Standard Code Library

Moyi

East Northern University

September 9102

Contents

一切的开始 **3**

宏定义 . . . 3
对拍 . . . 3
int128 . . . 4

数据结构 **4**

线段树 . . . 4
区间加 & 区间求和 . . . 5
区间加区间乘 & 区间求和 (带取模) . . . 5
广义线段树 . . . 7
树状数组 . . . 9
单点修改 & 区间查询 . . . 9
区间修改 & 单点查询 . . . 9
二维单点修改 & 区间查询 . . . 10
二维区间修改 & 单点查询 . . . 11
二维区间修改 & 区间查询 . . . 12
并查集 . . . 13
并查集 . . . 13
带权并查集 . . . 13
堆 . . . 13
平衡树 . . . 14
Treap . . . 14
Splay . . . 16

数学 **19**

gcd . . . 19
组合数 . . . 19
素数筛 . . . 20
扩展欧几里得 . . . 20
类欧几里得 . . . 21
逆元 . . . 21
快速幂 . . . 22
质因数分解 . . . 22
多项式 . . . 22
FWT . . . 22
公式 . . . 24
调和级数部分和 . . . 24
一些数论公式 . . . 24
一些数论函数求和的例子 . . . 24
斐波那契数列性质 . . . 24
一些组合公式 . . . 24
中国剩余定理 . . . 24
博弈 . . . 25

图论 **25**

最短路 . . . 25
DIJKSTRA . . . 25
SPFA+ 判负环 . . . 26
Floyd . . . 26
LCA . . . 27
倍增 . . . 27
欧拉路径/回路 . . . 28
强连通分量与 2-SAT . . . 29
拓扑排序 . . . 31
连通性相关 . . . 31
割点和割边（无向图 . . . 31

强连通分量缩点（有向图 . . . 33
最小生成树 . . . 33
杂项 . . . 35
前向星存边 . . . 35

计算几何 **35**
几角排序 . . . 35
基本 . . . 36
平面最近点对（分治） . . . 37
全板子 . . . 39

字符串 **45**
manacher . . . 45
哈希 . . . 45
KMP . . . 47
Trie . . . 48
AC 自动机 . . . 48

杂项 **49**
STL . . . 49

一切的开始

宏定义

- qwq 7

```cpp
#include<bits/stdc++.h>
using namespace std;

#define EPS (1e-10)
#define int long long
//#define lson (rt<<1)
//#define rson (rt<<1|1)
//#define mid ((l+r)>>1)
#define mst(a) memset(a,0,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcodeforce++)
inline int read(){
    int x = 0, f = 1, c = getchar();
    while(!isdigit(c)) {if(c=='-')f=-1;c=getchar();}
    while(isdigit(c)) {x=(x<<1)+(x<<3)+(c^48);c=getchar();}
    return f==1?x:-x;
}
const int maxn = 2e5 +7;
const int maxm = 2e5 +7;
const int inf = 0x3f3f3f3f;
const int mod = 1e9 +7;

int n;

signed main() {
    #ifdef moyi_qwq
        freopen("D:/source file/intxt/in.txt","r",stdin);
    #endif


    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

对拍

```cpp
#include<bits/stdc++.h>
using namespace std;
int n;
signed main() {
    while(1) {
        system("make.exe");
        system("std.exe");
        system("mine.exe");
        if(system("fc std.out mine.out")) {
            printf("Wrong Answer!\n");
            int qwq; cin>>qwq;
            return 0;
        }
    }
    return (0);
}
```

- make

```
signed main() {
    #ifdef moyi_qwq
        freopen("in.txt","w",stdout);
    #endif

    srand((int)time(0));
    int maxlen = 5;
    int maxnum = 5 - 1;
    int round = 2;
```

```
    cout<<1<<" "<<20000<<endl;
    for(int i = 1; i <= 20000; i++) {
        cout<<1<<" "<<1<<endl;
    }

    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

- std 和 mine.cpp 要加这个

```
    freopen("in.txt","r",stdin);
    freopen("mine.out","w",stdout);
```

整行输入

- getline

```
char s[];
int len;
cin.getline(s, len);
```

- gets

```
char s[];
gets(s);
```

int128

```
inline __int128 read(){
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9'){
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9'){
        x = x*10 + ch - '0';
        ch = getchar();
    }
    return x * f;
}
inline void print(__int128 x){
    if (x < 0){
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}
```

- 只有基本的加减乘除

```
    __int128 a = read();
    __int128 b = read();
    print(a+b);
```

数据结构

线段树

```
#define lson (rt<<1)
```

```
#define rson (rt<<1|1)
#define mid ((l+r)>>1)
```

区间加 & 区间求和

```
ll a[maxn];
struct SegTree{
    ll val, add;
}tree[maxn << 2];

void pushdown(int l, int r, int o) {

    tree[lson].val += (mid - l + 1) * tree[o].add;
    tree[rson].val += (r - mid) * tree[o].add;
    tree[lson].add += tree[o].add;
    tree[rson].add += tree[o].add;
    tree[o].add = 0;

    return ;
}

void buildtree(int l, int r, int o) {
    tree[o].add = 0;//多组输入
    if(l == r)
    {
        tree[o].val = a[l];
        return;
    }
    buildtree(l , mid, lson);
    buildtree(mid + 1, r, rson);
    tree[o].val = tree[lson].val + tree[rson].val;
    return;
}

ll query(int ql, int qr, int l, int r, int o) {
    if(l > qr || r < ql) return 0;
    if(ql <= l && qr >= r) return tree[o].val;
    pushdown(l, r, o);
    return query(ql, qr, l, mid, lson) + query(ql, qr, mid + 1, r, rson);
}

void update(int ql, int qr, int l, int r, int o, int addval) { //区间加
    if(l > qr || r < ql) return;
    if(ql <= l && qr >= r) {
        tree[o].val += addval * (r - l + 1);
        tree[o].add += addval;
        return;
    }
    pushdown(l, r, o);

    update(ql, qr, l, mid, lson, addval);
    update(ql, qr, mid + 1, r, rson, addval);
    tree[o].val = tree[lson].val + tree[rson].val;
    return;
}
```

```
buildtree(1, n, 1);
update(x, y, 1, n, 1, k); //区间加
cout<<query(x, y, 1, n, 1)<<endl; //区间和
```

区间加区间乘 & 区间求和 (带取模)

```
int n, m, md;
int a[maxn];
struct node{
    int val;
    int add, mult;
}tree[maxn<<2];

void buildtree(int l, int r, int rt) {
```

```
    if(l == r) {
        tree[rt].val = a[l];
        tree[rt].add = 0;
        tree[rt].mult = 1;
        tree[rt].val %= md;
        return ;
    }

    buildtree(l, mid, lson);
    buildtree(mid+1, r, rson);
    tree[rt].add = 0;
    tree[rt].mult = 1;
    tree[rt].val = tree[lson].val + tree[rson].val;
    tree[rt].val %= md;
    return ;
}
//先乘再加
void pushdown(int l, int r, int rt) {
    tree[lson].val = (tree[lson].val * tree[rt].mult + tree[rt].add * (mid-l+1)) % md;
    tree[lson].add = (tree[lson].add * tree[rt].mult + tree[rt].add) % md;
    tree[lson].mult = (tree[lson].mult * tree[rt].mult) % md;

    tree[rson].val = (tree[rson].val * tree[rt].mult + tree[rt].add * (r-mid)) % md;
    tree[rson].add = (tree[rson].add * tree[rt].mult + tree[rt].add) % md;
    tree[rson].mult = (tree[rson].mult * tree[rt].mult) % md;

    tree[rt].add = 0;
    tree[rt].mult = 1;
    return ;
}

int query(int l, int r, int rt, int L, int R) {
    if(L > r || R < l) return 0;
    if(l >= L && r <= R) {return tree[rt].val;}

    pushdown(l, r, rt);
    return query(l, mid, lson, L, R) + query(mid+1, r, rson, L, R);
}

void add(int l, int r, int rt, int L, int R, int val) {
    if(L > r || R < l) return ;
    if(l >= L && r <= R) {
        tree[rt].add += val;
        tree[rt].val += val * (r-l+1);
        tree[rt].val %= md;
        return ;
    }
    pushdown(l,r,rt);
    add(l, mid, lson, L, R, val);
    add(mid+1, r, rson, L, R, val);
    tree[rt].val = tree[lson].val + tree[rson].val;
        tree[rt].val %= md;
    return ;
}

void mult(int l, int r, int rt, int L, int R, int val) {
    if(L > r || R < l) return ;
    if(l >= L && r <= R) {
        pushdown(l, r, rt);
        tree[rt].mult *= val;
        tree[rt].val *= val;
        tree[rt].val %= md;
        return ;
    }

    pushdown(l, r, rt);
    mult(l, mid, lson, L, R, val);
    mult(mid+1, r, rson, L, R, val);
    tree[rt].val = tree[lson].val + tree[rson].val;
        tree[rt].val %= md;
}
```

```
buildtree(1, n, 1);
mult(1, n, 1, x, y, k);
add(1, n, 1, x, y, k);
cout<<query(1, n, 1, x, y) % md<<endl;
```

广义线段树

```
int a[maxn];
struct node {
    int val;
    /* 所需的属性 */
}tree[maxn<<2];
node Merge(node a, node b) {
    node res;
    /* 合并两个 node*/
    return res;
}
void update(int rt) { //更新 rt
    tree[rt] = Merge(tree[lson], tree[rson]);
}
void build(int l, int r,int rt) {
    if(l == r){
        /* 初始化 */
        return ;
    }
    build(l, mid, lson);
    build(mid+1, r, rson);
    update(rt);
}

void modify(int l, int r, int rt, int x, int y) { //单点修改
    if(l == r) {
        /* 所需的修改 */
        tree[rt].s = tree[rt].t = y;
        return ;
    }
    if(x <= mid) modify(l, mid, lson, x, y);
    else modify(mid+1, r, rson, x, y);
    update(rt);
}
node query(int l, int r, int rt, int L, int R) { //区间查询
    if(L <= l && r <= R) return tree[rt]; /* 包含进去了 */
    node tmp;
    tmp.len = -1;
    if(L <= mid) tmp = query(l ,mid, lson, L, R); //左边
    if(mid < R) { //右边 合并
        if(tmp.len == -1) tmp = query(mid+1, r, rson, L, R);
        else tmp = Merge(tmp, query(mid+1, r, rson, L, R));
    }
    return tmp;
}

signed main() {
    int n, q;
    cin>>n>>q;
    for(int i = 1; i <= n; i++) cin>>a[i];
    build(1,n,1);
    int f, x, y;
    for(int i = 1; i <= q; i++) {
        cin>>f>>x>>y;
        if(f == 1) modify(1,n,1,x,y);
        else if(f == 2) cout<<query(1,n,1,x,y).val<<endl;
    }
    return (0);
}
```

+ 加法乘法

```
int n, m, md;
int a[maxn];
struct node{
```

```
    int val;
    int add, mult;
}tree[maxn<<2];

void buildtree(int l, int r, int rt) {
    if(l == r) {
        tree[rt].val = a[l];
        tree[rt].add = 0;
        tree[rt].mult = 1;
        tree[rt].val %= md;
        return ;
    }

    buildtree(l, mid, lson);
    buildtree(mid+1, r, rson);
    tree[rt].add = 0;
    tree[rt].mult = 1;
    tree[rt].val = tree[lson].val + tree[rson].val;
    tree[rt].val %= md;
    return ;
}
//先乘再加
void pushdown(int l, int r, int rt) {
    tree[lson].val = (tree[lson].val * tree[rt].mult + tree[rt].add * (mid-l+1)) % md;
    tree[lson].add = (tree[lson].add * tree[rt].mult + tree[rt].add) % md;
    tree[lson].mult = (tree[lson].mult * tree[rt].mult) % md;

    tree[rson].val = (tree[rson].val * tree[rt].mult + tree[rt].add * (r-mid)) % md;
    tree[rson].add = (tree[rson].add * tree[rt].mult + tree[rt].add) % md;
    tree[rson].mult = (tree[rson].mult * tree[rt].mult) % md;

    tree[rt].add = 0;
    tree[rt].mult = 1;
    return ;
}

int query(int l, int r, int rt, int L, int R) {
    if(L > r || R < l) return 0;
    if(l >= L && r <= R) {return tree[rt].val;}

    pushdown(l, r, rt);
    return query(l, mid, lson, L, R) + query(mid+1, r, rson, L, R);
}

void add(int l, int r, int rt, int L, int R, int val) {
    if(L > r || R < l) return ;
    if(l >= L && r <= R) {
        tree[rt].add += val;
        tree[rt].val += val * (r-l+1);
        tree[rt].val %= md;
        return ;
    }
    pushdown(l,r,rt);
    add(l, mid, lson, L, R, val);
    add(mid+1, r, rson, L, R, val);
    tree[rt].val = tree[lson].val + tree[rson].val;
        tree[rt].val %= md;
    return ;
}

void mult(int l, int r, int rt, int L, int R, int val) {
    if(L > r || R < l) return ;
    if(l >= L && r <= R) {
        pushdown(l, r, rt);
        tree[rt].mult *= val;
        tree[rt].val *= val;
        tree[rt].val %= md;
        return ;
    }

    pushdown(l, r, rt);
```

```
    mult(l, mid, lson, L, R, val);
    mult(mid+1, r, rson, L, R, val);
    tree[rt].val = tree[lson].val + tree[rson].val;
        tree[rt].val %= md;
}
```

树状数组

- 注意：0 是无效下标要 1 开始
- lowbit 是最低的 1 的数比如 10001100 就是 100

单点修改 & 区间查询

```
int tree[maxn];
int n;
int lowbit(int x) {return x & -x;}
void add(int pos, int val) { //在 pos 位置加 val
    while(pos <= n)
    {
        tree[pos] += val;
        pos += lowbit(pos);
    }
}
int sum(int pos)  { // [1-pos] 的和
    int ans = 0;
    while(pos)
    {
        ans += tree[pos];
        pos -= lowbit(pos);
    }
    return ans;
}
int query(int l, int r) { // [l, r] 的和
    return sum(r) - sum(l-1);
}
```

```
add(i,x);
cout<<query(l, r)<<endl;
```

区间修改 & 单点查询

```
int tree[maxn];
int n, m;
int lowbit(int x) { return x & -x; }
int getsum(int x) { // 输出pos的值
    int ans = 0;
    while (x) {
        ans += tree[x];
        x -= lowbit(x);
    }
    return ans;
}
void add(int pos, int val) {
    while (pos <= n) {
        tree[pos] += val;
        pos += lowbit(pos);
    }
    return;
}
void modify(int l, int r, int val) { //[l,r]加上val
    add(l, val);
    add(r + 1, -val);
}
```

```
modify(x, y, k);
cout<<getsum(x)<<endl;
```

区间修改 & 区间查询

其中 n 是数组的最大长度，q 是无用变量

```
int tree1[maxn], tree2[maxn];
int n, q;
int lowbit(int x) {return x & -x;}
void add(int pos, int val) {
    int addval = val * pos;
    while(pos <= n) {
        tree1[pos] += val;
        tree2[pos] += addval;
        pos += lowbit(pos);
    }
}
int sum1(int pos) {
    // cout<<"sum1:"<<pos<<"=";
    int ans = 0;
    while(pos) {
        ans += tree1[pos];
        pos -= lowbit(pos);
    }
    // cout<<ans<<"\n";
    return ans;
}
int sum2(int pos) {
    // cout<<"sum2:"<<pos<<"=";
    int ans = 0;
    while(pos) {
        ans += tree2[pos];
        pos -= lowbit(pos);
    }
    // cout<<ans<<endl;
    return ans;
}
int sum(int pos) {return sum1(pos) * (pos + 1) - sum2(pos);}
void modify(int l, int r, int val) { // [l,r] 加上 val
    add(l ,val);
    add(r+1, -val);
}
int query(int l, int r) { // [l,r] 的区间和
    return sum(r) - sum(l-1);
}
```

```
modify(l,r,v);
cout<<query(l,r)<<endl;
```

二维单点修改 & 区间查询

```
int tree[maxn][maxn];
int xn, yy;
int lowbit(int x) {
    return x & -x;
}
void add(int x, int y, int val) { // (x,y)单点+val
    int my = y;
    while (x <= xn) {
        y = my;
        while (y <= yy) {
            tree[x][y] += val;
            y += lowbit(y);
        }
        x += lowbit(x);
    }
```

```
}
int getsum(int x, int y) {
    int ans = 0;
    int my = y;
    while (x) {
        y = my;
        while (y) {
            ans += tree[x][y];
            y -= lowbit(y);
        }
        x -= lowbit(x);
    }
    return ans;
}
int q_get(int x1, int y1, int x2, int y2) { // [x1,x2][y1,y2]的区间和
    int ans = 0;
    ans += getsum(x2, y2);
    ans -= getsum(x1 - 1, y2);
    ans -= getsum(x2, y1 - 1);
    ans += getsum(x1 - 1, y1 - 1);
    return ans;
}
```

```
add(x, y, val);
cout<<q_get(a, b, c, d)<<endl;
```

二维区间修改 & 单点查询

```
int n, m;
int a[maxn][maxn];
int tree[maxn][maxn]; //b[i][j] = a[i][j] + a[i-1][j-1] - a[i][j-1] - a[i-1][j]
int lowbit(int x) {return x&-x;}
int geta(int x, int y) { //(x,y)位置的值
    int ans = 0;
    int memy = y;
    while(x) {
        y = memy;
        while(y) {
            ans += tree[x][y];
            y -= lowbit(y);
        }
        x -= lowbit(x);
    }
    return ans;
}
void modify(int x, int y, int val) {
    int memoy = y;
    while(x <= n) {
        y = memoy;
        while(y <= m) {
            tree[x][y] += val;
            y += lowbit(y);
        }
        x += lowbit(x);
    }
}
void add(int xx1, int yy1, int xx2, int yy2, int val) { //区间[xx1,xx2]加val
```

```
    modify(xx1, yy1, val);
    modify(xx1, yy2 + 1, -val);
    modify(xx2 + 1, yy1, -val);
    modify(xx2 + 1, yy2 + 1, val);
}

add(a,b,c,d,k);
cout<<geta(x,y)<<endl;
```

二维区间修改 & 区间查询

```
int n, m;
int a[maxn][maxn];
int t1[maxn][maxn], t2[maxn][maxn], t3[maxn][maxn], t4[maxn][maxn];
//bij bij*j bij*i bij *i*j
int lowbit(int x) {return x & -x;}
void add(int x, int y, int val) {
    // cout<<x<<" "<<y<<" "<<val<<endl;
    int memoy = y, memox = x;
    while(x <= n) {
        y = memoy;
        while(y <= m) {
            t1[x][y] += val;
            t2[x][y] += val * memoy;
            t3[x][y] += val * memox;
            t4[x][y] += val * memox * memoy;
            y += lowbit(y);
        }
        x += lowbit(x);
    }
}
int ask(int x, int y) {
    int ans = 0;
    int memoy = y, memox = x;
    while(x) {
        y = memoy;
        while(y) {
            ans += (memoy+1)*(memox+1)*t1[x][y];
            ans -= t2[x][y] * (memox + 1);
            ans -= t3[x][y] * (memoy + 1);
            ans += t4[x][y];
            y -= lowbit(y);
        }
        x -= lowbit(x);
    }
    return ans;
}
void range_add(int xx1, int yy1, int xx2, int yy2, int val) { //区间加
    add(xx1,yy1, val);
    add(xx1, yy2 + 1, -val);
    add(xx2 + 1, yy1, -val);
    add(xx2+1,yy2+1,val);
}
int range_ask(int xx1, int yy1, int xx2, int yy2) { //区间和
    int ans = 0;
    ans += ask(xx1-1,yy1-1);
    ans -= ask(xx1-1,yy2);
```

```
    ans -= ask(xx2, yy1-1);
    ans += ask(xx2, yy2);
    return ans;
}

range_add(a,b,c,d,x);
cout<<range_ask(a,b,c,d)<<endl;
```

并查集

并查集

```
int n;
int fa[maxn];
void init() {for(int i = 0; i <= n; i++) fa[i]=i;}
int find(int x) {return fa[x]==x?x:fa[x]=find(fa[x]);} //寻找 x 的祖先
void merge(int x, int y) { //合并 x y
    int a = find(x), b = find(y);
    fa[a] = b;
}
```

带权并查集

```
int n, m;
int fa[maxn],d[maxn];
void init() {for(int i = 0; i <= n; i++) fa[i]=i;}
int find(int x) { //寻找 x 的祖先
    if(fa[x] == x) return x;
    else {
        int oldFa = fa[x];
        fa[x] = find(oldFa);
        d[x] = d[x] + d[oldFa];
        return fa[x];
    }
}
void merge(int x,int y,int w) { //合并 x y
    int fax = find(x),fay = find(y);
    if(fax == fay) return;
    fa[fax] = fay;
    d[fax] = -d[x] + d[y] + w;
}
int dist(int x,int y) { //x y 的距离 (必须在一个集里)
    int fax = find(x),fay = find(y);
    if(fax != fay) return -1;
    else return d[x] - d[y];
}
```

```
merge(x,y,z);
if(find(x)!=find(y)) cout<<"?"<<endl;
else cout<<dist(x,y)<<endl;
```

堆

```
class HEAP{
    public:
    int minheap[maxn];//1 index
    int heap_cnt = 0;
    void swap(int i, int j) {
        int t = minheap[i];
        minheap[i] = minheap[j];
        minheap[j] = t;
    }
    void push(int x) {
        heap_cnt++;
        minheap[heap_cnt] = x;
```

```
        for(int i = heap_cnt, j = i >> 1; ; ) {
            if(j == 0) break; //to the most top
            if(minheap[i] < minheap[j]) {
                swap(i,j);
            }
            i = j;
            j = i >> 1;
        }
        return ;
    }
    void pop() {
        minheap[1] = minheap[heap_cnt];
        heap_cnt--;
        for(int i = 1, j = i << 1; ; ){
            if(j > heap_cnt) break; // to the bottom
            if(j < heap_cnt && minheap[j+1] < minheap[j]) j++;// find the smaller one of sons
            if(minheap[j] > minheap[i]) break;// right position
            swap(i,j);
            i = j;
            j = i << 1;
        }
        return ;
    }
    int top() {
        return minheap[1];
    }
};
```

```
h.push(x);
cout<<h.top()<<endl;
h.pop();
```

平衡树

Treap

```
struct node{
    int key, priority;
    node *left, *right;
};
typedef node* Node;

Node _delete(Node &t, int key);

Node rt = NIL;

Node rightRotate(Node &t) {
    Node s = t->left;
    t->left = s->right;
    s->right = t;
    return s; // the new root of subtree
}
Node leftRotate(Node &t) {
    Node s = t->right;
    t->right = s->left;
    s->left = t;
    return s; // the new root of subtree
}
```

```
Node insert(Node &t, int key, int priority){        // search the corresponding place recursively
    if (t == NIL) {
        t = new node;
        Node &newnd = t;
        newnd->key = key;
        newnd->priority = priority;                 // create a new node when you reach a leaf
        newnd->left = NIL;
        newnd->right = NIL;
        return newnd;
    }
    if (key == t->key) {
        return t;                                   // ignore duplicated keys
    }


    if(key < t->key){                                 // move to the left child
        t->left = insert(t->left, key, priority);   // update the pointer to the left child
     if(t->priority < t->left->priority)       // rotate right if the left child has higher priority
            t = rightRotate(t);
    }
    else {                                          // move to the right child
        t->right = insert(t->right, key, priority); // update the pointer to the right child
     if(t->priority < t->right->priority) {      // rotate left if the right child has higher priority
            t = leftRotate(t);
        }
    }
    return t;
}
Node delete1(Node &t, int key) {                        // seach the target recursively
    if(t == NIL)
        return NIL;
    if(key < t->key) {                              // search the target recursively
        t->left = delete1(t->left, key);
    }
    else if (key > t->key)
        t->right = delete1(t->right, key);
    else
        return _delete(t, key);
    return t;
}
Node _delete(Node &t, int key) {                       // if t is the target node
    if(t->left == NIL && t->right == NIL)         // if t is a leaf
        return NIL;
   else if (t->left == NIL)               // if t has only the right child, then perform left rotate
        t = leftRotate(t);
   else if (t->right == NIL)              // if t has only the left child, then perform right rotate
        t = rightRotate(t);
    else {                                      // if t has both the left and right child
        if (t->left->priority > t->right->priority)  // pull up the child with higher priority
            t = rightRotate(t);
        else
            t = leftRotate(t);
    }
    return delete1(t, key);
}
void Print(Node rt, int f) {
```

```
    if(rt == NIL) return ;
    if(f == 1) cout<<" "<<rt->key;
    Print(rt->left,f);
    if(f == 2) cout<<" "<<rt->key;
    Print(rt->right,f);
}
bool Find(Node rt, int x) {
    if(rt == NIL) return false;
    if(x == rt->key) return true;
    else if(x < rt->key) return Find(rt->left,x);
    else return Find(rt->right, x);
    return false;
}

int n;

signed main() {
    #ifdef moyi_qwq
        freopen("D:/source file/intxt/in.txt","r",stdin);
    #endif

    cin>>n;
    string s;
    for(int i = 1; i <= n; i++) {
        cin>>s;
        if(s == "insert") {
            int k, p;
            cin>>k>>p;
            insert(rt,k,p);
        }
        else if(s == "print") {
            Print(rt,2); cout<<endl;
            Print(rt,1); cout<<endl;
        }
        else if(s == "find") {
            int x; cin>>x;
            if(Find(rt, x)) cout<<"yes"<<endl;
            else cout<<"no"<<endl;
        }
        else if(s == "delete") {
            int x; cin>>x;
            delete1(rt,x);
        }
    }


    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

Splay

```
inline ll read()
{
    rll x=0;
    bool fg=false;
```

```
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-') fg=true;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<3)+(x<<1)+(ch^48);
        ch=getchar();
    }
    return fg?~x+1:x;
}
const int N=1e5+5;
int n,tot,root;
int ch[N][2],fa[N];//左孩子,右孩子,父亲
ll val[N],siz[N],cnt[N];//点值
void pushup(int id) { //更新siz
    siz[id]=siz[ch[id][0]]+siz[ch[id][1]]+cnt[id];
}
void spin(int x) {
    rint y=fa[x],z=fa[y],d=(ch[y][1]==x);//d 判断x是y的左孩子还是右孩子
    ch[z][ch[z][1]==y]=x,fa[x]=z;//处理x与z的关系
    ch[y][d]=ch[x][d^1],fa[ch[x][d^1]]=y;//处理y的孩子与x的孩子的关系
    ch[x][d^1]=y;fa[y]=x;//处理y与x的关系
    pushup(y);//先更新y
    pushup(x);//在更新x
}
void splay(int x,int goal) {
    while(fa[x]!=goal)//判断是否已经到目标点的下边
    {
        rint y=fa[x],z=fa[y];
        if(z!=goal)//判断是情况一还是情况二、三
            (ch[y][0]==x)^(ch[z][0]==y)?spin(x):spin(y);
        //判断是情况二还是情况三
        spin(x);
    }
    if(goal==0) root=x;//如果移动到了根节点,则更新根节点
}
void insert(ll x) {
    int u=root,fat=0;
    while(u&&val[u]!=x) { //先向下找
        fat=u;
        u=ch[u][x>val[u]];
    }
    if(u)   cnt[u]++;
    else {
        u=++tot;
        if(fat) ch[fat][x>val[fat]]=u;//如果不是根节点,更新孩子节点
        fa[u]=fat;//插入操作
        val[u]=x;
        siz[u]=1;
        cnt[u]=1;
    }
    splay(u,0);//每次都要伸展,避免成链
}
```

```
void find(ll x) {
    int u=root;
    if(!u)  return;//不存在该节点,直接返回
    while(ch[u][x>val[u]]&&x!=val[u])//找到该节点的位置
        u=ch[u][x>val[u]];
    splay(u,0);//伸展
}
int get(ll x,int d) { //d:0找前驱 1找后继
    find(x);//先伸展
    int u=root;
    if((val[u]>x&&d)||(val[u]<x&&!d))   return u;
    //如果该节点已经符合要求,直接返回位置
    u=ch[u][d];//找到左右子树
    while(ch[u][d^1])   u=ch[u][d^1];
    //找左子树中最大的或右子树中最小的(关键看你找前驱还是后继)
    return u;//返回前驱或后继的位置
}
void del(ll x) {
    int pre=get(x,0),nxt=get(x,1);//找前驱后继
    splay(pre,0),splay(nxt,pre);//伸展
    int id=ch[nxt][0];//要删除的点
    if(cnt[id]>1)//如果这个数值有重复,直接--cnt即可
    {
        --cnt[id];
        splay(id,0);//伸展
    }
    else
    {
        ch[nxt][0]=0,fa[id]=0;//先切断联系
        val[id]=0,cnt[id]=0,siz[id]=0;//再进行删除
        pushup(nxt),pushup(pre);//最后更新siz
    }
}
ll k_th(int k) {
    int x=root;
    if(siz[x]<k)    return 0;//整棵树的大小都比k小,则没有这个数
    while(1)
    {
        int y=ch[x][0];
        if(k>siz[y]+cnt[x])//比左子树的大小和该节点的重复次数大
        {
            k-=(siz[y]+cnt[x]);
            x=ch[x][1];//去右子树中搜
        }
        else
        {
            if(siz[y]>=k)   x=y;//小于等于左子树,去左子树中搜
            else    return val[x];//否则,返回该值
        }
    }
}
signed main() {
    insert(-inf),insert(inf); ////////////////////// 注意 ////////////////////
    n=read();
    for(rint i=1;i<=n;++i) {
        int op=read();
```

```
        ll x=read();
        switch(op) {
            case 1:
                insert(x); //插入数x
                break;
            case 2:
                del(x); //删除数x（若有多个相同的数，只删除一个）；
                break;
            case 3:
                find(x); //查询x的排名（若有多个相同的数，输出最小的排名）；
                printf("%lld\n",siz[ch[root][0]]);//因为前面还加了一个-INF，所以不用再加1了
                break;
            case 4:
                printf("%lld\n",k_th(x+1)); //查询排名为x的数
                break;
            case 5:
                printf("%lld\n",val[get(x,0)]); //求x的前驱（前驱定义为小于x，且最大的数）
                break;
            case 6:
                printf("%lld\n",val[get(x,1)]); //求x的后继（后继定义为大于x，且最小的数）
                break;
        }
    }
    return 0;
}
```

数学

gcd

```
ll Gcd(ll a,ll b){return b==0?a:Gcd(b,a%b);}
```

组合数

```
int qpow(int x, int n) {
    if(!n) return 1;
    int t = 1;
    while(n) {
        if(n&1) t *= x;
        n >>= 1;
        x *= x;
        x = (x + mod) % mod;
        t = (t + mod) % mod;
    }
    return t;
}

int jc[maxn], inv_jc[maxn];
int get_inv(int x) {return qpow(x, mod - 2);}
int A(int x, int n) { //get x from n
    if(x>n)
        return 0;
    else
        return (jc[n]*inv_jc[n-x])%mod;
}
int C(int x, int n) { //get x from n
    if(x>n)
```

```
        return 0;
    else if(x==n||x==0)
        return 1;
    else
        return (A(x,n)*inv_jc[x])%mod;
}
```

素数筛

- 线性筛

visit[] 第 i 个数是不是质数,1 表示质数，prime[] 质数集合 2 3 5 7，k 质数数量, 筛到 maxn

```
bool visit[maxn]; //visit 存的第 i 个数是不是质数 0 表示质数
int k, prime[maxn]; //k 存的质数数量, 筛到 maxn,prime 存的质数集合 (0 开始) 2 3 5 7 ....

void initPrime()//init
{
    visit[0] = visit[1] = 1;
    for(int i = 2; i < maxn; i++)
    {
        if(!visit[i]) prime[k++] = i;
        for(int j = 0; j < k && i*prime[j]<maxn; j++)//遍历素数数组
        {
            visit[i * prime[j]] = 1;
            if(i % prime[j] == 0) break;
        }
    }
}
```

- 线性筛 + 欧拉函数

欧拉函数（Euler’s totient function），即 $\varphi(n)$，表示的是小于等于 n 和 n 互质的数的个数。存在 phi[i] 里。

```
//phi(i):= 小于等于 i 的与 i 互质的数的个数
const int p_max = 1e5 + 100;
int phi[p_max];
void get_phi() {
    static bool vis[p_max];
    static int prime[p_max], p_sz, d;
    vis[0] = vis[1] = 1; phi[1] = 1;
    for(int i = 2; i < p_max; i++) {
        if (!vis[i]) {
            prime[p_sz++] = i;
            phi[i] = i - 1;
        }
        for (int j = 0; j < p_sz && (d = i * prime[j]) < p_max; ++j) {
            vis[d] = 1;
            if (i % prime[j] == 0) {
                phi[d] = phi[i] * prime[j];
                break;
            }
            else phi[d] = phi[i] * (prime[j] - 1);
        }
    }
}
```

扩展欧几里得

- 求 $ax + by = gcd(a, b)$ 的一组解
- 如果 a 和 b 互素，那么 x 是 a 在模 b 下的逆元
- 注意 x 和 y 可能是负数

```
//ax+by=gcd(a,b) 是否有解 有解解就是 x
bool ex_gcd(int a, int b, int& x, int& y) {
    if(b == 0) {
        x = 1;
        y = 0;
        return a;
```

```
    }
    int d = ex_gcd(b, a%b, x, y);
    int temp = x;
    x = y;
    y = temp - a/b * y;
    return d;
}
//同余方程 ax+by=c 即 ax=c(mod b) 是否有解
bool CongruenceEquation(int a, int b,int c, int& x, int& y) {
    int d = ex_gcd(a,b,x,y);
    if(c%d != 0) return 0;
    int k = c / d;
    x *= k;
    y *= k;
    return 1;
}
```

类欧几里得

- $m = \lfloor \frac{an+b}{c} \rfloor$.
- $f(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时，$f(a,b,c,n) = (\frac{a}{c})n(n+1)/2 + (\frac{b}{c})(n+1) + f(a \bmod c, b \bmod c, c, n)$; 否则 $f(a,b,c,n) = nm - f(c, c-b-1, a, m-1)$。
- $g(a,b,c,n) = \sum_{i=0}^{n} i\lfloor \frac{ai+b}{c} \rfloor$: 当 $a \geq c$ or $b \geq c$ 时，$g(a,b,c,n) = (\frac{a}{c})n(n+1)(2n+1)/6 + (\frac{b}{c})n(n+1)/2 + g(a \bmod c, b \bmod c, c, n)$; 否则 $g(a,b,c,n) = \frac{1}{2}(n(n+1)m - f(c, c-b-1, a, m-1) - h(c, c-b-1, a, m-1))$。
- $h(a,b,c,n) = \sum_{i=0}^{n} \lfloor \frac{ai+b}{c} \rfloor^2$: 当 $a \geq c$ or $b \geq c$ 时，$h(a,b,c,n) = (\frac{a}{c})^2 n(n+1)(2n+1)/6 + (\frac{b}{c})^2(n+1) + (\frac{a}{c})(\frac{b}{c})n(n+1) + h(a \bmod c, b \bmod c, c, n) + 2(\frac{a}{c})g(a \bmod c, b \bmod c, c, n) + 2(\frac{b}{c})f(a \bmod c, b \bmod c, c, n)$; 否则 $h(a,b,c,n) = nm(m+1) - 2g(c, c-b-1, a, m-1) - 2f(c, c-b-1, a, m-1) - f(a,b,c,n)$。

逆元

- 如果 p 不是素数，使用拓展欧几里得
- 前置模板：快速幂 / 扩展欧几里得

```
int n;
ll mod = (int)(1e9+7);
inline ll qpow(ll base, ll rk) {
    ll ans = 1;
    ll now = (base % mod + mod) % mod;
    for(; rk; rk >>= 1) {
        if(rk & 1) {ans *= now; ans %= mod;}
        now *= now; now %= mod;
    }
    return ans;
}

//费马小定理求单个数逆元 mod 必须素数
ll get_inv(ll x) {return qpow(x, mod - 2);}

int inv_n[maxn];
//线性求 n 个数的逆元
void init_n() {
    inv_n[1] = 1;
    for(int i = 2; i <= n; i++) {
        inv_n[i] = 1ll * (mod - mod / i) * inv_n[mod % i] % mod;
    }
}

ll a[maxn], inv[maxn], s[maxn], sv[maxn];//原数组 逆元 前缀积 逆元前缀积
//线性求任意 n 个数的逆元
void init_any() {
    s[0] = 1;
    for(int i = 1; i <= n; i++) s[i] = s[i-1] * a[i] % mod;
    sv[n] = qpow(s[n], mod-2);
    for(int i = n; i >= 1; i--) sv[i-1] = sv[i] * a[i] % mod;
    for(int i = 1; i <= n; i++) inv[i] = sv[i] * s[i-1] % mod;
}
```

- 预处理阶乘及其逆元

```
LL invf[M], fac[M] = {1};
void fac_inv_init(LL n, LL p) {
    FOR (i, 1, n)
        fac[i] = i * fac[i - 1] % p;
    invf[n - 1] = bin(fac[n - 1], p - 2, p);
    FORD (i, n - 2, -1)
        invf[i] = invf[i + 1] * (i + 1) % p;
}
```

快速幂

- 如果模数是素数，则可在函数体内加上 `n %= MOD - 1;`（费马小定理）

```
ll FastPowerMod(ll x, ll n)
{
    if(!n) return 1;
    ll t = 1;
    while(n)
    {
        if(n&1) t *= x;
        n >>= 1;
        x *= x;
        x = (x + mod) % mod;
        t = (t + mod) % mod;
    }
    return t;
}
```

质因数分解

- 前置模板：素数筛
- 带指数

```
LL factor[30], f_sz, factor_exp[30];
void get_factor(LL x) {
    f_sz = 0;
    LL t = sqrt(x + 0.5);
    for (LL i = 0; pr[i] <= t; ++i)
        if (x % pr[i] == 0) {
            factor_exp[f_sz] = 0;
            while (x % pr[i] == 0) {
                x /= pr[i];
                ++factor_exp[f_sz];
            }
            factor[f_sz++] = pr[i];
        }
    if (x > 1) {
        factor_exp[f_sz] = 1;
        factor[f_sz++] = x;
    }
}
```

多项式

FWT

```
//快速沃尔什变换 位运算卷积
//https://www.luogu.com.cn/problem/P4717
#include<bits/stdc++.h>
using namespace std;

// #define int long long
#define mst(a) memset(a,0,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcod
typedef long long ll;
```

```
typedef unsigned long long ull;
const ll maxn = 2e5 +7;
const ll maxm = 2e5 +7;
const ll inf = 0x3f3f3f3f;
const ll mod = 998244353;//1e9+7
const ll inv2 = 499122177;
const ll
Cor[2][2] = {{1,0},{1,1}},
Cand[2][2] = {{1,1},{0,1}},
Cxor[2][2] = {{1,1},{1,mod-1}},
ICor[2][2] = {{1,0},{mod-1,1}},
ICand[2][2] = {{1,mod-1},{0,1}},
ICxor[2][2] = {{inv2, inv2},{inv2,mod-inv2}};

void FWT(ll *F, const ll c[2][2], int n) {
    for(int len = 1; len < n; len <<= 1)
        for(int p = 0; p < n; p += len*2)
            for(int i = p; i < p +len; i++) {
                ll sav = F[i];
                F[i] = (c[0][0]*F[i]+c[0][1]*F[i+len])%mod;
                F[i+len] = (c[1][0]*sav+c[1][1]*F[i+len])%mod;
            }
}

void bitmul(ll *F, ll *G, const ll C[2][2], const ll IC[2][2], int n) {
    FWT(F, C, n); FWT(G, C, n);
    for(int i = 0; i < n; i++) F[i] = F[i]*G[i]%mod;
    FWT(F, IC, n);
}

int n;
ll f[maxn], g[maxn], a[maxn], b[maxn];
signed main()
{
    // freopen("D:/c++source file/intxt/in.txt","r",stdin);
    ios :: sync_with_stdio(0);
    cin.tie(0);

    int n; cin>>n; n = (1<<n);
    for(int i = 0; i < n; i++) cin>>f[i];
    for(int i = 0; i < n; i++) cin>>g[i];
    memcpy(a,f,sizeof(ll)*n);memcpy(b,g,sizeof(ll)*n);
    bitmul(a,b, Cor, ICor, n);
    for(int i = 0; i < n; i++) cout<<a[i]<<" "; cout<<endl;

    memcpy(a,f,sizeof(ll)*n);memcpy(b,g,sizeof(ll)*n);
    bitmul(a,b, Cand, ICand, n);
    for(int i = 0; i < n; i++) cout<<a[i]<<" "; cout<<endl;

    memcpy(a,f,sizeof(ll)*n);memcpy(b,g,sizeof(ll)*n);
    bitmul(a,b, Cxor, ICxor, n);
    for(int i = 0; i < n; i++) cout<<a[i]<<" "; cout<<endl;

    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

公式

调和级数部分和

$$S=\frac{1}{1}+\frac{1}{2}+.....+\frac{1}{n}S=ln(n)+eulr+\frac{1}{2n}elur=0.57721\ 56649\ 01532\ 86060$$

一些数论公式

- 当 $x\geq\phi(p)$ 时有 $a^x\equiv a^{x\ mod\ \phi(p)+\phi(p)}\pmod p$
- $\mu^2(n)=\sum_{d^2|n}\mu(d)$
- $\sum_{d|n}\varphi(d)=n$
- $\sum_{d|n}2^{\omega(d)}=\sigma_0(n^2)$，其中 ω 是不同素因子个数
- $\sum_{d|n}\mu^2(d)=2^{\omega(d)}$

一些数论函数求和的例子

- $\sum_{i=1}^n i[gcd(i,n)=1]=\frac{n\varphi(n)+[n=1]}{2}$
- $\sum_{i=1}^n\sum_{j=1}^m[gcd(i,j)=x]=\sum_d\mu(d)\lfloor\frac{n}{dx}\rfloor\lfloor\frac{m}{dx}\rfloor$
- $\sum_{i=1}^n\sum_{j=1}^m gcd(i,j)=\sum_{i=1}^n\sum_{j=1}^m\sum_{d|gcd(i,j)}\varphi(d)=\sum_d\varphi(d)\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$
- $S(n)=\sum_{i=1}^n\mu(i)=1-\sum_{i=1}^n\sum_{d|i,d<i}\mu(d)\overset{t=\frac{i}{d}}{=}1-\sum_{t=2}^n S(\lfloor\frac{n}{t}\rfloor)$
  - 利用 $[n=1]=\sum_{d|n}\mu(d)$
- $S(n)=\sum_{i=1}^n\varphi(i)=\sum_{i=1}^n i-\sum_{i=1}^n\sum_{d|i,d<i}\varphi(i)\overset{t=\frac{i}{d}}{=}\frac{i(i+1)}{2}-\sum_{t=2}^n S(\frac{n}{t})$
  - 利用 $n=\sum_{d|n}\varphi(d)$
- $\sum_{i=1}^n\mu^2(i)=\sum_{i=1}^n\sum_{d^2|n}\mu(d)=\sum_{d=1}^{\lfloor\sqrt{n}\rfloor}\mu(d)\lfloor\frac{n}{d^2}\rfloor$
- $\sum_{i=1}^n\sum_{j=1}^n gcd^2(i,j)=\sum_d d^2\sum_t\mu(t)\lfloor\frac{n}{dt}\rfloor^2$
  $\overset{x=dt}{=}\sum_x\lfloor\frac{n}{x}\rfloor^2\sum_{d|x}d^2\mu(\frac{x}{d})$
- $\sum_{i=1}^n\varphi(i)=\frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n[i\perp j]-1=\frac{1}{2}\sum_{i=1}^n\mu(i)\cdot\lfloor\frac{n}{i}\rfloor^2-1$

斐波那契数列性质

- $F_{a+b}=F_{a-1}\cdot F_b+F_a\cdot F_{b+1}$
- $F_1+F_3+\cdots+F_{2n-1}=F_{2n},F_2+F_4+\cdots+F_{2n}=F_{2n+1}-1$
- $\sum_{i=1}^n F_i=F_{n+2}-1$
- $\sum_{i=1}^n F_i^2=F_n\cdot F_{n+1}$
- $F_n^2=(-1)^{n-1}+F_{n-1}\cdot F_{n+1}$
- $gcd(F_a,F_b)=F_{gcd(a,b)}$
- 模 n 周期（皮萨诺周期）
  - $\pi(p^k)=p^{k-1}\pi(p)$
  - $\pi(nm)=lcm(\pi(n),\pi(m)),\forall n\perp m$
  - $\pi(2)=3,\pi(5)=20$
  - $\forall p\equiv\pm1\pmod{10},\pi(p)|p-1$
  - $\forall p\equiv\pm2\pmod 5,\pi(p)|2p+2$

一些组合公式

- 错排公式：$D_1=0,D_2=1,D_n=(n-1)(D_{n-1}+D_{n-2})=n!(\frac{1}{2!}-\frac{1}{3!}+\cdots+(-1)^n\frac{1}{n!})=\lfloor\frac{n!}{e}+0.5\rfloor$
- 卡塔兰数（n 对括号合法方案数，n 个结点二叉树个数，$n\times n$ 方格中对角线下方的单调路径数，凸 $n+2$ 边形的三角形划分数，n 个元素的合法出栈序列数）：$C_n=\frac{1}{n+1}\binom{2n}{n}=\frac{(2n)!}{(n+1)!n!}$

中国剩余定理

- 无解返回 -1
- 前置模板：扩展欧几里得

```
LL CRT(LL *m, LL *r, LL n) {
    if (!n) return 0;
    LL M = m[0], R = r[0], x, y, d;
    FOR (i, 1, n) {
        d = ex_gcd(M, m[i], x, y);
        if ((r[i] - R) % d) return -1;
        x = (r[i] - R) / d * x % (m[i] / d);
        // 防爆 LL
        // x = mul((r[i] - R) / d, x, m[i] / d);
        R += x * M;
        M = M / d * m[i];
        R %= M;
    }
    return R >= 0 ? R : R + M;
}
```

博弈

- Nim 游戏：每轮从若干堆石子中的一堆取走若干颗。先手必胜条件为石子数量异或和非零。
- 阶梯 Nim 游戏：可以选择阶梯上某一堆中的若干颗向下推动一级，直到全部推下去。先手必胜条件是奇数阶梯的异或和非零（对于偶数阶梯的操作可以模仿）。
- Anti-SG：无法操作者胜。先手必胜的条件是：
  - SG 不为 0 且某个单一游戏的 SG 大于 1 。
  - SG 为 0 且没有单一游戏的 SG 大于 1。
- Every-SG：对所有单一游戏都要操作。先手必胜的条件是单一游戏中的最大 step 为奇数。
  - 对于终止状态 step 为 0
  - 对于 SG 为 0 的状态，step 是最大后继 step +1
  - 对于 SG 非 0 的状态，step 是最小后继 step +1
- 树上删边：叶子 SG 为 0，非叶子结点为所有子结点的 SG 值加 1 后的异或和。

尝试：

- 打表找规律
- 寻找一类必胜态（如对称局面）
- 直接博弈 dp

图论

最短路

DIJKSTRA

```
int n, m, s;
vector<pair<int, int>> e[maxn]; // v, w 到达点 长度
int dis[maxn]; //距离
bool vis[maxn]; //是否访问过
void Dijk() {
    mst(vis);
    for(int i = 0; i <= n +7; i++) dis[i] = inf;
    dis[s] = 0;
    priority_queue<pair<int, int>> q; // u,dis
    q.push(make_pair(s,0));
    while(!q.empty()) {
        int u = q.top().first, d = q.top().second;
        q.pop();
        for(auto y : e[u]) {
            int v = y.first, w = y.second;
            int td = d + w;
            if(td < dis[v]) {
                dis[v] = td;
                q.push(make_pair(v,td));
            }
        }
    }
}
```

SPFA+ 判负环

```
int n, m, s;
vector<pair<int, int>> e[maxn]; //to w 边终点, 边权
int dis[maxn], cnt[maxn], vis[maxn]; // 0-(n-1)

bool spfa(int s) { // 返回true如果没有负环 否则返回false
    for(int i = 0; i < n; i++) {
        dis[i] = inf;
    }
    queue<int> q;
    dis[s] = 0; vis[s] = 1;
    q.push(s);
    while(!q.empty()) {
        int u = q.front();
        q.pop(); vis[u] = 0;
        for(auto y : e[u]) {
            int v = y.first, w = y.second;
            if(dis[v] > dis[u] + w) {
                dis[v] = dis[u] + w;
                cnt[v] = cnt[u] + 1;  // 记录最短路经过的边数
                if (cnt[v] >= n) return false;
                // 在不经过负环的情况下, 最短路至多经过 n - 1 条边
                // 因此如果经过了多于 n 条边, 一定说明经过了负环
                if (!vis[v]) q.push(v), vis[v] = 1;
            }
        }
    }
    return true;
}
```

Floyd

初始化

```
int n, m;
int dis[maxn][maxn];

for(int i = 0; i < maxn; i++) { //初始化dis
    for(int j = 0; j < maxn; j++) {
        dis[i][j] = inf;
    }
}
for(int i = 0; i < n; i++) dis[i][i] = 0;//初始化dis
```

输入

```
for(int i = 1; i <= m; i++) {
    int u, v, w;
    cin>>u>>v>>w;
    dis[u][v] = w;
}
```

计算

```
for(int k = 0; k < n; k++) { //先枚举中转点
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            dis[i][j] = min(dis[i][j], dis[i][k] + dis[k][j]);
        }
```

```
    }
}
```

判负环

```
for(int i = 0; i < n; i++) {
    if(dis[i][i] < 0) { // 如果自己到自己是负数则有负环
        cout<<"NEGATIVE CYCLE"<<endl;
        return 0;
    }
}
```

输出

```
for(int i = 0; i < n; i++) {
    for(int j = 0; j < n; j++) {
        //注意不可达到判断不能直接==inf,要设置一个上界
        if(dis[i][j]>1e10) cout<<"INF";
        else cout<<dis[i][j];
        cout<<(j==n-1?"\n":" ");
    }
}
```

LCA

倍增

输入 N，M，S：树的结点个数、询问的个数和树根结点的序号。

接下来 N-1 行是边，M 行询问最近公共祖先是谁。

```
5 5 4
3 1
2 4
5 1
1 4
2 4
3 2
3 5
1 2
4 5
```

```
int N, M, S;
int fa[maxn][31], dep[maxn];
//fa[i][j]: 第 i 个点的第 2^j 个祖先
vector<int> G[maxn];

void dfs(int root,int fno) {
    fa[root][0] = fno;
    dep[root] = dep[fno] + 1;
    //初始化 fa
    for(int i = 1; i < 31; ++i) {
        fa[root][i] = fa[fa[root][i-1]][i-1];
    }
    //遍历
    for(auto y : G[root]) {
        if(y == fno) continue;
        dfs(y, root);
    }
}

int lca(int x, int y) {
    if(dep[x] > dep[y]) swap(x,y);
    int tem = dep[y] - dep[x];
    for(int i = 0; tem; ++i, tem >>= 1)
    {
```

```
        if(tem&1) y = fa[y][i];
    }
    if(y == x) return x;
    for(int i = 30; i >= 0 && y != x; --i) {
        if(fa[x][i] != fa[y][i]) {
            x = fa[x][i];
            y = fa[y][i];
        }
    }
    return fa[x][0];
}

int main() {
    cin>>N>>M>>S;
    for(int i = 1; i < N; i++) {
        int x, y;
        cin>>x>>y;
        G[x].push_back(y);
        G[y].push_back(x);
    }
    dfs(S, 0);
    for(int i = 1; i <= M; i++) {
        int x, y;
        cin>>x>>y;
        cout<<lca(x,y)<<endl;
    }
    return (0);
}
```

欧拉路径/回路

判别法:

- 对于无向图 G，G 是欧拉图当且仅当 G 是连通的且没有奇度顶点。(欧拉回路
- 对于无向图 G，G 是半欧拉图当且仅当 G 是连通的且 G 中恰有个或个奇度顶点。(欧拉路径
- 对于有向图 G，G 是欧拉图当且仅当 G 的所有顶点属于同一个强连通分量且每个顶点的入度和出度相同。(欧拉回路
- 对于有向图 G，G 是半欧拉图当且仅当：(欧拉路径
  - 如果将 G 中的所有有向边退化为无向边时，那么 G 的所有顶点属于同一个连通分量。
  - 最多只有一个顶点的出度与入度差为 1。
  - 最多只有一个顶点的入度与出度差为 1。
  - 所有其他顶点的入度和出度相同。

题:

无向图找欧拉路径,输出字典序最小的路径。

```
int n, m;
int e[maxn][maxn], du[maxn]; //邻接矩阵 点的度
stack<int> ans; //访问点的顺序

void dfs(int u) {
    for(int i = 1; i <= n; i++) {
        if(e[u][i]) {
            e[u][i]--; e[i][u]--;
            dfs(i);
        }
    }
    ans.push(u);
}

signed main() {
    cin>>m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin>>u>>v;
        n = max(n, u); n = max(n, v);
        e[u][v]++;
        e[v][u]++;
```

```
        du[u]++; du[v]++;
    }
    int s = 1;
    for(int i = 1; i <= n; i++) {
        if(du[i]%2==1) {s = i; break;}
    }
    dfs(s);
    while(ans.size()) {
        cout<<ans.top()<<endl;
        ans.pop();
    }
    return (0);
}
```

强连通分量与 2-SAT

题意:

2*n 个人 m 个关系。每个 2*i-1 和 2*i 之间要选一个。

u 和 v 不能共存。

```
3 2
1 3
2 4
```

输出方案或者不存在则 NIE

```
1
4
5
```

```
//2-SAT
//https://www.luogu.com.cn/problem/P5782
#include<bits/stdc++.h>
using namespace std;

// #define int long long
typedef long long ll;
typedef unsigned long long ull;
#define mst(a) memset(a,0,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcodeforce++)
const ll maxn = 2e5 +7;
const ll maxm = 2e5 +7;
const ll inf = 0x3f3f3f3f;
const ll mod = 100000007;

int n, m;//n 人 1~n(1&2 同党) m 关系
int oth(int x) {return x%2?x+1:x-1;}//另一个人
vector<int> e[maxn];//存边

int dfn[maxn], low[maxn], dfncnt;
int tj_stack[maxn], in_stack[maxn], tp;
int scc[maxn], scc_cnt;
void tarjan(int rt) {
    dfn[rt] = low[rt] = ++dfncnt;
    tj_stack[++tp] = rt, in_stack[rt] = 1;
    for(int i = 0; i < e[rt].size(); i++)
    {
        int y = e[rt][i];
        if(!dfn[y])
        {
            tarjan(y);
            low[rt] = min(low[rt], low[y]);
        }
        else if(in_stack[y]) low[rt] = min(low[rt], dfn[y]);
    }
    if(dfn[rt] == low[rt])
    {
        ++scc_cnt;
```

```
        while(tj_stack[tp] != rt)
        {
            scc[tj_stack[tp]] = scc_cnt;
            in_stack[tj_stack[tp]] = 0;
            tp--;
        }
        scc[tj_stack[tp]] = scc_cnt;
        in_stack[tj_stack[tp]] = 0;
        tp--;
    }
}

void init()
{
    mst(dfn);
    mst(low);
    dfncnt=0;
    tp = 0;
    for(int i = 1; i <= n; i++) e[i].clear();
    mst(scc);
    scc_cnt = 0;
    mst(tj_stack);
    mst(in_stack);
}

int main()
{
    // freopen("D:/c++source file/intxt/in.txt","r",stdin);
    ios :: sync_with_stdio(0);
    cin.tie(0);

    while(cin>>n>>m)
    {
        n *= 2;
        init();

        for(int i = 1; i <= m; i++)
        {
            int u, v;
            cin>>u>>v; //交叉连边
            e[u].push_back(oth(v));
            e[v].push_back(oth(u));
        }
        for(int i = 1; i <= n; i++)
        {
            if(!dfn[i]) tarjan(i); //求全部强连通分量
        }
        int f = 0;
        for(int i = 1; i <= n; i+=2)
        {
            if(scc[i] == scc[oth(i)]) //不存在合法解
            {
                cout<<"NIE"<<endl;
                f = 1;
                break;
            }
        }
        if(f) continue;
        for(int i = 1; i <= n; i+=2) // 输出一个解
        {
            cout<<(scc[i] > scc[oth(i)] ? oth(i) : i)<<endl;
        }
    }
    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
    /*
    先 tarjan 缩点 然后检查行不行 然后直接输出
    */
}
```

拓扑排序

```
int n, m;
int deg[maxn]; //入度
vector<int> e[maxn];
vector<int> ans;

bool toposort() { // 0-indexed
    queue<int> q;
    for(int i = 0; i < n; i++) {
        if(deg[i]==0) q.push(i);
    }
    while(!q.empty()) {
        int u = q.front();
        q.pop();
        ans.push_back(u);
        for(auto y : e[u]) {
            --deg[y];
            if(deg[y]==0) q.push(y);
        }
    }
    if(ans.size()==n) return true;
    else return false;
}
```

```
    cin>>n>>m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin>>u>>v;
        e[u].push_back(v);
        deg[v]++;
    }
    if(toposort()) {
        for(auto y : ans) cout<<y<<endl;
    }
    else {
        cout<<"None"<<endl; //题目DAG必能拓扑排序
    }
```

连通性相关

割点和割边（无向图

tarjan 同时处理割点和桥

```
int n, m;
vector<int> e[maxn];
int dfn[maxn], low[maxn], vis[maxn];
//访问顺序 不过fa最低访问dfn 记录
int iscut[maxn];
int isbridge[maxn], father[maxn];

void tarjan(int u, int fa, int dep) { //割点割边同时处理
    father[u] = fa;
    vis[u] = 1;
    int child = 0;
    low[u] = dfn[u] = dep;
    for(auto v : e[u]) {
        if(v != fa && vis[v] == 1) {
            low[u] = min(low[u], dfn[v]);
        }
        if(vis[v] == 0) {
            tarjan(v, u, dep + 1);
            child++;
```

```
            low[u] = min(low[u], low[v]);
            if( (fa==-1 && child>1) || (fa!=-1 && low[v] >= dfn[u])) {
                iscut[u] = true;
            }
            if(low[v] > dfn[u]) {
                isbridge[v] = true; //割边 会重复所以要isbridge[]记录
            }
        }
    }
    vis[u] = 2;
}
```

割点 main 函数:

```
    cin>>n>>m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin>>u>>v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    for(int i = 0; i < n; i++) {
        if(!dfn[i]) {
            tarjan(i,-1,1);
        }
    }
    for(int i = 0; i < n; i++) {
        if(iscut[i])
            cout<<i<<endl;
    }
```

割边 main 函数:

```
    cin>>n>>m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin>>u>>v;
        e[u].push_back(v);
        e[v].push_back(u);
    }
    for(int i = 0; i < n; i++) {
        if(!dfn[i]) {
            tarjan(i,-1,1);
        }
    }
    vector<pair<int, int>> ans;
    for(int i = 0; i < n; i++) {
        if(isbridge[i]) {
            if(i<father[i])
                ans.push_back(make_pair(i,father[i]));
            else
                ans.push_back(make_pair(father[i],i));
        }
    }
    sort(ans.begin(), ans.end());
    for(auto y : ans) {
        cout<<y.first<<" "<<y.second<<endl;
    }
```

强连通分量缩点（有向图

```
int n, m;
vector<int> e[maxn];//存边

int dfn[maxn], low[maxn], dfncnt;
int tj_stack[maxn], in_stack[maxn], tp;
int scc[maxn], scc_cnt;
void tarjan(int rt) {
    dfn[rt] = low[rt] = ++dfncnt;
    tj_stack[++tp] = rt, in_stack[rt] = 1;
    for(int i = 0; i < e[rt].size(); i++)
    {
        int y = e[rt][i];
        if(!dfn[y])
        {
            tarjan(y);
            low[rt] = min(low[rt], low[y]);
        }
        else if(in_stack[y]) low[rt] = min(low[rt], dfn[y]);
    }
    if(dfn[rt] == low[rt])
    {
        ++scc_cnt;
        while(tj_stack[tp] != rt)
        {
            scc[tj_stack[tp]] = scc_cnt;
            in_stack[tj_stack[tp]] = 0;
            tp--;
        }
        scc[tj_stack[tp]] = scc_cnt;
        in_stack[tj_stack[tp]] = 0;
        tp--;
    }
}
```

```
void init()
{
    mst(dfn);
    mst(low);
    dfncnt=0;
    tp = 0;
    for(int i = 1; i <= n; i++) e[i].clear();
    mst(scc);
    scc_cnt = 0;
    mst(tj_stack);
    mst(in_stack);
}
```

```
//例题 判断是否在同一个强连通分量
    cin>>n>>m;
    for(int i = 1; i <= m; i++) {
        int u, v; cin>>u>>v;
        e[u].push_back(v);
    }
    for(int i = 0; i < n; i++) {
        if(!dfn[i]) tarjan(i);
    }
    int q; cin>>q;
    for(int i = 1; i <= q; i++) {
        int a, b; cin>>a>>b;
        cout<<(scc[a]==scc[b])<<endl;
    }
```

最小生成树

```
//唯一
struct edge{
```

```
    int from, to, val;
}e[maxn];
int n, m, ans;
int fa[maxn];
bool cmp(edge a, edge b){return a.val < b.val;}
void init()
{
    for(int i = 0; i <= n; i++)
    {
        fa[i] = i;
    }
    ans = 0;
}
int find(int x) {return fa[x]==x?x:(fa[x] = find(fa[x]));}
bool uniKruskal()
{
    int sum1 = 0, sum2 = 0;//已使用的 可能使用的
    int p = 0;//相同指针
    int flag = 0, num = 0;
    for(int i = 1; i <= m + 1; i++)
    {
        if(p<i)
        {
            if(sum1 != sum2)
            {
                flag = 1;
                break;
            }
            sum1 = 0, sum2 = 0;
            for(int j = i; j <= m+1; j++)
            {
                if(e[j].val != e[i].val)
                {
                    p = j-1;
                    break;
                }
                if(find(e[j].from) != find(e[j].to))
                ++sum2;
            }
        }
        if(i>m) break;
        int x = find(e[i].from);
        int y = find(e[i].to);
        if(x != y && num != n-1)
        {
            num++;
            sum1++;
            // merge(x, y);
            fa[x] = fa[y];
            ans += e[i].val;
        }
    }
    if(flag) return false;
    else return true;
}

int main()
{
    freopen("D:/c++source file/intxt/in.txt","r",stdin);
    // ios :: sync_with_stdio(0);
    // cin.tie(0);
    cf
    {
        cin>>n>>m;
        init();
        for(int i = 1; i <= m; i++) cin>>e[i].from>>e[i].to>>e[i].val;
        sort(e+1,e+1+m, cmp);
        if(uniKruskal()) cout<<ans<<endl;
        else cout<<"Not Unique!"<<endl;
    }

```

```
74
75      //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
76      return (0);
77  }
```

杂项

前向星存边

```
int n;
int head[maxn], cnt = 0;
struct edge{
    int to;
    int w;
    int next;
}e[maxn<<2];

void add(int u, int v, int w) {
    e[++cnt].to = v;
    e[cnt].w = w;
    e[cnt].next = head[u];
    head[u] = cnt;
}

void iterate_edge(int u) { //遍历u的点
    for(int i = head[u]; i; i=e[i].next) {

    }
}
```

计算几何

几角排序

```
struct point{
    int x, y;
    double k;
    void calk() {k = atan(1.0*y/x);}
    bool operator < (const point b) const {
        if(x == 0) {
            if(b.x > 0) {return true;}
            else if(b.x == 0) {return ( (b.y<0) > (y<0) );}
            else if(b.x < 0) {return y > 0;}
        }
        else if(b.x == 0) {
            if(x > 0) {return false;}
            else if(x == 0) {return ( (b.y<0) > (y<0) );}
            else if(x < 0) {return b.y < 0;}
        }
        else { // 都不是0
            if(x*b.x<0) return x < b.x; //两边
            //else return y / x < b.y / b.x;
            else return y * b.x < b.y * x;
            //else return k<b.k;
        }
        return 1;
    }
```

```
} a[maxn];
```

基本

```
#include<bits/stdc++.h>
using namespace std;

#define EPS (1e-10)
#define int long long
//#define lson (rt<<1)
//#define rson ((rt<<1)+1)
//#define mid ((l+r)>>1)
#define mst(a) memset(a,0,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcod
typedef long long ll;
const ll maxn = 2e5 +7;
const ll maxm = 2e5 +7;
const ll inf = 0x3f3f3f3f;
const ll mod = 1e9 +7;
//////////////////////////

#define EPS (1e-10)
#define equals(a, b) (fabs((a)-(b)) < EPS)

//点和向量
class Point{
public:
    double x, y;

    Point(double x = 0, double y = 0) : x(x), y(y) {}

    Point operator + (Point p) {return Point(x+p.x,y+p.y);}
    Point operator - (Point p) {return Point(x-p.x,y-p.y);}
    Point operator * (double a) {return Point(a*x,a*y);}
    Point operator / (double a) {return Point(x/a,y/a);}

    double abs() {return sqrt(norm());}
    double norm() {return x * x + y * y;}

    bool operator < (const Point &p) const {
        return x != p.x ? x < p.x : y < p.y;
    }

    bool operator == (const Point &p) const {
        return fabs(x - p.x) < EPS && fabs(y - p.y) < EPS;
    }
    void ShowPoint() {cout<<x<<" "<<y<<endl;}
};
typedef Point Vector;

// 线段和线
struct Segment{
    Point p1, p2;
};
typedef Segment Line;

//圆和多边形
```

```
class Circle {
public:
    Point c;
    double r;
    Circle(Point c = Point(), double r = 0.0) : c(c), r(r){}
};

typedef vector<Point> Polygon;

//函数

double dot(Vector a, Vector b) { //点乘
    return a.x * b.x + a.y * b.y;
}
double cross(Vector a, Vector b) {
    return a.x*b.y - a.y*b.x;
}


//////////////////////////
int n;

signed main() {
    #ifdef moyi_qwq
        freopen("D:/source file/intxt/in.txt","r",stdin);
    #endif




    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

平面最近点对（分治）

```
//https://onlinejudge.u-aizu.ac.jp/courses/library/4/CGL/5/CGL_5_A
//平面最近点对 分治
#include<bits/stdc++.h>
using namespace std;

#define EPS (1e-10)
#define int long long
//#define lson (rt<<1)
//#define rson ((rt<<1)+1)
//#define mid ((l+r)>>1)
#define mst(a) memset(a,0,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcod
typedef long long ll;
const ll maxn = 2e5 +7;
const ll maxm = 2e5 +7;
const ll inf = 0x3f3f3f3f;
const ll mod = 1e9 +7;

int n;

struct point{
```

```
    double x, y;
    int id;
};

bool cmp_x(point a, point b) {
    if(a.x==b.x) return a.y < b.y;
    else return a.x<b.x;}
bool cmp_y(point a, point b) {
    if(a.y==b.y) return a.x<b.x;
    else return a.y<b.y;}

vector<point> a;
double mindist = 1e20;
int ansa, ansb;
void upd_ans(const point &a, const point &b) {
    double dist = sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y) + 0.0);
    if(dist < mindist) {
        mindist = dist; ansa = a.id, ansb = b.id;
    }
}
void csp(int l, int r) { //求[l, r]的最近点对
    if(r - l <= 3) {
        for(int i = l; i <= r; i++) {
            for(int j = i + 1; j <= r; j++) upd_ans(a[i], a[j]);
        }
        sort(a.begin()+l,a.begin()+r+1,cmp_y);
        return ;
    }

    int m = (r+l) / 2;
    double midx = a[m].x;
    csp(l,m); csp(m+1,r);
    inplace_merge(a.begin()+l,a.begin()+m+1,a.begin()+r+1, cmp_y);

    static point t[maxn];
    int tsz = 0;
    for (int i = l; i <= r; ++i) {
        if (abs(a[i].x - midx) < mindist) {
            for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y < mindist; --j) {
                upd_ans(a[i], t[j]);
            }
            t[tsz++] = a[i];
        }
    }
}

signed main() {
    #ifdef moyi_qwq
        freopen("D:/source file/intxt/in.txt","r",stdin);
    #endif

    cin>>n;
    for(int i = 0; i < n; i++) {
        point p; cin>>p.x>>p.y; p.id = i; a.push_back(p);
    }
    sort(a.begin(),a.begin()+n,cmp_x);
```

```
    csp(0,n-1);
    printf("%.10lf\n",mindist);

    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

全板子

```
#include<bits/stdc++.h>
using namespace std;

//#define lson (rt<<1)
//#define rson ((rt<<1)+1)
//#define mid ((l+r)>>1)
#define mst(a) memset(a,0,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcod
typedef long long ll;
const ll maxn = 2e5 +7;
const ll maxm = 2e5 +7;
const ll inf = 0x3f3f3f3f;
const ll mod = 1e9 +7;
////////////////////////////////////////////////////////////
// COUNTER CLOCKWISE
static const int CCW_COUNTER_CLOCKWISE = 1;
static const int CCW_CLOCKWISE = -1;
static const int CCW_ONLINE_BACK = 2;
static const int CCW_ONLINE_FRONT = -2;
static const int CCW_ON_SEGMENT = 0;

//Intercsect Circle & Circle
static const int ICC_SEPERATE = 4; // 不相交
static const int ICC_CIRCUMSCRIBE = 3; // 外接
static const int ICC_INTERSECT = 2; //相交
static const int ICC_INSCRIBE = 1; //内切
static const int ICC_CONTAIN = 0; //包含

#define EPS (1e-10)
#define equals(a, b) (fabs((a)-(b)) < EPS)

//点和向量
class Point{
public:
    double x, y;

    Point(double x = 0, double y = 0) : x(x), y(y) {}

    Point operator + (Point p) {return Point(x+p.x,y+p.y);}
    Point operator - (Point p) {return Point(x-p.x,y-p.y);}
    Point operator * (double a) {return Point(a*x,a*y);}
    Point operator / (double a) {return Point(x/a,y/a);}

    double abs() {return sqrt(norm());}
    double norm() {return x * x + y * y;}

    bool operator < (const Point &p) const {
        return x != p.x ? x < p.x : y < p.y;
```

```
    }

    bool operator == (const Point &p) const {
        return fabs(x - p.x) < EPS && fabs(y - p.y) < EPS;
    }
    void ShowPoint() {cout<<x<<" "<<y<<endl;}
};
typedef Point Vector;

double norm(Vector a) {
    return a.x * a.x + a.y * a.y;
}
double abs(Vector a) {
    return sqrt(norm(a));
}
// 线段和线
struct Segment{
    Point p1, p2;
    Segment() {}
    Segment(Point a, Point b): p1(a), p2(b) {}
};
typedef Segment Line;

//圆和多边形
class Circle {
public:
    Point c;
    double r;
    Circle(Point c = Point(), double r = 0.0) : c(c), r(r){}
};

typedef vector<Point> Polygon;

////////////////函数//////////////////////////////////////////

double dot(Vector a, Vector b) { //点乘
    return a.x * b.x + a.y * b.y;
}

double cross(Vector a, Vector b) {//叉乘
    return a.x*b.y - a.y*b.x;
}

Point project(Segment s, Point p) { // 投影
    Vector base = s.p2 - s.p1;
    double r = dot(p-s.p1, base) / base.norm();
    return s.p1 + base * r;
}

Point reflect(Segment s, Point p) { // 对称点
    return p + (project(s,p)-p) * 2.0;
}

double arg(Vector p) {return atan2(p.y, p.x);} // 角度
Vector polar(double a, double r) {return Point(cos(r) * a, sin(r) * a);} // 极坐标转直角坐标
```

```
int ccw(Point p0, Point p1, Point p2) { // p0-p1和p0-p2的方向 顺逆时针 大小
    Vector a = p1 - p0;
    Vector b = p2 - p0;
    //a.Show(); b.Show(); cout<<"|||"<<endl;
    if(cross(a, b) > EPS) {return CCW_COUNTER_CLOCKWISE;} // COUNTER_CLOCKWISE
    if(cross(a, b) < -EPS) {return CCW_CLOCKWISE;} // CLOCKWISE
    if(dot(a, b) < -EPS) {return CCW_ONLINE_BACK;} // ONLINE_BACK
    if(a.norm() < b.norm()) {return CCW_ONLINE_FRONT;} // ONLINE_FRONT
    if(a.norm() > b.norm()) {return CCW_ON_SEGMENT;} // ON_SEGMENT

    return 0;
}
bool convexPolygon(Polygon p) { //凸多边形
    int n = p.size();
    for(int i = 0; i < n; i++) {
        Vector v1, v2;
        v1 = p[(i+1)%n] - p[i];
        v2 = p[(i+2)%n] - p[(i+1)%p.size()];
        if(cross(v1,v2) < 0) return false;
    }
    return true;
}
bool intersect(Point p1, Point p2, Point p3, Point p4) { //p1-p2 p3-p4相交
    return (ccw(p1, p2, p3) * ccw(p1, p2, p4) <= 0 &&
            ccw(p3, p4, p1) * ccw(p3, p4, p2) <= 0);
}
bool intersect(Segment s1, Segment s2) { // 线段相交
    return intersect(s1.p1, s1.p2, s2.p1, s2.p2);
}
Point getCrossPoint(Segment s1, Segment s2) { // 线段交点
    Vector base = s2.p2 - s2.p1;
    double d1 = abs(cross(base, s1.p1 - s2.p1));
    double d2 = abs(cross(base, s1.p2 - s2.p1));
    double t = d1 / (d1 + d2);
    return s1.p1 + (s1.p2 - s1.p1) * t;
}
pair<Point, Point> getCrossPoints(Circle c, Line l) { //线和圆交点
    //assert(intersect(c, l));
    Vector pr = project(l,c.c);
    Vector e = (l.p2 - l.p1) / abs(l.p2 - l.p1);
    double base = sqrt(c.r * c.r - norm(pr - c.c));
    return make_pair(pr + e * base, pr - e * base);
}
pair<Point, Point> getCrossPoints(Circle c1, Circle c2) { // 圆和圆交点
    double d = abs(c1.c - c2.c);
    double cosv = (c1.r*c1.r+d*d-c2.r*c2.r) / (2*c1.r*d);
    if( abs(abs(cosv) - 1) < EPS ) cosv = 1.0 * (cosv < 0 ? -1 : 1);
    double a = acos(cosv);
    double t = arg(c2.c - c1.c);
    return make_pair(c1.c + polar(c1.r, t+a), c1.c + polar(c1.r, t-a));
}
Point getCrossPointLL(Line l1,Line l2){ //直线交点
  double a=cross(l1.p2-l1.p1,l2.p2-l2.p1);
  double b=cross(l1.p2-l1.p1,l1.p2-l2.p1);
  if(abs(a)<EPS&&abs(b)<EPS) return l2.p1; //共线
  return l2.p1+(l2.p2-l2.p1)*(b/a);
```

```
}
int contains(Polygon g, Point p) { // 多边形包含点 IN-2 ON-1 OUT-0
    int n = g.size();
    bool x = false;
    for(int i = 0; i < n; i++) {
        Point a = g[i] - p, b = g[(i+1)%n] - p;
        if( abs(cross(a, b)) < EPS && dot(a, b) < EPS) return 1;
        if( a.y > b.y ) swap(a, b);
        if( a.y < EPS && EPS < b.y && cross(a, b) > EPS) x = !x;
    }
    return (x ? 2 : 0);
}
Polygon andrewScan(Polygon s) { //凸包
    Polygon u, l;
    if(s.size() < 3) return s;
    sort(s.begin(), s.end());
    // x最小的加到u
    u.push_back(s[0]);
    u.push_back(s[1]);
    // x最大的加到l
    l.push_back(s[s.size()-1]);
    l.push_back(s[s.size()-2]);

    //上部
    for(int i = 2; i < int(s.size()); i++) { // 注意ccw规定了是否取边上多的点
        for(int n = u.size(); n >= 2 && ccw(u[n-2], u[n-1], s[i]) == 1; n--) {
            u.pop_back();
        }
        u.push_back(s[i]);
    }
    //下部
    for(int i = s.size() - 3; i >= 0; i--) { // 注意ccw规定了是否取边上多的点
        for(int n = l.size(); n >= 2 && ccw(l[n-2], l[n-1], s[i]) == 1; n--) {
            l.pop_back();
        }
        l.push_back(s[i]);
    }

    // 从左下开始的顺时针生成序列
    reverse(l.begin(), l.end());
    for(int i = u.size() - 2; i >= 1; i--) {l.push_back(u[i]);}
    // 返回逆时针的序列
    return l;
}
Polygon convexCut(Polygon p, Line l) { // 切凸多边形返回左侧
    Polygon q;
    int n = p.size();
    for(int i = 0; i < n; i++) {
        if(ccw(l.p1, l.p2, p[i]) != -1) {
            q.push_back(p[i]);
        }
        if(ccw(l.p1, l.p2, p[i])*ccw(l.p1, l.p2, p[(i+1)%n]) < 0) {
            Line tem; tem.p1 = p[i]; tem.p2 = p[(i+1)%n];
            q.push_back(getCrossPointLL(l, tem));
        }
    }
```

```
    return q;
}
//////////////////距离//////////////////////////////////////////
double getDistance(Point a, Point b) { // 两点距离
    return (a-b).abs();
}
double getDistanceLP(Line l, Point p) { //点和直线距离
    return abs(cross(l.p2-l.p1,p-l.p1) / abs(l.p2-l.p1));
}
double getDistanceSP(Segment s, Point p) { //点和线段距离
    if ( dot(s.p2 - s.p1, p - s.p1) < 0.0 ) return (p - s.p1).abs();
    if( dot(s.p1 - s.p2, p - s.p2) < 0.0 ) return (p - s.p2).abs();
    return getDistanceLP(s, p);
}
double getDistance(Segment s1, Segment s2) { //线段和线段距离
    if( intersect(s1, s2) ) return 0.0;
    return min(
        min(getDistanceSP(s1, s2.p1), getDistanceSP(s1, s2.p2)),
        min(getDistanceSP(s2, s1.p1), getDistanceSP(s2, s1.p2))
    );
}
double getAreaPolygon(Polygon v) { // 多边形面积 点按逆时针给出
    double res = 0.0;
    int len = v.size();
    for(int i = 0; i < len; i++) {
        res += cross(v[i], v[(i+1)%len]) / 2.0;
    }
    return res;
}
double diameter(Polygon s) { //求给定凸包的直径
    Polygon p = s;
    int n = p.size();
    if(n == 2) return abs(s[0]-s[1]);
    // i最右上点 j最左下点
    int i = 0, j = 0;
    for(int k = 0; k < n; k++) {
        if(p[i] < p[k]) i = k;
        if(!(p[j]<p[k])) j = k;
    }
    // 旋转卡壳
    double res = 0;
    int si = i, sj = j; // 记录起点
    while(i != sj || j != si) {
        res = max(res, abs(p[i] - p[j]));
        if(cross(p[(i+1)%n]-p[i],p[(j+1)%n]-p[j]) < 0.0) { // 旋转
            i = (i+1) % n;
        }
        else {
            j = (j+1) % n;
        }
    }
    return res;
}
//////////////////////////圆//////////////////////////////////////
int intersectCC(Circle c1, Circle c2) { //圆相交
    if(c1.r<c2.r) swap(c1,c2);
```

```
    double d = abs(c1.c-c2.c);
    double r = c1.r+c2.r;
    if(d == r) {return ICC_CIRCUMSCRIBE;}
    if(d > r) {return ICC_SEPERATE;}
    if(d+c2.r==c1.r) {return ICC_INSCRIBE;}
    if(d+c2.r<c1.r) {return ICC_CONTAIN;}

    return ICC_INTERSECT;
}
Circle getIncircle(Point pa, Point pb, Point pc){ //内切圆
  Vector v1 = pb - pa;
  Vector v2 = pc - pa;
  Vector v3 = pc - pb;
  Vector v4 = pa - pb;
  Vector c1 = polar(10000.0, (arg(v1) + arg(v2))/2);
  Vector c2 = polar(10000.0, (arg(v3) + arg(v4))/2);
  Point x = getCrossPointLL(Segment(pa, pa + c1), Segment(pb, pb + c2));
  double r = getDistanceLP(Segment(pa, pb), x);
  return Circle(x, r);
}
Circle getExcircle(Point pa, Point pb, Point pc){ //外接圆
  Vector v1 = (pb - pa);
  Vector v2 = (pc - pa);
  Vector v1r = Vector(-v1.y, v1.x);
  Vector v2r = Vector(-v2.y, v2.x);
  Point c1 = pa + v1/2;
  Point c2 = pa + v2/2;
  Point x = getCrossPointLL(Segment(c1, c1 + v1r), Segment(c2, c2 + v2r));
  double r = getDistance(pa, x);
  return Circle(x, r);
}
pair<Point, Point> Tangency(Circle c, Point p) { // 求圆点切线的交点 前提是有切点
    double dis = norm(p-c.c) - c.r*c.r;
    double r2 = sqrt (dis);
    Circle c2; c2.c = p; c2.r = r2;
    pair<Point, Point> res =  getCrossPoints(c, c2);
    return res;
}
vector<Point> getContact(Circle C1,Circle C2){ // 求两个圆公切线的交点 在C1上的

    vector<Point> ret;

    double p = C2.c.x-C1.c.x;
    double q = C2.c.y-C1.c.y;

    double A = p*p+q*q;

    double n_1 = (q*C1.r*(C1.r+C2.r)-p*C1.r*sqrt(A-(C1.r+C2.r)*(C1.r+C2.r)))/(A)+C1.c.y;
    double n_2 = (q*C1.r*(C1.r+C2.r)+p*C1.r*sqrt(A-(C1.r+C2.r)*(C1.r+C2.r)))/(A)+C1.c.y;

    double m_1 = (p*C1.r*(C1.r+C2.r)+q*C1.r*sqrt(A-(C1.r+C2.r)*(C1.r+C2.r)))/(A)+C1.c.x;
    double m_2 = (p*C1.r*(C1.r+C2.r)-q*C1.r*sqrt(A-(C1.r+C2.r)*(C1.r+C2.r)))/(A)+C1.c.x;

    if(A-(C1.r+C2.r)*(C1.r+C2.r) >= 0){
        ret.push_back(Point(m_1,n_1));
        ret.push_back(Point(m_2,n_2));
```

```
    }

    double n_3 = (q*C1.r*(C1.r-C2.r)-p*C1.r*sqrt(A-(C1.r-C2.r)*(C1.r-C2.r)))/(A)+C1.c.y;
    double n_4 = (q*C1.r*(C1.r-C2.r)+p*C1.r*sqrt(A-(C1.r-C2.r)*(C1.r-C2.r)))/(A)+C1.c.y;

    double m_3 = (p*C1.r*(C1.r-C2.r)+q*C1.r*sqrt(A-(C1.r-C2.r)*(C1.r-C2.r)))/(A)+C1.c.x;
    double m_4 = (p*C1.r*(C1.r-C2.r)-q*C1.r*sqrt(A-(C1.r-C2.r)*(C1.r-C2.r)))/(A)+C1.c.x;

    if(A-(C1.r-C2.r)*(C1.r-C2.r) >= 0){
        ret.push_back(Point(m_3,n_3));
        ret.push_back(Point(m_4,n_4));
    }

    return ret;
}
////////////////////////////////////////////////////////////////////////////
int n;

signed main() {
    #ifdef moyi_qwq
        freopen("D:/source file/intxt/in.txt","r",stdin);
    #endif


    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

字符串

manacher

```
namespace Manacher {
      //记得改 maxn 要开两倍
      char Manacher_s[maxn];
      int d[maxn];//变换后的字符串回文半径
      int maxlen;//最长回文串长度
      void manacher(char s[], int n) {
            memset(d,0,sizeof(d));
            int cnt = 0;
            Manacher_s[cnt++] = '#';
            for(int i = 0; i < n; i++) {Manacher_s[cnt++]=s[i];Manacher_s[cnt++]='#';}
            n = strlen(Manacher_s);
            int r = 0, p = 0;
            for(int i = 0; i < n; i++) {
                  if(i < r) d[i] = min(d[2*p-i], r-i);
                  else d[i] = 1;
                  while(i-d[i]>=0&&i+d[i]<n&&Manacher_s[i-d[i]]==Manacher_s[i+d[i]]) d[i]++;
                  if(d[i]+i-1>r) {r=d[i]+i-1;p=i;}
            }
            for(int i = 0; i < n; i++) d[i]--;
            maxlen = 0;
            for(int i = 0; i < n; i++) {
                  if(i&1) maxlen = max(maxlen, (d[i]>>1)*2+1);
                  else maxlen = max(maxlen, d[i]);
            }
      }
}
```

哈希

内置了自动双哈希开关（小心 TLE）。

```cpp
#include<bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef unsigned long long ull;
#define mst(a,x) memset(a,x,sizeof(a))
#define cf int Tcodeforces, Tcodeforce;cin>>Tcodeforces;for(Tcodeforce = 1; Tcodeforce <= Tcodeforces; Tcodeforce++)
const ll maxn = 2e5 +7;
const ll inf = 0x3f3f3f3f;
const ll mod = 100000007;

namespace HString
{
    //标号从零开始
    const int N = 1e3 +7;
    //字符数组长度
    const int x = 135;
    const int p1 = 1e9 + 7, p2 = 1e9 + 9;
    ull xp1[N], xp2[N], xp[N];

    void init_xp()
    {
        xp1[0] = xp2[0] = xp[0] = 1;
        for(int i = 1; i < N; i++)
        {
            xp1[i] = xp1[i - 1] * x % p1;
            xp2[i] = xp2[i - 1] * x % p2;
            xp[i] = xp[i - 1] * x;
        }
    }

    struct HashString
    {
        char s[N];//文本串
        int length, subsize;//长度为 length 的子串 子串数组的 size
        bool sorted;
        //h[i] 是 i 到尾部的串的哈希值
        ull h[N], hl[N];//h 是 i 到尾部的哈希值 hl 是子串哈希值们
        //用 char 数组 t 初始化结构体 返回哈希值
        ull init(const char *t)
        {
            if(xp[0] != 1) init_xp();
            length = strlen(t);
            strcpy(s, t);
            ull res1 = 0, res2 = 0;
            h[length] = 0;
            for(int j = length - 1; j >= 0; j--)
            {
            #ifdef ENABLE_DOUBLE_HASH
                res1 = (res1 * x + s[j]) % p1;
                res2 = (res2 * x + s[j]) % p2;
                h[j] = (res1 << 32) | res2;
            #else
                res1 = res1 * x + s[j];
                h[j] = res1;
            #endif
            }
            return h[0];
        }

        //获取子串哈希, 左闭右开
        ull get_substring_hash(int left, int right)  {
            int len = right - left;
        #ifdef ENABLE_DOUBLE_HASH
            unsigned int mask32 = ~(0u);
            ull left1 = h[left] >> 32, right1 = h[right] >> 32;
            ull left2 = h[left] & mask32, right2 = h[right] & mask32;
            return (((left1 - right1 * xp1[len] % p1 + p1) % p1) << 32) |
                   (((left2 - right2 * xp2[len] % p2 + p2) % p2));
        #else
            return h[left] - h[right] * xp[len];
```

```
        #endif
        }
        //获得长度为 sublen 的子串 存到 hl
        void get_all_subs_hash(int sublen) {
            subsize = length - sublen + 1;
            for (int i = 0; i < subsize; ++i)
                hl[i] = get_substring_hash(i, i + sublen);
            sorted = 0;
        }
        //排序
        void sort_substring_hash() {
            sort(hl, hl + subsize);
            sorted = 1;
        }
        //从子串中查找哈希值 key
        //必须先排序
        bool match(ull key) const {
            // if (!sorted) assert (0);
            if (!subsize) return false;
            return binary_search(hl, hl + subsize, key);
        }
    };
}
int main()
{
    //freopen("D:/c++source file/intxt/in.txt","r",stdin);


    //cerr<<"Time : "<<1000*((double)clock())/(double)CLOCKS_PER_SEC<<"ms";
    return (0);
}
```

KMP

```
namespace Kmp {
    //下标从 0 开始
    int nxt[maxn];//前缀函数值
    int cnt = 0;//可以匹配的点的数量
    vector<int> res;//可以匹配的点的下标
    void kmp(int a[], char s[], int n) {
        int j = a[0] = 0;
        for(int i = 1; i < n; i++) {
            while(j && s[i] != s[j]) j = a[j-1];
            a[i] = j += s[i] == s[j];
        }
    }
    void compare(char s[], char mode[], int la, int lb) {
        res.clear(); memset(nxt, 0, sizeof(nxt));
        kmp(nxt, mode, lb);
        int j = 0;
        for(int i = 0; i < la; i++) {
            while(j && s[i] != mode[j]) j = nxt[j-1];
            if(s[i] == mode[j]) j++;
            if(j == lb) res.push_back(i-j+1);
        }
        cnt = res.size();
    }
};
```

- 前缀函数（每一个前缀的最长 border）

```
void get_pi(int a[], char s[], int n) {
    int j = a[0] = 0;
    FOR (i, 1, n) {
        while (j && s[i] != s[j]) j = a[j - 1];
        a[i] = j += s[i] == s[j];
    }
}
```

- Z 函数（每一个后缀和该字符串的 LCP 长度）

```
void get_z(int a[], char s[], int n) {
```

```
    int l = 0, r = 0; a[0] = n;
    FOR (i, 1, n) {
        a[i] = i > r ? 0 : min(r - i + 1, a[i - l]);
        while (i + a[i] < n && s[a[i]] == s[i + a[i]]) ++a[i];
        if (i + a[i] - 1 > r) { l = i; r = i + a[i] - 1; }
    }
}
```

Trie

```
namespace trie {
    int t[N][26], sz, ed[N];
    void init() { sz = 2; memset(ed, 0, sizeof ed); }
    int _new() { memset(t[sz], 0, sizeof t[sz]); return sz++; }
    void ins(char* s, int p) {
        int u = 1;
        FOR (i, 0, strlen(s)) {
            int c = s[i] - 'a';
            if (!t[u][c]) t[u][c] = _new();
            u = t[u][c];
        }
        ed[u] = p;
    }
}
```

AC 自动机

```
const int N = 1e6 + 100, M = 26;

int mp(char ch) { return ch - 'a'; }

struct ACA {
    int ch[N][M], danger[N], fail[N];
    int sz;
    void init() {
        sz = 1;
        memset(ch[0], 0, sizeof ch[0]);
        memset(danger, 0, sizeof danger);
    }
    void insert(const string &s, int m) {
        int n = s.size(); int u = 0, c;
        FOR (i, 0, n) {
            c = mp(s[i]);
            if (!ch[u][c]) {
                memset(ch[sz], 0, sizeof ch[sz]);
                danger[sz] = 0; ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        danger[u] |= 1 << m;
    }
    void build() {
        queue<int> Q;
        fail[0] = 0;
        for (int c = 0, u; c < M; c++) {
            u = ch[0][c];
            if (u) { Q.push(u); fail[u] = 0; }
        }
        while (!Q.empty()) {
            int r = Q.front(); Q.pop();
            danger[r] |= danger[fail[r]];
            for (int c = 0, u; c < M; c++) {
                u = ch[r][c];
                if (!u) {
                    ch[r][c] = ch[fail[r]][c];
                    continue;
                }
                fail[u] = ch[fail[r]][c];
                Q.push(u);
            }
```

```
        }
    }
} ac;

char s[N];

int main() {
    int n; scanf("%d", &n);
    ac.init();
    while (n--) {
        scanf("%s", s);
        ac.insert(s, 0);
    }
    ac.build();

    scanf("%s", s);
    int u = 0; n = strlen(s);
    FOR (i, 0, n) {
        u = ac.ch[u][mp(s[i])];
        if (ac.danger[u]) {
            puts("YES");
            return 0;
        }
    }
    puts("NO");
    return 0;
}
```

杂项

STL

`vector<pair<int, int>> ans;`
`sort(ans.begin(), ans.end());`的时候会优先按照第一关键字排序

21.1.5 输出写 printf