

XCPC Templates

Komorebie

September 28, 2024

Contents

1	杂项	4
1.1	快读	4
1.2	__int128 输出流自定义	4
1.3	unordered_map 使用 pair 作为 key	4
1.4	cout 设置精度	4
1.5	对拍相关	5
1.5.1	bat	5
1.5.2	数据生成	5
1.5.3	文件输入输出	5
1.6	库函数	5
1.6.1	位运算函数	5
1.6.2	批量递增赋值函数	6
1.6.3	数组随机打乱	6
1.7	字符串转化	6
1.7.1	数字转字符串	6
1.7.2	字符串转数字	6
2	数据结构	6
2.1	并查集	6
2.2	树状数组	7
2.3	ST 表	8
2.4	线段树	8
2.5	主席树	10
2.6	树上启发式合并	11
2.7	莫队算法	14
3	字符串	14
3.1	KMP	14
3.2	Manacher	15
3.3	trie 树	15
3.4	Z 函数	16
3.5	AC 自动机	16
3.6	回文自动机	18
3.7	后缀排序	19

4 图论	20
4.1 最短路	20
4.1.1 Dijkstra 朴素版	20
4.1.2 Dijkstra 堆优化版	21
4.1.3 SPFA	21
4.1.4 Bellman-Ford	22
4.2 网络流	22
4.2.1 EK	22
4.2.2 Dinic	24
4.2.3 MCMF	26
4.3 tarjan	27
4.3.1 tarjan 求割点	27
4.3.2 tarjan 求强连通分量	28
4.3.3 tarjan 求点双连通分量 (圆方树)	29
4.3.4 tarjan 求边双连通分量	29
4.4 树上问题	30
4.4.1 树的直径	30
4.4.2 树的重心	30
4.4.3 最近公共祖先 (倍增)	30
4.4.4 树链剖分	31
4.5 拓扑排序	32
4.6 染色法判断二分图	33
4.7 匈牙利算法求最大匹配	33
4.8 差分约束	34
5 数学	36
5.1 试除法分解质因数	36
5.2 欧拉筛	36
5.3 欧拉函数和欧拉定理	37
5.3.1 欧拉函数	37
5.3.2 线性筛欧拉函数	37
5.4 莫比乌斯函数	38
5.5 线性筛约数个数	38
5.6 线性筛约数和	39
5.7 裴蜀定理	39
5.8 扩展欧几里得算法	40
5.9 快速幂	40
5.10 BSGS 离散对数	40
5.11 乘法逆元	41
5.12 快速递推求逆元	41
5.13 中国剩余定理	41
5.14 高斯消元	42
5.15 FFT	43
5.15.1 FFT 递归版	43
5.15.2 FFT 迭代版	43
5.16 线性基	44
5.16.1 高斯消元法求线性基	44
5.16.2 贪心法求线性基	44
5.17 杜教筛	45

5.18 数学常见结论	46
5.18.1 插板法	46
5.18.2 组合数学常见性质	47
5.18.3 斯特林数	48
5.18.4 卡特兰数	49
5.18.5 斐波那契数列	49
6 博弈论	50
6.1 巴什博弈	50
6.1.1 朴素巴什博弈	50
6.1.2 扩展巴什博弈	50
6.2 Nim 博弈	51
6.2.1 Nim 博弈	51
6.2.2 Nim_K	51
6.2.3 反 Nim 游戏	51
6.3 SG 游戏	52
6.3.1 SG 定理和 SG 函数	52
6.3.2 反 SG 博弈	52
7 计算几何	52
7.1 计算几何	52

1 杂项

1.1 快读

```
1 inline int read()
2 {
3     int x = 0, f = 1;
4     char ch = getchar();
5     while (ch < '0' || ch > '9') {
6         if (ch == '-') f = -1;
7         ch = getchar();
8     }
9     while (ch >= '0' && ch <= '9') x = x * 10 + ch - '0', ch = getchar();
10    return x * f;
11 }
```

1.2 __int128 输出流自定义

```
1 std::ostream &operator<<(std::ostream &os, __int128 n) {
2     std::string s;
3     while (n) {
4         s += '0' + n % 10;
5         n /= 10;
6     }
7     std::reverse(s.begin(), s.end());
8     return os << s;
9 }
```

1.3 unordered_map 使用 pair 作为 key

对于任意结构体使用哈希，要先重载等于号（冲突时），然后在哈希函数中将所有的哈希值异或起来返回即可。

```
1 struct pair_hash {
2     template <class T1, class T2>
3     size_t operator () (const pair<T1,T2> &p) const {
4         auto h1 = hash<T1>{}(p.first);
5         auto h2 = hash<T2>{}(p.second);
6         return h1 ^ h2; // 哈希组合
7     }
8 };
9
10 unordered_map<pair<int, int>, int, pair_hash>mp; // 使用方法
```

1.4 cout 设置精度

```
1 // 该函数在头文件<iomanip>中
2 // 保留小数点后12位
3 cout << fixed << setprecision(12);
4 // 控制输出流显示浮点数的有效数字个数，会四舍五入
```

```
5 cout << setprecision(12);
```

1.5 对拍相关

1.5.1 bat

```
1 @echo off
2 g++ -o my.exe my.cpp
3 g++ -o brute.exe brute.cpp
4 g++ -o maker.exe maker.cpp
5 :loop
6 maker.exe > data.in
7 my.exe < data.in > my.out
8 brute.exe < data.in > brute.out
9 fc my.out brute.out
10 if not errorlevel 1 goto loop
11 pause
```

1.5.2 数据生成

```
1 srand(time(0)); // 以当前时间作为随机种子, 需要 time.h 库, 每秒钟种子变化一次
2 int a = rand(); // rand 函数生成的整数在 [0, 2^15-1] 之间均匀分布
3 int rrand() { return (rand() << 15) | rand(); } // 生成[0, 2^30-1]间的整数
```

1.5.3 文件输入输出

```
1 freopen("data.txt", "r", stdin);
2 // 从 data.txt 文件中读入, 可以任意改变文件名
3 freopen("output.txt", "w", stdout);
4 // 输出到 output.txt 文件, 可以任意改变文件名
5 /*主体代码*/
6 fclose(stdin);
7 fclose(stdout);
8 // 关闭文件输入输出, 回到键盘输入输出, 不需要可以不写
```

1.6 库函数

1.6.1 位运算函数

```
1 // 返回x二进制下含1的数量
2 cout << __builtin_popcount(x); // 例如x=15时答案为4
3
4 // 返回x二进制下最后一个1的位置 (从1开始计算)
5 cout << __builtin_ffs(x); // 例如x=1答案为1, x=8答案为4
6
7 // 返回x二进制下后导0的个数
8 cout << __builtin_ctz(x); // 例如x=1答案为0, x=8答案为3
9
10 // 返回x二进制下第一个1的位置 (原函数是求前导0的个数)
11 cout << 31 - __builtin_clz(x); // x = 9(1001) 返回 3
12 cout << 63 - __builtin_clzll(x);
```

1.6.2 批量递增赋值函数

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

1.6.3 数组随机打乱

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

1.7 字符串转化

1.7.1 数字转字符串

to_string 函数会直接将你的各种类型的数字转换为字符串。【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、进制。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 // string to_string(T val);
2 double val = 12.12;
3 cout << to_string(val);
4 // char* itoa(int value, char* string, int radix);
5 char ans[10] = {};
6 itoa(12, ans, 2);
7 cout << ans << endl; /*1100*/
8
9 // 长整型函数名ltoa，最高支持到int型上限2^31。ultoa同理。
```

1.7.2 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。
```

2 数据结构

2.1 并查集

并查集究极版（支持维护 size 和到祖先的距离 dis）

```
1 struct DSU {
2     vector<int> p, sz, d;
3     DSU(int n)
4     {
5         p.resize(n + 1);
6         sz.resize(n + 1, 1);
7         d.resize(n + 1, 0);
8         iota(p.begin(), p.end(), 0);
9     }
10
11     int find(int x)
12     {
13         if (p[x] != x) {
14             int u = find(p[x]);
15             d[x] += d[p[x]];
16             p[x] = u;
17         }
18         return p[x];
19     }
20
21     bool same(int a, int b) { return find(a) == find(b); }
22
23     void merge(int a, int b)
24     {
25         p[find(a)] = find(b);
26         sz[b] += sz[a];
27     }
28
29     void merge(int a, int b, int dis)
30     {
31         p[find(a)] = find(b);
32         sz[b] += sz[a];
33         d[find(a)] = dis; // 根据具体问题, 初始化find(a)的偏移量
34     }
35
36     int get_sz(int a) { return sz[find(a)]; }
37 };
```

2.2 树状数组

```
1 int tr[N];
2
3 int lowbit(int x) { return x & -x; }
4
5 void add(int x, int c) // 第x位加上c
6 {
7     for (int i = x; i <= n; i += lowbit(i)) {
8         tr[i] += c;
9     }
10 }
11
```

```

12 int sum(int x) // 求前x位的和
13 {
14     int res = 0;
15     for (int i = x; i; i -= lowbit(i)) {
16         res += tr[i];
17     }
18     return res;
19 }

```

2.3 ST 表

```

1  template <typename T> struct ST {
2      const int B = 30;
3      int n;
4      vector<vector<T>> a;
5      ST(int n) : n(n) { a.resize(n + 1, vector<T>(B)); }
6      ST(vector<T> nums)
7      {
8          this->n = nums.size() - 1;
9          a.resize(n + 1, vector<T>(B));
10         for (int i = 1; i <= n; i++) a[i][0] = nums[i];
11         build();
12     }
13     ST(T nums[], int n)
14     {
15         this->n = n;
16         a.resize(n + 1, vector<T>(B));
17         for (int i = 1; i <= n; i++) a[i][0] = nums[i];
18         build();
19     }
20     T calc(T x, T y) { return max(x, y); }
21     void build()
22     {
23         for (int j = 1; (1 << j) <= n; j++)
24             for (int i = 1; i + (1 << j) - 1 <= n; i++) a[i][j] = calc(a[i][j - 1], a[
                i + (1 << (j - 1))][j - 1]);
25     }
26     T query(int l, int r)
27     {
28         if (r < l) return 0;
29         int k = log2(r - l + 1);
30         return calc(a[l][k], a[r - (1 << k) + 1][k]);
31     }
32 };

```

2.4 线段树

支持区间修改和查询区间和（懒标记）。

```

1 struct Seg_tree {
2     struct node {

```



```

3     int l, r;
4     ll sum, add;
5 };
6 vector<node> tr;
7 vector<int> a;
8 Seg_tree(int n)
9 {
10     tr.resize((n << 2) + 1);
11     a.resize(n + 1);
12     build(1, 1, n);
13 }
14 Seg_tree(vector<int> nums)
15 {
16     int n = nums.size() - 1;
17     a.assign(nums.begin(), nums.end());
18     tr.resize((n << 2) + 1);
19     build(1, 1, n);
20 }
21
22 void pushup(int u) { tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum; }
23
24 void pushdown(int u)
25 {
26     auto &root = tr[u], &left = tr[u << 1], &right = tr[u << 1 | 1];
27     if (root.add) {
28         left.add += root.add, right.add += root.add;
29         left.sum += (left.r - left.l + 1) * root.add, right.sum += (right.r -
30             right.l + 1) * root.add;
31         root.add = 0;
32     }
33 }
34 void build(int u, int l, int r)
35 {
36     if (l == r)
37         tr[u] = {l, r, a[l], 0};
38     else {
39         tr[u] = {l, r};
40         int mid = l + r >> 1;
41         build(u << 1, l, mid);
42         build(u << 1 | 1, mid + 1, r);
43         pushup(u);
44     }
45 }
46
47 void modify(int u, int l, int r, ll v)
48 {
49     if (tr[u].l >= l && tr[u].r <= r) {
50         tr[u].sum += 1ll * (tr[u].r - tr[u].l + 1) * v;
51         tr[u].add += v;
52     }
53     else {
54         pushdown(u);

```

```

55         int mid = tr[u].l + tr[u].r >> 1;
56         if (l <= mid) modify(u << 1, l, r, v);
57         if (r > mid) modify(u << 1 | 1, l, r, v);
58         pushup(u);
59     }
60 }
61
62 ll query(int u, int l, int r)
63 {
64     if (tr[u].l >= l && tr[u].r <= r)
65         return tr[u].sum;
66     else {
67         pushdown(u);
68         int mid = tr[u].l + tr[u].r >> 1;
69         ll sum = 0;
70         if (l <= mid) sum += query(u << 1, l, r);
71         if (r > mid) sum += query(u << 1 | 1, l, r);
72         return sum;
73     }
74 }
75 };

```

2.5 主席树

注意先离散化再将离散化后的值逐个插入 (modify)。

```

1 struct PresidentTree {
2     static constexpr int N = 2e5 + 10;
3     int ls[N * 25], rs[N * 25], idx = 0, cnt[N * 25], root[N * 25];
4
5     void modify(int& cur, int past, int x, int l, int r)
6     {
7         cur = ++idx;
8         ls[cur] = ls[past];
9         rs[cur] = rs[past];
10        cnt[cur] = cnt[past] + 1;
11        if (l == r) return;
12        int mid = l + r >> 1;
13        if (x <= mid)
14            modify(ls[cur], ls[past], x, l, mid);
15        else
16            modify(rs[cur], rs[past], x, mid + 1, r);
17    }
18
19    int query(int lx, int rx, int l, int r, double v) // [l, r] 中第一个出现次数严格大于
        v 的数, 不存在则输出 -1
20    {
21        if (l == r) return l - 1;
22        int mid = l + r >> 1;
23        int res = -1;
24        if ((double)(cnt[ls[rx]]) - (double)(cnt[ls[lx]]) > v) res = query(ls[lx], ls
            [rx], l, mid, v);

```

```

25     if (res == -1 && (double)(cnt[rs[rx]]) - (double)(cnt[rs[lx]]) > v) res =
        query(rs[lx], rs[rx], mid + 1, r, v);
26     return res;
27 }
28
29 int kth(int lx, int rx, int l, int r, int k) // [l, r] 中第k小的数
30 {
31     if (l == r) return l - 1;
32     int res = cnt[ls[rx]] - cnt[ls[lx]];
33     int mid = l + r >> 1;
34     if (k <= res)
35         return kth(ls[lx], ls[rx], l, mid, k);
36     else
37         return kth(rs[lx], rs[rx], mid + 1, r, k - res);
38 }
39 } T;

```

2.6 树上启发式合并

```

1 void dfs(int u, int f) // 与重链剖分相同的写法找重儿子
2 {
3     siz[u] = 1;
4     for (int i = Head[u]; ~i; i = Edge[i].next) {
5         int v = Edge[i].to;
6         if (v == f) continue;
7         dfs(v, u);
8         siz[u] += siz[v];
9         if (siz[v] > siz[son[u]]) son[u] = v;
10    }
11 }
12 int col[maxn], cnt[maxn]; // col存放某结点的颜色, cnt存放某颜色在“当前”子树中的数量
13 long long ans[maxn], sum; // ans是答案数组, sum用于累加计算出“当前”子树的答案
14 int flag, maxc; // flag用于标记重儿子, maxc用于更新最大值
15 // TODO 统计某结点及其所有轻儿子的贡献
16 void count(int u, int f, int val)
17 {
18     cnt[col[u]] += val; // val为正为负可以控制是增加贡献还是删除贡献
19     if (cnt[col[u]] > maxc) // 找最大值, 基操吧
20     {
21         maxc = cnt[col[u]];
22         sum = col[u];
23     }
24     else if (cnt[col[u]] == maxc) // 这样做是因为如果两个颜色数量相同那都得算
25         sum += col[u];
26     for (int i = Head[u]; ~i; i = Edge[i].next) // 排除被标记的重儿子, 统计其它儿子子树
        信息
27     {
28         int v = Edge[i].to;
29         if (v == f || v == flag) continue; // 不能写if(v==f||v==son[u]) continue;
30         count(v, u, val);
31     }
32 }

```

```

33 // dsu on tree的板子
34 void dfs(int u, int f, bool keep)
35 {
36     // 第一步：搞轻儿子及其子树算其答案删贡献
37     for (int i = Head[u]; ~i; i = Edge[i].next) {
38         int v = Edge[i].to;
39         if (v == f || v == son[u]) continue;
40         dfs(v, u, false);
41     }
42     // 第二步：搞重儿子及其子树算其答案不删贡献
43     if (son[u]) {
44         dfs(son[u], u, true);
45         flag = son[u];
46     }
47     // 第三步：暴力统计u及其所有轻儿子的贡献合并到刚算出的重儿子信息里
48     count(u, f, 1);
49     flag = 0;
50     ans[u] = sum;
51     // 把需要删除贡献的删一删
52     if (!keep) {
53         count(u, f, -1);
54         sum = maxc = 0; // 这是因为count函数中会改变这两个变量值
55     }
56 }
57 /***** 另一种写法 *****/
58 #include <bits/stdc++.h>
59 using namespace std;
60 #define endl '\n'
61 #define ll long long
62 #define PII pair<int, int>
63 #define pi acos(-1.0)
64 const int N = 1e5 + 10;
65 int n;
66 int a[N];
67 ll ans = 0;
68 vector<int> G[N];
69 vector<int> path;
70 int son[N], sz[N];
71 int cnt[22][2][(int)1e6 + 10];
72 int lca;
73
74 void dfs(int u, int fa)
75 {
76     sz[u] = 1;
77     for (int v : G[u]) {
78         if (v == fa) continue;
79         dfs(v, u);
80         sz[u] += sz[v];
81         if (sz[v] > sz[son[u]]) son[u] = v;
82     }
83 }
84
85 void calc(int u, int fa)

```

```

86 {
87     path.push_back(u);
88     if ((a[u] ^ lca) < (int)1e6 + 10)
89         for (int j = 0; j <= 20; j++) ans += 111 * cnt[j][((u >> j) & 1) ^ 1][a[u] ^
            lca] * (1 << j);
90     for (int v : G[u]) {
91         if (v == fa) continue;
92         calc(v, u);
93     }
94 }
95
96 void insert(int u, int fa)
97 {
98     for (int j = 0; j <= 20; j++) cnt[j][(u >> j) & 1][a[u]]++;
99     for (int v : G[u]) {
100         if (v == fa) continue;
101         insert(v, u);
102     }
103 }
104
105 void del(int u, int fa)
106 {
107     for (int j = 0; j <= 20; j++) cnt[j][(u >> j) & 1][a[u]]--;
108     for (int v : G[u]) {
109         if (v == fa) continue;
110         del(v, u);
111     }
112 }
113
114 void dfs(int u, int fa, bool keep)
115 {
116     for (int v : G[u]) {
117         if (v == fa || v == son[u]) continue;
118         dfs(v, u, false);
119     }
120     if (son[u]) {
121         dfs(son[u], u, true);
122     }
123     lca = a[u];
124     for (int v : G[u]) {
125         if (v == fa || v == son[u]) continue;
126         calc(v, u);
127         for (auto vv : path)
128             for (int j = 0; j <= 20; j++) cnt[j][(vv >> j) & 1][a[vv]]++;
129         path.clear();
130     }
131     for (int j = 0; j <= 20; j++) cnt[j][(u >> j) & 1][a[u]]++;
132     if (!keep) {
133         del(u, fa);
134     }
135 }
136
137 int main()

```

```

138 {
139     // ios::sync_with_stdio(false), cin.tie(nullptr);
140     scanf("%d", &n);
141     for (int i = 1; i <= n; i++) scanf("%d", a + i);
142     for (int i = 1; i <= n - 1; i++) {
143         int u, v;
144         scanf("%d%d", &u, &v);
145         G[u].push_back(v), G[v].push_back(u);
146     }
147     dfs(1, 0);
148     dfs(1, 0, true);
149     printf("%lld\n", ans);
150     return 0;
151 }

```

2.7 莫队算法

一定要先拓展区间再收缩区间，防止区间减到 0 及以下。

```

1 struct query {
2     int l, r, pos, id;
3     bool operator<(query a)
4     {
5         if (a.pos == this->pos) return a.pos & 1 ? this->r < a.r : this->r > a.r; //
           右端点奇偶波浪排序
6         return this->pos < a.pos;
7     }
8 } q[N];
9
10 int len = sqrt(n);
11 for (int i = 1; i <= m; i++)
12 {
13     cin >> q[i].l >> q[i].r;
14     q[i].id = i;
15     q[i].pos = q[i].l / len;
16 }
17 sort(q + 1, q + 1 + m, cmp1);
18 for (int i = 1, l = 1, r = 0; i <= m; i++)
19 {
20     while (l > q[i].l) add(--l);
21     while (r < q[i].r) add(++r);
22     while (l < q[i].l) sub(l++);
23     while (r > q[i].r) sub(r--);
24     ans[q[i].id] = res;
25 }

```

3 字符串

3.1 KMP

```

1 // 求Next数组:

```

```

2 // s[]是模式串, p[]是模板串, n是s的长度, m是p的长度
3 for (int i = 2, j = 0; i <= m; i ++ )
4 {
5     while (j && p[i] != p[j + 1]) j = ne[j];
6     if (p[i] == p[j + 1]) j ++ ;
7     ne[i] = j;
8 }
9 // 匹配
10 for (int i = 1, j = 0; i <= n; i ++ )
11 {
12     while (j && s[i] != p[j + 1]) j = ne[j];
13     if (s[i] == p[j + 1]) j ++ ;
14     if (j == m)
15     {
16         j = ne[j];
17         // 匹配成功后的逻辑
18     }
19 }

```

3.2 Manacher

```

1 std::vector<int> manacher(std::string s) // 最后减一才是半径
2 {
3     std::string t = "#";
4     for (auto c : s) {
5         t += c, t += '#';
6     }
7     int n = t.size();
8     std::vector<int> r(n);
9     for (int i = 0, j = 0; i < n; i++) {
10         if (2 * j - i >= 0 && j + r[j] > i) {
11             r[i] = std::min(r[2 * j - i], j + r[j] - i);
12         }
13         while (i - r[i] >= 0 && i + r[i] < n && t[i - r[i]] == t[i + r[i]]) {
14             r[i] += 1;
15         }
16         if (i + r[i] > j + r[j]) {
17             j = i;
18         }
19     }
20     return r;
21 }

```

3.3 trie 树

```

1 int son[N][26], cnt[N], idx;
2 // 0号点既是根节点, 又是空节点
3 // son[][]存储树中每个节点的子节点
4 // cnt[]存储以每个节点结尾的单词数量
5
6 // 插入一个字符串

```

```

7 void insert(char* str)
8 {
9     int p = 0;
10    for (int i = 0; str[i]; i++) {
11        int u = str[i] - 'a';
12        if (!son[p][u]) son[p][u] = ++idx;
13        p = son[p][u];
14    }
15    cnt[p]++;
16 }
17
18 // 查询字符串出现的次数
19 int query(char* str)
20 {
21     int p = 0;
22     for (int i = 0; str[i]; i++) {
23         int u = str[i] - 'a';
24         if (!son[p][u]) return 0;
25         p = son[p][u];
26     }
27     return cnt[p];
28 }

```

3.4 Z 函数

$$z[i] = lcp(s[1 : n - 1], s[i : n - 1])$$

```

1 vector<int> get_z(string s)
2 {
3     int n = s.size();
4     vector<int> z(n);
5     for (int i = 1, l = 0; i < n; i++)
6     {
7         if (i <= l + z[l] - 1) z[i] = min(z[i - l], l + z[l] - i);
8         while (i + z[i] < n && s[i + z[i]] == s[z[i]]) z[i]++;
9         if (i + z[i] > l + z[l]) l = i;
10    }
11    return z;
12 } //最后需要修改z[0] = n;

```

3.5 AC 自动机

```

1 struct ACAM {
2     int n;
3     int trie_size = 10;
4     vector<string> T;
5     vector<vector<int>> trie;
6     vector<int> pos, fail, cnt, id, q, end;
7
8     void init()
9     {

```



```

10     trie.resize(trie_size, vector<int>(26, 0));
11     pos.resize(n, 0);
12     fail.resize(trie_size, 0);
13     cnt.resize(trie_size, 0);
14     id.resize(trie_size, -1);
15     end.resize(trie_size, 0);
16 }
17 void build()
18 {
19     int idx = 1; // trie树
20     for (int i = 0; i < n; i++) {
21         int p = 1;
22         for (auto c : T[i]) {
23             int cur = c - 'a';
24             if (!trie[p][cur]) trie[p][cur] = ++idx;
25             p = trie[p][cur];
26         }
27         pos[i] = p;
28         id[p] = i;
29         end[p]++; // 统计以该节点结尾的模式串个数
30     }
31     auto& q = this->q; // 处理根节点的回跳
32     int ql = 0;
33     for (auto& c : trie[1]) {
34         if (c) {
35             fail[c] = 1;
36             q.push_back(c);
37         }
38         else
39             c = 1;
40     }
41     while (ql < q.size()) // BFS
42     {
43         int u = q[ql++];
44         for (int c = 0; c < 26; c++) {
45             if (trie[u][c]) // 有儿子存在时
46             {
47                 fail[trie[u][c]] = trie[fail[u]][c]; // 回跳边 (四边形)
48                 q.push_back(trie[u][c]);
49             }
50             else {
51                 trie[u][c] = trie[fail[u]][c]; // 转移边 (三角形)
52             }
53         }
54         end[u] += end[fail[u]];
55     }
56 }
57
58 void count(string S) // 统计每个结点在文本串中出现次数
59 {
60     auto& q = this->q;
61     for (int cur = 1, i = 0; i < S.size(); i++) {
62         int nxt = trie[cur][S[i] - 'a'];

```

```

63         cnt[cur = nxt]++;
64     }
65
66     reverse(q.begin(), q.end());
67     for (auto cur : q) {
68         cnt[fail[cur]] += cnt[cur];
69     }
70 }
71 };

```

3.6 回文自动机

应用

- 本质不同回文子串个数
- 回文子串出现次数

每个节点代表一个回文串，每个节点的 `len` 表示回文串的长度，`num` 表示回文串出现次数。0 号节点表示长度为 0 的回文串，1 号节点表示长度为 -1 的回文串。

```

1  struct PAM {
2      int len[N], num[N], fail[N], trie[N][26], tot = 1;
3      int getfail(int x, int i, string s)
4      {
5          while (i - len[x] - 1 < 0 || s[i] != s[i - len[x] - 1]) x = fail[x];
6          return x;
7      }
8      void build(string s)
9      {
10         int cur = 0;
11         fail[0] = 1, len[1] = -1;
12         for (int i = 0; i < s.size(); i++) {
13             int u = s[i] - 'A';
14             int pos = getfail(cur, i, s);
15             if (!trie[pos][u]) {
16                 trie[pos][s[i]] = ++tot;
17                 fail[tot] = trie[getfail(fail[pos], i, s)][u];
18                 len[tot] = len[pos] + 2;
19             }
20             cur = trie[pos][u];
21             num[cur]++;
22         }
23         for (int i = tot; i >= 2; i--) num[fail[i]] += num[i];
24     }
25 }
26
27 // 用法
28 void dfs(int u1, int u2)
29 {
30     if (u1 > 1 && u2 > 1) ans += 1ll * A.num[u1] * B.num[u2];
31     for (int i = 0; i < 26; i++) {

```

```

32     if (A.trie[u1][i] && B.trie[u2][i]) dfs(A.trie[u1][i], B.trie[u2][i]);
33 }
34 }
35 int main()
36 {
37     dfs(0, 0);
38     dfs(1, 1);
39 }

```

3.7 后缀排序

sa 表示排第 i 的后缀的前一半是第几个后缀, $sa2$ 表示排第 i 的后缀的后一半是第几个后缀, rk 表示第 i 个后缀排第几, 有 $sa[rk[i]] = i, rk[sa[i]] = i$.

$height[i] : lcp(sa[i], sa[i-1])$, 即排名为 i 的后缀与排名为 $i-1$ 的后缀的最长公共前缀。

$height[rk[i]]$, 即 i 号后缀与它前一名的后缀的最长公共前缀。

经典应用:

- 两个后缀的最大公共前缀: $lcp(x, y) = \min(height[x - y])$, 用 RMQ 维护, $O(1)$ 查询。
- 可重叠最长重复子串: $height$ 数组中最大值
- 本质不同的字串的数量: 枚举每一个后缀, 第 i 个后缀对答案的贡献为 $len - sa[i] + 1 - height[i]$

```

1  int sa[N], rk_base[2][N], *rk = rk_base[0], *rk2 = rk_base[1], sa2[N], cnt[N],
    height[N];
2
3  void get_sa(const char* s, int n)
4  {
5      int m = 122;
6      for (int i = 0; i <= m; ++i) cnt[i] = 0;
7      for (int i = 1; i <= n; ++i) cnt[rk[i] = s[i]] += 1;
8      for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
9      for (int i = 1; i <= n; ++i) sa[cnt[rk[i]]--] = i;
10     for (int d = 1; d <= n; d <= 1)
11     {
12         // 按第二关键字排序
13         int p = 0;
14         for (int i = n - d + 1; i <= n; i++) sa2[++p] = i;
15         for (int i = 1; i <= n; ++i)
16             if (sa[i] > d) sa2[++p] = sa[i] - d;
17         // 按第一关键字排序
18         for (int i = 0; i <= m; ++i) cnt[i] = 0;
19         for (int i = 1; i <= n; ++i) cnt[rk[i]] += 1;
20         for (int i = 1; i <= m; ++i) cnt[i] += cnt[i - 1];
21         for (int i = n; i; --i) sa[cnt[rk[sa2[i]]]--] = sa2[i];
22
23         rk2[sa[1]] = 1;
24         for (int i = 2; i <= n; ++i) rk2[sa[i]] = rk2[sa[i - 1]] + (rk[sa[i]] != rk[
            sa[i - 1]] || rk[sa[i] + d] != rk[sa[i - 1] + d]);

```

```

25
26     std::swap(rk, rk2);
27
28     m = rk[sa[n]];
29     if (m == n) break;
30 }
31 }
32
33 void get_height()
34 {
35     for (int i = 1; i <= n; i++) rk[sa[i]] = i;
36     for (int i = 1, k = 0; i <= n; i++)
37     {
38         if (rk[i] == 1) continue;
39         if (k) k--;
40         int j = sa[rk[i] - 1];
41         while (s[i + k] == s[j + k]) k++;
42         height[rk[i]] = k;
43     }
44 }

```

4 图论

4.1 最短路

4.1.1 Dijkstra 朴素版

时间复杂度: $O(n^2 + m)$

```

1  int g[N][N]; // 存储每条边
2  int dist[N]; // 存储1号点到每个点的最短距离
3  bool st[N]; // 存储每个点的最短路是否已经确定
4
5  // 求1号点到n号点的最短路, 如果不存在则返回-1
6  int dijkstra()
7  {
8      memset(dist, 0x3f, sizeof dist);
9      dist[1] = 0;
10
11     for (int i = 0; i < n - 1; i++)
12     {
13         int t = -1; // 在还未确定最短路的点中, 寻找距离最小的点
14         for (int j = 1; j <= n; j++)
15             if (!st[j] && (t == -1 || dist[t] > dist[j]))
16                 t = j;
17
18         // 用t更新其他点的距离
19         for (int j = 1; j <= n; j++)
20             dist[j] = min(dist[j], dist[t] + g[t][j]);
21
22         st[t] = true;
23     }

```

```

24
25     if (dist[n] == 0x3f3f3f3f) return -1;
26     return dist[n];
27 }

```

4.1.2 Dijkstra 堆优化版

时间复杂度: $O(m \log n)$

```

1  int dist[N]; // 存储所有点到1号点的距离
2  bool st[N]; // 存储每个点的最短距离是否已确定
3  // 求1号点到n号点的最短距离, 如果不存在, 则返回-1
4  int dijkstra()
5  {
6      memset(dist, 0x3f, sizeof dist);
7      dist[1] = 0;
8      priority_queue<PII, vector<PII>, greater<PII>> heap;
9      heap.push({0, 1}); // first存储距离, second存储节点编号
10     while (heap.size())
11     {
12         auto [d, u] = heap.top();
13         heap.pop();
14         if (st[u]) continue;
15         st[u] = true;
16
17         for (auto [v, w] : G[u])
18         {
19             if (dist[v] > distance + w)
20             {
21                 dist[v] = distance + w;
22                 heap.push({dist[v], v});
23             }
24         }
25     }
26
27     if (dist[n] == 0x3f3f3f3f) return -1;
28     return dist[n];
29 }

```

4.1.3 SPFA

最坏时间复杂度 $O(nm)$

```

1  int vis[N], cnt[N];
2  vector<PII> G[N];
3
4  bool spfa(int n, int s)
5  {
6      memset(dis, 0x3f, sizeof(dis));
7      dis[s] = 0;
8      queue<int> q;
9      while (q.size()) {

```

```

10     int u = q.front();
11     q.pop();
12     vis[u] = 0;
13     for (auto [v, w] : G[u]) {
14         if (dis[v] >= dis[u] + w) {
15             dis[v] = dis[u] + w;
16             cnt[v] = cnt[u] + 1; // 记录经过了多少点
17             if (cnt[v] >= n) // 存在负环
18                 return false;
19             if (!vis[v]) {
20                 q.push(v);
21                 vis[v] = 1;
22             }
23         }
24     }
25 }
26 return true;
27 }

```

4.1.4 Bellman-Ford

```

1 struct EDGE // for bellman-ford
2 {
3     int u, int v, int w;
4 };
5
6 vector<EDGE> edges;
7
8 bool Bellman_ford(int n, int s)
9 {
10     memset(dis, 0x3f, sizeof(dis));
11     dis[s] = 0;
12     bool flag;
13     for (int i = 1; i <= n; i++) {
14         flag = false;
15         for (auto [u, v, w] : edges) {
16             if (dis[u] == 0x3f3f3f3f) continue;
17             if (dis[v] > dis[u] + w) {
18                 dis[v] = dis[u] + w;
19                 flag = true;
20             }
21         }
22         if (!flag) break;
23     }
24     return flag; // 第n轮循环仍可以松弛说明存在负环
25 }

```

4.2 网络流

4.2.1 EK

```

1  memset(h, -1, sizeof(h)); // 将h初始化为-1, 因为边的标号从0开始。
2
3  void add(int a, int b, int c)
4  {
5      e[idx] = b, ne[idx] = h[a], f[idx] = c, h[a] = idx++;
6      e[idx] = a, ne[idx] = h[b], f[idx] = 0, h[b] = idx++;
7  }
8
9  bool bfs()
10 {
11     queue<int> q;
12     memset(st, false, sizeof(st));
13     q.push(S);
14     st[S] = true;
15     d[S] = inf;
16     while (q.size()) {
17         int u = q.front();
18         q.pop();
19         for (int i = h[u]; i != -1; i = ne[i]) {
20             int v = e[i];
21             if (!st[v] && f[i]) {
22                 st[v] = true;
23                 d[v] = min(d[u], f[i]);
24                 pre[v] = i;
25                 if (v == T) return true;
26                 q.push(v);
27             }
28         }
29     }
30     return false;
31 }
32
33 int EK()
34 {
35     int r = 0;
36     while (bfs()) {
37         r += d[T];
38         for (int i = T; i != S; i = e[pre[i ^ 1]]) {
39             f[pre[i]] -= d[T];
40             f[pre[i ^ 1]] += d[T];
41         }
42     }
43     return r;
44 }
45
46 struct EK {
47     const int n, S, T;
48     vector<int> h, e, ne, cur, f, d, pre, st;
49
50     EK(int n, int S, int T) : n(n), S(S), T(T) // 点数, 源点, 汇点
51     {
52         h.resize(n + 1, -1);
53         d.resize(n + 1);

```

```

54     cur.resize(n + 1);
55     pre.resize(n + 1);
56     st.resize(n + 1);
57 }
58
59 void addedge(int a, int b, int c)
60 {
61     e.push_back(b), ne.push_back(h[a]), f.push_back(c), h[a] = e.size() - 1;
62     e.push_back(a), ne.push_back(h[b]), f.push_back(0), h[b] = e.size() - 1;
63 }
64
65 bool bfs()
66 {
67     queue<int> q;
68     st.assign(n + 1, false);
69     q.push(S);
70     st[S] = true;
71     d[S] = 0x3f3f3f3f;
72     while (q.size()) {
73         int u = q.front();
74         q.pop();
75         for (int i = h[u]; i != -1; i = ne[i]) {
76             int v = e[i];
77             if (!st[v] && f[i]) {
78                 st[v] = true;
79                 d[v] = min(d[u], f[i]);
80                 pre[v] = i;
81                 if (v == T) return true;
82                 q.push(v);
83             }
84         }
85     }
86     return false;
87 }
88
89 int maxflow()
90 {
91     int r = 0;
92     while (bfs()) {
93         r += d[T];
94         for (int i = T; i != S; i = e[pre[i ^ 1]]) {
95             f[pre[i]] -= d[T];
96             f[pre[i ^ 1]] += d[T];
97         }
98     }
99     return r;
100 }
101 };

```

4.2.2 Dinic

```

1 struct Dinic {

```



```

2   const int n, S, T;
3   vector<int> h, e, ne, cur, f, d;
4
5   Dinic(int n, int S, int T) : n(n), S(S), T(T) // 点数, 源点, 汇点
6   {
7       h.resize(n + 1, -1);
8       d.resize(n + 1);
9       cur.resize(n + 1);
10  }
11
12  void addedge(int a, int b, int c)
13  {
14      e.push_back(b), ne.push_back(h[a]), f.push_back(c), h[a] = e.size() - 1;
15      e.push_back(a), ne.push_back(h[b]), f.push_back(0), h[b] = e.size() - 1;
16  }
17
18  bool bfs()
19  {
20      d.assign(n + 1, -1);
21      queue<int> q;
22      q.push(S);
23      d[S] = 0;
24      cur[S] = h[S];
25      while (q.size()) {
26          int u = q.front();
27          q.pop();
28          for (int i = h[u]; ~i; i = ne[i]) {
29              int v = e[i];
30              if (d[v] == -1 && f[i] > 0) {
31                  d[v] = d[u] + 1;
32                  cur[v] = h[v];
33                  if (v == T) return true;
34                  q.push(v);
35              }
36          }
37      }
38      return false;
39  }
40
41  int find(int u, int limit)
42  {
43      if (u == T) return limit;
44      int flow = 0;
45      for (int i = cur[u]; ~i; i = ne[i]) {
46          cur[u] = i;
47          int v = e[i];
48          if (d[v] == d[u] + 1 && f[i]) {
49              int t = find(v, min(limit - flow, f[i]));
50              if (!t) d[v] = -1; // important!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
51              f[i] -= t, f[i ^ 1] += t;
52              flow += t;
53          }
54      }

```

```

55     return flow;
56 }
57
58 int dinic()
59 {
60     int r = 0, flow;
61     while (bfs()) {
62         while (flow = find(S, 0x3f3f3f3f)) {
63             r += flow;
64         }
65     }
66     return r;
67 }
68 };

```

4.2.3 MCMF

```

1 struct MCMF {
2     vector<long long> ne, h, e, f, w, dis, incf, vis, pre;
3     int idx, n, S, T;
4     MCMF(int n, int m, int s, int t) // 点数, 边数
5     {
6         this->S = s, this->T = t, this->n = n, this->idx = 0;
7         ne.resize(m, 0);
8         e.resize(m, 0);
9         f.resize(m, 0);
10        w.resize(m, 0);
11        pre.resize(m, 0);
12        vis.resize(n + 1, 0);
13        h.resize(n + 1, -1);
14    };
15
16    void addedge(int a, int b, int c, int d)
17    {
18        int& idx = this->idx;
19        ne[idx] = h[a], e[idx] = b, f[idx] = c, w[idx] = d, h[a] = idx++;
20        ne[idx] = h[b], e[idx] = a, f[idx] = 0, w[idx] = -d, h[b] = idx++;
21    }
22
23    bool spfa()
24    {
25        queue<int> q;
26        dis.assign(n + 1, 0x3f3f3f3f);
27        incf.assign(n + 1, 0);
28        q.push(S);
29        dis[S] = 0, incf[S] = 0x3f3f3f3f;
30        while (q.size()) {
31            int u = q.front();
32            q.pop(), vis[u] = 0;
33            for (int i = h[u]; ~i; i = ne[i]) {
34                int v = e[i];
35                if (f[i] && dis[v] > dis[u] + w[i]) {

```

```

36         dis[v] = dis[u] + w[i];
37         pre[v] = i;
38         incf[v] = min(f[i], incf[u]);
39         if (vis[v] == 0) {
40             q.push(v), vis[v] = 1;
41
42         }
43     }
44 }
45 }
46 return incf[T] > 0;
47 }
48
49 pair<long long, long long> EK()
50 {
51     long long flow = 0, cost = 0;
52     while (spfa()) {
53         int t = incf[T];
54         flow += t, cost += t * dis[T];
55         for (int i = T; i != S; i = e[pre[i] ^ 1]) {
56             f[pre[i]] -= t;
57             f[pre[i] ^ 1] += t;
58         }
59     }
60     return make_pair(flow, cost);
61 }
62 };

```

4.3 tarjan

4.3.1 tarjan 求割点

low: 最多经过一条后向边能追溯到的最小树中结点编号。

一个顶点 u 是割点，当且仅当满足 (1) 或 (2)：

1. u 为树根，且 u 有多于一个子树。因为无向图 *DFS* 搜索树中不存在横叉边，所以若有多个子树，这些子树间不会有边相连。

2. u 不为树根，且满足存在 (u, v) 为树枝边（即 u 为 v 在搜索树中的父亲），并使得 $DFN(u) \leq Low(v)$ 。（因为删去 u 后 v 以及 v 的子树不能到达 u 的其他子树以及祖先）

```

1  int dfn[N], low[N], tim, vis[N], flag[N];
2  int ans;
3  vector<int> G[N];
4
5  void dfs(int u, int fa)
6  {
7      dfn[u] = low[u] = ++tim;
8      vis[u] = 1;
9      int children = 0;
10     for (int v : G[u]) {
11         if (!vis[v]) {
12             children++;

```

```

13     dfs(v, u);
14     low[u] = min(low[u], low[v]);
15     if (u != fa && low[v] >= dfn[u] && !flag[u]) {
16         flag[u] = 1;
17         ans++;
18     }
19 }
20 else if (v != fa) {
21     low[u] = min(low[u], dfn[v]);
22 }
23 }
24 if (u == fa && children >= 2 && !flag[u]) {
25     flag[u] = 1;
26     ans++;
27 }
28 }

```

4.3.2 tarjan 求强连通分量

low: 最多经过一条后向边或栈中横插边所能到达的栈中的最小编号。

```

1  stack<int> st;
2  int in_stack[N];
3
4  void tarjan(int u)
5  {
6      low[u] = dfn[u] = ++idx;
7      st.push(u);
8      in_stack[u] = 1;
9      for (int v : G[u]) {
10         if (!dfn[v]) {
11             tarjan(v);
12             low[u] = min(low[u], low[v]);
13         }
14         else if (in_stack[v]) {
15             low[u] = min(low[u], dfn[v]);
16         }
17     }
18     if (low[u] == dfn[u]) {
19         ++sc;
20         while (st.top() != u) {
21             scc[st.top()] = sc;
22             in_stack[st.top()] = 0;
23             st.pop();
24             sz[sc]++;
25         }
26         scc[st.top()] = sc;
27         in_stack[st.top()] = 0;
28         st.pop();
29         sz[sc]++;
30     }
31 }

```

4.3.3 tarjan 求点双连通分量 (圆方树)

```

1 vector<int> G[N]; // 原图
2 vector<int> T[N]; // 新图 (圆方树)
3 void tarjan(int u, int fa)
4 {
5     int son = 0;
6     dfn[u] = low[u] = ++tim;
7     st.push(u);
8     for (int v : G[u]) {
9         if (!dfn[v]) {
10             son++;
11             tarjan(v, u);
12             low[u] = min(low[u], low[v]);
13             if (low[v] >= dfn[u]) {
14                 scc++;
15                 while (st.top() != v) {
16                     ans[scc].push_back(st.top());
17                     T[scc].push_back(st.top());
18                     T[st.top()].push_back(scc);
19                     st.pop();
20                 }
21                 ans[scc].push_back(st.top());
22                 T[scc].push_back(st.top());
23                 T[st.top()].push_back(scc);
24                 st.pop();
25                 ans[scc].push_back(u);
26                 T[scc].push_back(u);
27                 T[u].push_back(scc);
28             }
29         }
30         else if (v != fa) // 返祖边
31         {
32             low[u] = min(low[u], dfn[v]);
33         }
34     }
35     if (fa == 0 && son == 0) ans[++scc].push_back(u); // 特判孤立点
36 }

```

4.3.4 tarjan 求边双连通分量

```

1 void tarjan(int u, int fa)
2 {
3     dfn[u] = low[u] = ++tim;
4     int son = 0;
5     for (int v : G[u]) {
6         if (!dfn[v]) {
7             son++;
8             tarjan2(v, u);
9             low[u] = min(low[u], low[v]);
10            if (low[v] > dfn[u]) // 找割边
11            {

```

```

12         cnt_bridge++;
13         es[mp[hh(u, v)]] .tag = 1;
14     }
15 }
16 else if (v != fa) {
17     low[u] = min(low[u], dfn[v]);
18 }
19 }
20 }

```

4.4 树上问题

4.4.1 树的直径

```

1 void dfs(int u, int fa) // 树形dp法 一个数组实现 存的是到子树中的最长路径
2 {
3     for (int v : G[u]) {
4         if (v == fa) continue;
5         dfs(v, u);
6         diameter = max(diameter, d1[u] + d1[v] + 1);
7         d1[u] = max(d1[u], d1[v] + 1);
8     }
9 }

```

4.4.2 树的重心

```

1 vector<int> centroid;
2 int sz[N], weight[N]; // weight记录子树大小最大值
3
4 void dfs(int u, int fa)
5 {
6     sz[u] = 1, weight[u] = 0;
7     for (auto v : G[u]) {
8         if (v == fa) continue;
9         dfs(v, u);
10        sz[u] += sz[v];
11        weight[u] = max(weight[u], sz[v]);
12    }
13    weight[u] = max(n - weight[u], weight[u]);
14    if (weight[u] <= n / 2) // 所有子树大小都不超过n/2
15    {
16        centroid.push_back(u);
17    }
18 }

```

4.4.3 最近公共祖先 (倍增)

```

1 struct LCA {
2     vector<vector<int>>> G;
3     vector<vector<int>>> fa;
4     vector<int> dep;

```

```

5   LCA(int n)
6   {
7       G.resize(n + 1);
8       fa.resize(n + 1, vector<int>(31, 0));
9       dep.resize(n + 1, 0);
10  }
11
12  void dfs(int u, int f)
13  {
14      fa[u][0] = f;
15      dep[u] = dep[f] + 1;
16      for (int i = 1; i < 31; i++) {
17          fa[u][i] = fa[fa[u][i - 1]][i - 1];
18      }
19      for (int v : e[u]) {
20          if (v != f) {
21              dfs(v, u);
22          }
23      }
24  }
25
26  int lca(int x, int y)
27  {
28      if (dep[x] < dep[y]) {
29          swap(x, y);
30      }
31      int d = dep[x] - dep[y];
32      for (int i = 0; (1 << i) <= d; i++) {
33          if ((d >> i) & 1) x = fa[x][i];
34      }
35      if (x == y) return x;
36      for (int i = log2(dep[y]); i >= 0; i--) {
37          if (fa[x][i] != fa[y][i]) {
38              x = fa[x][i];
39              y = fa[y][i];
40          }
41      }
42      return fa[x][0];
43  }
44
45  int dist(int x, int y) { return dep[x] + dep[y] - 2 * dep[lca(x, y)]; }
46  };

```

4.4.4 树链剖分

```

1  struct HLD {
2      int n, idx;
3      vector<vector<int>>> G;
4      vector<int> sz, dep, top, son, parent;
5      HLD(int n)
6      {
7          this->n = n;

```

```

8      G.resize(n + 1);
9      sz.resize(n + 1);
10     dep.resize(n + 1);
11     top.resize(n + 1);
12     son.resize(n + 1);
13     parent.resize(n + 1);
14 }
15 void dfs1(int u) // 处理出深度和重儿子
16 {
17     sz[u] = 1;
18     dep[u] = dep[parent[u]] + 1;
19     for (auto v : G[u]) {
20         if (v == parent[u]) continue;
21         parent[v] = u;
22         dfs1(v);
23         sz[u] += sz[v];
24         if (sz[v] > sz[son[u]]) son[u] = v;
25     }
26 }
27 void dfs2(int u, int up)
28 {
29     top[u] = up;
30     if (son[u]) dfs2(son[u], up);
31     for (int v : G[u]) {
32         if (v == parent[u] || v == son[u]) continue;
33         dfs2(v, v);
34     }
35 }
36 int lca(int x, int y)
37 {
38     while (top[x] != top[y]) {
39         if (dep[top[x]] > dep[top[y]]) {
40             x = parent[top[x]];
41         }
42         else {
43             y = parent[top[y]];
44         }
45     }
46     return dep[x] < dep[y] ? x : y;
47 }
48 int calc(int x, int y) { return dep[x] + dep[y] - 2 * dep[lca(x, y)]; } // 查询两
    点距离
49 };

```

4.5 拓扑排序

```

1 bool topsort()
2 {
3     int hh = 0, tt = -1;
4     // d[i] 存储点i的入度
5     for (int i = 1; i <= n; i++)
6         if (!d[i]) q[++tt] = i;

```



```

7   while (hh <= tt) {
8       int u = q[hh++];
9
10      for (int v : G[u]) {
11          if (--d[v] == 0) q[++tt] = v;
12      }
13  }
14  // 如果所有点都入队了, 说明存在拓扑序列; 否则不存在拓扑序列。
15  return tt == n - 1;
16  }

```

4.6 染色法判断二分图

```

1  int n; // n表示点数
2  int color[N]; // 表示每个点的颜色, -1表示为染色, 0表示白色, 1表示黑色
3
4  // 参数: u表示当前节点, c表示当前点的颜色
5  bool dfs(int u, int c)
6  {
7      color[u] = c;
8      for (int v : G[u]) {
9          if (color[v] == -1) {
10             if (!dfs(v, !c)) return false;
11         }
12         else if (color[v] == c)
13             return false;
14     }
15
16     return true;
17 }
18
19 bool check()
20 {
21     memset(color, -1, sizeof color);
22     bool flag = true;
23     for (int i = 1; i <= n; i++)
24         if (color[i] == -1)
25             if (!dfs(i, 0)) {
26                 flag = false;
27                 break;
28             }
29     return flag;
30 }

```

4.7 匈牙利算法求最大匹配

```

1  int n1, n2; // n1表示第一个集合中的点数, n2表示第二个集合中的点数
2  vector<int> G[N]; // 匈牙利算法中只会用到从第二个集合指向第一个集合的边, 所以这里只用存一个方向的边
3  int match[N]; // 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
4  bool st[N]; // 表示第二个集合中的每个点是否已经被遍历过

```

```

5
6 bool find(int u)
7 {
8     for (int v : G[u]) {
9         if (!st[v]) {
10             st[v] = true;
11             if (match[v] == 0 || find(match[v])) {
12                 match[v] = u;
13                 return true;
14             }
15         }
16     }
17     return false;
18 }
19
20 int res = 0;
21 for (int i = 1; i <= n1; i++) // 求最大匹配数，依次枚举第一个集合中的每个点能否匹配第二个
    集合中的点
22 {
23     memset(st, false, sizeof st);
24     if (find(i)) res++;
25 }

```

4.8 差分约束

求解差分约束系统，有 m 条约束条件，每条都为形如 $x_a - x_b \geq c_k$ ， $x_a - x_b \leq c_k$ 或 $x_a = x_b$ 的形式，判断该差分约束系统有没有解，如果有解，求出一组解。

题意	转化	连边
$x_a - x_b \geq c$	$x_b - x_a \leq -c$	add(a, b, -c);
$x_a - x_b \leq c$	$x_a - x_b \leq c$	add(b, a, c);
$x_a = x_b$	$x_a - x_b \leq 0, x_b - x_a \leq 0$	add(b, a, 0), add(a, b, 0);

若要求出一组解，则每个点到源点的最短路即为解。具体过程见参考代码

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1e6 + 10;
4 int n, m;
5 int h[N], e[N], ne[N], we[N], idx;
6 int dis[N], cnt[N], vis[N];
7
8 struct node {
9     int v, w;
10 };
11
12 vector<node> G[N];
13
14 void add(int u, int v, int w) { ne[++idx] = h[u], h[u] = idx, e[idx] = v, we[idx] =
    w; }
15
16 bool spfa(int num, int s)
17 {

```

```

18     memset(dis, 0x3f, sizeof(dis));
19     dis[s] = 0;
20     vis[s] = 1;
21     queue<int> q;
22     q.push(s);
23     while (q.size()) {
24         int u = q.front();
25         q.pop();
26         vis[u] = 0;
27         for (auto it : G[u]) {
28             int v = it.v, w = it.w;
29             if (dis[v] > dis[u] + w) {
30                 dis[v] = dis[u] + w;
31                 cnt[v] = cnt[u] + 1; // 记录经过了多少点
32                 if (cnt[v] > num) // 存在负环
33                     return false;
34                 if (!vis[v]) {
35                     q.push(v);
36                     vis[v] = 1;
37                 }
38             }
39         }
40     }
41     return true;
42 }
43
44 int main()
45 {
46     ios::sync_with_stdio(false);
47     cin.tie(0);
48     cin >> n >> m;
49     for (int i = 1; i <= n; i++) {
50         for (int j = 1; j <= m; j++) {
51             char c;
52             cin >> c;
53             if (c == '>') {
54                 G[i].push_back({n + j, -1});
55             }
56             else if (c == '<') {
57                 G[n + j].push_back({i, -1});
58             }
59             else {
60                 G[i].push_back({n + j, 0});
61                 G[n + j].push_back({i, 0});
62             }
63         }
64     }
65     for (int i = 1; i <= n + m; i++) {
66         G[0].push_back({i, 0});
67     }
68     if (spfa(n + m + 1, 0)) {
69         cout << "Yes" << endl;
70         int minn = 0x3f3f3f3f, d = 0;

```

```

71     for (int i = 1; i <= n + m; i++) {
72         minn = min(minn, dis[i]);
73     }
74     if (minn <= 0) {
75         d = -minn + 1;
76     }
77     for (int i = 1; i <= n; i++) {
78         cout << dis[i] + d << " \n"[i == n];
79     }
80     cout << dis[n] + d << endl;
81     for (int i = 1; i <= m; i++) {
82         cout << dis[n + i] + d << " \n"[i == m];
83     }
84     cout << dis[n + m] + d;
85 }
86 else
87     cout << "No";
88 return 0;
89 }

```

5 数学

5.1 试除法分解质因数

```

1 void divide(int x)
2 {
3     for (int i = 2; i <= x / i; i++)
4         if (x % i == 0)
5             {
6                 int s = 0;
7                 while (x % i == 0) x /= i, s++;
8                 cout << i << ' ' << s << endl;
9             }
10    if (x > 1) cout << x << ' ' << 1 << endl;
11    cout << endl;
12 }

```

5.2 欧拉筛

```

1 vector<ll> primes; // primes[]存储所有素数
2
3 void get_primes(int n)
4 {
5     vector<bool> st(n + 1); // st[x]存储x是否被筛掉
6     for (int i = 2; i <= n; i++) {
7         if (!st[i]) primes.push_back(i);
8         for (int j = 0; j < primes.size() && primes[j] <= n / i; j++) {
9             st[primes[j] * i] = true;
10            if (i % primes[j] == 0) break;
11        }
12    }

```

13 }

5.3 欧拉函数和欧拉定理

5.3.1 欧拉函数

欧拉函数定义：1 到 N 中与 N 互质数的个数称为欧拉函数，即 $\varphi(n) = \sum_{i=1}^n [\gcd(n, i) = 1]$ ，记作 $\varphi(N)$ 。

$$\varphi(x) = x \prod_{i=1}^n \left(1 - \frac{1}{p_i}\right)$$

求解单个数的欧拉函数直接质因数分解

欧拉定理：若 a 与 m 互质，则 $a^{\varphi(m)} \equiv 1 \pmod{m}$ ，变式 $a^b \equiv a^{b \% \varphi(m)} \pmod{m}$ 。

扩展欧拉定理：若 a 与 m 不互质，则 $a^b \equiv a^{b \% \varphi(m) + \varphi(m)} \pmod{m}$ 。

$$a^c \equiv \begin{cases} a^{c \bmod \varphi(m)} & \gcd(a, m) = 1 \\ a^c & \gcd(a, m) \neq 1, c < \varphi(m) \\ a^{c \bmod \varphi(m) + \varphi(m)} & \gcd(a, m) \neq 1, c \geq \varphi(m) \end{cases}$$

```

1 int phi(int n) // 求解 phi(n)
2 {
3     int ans = n;
4     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
5         // 和 sqrt 超时风险
6         if (n % i == 0) {
7             ans = ans / i * (i - 1);
8             while (n % i == 0) n /= i;
9         }
10    }
11    if (n > 1) ans = ans / n * (n - 1); // 特判 n 为质数的情况
12    return ans;
13 }
```

5.3.2 线性筛欧拉函数

```

1 int phi[N], st[N];
2 vector<int>primes;
3 void sieve(int n)
4 {
5     st[1] = phi[1] = 1;
6     for (int i = 2; i <= n; i++) {
7         if (!st[i]) primes.push_back(i), phi[i] = i - 1;
8         for (int j = 1; j < primes.size() && i * primes[j] <= n; j++) {
9             st[i * primes[j]] = 1;
10            if (i % primes[j])
11                phi[i * primes[j]] = phi[i] * phi[primes[j]];
12            else {
13                phi[i * primes[j]] = phi[i] * primes[j];
14            }
15        }
16    }
17 }
```

```

14         break;
15     }
16 }
17 }
18 }

```

5.4 莫比乌斯函数

设 $n = \prod_{i=1}^m p_i^{c_i}$, 则

$$\mu(n) = \begin{cases} 1 & n = 1 \\ (-1)^m \prod_{i=1}^m c_i = 1 (\text{则 } c_1 = c_2 = \cdots = c_m = 1) & \\ 0 & \text{otherwise} \end{cases}$$

```

1  int mu[N], st[N];
2  vector<int>primes;
3  void sieve(int n)
4  {
5      st[1] = mu[1] = 1;
6      for (int i = 2; i <= n; i++) {
7          if (!st[i]) primes.push_back(i), mu[i] = -1;
8          for (int j = 1; j < primes.size() && i * primes[j] <= n; j++) {
9              st[i * primes[j]] = 1;
10             if (i % primes[j])
11                 mu[i * primes[j]] = -mu[i];
12             else {
13                 mu[i * primes[j]] = 0;
14                 break;
15             }
16         }
17     }
18 }

```

5.5 线性筛约数个数

记 d_i 为 i 的约数个数, $d(i) = \prod_{k=1}^i (a_i + 1)$ 维护每一个数的最小值因子出现的次数 (即 a_1) 即可。

```

1  int d[N], a[N], st[N];
2  vector<int>primes;
3  void sieve(int n)
4  {
5      st[1] = d[1] = 1;
6      for (int i = 2; i <= n; i++) {
7          if (!st[i]) primes.push_back(i), d[i] = 2, a[i] = 1;
8          for (int j = 1; j < primes.size() && i * primes[j] <= n; j++) {
9              st[i * primes[j]] = 1;
10             if (i % primes[j])

```

```

11         d[i * primes[j]] = d[i] * d[primes[j]], a[i * primes[j]] = 1;
12     else {
13         d[i * primes[j]] = d[i] / (a[i] + 1) * (a[i] + 2);
14         a[i * primes[j]] = a[i] + 1;
15         break;
16     }
17 }
18 }
19 }

```

5.6 线性筛约数和

记 $\sigma(i)$ 表示 i 的约数和

$$\sigma(i) = \prod_{k=1}^i \left(\sum_{a_i=0}^{p_i} p_i^i \right)$$

维护 $\text{low}(i)$ 表示 i 的最小质因子的指数次幂, 即 $p_1^{a_1}$, $\text{sum}(i)$ 表示 i 的最小质因子对答案的贡献, 即 $\sum_{a_1=0}^{p_1} p_1^{a_1}$ 。可能会爆 `int`。

```

1  int low[N], sum[N], sigma[N], st[N];
2  vector<int> primes;
3  void sieve(int n)
4  {
5      st[1] = low[1] = sum[1] = sigma[1] = 1;
6      for (int i = 2; i <= n; i++) {
7          if (!st[i]) primes.push_back(i), low[i] = i, sum[i] = sigma[i] = i + 1;
8          for (int j = 1; j < primes.size() && i * primes[j] <= n; j++) {
9              st[i * primes[j]] = 1;
10             if (i % primes[j] == 0) {
11                 low[i * primes[j]] = low[i] * primes[j];
12                 sum[i * primes[j]] = sum[i] + low[i * primes[j]];
13                 sigma[i * primes[j]] = sigma[i] / sum[i] * sum[i * primes[j]];
14                 break;
15             }
16             low[i * primes[j]] = primes[j];
17             sum[i * primes[j]] = primes[j] + 1;
18             sigma[i * primes[j]] = sigma[i] * sigma[primes[j]];
19         }
20     }
21 }

```

5.7 裴蜀定理

如果 a, b 均为整数, 一定存在整数 x, y 使得 $ax + by = \gcd(a, b)$ 成立。

推论: 对于方程 $ax + by = c$, 如果 $\gcd(a, b) \mid c$, 则方程一定有解, 反之一定无解。

5.8 扩展欧几里得算法

求得的是 $ax + by = \gcd(a, b)$ 的一组特解，该方程的通解可以表示为

$$\begin{cases} x' = x + k \frac{b}{\gcd(a, b)} \\ y' = y - k \frac{a}{\gcd(a, b)} \end{cases} \quad (k \in \mathbb{Z})$$

```

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

```

5.9 快速幂

```

1 int qpow(int x, int k, int Mod) {
2     int res = 1;
3     while(k) {
4         if(k & 1)
5             res = 1ll * res * x % Mod;
6         x = 1ll * x * x % Mod;
7         k >>= 1;
8     }
9     return res;
10 }

```

5.10 BSGS 离散对数

```

1 ll BSGS(ll x, ll y, ll mod) // y是x的多少次方 (模意义下)
2 {
3     x %= mod, y %= mod;
4     if (x == 0) return y == 0 ? 1 : -1;
5     if (y == 1) return 0;
6     ll m = ceil(sqrt(mod));
7     ll t = y;
8     unordered_map<ll, ll> mp;
9     mp[t] = 0;
10    for (int i = 1; i < m; i++) {
11        t = (t * x) % mod;
12        mp[t] = i;
13    }
14    ll res = 1;
15    t = 1;
16    for (int i = 1; i <= m; i++) res = (res * x) % mod;
17    for (int i = 1; i <= m; i++) {

```



```

18     t = (res * t) % mod;
19     if (mp.count(t)) {
20         return (m * i - mp[t]) % mod;
21     }
22 }
23 return -1;
24 }

```

5.11 乘法逆元

- 当 mod 为质数时, $ax \equiv 1 \pmod{b}$ 由费马小定理有 $ax \equiv a^{b-1} \pmod{b}$, $\therefore x \equiv a^{b-2} \pmod{b}$, 快速幂求 a^{b-2} 即可
- 扩展欧几里得方法 (要求 $\text{gcd}(a, b) = 1$) : 等价于求 $ax \equiv 1 \pmod{p}$ 的解, 可以写为 $ax + pk = 1$, 求解 x, k 即可。

```

1 void exgcd(ll a, ll b, ll& x, ll& y)
2 {
3     if (b == 0) {
4         x = 1;
5         y = 0;
6         return;
7     }
8     exgcd(b, a % b, x, y);
9     ll tmp = x;
10    x = y;
11    y = tmp - a / b * y;
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 {
15     ll x, y, d = exgcd(a, mod, x, y);
16     return d == 1 ? (x % mod + mod) % mod : -1;
17 }

```

5.12 快速递推求逆元

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;

```

5.13 中国剩余定理

$$x \bmod a_i = b_i$$

```

1 ll CRT(int n, ll a[], ll b[]) // a为模数, b为余数
2 {
3     ll ans = 0;
4     ll s = 1;

```

```

5   for (int i = 1; i <= n; i++) {
6       s *= a[i];
7   }
8   for (int i = 1; i <= n; i++) {
9       ll m = s / a[i];
10      ll x, y;
11      exgcd(m, a[i], x, y); // x为m在模a[i]下的逆元
12      ans = (ans + (m * x * b[i]) % s + s) % s; // m*x不要对a[i]取模
13  }
14  return ans;
15  }

```

5.14 高斯消元

求解模意义下 n 元线性方程组

```

1   std::cin >> n >> p;
2   for (int i = 1; i <= n; i++) { // 读入增广矩阵
3       for (int j = 1; j <= n + 1; j++) {
4           std::cin >> a[i][j];
5           a[i][j] %= p;
6       }
7   }
8   int cnt = 1;
9   for (int i = 1; i <= n; i++) {
10      int r = cnt;
11      for (int j = cnt; j <= n; j++) {
12          if (abs(a[j][i]) > abs(a[r][i])) {
13              r = j;
14          }
15      }
16      if (a[r][i] == 0) continue;
17      if (r != cnt) std::swap(a[cnt], a[r]);
18      ll inv = qpow(a[cnt][i], p - 2);
19      for (int j = n + 1; j >= i; j--) {
20          a[cnt][j] = (a[cnt][j] * inv) % p;
21      }
22      for (int j = cnt + 1; j <= n; j++) {
23          if (a[j][i]) {
24              for (int k = n + 1; k >= i; k--) {
25                  a[j][k] = (a[j][k] - (a[j][i] * a[cnt][k]) % p + p) % p;
26              }
27          }
28      }
29      cnt++;
30  }
31  if (cnt < n + 1) {
32      for (int i = cnt; i <= n; i++) {
33          if (a[i][n + 1]) {
34              std::cout << -1 << "\n"; // 无解
35              return 0;
36          }

```

```

37     }
38     std::cout << 0 << "\n"; // 多解
39     return 0;
40 }
41 for (int i = n; i >= 1; i--) {
42     for (int j = i + 1; j <= n; j++) {
43         a[i][n + 1] = (a[i][n + 1] - (a[i][j] * a[j][n + 1]) % p + p) % p;
44     }
45 }
46 for (int i = 1; i <= n; i++) {
47     std::cout << "x" << i << "=" << a[i][n + 1] << "\n";
48 }
49 return 0;

```

5.15 FFT

5.15.1 FFT 递归版

```

1 void FFT(complex<double> *A, int limit, int op)
2 {
3     if (limit == 1)
4         return;
5     complex<double> A1[limit / 2], A2[limit / 2];
6     for (int i = 0; i < limit / 2; i++)
7     {
8         A1[i] = A[i * 2], A2[i] = A[i * 2 + 1];
9     }
10    FFT(A1, limit / 2, op), FFT(A2, limit / 2, op);
11    complex<double> w1 ({cos(2 * pi / limit), sin(2 * pi / limit) * op});
12    complex<double> wk({1, 0});
13    for (int i = 0; i < limit / 2; i++)
14    {
15        A[i] = A1[i] + A2[i] * wk;
16        A[i + limit / 2] = A1[i] - A2[i] * wk;
17        wk = wk * w1;
18    }
19 }

```

5.15.2 FFT 迭代版

```

1 void change(complex<double> *A, int len)
2 {
3     for (int i = 0; i < len; i++)
4         R[i] = R[i / 2] / 2 + ((i & 1) ? len / 2 : 0);
5     for (int i = 0; i < len; i++)
6         if (i < R[i]) swap(A[i], A[R[i]]);
7 }
8
9 void FFT(complex<double> *A, int limit, int op)
10 {
11     change(A, limit);
12     for (int k = 2; k <= limit; k <<= 1)

```

```

13 {
14     complex<double> w1 ({cos(2 * pi / k), sin(2 * pi / k) * op});
15     for (int i = 0; i < limit; i += k)
16     {
17         complex<double> wk({1, 0});
18         for (int j = 0; j < k / 2; j++)
19         {
20             complex<double> x = A[i + j], y = A[i + j + k / 2] * wk;
21             A[i + j] = x + y;
22             A[i + j + k / 2] = x - y;
23             wk = wk * w1;
24         }
25     }
26 }
27 }

```

5.16 线性基

5.16.1 高斯消元法求线性基

高斯消元法构造线性基特点：

- 从大到小排列
- 各个基的高位没有重复的 1

```

1 int n, k;
2 ll a[N];
3
4 void gauss()
5 {
6     for (int i = 63; i >= 0; i--) {
7         for (int j = k; j < n; j++) {
8             if (a[j] >> i & 1) {
9                 swap(a[i], a[j]);
10                break;
11            }
12        }
13        if ((a[i] >> i & 1) == 0) continue;
14        for (int j = 0; j < n; j++) {
15            if (j != i && (a[j] >> i & 1)) a[j] ^= a[i];
16        }
17        k++;
18        if (k == n) break;
19    }
20 }

```

5.16.2 贪心法求线性基

贪心法构造线性基的特点：

- 从大到小排列

- 各个基的高位可能存在重复的 1
- 线性基不是唯一的，与原集合即插入顺序有关

```

1  int n;
2  ll a[N];
3
4  void insert(ll x)
5  {
6      for (int i = 63; i >= 0; i--) {
7          if ((x >> i) & 1) {
8              if (a[i])
9                  x ^= a[i];
10             else {
11                 a[i] = x;
12                 break;
13             }
14         }
15     }
16 }

```

5.17 杜教筛

杜教筛被用于处理一类数论函数的前缀和问题。求解 $S(n) = \sum_{i=1}^n f(i)$ 的问题，其中 $f(i)$ 是一个数论函数，杜教筛可以在 $O(n^{2/3})$ 的时间复杂度内解决。找一个数论函数 g

$$\begin{aligned}
 \sum_{i=1}^n (f * g)(i) &= \sum_{i=1}^n \sum_{d|i} g(d) f\left(\frac{i}{d}\right) \\
 &= \sum_{i=1}^n \sum_{j=1}^{\lfloor n/i \rfloor} g(i) f(j) \\
 &= \sum_{i=1}^n g(i) \sum_{j=1}^{\lfloor n/i \rfloor} f(j) \\
 &= \sum_{i=1}^n g(i) S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)
 \end{aligned}$$

则可以得到递推式：

$$g(1)S(n) = \sum_{i=1}^n (f * g)(i) - \sum_{i=2}^n g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

g 需要满足：

- 可以快速计算 $\sum_{i=1}^n (f * g)(i)$
- 可以快速计算 g 的前缀和，以用数论分块求解 $\sum_{i=2}^n g(i)S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$

时间复杂度： $O(S + \frac{n}{\sqrt{S}})$ ，其中 S 是预处理的 g 的前缀和的规模。取 $S = n^{2/3}$ ，时间复杂度最小为 $O(n^{2/3})$ 。

例如，利用杜教筛求 $\sum_{i=1}^n \mu(i)$ 的前缀和：利用 $\mu * \mathbf{1} = \varepsilon$ ，则可以 $g = \mathbf{1}$ ，则有：

$$S(n) = \sum_{i=1}^n \mu * \mathbf{1}(i) - \sum_{i=2}^n \mathbf{1} S\left(\left\lfloor \frac{n}{i} \right\rfloor\right) = 1 - \sum_{i=2}^n S\left(\left\lfloor \frac{n}{i} \right\rfloor\right)$$

```

1 unordered_map<ll, ll> mp;
2
3 ll s(ll n)
4 {
5     if (n < N) return mu[n]; // 注意这里的 mu 数组存的是莫比乌斯函数的前缀和
6     if (mp[n]) return mp[n];
7     ll ans = 1;
8     for (ll l = 2, r; l <= n; l = r + 1) {
9         r = min(n, n / (n / l));
10        ans -= s(n / l) * (r - l + 1);
11    }
12    mp[n] = ans;
13    return mp[n];
14 }
```

5.18 数学常见结论

5.18.1 插板法

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

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

- 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

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

- 球不同，盒同、不能空：第二类斯特林数 $\text{Stirling2}(n, m)$ ，答案为

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

- 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 。

5.18.2 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：

$$\sum_{i=1}^n \sum_{j=i+1}^n (a_i b_j - a_j b_i)^2 = \left(\sum_{i=1}^n a_i \right)^2 \left(\sum_{i=1}^n b_i \right)^2 - \left(\sum_{i=1}^n a_i b_i \right)^2。$$

5.18.3 斯特林数

第二类斯特林数 (斯特林子集数) $\left\{ \begin{matrix} n \\ k \end{matrix} \right\}$, 也可记做 $S(n, k)$, 表示将 n 个两两不同的元素, 划分为 k 个互不区分的非空子集的方案数。递推式

$$\left\{ \begin{matrix} n \\ k \end{matrix} \right\} = \left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\} + k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\}$$

边界是 $\left\{ \begin{matrix} n \\ 0 \end{matrix} \right\} = [n=0]$

考虑用组合意义来证明。我们插入一个新元素时, 有两种方案:

- 将新元素单独放入一个子集, 有 $\left\{ \begin{matrix} n-1 \\ k-1 \end{matrix} \right\}$ 种方案;
- 将新元素放入一个现有的非空子集, 有 $k \left\{ \begin{matrix} n-1 \\ k \end{matrix} \right\}$ 种方案。

根据加法原理, 将两式相加即可得到递推式。

第一类斯特林数 (斯特林轮换数) $\left[\begin{matrix} n \\ k \end{matrix} \right]$, 也可记做 $s(n, k)$, 表示将 n 个两两不同的元素, 划分为 k 个互不区分的非空轮换的方案数。

一个轮换就是一个首尾相接的环形排列。我们可以写出一个轮换 $[A, B, C, D]$, 并且我们认为 $[A, B, C, D] = [B, C, D, A] = [C, D, A, B] = [D, A, B, C]$, 即, 两个可以通过旋转而互相得到的轮换是等价的。注意, 我们不认为两个可以通过翻转而相互得到的轮换等价, 即 $[A, B, C, D] \neq [D, C, B, A]$ 。递推式

$$\left[\begin{matrix} n \\ k \end{matrix} \right] = \left[\begin{matrix} n-1 \\ k-1 \end{matrix} \right] + (n-1) \left[\begin{matrix} n-1 \\ k \end{matrix} \right]$$

边界是 $\left[\begin{matrix} n \\ 0 \end{matrix} \right] = [n=0]$ 。

该递推式的证明可以考虑其组合意义。我们插入一个新元素时, 有两种方案:

- 将该新元素置于一个单独的轮换中, 共有 $\left[\begin{matrix} n-1 \\ k-1 \end{matrix} \right]$ 种方案;
- 将该元素插入到任何一个现有的轮换中, 共有 $(n-1) \left[\begin{matrix} n-1 \\ k \end{matrix} \right]$ 种方案。

根据加法原理, 将两式相加即可得到递推式。

5.18.4 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 $2Cat_{n-1}$ 。

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

5.18.5 斐波那契数列

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

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：

- $F_a \mid F_b \Leftrightarrow a \mid b$ ；
- $\gcd(F_a, F_b) = F_{\gcd(a, b)}$ ；

- 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。

6 博弈论

6.1 巴什博弈

6.1.1 朴素巴什博弈

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)。

6.1.2 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)。

6.2 Nim 博弈

6.2.1 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。

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)。

6.2.2 Nim_K

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。

6.2.3 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”) 时;
- 至少有一堆的石头数量大于 1, 且 $Sum_N \neq 0$ 。

6.3 SG 游戏

6.3.1 SG 定理和 SG 函数

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 getsg(int x) {
2      if (sg[x] != -1) return sg[x];
3
4      unordered_set<int> S;
5      for (int v:G[x]) // 可能的转移
6          if(x >= v)
7              S.insert(sg(x - v));
8
9      for (int i = 0; ; ++ i)
10         if (S.count(i) == 0)
11             return sg[x] = i;
12 }

```

6.3.2 反 SG 博弈

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。

7 计算几何

7.1 计算几何

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  #define endl '\n'
4  #define ll long long
5  #define double long double
6  #define PII pair<int, int>
7  #define pi acos(-1.0)
8  const int N = 1e5 + 10;
9
10 #define eps 1e-8
11 inline int sgn(double x) { return fabs(x) < eps ? 0 : (x > 0 ? 1 : -1); } // 判断正
    负
12
13 struct Point {
14     double x, y;
15     Point(double nx = 0, double ny = 0) : x(nx), y(ny) {}
16     void read() { cin >> this->x >> this->y; }
17     inline bool operator<(Point A)
18     {
19         if (sgn(this->x - A.x) == 0) return this->y < A.y - eps;
20         return this->x < A.x - eps;
21     }
22     inline Point operator+(Point A) { return Point(A.x + this->x, A.y + this->y); }
23     inline Point operator-(Point A) { return Point(this->x - A.x, this->y - A.y); }
24     inline Point operator*(double x) { return Point(this->x * x, this->y * x); }
25     inline Point operator/(double x) { return Point(this->x / x, this->y / x); }
26     inline bool operator==(Point A) { return sgn(this->x - A.x) == 0 && sgn(this->y
        - A.y) == 0; }
27     inline double operator*(Point A) { return this->x * A.x + this->y * A.y; } // 点
        乘
28     inline double operator^(Point A) { return this->x * A.y - this->y * A.x; } // 叉
        积 (注意先后顺序)
29     inline double len() { return sqrt((*this) * (*this)); }
30     inline double len2() { return (*this) * (*this); }
31     inline double dis2(Point A) { return (*this - A).len2(); }
32     inline double dis(Point A) { return sqrt(this->dis2(A)); }
33     inline Point unit() { return Point(this->x / this->len(), this->y / this->len())
        ; }
34     inline int toleft(Point A)
35     {
36         double t = (*this) ^ A;
37         return (t > eps) - (t < -eps);
38     }
39     inline Point Normal() { return Point(-this->y / this->len(), this->x / this->len
        ()); } // 与A正交的单位向量
40     inline Point rotate(double rad) { return Point(this->x * cos(rad) - this->y *
        sin(rad), this->x * sin(rad) + this->y * cos(rad)); }
41     inline Point rotate(double cosh, double sinh) { return Point(this->x * cosh -
        this->y * sinh, this->x * sinh + this->y * cosh); }
42 };
43 typedef Point Vector;
44

```

```

45 inline double Len(Vector A) { return sqrt(A * A); }
46 inline double Ang(Vector A, Vector B) { return acos(A * B) / A.len() / B.len(); } //
    两个向量夹角
47 inline double Area(Vector A, Vector B) { return (A ^ B) / 2; } // 两个向量组成的三角
    形的面积
48 inline int toleft(Vector A, Vector B) // toleft测试: B在A左边为1, 右边为-1, 方向相同为
    0
49 {
50     double t = A ^ B;
51     return (t > eps) - (t < -eps);
52 }
53
54 bool argcmp(Point a, Point b)
55 {
56     auto quad = [](Point& a) {
57         if (a.y < -eps) return 1;
58         if (a.y > eps) return 4;
59         if (a.x < -eps) return 5;
60         if (a.x > eps) return 3;
61         return 2;
62     };
63     int qa = quad(a), qb = quad(b);
64     if (qa != qb) return qa < qb;
65     auto t = a ^ b;
66     // if (abs(t) <= eps) return a * a < b * b - eps; // 不同长度的向量需要分开
67     return t > eps;
68 }
69
70 struct Line {
71     Point A, B;
72     Line(Point x = Point(0, 0), Point y = Point(0, 0)) : A(x), B(y) {}
73     void read() { cin >> this->A.x >> this->A.y >> this->B.x >> this->B.y; }
74     double dis(Point p) { return fabs((p - this->A) ^ (this->A - this->B)) / Len(
        this->A - this->B); } // 点到直线的距离
75     Point projection(Point P)
76     {
77         double res = (B - A) * (P - A) / (B - A).len();
78         return A + (B - A) / ((B - A).len()) * res;
79     }
80 };
81
82 struct Seg {
83     Point A, B;
84     Seg(Point x = Point(0, 0), Point y = Point(0, 0)) : A(x), B(y) {}
85     void read() { cin >> this->A.x >> this->A.y >> this->B.x >> this->B.y; }
86     double dis(Point p) // 点到线段距离
87     {
88         if ((p - this->A) * (this->B - this->A) < -eps || (p - this->B) * (this->A -
            this->B) < -eps) return min(Len(p - this->A), Len(p - this->B));
89         return Line{A, B}.dis(p);
90     }
91     int on(Point p) // -1表示在端点, 1表示在线段上, 0表示不在线段上
92     {

```

```

93     if (p == this->A || p == this->B) return -1;
94     return (toleft(p - this->A, p - this->B) == 0 && (p - this->A) * (p - this->B
95         ) < -eps);
96 }
97 };
98 inline bool IsIntersect(Seg S1, Seg S2) // 两个线段是否相交
99 {
100     double f1 = (S1.B - S1.A) ^ (S2.A - S1.A), f2 = (S1.B - S1.A) ^ (S2.B - S1.A);
101     double g1 = (S2.B - S2.A) ^ (S1.A - S2.A), g2 = (S2.B - S2.A) ^ (S1.B - S2.A);
102     return ((f1 < 0) ^ (f2 < 0)) && ((g1 < 0) ^ (g2 < 0));
103 }
104 inline Point LineIntersection(Line L1, Line L2) { return L1.A + (L1.B - L1.A) * ((L2
    .B - L2.A) ^ (L1.A - L2.A)) / ((L1.B - L1.A) ^ (L2.B - L2.A)); } // 两条直线求交
    点
105
106 struct Polygon {
107     vector<Point> p;
108
109     double circ() // 周长
110     {
111         double sum = 0;
112         for (int i = 0; i < p.size(); i++) {
113             sum += Len(p[i] - p[i + 1 == p.size() ? 0 : i + 1]);
114         }
115         return sum;
116     }
117
118     double area() // 逆时针存储时为正
119     {
120         double sum = 0;
121         for (int i = 0; i < p.size(); i++) {
122             sum += p[i] ^ p[(i + 1) % p.size()];
123         }
124         return sum * 0.5;
125     }
126
127     int in_polyon(Point a) // -1表示在边上, 0在多边形外, 1在多边形内
128     {
129         int cnt = 0;
130         for (int i = 0; i < p.size(); i++) {
131             Seg s(p[i], p[i + 1 == p.size() ? 0 : i + 1]);
132             if (s.on(a)) return -1;
133             double d1 = a.y - s.A.y, d2 = a.y - s.B.y;
134             double det = (s.A - a) ^ (s.B - a);
135             if ((sgn(det) >= 0 && sgn(d1) < 0 && sgn(d2) >= 0) || (sgn(det) <= 0 &&
                sgn(d2) < 0 && sgn(d1) >= 0)) cnt++;
136         }
137         return cnt & 1;
138     }
139 };
140
141 struct Convex : Polygon {

```

```

142
143 int in(Point a) // -1表示在边上, 0在凸多边形外, 1在凸多边形内
144 {
145     auto& p = this->p;
146     if (p.size() == 1) return a == p[0] ? -1 : 0;
147     if (p.size() == 2) return Seg{p[0], p[1]}.on(a) ? -1 : 0;
148     if (a == p[0]) return -1;
149     if (toleft(p[1] - p[0], a - p[0]) == -1 || toleft(p.back() - p[0], a - p[0])
        == 1) return 0;
150     auto cmp = [&](Point u, Point v) { return toleft(u - p[0], v - p[0]) == 1; };
151     int l = lower_bound(p.begin() + 1, p.end(), a, cmp) - p.begin();
152     if (l == 1) return Seg{p[0], p[1]}.on(a) ? -1 : 0;
153     if (l == p.size() - 1 && Seg{p[0], p[1]}.on(a)) return -1;
154     if (Seg{p[l - 1], p[l]}.on(a)) return -1;
155     return (toleft(p[l] - p[l - 1], a - p[l - 1]) > 0);
156 }
157
158 Convex operator+(Convex& c) // 闵可夫斯基和
159 {
160     auto& p = this->p;
161     vector<Point> res;
162     vector<Seg> e1(p.size()), e2(c.p.size()), edge(p.size() + c.p.size());
163     res.reserve(p.size() + c.p.size());
164     auto cmp = [](Seg& u, Seg& v) { return argcmp(u.B - u.A, v.B - v.A); };
165     for (int i = 0; i < p.size(); i++) e1[i] = {p[i], p[i + 1 == p.size() ? 0 : i
        + 1]};
166     for (int i = 0; i < c.p.size(); i++) e2[i] = {c.p[i], c.p[i + 1 == c.p.size()
        ? 0 : i + 1]};
167     rotate(e1.begin(), min_element(e1.begin(), e1.end(), cmp), e1.end());
168     rotate(e2.begin(), min_element(e2.begin(), e2.end(), cmp), e2.end());
169     merge(e1.begin(), e1.end(), e2.begin(), e2.end(), edge.begin(), cmp);
170     auto check = [&](Point& u) {
171         auto last = res.back(), last2 = res[res.size() - 2];
172         return toleft(last - last2, u - last) == 0 && (last - last2) * (u - last)
            >= -eps;
173     };
174     auto u = e1[0].A + e2[0].A;
175     for (auto v : edge) {
176         while (res.size() > 1 && check(u)) res.pop_back();
177         res.push_back(u);
178         u = u + v.B - v.A;
179     }
180     if (res.size() > 1 && check(res[0])) res.pop_back();
181     return Convex{res};
182 }
183
184 double diameter()
185 {
186     double ans = 0;
187     auto area = [](Point u, Point v, Point w) -> double { return (w - u) ^ (w - v
        ); }; // 要求逆时针存点
188     for (int i = 0, j = 1; i < p.size(); i++) {
189         int nxt = i + 1 == p.size() ? 0 : i + 1;

```



```

190         while (sgn(area(p[i], p[nxt], p[j]) - area(p[i], p[nxt], p[j + 1] == p.size
           () ? 0 : j + 1])) <= 0) j = j + 1 == p.size() ? 0 : j + 1;
191         ans = max({ans, p[i].dis(p[j]), p[nxt].dis(p[j])});
192     }
193     return ans;
194 }
195 };
196
197 Convex Andrew(vector<Point> p) // Andrew求凸包O(n\log n)
198 {
199     vector<Point> st;
200     if (p.empty()) return Convex{st};
201     sort(p.begin(), p.end(), [](Point& A, Point& B) { return fabs(A.x - B.x) > eps ?
           A.x < B.x : A.y < B.y; });
202
203     auto check = [&](Point cur) {
204         auto last = st.back(), last2 = st[st.size() - 2];
205         return toleft(last - last2, cur - last) < 0; // <=时会跳过经过的点
206     };
207
208     for (auto cur : p) {
209         while (st.size() > 1 && check(cur)) st.pop_back();
210         st.push_back(cur);
211     }
212     int t = st.size();
213     p.pop_back();
214     reverse(p.begin(), p.end());
215     for (auto cur : p) {
216         while (st.size() > t && check(cur)) st.pop_back();
217         st.push_back(cur);
218     }
219     st.pop_back();
220     return Convex{st};
221 }
222
223 double ClosestPair(vector<Point> p) // 平面最近点对 O(n\log n), 调用之前先排序
224 {
225     if (p.size() == 1) return 1e20;
226     if (p.size() == 2) {
227         return p[0].dis(p[1]);
228     }
229     int mid = p.size() / 2;
230     double d = min(ClosestPair(vector<Point>(p.begin(), p.begin() + mid)),
                     ClosestPair(vector<Point>(p.begin() + mid, p.end())));
231     vector<Point> tmp;
232     for (int i = 0; i < p.size(); i++) {
233         if (fabs(p[i].x - p[mid].x) < d + eps) {
234             tmp.push_back(p[i]);
235         }
236     }
237     sort(tmp.begin(), tmp.end(), [&](const Point& a, const Point& b) { return a.y <
           b.y; });
238     for (int i = 0; i < tmp.size(); i++) {

```

```

239     for (int j = i + 1; j < tmp.size() && tmp[j].y - tmp[i].y < d + eps; j++) {
240         d = min(d, tmp[i].dis(tmp[j]));
241     }
242 }
243 return d;
244 }
245
246 struct Circle {
247     Point O;
248     double r;
249
250     bool operator==(const Circle& A) { return this->O == A.O && sgn(this->r - A.r)
        == 0; }
251     double area() { return r * r * pi; }
252     double circ() { return r * pi * 2; }
253
254     int in(Point a) // -1 圆上 | 0 圆外 | 1 圆内
255     {
256         if (a.dis(this->O) > r + eps) return 0;
257         if (sgn(a.dis(this->O) - r) == 0) return -1;
258         return 1;
259     }
260
261     int relation(Circle a) // -1 相同 | 0 相离 | 1 外切 | 2 相交 | 3 内切a | 4 被a内切
        | 5 内含于a | 6 被a内含
262     {
263         double d = a.O.dis(this->O);
264         if (*this == a) return -1;
265         if (d > a.r + this->r + eps) return 0;
266         if (sgn(d - a.r - this->r) == 0) return 1;
267         if (sgn(d - a.r + this->r) == 0) return 3;
268         if (sgn(d + a.r - this->r) == 0) return 4;
269         if (d + this->r < a.r - eps) return 5;
270         if (d + a.r < this->r - eps) return 6;
271         return 2;
272     }
273
274     vector<Point> cross(Line l)
275     {
276         double d = l.dis(this->O);
277         if (d > this->r + eps) return vector<Point>();
278         Point p = l.projection(this->O);
279         if (sgn(d - this->r) == 0) return vector<Point>{p};
280         double k = sqrt(this->r * this->r - d * d);
281         Vector v = (l.A - l.B).unit() * k;
282         return vector<Point>{p + v, p - v};
283     }
284
285     vector<Point> cross(Circle a)
286     {
287         int t = this->relation(a);
288         if (t == -1 || t == 0 || t == 5 || t == 6) return vector<Point>();
289         double d = O.dis(a.O);

```

```

290     Vector dr = (a.O - O).unit() * r;
291     if (t == 1 || t == 4) return vector<Point>{O + dr};
292     if (t == 3) return vector<Point>{O - dr};
293     double costh = (r * r + d * d - a.r * a.r) / (2 * d * r);
294     double sinh = sqrt(1 - costh * costh);
295     return vector<Point>{O + dr.rotate(costh, sinh), O + dr.rotate(costh, -sinh
        )};
296 }
297
298 double area(Circle a)
299 {
300     int t = relation(a);
301     if (t == -1) return area();
302     if (t == 0 || t == 1) return 0;
303     if (t >= 3) return min(area(), a.area());
304     vector<Point> p = cross(a);
305     double d = O.dis(a.O);
306     double costh1 = (r * r + d * d - a.r * a.r) / (2 * r * d), costh2 = (a.r * a.
        r + d * d - r * r) / (2 * a.r * d);
307     double sinh1 = sqrt(1 - costh1 * costh1), sinh2 = sqrt(1 - costh2 * costh2)
        ;
308     double th1 = acos(costh1), th2 = acos(costh2);
309     return r * r * (th1 - costh1 * sinh1) + a.r * a.r * (th2 - costh2 * sinh2);
310 }
311
312 vector<Point> tangent_points(Point a)
313 {
314     int t = this->in(a);
315     if (t == 1) return vector<Point>();
316     double d = O.dis(a);
317     Vector dr = (a - O).unit() * r;
318     if (t == -1) {
319         return vector<Point>{O + dr};
320     }
321     double costh = r / d;
322     double sinh = sqrt(1 - costh * costh);
323     return vector<Point>{O + dr.rotate(costh, sinh), O + dr.rotate(costh, -sinh
        )};
324 }
325
326 vector<Point> tangent_points(Circle a)
327 {
328     int t = relation(a);
329     vector<Point> res;
330     if (t == -1 || t == 5 || t == 6) return res;
331     if (t == 1 || t == 4) {
332         res.push_back(O + (a.O - O).unit() * r);
333     }
334     else if (t == 3) {
335         res.push_back(O - (a.O - O).unit() * r);
336     }
337     double d = O.dis(a.O);
338     Vector e = (a.O - O).unit();

```

```

339     if (t <= 2) {
340         double costh = (r - a.r) / d;
341         double sinth = sqrt(1 - costh * costh);
342         Vector u = e.rotate(costh, sinth), v = e.rotate(costh, -sinth);
343         res.push_back(0 + u * r), res.push_back(0 + u * a.r);
344         res.push_back(0 + v * r), res.push_back(0 + v * a.r);
345     }
346     if (t == 0) {
347         double costh = (r + a.r) / d;
348         double sinth = sqrt(1 - costh * costh);
349         Vector u = e.rotate(costh, sinth), v = e.rotate(costh, -sinth);
350         res.push_back(0 + u * r), res.push_back(0 - u * a.r);
351         res.push_back(0 + v * r), res.push_back(0 - v * a.r);
352     }
353     return res;
354 }
355 };
356
357 Circle excircle(vector<Point> p) // 外接圆(1)(尽量用 long double)(推荐写法)
358 {
359     double s = (p[0] - p[1]) ^ (p[0] - p[2]);
360     Point O;
361     O.x = (p[0].y * (p[1].len2() - p[2].len2()) + p[1].y * (p[2].len2() - p[0].len2()
362         ()) + p[2].y * (p[0].len2() - p[1].len2())) / (-2 * s);
363     O.y = (p[0].x * (p[1].len2() - p[2].len2()) + p[1].x * (p[2].len2() - p[0].len2()
364         ()) + p[2].x * (p[0].len2() - p[1].len2())) / (2 * s);
365     double r = O.dis(p[0]);
366     return Circle{O, r};
367 }
368
369 // Circle excircle(vector<Point> p) // 外接圆(2)(尽量用 long double)
370 // {
371 //     auto get_midline = [&](Point a, Point b) {
372 //         Point mid = (a + b) / 2;
373 //         Vector v = Vector(-(a - b).y, (a - b).x);
374 //         return Line(mid, mid + v);
375 //     };
376 //     Point O = LineIntersection(get_midline(p[0], p[1]), get_midline(p[0], p[2]));
377 //     double r = O.dis(p[0]);
378 //     return Circle{O, r};
379 // }
380
381 Circle incircle(vector<Point> p) // 内切圆(尽量用 long double)
382 {
383     auto get_angmidline = [&](Point O, Point a, Point b) { return Line(O, O + (a - O)
384         ).unit() + (b - O).unit()); };
385     Point O = LineIntersection(get_angmidline(p[0], p[1], p[2]), get_angmidline(p
386         [1], p[0], p[2]));
387     double r = Seg{p[0], p[1]}.dis(O);
388     return Circle{O, r};
389 }

```