy;
324            t[k].lazy = -1;
325        }
326        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
327            t[t[k].l].w = t[t[k].l].w;
328            t[t[k].r].w = t[t[k].r].w;
329        } else {
330            self_update(t[k].l, l, r);
331            self_update(t[k].r, l, r);
332        }
333    }
334
335    void self_query(int k, int l, int r) {
336        if (t[k].l == t[k].r) {
337            return t[t[k].l].w;
338        }
339        if (t[k].lazy != -1) {
340            t[t[k].l].w += t[k].lazy;
341            t[t[k].r].w += t[k].lazy;
342            t[k].lazy = -1;
343        }
344        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
345            return t[t[k].l].w;
346        } else {
347            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
348        }
349    }
350
351    void pushdown(int k) {
352        if (t[k].lazy == -1) {
353            return;
354        }
355        if (t[t[k].l].l == t[t[k].l].r) {
356            t[t[k].l].w += t[k].lazy;
357            t[t[k].l].lazy = -1;
358        } else {
359            t[t[k].l].w += t[k].lazy;
360            t[t[k].r].w += t[k].lazy;
361            t[t[k].l].lazy = -1;
362            t[t[k].r].lazy = -1;
363        }
364        t[k].lazy = -1;
365    }
366
367    void pushup(int k) {
368        t[k].w = t[t[k].l].w + t[t[k].r].w;
369    }
370
371    void self(int k, int l, int r) {
372        if (t[k].l == t[k].r) {
373            cout << t[t[k].l].w;
374            return;
375        }
376        if (t[k].lazy != -1) {
377            t[t[k].l].w += t[k].lazy;
378            t[t[k].r].w += t[k].lazy;
379            t[k].lazy = -1;
380        }
381        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
382            cout << " ";
383            self(t[k].l, l, r);
384        } else {
385            self(t[k].l, l, r);
386            self(t[k].r, l, r);
387        }
388    }
389
390    void print() {
391        self(0, 0, n);
392    }
393
394    void update(int k, int l, int r, int v) {
395        if (t[k].l == t[k].r) {
396            t[k].w = v;
397            t[k].lazy = -1;
398            return;
399        }
400        if (t[k].lazy != -1) {
401            t[t[k].l].w += t[k].lazy;
402            t[t[k].r].w += t[k].lazy;
403            t[k].lazy = -1;
404        }
405        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
406            t[t[k].l].w += v;
407            t[t[k].r].w += v;
408        } else {
409            update(t[k].l, l, r, v);
410            update(t[k].r, l, r, v);
411        }
412    }
413
414    int query(int k, int l, int r) {
415        if (t[k].l == t[k].r) {
416            return t[t[k].l].w;
417        }
418        if (t[k].lazy != -1) {
419            t[t[k].l].w += t[k].lazy;
420            t[t[k].r].w += t[k].lazy;
421            t[k].lazy = -1;
422        }
423        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
424            return t[t[k].l].w;
425        } else {
426            return query(t[k].l, l, r) + query(t[k].r, l, r);
427        }
428    }
429
430    void self_update(int k, int l, int r) {
431        if (t[k].l == t[k].r) {
432            t[k].w = t[t[k].l].w;
433            t[k].lazy = -1;
434            return;
435        }
436        if (t[k].lazy != -1) {
437            t[t[k].l].w += t[k].lazy;
438            t[t[k].r].w += t[k].lazy;
439            t[k].lazy = -1;
440        }
441        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
442            t[t[k].l].w = t[t[k].l].w;
443            t[t[k].r].w = t[t[k].r].w;
444        } else {
445            self_update(t[k].l, l, r);
446            self_update(t[k].r, l, r);
447        }
448    }
449
450    void self_query(int k, int l, int r) {
451        if (t[k].l == t[k].r) {
452            return t[t[k].l].w;
453        }
454        if (t[k].lazy != -1) {
455            t[t[k].l].w += t[k].lazy;
456            t[t[k].r].w += t[k].lazy;
457            t[k].lazy = -1;
458        }
459        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
460            return t[t[k].l].w;
461        } else {
462            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
463        }
464    }
465
466    void pushdown(int k) {
467        if (t[k].lazy == -1) {
468            return;
469        }
470        if (t[t[k].l].l == t[t[k].l].r) {
471            t[t[k].l].w += t[k].lazy;
472            t[t[k].l].lazy = -1;
473        } else {
474            t[t[k].l].w += t[k].lazy;
475            t[t[k].r].w += t[k].lazy;
476            t[t[k].l].lazy = -1;
477            t[t[k].r].lazy = -1;
478        }
479        t[k].lazy = -1;
480    }
481
482    void pushup(int k) {
483        t[k].w = t[t[k].l].w + t[t[k].r].w;
484    }
485
486    void self(int k, int l, int r) {
487        if (t[k].l == t[k].r) {
488            cout << t[t[k].l].w;
489            return;
490        }
491        if (t[k].lazy != -1) {
492            t[t[k].l].w += t[k].lazy;
493            t[t[k].r].w += t[k].lazy;
494            t[k].lazy = -1;
495        }
496        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
497            cout << " ";
498            self(t[k].l, l, r);
499        } else {
500            self(t[k].l, l, r);
501            self(t[k].r, l, r);
502        }
503    }
504
505    void print() {
506        self(0, 0, n);
507    }
508
509    void update(int k, int l, int r, int v) {
510        if (t[k].l == t[k].r) {
511            t[k].w = v;
512            t[k].lazy = -1;
513            return;
514        }
515        if (t[k].lazy != -1) {
516            t[t[k].l].w += t[k].lazy;
517            t[t[k].r].w += t[k].lazy;
518            t[k].lazy = -1;
519        }
520        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
521            t[t[k].l].w += v;
522            t[t[k].r].w += v;
523        } else {
524            update(t[k].l, l, r, v);
525            update(t[k].r, l, r, v);
526        }
527    }
528
529    int query(int k, int l, int r) {
530        if (t[k].l == t[k].r) {
531            return t[t[k].l].w;
532        }
533        if (t[k].lazy != -1) {
534            t[t[k].l].w += t[k].lazy;
535            t[t[k].r].w += t[k].lazy;
536            t[k].lazy = -1;
537        }
538        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
539            return t[t[k].l].w;
540        } else {
541            return query(t[k].l, l, r) + query(t[k].r, l, r);
542        }
543    }
544
545    void self_update(int k, int l, int r) {
546        if (t[k].l == t[k].r) {
547            t[k].w = t[t[k].l].w;
548            t[k].lazy = -1;
549            return;
550        }
551        if (t[k].lazy != -1) {
552            t[t[k].l].w += t[k].lazy;
553            t[t[k].r].w += t[k].lazy;
554            t[k].lazy = -1;
555        }
556        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
557            t[t[k].l].w = t[t[k].l].w;
558            t[t[k].r].w = t[t[k].r].w;
559        } else {
560            self_update(t[k].l, l, r);
561            self_update(t[k].r, l, r);
562        }
563    }
564
565    void self_query(int k, int l, int r) {
566        if (t[k].l == t[k].r) {
567            return t[t[k].l].w;
568        }
569        if (t[k].lazy != -1) {
570            t[t[k].l].w += t[k].lazy;
571            t[t[k].r].w += t[k].lazy;
572            t[k].lazy = -1;
573        }
574        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
575            return t[t[k].l].w;
576        } else {
577            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
578        }
579    }
580
581    void pushdown(int k) {
582        if (t[k].lazy == -1) {
583            return;
584        }
585        if (t[t[k].l].l == t[t[k].l].r) {
586            t[t[k].l].w += t[k].lazy;
587            t[t[k].l].lazy = -1;
588        } else {
589            t[t[k].l].w += t[k].lazy;
590            t[t[k].r].w += t[k].lazy;
591            t[t[k].l].lazy = -1;
592            t[t[k].r].lazy = -1;
593        }
594        t[k].lazy = -1;
595    }
596
597    void pushup(int k) {
598        t[k].w = t[t[k].l].w + t[t[k].r].w;
599    }
600
601    void self(int k, int l, int r) {
602        if (t[k].l == t[k].r) {
603            cout << t[t[k].l].w;
604            return;
605        }
606        if (t[k].lazy != -1) {
607            t[t[k].l].w += t[k].lazy;
608            t[t[k].r].w += t[k].lazy;
609            t[k].lazy = -1;
610        }
611        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
612            cout << " ";
613            self(t[k].l, l, r);
614        } else {
615            self(t[k].l, l, r);
616            self(t[k].r, l, r);
617        }
618    }
619
620    void print() {
621        self(0, 0, n);
622    }
623
624    void update(int k, int l, int r, int v) {
625        if (t[k].l == t[k].r) {
626            t[k].w = v;
627            t[k].lazy = -1;
628            return;
629        }
630        if (t[k].lazy != -1) {
631            t[t[k].l].w += t[k].lazy;
632            t[t[k].r].w += t[k].lazy;
633            t[k].lazy = -1;
634        }
635        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
636            t[t[k].l].w += v;
637            t[t[k].r].w += v;
638        } else {
639            update(t[k].l, l, r, v);
640            update(t[k].r, l, r, v);
641        }
642    }
643
644    int query(int k, int l, int r) {
645        if (t[k].l == t[k].r) {
646            return t[t[k].l].w;
647        }
648        if (t[k].lazy != -1) {
649            t[t[k].l].w += t[k].lazy;
650            t[t[k].r].w += t[k].lazy;
651            t[k].lazy = -1;
652        }
653        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
654            return t[t[k].l].w;
655        } else {
656            return query(t[k].l, l, r) + query(t[k].r, l, r);
657        }
658    }
659
660    void self_update(int k, int l, int r) {
661        if (t[k].l == t[k].r) {
662            t[k].w = t[t[k].l].w;
663            t[k].lazy = -1;
664            return;
665        }
666        if (t[k].lazy != -1) {
667            t[t[k].l].w += t[k].lazy;
668            t[t[k].r].w += t[k].lazy;
669            t[k].lazy = -1;
670        }
671        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
672            t[t[k].l].w = t[t[k].l].w;
673            t[t[k].r].w = t[t[k].r].w;
674        } else {
675            self_update(t[k].l, l, r);
676            self_update(t[k].r, l, r);
677        }
678    }
679
680    void self_query(int k, int l, int r) {
681        if (t[k].l == t[k].r) {
682            return t[t[k].l].w;
683        }
684        if (t[k].lazy != -1) {
685            t[t[k].l].w += t[k].lazy;
686            t[t[k].r].w += t[k].lazy;
687            t[k].lazy = -1;
688        }
689        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
690            return t[t[k].l].w;
691        } else {
692            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
693        }
694    }
695
696    void pushdown(int k) {
697        if (t[k].lazy == -1) {
698            return;
699        }
700        if (t[t[k].l].l == t[t[k].l].r) {
701            t[t[k].l].w += t[k].lazy;
702            t[t[k].l].lazy = -1;
703        } else {
704            t[t[k].l].w += t[k].lazy;
705            t[t[k].r].w += t[k].lazy;
706            t[t[k].l].lazy = -1;
707            t[t[k].r].lazy = -1;
708        }
709        t[k].lazy = -1;
710    }
711
712    void pushup(int k) {
713        t[k].w = t[t[k].l].w + t[t[k].r].w;
714    }
715
716    void self(int k, int l, int r) {
717        if (t[k].l == t[k].r) {
718            cout << t[t[k].l].w;
719            return;
720        }
721        if (t[k].lazy != -1) {
722            t[t[k].l].w += t[k].lazy;
723            t[t[k].r].w += t[k].lazy;
724            t[k].lazy = -1;
725        }
726        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
727            cout << " ";
728            self(t[k].l, l, r);
729        } else {
730            self(t[k].l, l, r);
731            self(t[k].r, l, r);
732        }
733    }
734
735    void print() {
736        self(0, 0, n);
737    }
738
739    void update(int k, int l, int r, int v) {
740        if (t[k].l == t[k].r) {
741            t[k].w = v;
742            t[k].lazy = -1;
743            return;
744        }
745        if (t[k].lazy != -1) {
746            t[t[k].l].w += t[k].lazy;
747            t[t[k].r].w += t[k].lazy;
748            t[k].lazy = -1;
749        }
750        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
751            t[t[k].l].w += v;
752            t[t[k].r].w += v;
753        } else {
754            update(t[k].l, l, r, v);
755            update(t[k].r, l, r, v);
756        }
757    }
758
759    int query(int k, int l, int r) {
760        if (t[k].l == t[k].r) {
761            return t[t[k].l].w;
762        }
763        if (t[k].lazy != -1) {
764            t[t[k].l].w += t[k].lazy;
765            t[t[k].r].w += t[k].lazy;
766            t[k].lazy = -1;
767        }
768        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
769            return t[t[k].l].w;
770        } else {
771            return query(t[k].l, l, r) + query(t[k].r, l, r);
772        }
773    }
774
775    void self_update(int k, int l, int r) {
776        if (t[k].l == t[k].r) {
777            t[k].w = t[t[k].l].w;
778            t[k].lazy = -1;
779            return;
780        }
781        if (t[k].lazy != -1) {
782            t[t[k].l].w += t[k].lazy;
783            t[t[k].r].w += t[k].lazy;
784            t[k].lazy = -1;
785        }
786        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
787            t[t[k].l].w = t[t[k].l].w;
788            t[t[k].r].w = t[t[k].r].w;
789        } else {
790            self_update(t[k].l, l, r);
791            self_update(t[k].r, l, r);
792        }
793    }
794
795    void self_query(int k, int l, int r) {
796        if (t[k].l == t[k].r) {
797            return t[t[k].l].w;
798        }
799        if (t[k].lazy != -1) {
800            t[t[k].l].w += t[k].lazy;
801            t[t[k].r].w += t[k].lazy;
802            t[k].lazy = -1;
803        }
804        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
805            return t[t[k].l].w;
806        } else {
807            return self_query(t[k].l, l, r) + self_query(t[k].r, l, r);
808        }
809    }
810
811    void pushdown(int k) {
812        if (t[k].lazy == -1) {
813            return;
814        }
815        if (t[t[k].l].l == t[t[k].l].r) {
816            t[t[k].l].w += t[k].lazy;
817            t[t[k].l].lazy = -1;
818        } else {
819            t[t[k].l].w += t[k].lazy;
820            t[t[k].r].w += t[k].lazy;
821            t[t[k].l].lazy = -1;
822            t[t[k].r].lazy = -1;
823        }
824        t[k].lazy = -1;
825    }
826
827    void pushup(int k) {
828        t[k].w = t[t[k].l].w + t[t[k].r].w;
829    }
830
831    void self(int k, int l, int r) {
832        if (t[k].l == t[k].r) {
833            cout << t[t[k].l].w;
834            return;
835        }
836        if (t[k].lazy != -1) {
837            t[t[k].l].w += t[k].lazy;
838            t[t[k].r].w += t[k].lazy;
839            t[k].lazy = -1;
840        }
841        if (l <= t[t[k].l].r && r >= t[t[k].r].l) {
842            cout << " ";
843            self(t[k].l, l, r);
844        } else {
845            self(t[k].l, l, r);
846            self(t[k].r, l, r);
847        }
848    }
849
8
```

```

29     };
30     build(build, 1, n);
31 }
32 void pushdown(node &p, int lazy) { /* 【在此更新下递函数】 */
33     p.w += (p.r - p.l + 1) * lazy;
34     p.lazy += lazy;
35 }
36 void pushdown(int k) { // 不需要动
37     if (t[k].lazy == 0) return;
38     pushdown(t[GL], t[k].lazy);
39     pushdown(t[GR], t[k].lazy);
40     t[k].lazy = 0;
41 }
42 void pushup(int k) { // 不需要动
43     auto pushup = [&](node &p, node &l, node &r) { /* 【在此更新上传函数】 */
44         p.w = l.w + r.w;
45     };
46     pushup(t[k], t[GL], t[GR]);
47 }
48 void modify(int l, int r, int val, int k = 1) {
49     if (l <= t[k].l && t[k].r <= r) {
50         pushdown(t[k], val);
51         return;
52     }
53     pushdown(k);
54     int mid = (t[k].l + t[k].r) / 2;
55     if (l <= mid) modify(l, r, val, GL);
56     if (mid < r) modify(l, r, val, GR);
57     pushup(k);
58 }
59 int ask(int l, int r, int k = 1) {
60     if (l <= t[k].l && t[k].r <= r) {
61         return t[k].w;
62     }
63     pushdown(k);
64     int mid = (t[k].l + t[k].r) / 2;
65     int ans = 0;
66     if (l <= mid) ans += ask(l, r, GL);
67     if (mid < r) ans += ask(l, r, GR);
68     return ans;
69 }
70 };
71
72 struct HLD {
73     int n, idx;
74     vector<vector<int>> ver;
75     vector<int> siz, dep;
76     vector<int> top, son, parent;
77     vector<int> in, id, val;
78     Segt segt;
79
80     HLD(int n) {
81         this->n = n;
82         ver.resize(n + 1);
83         siz.resize(n + 1);
84         dep.resize(n + 1);
85
86         top.resize(n + 1);
87         son.resize(n + 1);
88         parent.resize(n + 1);
89
90         idx = 0;
91         in.resize(n + 1);
92         id.resize(n + 1);

```

```

93     val.resize(n + 1);
94 }
95 void add(int x, int y) { // 建立双向边
96     ver[x].push_back(y);
97     ver[y].push_back(x);
98 }
99 void dfs1(int x) {
100     siz[x] = 1;
101     dep[x] = dep[parent[x]] + 1;
102     for (auto y : ver[x]) {
103         if (y == parent[x]) continue;
104         parent[y] = x;
105         dfs1(y);
106         siz[x] += siz[y];
107         if (siz[y] > siz[son[x]]) {
108             son[x] = y;
109         }
110     }
111 }
112 void dfs2(int x, int up) {
113     id[x] = ++idx;
114     val[idx] = in[x]; // 建立编号
115     top[x] = up;
116     if (son[x]) dfs2(son[x], up);
117     for (auto y : ver[x]) {
118         if (y == parent[x] || y == son[x]) continue;
119         dfs2(y, y);
120     }
121 }
122 void modify(int l, int r, int val) { // 链上修改
123     while (top[l] != top[r]) {
124         if (dep[top[l]] < dep[top[r]]) {
125             swap(l, r);
126         }
127         segt.modify(id[top[l]], id[l], val);
128         l = parent[top[l]];
129     }
130     if (dep[l] > dep[r]) {
131         swap(l, r);
132     }
133     segt.modify(id[l], id[r], val);
134 }
135 void modify(int root, int val) { // 子树修改
136     segt.modify(id[root], id[root] + siz[root] - 1, val);
137 }
138 int ask(int l, int r) { // 链上查询
139     int ans = 0;
140     while (top[l] != top[r]) {
141         if (dep[top[l]] < dep[top[r]]) {
142             swap(l, r);
143         }
144         ans += segt.ask(id[top[l]], id[l]);
145         l = parent[top[l]];
146     }
147     if (dep[l] > dep[r]) {
148         swap(l, r);
149     }
150     return ans + segt.ask(id[l], id[r]);
151 }
152 int ask(int root) { // 子树查询
153     return segt.ask(id[root], id[root] + siz[root] - 1);
154 }
155 void work(auto in, int root = 1) { // 在此初始化
156     assert(in.size() == n + 1);

```

```

157     this->in = in;
158     dfs1(root);
159     dfs2(root, root);
160     segt.init(val); // 建立线段树
161 }
162 void work(int root = 1) { // 在此初始化
163     dfs1(root);
164     dfs2(root, root);
165     segt.init(val); // 建立线段树
166 }
167 };

```

8.7 小波矩阵树：高效静态区间第 K 大查询

手写 `bitset` 压位，以 $\mathcal{O}(N \log N)$ 的时间复杂度和 $\mathcal{O}(N + \frac{N \log N}{64})$ 的空间建树后，实现单次 $\mathcal{O}(\log N)$ 复杂度的区间第 k 大值询问。已经过偏移，请使用 $1\text{-}idx$ 。

```

1 #define __count(x) __builtin_popcountll(x)
2 struct Wavelet {
3     vector<int> val, sum;
4     vector<u64> bit;
5     int t, n;
6
7     int getSum(int i) {
8         return sum[i >> 6] + __count(bit[i >> 6] & ((1ULL << (i & 63)) - 1));
9     }
10
11    Wavelet(vector<int> v) : val(v), n(v.size()) {
12        sort(val.begin(), val.end());
13        val.erase(unique(val.begin(), val.end()), val.end());
14
15        int n_ = val.size();
16        t = __lg(2 * n_ - 1);
17        bit.resize((t * n_ + 64) >> 6);
18        sum.resize(bit.size());
19        vector<int> cnt(n_ + 1);
20
21        for (int &x : v) {
22            x = lower_bound(val.begin(), val.end(), x) - val.begin();
23            cnt[x + 1]++;
24        }
25        for (int i = 1; i < n_; ++i) {
26            cnt[i] += cnt[i - 1];
27        }
28        for (int j = 0; j < t; ++j) {
29            for (int i : v) {
30                int tmp = i >> (t - 1 - j);
31                int pos = (tmp >> 1) << (t - j);
32                auto setBit = [&](int i, u64 v) {
33                    bit[i >> 6] |= (v << (i & 63));
34                };
35                setBit(j * n + cnt[pos], tmp & 1);
36                cnt[pos]++;
37            }
38            for (int i : v) {
39                cnt[(i >> (t - j)) << (t - j)]--;
40            }
41        }
42        for (int i = 1; i < sum.size(); ++i) {
43            sum[i] = sum[i - 1] + __count(bit[i - 1]);
44        }
45    }

```

```

46     int small(int l, int r, int k) {
47         r++, k--;
48         for (int j = 0, x = 0, y = n, res = 0;; ++j) {
49             if (j == t) return val[res];
50             int A = getSum(n * j + x), B = getSum(n * j + 1);
51             int C = getSum(n * j + r), D = getSum(n * j + y);
52             int ab_zeros = r - l - C + B;
53             if (ab_zeros > k) {
54                 res = res << 1;
55                 y -= D - A;
56                 l -= B - A;
57                 r -= C - A;
58             } else {
59                 res = (res << 1) | 1;
60                 k -= ab_zeros;
61                 x += y - x - D + A;
62                 l += y - l - D + B;
63                 r += y - r - D + C;
64             }
65         }
66     }
67 }
68 int large(int l, int r, int k) {
69     return small(l, r, r - l - k);
70 }
71 };

```

8.8 普通莫队

以 $\mathcal{O}(N\sqrt{N})$ 的复杂度完成 Q 次查询的离线查询，其中每个分块的大小取 $\sqrt{N} = \sqrt{10^5} = 317$ ，也可以使用 `n / min<int>(n, sqrt(q))`、`ceil((double)n / (int)sqrt(n))` 或者 `sqrt(n)` 划分。

```

1 signed main() {
2     int n;
3     cin >> n;
4     vector<int> w(n + 1);
5     for (int i = 1; i <= n; i++) {
6         cin >> w[i];
7     }
8
9     int q;
10    cin >> q;
11    vector<array<int, 3>> query(q + 1);
12    for (int i = 1; i <= q; i++) {
13        int l, r;
14        cin >> l >> r;
15        query[i] = {l, r, i};
16    }
17
18    int Knum = n / min<int>(n, sqrt(q)); // 计算块长
19    vector<int> K(n + 1);
20    for (int i = 1; i <= n; i++) { // 固定块长
21        K[i] = (i - 1) / Knum + 1;
22    }
23    sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
24        if (K[x[0]] != K[y[0]]) return x[0] < y[0];
25        if (K[x[0]] & 1) return x[1] < y[1];
26        return x[1] > y[1];
27    });
28
29    int l = 1, r = 0, val = 0;
30    vector<int> ans(q + 1);

```

```

31     for (int i = 1; i <= q; i++) {
32         auto [ql, qr, id] = query[i];
33         auto add = [&](int x) -> void {};
34         auto del = [&](int x) -> void {};
35         while (l > ql) add(w[--l]);
36         while (r < qr) add(w[++r]);
37         while (l < ql) del(w[l++]);
38         while (r > qr) del(w[r--]);
39         ans[id] = val;
40     }
41     for (int i = 1; i <= q; i++) {
42         cout << ans[i] << endl;
43     }
44 }
```

需要注意的是，在普通莫队中，`K` 数组的作用是根据左边界值进行排序，当询问次数很少时 ($q \ll n$)，可以直接合并到 `query` 数组中。

```

1 vector<array<int, 4>> query(q);
2 for (int i = 1; i <= q; i++) {
3     int l, r;
4     cin >> l >> r;
5     query[i] = {l, r, i, (l - 1) / Knum + 1}; // 合并
6 }
7 sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
8     if (x[3] != y[3]) return x[3] < y[3];
9     if (x[3] & 1) return x[1] < y[1];
10    return x[1] > y[1];
11});
```

8.9 带修改的莫队（带时间维度的莫队）

以 $\mathcal{O}(N^{\frac{5}{3}})$ 的复杂度完成 Q 次询问的离线查询，其中每个分块的大小取 $N^{\frac{2}{3}} = \sqrt[3]{100000^2} = 2154$ （直接取会略快），也可以使用 `pow(n, 0.6666)` 划分。

```

1 signed main() {
2     int n, q;
3     cin >> n >> q;
4     vector<int> w(n + 1);
5     for (int i = 1; i <= n; i++) {
6         cin >> w[i];
7     }
8
9     vector<array<int, 4>> query = {{}}; // {左区间, 右区间, 累计修改次数, 下标}
10    vector<array<int, 2>> modify = {{}}; // {修改的值, 修改的元素下标}
11    for (int i = 1; i <= q; i++) {
12        char op;
13        cin >> op;
14        if (op == 'Q') {
15            int l, r;
16            cin >> l >> r;
17            query.push_back({l, r, (int)modify.size() - 1, (int)query.size()});
18        } else {
19            int idx, w;
20            cin >> idx >> w;
21            modify.push_back({w, idx});
22        }
23    }
24
25    int Knum = 2154; // 计算块长
```

```

26     vector<int> K(n + 1);
27     for (int i = 1; i <= n; i++) { // 固定块长
28         K[i] = (i - 1) / Knum + 1;
29     }
30     sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
31         if (K[x[0]] != K[y[0]]) return x[0] < y[0];
32         if (K[x[1]] != K[y[1]]) return x[1] < y[1];
33         return x[3] < y[3];
34     });
35
36     int l = 1, r = 0, val = 0;
37     int t = 0; // 累计修改次数
38     vector<int> ans(query.size());
39     for (int i = 1; i < query.size(); i++) {
40         auto [ql, qr, qt, id] = query[i];
41         auto add = [&](int x) -> void {};
42         auto del = [&](int x) -> void {};
43         auto time = [&](int x, int l, int r) -> void {};
44         while (l > ql) add(w[--l]);
45         while (r < qr) add(w[++r]);
46         while (l < ql) del(w[l++]);
47         while (r > qr) del(w[r--]);
48         while (t < qt) time(++t, ql, qr);
49         while (t > qt) time(t--, ql, qr);
50         ans[id] = val;
51     }
52     for (int i = 1; i < ans.size(); i++) {
53         cout << ans[i] << endl;
54     }
55 }
```

8.10 分数运算类

定义了分数的四则运算，如果需要处理浮点数，那么需要将函数中的 `gcd` 运算替换为 `fgcd`。

```

1 template<class T> struct Frac {
2     T x, y;
3     Frac() : Frac(0, 1) {}
4     Frac(T x_) : Frac(x_, 1) {}
5     Frac(T x_, T y_) : x(x_), y(y_) {
6         if (y < 0) {
7             y = -y;
8             x = -x;
9         }
10    }
11
12    constexpr double val() const {
13        return 1. * x / y;
14    }
15    constexpr Frac norm() const { // 调整符号、转化为最简形式
16        T p = gcd(x, y);
17        return {x / p, y / p};
18    }
19    friend constexpr auto &operator<<(ostream &o, const Frac &j) {
20        T p = gcd(j.x, j.y);
21        if (j.y == p) {
22            return o << j.x / p;
23        } else {
24            return o << j.x / p << "/" << j.y / p;
25        }
26    }
27    constexpr Frac &operator/=(const Frac &i) {
```

```

28     x *= i.y;
29     y *= i.x;
30     if (y < 0) {
31         x = -x;
32         y = -y;
33     }
34     return *this;
35 }
36 constexpr Frac &operator+=(const Frac &i) { return x = x * i.y + y * i.x, y *=
i.y, *this; }
37 constexpr Frac &operator-=(const Frac &i) { return x = x * i.y - y * i.x, y *=
i.y, *this; }
38 constexpr Frac &operator*=(const Frac &i) { return x *= i.x, y *= i.y, *this; }
39 friend constexpr Frac operator+(const Frac i, const Frac j) { return i += j; }
40 friend constexpr Frac operator-(const Frac i, const Frac j) { return i -= j; }
41 friend constexpr Frac operator*(const Frac i, const Frac j) { return i *= j; }
42 friend constexpr Frac operator/(const Frac i, const Frac j) { return i /= j; }
43 friend constexpr Frac operator-(const Frac i) { return Frac(-i.x, i.y); }
44 friend constexpr bool operator<(const Frac i, const Frac j) { return i.x * j.y <
i.y * j.x; }
45 friend constexpr bool operator>(const Frac i, const Frac j) { return i.x * j.y >
i.y * j.x; }
46 friend constexpr bool operator==(const Frac i, const Frac j) { return i.x * j.y ==
i.y * j.x; }
47 friend constexpr bool operator!=(const Frac i, const Frac j) { return i.x * j.y !=
i.y * j.x; }
48 };

```

8.11 主席树（可持久化线段树）

以 $\mathcal{O}(N \log N)$ 的时间复杂度建树、查询、修改。

```

1 struct PresidentTree {
2     static constexpr int N = 2e5 + 10;
3     int cntNodes, root[N];
4     struct node {
5         int l, r;
6         int cnt;
7     } tr[4 * N + 17 * N];
8     void modify(int &u, int v, int l, int r, int x) {
9         u = ++cntNodes;
10        tr[u] = tr[v];
11        tr[u].cnt++;
12        if (l == r) return;
13        int mid = (l + r) / 2;
14        if (x <= mid)
15            modify(tr[u].l, tr[v].l, l, mid, x);
16        else
17            modify(tr[u].r, tr[v].r, mid + 1, r, x);
18    }
19    int kth(int u, int v, int l, int r, int k) {
20        if (l == r) return l;
21        int res = tr[tr[v].l].cnt - tr[tr[u].l].cnt;
22        int mid = (l + r) / 2;
23        if (k <= res)
24            return kth(tr[u].l, tr[v].l, l, mid, k);
25        else
26            return kth(tr[u].r, tr[v].r, mid + 1, r, k - res);
27    }
28 };

```

8.12 KD Tree

在第 k 维上的单次查询复杂度最坏为 $\mathcal{O}(n^{1-k^{-1}})$ 。

```

1 struct KDT {
2     constexpr static int N = 1e5 + 10, K = 2;
3     double alpha = 0.725;
4     struct node {
5         int info[K];
6         int mn[K], mx[K];
7     } tr[N];
8     int ls[N], rs[N], siz[N], id[N], d[N];
9     int idx, rt, cur;
10    int ans;
11    KDT() {
12        rt = 0;
13        cur = 0;
14        memset(ls, 0, sizeof ls);
15        memset(rs, 0, sizeof rs);
16        memset(d, 0, sizeof d);
17    }
18    void apply(int p, int son) {
19        if (son) {
20            for (int i = 0; i < K; i++) {
21                tr[p].mn[i] = min(tr[p].mn[i], tr[son].mn[i]);
22                tr[p].mx[i] = max(tr[p].mx[i], tr[son].mx[i]);
23            }
24            siz[p] += siz[son];
25        }
26    }
27    void maintain(int p) {
28        for (int i = 0; i < K; i++) {
29            tr[p].mn[i] = tr[p].info[i];
30            tr[p].mx[i] = tr[p].info[i];
31        }
32        siz[p] = 1;
33        apply(p, ls[p]);
34        apply(p, rs[p]);
35    }
36    int build(int l, int r) {
37        if (l > r) return 0;
38        vector<double> avg(K);
39        for (int i = 0; i < K; i++) {
40            for (int j = l; j <= r; j++) {
41                avg[i] += tr[id[j]].info[i];
42            }
43            avg[i] /= (r - l + 1);
44        }
45        vector<double> var(K);
46        for (int i = 0; i < K; i++) {
47            for (int j = l; j <= r; j++) {
48                var[i] += (tr[id[j]].info[i] - avg[i]) * (tr[id[j]].info[i] -
avg[i]);
49            }
50        }
51        int mid = (l + r) / 2;
52        int x = max_element(var.begin(), var.end()) - var.begin();
53        nth_element(id + l, id + mid, id + r + 1, [&](int a, int b) {
54            return tr[a].info[x] < tr[b].info[x];
55        });
56        d[id[mid]] = x;
57        ls[id[mid]] = build(l, mid - 1);
58        rs[id[mid]] = build(mid + 1, r);

```

```

59         maintain(id[mid]);
60     return id[mid];
61 }
62 void print(int p) {
63     if (!p) return;
64     print(ls[p]);
65     id[++idx] = p;
66     print(rs[p]);
67 }
68 void rebuild(int &p) {
69     idx = 0;
70     print(p);
71     p = build(1, idx);
72 }
73 bool bad(int p) {
74     return alpha * siz[p] <= max(siz[ls[p]], siz[rs[p]]);
75 }
76 void insert(int &p, int cur) {
77     if (!p) {
78         p = cur;
79         maintain(p);
80         return;
81     }
82     if (tr[p].info[d[p]] > tr[cur].info[d[p]]) insert(ls[p], cur);
83     else insert(rs[p], cur);
84     maintain(p);
85     if (bad(p)) rebuild(p);
86 }
87 void insert(vector<int> &a) {
88     cur++;
89     for (int i = 0; i < K; i++) {
90         tr[cur].info[i] = a[i];
91     }
92     insert(rt, cur);
93 }
94 bool out(int p, vector<int> &a) {
95     for (int i = 0; i < K; i++) {
96         if (a[i] < tr[p].mn[i]) {
97             return true;
98         }
99     }
100    return false;
101 }
102 bool in(int p, vector<int> &a) {
103     for (int i = 0; i < K; i++) {
104         if (a[i] < tr[p].info[i]) {
105             return false;
106         }
107     }
108     return true;
109 }
110 bool all(int p, vector<int> &a) {
111     for (int i = 0; i < K; i++) {
112         if (a[i] < tr[p].mx[i]) {
113             return false;
114         }
115     }
116     return true;
117 }
118 void query(int p, vector<int> &a) {
119     if (!p) return;
120     if (out(p, a)) return;
121     if (all(p, a)) {
122         ans += siz[p];

```

```

123         return;
124     }
125     if (in(p, a)) ans++;
126     query(ls[p], a);
127     query(rs[p], a);
128 }
129 int query(vector<int> &a) {
130     ans = 0;
131     query(rt, a);
132     return ans;
133 }
134 };

```

8.13 ST 表

用于解决区间可重复贡献问题，需要满足 x 运算符 $x = x$ （如区间最大值： $\max(x, x) = x$ 、区间 \gcd ： $\gcd(x, x) = x$ 等），但是不支持修改操作。 $\mathcal{O}(N \log N)$ 预处理， $\mathcal{O}(1)$ 查询。

```

1 struct ST {
2     const int n, k;
3     vector<int> in1, in2;
4     vector<vector<int>> Max, Min;
5     ST(int n) : n(n), in1(n + 1), in2(n + 1), k(__lg(n)) {
6         Max.resize(k + 1, vector<int>(n + 1));
7         Min.resize(k + 1, vector<int>(n + 1));
8     }
9     void init() {
10        for (int i = 1; i <= n; i++) {
11            Max[0][i] = in1[i];
12            Min[0][i] = in2[i];
13        }
14        for (int i = 0, t = 1; i < k; i++, t <<= 1) {
15            const int T = n - (t << 1) + 1;
16            for (int j = 1; j <= T; j++) {
17                Max[i + 1][j] = max(Max[i][j], Max[i][j + t]);
18                Min[i + 1][j] = min(Min[i][j], Min[i][j + t]);
19            }
20        }
21    }
22    int getMax(int l, int r) {
23        if (l > r) {
24            swap(l, r);
25        }
26        int k = __lg(r - l + 1);
27        return max(Max[k][l], Max[k][r - (1 << k) + 1]);
28    }
29    int getMin(int l, int r) {
30        if (l > r) {
31            swap(l, r);
32        }
33        int k = __lg(r - l + 1);
34        return min(Min[k][l], Min[k][r - (1 << k) + 1]);
35    }
36 };

```

8.14 基于状压的线性 RMQ 算法

严格 $\mathcal{O}(N)$ 预处理， $\mathcal{O}(1)$ 查询。

```

1  template<class T, class Cmp = less<T>> struct RMQ {
2      const Cmp cmp = Cmp();
3      static constexpr unsigned B = 64;
4      using u64 = unsigned long long;
5      int n;
6      vector<vector<T>> a;
7      vector<T> pre, suf, ini;
8      vector<u64> stk;
9      RMQ() {}
10     RMQ(const vector<T> &v) {
11         init(v);
12     }
13     void init(const vector<T> &v) {
14         n = v.size();
15         pre = suf = ini = v;
16         stk.resize(n);
17         if (!n) {
18             return;
19         }
20         const int M = (n - 1) / B + 1;
21         const int lg = __lg(M);
22         a.assign(lg + 1, vector<T>(M));
23         for (int i = 0; i < M; i++) {
24             a[0][i] = v[i * B];
25             for (int j = 1; j < B && i * B + j < n; j++) {
26                 a[0][i] = min(a[0][i], v[i * B + j], cmp);
27             }
28         }
29         for (int i = 1; i < n; i++) {
30             if (i % B) {
31                 pre[i] = min(pre[i], pre[i - 1], cmp);
32             }
33         }
34         for (int i = n - 2; i >= 0; i--) {
35             if (i % B != B - 1) {
36                 suf[i] = min(suf[i], suf[i + 1], cmp);
37             }
38         }
39         for (int j = 0; j < lg; j++) {
40             for (int i = 0; i + (2 << j) <= M; i++) {
41                 a[j + 1][i] = min(a[j][i], a[j][i + (1 << j)], cmp);
42             }
43         }
44         for (int i = 0; i < M; i++) {
45             const int l = i * B;
46             const int r = min(1U * n, l + B);
47             u64 s = 0;
48             for (int j = l; j < r; j++) {
49                 while (s && cmp(v[j], v[__lg(s) + 1])) {
50                     s ^= 1ULL << __lg(s);
51                 }
52                 s |= 1ULL << (j - 1);
53                 stk[j] = s;
54             }
55         }
56     }
57     T operator()(int l, int r) {
58         if (l / B != (r - 1) / B) {
59             T ans = min(suf[l], pre[r - 1], cmp);

```

```

60         l = l / B + 1;
61         r = r / B;
62         if (l < r) {
63             int k = __lg(r - 1);
64             ans = min({ans, a[k][1], a[k][r - (1 << k)]}, cmp);
65         }
66         return ans;
67     } else {
68         int x = B * (l / B);
69         return ini[__builtin_ctzll(stk[r - 1] >> (l - x)) + 1];
70     }
71 }
72 };

```

8.15 pbds 扩展库实现平衡二叉树

记得加上相应的头文件，同时需要注意定义时的参数，一般只需要修改第三个参数：即定义的是大根堆还是小根堆。

附常见成员函数：

```

1 empty() / size()
2 insert(x) // 插入元素x
3 erase(x) // 删除元素/迭代器x
4 order_of_key(x) // 返回元素x的排名
5 find_by_order(x) // 返回排名为x的元素迭代器
6 lower_bound(x) / upper_bound(x) // 返回迭代器
7 join(Tree) // 将Tree树的全部元素并入当前的树
8 split(x, Tree) // 将大于x的元素放入Tree树

```

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 using namespace __gnu_pbds;
3 using V = pair<int, int>;
4 tree<V, null_type, less<V>, rb_tree_tag, tree_order_statistics_node_update> ver;
5 map<int, int> dic;
6
7 int n; cin >> n;
8 for (int i = 1, op, x; i <= n; i++) {
9     cin >> op >> x;
10    if (op == 1) { // 插入一个元素x，允许重复
11        ver.insert({x, ++dic[x]});
12    } else if (op == 2) { // 删除元素x，若有重复，则任意删除一个
13        ver.erase({x, dic[x]-1});
14    } else if (op == 3) { // 查询元素x的排名（排名定义为比当前数小的数的个数+1）
15        cout << ver.order_of_key({x, 1}) + 1 << endl;
16    } else if (op == 4) { // 查询排名为x的元素
17        cout << ver.find_by_order(--x)->first << endl;
18    } else if (op == 5) { // 查询元素x的前驱
19        int idx = ver.order_of_key({x, 1}) - 1; // 无论x存不存在，idx都代表x的位置，需要-1
20        cout << ver.find_by_order(idx)->first << endl;
21    } else if (op == 6) { // 查询元素x的后继
22        int idx = ver.order_of_key({x, dic[x]}); // 如果x不存在，那么idx就是x的后继
23        if (ver.find({x, 1}) != ver.end()) idx++; // 如果x存在，那么idx是x的位置，需要+1
24        cout << ver.find_by_order(idx)->first << endl;
25    }
26 }

```

8.16 vector 模拟实现平衡二叉树

```

1 #define ALL(x) x.begin(), x.end()
2 #define pre lower_bound
3 #define suf upper_bound
4 int n; cin >> n;
5 vector<int> ver;
6 for (int i = 1, op, x; i <= n; i++) {
7     cin >> op >> x;
8     if (op == 1) ver.insert(pre(ALL(ver)), x), x);
9     if (op == 2) ver.erase(pre(ALL(ver)), x));
10    if (op == 3) cout << pre(ALL(ver), x) - ver.begin() + 1 << endl;
11    if (op == 4) cout << ver[x - 1] << endl;
12    if (op == 5) cout << ver[pre(ALL(ver), x) - ver.begin() - 1] << endl;
13    if (op == 6) cout << ver[suf(ALL(ver), x) - ver.begin()] << endl;
14 }

```

8.17 线性基（高斯消元法）

设向量长度为 N （一般取 63），总数为 M ，时间复杂度为 $\mathcal{O}(NM)$ 。

```

1 struct LB { // Linear Basis
2     using i64 = long long;
3     const int BASE = 63;
4     vector<i64> d, p;
5     int cnt, flag;
6
7     LB() {
8         d.resize(BASE + 1);
9         p.resize(BASE + 1);
10        cnt = flag = 0;
11    }
12    bool insert(i64 val) {
13        for (int i = BASE - 1; i >= 0; i--) {
14            if (val & (1ll << i)) {
15                if (!d[i]) {
16                    d[i] = val;
17                    return true;
18                }
19                val ^= d[i];
20            }
21        }
22        flag = 1; // 可以异或出0
23        return false;
24    }
25    bool check(i64 val) { // 判断 val 是否能被异或得到
26        for (int i = BASE - 1; i >= 0; i--) {
27            if (val & (1ll << i)) {
28                if (!d[i]) {
29                    return false;
30                }
31                val ^= d[i];
32            }
33        }
34        return true;
35    }
36    i64 ask_max() {
37        i64 res = 0;
38        for (int i = BASE - 1; i >= 0; i--) {
39            if ((res ^ d[i]) > res) res ^= d[i];
40        }

```

```

41     return res;
42 }
43 i64 ask_min() {
44     if (flag) return 0; // 特判 0
45     for (int i = 0; i <= BASE - 1; i++) {
46         if (d[i]) return d[i];
47     }
48 }
49 void rebuild() { // 第k小值独立预处理
50     for (int i = BASE - 1; i >= 0; i--) {
51         for (int j = i - 1; j >= 0; j--) {
52             if (d[i] & (1ll << j)) d[i] ^= d[j];
53         }
54     }
55     for (int i = 0; i <= BASE - 1; i++) {
56         if (d[i]) p[cnt++] = d[i];
57     }
58 }
59 i64 kthquery(i64 k) { // 查询能被异或得到的第 k 小值，如不存在则返回 -1
60     if (flag) k--; // 特判 0，如果不需要 0，直接删去
61     if (!k) return 0;
62     i64 res = 0;
63     if (k >= (1ll << cnt)) return -1;
64     for (int i = BASE - 1; i >= 0; i--) {
65         if (k & (1ll << i)) res ^= p[i];
66     }
67     return res;
68 }
69 void Merge(const LB &b) { // 合并两个线性基
70     for (int i = BASE - 1; i >= 0; i--) {
71         if (b.d[i]) {
72             insert(b.d[i]);
73         }
74     }
75 }
76 };

```

8.18 珂朵莉树 (OD Tree)

区间赋值的数据结构都可以骗分，在数据随机的情况下，复杂度可以保证，时间复杂度： $\mathcal{O}(N \log \log N)$ 。

```

1 struct ODT {
2     struct node {
3         int l, r;
4         mutable LL v;
5         node(int l, int r = -1, LL v = 0) : l(l), r(r), v(v) {}
6         bool operator<(const node &o) const {
7             return l < o.l;
8         }
9     };
10    set<node> s;
11    ODT() {
12        s.clear();
13    }
14    auto split(int pos) {
15        auto it = s.lower_bound(node(pos));
16        if (it != s.end() && it->l == pos) return it;
17        it--;
18        int l = it->l, r = it->r;
19        LL v = it->v;
20        s.erase(it);

```

```

21     s.insert(node(l, pos - 1, v));
22     return s.insert(node(pos, r, v)).first;
23 }
24 void assign(int l, int r, LL x) {
25     auto itr = split(r + 1), itl = split(l);
26     s.erase(itl, itr);
27     s.insert(node(l, r, x));
28 }
29 void add(int l, int r, LL x) {
30     auto itr = split(r + 1), itl = split(l);
31     for (auto it = itl; it != itr; it++) {
32         it->v += x;
33     }
34 }
35 LL kth(int l, int r, int k) {
36     vector<pair<LL, int>> a;
37     auto itr = split(r + 1), itl = split(l);
38     for (auto it = itl; it != itr; it++) {
39         a.push_back(pair<LL, int>(it->v, it->r - it->l + 1));
40     }
41     sort(a.begin(), a.end());
42     for (auto [val, len] : a) {
43         k -= len;
44         if (k <= 0) return val;
45     }
46 }
47 LL power(LL a, int b, int mod) {
48     a %= mod;
49     LL res = 1;
50     for (; b; b /= 2, a = a * a % mod) {
51         if (b % 2) {
52             res = res * a % mod;
53         }
54     }
55     return res;
56 }
57 LL powersum(int l, int r, int x, int mod) {
58     auto itr = split(r + 1), itl = split(l);
59     LL ans = 0;
60     for (auto it = itl; it != itr; it++) {
61         ans = (ans + power(it->v, x, mod) * (it->r - it->l + 1) % mod) % mod;
62     }
63     return ans;
64 }
65 };

```

8.19 取模运算类

集成了常见的取模四则运算，运算速度与手动取模相差无几，效率极高。

```

1 using i64 = long long;
2
3 template<class T> constexpr T mypow(T n, i64 k) {
4     T r = 1;
5     for (; k; k /= 2, n *= n) {
6         if (k % 2) {
7             r *= n;
8         }
9     }
10    return r;
11 }
12

```

```

13 template<class T> constexpr T power(int n) {
14     return mypow(T(2), n);
15 }
16
17 template<const int &MOD> struct Zmod {
18     int x;
19     Zmod(signed x = 0) : x(norm(x % MOD)) {}
20     Zmod(i64 x) : x(norm(x % MOD)) {}
21
22     constexpr int norm(int x) const noexcept {
23         if (x < 0) [[unlikely]] {
24             x += MOD;
25         }
26         if (x >= MOD) [[unlikely]] {
27             x -= MOD;
28         }
29         return x;
30     }
31     explicit operator int() const {
32         return x;
33     }
34     constexpr int val() const {
35         return x;
36     }
37     constexpr Zmod operator-() const {
38         Zmod val = norm(MOD - x);
39         return val;
40     }
41     constexpr Zmod inv() const {
42         assert(x != 0);
43         return mypow(*this, MOD - 2);
44     }
45     friend constexpr auto &operator>>(istream &in, Zmod &j) {
46         int v;
47         in >> v;
48         j = Zmod(v);
49         return in;
50     }
51     friend constexpr auto &operator<<(ostream &o, const Zmod &j) {
52         return o << j.val();
53     }
54     constexpr Zmod &operator++() {
55         x = norm(x + 1);
56         return *this;
57     }
58     constexpr Zmod &operator--() {
59         x = norm(x - 1);
60         return *this;
61     }
62     constexpr Zmod operator++(signed) {
63         Zmod res = *this;
64         ++*this;
65         return res;
66     }
67     constexpr Zmod operator--(signed) {
68         Zmod res = *this;
69         --*this;
70         return res;
71     }
72     constexpr Zmod &operator+=(const Zmod &i) {
73         x = norm(x + i.x);
74         return *this;
75     }
76     constexpr Zmod &operator-=(const Zmod &i) {

```

```

77     x = norm(x - i.x);
78     return *this;
79 }
80 constexpr Zmod &operator*=(const Zmod &i) {
81     x = i64(x) * i.x % MOD;
82     return *this;
83 }
84 constexpr Zmod &operator/=(const Zmod &i) {
85     return *this *= i.inv();
86 }
87 constexpr Zmod &operator%=(const int &i) {
88     return x %= i, *this;
89 }
90 friend constexpr Zmod operator+(const Zmod i, const Zmod j) {
91     return Zmod(i) += j;
92 }
93 friend constexpr Zmod operator-(const Zmod i, const Zmod j) {
94     return Zmod(i) -= j;
95 }
96 friend constexpr Zmod operator*(const Zmod i, const Zmod j) {
97     return Zmod(i) *= j;
98 }
99 friend constexpr Zmod operator/(const Zmod i, const Zmod j) {
100    return Zmod(i) /= j;
101 }
102 friend constexpr Zmod operator%(const Zmod i, const int j) {
103    return Zmod(i) %= j;
104 }
105 friend constexpr bool operator==(const Zmod i, const Zmod j) {
106    return i.val() == j.val();
107 }
108 friend constexpr bool operator!=(const Zmod i, const Zmod j) {
109    return i.val() != j.val();
110 }
111 friend constexpr bool operator<(const Zmod i, const Zmod j) {
112    return i.val() < j.val();
113 }
114 friend constexpr bool operator>(const Zmod i, const Zmod j) {
115    return i.val() > j.val();
116 }
117 };
118
119 int MOD[] = {998244353, 1000000007};
120 using Z = Zmod<MOD[1]>;

```

8.20 大整数类（高精度计算）

```

1 const int base = 1000000000;
2 const int base_digits = 9; // 分解为九个数位一个数字
3 struct bigint {
4     vector<int> a;
5     int sign;
6
7     bigint() : sign(1) {}
8     bigint operator-() const {
9         bigint res = *this;
10        res.sign = -sign;
11        return res;
12    }
13    bigint(long long v) {
14        *this = v;
15    }

```

```

16    bigint(const string &s) {
17        read(s);
18    }
19    void operator=(const bigint &v) {
20        sign = v.sign;
21        a = v.a;
22    }
23    void operator=(long long v) {
24        a.clear();
25        sign = 1;
26        if (v < 0) sign = -1, v = -v;
27        for (; v > 0; v = v / base) {
28            a.push_back(v % base);
29        }
30    }
31
32    // 基础加减乘除
33    bigint operator+(const bigint &v) const {
34        if (sign == v.sign) {
35            bigint res = v;
36            for (int i = 0, carry = 0; i < (int)max(a.size(), v.a.size()) || carry;
37                ++i) {
38                if (i == (int)res.a.size()) {
39                    res.a.push_back(0);
40                }
41                res.a[i] += carry + (i < (int)a.size() ? a[i] : 0);
42                carry = res.a[i] >= base;
43                if (carry) {
44                    res.a[i] -= base;
45                }
46            }
47            return res;
48        }
49        return *this - (-v);
50    }
51    bigint operator-(const bigint &v) const {
52        if (sign == v.sign) {
53            if (abs() >= v.abs()) {
54                bigint res = *this;
55                for (int i = 0, carry = 0; i < (int)v.a.size() || carry; ++i) {
56                    res.a[i] -= carry + (i < (int)v.a.size() ? v.a[i] : 0);
57                    carry = res.a[i] < 0;
58                    if (carry) {
59                        res.a[i] += base;
60                    }
61                }
62                res.trim();
63                return res;
64            }
65            return -(v - *this);
66        }
67        return *this + (-v);
68    }
69    void operator*=(int v) {
70        check(v);
71        for (int i = 0, carry = 0; i < (int)a.size() || carry; ++i) {
72            if (i == (int)a.size()) {
73                a.push_back(0);
74            }
75            long long cur = a[i] * (long long)v + carry;
76            carry = (int)(cur / base);
77            a[i] = (int)(cur % base);
78        }
79        trim();
80    }

```

```

79 }
80 void operator/=(int v) {
81     check(v);
82     for (int i = (int)a.size() - 1, rem = 0; i >= 0; --i) {
83         long long cur = a[i] + rem * (long long)base;
84         a[i] = (int)(cur / v);
85         rem = (int)(cur % v);
86     }
87     trim();
88 }
89 int operator%(int v) const {
90     if (v < 0) {
91         v = -v;
92     }
93     int m = 0;
94     for (int i = a.size() - 1; i >= 0; --i) {
95         m = (a[i] + m * (long long)base) % v;
96     }
97     return m * sign;
98 }
99
100 void operator+=(const bigint &v) {
101     *this = *this + v;
102 }
103 void operator-=(const bigint &v) {
104     *this = *this - v;
105 }
106 bigint operator*(int v) const {
107     bigint res = *this;
108     res *= v;
109     return res;
110 }
111 bigint operator/(int v) const {
112     bigint res = *this;
113     res /= v;
114     return res;
115 }
116 void operator%=(const int &v) {
117     *this = *this % v;
118 }
119
120 bool operator<(const bigint &v) const {
121     if (sign != v.sign) return sign < v.sign;
122     if (a.size() != v.a.size()) return a.size() * sign < v.a.size() * v.sign;
123     for (int i = a.size() - 1; i >= 0; i--)
124         if (a[i] != v.a[i]) return a[i] * sign < v.a[i] * sign;
125     return false;
126 }
127 bool operator>(const bigint &v) const {
128     return v < *this;
129 }
130 bool operator<=(const bigint &v) const {
131     return !(v < *this);
132 }
133 bool operator>=(const bigint &v) const {
134     return !(*this < v);
135 }
136 bool operator==(const bigint &v) const {
137     return !(*this < v) && !(v < *this);
138 }
139 bool operator!=(const bigint &v) const {
140     return *this < v || v < *this;
141 }
142

```

```

143     bigint abs() const {
144         bigint res = *this;
145         res.sign *= res.sign;
146         return res;
147     }
148     void check(int v) { // 检查输入的是否为负数
149         if (v < 0) {
150             sign = -sign;
151             v = -v;
152         }
153     }
154     void trim() { // 去除前导零
155         while (!a.empty() && !a.back()) a.pop_back();
156         if (a.empty()) sign = 1;
157     }
158     bool isZero() const { // 判断是否等于零
159         return a.empty() || (a.size() == 1 && !a[0]);
160     }
161     friend bigint gcd(const bigint &a, const bigint &b) {
162         return b.isZero() ? a : gcd(b, a % b);
163     }
164     friend bigint lcm(const bigint &a, const bigint &b) {
165         return a / gcd(a, b) * b;
166     }
167     void read(const string &s) {
168         sign = 1;
169         a.clear();
170         int pos = 0;
171         while (pos < (int)s.size() && (s[pos] == '-' || s[pos] == '+')) {
172             if (s[pos] == '-') sign = -sign;
173             ++pos;
174         }
175         for (int i = s.size() - 1; i >= pos; i -= base_digits) {
176             int x = 0;
177             for (int j = max(pos, i - base_digits + 1); j <= i; j++) x = x * 10 +
s[j] - '0';
178             a.push_back(x);
179         }
180         trim();
181     }
182     friend istream &operator>>(istream &stream, bigint &v) {
183         string s;
184         stream >> s;
185         v.read(s);
186         return stream;
187     }
188     friend ostream &operator<<(ostream &stream, const bigint &v) {
189         if (v.sign == -1) stream << '-';
190         stream << (v.a.empty() ? 0 : v.a.back());
191         for (int i = (int)v.a.size() - 2; i >= 0; --i)
192             stream << setw(base_digits) << setfill('0') << v.a[i];
193         return stream;
194     }
195
196     /* 大整数乘除大整数部分 */
197     typedef vector<long long> vll;
198     bigint operator*(const bigint &v) const { // 大整数乘大整数
199         vector<int> a6 = convert_base(this->a, base_digits, 6);
200         vector<int> b6 = convert_base(v.a, base_digits, 6);
201         vll a(a6.begin(), a6.end());
202         vll b(b6.begin(), b6.end());
203         while (a.size() < b.size()) a.push_back(0);
204         while (b.size() < a.size()) b.push_back(0);
205         while (a.size() & (a.size() - 1)) a.push_back(0), b.push_back(0);

```

```

206     vll c = karatsubaMultiply(a, b);
207     bigint res;
208     res.sign = sign * v.sign;
209     for (int i = 0, carry = 0; i < (int)c.size(); i++) {
210         long long cur = c[i] + carry;
211         res.a.push_back((int)(cur % 1000000));
212         carry = (int)(cur / 1000000);
213     }
214     res.a = convert_base(res.a, 6, base_digits);
215     res.trim();
216     return res;
217 }
218 friend pair<bigint, bigint> divmod(const bigint &a1,
219                                         const bigint &b1) { // 大整数除大整数，同时返

```

回答答案与余数

```

220     int norm = base / (b1.a.back() + 1);
221     bigint a = a1.abs() * norm;
222     bigint b = b1.abs() * norm;
223     bigint q, r;
224     q.a.resize(a.a.size());
225     for (int i = a.a.size() - 1; i >= 0; i--) {
226         r *= base;
227         r += a.a[i];
228         int s1 = r.a.size() <= b.a.size() ? 0 : r.a[b.a.size()];
229         int s2 = r.a.size() <= b.a.size() - 1 ? 0 : r.a[b.a.size() - 1];
230         int d = ((long long)base * s1 + s2) / b.a.back();
231         r -= b * d;
232         while (r < 0) r += b, --d;
233         q.a[i] = d;
234     }
235     q.sign = a1.sign * b1.sign;
236     r.sign = a1.sign;
237     q.trim();
238     r.trim();
239     return make_pair(q, r / norm);
240 }
241 static vector<int> convert_base(const vector<int> &a, int old_digits, int
new_digits) {
242     vector<long long> p(max(old_digits, new_digits) + 1);
243     p[0] = 1;
244     for (int i = 1; i < (int)p.size(); i++) p[i] = p[i - 1] * 10;
245     vector<int> res;
246     long long cur = 0;
247     int cur_digits = 0;
248     for (int i = 0; i < (int)a.size(); i++) {
249         cur += a[i] * p[cur_digits];
250         cur_digits += old_digits;
251         while (cur_digits >= new_digits) {
252             res.push_back((int)(cur % p[new_digits]));
253             cur /= p[new_digits];
254             cur_digits -= new_digits;
255         }
256     }
257     res.push_back((int)cur);
258     while (!res.empty() && !res.back()) res.pop_back();
259     return res;
260 }
261 static vll karatsubaMultiply(const vll &a, const vll &b) {
262     int n = a.size();
263     vll res(n + n);
264     if (n <= 32) {
265         for (int i = 0; i < n; i++) {
266             for (int j = 0; j < n; j++) {
267                 res[i + j] += a[i] * b[j];

```

```

268         }
269     }
270     return res;
271 }
272
273     int k = n >> 1;
274     vll a1(a.begin(), a.begin() + k);
275     vll a2(a.begin() + k, a.end());
276     vll b1(b.begin(), b.begin() + k);
277     vll b2(b.begin() + k, b.end());
278
279     vll a1b1 = karatsubaMultiply(a1, b1);
280     vll a2b2 = karatsubaMultiply(a2, b2);
281
282     for (int i = 0; i < k; i++) a2[i] += a1[i];
283     for (int i = 0; i < k; i++) b2[i] += b1[i];
284
285     vll r = karatsubaMultiply(a2, b2);
286     for (int i = 0; i < (int)a1b1.size(); i++) r[i] -= a1b1[i];
287     for (int i = 0; i < (int)a2b2.size(); i++) r[i] -= a2b2[i];
288
289     for (int i = 0; i < (int)r.size(); i++) res[i + k] += r[i];
290     for (int i = 0; i < (int)a1b1.size(); i++) res[i] += a1b1[i];
291     for (int i = 0; i < (int)a2b2.size(); i++) res[i + n] += a2b2[i];
292     return res;
293 }
294
295 void operator*=(const bigint &v) {
296     *this = *this * v;
297 }
298 bigint operator/(const bigint &v) const {
299     return divmod(*this, v).first;
300 }
301 void operator/=(const bigint &v) {
302     *this = *this / v;
303 }
304 bigint operator%(const bigint &v) const {
305     return divmod(*this, v).second;
306 }
307 void operator%=(const bigint &v) {
308     *this = *this % v;
309 }
310 };

```

8.21 常见结论

题意：（区间移位问题）要求将整个序列左移/右移若干个位置，例如，原序列为 $A = (a_1, a_2, \dots, a_n)$ ，右移 x 位后变为 $A = (a_{x+1}, a_{x+2}, \dots, a_n, a_1, a_2, \dots, a_x)$ 。

区间的端点只是一个数字，即使被改变了，通过一定的转换也能够还原，所以我们可以 $\mathcal{O}(1)$ 解决这一问题。为了方便计算，我们规定下标从 0 开始，即整个线段的区间为 $[0, n]$ ，随后，使用一个偏移量 `shift` 记录。使用 `shift = (shift + x) % n;` 更新偏移量；此后的区间查询/修改前，再将坐标偏移回去即可，下方代码使用区间修改作为示例。

```

1  cin >> l >> r >> x;
2  l--; // 坐标修改为 0 开始
3  r--;
4  l = (l + shift) % n; // 偏移
5  r = (r + shift) % n;
6  if (l > r) { // 区间分离则分别操作
7      segt.modify(l, n - 1, x);
8      segt.modify(0, r, x);
9  } else {
10     segt.modify(l, r, x);
11 }

```

8.22 常见例题

题意：（带修莫队 - 维护队列）要求能够处理以下操作：

- `'Q' l r` : 询问区间 $[l, r]$ 有几个颜色；
- `'R' idx w` : 将下标 `idx` 的颜色修改为 `w`。

输入格式为：第一行 n 和 q ($1 \leq n, q \leq 133333$) 分别代表区间长度和操作数量；第二行 n 个整数 $a_1, a_2 \dots, a_n$ ($1 \leq a_i \leq 10^6$) 代表初始颜色；随后 q 行为具体操作。

```

1  const int N = 1e6 + 7;
2  signed main() {
3      int n, q;
4      cin >> n >> q;
5      vector<int> w(n + 1);
6      for (int i = 1; i <= n; i++) {
7          cin >> w[i];
8      }
9
10     vector<array<int, 4>> query = {{}}; // {左区间, 右区间, 累计修改次数, 下标}
11     vector<array<int, 2>> modify = {{}}; // {修改的值, 修改的元素下标}
12     for (int i = 1; i <= q; i++) {
13         char op;
14         cin >> op;
15         if (op == 'Q') {
16             int l, r;
17             cin >> l >> r;
18             query.push_back({l, r, (int)modify.size() - 1, (int)query.size()});
19         } else {
20             int idx, w;
21             cin >> idx >> w;
22             modify.push_back({w, idx});
23         }
24     }
25
26     int Knum = 2154; // 计算块长
27     vector<int> K(n + 1);
28     for (int i = 1; i <= n; i++) { // 固定块长
29         K[i] = (i - 1) / Knum + 1;
30     }
31     sort(query.begin() + 1, query.end(), [&](auto x, auto y) {
32         if (K[x[0]] != K[y[0]]) return x[0] < y[0];
33         if (K[x[1]] != K[y[1]]) return x[1] < y[1];
34         return x[3] < y[3];
35     });
36
37     int l = 1, r = 0, val = 0;
38     int t = 0; // 累计修改次数
39     vector<int> ans(query.size()), cnt(N);

```

```

40     for (int i = 1; i < query.size(); i++) {
41         auto [ql, qr, qt, id] = query[i];
42         auto add = [&](int x) -> void {
43             if (cnt[x] == 0) ++ val;
44             ++ cnt[x];
45         };
46         auto del = [&](int x) -> void {
47             -- cnt[x];
48             if (cnt[x] == 0) -- val;
49         };
50         auto time = [&](int x, int l, int r) -> void {
51             if (l <= modify[x][1] && modify[x][1] <= r) { //当修改的位置在询问期间内部
时才会改变num的值
52                 del(w[modify[x][1]]);
53                 add(modify[x][0]);
54             }
55             swap(w[modify[x][1]], modify[x][0]); //直接交换修改数组的值与原始值，减少额外的数组开销，且方便复原
56         };
57         while (l > ql) add(w[--l]);
58         while (r < qr) add(w[++r]);
59         while (l < ql) del(w[l++]);
60         while (r > qr) del(w[r--]);
61         while (t < qt) time(++t, ql, qr);
62         while (t > qt) time(t--, ql, qr);
63         ans[id] = val;
64     }
65     for (int i = 1; i < ans.size(); i++) {
66         cout << ans[i] << endl;
67     }
68 }
```

/END/

9 动态规划

9.1 01背包

有 n 件物品和一个容量为 W 的背包，第 i 件物品的体积为 $w[i]$ ，价值为 $v[i]$ ，求解将哪些物品装入背包中使总价值最大。

思路：

当放入一个体积为 $w[i]$ 的物品后，价值增加了 $v[i]$ ，于是我们可以构建一个二维的 $dp[i][j]$ 数组，装入第 i 件物品时，背包容量为 j 能实现的 **最大价值**，可以得到 **转移方程**
 $dp[i][j] = \max(dp[i - 1][j], dp[i - 1][j - w[i]] + v[i])$ 。

```

1 | for (int i = 1; i <= n; i++) {
2 |     for (int j = 0; j <= W; j++) {
3 |         dp[i][j] = dp[i - 1][j];
4 |         if (j >= w[i])
5 |             dp[i][j] = max(dp[i][j], dp[i - 1][j - w[i]] + v[i]);
6 |     }

```

我们可以发现，第 i 个物品的状态是由第 $i - 1$ 个物品转移过来的，每次的 j 转移过来后，第 $i - 1$ 个方程的 j 已经没用了，于是我们想到可以把二维方程压缩成 **一维** 的，用以 **优化空间复杂度**。

```

1 | for (int i = 1; i <= n; i++) //当前装第 i 件物品
2 |     for (int j = W; j >= w[i]; j--) //背包容量为 j
3 |         dp[j] = max(dp[j], dp[j - w[i]] + v[i]); //判断背包容量为 j 的情况下能是实现总
   |   价值最大是多少

```

9.2 完全背包

有 n 件物品和一个容量为 W 的背包，第 i 件物品的体积为 $w[i]$ ，价值为 $v[i]$ ，每件物品有**无限个**，求解将哪些物品装入背包中使总价值最大。

思路：

思路和**01背包**差不多，但是每一件物品有**无限个**，其实就是从每**种**物品中取 $0, 1, 2, \dots$ 件物品加入背包中

```

1 | for (int i = 1; i <= n; i++) {
2 |     for (int j = 0; j <= W; j++) {
3 |         for (int k = 0; k * w[i] <= j; k++) //选取几个物品
4 |             dp[i][j] = max(dp[i][j], dp[i - 1][j - k * w[i]] + k * v[i]);

```

实际上，我们可以发现，取 k 件物品可以从取 $k - 1$ 件转移过来，那么我们就可以将 k 的循环优化掉

```

1 | for (int i = 1; i <= n; i++) {
2 |     for (int j = 0; j <= W; j++) {
3 |         dp[i][j] = dp[i - 1][j];
4 |         if (j >= w[i])
5 |             dp[i][j] = max(dp[i][j], dp[i][j - w[i]] + v[i]);
6 |     }

```

和 01 背包类似地压缩成一维

```

1 | for (int i = 1; i <= n; i++)
2 |   for (int j = w[i]; j <= W; j++)
3 |     dp[j] = max(dp[j], dp[j - w[i]] + v[i]);

```

9.3 多重背包

有 n 种物品和一个容量为 W 的背包，第 i 种物品的体积为 $w[i]$ ，价值为 $v[i]$ ，数量为 $s[i]$ ，求解将哪些物品装入背包中使总价值最大。

思路：

对于每一种物品，都有 $s[i]$ 种取法，我们可以将其转化为**01背包问题**

```

1 | for (int i = 1; i <= n; i++){
2 |   for (int j = W; j >= 0; j--){
3 |     for (int k = 0; k <= s[i]; k++){
4 |       if (j - k * w[i] < 0) break;
5 |       dp[j] = max(dp[j], dp[j - k * w[i]] + k * v[i]);
6 |     }
}

```

上述方法的时间复杂度为 $O(n * m * s)$ 。

```

1 | for (int i = 1; i <= n; i++){
2 |   scanf("%lld%lld%lld", &x, &y, &s); //x 为体积， y 为价值， s 为数量
3 |   t = 1;
4 |   while (s >= t){
5 |     w[++num] = x * t;
6 |     v[num] = y * t;
7 |     s -= t;
8 |     t *= 2;
9 |   }
10 |  w[++num] = x * s;
11 |  v[num] = y * s;
12 | }
13 | for (int i = 1; i <= num; i++)
14 |   for (int j = W; j >= w[i]; j--)
15 |     dp[j] = max(dp[j], dp[j - w[i]] + v[i]);

```

尽管采用了**二进制优化**，时间复杂度还是太高，采用**单调队列优化**，将时间复杂度优化至 $O(n * m)$

```

1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | const int N = 2e5 + 10;
4 | int n, W, w, v, s, f[N], g[N], q[N];
5 | int main(){
6 |   ios::sync_with_stdio(false); cin.tie(0);
7 |   cin >> n >> W;
8 |   for (int i = 0; i < n; i ++ ){
9 |     memcpy (g, f, sizeof f);
10 |    cin >> w >> v >> s;
11 |    for (int j = 0; j < w; j ++ ){
12 |      int head = 0, tail = -1;
13 |      for (int k = j; k <= W; k += w){
14 |        if (head <= tail && k - s * w > q[head]) head ++ ; //保证队列长度 <=
|           s
15 |        while (head <= tail && g[q[tail]] - (q[tail] - j) / w * v <= g[k] -
|           (k - j) / w * v) tail -- ; //保证队列单调递减
16 |        q[ ++ tail] = k;
17 |        f[k] = g[q[head]] + (k - q[head]) / w * v;
}

```

```

18     }
19 }
20 }
21 cout << f[W] << "\n";
22 return 0;
23 }
```

9.4 混合背包

放入背包的物品可能只有 **1** 件（01背包），也可能有**无限件**（完全背包），也可能只有**可数的几件**（多重背包）。

思路：

分类讨论即可，哪一类就用哪种方法去 *dp*。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 int n, W, w, v, s;
4 int main(){
5     cin >> n >> W;
6     vector<int> f(W + 1);
7     for (int i = 0; i < n; i ++ ){
8         cin >> w >> v >> s;
9         if (s == -1){
10             for (int j = W; j >= w; j -- )
11                 f[j] = max(f[j], f[j - w] + v);
12         }
13         else if (s == 0){
14             for (int j = w; j <= W; j ++ )
15                 f[j] = max(f[j], f[j - w] + v);
16         }
17         else {
18             int t = 1, cnt = 0;
19             vector<int> x(s + 1), y(s + 1);
20             while (s >= t){
21                 x[++cnt] = w * t;
22                 y[cnt] = v * t;
23                 s -= t;
24                 t *= 2;
25             }
26             x[++cnt] = w * s;
27             y[cnt] = v * s;
28             for (int i = 1; i <= cnt; i ++ )
29                 for (int j = W; j >= x[i]; j -- )
30                     f[j] = max(f[j], f[j - x[i]] + y[i]);
31         }
32     }
33     cout << f[W] << "\n";
34     return 0;
35 }
```

9.5 二维费用的背包

有 n 件物品和一个容量为 W 的背包，背包能承受的最大重量为 M ，每件物品只能用一次，第 i 件物品的体积是 $w[i]$ ，重量为 $m[i]$ ，价值为 $v[i]$ ，求解将哪些物品放入背包中使总体积不超过背包容量，总重量不超过背包最大容量，且总价值最大。

思路：

背包的限制条件由一个变成两个，那么我们的循环再多一维即可。

```

1 | for (int i = 1; i <= n; i++)
2 |   for (int j = w; j >= w; j--) //容量限制
3 |     for (int k = M; k >= m; k--) //重量限制
4 |       dp[j][k] = max(dp[j][k], dp[j - w][k - m] + v);

```

9.6 分组背包

有 n 组物品，一个容量为 W 的背包，每组物品有若干，同一组的物品最多选一个，第 i 组第 j 件物品的体积为 $w[i][j]$ ，价值为 $v[i][j]$ ，求解将哪些物品装入背包，可使物品总体积不超过背包容量，且使总价值最大。

思路：

考虑每组中的某件物品选不选，可以选的话，去下一组选下一个，否则在这组继续寻找可以选的物品，当这组遍历完后，去下一组寻找。

```

1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | const int N = 110;
4 | int n, W, s[N], w[N][N], v[N][N], dp[N];
5 | int main(){
6 |   cin >> n >> W;
7 |   for (int i = 1; i <= n; i++){
8 |     scanf("%d", &s[i]);
9 |     for (int j = 1; j <= s[i]; j++)
10 |       scanf("%d %d", &w[i][j], &v[i][j]);
11 |   }
12 |   for (int i = 1; i <= n; i++)
13 |     for (int j = W; j >= 0; j--)
14 |       for (int k = 1; k <= s[i]; k++)
15 |         if (j - w[i][k] >= 0)
16 |           dp[j] = max(dp[j], dp[j - w[i][k]] + v[i][k]);
17 |   cout << dp[W] << "\n";
18 |   return 0;
19 | }

```

9.7 有依赖的背包

有 n 个物品和一个容量为 W 的背包，物品之间有依赖关系，且之间的依赖关系组成一颗 **树** 的形状，如果选择一个物品，则必须选择它的 **父节点**，第 i 件物品的体积是 $w[i]$ ，价值为 $v[i]$ ，依赖的父节点的编号为 $p[i]$ ，若 $p[i]$ 等于 -1，则为 **根节点**。求将哪些物品装入背包中，使总体积不超过总容量，且总价值最大。

思路：

定义 $f[i][j]$ 为以第 i 个节点为根，容量为 j 的背包的最大价值。那么结果就是 $f[root][W]$ ，为了知道根节点的最大价值，得通过其子节点来更新。所以采用递归的方式。对于每一个点，先将这个节点装入背包，然后找到剩余容量可以实现的最大价值，最后更新父节点的最大价值即可。

```

1 | #include <bits/stdc++.h>
2 | using namespace std;
3 | const int N = 110;
4 | int n, W, w[N], v[N], p, f[N][N], root;
5 | vector <int> g[N];
6 | void dfs(int u){
7 |   for (int i = w[u]; i <= W; i++)
8 |     f[u][i] = v[u];
9 |   for (auto v : g[u]){

```

```

10     dfs(v);
11     for (int j = W; j >= w[u]; j -- )
12         for (int k = 0; k <= j - w[u]; k ++ )
13             f[u][j] = max(f[u][j], f[u][j - k] + f[v][k]);
14     }
15 }
16 int main(){
17     cin >> n >> W;
18     for (int i = 1; i <= n; i ++ ){
19         cin >> w[i] >> v[i] >> p;
20         if (p == -1) root = i;
21         else g[p].push_back(i);
22     }
23     dfs(root);
24     cout << f[root][W] << "\n";
25     return 0;
26 }
```

9.8 背包问题求方案数

有 n 件物品和一个容量为 W 的背包，每件物品只能用一次，第 i 件物品的重量为 $w[i]$ ，价值为 $v[i]$ ，求解将哪些物品放入背包使总重量不超过背包容量，且总价值最大，输出 **最优先法的方案数**，答案可能很大，输出答案模 $10^9 + 7$ 的结果。

思路：

开一个储存方案数的数组 cnt ， $cnt[i]$ 表示容量为 i 时的 **方案数**，先将 cnt 的每一个值都初始化为 1，因为 **不装任何东西就是一种方案**，如果装入这件物品使总的价值 **更大**，那么装入后的方案数 **等于** 装之前的方案数，如果装入后总价值 **相等**，那么方案数就是 **二者之和**

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define LL long long
4 const int mod = 1e9 + 7, N = 1010;
5 LL n, W, cnt[N], f[N], w, v;
6 int main(){
7     cin >> n >> W;
8     for (int i = 0; i <= W; i ++ )
9         cnt[i] = 1;
10    for (int i = 0; i < n; i ++ ){
11        cin >> w >> v;
12        for (int j = W; j >= w; j -- )
13            if (f[j] < f[j - w] + v){
14                f[j] = f[j - w] + v;
15                cnt[j] = cnt[j - w];
16            }
17            else if (f[j] == f[j - w] + v){
18                cnt[j] = (cnt[j] + cnt[j - w]) % mod;
19            }
20    }
21    cout << cnt[W] << "\n";
22    return 0;
23 }
```

9.9 背包问题求具体方案

```

1 signed main() {
2     int Task = 1;
3     for (cin >> Task; Task; Task--) {
4         int n, m;
```

```

5     cin >> n >> m;
6
7     vector<int> w(n + 1), v(n + 1);
8     vector<vector<int>> dp(n + 2, vector<int>(m + 2));
9     for (int i = 1; i <= n; i++) {
10        cin >> w[i] >> v[i];
11    }
12
13    for (int i = n; i >= 1; i--) {
14        for (int j = 0; j <= m; j++) {
15            dp[i][j] = dp[i + 1][j];
16            if (j >= w[i]) {
17                dp[i][j] = max(dp[i][j], dp[i + 1][j - w[i]] + v[i]);
18            }
19        }
20    }
21
22    vector<int> ans;
23    for (int i = 1; i <= n; i++) {
24        if (m - w[i] >= 0 && dp[i][m] == dp[i + 1][m - w[i]] + v[i]) {
25            ans.push_back(i);
26            // cout << i << " ";
27            m -= w[i];
28        }
29    }
30    cout << ans.size() << "\n";
31    for (auto i : ans) {
32        cout << i << " ";
33    }
34    cout << "\n";
35}
36

```

9.10 数位 DP

```

1 /* pos 表示当前枚举到第几位
2 sum 表示 d 出现的次数
3 limit 为 1 表示枚举的数字有限制
4 zero 为 1 表示有前导 0
5 d 表示要计算出现次数的数 */
6 const int N = 15;
7 LL dp[N][N];
8 int num[N];
9 LL dfs(int pos, LL sum, int limit, int zero, int d) {
10    if (pos == 0) return sum;
11    if (!limit && !zero && dp[pos][sum] != -1) return dp[pos][sum];
12    LL ans = 0;
13    int up = (limit ? num[pos] : 9);
14    for (int i = 0; i <= up; i++) {
15        ans += dfs(pos - 1, sum + ((!zero || i) && (i == d)), limit && (i == num[pos]),
16                   zero && (i == 0), d);
17    }
18    if (!limit && !zero) dp[pos][sum] = ans;
19    return ans;
20}
21 LL solve(LL x, int d) {
22    memset(dp, -1, sizeof dp);
23    int len = 0;
24    while (x) {
25        num[++len] = x % 10;
26        x /= 10;

```

```

27     }
28     return dfs(len, 0, 1, 1, d);
29 }
```

9.11 状压 DP

题意：在 $n * n$ 的棋盘里面放 k 个国王，使他们互不攻击，共有多少种摆放方案。国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共8个格子。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define LL long long
4 const int N = 15, M = 150, K = 1500;
5 LL n, k;
6 LL cnt[K]; //每个状态的二进制中 1 的数量
7 LL tot; //合法状态的数量
8 LL st[K]; //合法的状态
9 LL dp[N][M][K]; //第 i 行，放置了 j 个国王，状态为 k 的方案数
10 int main(){
11     ios::sync_with_stdio(false);cin.tie(0);
12     cin >> n >> k;
13     for (int s = 0; s < (1 << n); s ++ ){ //找出合法状态
14         LL sum = 0, t = s;
15         while(t){ //计算 1 的数量
16             sum += (t & 1);
17             t >>= 1;
18         }
19         cnt[s] = sum;
20         if ( ((s << 1) | (s >> 1) ) & s == 0 ){ //判断合法性
21             st[ ++ tot] = s;
22         }
23     }
24     dp[0][0][0] = 1;
25     for (int i = 1; i <= n + 1; i ++ ){
26         for (int j1 = 1; j1 <= tot; j1 ++ ){ //当前的状态
27             LL s1 = st[j1];
28             for (int j2 = 1; j2 <= tot; j2 ++ ){ //上一行的状态
29                 LL s2 = st[j2];
30                 if ( ( (s2 | (s2 << 1) | (s2 >> 1)) & s1 ) == 0 ){
31                     for (int j = 0; j <= k; j ++ ){
32                         if (j - cnt[s1] >= 0)
33                             dp[i][j][s1] += dp[i - 1][j - cnt[s1]][s2];
34                     }
35                 }
36             }
37         }
38     }
39     cout << dp[n + 1][k][0] << "\n";
40     return 0;
41 }
```

9.12 常用例题

题意：在一篇文章（包含大小写英文字母、数字、和空白字符（制表/空格/回车））中寻找 helloworld（任意一个字母的大小写都行）的子序列出现了多少次，输出结果对 $10^9 + 7$ 的余数。

字符串 DP，构建一个二维 DP 数组， $dp[i][j]$ 的 i 表示文章中的第几个字符， j 表示寻找的字符串的第几个字符，当字符串中的字符和文章中的字符相同时，即找到符合条件的字符， $dp[i][j] = dp[i - 1][j] + dp[i - 1][j - 1]$ ，因为字符串中的每个字符不会对后面的结果产生影响，所以 DP 方程可以优化成一维的，由于字符串中有重复的字符，所以比较时应该从后往前。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 #define LL long long
4 const int mod = 1e9 + 7;
5 char c, s[20] = "!helloworld";
6 LL dp[20];
7 int main(){
8     dp[0] = 1;
9     while ((c = getchar()) != EOF)
10         for (int i = 10; i >= 1; i--)
11             if (c == s[i] || c == s[i] - 32)
12                 dp[i] = (dp[i] + dp[i - 1]) % mod;
13     cout << dp[10] << "\n";
14     return 0;
15 }

```

题意：（最长括号匹配）给一个只包含‘(’，‘)’，‘[’，‘]’的非空字符串，“(”和“)”是匹配的，寻找字符串中最长的括号匹配的子串，若有两串长度相同，输出靠前的一串。

设给定的字符串为 s ，可以定义数组 $dp[i]$, $dp[i]$ 表示以 $s[i]$ 结尾的字符串里最长的括号匹配的字符。显然，从 $i - dp[i] + 1$ 到 i 的字符串是括号匹配的，当找到一个字符是‘(’或‘]’时，再去判断第 $i - 1 - dp[i - 1]$ 的字符和第 i 位的字符是否匹配，如果是，那么 $dp[i] = dp[i - 1] + 2 + dp[i - 2 - dp[i - 1]]$ 。

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int maxn = 1e6 + 10;
4 string s;
5 int len, dp[maxn], ans, id;
6 int main(){
7     cin >> s;
8     len = s.length();
9     for (int i = 1; i < len; i++){
10         if ((s[i] == ')') && s[i - 1 - dp[i - 1]] == '(') || (s[i] == '[') && s[i - 1 - dp[i - 1]] == ']'){
11             dp[i] = dp[i - 1] + 2 + dp[i - 2 - dp[i - 1]];
12             if (dp[i] > ans) {
13                 ans = dp[i]; //记录长度
14                 id = i; //记录位置
15             }
16         }
17     }
18     for (int i = id - ans + 1; i <= id; i++)
19         cout << s[i];
20     cout << "\n";
21     return 0;
22 }

```

题意：去掉区间内包含“4”和“62”的数字，输出剩余的数字个数

```

1 int T,n,m,len,a[20];//a数组用于判断每一位能取到的最大值
2 ll l,r,dp[20][15];
3 ll dfs(int pos,int pre,int limit){//记搜
4     //pos搜到的位置，pre前一位数
5     //limit判断是否有最高位限制
6     if(pos>len) return 1;//剪枝
7     if(dp[pos][pre]!=-1 && !limit) return dp[pos][pre];//记录当前值
8     ll ret=0;//暂时记录当前方案数

```

```

9  int res=limit?a[len-pos+1]:9;//res当前位能取到的最大值
10 for(int i=0;i<=res;i++)
11     if(!(i==4 || (pre==6 && i==2)))
12         ret+=dfs(pos+1,i,i==res&&limit);
13     if(!limit) dp[pos][pre]=ret;//当前状态方案数记录
14     return ret;
15 }
16 ll part(ll x){//把数按位拆分
17     len=0;
18     while(x) a[++len]=x%10,x/=10;
19     memset(dp,-1,sizeof dp);//初始化-1(因为有可能某些情况下的方案数是0)
20     return dfs(1,0,1);//进入记搜
21 }
22 int main(){
23     cin>>n;
24     while(n--){
25         cin>>l>>r;
26         if(l==0 && r==0)break;
27         if(l) printf("%lld\n",part(r)-part(l-1));//[l,r](l!=0)
28         else printf("%lld\n",part(r)-part(l));//从0开始要特判
29     }
30 }
```

/END/

10 串

10.1 子串与子序列

中文名称	常见英文名称	解释
子串	substring	连续的选择一段字符（可以全选、可以不选）组成的新字符串
子序列	subsequence	从左到右取出若干个字符（可以不取、可以全取、可以不连续）组成的新字符串

10.2 字符串模式匹配算法 KMP

应用：

1. 在字符串中查找子串；
2. 最小周期：字符串长度-整个字符串的 border；
3. 最小循环节：区别于周期，当字符串长度 $n \bmod (n - \text{nxt}[n]) = 0$ 时，等于最小周期，否则为 n 。

以最坏 $\mathcal{O}(N + M)$ 的时间计算 t 在 s 中出现的全部位置。

```

1 auto kmp = [&](string s, string t) {
2     int n = s.size(), m = t.size();
3     vector<int> kmp(m + 1), ans;
4     s = "@" + s;
5     t = "@" + t;
6     for (int i = 2, j = 0; i <= m; i++) {
7         while (j && t[i] != t[j + 1]) {
8             j = kmp[j];
9         }
10        j += t[i] == t[j + 1];
11        kmp[i] = j;
12    }
13    for (int i = 1, j = 0; i <= n; i++) {
14        while (j && s[i] != t[j + 1]) {
15            j = kmp[j];
16        }
17        if (s[i] == t[j + 1] && ++j == m) {
18            ans.push_back(i - m + 1); // t 在 s 中出现的位置
19        }
20    }
21    return ans;
22};
```

10.3 Z函数（扩展 KMP）

获取字符串 s 和 $s[i, n - 1]$ （即以 $s[i]$ 开头的后缀）的最长公共前缀（LCP）的长度，总复杂度 $\mathcal{O}(N)$ 。

```

1 vector<int> zFunction(string s) {
2     int n = s.size();
3     vector<int> z(n);
4     z[0] = n;
5     for (int i = 1, j = 1; i < n; i++) {
6         z[i] = max(0, min(j + z[j] - i, z[i - j]));
7         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) {
8             z[i]++;
9         }
10    }
11}
```

```

9      }
10     if (i + z[i] > j + z[j]) {
11         j = i;
12     }
13 }
14 return z;
15 }
```

10.4 最长公共子序列 LCS

求解两个串的最长公共子序列的长度。

10.4.1 小数据解

针对 10^3 以内的数据。

```

1 const int N = 1e3 + 10;
2 char a[N], b[N];
3 int n, m, f[N][N];
4 void solve(){
5     cin >> n >> m >> a + 1 >> b + 1;
6     for (int i = 1; i <= n; i++){
7         for (int j = 1; j <= m; j++){
8             f[i][j] = max(f[i - 1][j], f[i][j - 1]);
9             if (a[i] == b[j]) f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
10        }
11    cout << f[n][m] << "\n";
12 }
13 int main(){
14     solve();
15     return 0;
16 }
```

10.4.2 大数据解

针对 10^5 以内的数据。

```

1 const int INF = 0x7fffffff;
2 int n, a[maxn], b[maxn], f[maxn], p[maxn];
3 int main(){
4     cin >> n;
5     for (int i = 1; i <= n; i++){
6         scanf("%d", &a[i]);
7         p[a[i]] = i; //将第二个序列中的元素映射到第一个中
8     }
9     for (int i = 1; i <= n; i++){
10        scanf("%d", &b[i]);
11        f[i] = INF;
12    }
13    int len = 0;
14    f[0] = 0;
15    for (int i = 1; i <= n; i++){
16        if (p[b[i]] > f[len]) f[++len] = p[b[i]];
17        else {
18            int l = 0, r = len;
19            while (l < r){
20                int mid = (l + r) >> 1;
21                if (f[mid] > p[b[i]]) r = mid;
22                else l = mid + 1;
23            }
24            f[1] = min(f[1], p[b[i]]);
25        }
26    }
27 }
```

```

25     }
26 }
27 cout << len << "\n";
28 return 0;
29 }
```

10.5 字符串哈希

10.5.1 双哈希封装

随机质数列表：1111111121、1211111123、1311111119。

```

1 const int N = 1 << 21;
2 static const int mod1 = 1E9 + 7, base1 = 127;
3 static const int mod2 = 1E9 + 9, base2 = 131;
4 using U = Zmod<mod1>;
5 using V = Zmod<mod2>;
6 vector<U> val1;
7 vector<V> val2;
8 void init(int n = N) {
9     val1.resize(n + 1), val2.resize(n + 2);
10    val1[0] = 1, val2[0] = 1;
11    for (int i = 1; i <= n; i++) {
12        val1[i] = val1[i - 1] * base1;
13        val2[i] = val2[i - 1] * base2;
14    }
15 }
16 struct String {
17     vector<U> hash1;
18     vector<V> hash2;
19     string s;
20
21     String(string s_) : s(s_), hash1{1}, hash2{1} {
22         for (auto it : s) {
23             hash1.push_back(hash1.back() * base1 + it);
24             hash2.push_back(hash2.back() * base2 + it);
25         }
26     }
27     pair<U, V> get() { // 输出整串的哈希值
28         return {hash1.back(), hash2.back()};
29     }
30     pair<U, V> substring(int l, int r) { // 输出子串的哈希值
31         if (l > r) swap(l, r);
32         U ans1 = hash1[r + 1] - hash1[l] * val1[r - l + 1];
33         V ans2 = hash2[r + 1] - hash2[l] * val2[r - l + 1];
34         return {ans1, ans2};
35     }
36     pair<U, V> modify(int idx, char x) { // 修改 idx 位为 x
37         int n = s.size() - 1;
38         U ans1 = hash1.back() + val1[n - idx] * (x - s[idx]);
39         V ans2 = hash2.back() + val2[n - idx] * (x - s[idx]);
40         return {ans1, ans2};
41     }
42 };
```

10.5.2 前后缀去重

sample please ease | 去重后得到 samplease 。

```

1 string compress(vector<string> in) { // 前后缀压缩
2     vector<U> hash1{1};
3     vector<V> hash2{1};
4     string ans = "#";
5     for (auto s : in) {
6         s = "#" + s;
7         int st = 0;
8         U chk1 = 0;
9         V chk2 = 0;
10        for (int j = 1; j < s.size() && j < ans.size(); j++) {
11            chk1 = chk1 * base1 + s[j];
12            chk2 = chk2 * base2 + s[j];
13            if ((hash1.back() == hash1[ans.size() - 1 - j] * val1[j] + chk1) &&
14                (hash2.back() == hash2[ans.size() - 1 - j] * val2[j] + chk2)) {
15                st = j;
16            }
17        }
18        for (int j = st + 1; j < s.size(); j++) {
19            ans += s[j];
20            hash1.push_back(hash1.back() * base1 + s[j]);
21            hash2.push_back(hash2.back() * base2 + s[j]);
22        }
23    }
24    return ans.substr(1);
25 }
```

10.6 马拉车

$\mathcal{O}(N)$ 时间求出字符串的最长回文子串。

```

1 string s;
2 cin >> s;
3 int n = s.length();
4 string t = "-#";
5 for (int i = 0; i < n; i++) {
6     t += s[i];
7     t += '#';
8 }
9 int m = t.length();
10 t += '+';
11 int mid = 0, r = 0;
12 vector<int> p(m);
13 for (int i = 1; i < m; i++) {
14     p[i] = i < r ? min(p[2 * mid - i], r - i) : 1;
15     while (t[i - p[i]] == t[i + p[i]]) p[i]++;
16     if (i + p[i] > r) {
17         r = i + p[i];
18         mid = i;
19     }
20 }
```

10.7 字典树 trie

10.7.1 基础封装

```

1 struct Trie {
2     int ch[N][63], cnt[N], idx = 0;
3     map<char, int> mp;
4     void init() {
5         LL id = 0;
6         for (char c = 'a'; c <= 'z'; c++) mp[c] = ++id;
7         for (char c = 'A'; c <= 'Z'; c++) mp[c] = ++id;
8         for (char c = '0'; c <= '9'; c++) mp[c] = ++id;
9     }
10    void insert(string s) {
11        int u = 0;
12        for (int i = 0; i < s.size(); i++) {
13            int v = mp[s[i]];
14            if (!ch[u][v]) ch[u][v] = ++idx;
15            u = ch[u][v];
16            cnt[u]++;
17        }
18    }
19    LL query(string s) {
20        int u = 0;
21        for (int i = 0; i < s.size(); i++) {
22            int v = mp[s[i]];
23            if (!ch[u][v]) return 0;
24            u = ch[u][v];
25        }
26        return cnt[u];
27    }
28    void Clear() {
29        for (int i = 0; i <= idx; i++) {
30            cnt[i] = 0;
31            for (int j = 0; j <= 62; j++) {
32                ch[i][j] = 0;
33            }
34        }
35        idx = 0;
36    }
37 } trie;

```

10.7.2 01 字典树

```

1 struct Trie {
2     int n, idx;
3     vector<vector<int>> ch;
4     Trie(int n) {
5         this->n = n;
6         idx = 0;
7         ch.resize(30 * (n + 1), vector<int>(2));
8     }
9     void insert(int x) {
10        int u = 0;
11        for (int i = 30; ~i; i--) {
12            int &v = ch[u][x >> i & 1];
13            if (!v) v = ++idx;
14            u = v;
15        }
16    }
17    int query(int x) {
18        int u = 0, res = 0;

```

```

19     for (int i = 30; ~i; i--) {
20         int v = x >> i & 1;
21         if (ch[u][!v]) {
22             res += (1 << i);
23             u = ch[u][!v];
24         } else {
25             u = ch[u][v];
26         }
27     }
28     return res;
29 }
30 };

```

10.8 后缀数组 SA

以 $\mathcal{O}(N)$ 的复杂度求解。

```

1 struct SuffixArray {
2     int n;
3     vector<int> sa, rk, lc;
4     SuffixArray(const string &s) {
5         n = s.length();
6         sa.resize(n);
7         lc.resize(n - 1);
8         rk.resize(n);
9         iota(sa.begin(), sa.end(), 0);
10        sort(sa.begin(), sa.end(), [&](int a, int b) { return s[a] < s[b]; });
11        rk[sa[0]] = 0;
12        for (int i = 1; i < n; ++i) {
13            rk[sa[i]] = rk[sa[i - 1]] + (s[sa[i]] != s[sa[i - 1]]);
14        }
15        int k = 1;
16        vector<int> tmp, cnt(n);
17        tmp.reserve(n);
18        while (rk[sa[n - 1]] < n - 1) {
19            tmp.clear();
20            for (int i = 0; i < k; ++i) {
21                tmp.push_back(n - k + i);
22            }
23            for (auto i : sa) {
24                if (i >= k) {
25                    tmp.push_back(i - k);
26                }
27            }
28            fill(cnt.begin(), cnt.end(), 0);
29            for (int i = 0; i < n; ++i) {
30                ++cnt[rk[i]];
31            }
32            for (int i = 1; i < n; ++i) {
33                cnt[i] += cnt[i - 1];
34            }
35            for (int i = n - 1; i >= 0; --i) {
36                sa[--cnt[rk[tmp[i]]]] = tmp[i];
37            }
38            swap(rk, tmp);
39            rk[sa[0]] = 0;
40            for (int i = 1; i < n; ++i) {
41                rk[sa[i]] = rk[sa[i - 1]] + (tmp[sa[i - 1]] < tmp[sa[i]] || sa[i - 1] + k == n ||
42                                            tmp[sa[i - 1] + k] < tmp[sa[i] + k]);
43            }
44            k *= 2;

```

```

45 }
46     for (int i = 0, j = 0; i < n; ++i) {
47         if (rk[i] == 0) {
48             j = 0;
49             continue;
50         }
51         for (j -= j > 0;
52              i + j < n && sa[rk[i] - 1] + j < n && s[i + j] == s[sa[rk[i] - 1] +
53              j];) {
54             ++j;
55         }
56         lc[rk[i] - 1] = j;
57     }
58 }

```

10.9 AC 自动机

定义 $|s_i|$ 是模板串的长度， $|S|$ 是文本串的长度， $|\Sigma|$ 是字符集的大小（常数，一般为 26），时间复杂度为 $\mathcal{O}(\sum |s_i| + |S|)$ 。

```

1 // Trie+Kmp，多模式串匹配
2 struct ACAutomaton {
3     static constexpr int N = 1e6 + 10;
4     int ch[N][26], fail[N], cntNodes;
5     int cnt[N];
6     ACAutomaton() {
7         cntNodes = 1;
8     }
9     void insert(string s) {
10         int u = 1;
11         for (auto c : s) {
12             int &v = ch[u][c - 'a'];
13             if (!v) v = ++cntNodes;
14             u = v;
15         }
16         cnt[u]++;
17     }
18     void build() {
19         fill(ch[0], ch[0] + 26, 1);
20         queue<int> q;
21         q.push(1);
22         while (!q.empty()) {
23             int u = q.front();
24             q.pop();
25             for (int i = 0; i < 26; i++) {
26                 int &v = ch[u][i];
27                 if (!v)
28                     v = ch[fail[u]][i];
29                 else {
30                     fail[v] = ch[fail[u]][i];
31                     q.push(v);
32                 }
33             }
34         }
35     }
36     LL query(string t) {
37         LL ans = 0;
38         int u = 1;
39         for (auto c : t) {
40             u = ch[u][c - 'a'];
41             for (int v = u; v && ~cnt[v]; v = fail[v]) {

```

```

42         ans += cnt[v];
43         cnt[v] = -1;
44     }
45 }
46 return ans;
47 }
48 };

```

10.10 回文自动机 PAM (回文树)

应用：

1. 本质不同的回文串个数： $idx - 2$ ；
2. 回文子串出现次数。

对于一个字符串 s ，它的本质不同回文子串个数最多只有 $|s|$ 个，那么，构造 s 的回文树的时间复杂度是 $\mathcal{O}(|s|)$ 。

```

1 struct PalindromeAutomaton {
2     constexpr static int N = 5e5 + 10;
3     int tr[N][26], fail[N], len[N];
4     int cntNodes, last;
5     int cnt[N];
6     string s;
7     PalindromeAutomaton(string s) {
8         memset(tr, 0, sizeof tr);
9         memset(fail, 0, sizeof fail);
10        len[0] = 0, fail[0] = 1;
11        len[1] = -1, fail[1] = 0;
12        cntNodes = 1;
13        last = 0;
14        this->s = s;
15    }
16    void insert(char c, int i) {
17        int u = get_fail(last, i);
18        if (!tr[u][c - 'a']) {
19            int v = ++cntNodes;
20            fail[v] = tr[get_fail(fail[u], i)][c - 'a'];
21            tr[u][c - 'a'] = v;
22            len[v] = len[u] + 2;
23            cnt[v] = cnt[fail[v]] + 1;
24        }
25        last = tr[u][c - 'a'];
26    }
27    int get_fail(int u, int i) {
28        while (i - len[u] - 1 <= -1 || s[i - len[u] - 1] != s[i]) {
29            u = fail[u];
30        }
31        return u;
32    }
33 };

```

10.11 后缀自动机 SAM

定义 $|\Sigma|$ 是字符集的大小，复杂度为 $\mathcal{O}(N \log |\Sigma|)$ 。

```

1 // 有向无环图
2 struct SuffixAutomaton {
3     static constexpr int N = 1e6;
4     struct node {

```

```

5     int len, link, nxt[26];
6     int siz;
7 } t[N << 1];
8 int cntNodes;
9 SuffixAutomaton() {
10     cntNodes = 1;
11     fill(t[0].nxt, t[0].nxt + 26, 1);
12     t[0].len = -1;
13 }
14 int extend(int p, int c) {
15     if (t[p].nxt[c]) {
16         int q = t[p].nxt[c];
17         if (t[q].len == t[p].len + 1) {
18             return q;
19         }
20         int r = ++cntNodes;
21         t[r].siz = 0;
22         t[r].len = t[p].len + 1;
23         t[r].link = t[q].link;
24         copy(t[q].nxt, t[q].nxt + 26, t[r].nxt);
25         t[q].link = r;
26         while (t[p].nxt[c] == q) {
27             t[p].nxt[c] = r;
28             p = t[p].link;
29         }
30         return r;
31     }
32     int cur = ++cntNodes;
33     t[cur].len = t[p].len + 1;
34     t[cur].siz = 1;
35     while (!t[p].nxt[c]) {
36         t[p].nxt[c] = cur;
37         p = t[p].link;
38     }
39     t[cur].link = extend(p, c);
40     return cur;
41 }
42 };

```

10.12 子序列自动机

对于给定的长度为 n 的主串 s ，以 $\mathcal{O}(n)$ 的时间复杂度预处理、 $\mathcal{O}(m + \log \text{size}:s)$ 的复杂度判定长度为 m 的询问串是否是主串的子序列。

10.12.1 自动离散化、自动类型匹配封装

```

1 template<class T> struct SequenceAutomaton {
2     vector<T> alls;
3     vector<vector<int>> ver;
4
5     SequenceAutomaton(auto in) {
6         for (auto &i : in) {
7             alls.push_back(i);
8         }
9         sort(alls.begin(), alls.end());
10        alls.erase(unique(alls.begin(), alls.end()), alls.end());
11
12        ver.resize(alls.size() + 1);
13        for (int i = 0; i < in.size(); i++) {
14            ver[get(in[i])].push_back(i + 1);
15        }
16    }

```

```

17     bool count(T x) {
18         return binary_search(alls.begin(), alls.end(), x);
19     }
20     int get(T x) {
21         return lower_bound(alls.begin(), alls.end(), x) - alls.begin();
22     }
23     bool contains(auto in) {
24         int at = 0;
25         for (auto &i : in) {
26             if (!count(i)) {
27                 return false;
28             }
29
30             auto j = get(i);
31             auto it = lower_bound(ver[j].begin(), ver[j].end(), at + 1);
32             if (it == ver[j].end()) {
33                 return false;
34             }
35             at = *it;
36         }
37         return true;
38     }
39 };

```

10.12.2 朴素封装

原时间复杂度中的 $\text{size}:s$ 需要手动设置。类型需要手动设置。

```

1 struct SequenceAutomaton {
2     vector<vector<int>> ver;
3
4     SequenceAutomaton(vector<int> &in, int size) : ver(size + 1) {
5         for (int i = 0; i < in.size(); i++) {
6             ver[in[i]].push_back(i + 1);
7         }
8     }
9     bool contains(vector<int> &in) {
10         int at = 0;
11         for (auto &i : in) {
12             auto it = lower_bound(ver[i].begin(), ver[i].end(), at + 1);
13             if (it == ver[i].end()) {
14                 return false;
15             }
16             at = *it;
17         }
18         return true;
19     }
20 };

```

/END/

11 博弈论

11.1 巴什博弈

有 N 个石子，两名玩家轮流行动，按以下规则取石子：

规定：每人每次可以取走 $X(1 \leq X \leq M)$ 个石子，拿到最后一颗石子的一方获胜。

双方均采用最优策略，询问谁会获胜。

两名玩家轮流报数。

规定：第一个报数的人可以报 $X(1 \leq X \leq M)$ ，后报数的人需要比前者所报数大 $Y(1 \leq Y \leq M)$ ，率先报到 N 的人获胜。

双方均采用最优策略，询问谁会获胜。

- $N = K \cdot (M + 1)$ (其中 $K \in \mathbb{N}^+$)，后手必胜（后手可以控制每一回合结束时双方恰好取走 $M + 1$ 个，重复 K 轮后即胜利）；
- $N = K \cdot (M + 1) + R$ (其中 $K \in \mathbb{N}^+, 0 < R < M + 1$)，先手必胜（先手先取走 R 个，之后控制每一回合结束时双方恰好取走 $M + 1$ 个，重复 K 轮后即胜利）。

11.2 扩展巴什博弈

有 N 颗石子，两名玩家轮流行动，按以下规则取石子：。

规定：每人每次可以取走 $X(a \leq X \leq b)$ 个石子，如果最后剩余物品的数量小于 a 个，则不能再取，拿到最后一颗石子的一方获胜。

双方均采用最优策略，询问谁会获胜。

- $N = K \cdot (a + b)$ 时，后手必胜；
- $N = K \cdot (a + b) + R_1$ (其中 $K \in \mathbb{N}^+, 0 < R_1 < a$) 时，后手必胜（这些数量不够再取一次，先手无法逆转局面）；
- $N = K \cdot (a + b) + R_2$ (其中 $K \in \mathbb{N}^+, a \leq R_2 \leq b$) 时，先手必胜；
- $N = K \cdot (a + b) + R_3$ (其中 $K \in \mathbb{N}^+, b < R_3 < a + b$) 时，先手必胜（这些数量不够再取一次，后手无法逆转局面）；

11.3 Nim 博弈

有 N 堆石子，给出每一堆的石子数量，两名玩家轮流行动，按以下规则取石子：

规定：每人每次任选一堆，取走正整数颗石子，拿到最后一颗石子的一方获胜（注：几个特点是不能跨堆、不能不拿）。

双方均采用最优策略，询问谁会获胜。

记初始时各堆石子的数量 (A_1, A_2, \dots, A_n) ，定义尼姆和 $Sum_N = A_1 \oplus A_2 \oplus \dots \oplus A_n$ 。

当 $Sum_N = 0$ 时先手必败，反之先手必胜。

11.4 Nim 游戏具体取法

先计算出尼姆和，再对每一堆石子计算 $A_i \oplus Sum_N$ ，记为 X_i 。

若得到的值 $X_i < A_i$ ， X_i 即为一个可行解，即剩下 X_i 颗石头，取走 $A_i - X_i$ 颗石头（这里取小于号是因为至少要取走 1 颗石子）。

11.5 Moore's Nim 游戏 (Nim-K 游戏)

有 N 堆石子，给出每一堆的石子数量，两名玩家轮流行动，按以下规则取石子：

规定：每人每次任选不超过 K 堆，对每堆都取走不同的正整数颗石子，拿到最后一颗石子的一方获胜。

双方均采用最优策略，询问谁会获胜。

把每一堆石子的石子数用二进制表示，定义 One_i 为二进制第 i 位上 1 的个数。

以下局面先手必胜：

对于每一位， $One_1, One_2, \dots, One_N$ 均不为 $K + 1$ 的倍数。

11.6 Anti-Nim 游戏 (反 Nim 游戏)

有 N 堆石子，给出每一堆的石子数量，两名玩家轮流行动，按以下规则取石子：

规定：每人每次任选一堆，取走正整数颗石子，拿到最后一颗石子的一方出局。

双方均采用最优策略，询问谁会获胜。

- 所有堆的石头数量均不超过 1，且 $Sum_N = 0$ （也可看作“且有偶数堆”）；
- 至少有一堆的石头数量大于 1，且 $Sum_N \neq 0$ 。

11.7 阶梯 - Nim 博弈

有 N 级台阶，每一级台阶上均有一定数量的石子，给出每一级石子的数量，两名玩家轮流行动，按以下规则操作石子：

规定：每人每次任选一级台阶，拿走正整数颗石子放到下一级台阶中，已经拿到地面上的石子不能再拿，拿到最后一颗石子的一方获胜。

双方均采用最优策略，询问谁会获胜。

对奇数台阶做传统 Nim 博弈，当 $Sum_N = 0^{}$ 时先手必败，反之先手必胜。****

11.8 SG 游戏 (有向图游戏)

我们使用以下几条规则来定义暴力求解的过程：

- 使用数字来表示输赢情况，0 代表局面必败，非 0 代表**存在必胜可能**，我们称这个数字为这个局面的SG值；
- 找到最终态，根据题意人为定义最终态的输赢情况；
- 对于非最终态的某个节点，其SG值为所有子节点的SG值取 **mex**；
- 单个游戏的输赢态即对应根节点的SG值是否为 0，为 0 代表先手必败，非 0 代表先手必胜；
- 多个游戏的总SG值为单个游戏SG值的异或和。

使用哈希表，以 $\mathcal{O}(N + M)$ 的复杂度计算。

```

1 int n, m, a[N], num[N];
2 int sg(int x) {
3     if (num[x] != -1) return num[x];
4
5     unordered_set<int> S;
6     for (int i = 1; i <= m; ++ i)
7         if(x >= a[i])
8             S.insert(sg(x - a[i]));
9
10    for (int i = 0; ; ++ i)
11        if (S.count(i) == 0)
12            return num[x] = i;
13    }
14 void Solve() {
15     cin >> m;
16     for (int i = 1; i <= m; ++ i) cin >> a[i];
17     cin >> n;
18
19     int ans = 0; memset(num, -1, sizeof num);
20     for (int i = 1; i <= n; ++ i) {
21         int x; cin >> x;
22         ans ^= sg(x);
23     }
24
25     if (ans == 0) no;
26     else yes;
27 }

```

11.9 Anti-SG 游戏 (反 SG 游戏)

SG 游戏中最先不能行动的一方获胜。

以下局面先手必胜：

- 单局游戏的**SG**值均不超过 1，且总**SG**值为 0；
- 至少有一局单局游戏的**SG**值大于 1，且总**SG**值不为 0。

在本质上，这与 Anti-Nim 游戏的结论一致。

11.10 Lasker's-Nim 游戏 (Multi-SG 游戏)

有 N 堆石子，给出每一堆的石子数量，两名玩家轮流行动，每人每次任选以下规定的一种操作石子：

- 任选一堆，取走正整数颗石子；
- 任选数量大于 2 的一堆，分成两堆非空石子。

拿到最后一颗石子的一方获胜。双方均采用最优策略，询问谁会获胜。

本题使用**SG**函数求解，**SG**值定义为：

$$SG(x) = \begin{cases} x-1 & , x \bmod 4 = 0 \\ x & , x \bmod 4 = 1 \\ x & , x \bmod 4 = 2 \\ x+1 & , x \bmod 4 = 3 \end{cases}$$

11.11 Every-SG 游戏

给出一个有向无环图，其中 K 个顶点上放置了石子，两名玩家轮流行动，按以下规则操作石子：

- 移动图上所有还能够移动的石子；
- 无法移动石子的一方出局。双方均采用最优策略，询问谁会获胜。

定义 $step$ 为某一局游戏至多需要经过的回合数。

以下局面先手必胜： $step$ 为奇数。

11.12 威佐夫博弈

有两堆石子，给出每一堆的石子数量，两名玩家轮流行动，每人每次任选以下规定的一种操作石子：

- 任选一堆，取走正整数颗石子；
- 从两队中同时取走正整数颗石子。

拿到最后一颗石子的一方获胜。双方均采用最优策略，询问谁会获胜。

以下局面先手必败：

$(1, 2), (3, 5), (4, 7), (6, 10), \dots$ 具体而言，每一对的第一个数为此前没出现过的最小整数，第二个数为第一个数加上 $1, 2, 3, 4, \dots$ 。

更一般地，对于第 k 对数，第一个数为 $First_k = \left\lfloor \frac{k*(1+\sqrt{5})}{2} \right\rfloor$ ，第二个数为 $Second_k = First_k + k$ 。

其中，在两堆石子的数量均大于 10^9 次时，由于需要使用高精度计算，我们需要人为定义 $\frac{1+\sqrt{5}}{2}$ 的取值为 $lorry = 1.618033988749894848204586834$ 。

```

1 const double lorry = (sqrt(5.0) + 1.0) / 2.0;
2 //const double lorry = 1.618033988749894848204586834;
3 void Solve() {
4     int n, m; cin >> n >> m;
5     if (n < m) swap(n, m);
6     double x = n - m;
7     if ((int)(lorry * x) == m) cout << "lose\n";
8     else cout << "win\n";
9 }
```

11.13 斐波那契博弈

有一堆石子，数量为 N ，两名玩家轮流行动，按以下规则取石子：

先手第1次可以取任意多颗，但不能全部取完，此后每人取的石子数不能超过上个人的两倍，拿到最后一颗石子的一方获胜。

双方均采用最优策略，询问谁会获胜。

当且仅当 N 为斐波那契数时先手必败。

```

1 int fib[100] = {1, 2};
2 map<int, bool> mp;
3 void Force() {
4     for (int i = 2; i <= 86; ++ i) fib[i] = fib[i - 1] + fib[i - 2];
5     for (int i = 0; i <= 86; ++ i) mp[fib[i]] = 1;
6 }
7 void Solve() {
8     int n; cin >> n;
9     if (mp[n] == 1) cout << "lose\n";
10    else cout << "win\n";
11 }

```

11.14 树上删边游戏

给出一棵 N 个节点的有根树，两名玩家轮流行动，按以下规则操作：

选择任意一棵子树并删除（即删去任意一条边，不与根相连的部分会同步被删去）；

删掉最后一棵子树的一方获胜（换句话说，删去根节点的一方失败）。双方均采用最优策略，询问谁会获胜。

结论：当根节点SG值非 1 时先手必胜。

相较于传统SG值的定义，本题的SG函数值定义为：

- 叶子节点的SG值为 0 。
- 非叶子节点的SG值为其所有孩子节点SG值 +1 的异或和。

```

1 auto dfs = [&](auto self, int x, int fa) -> int {
2     int x = 0;
3     for (auto y : ver[x]) {
4         if (y == fa) continue;
5         x ^= self(self, y, x);
6     }
7     return x + 1;
8 };
9 cout << (dfs(dfs, 1, 0) == 1 ? "Bob\n" : "Alice\n");

```

11.15 无向图删边游戏 (Fusion Principle 定理)

给出一张 N 个节点的无向联通图，有一个点作为图的根，两名玩家轮流行动，按以下规则操作：

选择任意一条边删除，不与根相连的部分会同步被删去；

删掉最后一条边的一方获胜。双方均采用最优策略，询问谁会获胜。

- 对于奇环，我们将其缩成一个新点+一条新边；
- 对于偶环，我们将其缩成一个新点；
- 所有连接到原来环上的边全部与新点相连。

此时，本模型转化为“树上删边游戏”。

/END/

12 STL

12.1 库函数

12.1.1 pb_ds 库

其中 `gp_hash_table` 使用的最多，其等价于 `unordered_map`，内部是无序的。

```
1 #include <bits/extc++.h>
2 #include <ext/pb_ds/assoc_container.hpp>
3 template<class S, class T> using omap = __gnu_pbds::gp_hash_table<S, T, myhash>;
```

12.1.2 查找后继 `lower_bound`、`upper_bound`

`lower` 表示 \geq ，`upper` 表示 $>$ 。使用前记得先进行排序。

```
1 //返回a数组[start,end)区间中第一个>=x的地址【地址！！！】
2 cout << lower_bound(a + start, a + end, x);
3
4 cout << lower_bound(a, a + n, x) - a; //在a数组中查找第一个>=x的元素下标
5 upper_bound(a, a + n, k) - lower_bound(a, a + n, k) //查找k在a中出现了几次
```

12.1.3 数组打乱 `shuffle`

```
1 mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
2 shuffle(ver.begin(), ver.end(), rnd);
```

12.1.4 二分搜索 `binary_search`

用于查找某一元素是否在容器中，相当于 `find` 函数。在使用前需要先进行排序。

```
1 //在a数组[start,end)区间中查找x是否存在，返回bool型
2 cout << binary_search(a + start, a + end, x);
```

12.1.5 批量递增赋值函数 `iota`

对容器递增初始化。

```
1 //将a数组[start,end)区间复制成“x , x+1 , x+2 , ...”
2 iota(a + start, a + end, x);
```

12.1.6 数组去重函数 `unique`

在使用前需要先进行排序。

其作用是，对于区间 `[开始位置, 结束位置]`，不停的把后面不重复的元素移到前面来，也可以说是用不重复的元素占领重复元素的位置。并且返回去重后容器中不重复序列的最后一个元素的下一个元素。所以在进行操作后，数组、容器的大小并没有发生改变。

```
1 //将a数组[start,end)区间去重，返回迭代器
2 unique(a + start, a + end);
3
4 //与erase函数结合，达到去重+删除的目的
5 a.erase(unique(ALL(a)), a.end());
```

12.1.7 bit 库与位运算函数 `_builtin_`

```

1 _builtin_popcount(x) // 返回x二进制下含1的数量，例如x=15=(1111)时答案为4
2
3 _builtin_ffs(x) // 返回x右数第一个1的位置(1->idx)，1(1)返回 1, 8(1000)返回 4,
4 26(11010)返回 2
5
6 _builtin_ctz(x) // 返回x二进制下后导0的个数，1(1)返回 0, 8(1000)返回 3
7 bit_width(x) // 返回x二进制下的位数，9(1001)返回 4, 26(11010)返回 5

```

注：以上函数的 `long long` 版本只需要在函数后面加上 `ll` 即可（例如 `_builtin_popcountll(x)`），`unsigned long long` 加上 `ull`。

12.1.8 数字转字符串函数

`itoa` 虽然能将整数转换成任意进制的字符串，但是其不是标准的C函数，且为Windows独有，且不支持 `long long`，建议手写。

```

1 // to_string函数会直接将你的各种类型的数字转换为字符串。
2 // string to_string(T val);
3 double val = 12.12;
4 cout << to_string(val);

```

```

1 // 【不建议使用】itoa允许你将整数转换成任意进制的字符串，参数为待转换整数、目标字符数组、进
2 制。
3 // char* itoa(int value, char* string, int radix);
4 char ans[10] = {};
5 itoa(12, ans, 2);
6 cout << ans << endl; /*1100*/
7 // 长整型函数名ltoa，最高支持到int型上限2^31。ultoa同理。

```

12.1.9 字符串转数字

```

1 // stoi直接使用
2 cout << stoi("12") << endl;
3
4 // 【不建议使用】stoi转换进制，参数为待转换字符串、起始位置、进制。
5 // int stoi(string value, int st, int radix);
6 cout << stoi("1010", 0, 2) << endl; /*10*/
7 cout << stoi("c", 0, 16) << endl; /*12*/
8 cout << stoi("0x3f3f3f3f", 0, 0) << endl; /*1061109567*/
9
10 // 长整型函数名stoll，最高支持到long long型上限2^63。stoull、stod、stold同理。

```

```

1 // atoi直接使用，空字符返回0，允许正负符号，数字字符前有其他字符返回0，数字字符前有空白字符自
2 动去除
3 cout << atoi("12") << endl;
4 cout << atoi(" 12") << endl; /*12*/
5 cout << atoi("-12abc") << endl; /*-12*/
6 cout << atoi("abc12") << endl; /*0*/
7 // 长整型函数名atoll，最高支持到long long型上限2^63。

```

12.1.10 全排列算法 next_permutation、prev_permutation

在提及这个函数时，我们先需要补充几点字典序相关的知识。

对于三个字符所组成的序列 {a,b,c}，其按照字典序的6种排列分别为：

{abc} , {acb} , {bac} , {bca} , {cab} , {cba}

其排序原理是：先固定 a (序列内最小元素)，再对之后的元素排列。而 b < c，所以 abc < acb。同理，先固定 b (序列内次小元素)，再对之后的元素排列。即可得出以上序列。

`next_permutation` 算法，即是按照**字典序顺序**输出的全排列；相对应的，`prev_permutation`则是按照**逆字典序顺序**输出的全排列。可以是数字，亦可以是其他类型元素。其直接在序列上进行更新，故直接输出序列即可。

```

1 int n;
2 cin >> n;
3 vector<int> a(n);
4 // iota(a.begin(), a.end(), 1);
5 for (auto &it : a) cin >> it;
6 sort(a.begin(), a.end());
7
8 do {
9     for (auto it : a) cout << it << " ";
10    cout << endl;
11 } while (next_permutation(a.begin(), a.end()));

```

12.1.11 字符串转换为数值函数 stoi

可以快捷的将一串字符串转换为指定进制的数字。

使用方法

- `stoi(字符串, 0, x进制)` : 将一串 x 进制的字符串转换为 int 型数字。

```

void Solve() {
    cout << stoi("1010", 0, 2) << endl;
    cout << stoi("c", 0, 16) << endl;
    cout << stoi("0x3f3f3f3f", 0, 0) << endl;
    cout << stoi("10", 0, 8) << endl;
    cout << stoll("aaaaaaaaaa", 0, 16) << endl;
}

C:\Users\26099\Desktop\万能头文件.exe
10
12
1061109567
8
11728124029610

```

- `stoll(字符串, 0, x进制)` : 将一串 x 进制的字符串转换为 long long 型数字。
- `stoull, stod, stold` 同理。

12.1.12 数值转换为字符串函数 `to_string`

允许将各种数值类型转换为字符串类型。

```
1 //将数值num转换为字符串s
2 string s = to_string(num);
```

12.1.13 判断非递减 `is_sorted`

```
1 //a数组[start,end)区间是否是非递减的，返回bool型
2 cout << is_sorted(a + start, a + end);
```

12.1.14 累加 `accumulate`

```
1 //将a数组[start,end)区间的元素进行累加，并输出累加和+x的值
2 cout << accumulate(a + start, a + end, x);
```

12.1.15 迭代器 `iterator`

```
1 //构建一个UUU容器的正向迭代器，名字叫it
2 UUU::iterator it;
3
4 vector<int>::iterator it; //创建一个正向迭代器，++ 操作时指向下一个
5 vector<int>::reverse_iterator it; //创建一个反向迭代器，++ 操作时指向上一个
```

12.1.16 其他函数

`exp2(x)` : 返回 2^x

`log2(x)` : 返回 $\log_2(x)$

`gcd(x, y) / lcm(x, y)` : 以 log 的复杂度返回 $\gcd(|x|, |y|)$ 与 $\text{lcm}(|x|, |y|)$ ，且返回值符号也为正数。

12.2 容器与成员函数

12.2.1 元组 `tuple`

```
1 //获取obj对象中的第index个元素—get<index>(obj)
2 //需要注意的是这里的index只能手动输入，使用for循环这样的自动输入是不可以的
3 tuple<string, int, int> Student = {"Wida", 23, 45000};
4 cout << get<0>(Student) << endl; //获取Student对象中的第一个元素，这里的输出结果应
为“Wida”
```

12.2.2 数组 `array`

```
1 array<int, 3> x; // 建立一个包含三个元素的数组x
2
3 [] // 跟正常数组一样，可以使用随机访问
4 cout << x[0]; // 获取数组重的第一个元素
```

12.2.3 变长数组 vector

```

1  resize(n) // 重设容器大小，但是不改变已有元素的值
2  assign(n, 0) // 重设容器大小为n，且替换容器内的内容为0
3
4 // 尽量不要使用[]的形式声明多维变长数组，而是使用嵌套的方式替代
5 vector<int> ver[n + 1]; // 不好的声明方式
6 vector<vector<int>> ver(n + 1);
7
8 // 嵌套时只需要在最后一个注明变量类型
9 vector dis(n + 1, vector<int>(m + 1));
10 vector dis(m + 1, vector(n + 1, vector<int>(n + 1)));

```

12.2.4 栈 stack

栈顶入，栈顶出。先进后出。

```

1 //没有clear函数
2 size() / empty()
3 push(x) //向栈顶插入x
4 top() //获取栈顶元素
5 pop() //弹出栈顶元素

```

12.2.5 队列 queue

队尾进，队头出。先进先出。

```

1 //没有clear函数
2 size() / empty()
3 push(x) //向队尾插入x
4 front() / back() //获取队头、队尾元素
5 pop() //弹出队头元素

```

```

1 //没有clear函数，但是可以用重新构造替代
2 queue<int> q;
3 q = queue<int>();

```

12.2.6 双向队列 deque

```

1 size() / empty() / clear()
2 push_front(x) / push_back(x)
3 pop_front(x) / pop_back(x)
4 front() / back()
5 begin() / end()
6 []

```

12.2.7 优先队列 priority_queue

默认升序（大根堆），自定义排序需要重载 `<`。

```

1 //没有clear函数
2 priority_queue<int, vector<int>, greater<int> > p; //重定义为降序（小根堆）
3 push(x); //向栈顶插入x
4 top(); //获取栈顶元素
5 pop(); //弹出栈顶元素

```

```

1 //重载运算符【注意，符号相反！！！】
2 struct Node {
3     int x; string s;
4     friend bool operator < (const Node &a, const Node &b) {
5         if (a.x != b.x) return a.x > b.x;
6         return a.s > b.s;
7     }
8 };

```

12.2.8 字符串 string

```
1 | size() / empty() / clear()
```

```

1 //从字符串S的S[start]开始，取出长度为len的子串—S.substr(start, len)
2 //len省略时默认取到结尾，超过字符串长度时也默认取到结尾
3 cout << S.substr(1, 12);
4
5 find(x) / rfind(x); //顺序、逆序查找x，返回下标，没找到时返回一个极大值【！建议与 size() 比较，而不要和 -1 比较，后者可能出错】
6 //注意，没有count函数

```

12.2.9 有序、多重有序集合 set、multiset

默认升序（大根堆），set 去重，multiset 不去重， $\mathcal{O}(\log N)$ 。

```

1 set<int, greater<> > s; //重定义为降序（小根堆）
2 size() / empty() / clear()
3 begin() / end()
4 ++ / -- //返回前驱、后继
5
6 insert(x); //插入x
7 find(x) / rfind(x); //顺序、逆序查找x，返回迭代器【迭代器！！！】，没找到时返回end()
8 count(x); //返回x的个数
9 lower_bound(x); //返回第一个>=x的迭代器【迭代器！！！】
10 upper_bound(x); //返回第一个>x的迭代器【迭代器！！！】

```

特殊函数 `next` 和 `prev` 详解：

```

1 auto it = s.find(x); // 建立一个迭代器
2 prev(it) / next(it); // 默认返回迭代器it的前/后一个迭代器
3 prev(it, 2) / next(it, 2); // 可选参数可以控制返回前/后任意个迭代器
4
5 /* 以下是一些应用 */
6 auto pre = prev(s.lower_bound(x)); // 返回第一个<x的迭代器
7 int ed = *prev(s.end(), 1); // 返回最后一个元素

```

`erase(x);` 有两种删除方式：

- 当x为某一元素时，删除所有这个数，复杂度为 $\mathcal{O}(num_x + \log N)$ ；
- 当x为迭代器时，删除这个迭代器。

```

1 //连续头部删除
2 set<int> S = {0, 9, 98, 1087, 894, 34, 756};
3 auto it = S.begin();
4 int len = S.size();
5 for (int i = 0; i < len; ++ i) {
6     if (*it >= 500) continue;
7     it = S.erase(it); //删除所有小于500的元素
8 }
9 //错误用法如下【千万不能这样用！！！】
10 //for (auto it : S) {
11 //    if (it >= 500) continue;
12 //    S.erase(it); //删除所有小于500的元素
13 //}

```

12.2.10 map、multimap

默认升序(大根堆)，map去重，multimap不去重， $\mathcal{O}(\log S)$ ，其中 S 为元素数量。

```

1 map<int, int, greater<>> mp; //重定义为降序(小根堆)
2 size() / empty() / clear()
3 begin() / end()
4 ++ / -- //返回前驱、后继
5
6 insert({x, y}); //插入二元组
7 [] //随机访问，multimap不支持
8 count(x); //返回x为下标的个数
9 lower_bound(x); //返回第一个下标>=x的迭代器
10 upper_bound(x); //返回第一个下标>x的迭代器

```

`erase(x);` 有两种删除方式：

- 当x为某一元素时，删除所有以这个元素为下标的二元组，复杂度为 $\mathcal{O}(num_x + \log N)$ ；
- 当x为迭代器时，删除这个迭代器。

慎用随机访问！ ——当不确定某次查询是否存在于容器中时，不要直接使用下标查询，而是先使用 `count()` 或者 `find()` 方法检查key值，防止不必要的零值二元组被构造。

```

1 int q = 0;
2 if (mp.count(i)) q = mp[i];

```

慎用自带的 pair、tuple 作为key值类型！使用自定义结构体！

```

1 struct fff {
2     LL x, y;
3     friend bool operator < (const fff &a, const fff &b) {
4         if (a.x != b.x) return a.x < b.x;
5         return a.y < b.y;
6     }
7 };
8 map<fff, int> mp;

```

12.2.11 bitset

将数据转换为二进制，从高位到低位排序，以0为最低位。当位数相同时支持全部的位运算。

```

1 // 如果输入的是01字符串，可以直接使用">>"读入
2 bitset<10> s;

```

```

3  cin >> s;
4
5 //使用只含01的字符串构造—bitset<容器长度>B (字符串)
6 string S; cin >> S;
7 bitset<32> B (S);
8
9 //使用整数构造 ( 两种方式 )
10 int x; cin >> x;
11 bitset<32> B1 (x);
12 bitset<32> B2 = x;
13
14 // 构造时，尖括号里的数字不能是变量
15 int x; cin >> x;
16 bitset<x> ans; // 错误构造
17
18 [] //随机访问
19 set(x) //将第x位置1，x省略时默认全部位置1
20 reset(x) //将第x位置0，x省略时默认全部位置0
21 flip(x) //将第x位取反，x省略时默认全部位取反
22 to_ullong() //重转换为ULL类型
23 to_string() //重转换为ULL类型
24 count() //返回1的个数
25 any() //判断是否至少有一个1
26 none() //判断是否全为0
27
28 _Find_fisrt() // 找到从低位到高位第一个1的位置
29 _Find_next(x) // 找到当前位置x的下一个1的位置，复杂度 O(n/w + count)
30
31 bitset<23> B1("11101001"), B2("11101000");
32 cout << (B1 ^ B2) << "\n"; //按位异或
33 cout << (B1 | B2) << "\n"; //按位或
34 cout << (B1 & B2) << "\n"; //按位与
35 cout << (B1 == B2) << "\n"; //比较是否相等
36 cout << B1 << " " << B2 << "\n"; //你可以直接使用cout输出

```

12.2.12 哈希系列 unordered

通常指代 `unordered_map`、`unordered_set`、`unordered_multimap`、`unordered_multiset`，与原版相比不进行排序。

如果将不支持哈希的类型作为 `key` 值代入，编译器就无法正常运行，这时需要我们为其手写哈希函数。而我们写的这个哈希函数的正确性其实并不是特别重要（但是不可以没有），当发生冲突时编译器会调用 `key` 的 `operator ==` 函数进行进一步判断。[参考](#)

12.2.13 对 pair、tuple 定义哈希

```

1 struct hash_pair {
2     template <class T1, class T2>
3     size_t operator()(const pair<T1, T2> &p) const {
4         return hash<T1>()(p.fi) ^ hash<T2>()(p.se);
5     }
6 };
7 unordered_set<pair<int, int>, int, hash_pair> S;
8 unordered_map<tuple<int, int, int>, int, hash_pair> M;

```

12.2.14 对结构体定义哈希

需要两个条件，一个是在结构体中重载等于号（区别于非哈希容器需要重载小于号，如上所述，当冲突时编译器需要根据重载的等于号判断），第二是写一个哈希函数。注意 `hash<>()` 的尖括号中的类型匹配。

```

1 struct fff {
2     string x, y;
3     int z;
4     friend bool operator == (const fff &a, const fff &b) {
5         return a.x == b.x || a.y == b.y || a.z == b.z;
6     }
7 };
8 struct hash_fff {
9     size_t operator()(const fff &p) const {
10         return hash<string>()(p.x) ^ hash<string>()(p.y) ^ hash<int>()(p.z);
11     }
12 };
13 unordered_map<fff, int, hash_fff> mp;

```

12.2.15 对 vector 定义哈希

以下两个方法均可。注意 `hash<>()` 的尖括号中的类型匹配。

```

1 struct hash_vector {
2     size_t operator()(const vector<int> &p) const {
3         size_t seed = 0;
4         for (auto it : p) {
5             seed ^= hash<int>()(it);
6         }
7         return seed;
8     }
9 };
10 unordered_map<vector<int>, int, hash_vector> mp;

```

```

1 namespace std {
2     template<> struct hash<vector<int>> {
3         size_t operator()(const vector<int> &p) const {
4             size_t seed = 0;
5             for (int i : p) {
6                 seed ^= hash<int>()(i) + 0x9e3779b9 + (seed << 6) + (seed >> 2);
7             }
8             return seed;
9         }
10     };
11 }
12 unordered_set<vector<int> > S;

```

12.3 程序标准化

12.3.1 使用 Lambda 函数

- `function` 统一写法

需要注意的是，虽然 `function` 定义时已经声明了返回值类型了，但是有的时候会出错（例如，声明返回 `long long` 但是返回 `int`，原因没去了解），所以推荐在后面使用 `->` 再行声明一遍。

```

1 | function<void(int, int)> clac = [&](int x, int y) -> void {
2 | };
3 | clac(1, 2);
4 |
5 | function<bool(int)> dfs = [&](int x) -> bool {
6 |     return dfs(x + 1);
7 | };
8 | dfs(1);

```

- `auto` 非递归写法

不需要使用递归函数时，直接用 `auto` 替换 `function` 即可。

```

1 | auto clac = [&](int x, int y) -> void {
2 | };

```

- `auto` 递归写法

相较于 `function` 写法，需要额外引用一遍自身。

```

1 | auto dfs = [&](auto self, int x) -> bool {
2 |     return self(self, x + 1);
3 | };
4 | dfs(dfs, 1);

```

12.3.2 使用构造函数

可以将一些必要的声明和预处理放在构造函数，在编译时，无论放置在程序的哪个位置，都会先于主函数进行。下方是我将输入流控制声明的过程。

```

1 | int __FAST_IO__ = []() { // 函数名称可以随意修改
2 |     ios::sync_with_stdio(0), cin.tie(0);
3 |     cout.tie(0);
4 |     cout << fixed << setprecision(12);
5 |     freopen("out.txt", "r", stdin);
6 |     freopen("in.txt", "w", stdout);
7 |     return 0;
8 | }();

```

/END/

13 卡常

13.1 基础算法 | 最大公约数 gcd | 位运算加速

略快于内置函数。

```

1 LL gcd(LL a, LL b) {
2     #define tz __builtin_ctzll
3     if (!a || !b) return a | b;
4     int t = tz(a | b);
5     a >>= tz(a);
6     while (b) {
7         b >>= tz(b);
8         if (a > b) swap(a, b);
9         b -= a;
10    }
11    return a << t;
12    #undef tz
13 }
```

13.2 数论 | 质数判定 | 预分类讨论加速

常数优化，达到 $\mathcal{O}(\frac{\sqrt{N}}{3})$ 。

```

1 bool is_prime(int n) {
2     if (n < 2) return false;
3     if (n == 2 || n == 3) return true;
4     if (n % 6 != 1 && n % 6 != 5) return false;
5     for (int i = 5, j = n / i; i <= j; i += 6) {
6         if (n % i == 0 || n % (i + 2) == 0) {
7             return false;
8         }
9     }
10    return true;
11 }
```

13.3 数论 | 质数判定 | Miller-Rabin

借助蒙哥马利模乘加速取模运算。

```

1 using u64 = uint64_t;
2 using u128 = __uint128_t;
3
4 struct Montgomery {
5     u64 m, m2, im, l1, l2;
6     Montgomery() {}
7     Montgomery(u64 m) : m(m) {
8         l1 = -(u64)m % m, l2 = -(u128)m % m;
9         m2 = m << 1, im = m;
10    for (int i = 0; i < 5; i++) {
11        im *= 2 - m * im;
12    }
13 }
14 inline u64 operator()(i64 a, i64 b) const {
15     u128 c = (u128)a * b;
16     return u64(c >> 64) + m - u64((u64)c * im * (u128)m >> 64);
17 }
18 inline u64 reduce(u64 a) const {
19     a = m - u64(a * im * (u128)m >> 64);
```

```

20         return a >= m ? a - m : a;
21     }
22     inline u64 trans(i64 a) const {
23         return (*this)(a, 12);
24     }
25
26     inline u64 mul(i64 a, i64 b) const {
27         u64 r = (*this)(trans(a), trans(b));
28         return reduce(r);
29     }
30     u64 pow(u64 a, u64 n) {
31         u64 r = 1;
32         a = trans(a);
33         for (; n; n >= 1, a = (*this)(a, a)) {
34             if (n & 1) r = (*this)(r, a);
35         }
36         return reduce(r);
37     }
38 };
39
40 bool isprime(i64 n) {
41     if (n < 2 || n % 6 != 1) {
42         return (n | 1) == 3;
43     }
44     u64 s = __builtin_ctzll(n - 1), d = n >> s;
45     Montgomery M(n);
46     for (i64 a : {2, 325, 9375, 28178, 450775, 9780504, 1795265022}) {
47         u64 p = M.pow(a, d), i = s;
48         while (p != 1 && p != n - 1 && a % n && i--) {
49             p = M.mul(p, p);
50         }
51         if (p != n - 1 && i != s) return false;
52     }
53     return true;
54 }
```

13.4 数论 | 质因数分解 | Pollard-Rho

```

1 struct Montgomery {} M(10); // 注意预赋值
2 bool isprime(i64 n) {}
3
4 i64 rho(i64 n) {
5     if (!(n & 1)) return 2;
6     i64 x = 0, y = 0, prod = 1;
7     auto f = [&](i64 x) -> i64 {
8         return M.mul(x, x) + 5; // 这里的种子能被 hack，如果是在线比赛，请务必 rand 生成
9     };
10    for (int t = 30, z = 0; t % 64 || gcd(prod, n) == 1; ++t) {
11        if (x == y) x = ++z, y = f(x);
12        if (i64 q = M.mul(prod, x + n - y)) prod = q;
13        x = f(x), y = f(f(y));
14    }
15    return gcd(prod, n);
16 }
17
18 vector<i64> factorize(i64 x) {
19     vector<i64> res;
20     auto f = [&](auto f, i64 x) {
21         if (x == 1) return;
22         M = Montgomery(x); // 重设模数
23         if (isprime(x)) return res.push_back(x);
24     };
25 }
```

```

24     i64 y = rho(x);
25     f(f, y), f(f, x / y);
26   };
27   f(f, x), sort(res.begin(), res.end());
28   return res;
29 }

```

13.5 数论 | 取模运算类 | 蒙哥马利模乘

```

1  using u64 = uint64_t;
2  using u128 = __uint128_t;
3
4  struct Montgomery {
5      u64 m, m2, im, l1, l2;
6      Montgomery() {}
7      Montgomery(u64 m) : m(m) {
8          l1 = -(u64)m % m, l2 = -(u128)m % m;
9          m2 = m << 1, im = m;
10         for (int i = 0; i < 5; i++) im *= 2 - m * im;
11     }
12     inline u64 operator()(i64 a, i64 b) const {
13         u128 c = (u128)a * b;
14         return u64(c >> 64) + m - u64((u64)c * im * (u128)m >> 64);
15     }
16     inline u64 reduce(u64 a) const {
17         a = m - u64(a * im * (u128)m >> 64);
18         return a >= m ? a - m : a;
19     }
20     inline u64 trans(i64 a) const {
21         return (*this)(a, l2);
22     }
23
24     inline u64 add(i64 a, i64 b) const {
25         u64 c = trans(a) + trans(b);
26         if (c >= m2) c -= m2;
27         return reduce(c);
28     }
29     inline u64 sub(i64 a, i64 b) const {
30         u64 c = trans(a) - trans(b);
31         if (c >= m2) c += m2;
32         return reduce(c);
33     }
34     inline u64 mul(i64 a, i64 b) const {
35         return reduce((*this)(trans(a), trans(b)));
36     }
37     inline u64 div(i64 a, i64 b) const {
38         a = trans(a), b = trans(b);
39         u64 n = m - 2, inv = l1;
40         for (; n; n >= 1, b = (*this)(b, b))
41             if (n & 1) inv = (*this)(inv, b);
42         return reduce((*this)(a, inv));
43     }
44     u64 pow(u64 a, u64 n) {
45         u64 r = l1;
46         a = trans(a);
47         for (; n; n >= 1, a = (*this)(a, a))
48             if (n & 1) r = (*this)(r, a);
49         return reduce(r);
50     }
51 };

```

13.6 网络流 | 最大流 | 预流推进 HLPP

理论最坏复杂度为 $\mathcal{O}(N^2\sqrt{M})$ ，例题范围： $N = 1200$, $m = 1.2 \times 10^5$ 。

```

1 template <typename T> struct PushRelabel {
2     const int inf = 0x3f3f3f3f;
3     const T INF = 0x3f3f3f3f3f3f3f3f;
4     struct Edge {
5         int to, cap, flow, anti;
6         Edge(int v = 0, int w = 0, int id = 0) : to(v), cap(w), flow(0), anti(id)
7     {};
8     vector<vector<Edge>> e;
9     vector<vector<int>> gap;
10    vector<T> ex; // 超额流
11    vector<bool> ingap;
12    vector<int> h;
13    int n, gobalcnt, maxH = 0;
14    T maxflow = 0;
15
16    PushRelabel(int n) : n(n), e(n + 1), ex(n + 1), gap(n + 1) {}
17    void addedge(int u, int v, int w) {
18        e[u].push_back({v, w, (int)e[v].size()});
19        e[v].push_back({u, 0, (int)e[u].size() - 1});
20    }
21    void PushEdge(int u, Edge &edge) {
22        int v = edge.to, d = min(ex[u], 1LL * edge.cap - edge.flow);
23        ex[u] -= d;
24        ex[v] += d;
25        edge.flow += d;
26        e[v][edge.anti].flow -= d;
27        if (h[v] != inf && d > 0 && ex[v] == d && !ingap[v]) {
28            ++gobalcnt;
29            gap[h[v]].push_back(v);
30            ingap[v] = 1;
31        }
32    }
33    void PushPoint(int u) {
34        for (auto k = e[u].begin(); k != e[u].end(); k++) {
35            if (h[k->to] + 1 == h[u] && k->cap > k->flow) {
36                PushEdge(u, *k);
37                if (!ex[u]) break;
38            }
39        }
40        if (!ex[u]) return;
41        if (gap[h[u]].empty()) {
42            for (int i = h[u] + 1; i <= min(maxH, n); i++) {
43                for (auto v : gap[i]) {
44                    ingap[v] = 0;
45                }
46                gap[i].clear();
47            }
48        }
49        h[u] = inf;
50        for (auto [to, cap, flow, anti] : e[u]) {
51            if (cap > flow) {
52                h[u] = min(h[u], h[to] + 1);
53            }
54        }
55        if (h[u] >= n) return;
56        maxH = max(maxH, h[u]);
57        if (!ingap[u]) {
58            gap[h[u]].push_back(u);
}

```

```

59         ingap[u] = 1;
60     }
61 }
62 void init(int t, bool f = 1) {
63     ingap.assign(n + 1, 0);
64     for (int i = 1; i <= maxH; i++) {
65         gap[i].clear();
66     }
67     gobalcnt = 0, maxH = 0;
68     queue<int> q;
69     h.assign(n + 1, inf);
70     h[t] = 0, q.push(t);
71     while (q.size()) {
72         int u = q.front();
73         q.pop(), maxH = h[u];
74         for (auto &[v, cap, flow, anti] : e[u]) {
75             if (h[v] == inf && e[v][anti].cap > e[v][anti].flow) {
76                 h[v] = h[u] + 1;
77                 q.push(v);
78                 if (f) {
79                     gap[h[v]].push_back(v);
80                     ingap[v] = 1;
81                 }
82             }
83         }
84     }
85 }
86 T work(int s, int t) {
87     init(t, 0);
88     if (h[s] == inf) return maxflow;
89     h[s] = n;
90     ex[s] = INF;
91     ex[t] = -INF;
92     for (auto k = e[s].begin(); k != e[s].end(); k++) {
93         PushEdge(s, *k);
94     }
95     while (maxH > 0) {
96         if (gap[maxH].empty()) {
97             maxH--;
98             continue;
99         }
100        int u = gap[maxH].back();
101        gap[maxH].pop_back();
102        ingap[u] = 0;
103        PushPoint(u);
104        if (gobalcnt >= 10 * n) {
105            init(t);
106        }
107    }
108    ex[s] -= INF;
109    ex[t] += INF;
110    return maxflow = ex[t];
111 }
112 };

```

13.7 快读

注意读入到文件结尾才结束，直接运行会无输出。

```

1 char buf[1 << 21], *p1 = buf, *p2 = buf;
2 inline char getc() {

```

```

3     return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin), p1 == p2) ?
0 : *p1++;
4 }
5 template<typename T> void Cin(T &a) {
6     T ans = 0;
7     bool f = 0;
8     char c = getc();
9     for (; c < '0' || c > '9'; c = getc()) {
10        if (c == '-') f = -1;
11    }
12    for (; c >= '0' && c <= '9'; c = getc()) {
13        ans = ans * 10 + c - '0';
14    }
15    a = f ? -ans : ans;
16 }
17 template<typename T, typename... Args> void Cin(T &a, Args &...args) {
18     Cin(a), Cin(args...);
19 }
20 template<typename T> void Cout(T x) { // 注意，这里输出不带换行
21     if (x < 0) putchar('-'), x = -x;
22     if (x > 9) Cout(x / 10);
23     putchar(x % 10 + '0');
24 }
```

/END/

14 杂类

14.1 统计区间不同数字的数量（离线查询）

核心在于使用 `pre` 数组滚动维护每一个数字出现的最后位置，配以树状数组统计数量。由于滚动维护具有后效性，所以需要离线操作，从前往后更新。时间复杂度 $\mathcal{O}(N \log N)$ ，常数瓶颈在于 `map`，用手造哈希或者离散化可以优化到理想区间；同时也有莫队做法，复杂度稍劣。[例题链接](#)。

```

1 signed main() {
2     int n;
3     cin >> n;
4     vector<int> in(n + 1);
5     for (int i = 1; i <= n; i++) {
6         cin >> in[i];
7     }
8
9     int q;
10    cin >> q;
11    vector<array<int, 3>> query;
12    for (int i = 0; i < q; i++) {
13        int l, r;
14        cin >> l >> r;
15        query.push_back({r, l, i});
16    }
17    sort(query.begin(), query.end());
18
19    vector<pair<int, int>> ans;
20    map<int, int> pre;
21    int st = 1;
22    BIT bit(n);
23    for (auto [r, l, id] : query) {
24        for (int i = st; i <= r; i++, st++) {
25            if (pre.count(in[i])) { // 消除此前操作的影响
26                bit.add(pre[in[i]], -1);
27            }
28            bit.add(i, 1);
29            pre[in[i]] = i; // 更新操作
30        }
31        ans.push_back({id, bit.ask(r) - bit.ask(l - 1)});
32    }
33
34    sort(ans.begin(), ans.end());
35    for (auto [id, w] : ans) {
36        cout << w << endl;
37    }
38}

```

14.2 选数（DFS 解）

从 N 个整数中任选 K 个整数相加。使用 DFS 求解。

```

1 int n, k; cin >> n >> k;
2 vector<int> in(n), now(n);
3 for (auto &it : in) { cin >> it; }
4 auto dfs = [&](auto self, int k, int bit, int idx) -> void {
5     for (int i = idx; i < n; i++) {
6         now[bit] = in[i];
7         if (bit < k - 1) { self(self, k, bit + 1, i + 1); }
8         if (bit == k - 1) {
9             int add = 0;

```

```

10         for (int j = 0; j < k; j++) {
11             add += now[j];
12         }
13         cout << add << endl;
14     }
15 }
16 };
17 dfs(dfs, k, 0, 0);

```

14.3 选数（位运算状压）

```

1 int n, k; cin >> n >> k;
2 vector<int> in(n);
3 for (auto &it : in) { cin >> it; }
4 int comb = (1 << k) - 1, U = 1 << n;
5 while (comb < U) {
6     int add = 0;
7     for (int i = 0; i < n; i++) {
8         if (1 << i & comb) {
9             add += in[i];
10        }
11    }
12    cout << add << "\n";
13
14    int x = comb & -comb;
15    int y = comb + x;
16    int z = comb & ~y;
17    comb = (z / x >> 1) | y;
18 }

```

14.4 网格路径计数

从 $(0, 0)$ 走到 (a, b) ，规定每次只能从 (x, y) 走到左下或者右下，方案数记为 $f(a, b)$ 。

- $f(a, b) = \binom{a+b}{\frac{a+b}{2}}$ ；
- 若路径和直线 $y = k, k \notin [0, b]$ 不能有交点，则方案数为 $f(a, b) - f(a, 2k - b)$ ；
- 若路径和两条直线 $y = k_1, y = k_2$ ($k_1 < 0 \leq b < k_2$) 不能有交点，方案数记为 $g(a, b, k_1, k_2)$ ，可以使用 $\mathcal{O}(N)$ 递归求解；
- 若路径必须碰到 $y = k_1$ 但是不能碰到 $y = k_2$ ，方案数记为 $h(a, b, k_1, k_2)$ ，可以使用 $\mathcal{O}(N)$ 递归求解（递归过程中两条直线距离会越来越大）。

从 $(0, 0)$ 走到 $(a, 0)$ ，规定每次只能走到左下或者右下，且必须有恰好一次传送（向下 b 单位），且不能走到 x 轴下方，方案数为 $\binom{a+1}{\frac{a-b}{2} + k + 1}$ 。

14.5 德州扑克

读入牌型，并且支持两副牌之间的大小比较。[代码参考](#)

```

1 struct card {
2     int suit, rank;
3     friend bool operator < (const card &a, const card &b) {
4         return a.rank < b.rank;
5     }
6     friend bool operator == (const card &a, const card &b) {
7         return a.rank == b.rank;
8     }

```

```

9   friend bool operator != (const card &a, const card &b) {
10    return a.rank != b.rank;
11 }
12 friend auto &operator>> (istream &it, card &c) {
13    string S, T; it >> S;
14    T = "__23456789TJQKA"; //点数
15    FOR (i, 0, T.sz - 1) {
16       if (T[i] == S[0]) C.rank = i;
17    }
18    T = "__SHCD"; //花色
19    FOR (i, 0, T.sz - 1) {
20       if (T[i] == S[1]) C.suit = i;
21    }
22    return it;
23 }
24 };
25 struct game {
26    int level;
27    vector<card> peo;
28    int a, b, c, d, e;
29    int u, v, w, x, y;
30    bool Rk10() { //Rk10: Royal Flush , 五张牌同花色 , 且点数为AKQJT ( 14,13,12,11,10 )
31        sort(ALL(peo));
32        reverse(ALL(peo));
33        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
34 peo[4].rank;
35        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
36 peo[4].suit;
37
38        if (u != v || v != w || w != x || x != y) return 0;
39        if (a == 14 && b == 13 && c == 12 && d == 11 && e == 10) return 1;
40        return 0;
41    }
42    bool Dif(vector<card> &peo) { //专门用于检查A2345这种顺子的情况 ( 这是最小的顺子 )
43        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
44 peo[4].rank;
45        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
46 peo[4].suit;
47
48        if (a != 14 || b != 5 || c != 4 || d != 3 || e != 2) return 0;
49        vector<card> peo2 = {peo[1], peo[2], peo[3], peo[4], peo[0]}; //重新排序
50        peo = peo2;
51        return 1;
52    }
53    bool Rk9() { //Rk9: Straight Flush , 五张牌同花色 , 且顺连 【r1 > r2 > r3 > r4 >
54 r5】
55        sort(ALL(peo));
56        reverse(ALL(peo));
57        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
58 peo[4].rank;
59        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
60 peo[4].suit;
61
62        if (u != v || v != w || w != x || x != y) return 0;
63        if (Dif(peo)) return 1; //特判 : A2345
64        if (a == b + 1 && b == c + 1 && c == d + 1 && d == e + 1) return 1;
65        return 0;
66    }
67    bool Rk8() { //Rk8: Four of a Kind , 四张牌点数一样 【r1 = r2 = r3 = r4】
68        sort(ALL(peo));
69        a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
70 peo[4].rank;
71        u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
72 peo[4].suit;

```

```

64
65     if (a == b && b == c && c == d) return 1;
66     if (b == c && c == d && d == e) {
67         reverse(ALL(peo));
68         return 1;
69     }
70     return 0;
71 }
72     bool Rk7() { //Rk7: Fullhouse , 三张牌点数一样，另外两张点数也一样【r1 = r2 = r3 , r4
= r5】
73     sort(ALL(peo));
74     a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
peo[4].rank;
75     u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
peo[4].suit;
76
77     if (a == b && b == c && d == e) return 1;
78     if (a == b && c == d && d == e) {
79         reverse(ALL(peo));
80         return 1;
81     }
82     return 0;
83 }
84     bool Rk6() { //Rk6: Flush , 五张牌同花色【r1 > r2 > r3 > r4 > r5】
85     sort(ALL(peo));
86     reverse(ALL(peo));
87     a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
peo[4].rank;
88     u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
peo[4].suit;
89
90     if (u != v || v != w || w != x || x != y) return 0;
91     return 1;
92 }
93     bool Rk5() { //Rk5: Straight , 五张牌顺连【r1 > r2 > r3 > r4 > r5】
94     sort(ALL(peo));
95     reverse(ALL(peo));
96     a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
peo[4].rank;
97     u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
peo[4].suit;
98
99     if (Dif(peo)) return 1; //特判：A2345
100    if (a == b + 1 && b == c + 1 && c == d + 1 && d == e + 1) return 1;
101    return 0;
102 }
103     bool Rk4() { //Rk4: Three of a kind , 三张牌点数一样【r1 = r2 = r3 , r4 > r5】
104     sort(ALL(peo));
105     reverse(ALL(peo));
106     a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
peo[4].rank;
107     u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
peo[4].suit;
108
109     if (a == b && b == c) return 1;
110     if (b == c && c == d) {
111         swap(peo[3], peo[0]);
112         return 1;
113     }
114     if (c == d && d == e) {
115         swap(peo[3], peo[0]);
116         swap(peo[4], peo[1]);
117         return 1;
118     }

```

```

119     return 0;
120 }
121 bool Rk3() { //Rk3: Two Pairs , 两张牌点数一样 , 另外有两张点数也一样 (两个对子) 【r1
122 = r2 > r3 = r4】
123     sort(ALL(peo));
124     reverse(ALL(peo));
125     a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
126 peo[4].rank;
127     u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
128 peo[4].suit;
129
130     if (a == b && c == d) return 1;
131     if (a == b && d == e) {
132         swap(peo[2], peo[4]);
133         return 1;
134     }
135     if (b == c && d == e) {
136         swap(peo[0], peo[2]);
137         swap(peo[2], peo[4]);
138         return 1;
139     }
140     return 0;
141 }
142 bool Rk2() { //Rk2: One Pairs , 两张牌点数一样 (一个对子) 【r1 = r2 , r3 > r4 > r5】
143     sort(ALL(peo));
144     reverse(ALL(peo));
145     a = peo[0].rank, b = peo[1].rank, c = peo[2].rank, d = peo[3].rank, e =
146 peo[4].rank;
147     u = peo[0].suit, v = peo[1].suit, w = peo[2].suit, x = peo[3].suit, y =
148 peo[4].suit;
149
150     vector<card> peo2;
151     if (a == b) return 1;
152     if (b == c) {
153         peo2 = {peo[1], peo[2], peo[0], peo[3], peo[4]};
154         peo = peo2;
155         return 1;
156     }
157     if (c == d) {
158         peo2 = {peo[2], peo[3], peo[0], peo[1], peo[4]};
159         peo = peo2;
160         return 1;
161     }
162     if (d == e) {
163         peo2 = {peo[3], peo[4], peo[0], peo[1], peo[2]};
164         peo = peo2;
165         return 1;
166     }
167     return 0;
168 }
169 bool Rk1() { //Rk1: high card
170     sort(ALL(peo));
171     reverse(ALL(peo));
172     return 1;
173 }
174 game (vector<card> New_peo) {
175     peo = New_peo;
176     if (Rk10()) { level = 10; return; }
177     if (Rk9()) { level = 9; return; }
178     if (Rk8()) { level = 8; return; }
179     if (Rk7()) { level = 7; return; }
180     if (Rk6()) { level = 6; return; }
181     if (Rk5()) { level = 5; return; }
182     if (Rk4()) { level = 4; return; }

```

```

178     if (Rk3()) { level = 3; return; }
179     if (Rk2()) { level = 2; return; }
180     if (Rk1()) { level = 1; return; }
181 }
182 friend bool operator < (const game &a, const game &b) {
183     if (a.level != b.level) return a.level < b.level;
184     FOR (i, 0, 4) if (a.peo[i] != b.peo[i]) return a.peo[i] < b.peo[i];
185     return 0;
186 }
187 friend bool operator == (const game &a, const game &b) {
188     if (a.level != b.level) return 0;
189     FOR (i, 0, 4) if (a.peo[i] != b.peo[i]) return 0;
190     return 1;
191 }
192 };
193 void debug(vector<card> peo) {
194     for (auto it : peo) cout << it.rank << " " << it.suit << " ";
195     cout << "\n\n";
196 }
197 int clac(vector<card> Ali, vector<card> Bob) {
198     game atype(Ali), btype(Bob);
199     if (atype < btype) return -1;
200     else if (atype == btype) return 0;
201     return 1;
202 }

```

14.6 N*M 数独字典序最小方案

规则：每个宫大小为 $2^N * 2^M$ ，大图一共由 $M * N$ 个宫组成（总大小即 $2^N 2^M * 2^N 2^M$ ），要求每行、每列、每宫都要出现 1 到 $2^N * 2^M$ 的全部数字。输出字典序最小方案。

下例为 2, 1 和 1, 2 时数独字典序最小的示意。

	0	1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7	8	
1	3	4	1	2	7	8	5	6	
2	5	6	7	8	1	2	3	4	
3	7	8	5	6	3	4	1	2	
4	2	1	4	3	6	5	8	7	
5	4	3	2	1	8	7	6	5	
6	6	5	8	7	2	1	4	3	
7	8	7	6	5	4	3	2	1	

	0	1	2	3	4	5	6	7	
0	1	2	3	4	5	6	7	8	
1	5	6	7	8	1	2	3	4	
2	2	1	4	3	6	5	8	7	
3	6	5	8	7	2	1	4	3	
4	3	4	1	2	7	8	5	6	
5	7	8	5	6	3	4	1	2	
6	4	3	2	1	8	7	6	5	
7	8	7	6	5	4	3	2	1	

公式： (i, j) 格所填的内容为 $(i \bmod 2^N \oplus \left\lfloor \frac{j}{2^M} \right\rfloor) \cdot 2^M + (\left\lfloor \frac{i}{2^N} \right\rfloor \oplus j \bmod 2^M) + 1$ ，注意 i, j 从 0 开始。

14.7 高精度进制转换

2 – 62 进制相互转换。输入格式：“转换前进制 转换后进制 要转换的数据”。注释：进制排序为 0-9，A-Z，a-z。

```

1 struct numpy {
2     vector<int> mp; // 将字符转化为数字
3     vector<char> mp2; // 将数字转化为字符
4     numpy() : mp(123), mp2(62) { // 0-9A-Za-z
5         for (int i = 0; i < 10; i++) mp[i + 48] = i, mp2[i] = i + 48;
6         for (int i = 10; i < 36; i++) mp[i + 55] = i, mp2[i] = i + 55;
7         for (int i = 36; i < 62; i++) mp[i + 61] = i, mp2[i] = i + 61;
8     }
9
10    // 转换前进制 转换后进制 要转换的数据
11    string solve(int a, int b, const string &s) {
12        vector<int> nums, ans;
13        for (auto c : s) {
14            nums.push_back(mp[c]);
15        }
16        reverse(nums.begin(), nums.end());
17        while (nums.size()) { // 短除法，将整个大数一直除 b，取余数
18            int remainder = 0;
19            for (int i = nums.size() - 1; ~i; i--) {
20                nums[i] += remainder * a;
21                remainder = nums[i] % b;
22                nums[i] /= b;
23            }
24            ans.push_back(remainder); // 得到余数
25            while (nums.size() && nums.back() == 0) {
26                nums.pop_back(); // 去掉前导 0
27            }
28        }
29        reverse(ans.begin(), ans.end());
30
31        string sh;
32        for (int i : ans) sh += mp2[i];
33        return sh;
34    }
35};
```

14.8 物品装箱

有 N 个物品，第 i 个物品为 $a[i]$ ，有无限个容量为 C 的空箱子。两种装箱方式，输出需要多少个箱子才能装完所有物品。

14.8.1 从前往后装（线段树解）

选择最前面的能放下物品的箱子放入物品。

```

1 const int N = 1e6 + 10;
2 int T, n, a[N], c, tr[N << 2];
3 void pushup(int u){
4     tr[u] = max(tr[u << 1], tr[u << 1 | 1]);
5 }
6 void build(int u, int l, int r){
7     if (l == r) tr[u] = c;
8     else {
9         int mid = l + r >> 1;
10        build(u << 1, l, mid);
11        build(u << 1 | 1, mid + 1, r);
```

```

12     pushup(u);
13 }
14 }
15 void update(int u, int l, int r, int p, int k){
16     if (l > p || r < p) return;
17     if (l == r) tr[u] -= k;
18     else {
19         int mid = l + r >> 1;
20         update(u << 1, l, mid, p, k);
21         update(u << 1 | 1, mid + 1, r, p, k);
22         pushup(u);
23     }
24 }
25 int query(int u, int l, int r, int k){
26     if (l == r){
27         if (tr[u] >= k) return l;
28         return n + 1;
29     }
30     int mid = l + r >> 1;
31     if (tr[u << 1] >= k) return query(u << 1, l, mid, k);
32     else return query(u << 1 | 1, mid + 1, r, k);
33 }
34 int main() {
35     cin >> n >> c;
36     for (int i = 1; i <= n; i++) cin >> a[i];
37     build(1, 1, n);
38     for (int i = 1; i <= n; i++)
39         update(1, 1, n, query(1, 1, n, a[i]), a[i]);
40     cout << query(1, 1, n, c) - 1 << " ";
41 }
```

14.8.2 选择最优的箱子装 (multiset 解)

选择能放下物品且剩余容量最小的箱子放物品

```

1 void solve(){
2     cin >> n >> c;
3     for (int i = 1; i <= n; i++) cin >> a[i];
4     multiset <int> s;
5     for (int i = 1; i <= n; i++){
6         auto it = s.lower_bound(a[i]);
7         if (it == s.end()) s.insert(c - a[i]);
8         else {
9             int x = *it;
10            // multiset 可以存放重复数据，如果是删除某个值的话，会去掉多个箱子
11            // 导致答案错误，所以直接删除对应位置的元素
12            s.erase(it);
13            s.insert(x - a[i]);
14        }
15    }
16    cout << s.size() << "\n";
17 }
```

14.9 浮点数比较

比较下列浮点数的大小： x^{yz} , x^{zy} , $(x^y)^z$, $(x^z)^y$, y^{xz} , y^{zx} , $(y^x)^z$, $(y^z)^x$, z^{xy} , z^{yx} , $(z^x)^y$ 和 $(z^y)^x$ 。

```
1 vector<pair<ld, int>> val = {
2     {log(x) * pow(y, z), 0}, {log(x) * pow(z, y), 1}, {log(x) * y * z, 2},
3     {log(x) * z * y, 3},     {log(y) * pow(x, z), 4}, {log(y) * pow(z, x), 5},
4     {log(y) * x * z, 6},     {log(y) * z * x, 7},     {log(z) * pow(x, y), 8},
5     {log(z) * pow(y, x), 9}, {log(z) * x * y, 10},    {log(z) * y * x, 11}};
6
7 sort(val.begin(), val.end(), [&](auto x, auto y) {
8     if (equal(x.first, y.first)) return x.second < y.second; // queal比较两个浮点数是否相等
9     return x.first > y.first;
10 });
11 cout << ans[val.front().second] << endl;
```

/END/

14.10 阿达马矩阵 (Hadamard matrix)

构造题用，其有一些性质：将 0 看作 -1 ；1 看作 $+1$ ，整个矩阵可以构成一个 2^k 维向量组，任意两个行、列向量的点积均为 0 [See](#)。例如，在 $k = 2$ 时行向量 $\vec{2}$ 和行向量 $\vec{3}$ 的点积为 $1 \cdot 1 + (-1) \cdot 1 + 1 \cdot (-1) + (-1) \cdot (-1) = 0$ 。

构造方式： $\begin{bmatrix} T & T \\ T & !T \end{bmatrix}$

```
"1111",
"1010",
"11", "1100",
"10", "1001",
```

```
1 int n;
2 cin >> n;
3 int N = pow(2, n);
4 vector<vector<int>> ans(N, vector<int>(N));
5 ans[0][0] = 1;
6 for (int t = 0; t < n; t++) {
7     int m = pow(2, t);
8     for (int i = 0; i < m; i++) {
9         for (int j = m; j < 2 * m; j++) {
10            ans[i][j] = ans[i][j - m];
11        }
12    }
13    for (int i = m; i < 2 * m; i++) {
14        for (int j = 0; j < m; j++) {
15            ans[i][j] = ans[i - m][j];
16        }
17    }
18    for (int i = m; i < 2 * m; i++) {
19        for (int j = m; j < 2 * m; j++) {
20            ans[i][j] = 1 - ans[i - m][j - m];
21        }
22    }
23 }
```

14.11 幻方

构造题用，其有一些性质（保证 N 为奇数）：1 到 N^2 每个数字恰好使用一次，且每行、每列及两条对角线上的数字之和都相同，且为奇数 [See](#)。

构造方式：将 1 写在第一行的中间，随后不断向右上角位置填下一个数字，直到填满。

```
"8 1 6",
"3 5 7",
"4 9 2",
```

```

1 int n;
2 cin >> n;
3 int x = 1, y = (n + 1) / 2;
4 vector<int> ans(n + 1, vector<int>(n + 1));
5 for (int i = 1; i <= n * n; i++) {
6     ans[x][y] = i;
7     if (!ans[(x - 2 + n) % n + 1][y % n + 1]){
8         x = (x - 2 + n) % n + 1;
9         y = y % n + 1;
10    } else {
11        x = x % n + 1;
12    }
13}

```

14.12 最长严格/非严格递增子序列 (LIS)

14.12.1 一维

注意子序列是不连续的。使用二分搜索，以 $\mathcal{O}(N \log N)$ 复杂度通过，另也有 $\mathcal{O}(N^2)$ 的 dp 解法。
dis dis

```

1 vector<int> val; // 堆数
2 for (int i = 1, x; i <= n; i++) {
3     cin >> x;
4     int it = upper_bound(val.begin(), val.end(), x) - val.begin(); // low/upp: 严格/
5     非严格递增
6     if (it >= val.size()) { // 新增一堆
7         val.push_back(x);
8     } else { // 更新对应位置元素
9         val[it] = x;
10    }
11}
11 cout << val.size() << endl;

```

14.12.2 二维+输出方案

```

1 vector<array<int, 3>> in(n + 1);
2 for (int i = 1; i <= n; i++) {
3     cin >> in[i][0] >> in[i][1];
4     in[i][2] = i;
5 }
6 sort(in.begin() + 1, in.end(), [&](auto x, auto y) {
7     if (x[0] != y[0]) return x[0] < y[0];
8     return x[1] > y[1];
9 });
10
11 vector<int> val{0}, idx{0}, pre(n + 1);
12 for (int i = 1; i <= n; i++) {
13     auto [x, y, z] = in[i];
14     int it = lower_bound(val.begin(), val.end(), y) - val.begin(); // low/upp: 严格/
15     非严格递增
16     if (it >= val.size()) { // 新增一堆
17         pre[z] = idx.back();
18         val.push_back(y);
19         idx.push_back(z);
20     } else { // 更新对应位置元素
21         pre[z] = idx[it - 1];
22         val[it] = y;
23         idx[it] = z;
24     }
25 }

```

```

24 }
25
26 vector<int> ans;
27 for (int i = idx.back(); i != 0; i = pre[i]) {
28     ans.push_back(i);
29 }
30 reverse(ans.begin(), ans.end());
31 cout << ans.size() << "\n";
32 for (auto it : ans) {
33     cout << it << " ";
34 }
```

14.13 cout 输出流控制

设置字段宽度：`setw(x)`，该函数可以使得补全 x 位输出，默认用空格补全。

```

1 bool Solve() {
2     cout << 12 << endl;
3     cout << setw(12) << 12 << endl;
4     return 0;
5 }
```



设置填充字符：`setfill(x)`，该函数可以设定补全类型，注意这里的 x 只能为 `char` 类型。

```

1 bool Solve() {
2     cout << 12 << endl;
3     cout << setw(12) << setfill('*') << 12 << endl;
4     return 0;
5 }
```



14.14 读取一行数字，个数未知

```

1 string s;
2 getline(cin, s);
3 stringstream ss;
4 ss << s;
5 while (ss >> s) {
6     auto res = stoi(s);
7     cout << res * 100 << endl;
8 }
```

14.15 约瑟夫问题

n 个人编号 $0, 1, 2 \dots, n - 1$ ，每次数到 k 出局，求最后剩下的人的编号。

$\mathcal{O}(N)$ 。

```

1 | int jos(int n,int k){
2 |     int res=0;
3 |     repeat(i,1,n+1)res=(res+k)%i;
4 |     return res; // res+1, 如果编号从1开始
5 |

```

$\mathcal{O}(K \log N)$ ，适用于 K 较小的情况。

```

1 | int jos(int n,int k){
2 |     if(n==1 || k==1) return n-1;
3 |     if(k>n) return (jos(n-1,k)+k)%n; // 线性算法
4 |     int res=jos(n-n/k,k)-n%k;
5 |     if(res<0)res+=n; // mod n
6 |     else res+=res/(k-1); // 还原位置
7 |     return res; // res+1, 如果编号从1开始
8 |

```

14.16 日期换算（基姆拉尔森公式）

已知年月日，求星期数。

```

1 | int week(int y,int m,int d){
2 |     if(m<=2)m+=12,y--;
3 |     return (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7+1;
4 |

```

14.17 单调队列

查询区间 k 的最大最小值。

```

1 | deque<int> D;
2 | int n,k,x,a[MAX];
3 | int main(){
4 |     IOS();
5 |     cin>>n>>k;
6 |     for(int i=1;i<=n;i++) cin>>a[i];
7 |     for(int i=1;i<=n;i++){
8 |         while(!D.empty() && a[D.back()]<=a[i]) D.pop_back();
9 |         D.emplace_back(i);
10 |        if(!D.empty()) if(i-D.front()>=k) D.pop_front();
11 |        if(i>=k)cout<<a[D.front()]<<endl;
12 |    }
13 |    return 0;
14 |

```

14.18 高精度快速幂

求解 $n^k \bmod p$ ，其中 $0 \leq n, k \leq 10^{1000000}$, $1 \leq p \leq 10^9$ 。容易发现 n 可以直接取模，瓶颈在于 k See。

14.18.1 魔改十进制快速幂（暴力计算）

该算法复杂度 $\mathcal{O}(\text{len}(k))$ 。

```

1 int mypow10(int n, vector<int> k, int p) {
2     int r = 1;
3     for (int i = k.size() - 1; i >= 0; i--) {
4         for (int j = 1; j <= k[i]; j++) {
5             r = r * n % p;
6         }
7         int v = 1;
8         for (int j = 0; j <= 9; j++) {
9             v = v * n % p;
10        }
11        n = v;
12    }
13    return r;
14 }
15 signed main() {
16     string n_, k_;
17     int p;
18     cin >> n_ >> k_ >> p;
19
20     int n = 0; // 转化并计算 n % p
21     for (auto it : n_) {
22         n = n * 10 + it - '0';
23         n %= p;
24     }
25     vector<int> k; // 转化 k
26     for (auto it : k_) {
27         k.push_back(it - '0');
28     }
29     cout << mypow10(n, k, p) << endl; // 暴力快速幂
30 }
```

14.18.2 扩展欧拉定理（欧拉降幂公式）

$$n^k \equiv \begin{cases} n^{k \bmod \varphi(p)} & \gcd(n, p) = 1 \\ n^{k \bmod \varphi(p)+\varphi(p)} & \gcd(n, p) \neq 1 \wedge k \geq \varphi(p) \\ n^k & \gcd(n, p) \neq 1 \wedge k < \varphi(p) \end{cases}$$

最终我们可以将幂降到 $\varphi(p)$ 的级别，使得能够直接使用快速幂解题，复杂度瓶颈在求解欧拉函数 $\mathcal{O}(\sqrt{p})$ 。

```

1 int phi(int n) { //求解 phi(n)
2     int ans = n;
3     for (int i = 2; i <= n / i; i++) {
4         if (n % i == 0) {
5             ans = ans / i * (i - 1);
6             while (n % i == 0) {
7                 n /= i;
8             }
9         }
10    }
11    if (n > 1) { //特判 n 为质数的情况
12        ans = ans / n * (n - 1);
13    }
14    return ans;
15 }
16 signed main() {
17     string n_, k_;
```

```

18 int p;
19 cin >> n_ >> k_ >> p;
20
21 int n = 0; // 转化并计算 n % p
22 for (auto it : n_) {
23     n = n * 10 + it - '0';
24     n %= p;
25 }
26 int mul = phi(p), type = 0, k = 0; // 转化 k
27 for (auto it : k_) {
28     k = k * 10 + it - '0';
29     type |= (k >= mul);
30     k %= mul;
31 }
32 if (type) {
33     k += mul;
34 }
35 cout << mypow(n, k, p) << endl;
36 }

```

14.19 int128 输入输出流控制

int128 只在基于 Lumix 系统的环境下可用，需要 C++20。38位精度，除输入输出外与普通数据类型无差别。该封装支持负数读入，需要注意 `write` 函数结尾不输出多余空格与换行。

```

1 namespace my128 { // 读入优化封装，支持__int128
2     using i64 = __int128_t;
3     i64 abs(const i64 &x) {
4         return x > 0 ? x : -x;
5     }
6     auto &operator>>(istream &it, i64 &j) {
7         string val;
8         it >> val;
9         reverse(val.begin(), val.end());
10        i64 ans = 0;
11        bool f = 0;
12        char c = val.back();
13        val.pop_back();
14        for (; c < '0' || c > '9'; c = val.back(), val.pop_back()) {
15            if (c == '-') {
16                f = 1;
17            }
18        }
19        for (; c >= '0' && c <= '9'; c = val.back(), val.pop_back()) {
20            ans = ans * 10 + c - '0';
21        }
22        j = f ? -ans : ans;
23        return it;
24    }
25    auto &operator<<(ostream &os, const i64 &j) {
26        string ans;
27        function<void(i64)> write = [&](i64 x) {
28            if (x < 0) ans += '-', x = -x;
29            if (x > 9) write(x / 10);
30            ans += x % 10 + '0';
31        };
32        write(j);
33        return os << ans;
34    }
35 } // namespace my128

```

14.20 对拍版子

- 文件控制

```
1 // BAD.cpp, 存放待寻找错误的代码
2 freopen("A.txt", "r", stdin);
3 freopen("BAD.out", "w", stdout);
4
5 // 1.cpp, 存放暴力或正确的代码
6 freopen("A.txt", "r", stdin);
7 freopen("1.out", "w", stdout);
8
9 // Ask.cpp
10 freopen("A.txt", "w", stdout);
```

- C++ 版 bat

```
1 int main() {
2     int T = 1E5;
3     while(T--) {
4         system("BAD.exe");
5         system("1.exe");
6         system("A.exe");
7         if (system("fc BAD.out 1.out")) {
8             puts("WA");
9             return 0;
10        }
11    }
12 }
```

14.21 随机数生成与样例构造

```
1 mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
2 int r(int a, int b) {
3     return rnd() % (b - a + 1) + a;
4 }
5
6
7 void graph(int n, int root = -1, int m = -1) {
8     vector<pair<int, int>> t;
9     for (int i = 1; i < n; i++) { // 先建立一棵以0为根节点的树
10         t.emplace_back(i, r(0, i - 1));
11     }
12
13     vector<pair<int, int>> edge;
14     set<pair<int, int>> uni;
15     if (root == -1) root = r(0, n - 1); // 确定根节点
16     for (auto [x, y] : t) { // 偏移建树
17         x = (x + root) % n + 1;
18         y = (y + root) % n + 1;
19         edge.emplace_back(x, y);
20         uni.emplace(x, y);
21     }
22
23     if (m != -1) { // 如果是图，则在树的基础上继续加边
24         for (int i = n; i <= m; i++) {
25             while (true) {
26                 int x = r(1, n), y = r(1, n);
27                 if (x == y) continue; // 拒绝自环
28                 if (uni.count({x, y})) continue; // 拒绝重边
```

```

29         edge.emplace_back(x, y);
30         uni.emplace(x, y);
31     }
32 }
33 }
34
35 random_shuffle(edge.begin(), edge.end()); // 打乱节点
36 for (auto [x, y] : edge) {
37     cout << x << " " << y << endl;
38 }
39 }
```

14.22 手工哈希

```

1 struct myhash {
2     static uint64_t hash(uint64_t x) {
3         x += 0x9e3779b97f4a7c15;
4         x = (x ^ (x >> 30)) * 0xbff58476d1ce4e5b9;
5         x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
6         return x ^ (x >> 31);
7     }
8     size_t operator()(uint64_t x) const {
9         static const uint64_t SEED =
10            chrono::steady_clock::now().time_since_epoch().count();
11        return hash(x + SEED);
12    }
13    size_t operator()(pair<uint64_t, uint64_t> x) const {
14        static const uint64_t SEED =
15            chrono::steady_clock::now().time_since_epoch().count();
16        return hash(x.first + SEED) ^ (hash(x.second + SEED) >> 1);
17    }
18 } // unordered_map<int, int, myhash>
```

14.23 Python常用语法

14.23.1 读入与定义

- 读入多个变量并转换类型：`X, Y = map(int, input().split())`
- 读入列表：`X = eval(input())`
- 多维数组定义：`X = [[0 for j in range(0, 100)] for i in range(0, 200)]`

14.23.2 格式化输出

- 保留小数输出：`print("{:.12f}".format(X))` 指保留 12 位小数
- 对齐与宽度：`print("{:<12f}".format(X))` 指左对齐，保留 12 个宽度

14.23.3 排序

- 倒序排序：使用 `reverse` 实现倒序 `X.sort(reverse=True)`
- 自定义排序：下方代码实现了先按第一关键字降序、再按第二关键字升序排序。

```

1 | X.sort(key=lambda x: x[1])
2 | X.sort(key=lambda x: x[0], reverse=True)
```

14.23.4 文件IO

- 打开要读取的文件：`r = open('X.txt', 'r', encoding='utf-8')`
- 打开要写入的文件：`w = open('Y.txt', 'w', encoding='utf-8')`
- 按行写入：`w.write(XX)`

14.23.5 增加输出流长度、递归深度

```

1 import sys
2 sys.set_int_max_str_digits(200000)
3 sys.setrecursionlimit(100000)

```

14.23.6 自定义结构体

自定义结构体并且自定义排序

```

1 class node:
2     def __init__(self, A, B, C):
3         self.A = A
4         self.B = B
5         self.C = C
6
7 w = []
8 for i in range(1, 5):
9     a, b, c = input().split()
10    w.append(node(a, b, c))
11 w.sort(key=lambda x: x.C, reverse=True)
12 for i in w:
13     print(i.A, i.B, i.C)
14

```

14.23.7 数据结构

- 模拟于 C^{map}₊₊，定义：`dic = dict()`
- 模拟栈与队列：使用常见的 `list` 即可完成，`list.insert(0, X)` 实现头部插入、`list.pop()` 实现尾部弹出、`list.pop(0)` 实现头部弹出

14.23.8 其他

- 获取ASCII码：`ord()` 函数
- 转换为ASCII字符：`chr()` 函数

14.24 OJ测试

对于一个未知属性的OJ，应当在正式赛前进行以下全部测试：

14.24.1 GNU C++ 版本测试

```

1 for (int i : {1, 2}) {} // GNU C++11 支持范围表达式
2
3 auto cc = [&](int x) { x++; }; // GNU C++11 支持 auto 与 lambda 表达式
4 cc(2);
5
6 tuple<string, int, int> V; // GNU C++11 引入
7 array<int, 3> C; // GNU C++11 引入
8
9 auto dfs = [&](auto self, int x) -> void { // GNU C++14 支持 auto 自递归
10     if (x > 10) return;

```

```

11     self(self, x + 1);
12 };
13 dfs(dfs, 1);
14
15 vector<int> in(1, vector<int>(1)); // GNU C++17 支持 vector 模板类型缺失
16
17 map<int, int> dic;
18 for (auto [u, v] : dic) {} // GNU C++17 支持 auto 解绑
19 dic.contains(12); // GNU C++20 支持 contains 函数
20
21 constexpr double Pi = numbers::pi; // C++20 支持

```

14.24.2 编译器位数测试

```
1 | using i64 = __int128; // 64 位 GNU C++11 支持
```

14.24.3 评测器环境测试

Windows 系统输出 -1；反之则为一个随机数。

```

1 | #define int long long
2 | map<int, int> dic;
3 | int x = dic.size() - 1;
4 | cout << x << endl;

```

14.24.4 运算速度测试

	本地-20(64)	CodeForces-20(64)	AtCoder-20(64)	牛客-17(64)	学院oj	CodeForces-17(32)	马蹄集
4E3量级-硬跑	2454	2886	874	4121	4807	2854	4986
4E3量级-手动加速	556	686	873	1716	1982	2246	2119

```

1 // #pragma GCC optimize("Ofast", "unroll-loops")
2 #include <bits/stdc++.h>
3 using namespace std;
4
5 signed main() {
6     int n = 4E3, cnt = 0;
7     bitset<30> ans;
8     for (int i = 1; i <= n; i++) {
9         for (int j = 1; j <= n; j += 2) {
10            for (int k = 1; k <= n; k += 4) {
11                ans |= i | j | k;
12                cnt++;
13            }
14        }
15    }
16    cout << cnt << "\n";
17 }

```

```

1 // #pragma GCC optimize("Ofast", "unroll-loops")
2
3 #include <bits/stdc++.h>
4 using namespace std;
5 mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());

```

```
6
7 signed main() {
8     size_t n = 340000000, seed = 0;
9     for (int i = 1; i <= n; i++) {
10         seed ^= rnd();
11     }
12
13     return 0;
14 }
```

14.25 编译器设置

```
1 g++ -O2 -std=c++20 -pipe
2 -Wall -Wextra -Wconversion /* 这部分是警告相关，可能用不到 */
3 -fstack-protector
4 -Wl,--stack=268435456
```

/END/



Edited by *Wida*

Version: 1.8.8 (2024.11.19)

[www.github.com/hh2048](https://github.com/hh2048)
www.cnblogs.com/WIDA