# 算法模板



# 基础

## 预处理

```
#include<bits/stdc++.h>
#define int long long
#define endl '\n'
#define fst first
#define snd second

using namespace std;

signed main() {
    cin.tie(nullptr) -> sync_with_stdio(false);
    return 0;
}
```

# 链式前向星存图

```
// ！记得初始化!
// idx、head[N]……视情况而定
void add(int u, int v) {
    ++ idx; //边的序号>=1
    to[idx] = v; //当前这条边去向的点
    nxt[idx] = head[u];//这条边循环遍历的下一条边序号是u连出去的上一条
    head[u] = idx;//更新u连出去的最新一条
}
```

# 二分

```
lower_bound(a.begin(), a.end(), x) //返回容器a中第一个大于等于x的元素的地址
upper_bound(a.begin(), a.end(), x) //返回容器a中第一个大于x的元素的地址
//增加修改器后
upper_bound(a.begin(), a.end(), x, greater<int>() ) //返回容器a中第一个小于x的元素的地址

int l = 0, r = maxn + 1;//保证初始情况的l满足条件，r不满足条件
while(r - l > 1) {
    int mid = (l + r) >> 1;
    if(check(mid)) l = mid;
    else r = mid;
}
cout << l << endl;

int l = 0, r = maxn + 1;
while(r - l > 1) {
    int mid = (l + r) >> 1;
    if(check(mid)) r = mid;
    else l = mid;
}
cout << r << endl;
```

# 数据结构

## 并查集

```
int find(int x) {
    return p[x] == x ? x : p = find(p[x]);
}
int unite(int x, int y) {
    x = find(x), y = find(y);
    if(x == y) return;
    p[y] = x;
    // p[find(x)] = find(y);
}
```

## 带权并查集

```
int find(int x) {
    return p[x] == x ? x : p = find(p[x]);
    // if(x != p[x]) return p[x] = find(p[x]);
    // return p[x];
}
void unite(int x, int y) {
    x = find(x), y = find(y);
    if(x == y) return;
    if(siz[x] < siz[y]) swap(x, y);
    pa[y] = x;
    siz[x] += siz[y]; //块的大小
    sum[x] += sum[y]; //块的总和，视情况而定
}

void move(int x, int y) { //将点x移动到y的块上
    auto fx = find(x), fy = find(y);
    if(fx == fy) return;
    pa[x] = fy;
    --siz[fx], ++ siz[fy]; //块的大小
    sum[fx] -= x, sum[fy] += x; //块的总和，视情况而定
}
```

# 最大子段和

## 最大子段和

```
int main() {
    int n, x, cur = -1e9, ans = -1e9;
    cin >> n;
    for(int i = 1; i <= n; i ++) {
        cin >> x;
        cur = max(x, cur + x); //滚动cur可不开xl数组
        ans = max(ans, cur);
        // xl[i] = max(x, xl[i - 1] + x); //记录长度为i的最大子段和
        // ans = max(ans, xl[i]);
    }
    cout << ans << endl;
    return 0;
}
```

高精度加法

#include<bits/stdc++.h>

using namespace std;

vector add(vector &A, vector &B) {

vector C;

if(A.size() < B.size()) return add(B, A);

int t = 0;

for(int i = 0; i < A.size(); i ++) {

t += A[i];

if(i < B.size()) t += B[i];

C.push_back(t % 10);

t /= 10;

}

if(t) C.push_back(1);

return C;

}

```
signed main() {
    cin.tie(nullptr) -> sync_with_stdio(false);

    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for(int i = a.size() - 1; i >= 0; i --) A.push_back(a[i] - '0');
    for(int i = b.size() - 1; i >= 0; i --) B.push_back(b[i] - '0');
    auto C = add(A, B);
    for(int i = C.size() - 1; i >= 0; i --) cout << C[i];
    cout << '\n';
    return 0;
}
```

高精度减法：

#include<bits/stdc++.h>

using namespace std;

```
bool cmp(vector<int> &A, vector<int> &B) {
    if(A.size() != B.size()) return A.size() > B.size();
    for(int i = A.size(); i >= 0; i --) {
        if(A[i] != B[i]) return A[i] > B[i];
    }
    return true;
}
vector<int> sub(vector<int> &A, vector<int> &B) {
    vector<int> C;

    for(int i = 0, t = 0; i < A.size(); i ++) {
        t = A[i] - t;
        if(i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if(t < 0) t = 1;
        else t = 0;
    }
    while(C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
int main() {
    cin.tie(nullptr) -> sync_with_stdio(false);

    string a, b;
    vector<int> A, B;
    cin >> a >> b;
    for(int i = a.size() - 1; i >= 0; i --) A.push_back(a[i] - '0');
    for(int i = b.size() - 1; i >= 0; i --) B.push_back(b[i] - '0');
    if(cmp(A, B)) {
        auto C = sub(A, B);
        for(int i = C.size() - 1; i >= 0; i --) cout << C[i];
    }
    else {
        auto C = sub(B, A);
        cout << '-';
        for(int i = C.size() - 1; i >= 0; i --) cout << C[i];
    }
    return 0;
}
```

高精度乘法(大数乘小数)：

#include<bits/stdc++.h>
using namespace std;
vector mul(vector &A, int b) {
if(!b) return {0};
vector C;
int t = 0;
for(int i = 0; i < A.size() || t; i ++) {
if(i < A.size()) t += A[i] * b;
C.push_back(t % 10);
t /= 10;
}
return C;
}

```
int main() {
    cin.tie(nullptr) -> sync_with_stdio(false);

    string a;
    int b;
    cin >> a >> b;
    vector<int> A;
    for(int i = a.size() - 1; i >= 0; i --) A.push_back(a[i] - '0');
    vector<int> C = mul(A, b);
    for(int i = C.size() - 1; i >= 0; --i) cout << C[i];
    cout << "\n";
    return 0;
}
```

高精度除法：

#include<bits/stdc++.h>
using namespace std;
vector div(vector &A, int b, int &r) {
vector C;
r = 0;
for(int i = A.size() - 1; i >= 0; i --) {
r = r * 10 + A[i];

C.push_back(r / b);
r %= b;
}
reverse(C.begin(), C.end());
while(C.size() > 1 && C.back() == 0) C.pop_back();
return C;
}

```
int main() {
    cin.tie(nullptr) -> sync_with_stdio(false);
    string a;
    int b;
    cin >> a >> b;
    vector<int> A;
    for(int i = a.size() - 1; i >= 0; i --) A.push_back(a[i] - '0');
    int r; //r为余数
    auto C = div(A, b, r);
    for(int i = C.size() - 1; i >= 0; --i) cout << C[i];
    cout << "\n" << r << "\n";
    return 0;
}
```

前缀和与差分
离散化
P1955

单调队列
单调栈

KMP
//https://www.luogu.com.cn/problem/P3375
#include
using namespace std;
const int N = 1e6 + 10;
string s, p;//文本串、模式串
int ne[N], n, m; //ne[i]指的是模式串前i长度的真前缀和真后缀的最大相同长度
int main() {

```
cin >> s >> p;
n = p.size(), m = s.size();
//ne数组
for(int i = 2, j = 0; i <= n; i ++) {
while(j && p[i - 1] != p[j]) j = ne[j];
if(p[i - 1] == p[j]) j ++;
ne[i] = j;
}
//kmp匹配
for(int i = 1, j = 0; i <= m; i ++) {
while(j && s[i - 1] != p[j]) j = ne[j];
if(s[i - 1] == p[j]) j ++;
if(j == n) {
cout << i - n + 1 << endl;
j = ne[j];
}
}
for(int i = 1; i <= n; i ++) {
cout << ne[i] << " \n"[i == n];
}
return 0;
}
```

马拉车

```
#include<bits/stdc++.h>
#define endl '\n'
#define fst first
#define snd second
```

```
using namespace std;

const int N = 22000005;
string s1, s2;
int p[N];

void getstr() {
    s2 += "@";
    int len = s1.length();
    for(int i = 0; i < len; i ++) {
        s2 += "#";
        s2 += s1[i];
    }
    s2 += "#";
    s2 += "*";
}

int manacher() {
    int len = s2.length();
    int id = 0, r = -1, maxn = -1;

    for(int i = 0; i < len; i ++) {
        if(i < r) p[i] = min(r - i, p[2 * id - i]); //(j+i)/2=id;
        else p[i] = 1;

        while(s2[i + p[i]] == s2[i - p[i]]) p[i] ++;

        if(i + p[i] > r) {
            r = i + p[i];
            id = i;
            maxn = max(maxn, p[i] - 1);
        }
    }
    //#b#b#长度为2，'#'扩展出去的半径为3
    //#a#b#a#长度为3，'b'扩展出去的长度为4
    // #扩展出去的回文是奇长度半径，反之是偶长度半径，都得-1还原奇偶性;
    return maxn;
}
```

```
int main() {
    cin.tie(nullptr) -> sync_with_stdio(false);

    getline(cin, s1);
    getstr();
    cout << manacher() << endl;
    return 0;
}
```

Trie树

对顶堆
st表
树状数组
//单点修改，区间查询
//log复杂度查询区间和
#include
using namespace std;
int n,m,tree[2000010];

```
int lowbit(int x) {
        return x & -x;
}

//log
void add(int x, int k) {
        while(x <= n) {
                tree[x] += k;
                x += lowbit(x);
        }
}
//log
int sum(int x) {
        int ans = 0;
        while(x != 0) {
                ans += tree[x];
                x -= lowbit(x);
        }
        return ans;
}

int main() {
        cin >> n >> m;
        for(int i = 1; i <= n; i ++) { //nlogn创建树状数组
                int a; cin >> a;
                add(i, a);
        }
        for(int i = 1; i <= m; i ++) {
                int a, b, c;
                cin >> a >> b >> c;
                if(a == 1) add(b, c);
                if(a == 2) cout << sum(c) - sum(b - 1) << endl;
        }
        return 0;
}
```

线段树

//区间操作

好用的剪枝：

搜索中加入最优性剪枝,当前枚举下一个状态时如果回到上一个状态肯定不是最优,
所以我们在枚举下一状态时加入对这种情况的判断

```
1.
    将状态数组对称排列会很方便进行这一操作
    int dx[4] = {0, 1, -1, 0};
    int dy[4] = {1, 0, 0, -1};

    int dx[8]={-2, -2, -1, 1, -1, 1, 2, 2};
    int dy[8]={-1, 1, 2, 2, -2, -2, -1, 1};

    A_star(?, ?, ?, pre), pre记录怎么到下一个状态
    for(int i = 0; i < 4; i ++) {
        //代码块
        if(pre + i == 3) continue;//对称
        //if(pre + i == 7) continue;
        //代码块
    }

2.
    void A_star(int step, int x, int y, int stx, int sty) {
        for(int i = 0; i < 4; i ++) {
            int nx = x + dx[i], ny = y + dy[i];
            //代码块
            if(nx == stx && ny == sty) continue;  //(stx, sty)是到(x, y)的上一个状态
            //代码块                                //(nx, ny)是(x, y)的下一个状态
            A_star(?, nx, ny, x, y);
        }
    }
```

A*估价剪枝
//test估价函数f(n) = g(n) + h(n);估价 = 实际 + 完美未来
//可能不可能实现，但还要取最优的步数
例题：
//P1379 八数码难题:https://www.luogu.com.cn/problem/P1379
#include<bits/stdc++.h>
#define int long long
#define endl '\n'

```
#define first fst
#define second snd
```

```
using namespace std;

int ans[4][4] = {
        {0, 0, 0, 0},
        {0, 1, 2, 3},
        {0, 8, 0, 4},
        {0, 7, 6, 5}
};
int a[4][4];
int dx[4] = {0, 1, -1, 0};
int dy[4] = {1, 0, 0, -1};
string s;
int k, judge, x, y;

int test(int step) {
        int cnt = 0;
        for(int i = 1;i <= 3; i ++) {
                for(int j = 1; j <= 3; j ++) {
                        if(a[i][j] != ans[i][j]) {
                                if( ++ cnt + step > k)
                                        return 0;
                        }
                }
        }
        return 1;
}

bool check() {
        for(int i = 1;i <= 3; i ++) {
                for(int j = 1; j <= 3; j ++) {
                        if(a[i][j] != ans[i][j])
                                return 0;
                }
        }
        return 1;
}

void Astar(int step, int x, int y, int stx, int sty){
        if(step == k) {
```

```
                if(check()) {
                        judge = 1;
                }
                return;
        }
        if(judge) return;
        for(int i = 0; i < 4; i ++) {
                int nx = x + dx[i], ny = y + dy[i];
                if(nx < 1 || nx > 3 || ny < 1 || ny > 3) continue;
                if(nx == stx && ny == sty) continue;
                swap(a[nx][ny], a[x][y]);
                if(test(step) && !judge) {
                        Astar(step + 1, nx, ny, x, y);
                }
                swap(a[x][y], a[nx][ny]);
        }
}

signed main() {
        ios::sync_with_stdio(false);
        cin.tie(nullptr);

        cin >> s;
        for(int i = 0; i < 9; i ++) {
                a[i / 3 + 1][i % 3 + 1] = s[i] - '0';
                if(s[i] == '0') {
                        x = i / 3 + 1;
                        y = i % 3 + 1;
                }
        }

        if(check()) {
                cout << 0 << endl;
                return 0;
        }

        while( ++ k) {
                Astar(0, x, y, -1, -1);
                if(judge) {
```

```
                        cout << k << endl;
                        break;
                }
        }

        return 0;
}
```

记忆化搜索

```
int g[N];//定义记忆化数组
int ans = 最坏情况, now;

void dfs(传入数值) {
        if(g[规模] != 无效数值) return; //或记录解，视情况而定，有时有效即可返回
        if(到达目的地) ans = 从当前解与已有解中选取最优 //输出解，视情况而定
        for() {
                if(ok) {
                        //操作;
                        dfs(缩小规模);
                        //撤回操作
                }
        }
}

int main() {
        memset(g, 无效数值, sizeof g); //初始化记忆化数组
}
```

例题：

//P1434 [SHOI2002] 滑雪https://www.luogu.com.cn/problem/P1434

```
#include<bits/stdc++.h>
using namespace std;
const int N = 250;
int dx[4] = {0, 1, -1, 0};
int dy[4] = {1, 0, 0, -1};
int n, m, a[N][N], s[N][N], ans;
```

bool vis[N][N];

```
bool check(int x, int y) {
        return x >= 1 && x <= n && y >= 1 && y <= m;
}

int dfs(int x, int y) {
        if(s[x][y]) return s[x][y]; //!!!
        s[x][y] = 1;//自身
        for(int i = 0; i < 4; i ++) {
                int nx = x + dx[i], ny = y + dy[i];
                if(!check(nx, ny) || a[x][y] <= a[nx][ny]) continue;
                dfs(nx, ny);
                s[x][y] = max(s[x][y], s[nx][ny] + 1);//走到下一步加上本身
        }
        return s[x][y];  //最底层返回能走的最大长度
}

int main() {
        cin >> n >> m;
        for(int i = 1; i <= n; i ++) {
                for(int j = 1; j <= m; j ++) {
                        cin >> a[i][j];
                }
        }
        for(int i = 1; i <= n; i ++) {
                for(int j = 1; j <= m; j ++) {
                        ans = max(ans, dfs(i, j)); //从不同的n*m个点出发
                }
        }
        cout << ans << endl;
        return 0;
}
```

多源bfs(多个起点出发)

例题：
//牛客小A与小B：https://ac.nowcoder.com/acm/problem/23486
#include

```
#include
#include
#include
```

```
using namespace std;
typedef pair<int, int> PII;
const int N = 1010;
int dx[8]={0,0,1,-1,1,1,-1,-1};
int dy[8]={1,-1,0,0,1,-1,1,-1};
queue<PII> q[2];  //0代表小\A , 1代表小\B
bool vis[2][N][N];
char g[N][N];
int n, m;

bool bfs(int w) {
        int sz = q[w].size(); //这一秒q这个队列有几种待出发状态
        while(sz --) {
                auto t = q[w].front();
                q[w].pop();
                for(int i = 0; i < 8 - (4 * w); i ++) {
                        int x = t.first + dx[i];
                        int y = t.second + dy[i];
                        if(x < 1 || y < 1 || x > n || y > m || g[x][y] == '#') cont
                        if(vis[1 - w][x][y]) return true;//1^w == 1 - w//最早相遇
                        if(!vis[w][x][y]) {
                                q[w].push({x, y});
                                vis[w][x][y] = true;
                        }
                }
        }
        return false;
}
int final() {
        int ans = 0;
        while(!q[0].empty() || !q[1].empty()) { //ans代表时间
                ans ++;
                if(bfs(0)) return ans;
                if(bfs(1)) return ans;
                if(bfs(1)) return ans; //小\a跑两次
        }
        return -1;
}
int main() {
```

```
        cin >> n >> m;
        memset(vis, false, sizeof vis);
        for(int i = 1; i <= n;i ++) {
                for(int j = 1; j <= m; j ++) {
                        cin >> g[i][j];
                        if(g[i][j] == 'C') {
                                q[0].push({i, j});
                                vis[0][i][j] = true;
                        }
                        if(g[i][j] == 'D') {
                                q[1].push({i, j});
                                vis[1][i][j] = true;
                        }
                }
        }
        int ans = final();
        if(ans == -1) cout << "NO" << endl;
        else {
                cout << "YES" << endl;
                cout << ans << endl;
        }
        return 0;
}
```

双端队列BFS/0-1BFS

//例题：https://codeforces.com/problemset/problem/173/B

折半搜索/meet in the middle

//双向搜索用来解决N等于三四十的搜索问题。将$2^{40分解为2}$20~1e6的复杂度

//(状压开关灯)https://www.luogu.com.cn/problem/P2962

```
#include<bits/stdc++.h>
#define int long long
#define endl '\n'
#define first fst
#define second snd
using namespace std;
map<int, int> mp;
int a[50] = {1};
```

```
int n, m, ans = 0x7fffff;
signed main() {
ios::sync_with_stdio(false);
cin.tie(nullptr);
cin >> n >> m;
for(int i = 0; i < n; i ++) a[i] = (1ll << i);
for(int i = 1; i <= m; i ++) {
int x, y;
cin >> x >> y;
x --, y --;
a[x] |= (1ll << y);
a[y] |= (1ll << x);
}
```

```
        for(int i = 0; i < 1 << (n / 2); i ++) {
                int t = 0;
                int cnt = 0;
                for(int j = 0; j < n / 2; j ++) {
                        if(i >> j & 1) {//不会超过一半量的搜索情况
                                t ^= a[j];
                                cnt ++;
                        }
                }
                if(!mp.count(t)) {
                        mp[t] = cnt;
                }
                else mp[t] = min(mp[t], cnt);
        }

        for(int i = 0; i < 1 << (n - n / 2); i ++) {
                int t = 0;
                int cnt = 0;
                for(int j = 0; j < n - (n / 2); j ++) {
                        if(i >> j & 1) {
                                t ^= a[n / 2 + j];
                                cnt ++;
                        }
                }
                if(mp.count(((1ll << n) - 1) ^ t)) {
                        ans = min(ans, cnt + mp[((1ll << n) - 1) ^ t]);
                }
        }

        cout << ans << endl;
        return 0;
}
```

//https://www.luogu.com.cn/problem/P4799

/*我们把40个数，分为前一半和后一半，算出他们能够组合出的价格
并分别记录进数组里（当然已经超过M就算了），把其中一个数组排序。
（这里是ka）接下来后一半数里肯定是要选数的，我们从前往后遍历kb，
就相当于遍历选后一半的所有情况，比如当kb遍历到500的时候，
由于M == 1000，所以ka里小于等于500的数都可以贡献答案，

```
因为ka排好序了，所以直接二分查找就可以确定小于等于M - kb[i]的数的个数
*/
#include<bits/stdc++.h>
#define int long long
#define endl '\n'
#define fst first
#define snd second
using namespace std;
const int N = 55;
int t[N], suma[2000000], sumb[2000000], cnta, cntb;
int n, m, ans;
```

```
void dfs(int l, int r, int sum, int a[], int &cnt) {
    if(sum > m) return;
    if(l > r) {
        a[++ cnt] = sum;
        return;
    }
    dfs(l + 1, r, sum + t[l], a, cnt);
    dfs(l + 1, r, sum, a, cnt);
}

signed main() {
    cin >> n >> m;
    for(int i = 1; i <= n; i ++) {
        cin >> t[i];
    }
    int mid = n / 2;
    dfs(1, mid, 0, suma, cnta);
    dfs(mid + 1, n, 0, sumb, cntb);

    sort(suma + 1, suma + 1 + cnta);

    for(int i = 1; i <= cntb; i ++) {
        ans += upper_bound(suma + 1, suma + 1 + cnta, m - sumb[i]) - suma - 1;
    }
    cout << ans << endl;
    return 0;
}
```

//https://www.luogu.com.cn/problem/P3067

IDA*(迭代加深DFS + A*) //广搜下的深搜，加估价函数

//https://www.luogu.com.cn/problem/P2324

#include<bits/stdc++.h>

#define int long long

#define endl '\n'

#define first fst

#define second snd

```
using namespace std;

char ans[6][6] = {
        {'*', '*', '*', '*', '*', '*'},
        {'*', '1', '1', '1', '1', '1'},
        {'*', '0', '1', '1', '1', '1'},
        {'*', '0', '0', '*', '1', '1'},
        {'*', '0', '0', '0', '0', '1'},
        {'*', '0', '0', '0', '0', '0'},
};
char a[6][6];
int dx[8] = {-2, -1, 1, 2, 2, 1, -1, -2};
int dy[8] = {-1, -2, -2, -1, 1, 2, 2, 1};
int x, y, k, judge;

bool check() {
        for(int i = 1; i <= 5; i ++) {
                for(int j = 1; j <= 5; j ++) {
                        if(a[i][j] != ans[i][j]) {
                                return 0;
                        }
                }
        }
        return 1;
}

bool test(int step) { //估价函数
        int cnt = 0;
        for(int i = 1; i <= 5; i ++) {
                for(int j = 1; j <= 5; j ++) {
                        if(a[i][j] != ans[i][j]) {
                                if( ++ cnt + step > k) {
                                        return 0;
                                }
                        }
                }
        }
        return 1;
}
```

```
void Astar(int step, int x, int y, int stx, int sty) {
        if(step == k) {
                if(check()) {
                        judge = 1;
                }
                return;
        }
        if(judge) return;
        for(int i = 0; i < 8; i ++) {
                int nx = x + dx[i], ny = y + dy[i];
                if(nx < 1 || nx > 5 || ny < 1 || ny > 5) continue;
                if(nx == stx && ny == sty) continue; //反操作,剪枝
                swap(a[x][y], a[nx][ny]);
                if(test(step) && !judge) { //估值可行
                        Astar(step + 1, nx, ny, x, y);
                }
                swap(a[x][y], a[nx][ny]);
        }
}
signed main() {
        ios::sync_with_stdio(false);
        cin.tie(nullptr);
        int T; cin >> T;
        while(T --) {
                judge = 0, k = 0;
                for(int i = 1; i <= 5; i ++) {
                        for(int j = 1; j <= 5; j ++) {
                                cin >> a[i][j];
                                if(a[i][j] == '*') {
                                        x = i, y = j;
                                }
                        }
                }
                if(check()) {
                        cout << 0 << endl;
                        continue;
                }
                while( ++ k) {
```

```
                        if(k > 15) {
                                cout << -1 << endl;
                                break;
                        }
                        Astar(0, x, y, -1, -1);
                        if(judge) {
                                cout << k << endl;
                                break;
                        }
                }
        }
        return 0;
}
```

//铁盘整理https://www.luogu.com.cn/problem/P2534

#include<bits/stdc++.h>

#define int long long

#define endl '\n'

#define first fst

#define second snd

```
using namespace std;

const int N = 20;
int a[N], ans[N];
int n, k, judge;

bool check() {
        for(int i = 1; i <= n; i ++) {
                if(a[i] != ans[i]) {
                        return 0;
                }
        }
        return 1;
}

int test() {//最完美估价 //最好情况下
    int cnt = 0;
    for (int i = 1; i <= n; ++i)
        cnt += abs(a[i] - a[i + 1]) != 1;
    return cnt;
}

void dfs(int step, int p) { //step为深度
        if(judge) return;
        if(step + test() > k) return;
        if(!test()) {
                judge = 1;
                return;
        }
        for(int i = 1; i <= n; i ++) {
                if(i == p) continue;
                for(int j = 1; j <= i / 2; j ++) {
                        swap(a[j], a[i - j + 1]);
                }

                dfs(step + 1, i);

                for(int j = 1; j <= i / 2; j ++) {
                        swap(a[j], a[i - j + 1]);
```

```
                }
        }
}

signed main() {
    cin >> n;
    for(int i = 1; i <= n; i ++) {
            cin >> a[i];
            ans[i] = a[i];
    }

    sort(ans + 1, ans + n + 1);

    for(int i = 1; i <= n; i ++) {
            a[i] = lower_bound(ans + 1, ans + 1 + n, a[i]) - ans;
    }
    a[n + 1] = n + 1;
    if(check()) {
            cout << 0 << endl;
            return 0;
    }
    while( ++ k) { //迭代加深的深度
            dfs(0, -1);
            if(judge) {
                    cout << k << endl;
                    return 0;
            }
    }
    return 0;
}
```

最短路

拓扑排序

线性筛

```
const int N = 10000005;
int p[N], st[N], cnt; //st[i]表示i的最小素因子,st[i] == i时为质数
void ola_primes(int x) {
for(int i = 2; i <= x; i ++) {
```

```
if(!st[i]) {
p[cnt ++] = i;
st[i] = i;
}
for(int j = 0; p[j] <= x /i; j ++) {
st[p[j] * i] = p[j];
if(i % p[j] == 0) {
break;
}
}
}
}
```

逆元

背包
区间DP

换根DP
板子

```
dfs1(int x, int fa) { //跑第一遍树获取初始信息
for(int i = head[x]; i; i = nxt[i]) {
int v = to[i];
if(v == fa) continue; //构成自环退出
dfs1(v, x);
//补充代码块
}
}
dfs2(int x, int fa) { //换根操作
for(int i = head[x]; i; i = nxt[i]) {
int v = to[i];
if(v == fa) continue;
//补充代码块
dfs2(v, x); //换根的影响
//补充代码块
}
```

}

例题：

//CCPC河南邀请赛K题树上问题：https://codeforces.com/gym/105158

```
#include<bits/stdc++.h>
#define int long long
#define endl '\n'
#define first fst
#define second snd
```

```
using namespace std;

const int N = 1e5 + 10;
int a[N], to[2 * N], nxt[2 * N], head[N], idx;
int now, ans[N], n;

void add(int u, int v) {
        ++ idx; //边的序号>=1
        to[idx] = v; //这条边指向v
        nxt[idx] = head[u]; //这条边循环遍历的下一条边序号是u连出去的上一条
        head[u] = idx; //更新u连出去的最新一条
}

void dfs1(int x, int fa) {
        if(2 * a[x] < a[fa]) now ++;
        for(int i = head[x]; i; i = nxt[i]) {
                int v = to[i];
                if(v == fa) continue;
                dfs1(v, x);
        }
}

void dfs2(int x, int fa) {
        ans[x] = now;
        for(int i = head[x]; i; i = nxt[i]) {
                int v = to[i];
                if(v == fa) continue;
                if(2 * a[v] < a[x]) now --;
                if(2 * a[x] < a[v]) now ++;
                dfs2(v, x);
                if(2 * a[v] < a[x]) now ++;
                if(2 * a[x] < a[v]) now --;
        }

}

void solve() {
        cin >> n;
        for(int i = 1; i <= n; i ++) cin >> a[i];
```

```
        for(int i = 1; i <= 2 * n; i ++) to[i] = nxt[i] = head[i] = 0;
        now = 0, idx = 0;
        for(int i = 1; i < n; i ++) {
                int u, v;
                cin >> u >> v;
                add(u, v);
                add(v, u);
        }

        dfs1(1, 0);
        dfs2(1, 0);

        int res = 0;
        for(int i = 1; i <= n; i ++) {
                if(!ans[i]) res ++;
        }
        cout << res << endl;
}

signed main() {
        ios::sync_with_stdio(false);
        cin.tie(nullptr);

        int T; cin >> T;
        while(T --) {
                solve();
        }
        return 0;
}
```

//"华为杯"华南理工大学程序设计竞赛H题信号塔：https://ac.nowcoder.com/acm/contest/79505/H

```
#include<bits/stdc++.h>
#define int long long
#define endl '\n'
#define fst first
#define snd second
using namespace std;
```

```
const int N = 2e5 + 10;
int head[N], to[2 * N], nxt[2 * N], idx;
int ans[N], a[N], dp1[N], dp2[N], n;

void add(int u, int v) {
    ++ idx;
    to[idx] = v;
    nxt[idx] = head[u];
    head[u] = idx;
}

void dfs1(int x, int fa) {
    dp1[x] = 1, dp2[x] = 0;
    for(int i = head[x]; i; i = nxt[i]) {
        int v = to[i];
        if(v == fa) continue;
        dfs1(v, x);
        dp2[x] += dp1[v];
        dp1[x] += min(dp1[v], dp2[v]);
    }
}

void dfs2(int x, int fa) {
    for(int i = head[x]; i; i = nxt[i]) {
        int v = to[i];
        if(v == fa) continue;
        int res1 = dp1[x] - min(dp1[v], dp2[v]);
        int res2 = dp2[x] - dp1[v];
        ans[(i + 1) / 2] = min(res1, res2) + min(dp1[v], dp2[v]); //1、2号点为1号边以
        dp1[v] += min(res1, res2);
        dp2[v] += res1;
        dfs2(v, x);
    }
}

void solve() {
    cin >> n;
    for(int i = 1; i <= n; i ++) {
        head[i] = 0;
```

```
    }
    idx = 0;
    for(int i = 1; i < n; i ++) {
        int u, v;
        cin >> u >> v;
        add(u, v);
        add(v, u);
    }
    dfs1(1, -1);
    dfs2(1, -1);
    for(int i = 1; i < n; i ++) {
        cout << n - ans[i] << " \n"[i == n - 1];
    }
}

signed main() {
    cin.tie(nullptr) -> sync_with_stdio(false);
    int T; cin >> T;
    while(T --) {
        solve();
    }
    return 0;
}
```

//2022年GDCPC的L题起航者：https://pintia.cn/problem-sets/1534086341544497152/exam/problems/type/7?problemSetProblemId=1534088931057451019&page=0

```
#include<bits/stdc++.h>
#define int long long
#define endl '\n'
#define first fst
#define second snd
```

```
using namespace std;
const int N = 1e6 + 10;
int idx, head[N], to[2 * N], nxt[2 * N];
int maxto[N], maxto2[N], a[N], dp[N];
int n, ans[N], ansid;

void add(int u, int v) {
        ++ idx;
        to[idx] = v;
        nxt[idx] = head[u];
        head[u] = idx;
}

void dfs1(int x, int fa) {
        dp[x] = a[x];
        int maxn = -1, res = 0;
        for(int i = head[x]; i; i = nxt[i]) {
                int v = to[i];
                if(v == fa) continue;
                if(a[v] > maxn) {
                        maxn = a[v];
                        res = v;
                }

                dfs1(v, x);
        }
        dp[x] += dp[res];
}

void dfs2(int x, int fa) {
        ans[x] = dp[x];
        for(int i = head[x]; i; i = nxt[i]) {
                int v = to[i];
                if(v == fa) continue;
                int q = dp[x], p = dp[v];
                if(maxto[x] == v) dp[x] = a[x] + dp[maxto2[x]]; //x走向次大
                if(maxto[v] == x) dp[v] = a[v] + dp[x];
                //这样写的顺序顺便将x与v各自的最大边是彼此的情况笼罩了
                dfs2(v, x);
```

```
                dp[x] = q; //还原
                dp[v] = p;
        }
}

signed main() {
        ios::sync_with_stdio(false);
        cin.tie(nullptr);

        cin >> n;

        for(int i = 1; i < n; i ++) {
                int u; cin >> u;
                add(i + 1, u);
                add(u, i + 1);
        }
        for(int i = 1; i <= n; i ++) cin >> a[i];
        for(int i = 1; i <= n; i ++) { //预处理连出去的最大边与次大边
                int maxn = -1, maxn2 = -1;
                for(int j = head[i]; j; j = nxt[j]) {
                        if(a[to[j]] > maxn) {
                                maxn = a[to[j]];
                                maxto[i] = to[j];
                        }
                }
                for(int j = head[i]; j; j = nxt[j]) {
                        if(a[to[j]] == maxn) continue;
                        if(a[to[j]] > maxn2) {
                                maxn2 = a[to[j]];
                                maxto2[i] = to[j];
                        }
                }
        }
        dfs1(1, 0);
        dfs2(1, 0);
        int now = -1;
        for(int i = 1; i <= n; i ++) {
                if(ans[i] > now) {
                        now = ans[i];
```

```
                        ansid = i;
                }
        }
        cout << ansid << endl << now << endl;
        return 0;
}
```

# 最长上升子序列

最长上升子序列的长度=最少不上升子序列的个数

导弹拦截

```
int a[N], cnt;
int main() {
    while(cin >> a[++ cnt]);

    vector<int> q1, q2;
    q1.push_back(a[1]), q2.push_back(a[1]);

    for(int i = 2; i <= cnt - 1 ; i ++) {
        if(a[i] > q1.back()) q1.push_back(a[i]);//满足上升条件直接加队尾
        else *lower_bound(q1.begin(), q1.end(), a[i]) = a[i];
        //否则找到第一个大于等于a[i]的数换成a[i]，更小的值的队尾空间更大，size没变
        //如果找到的位置不是在队尾，那也不会对size造成影响

        if(a[i] <= q2.back()) q2.push_back(a[i]);
        else *upper_bound(q2.begin(), q2.end(), a[i], greater<int>()) = a[i];
    }
    cout << q2.size() << endl << q1.size() << endl;
    return 0;
}
```