# ACM 常用算法模板

Vijay

https://www.vijay.vip
Email:vijaysue@yeah.net

Last build at May 14, 2023

# Contents

# 1 搜索 Search

## 1.1 A*算法

### 1.1.1 八数码

```
struct matrix {
    vector<vector<int>> g;
    bool operator<(matrix a) const {
        return g < a.g;
    }
} st, v;
int h(matrix a) {       //h 函数定义:不在标准位置的数字个数
    int res{};
    for (int i = 0; i < 3; ++i) for (int j = 0; j < 3; ++j) if (a.g[i][j] !=
st.g[i][j]) ++res;
    return res;
}
struct node {
    matrix m;
    int cnt;        //移动次数
    bool operator<(node a) const { return cnt + h(m) > a.cnt + h(a.m); }
};

void solution() {
    st.g = vector<vector<int>>{{1, 2, 3}, {8, 0, 4}, {7, 6, 5}};        //
定义标准矩阵
    v.g = vector<vector<int>>(3, vector<int>(3));
    for (auto& x : v.g) for (int i = 0; i < 3; ++i) {       //输入
        char c;
        cin >> c;
        x[i] = c - '0';
    }
    priority_queue<node> q;
    set<matrix> s;      //防止重复
    int fx = 0, fy = 0;     //空格的位置
    q.push({v, 0});
    while (!q.empty()) {
        auto t = q.top();
        q.pop();
        if (!h(t.m)) {      //判断与标准矩阵是否一阵
            cout << t.cnt << '\n';
            return;
        }
         for (int i = 0; i < 3; ++i) for (int j = 0; j < 3; ++j) if
(!t.m.g[i][j]) fx = i, fy = j;      //查找空格位置
        for (auto& dir : DIRS) {        //向四种方向移动
            int x = fx + dir.first, y = fy + dir.second;
            if (x >= 0 && y >= 0 && x < 3 && y < 3) {
                swap(t.m.g[fx][fy], t.m.g[x][y]);
                if (!s.count(t.m)) {
                    s.emplace(t.m);
                    q.push({t.m, t.cnt + 1});
                }
                swap(t.m.g[fx][fy], t.m.g[x][y]);       //撤销操作
            }
        }
    }
}
```

### 1.1.2 K 短路

```
vector<double> dist;        //从终点到各点的最短距离

void solution() {
    int n, m, u, v;     //点数，边数
    double e, s;        //边权，总权值
    cin >> n >> m >> s;
    dist = vector<double>(n + 1, INF);
    dist[n] = 0;
    vector<bool> vis(n + 1, false);     //判断节点是否已被访问
    vector<int> cnt(n + 1, 0);          //每个点被访问的次数
    vector<vector<pair<int,double>>> adj(n + 1), adj2(n + 1);   //邻接表;
    for (int i = 0; i < m; ++i) {
        cin >> u >> v >> e;
        adj[u].emplace_back(v, e);
        adj2[v].emplace_back(u, e);
    }
    struct node {       //使用A*所需的结构体
        int idx{};      //节点编号
        double d{};     //表示从起点到该节点的实际距离
        bool operator<(node a) const {
            return d + dist[idx] > a.d + dist[a.idx];
        }
    };
    struct node2 {      //计算终点到所有节点的最短路
        int idx{};
        double d{};
        bool operator<(node2 a) const { return d > a.d; }
    };
    priority_queue<node> q;
    priority_queue<node2> q2;
    int ans{};
    q2.push({n, 0});
    while (!q2.empty()) {
        auto t = q2.top();
        q2.pop();
        if (vis[t.idx]) continue;
        vis[t.idx] = true;
        dist[t.idx] = t.d;
        for (auto& e : adj2[t.idx]) q2.push({e.first, e.second + t.d});
    }
    int k = (int)s / dist[1];
    q.push({1, 0});
    while (!q.empty()) {        //使用 A*算法
        auto t = q.top();
        q.pop();
        ++cnt[t.idx];
        if (t.idx == n) {
            s -= t.d;
            if (s < 0) {
                cout << ans << '\n';
                return;
            }
            ++ans;
        }
        for (auto& e : adj[t.idx])
            if (cnt[e.first] <= k && t.d + e.second <= s)
               q.push({e.first, e.second + t.d});
    }
    cout << ans << '\n';
}
```

## 1.2 Dancing Links X 算法

### 1.2.1 精确覆盖问题

```
const int MX = 1e5 + 10;
int n, m, ans, stk[MX];

struct DLX {
    int n{}, m{}, idx{};        //行数，列数，当前列的索引
    int L[MX], R[MX], U[MX], D[MX], col[MX], row[MX], head[MX], sz[MX];
    //指针，行列，问题解，表头，每列数量
    void init (const int& r, const int& c) {    //初始化循环双向链表
        n = r, m = c;
        for (int i = 0; i <= c; ++i) {
            L[i] = i - 1;
            R[i] = i + 1;
            U[i] = D[i] = i;
        }
        L[0] = c, R[c] = 0;
        idx = c;
        memset(head, 0 ,sizeof(head));
        memset(sz, 0, sizeof(sz));
    }
    void insert(const int& r, const int& c) {//插入节点
        col[++idx] = c, row[idx] = r, ++sz[c];
        D[idx] = D[c], U[D[c]] = idx, U[idx] = c, D[c] = idx;
        if (!head[r]) head[r] = L[idx] = R[idx] = idx;
        else {
            R[idx] = R[head[r]], L[R[head[r]]] = idx;
            L[idx] = head[r], R[head[r]] = idx;
        }
    }
    void remove(const int& c) {     //移除 c 列以及所有与其相交的行
        L[R[c]] = L[c], R[L[c]] = R[c];
        for (int i = D[c]; i != c; i = D[i])
            for (int j = R[i]; j != i; j = R[j])
                U[D[j]] = U[j], D[U[j]] = D[j], --sz[col[j]];
    }
    void resume(const int& c) { //恢复 c 列以及所有与其相交的行
        L[R[c]] = R[L[c]] = c;
        for (int i = U[c]; i != c; i = U[i])
            for (int j = L[i]; j != i; j = L[j])
                U[D[j]] = D[U[j]] = j, ++sz[col[j]];
    }
    bool dance(int dep) {   //递归求解
        int c = R[0];
        if (!c) {
            ans = dep;
            return true;
        }
        for (int i = c; i != 0; i = R[i]) {
            if (sz[i] < sz[c]) c = i;
        }
        remove(c);
        for (int i = D[c]; i != c; i = D[i]) {
            stk[dep] = row[i];
            for (int j = R[i]; j != i; j = R[j]) remove(col[j]);
            if (dance(dep + 1)) return true;
            for (int j = L[i]; j != i; j = L[j]) resume(col[j]);
        }
        resume(c);
        return false;
    }
} solver;

void solution() {
    cin >> n >> m;
    solver.init(n, m);
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            int x;
            cin >> x;
            if (x) solver.insert(i, j);
        }
    }
    solver.dance(0);    //从第一行开始递归求解
    if (ans) for (int i = 0; i < ans; ++i) cout << stk[i] << ' ';
    else cout << "No Solution!" << '\n';
}
```

### 1.2.2 重复覆盖问题

```
const int MX = 1e5 + 10;
int n, m, ans;

struct DLX {
    bool vis[MX];
    int n{}, m{}, idx{};        //行数，列数，当前列的索引
    int L[MX], R[MX], U[MX], D[MX], col[MX], row[MX], head[MX], sz[MX];
```

```
//指针，行列，问题解，表头，每列数量
void init (const int& r, const int& c) {    //初始化循环双向链表
    n = r, m = c;
    for (int i = 0; i <= c; ++i) {
        L[i] = i - 1;
        R[i] = i + 1;
        U[i] = D[i] = i;
    }
    L[0] = c, R[c] = 0;
    idx = c;
    memset(head, 0 ,sizeof(head));
    memset(sz, 0, sizeof(sz));
}
void insert(const int& r, const int& c) {//插入节点
    col[++idx] = c, row[idx] = r, ++sz[c];
    D[idx] = D[c], U[D[c]] = idx, U[idx] = c, D[c] = idx;
    if (!head[r]) head[r] = L[idx] = R[idx] = idx;
    else {
        R[idx] = R[head[r]], L[R[head[r]]] = idx;
        L[idx] = head[r], R[head[r]] = idx;
    }
}
void remove(const int& c) {     // 删除和恢复和精确覆盖有所不同
    for(int i = D[c]; i != c; i = D[i])
        L[R[i]] = L[i], R[L[i]] = R[i];

}
void resume(const int& c) {
    for(int i = U[c]; i != c; i = U[i])
        L[R[i]] = R[L[i]] = i;

}
int h() {   // 预估函数
    int ret = 0;
    memset(vis, 0, sizeof vis);
    for(int i = R[0]; i; i = R[i]) {
        if(vis[i]) continue;
        ++ret; vis[i] = true;
        for(int j = D[i]; j != i; j = D[j])
            for(int k = R[j]; k != j; k = R[k])
                vis[col[k]] = true;
    }
    return ret;
}
void dance(int dep) {   //递归求解
    if (dep + h() >= ans) return;
    int c = R[0];
    if (!c) {
        if (dep < ans) ans = dep;
        return;
    }
    for (int i = c; i != 0; i = R[i]) {
        if (sz[i] < sz[c]) c = i;
    }
    remove(c);
```

```
        for (int i = D[c]; i != c; i = D[i]) {
            remove(i);
            for (int j = R[i]; j != i; j = R[j]) remove(j);
            dance(dep + 1);
            for (int j = L[i]; j != i; j = L[j]) resume(j);
            resume(i);
        }
        return;
    }
} solver;

void solution() {
    while (cin >> n >> m) {
        solver.init(n, n);
        for (int i = 1; i <= n; ++i) solver.insert(i, i);
        for (int i = 1; i <= m; ++i) {
            int x, y;
            cin >> x >> y;
            solver.insert(x, y);
            solver.insert(y, x);
        }
        solver.dance(1);
        cout << ans << '\n';
    }
}
```

# 2 字符串 String

## 2.1 KMP 算法

```
vector<int> prefix_function(string s) {     //前缀函数
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j - 1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
    }
    return pi;
}
vector<int> find_occurrences(string text, string pattern) {     //KMP 算法
    string cur = pattern + '#' + text;
    int sz1 = text.size(), sz2 = pattern.size();
    vector<int> v;
    vector<int> lps = prefix_function(cur);
    for (int i = sz2 + 1; i <= sz1 + sz2; i++) {
        if (lps[i] == sz2)
        v.push_back(i - 2 * sz2);
    }
    return v;
}
```

```
void compute_automaton(string s, vector<vector<int>>& aut) { //前缀函数自动机
    s += '#';
    int n = s.size();
    vector<int> pi = prefix_function(s);
    aut.assign(n, vector<int>(26));
    for (int i = 0; i < n; i++) {
        for (int c = 0; c < 26; c++) {
            if (i > 0 && 'a' + c != s[i])
                aut[i][c] = aut[pi[i - 1]][c];
            else
                aut[i][c] = i + ('a' + c == s[i]);
        }
    }
}
```

## 2.2 Z 算法

```
vector<int> z_function(string s) {      //最长公共前缀
    int n = (int)s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r && z[i - l] < r - i + 1) {
        z[i] = z[i - l];
        } else {
        z[i] = max(0, r - i + 1);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
        }
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
    return z;
}
```

## 2.3 字典树

### 2.3.1 查询是否出现

```
struct Trie {
    int next[MX][26], cnt;
    bool exist[MX];  // 该结点结尾的字符串是否存在

    void insert(string s) {  // 插入字符串
        int p = 0;
        for (int i = 0; i < s.size(); ++i) {
            int c = s[i] - 'a';
            if (!next[p][c]) next[p][c] = ++cnt;  // 如果没有，就添加结点
            p = next[p][c];
        }
        exist[p] = true;
    }

    bool find(string s) {  // 查找字符串
```

```
        int p = 0;
        for (int i = 0; i < s.size(); ++i) {
            int c = s[i] - 'a';
            if (!next[p][c]) return false;
            p = next[p][c];
        }
        return exist[p];
    }
};
```

### 2.3.2 查询前缀出现次数

```
struct Trie {
    int next[MX][26],cnt,pre[MX];
    bool p;
    int n, m;
    void insert(string s) {
        int p = 0;
        for(int i=0;i<s.size();i++)
        {
            int c = s[i]-'a';
            if(!next[p][c]) next[p][c] = ++cnt;
            pre[next[p][c]]++;//前缀保存
            p = next[p][c];
        }
    }
    int search(string s) {
        int p = 0;
        for(int i = 0; i < s.size();i++)
        {
            int c = s[i] - 'a';
            if(!next[p][c]) return 0;
            p=next[p][c];
        }//root 经过此循环后变成前缀最后一个字母所在位置
        return pre[p];
    }
};
```

## 2.4 AC 自动机

```
#include <deque>
#include <iostream>

void promote()
{
    std::ios::sync_with_stdio(0);
    std::cin.tie(0);
    std::cout.tie(0);
    return;
}

typedef char chr;
typedef std::deque<int> dic;
```

```
const int maxN = 2e5;
const int maxS = 2e5;
const int maxT = 2e6;

int n;
chr s[maxS + 10];
chr t[maxT + 10];
int cnt[maxN + 10];

struct AhoCorasickAutomaton
{
    struct Node
    {
        int son[30];
        int val;
        int fail;
        int head;
        dic index;
    } node[maxS + 10];

    struct Edge
    {
        int head;
        int next;
    } edge[maxS + 10];

    int root;
    int ncnt;
    int ecnt;

    void Insert(chr *str, int i)
    {
        int u = root;
        for (int i = 1; str[i]; i++)
        {
            if (node[u].son[str[i] - 'a' + 1] == 0)
                node[u].son[str[i] - 'a' + 1] = ++ncnt;
            u = node[u].son[str[i] - 'a' + 1];
        }
        node[u].index.push_back(i);
        return;
    }

    void Build()
    {
        dic q;
        for (int i = 1; i <= 26; i++)
            if (node[root].son[i])
                q.push_back(node[root].son[i]);
        while (!q.empty())
        {
            int u = q.front();
            q.pop_front();
            for (int i = 1; i <= 26; i++)
            {
                if (node[u].son[i])
                {
                    node[node[u].son[i]].fail = node[node[u].fail].son[i];
                    q.push_back(node[u].son[i]);
                }
                else
                {
                    node[u].son[i] = node[node[u].fail].son[i];
                }
            }
        }
        return;
    }

    void Query(chr *str)
    {
        int u = root;
        for (int i = 1; str[i]; i++)
        {
            u = node[u].son[str[i] - 'a' + 1];
            node[u].val++;
        }
        return;
    }

    void addEdge(int tail, int head)
    {
        ecnt++;
        edge[ecnt].head = head;
        edge[ecnt].next = node[tail].head;
        node[tail].head = ecnt;
        return;
    }

    void DFS(int u)
    {
        for (int e = node[u].head; e; e = edge[e].next)
        {
            int v = edge[e].head;
            DFS(v);
            node[u].val += node[v].val;
        }
        for (auto i : node[u].index)
            cnt[i] += node[u].val;
        return;
    }

    void FailTree()
    {
        for (int u = 1; u <= ncnt; u++)
            addEdge(node[u].fail, u);
        DFS(root);
        return;
    }
} ACM;

int main()
```

```
{
    std::cin >> n;
    for (int i = 1; i <= n; i++)
    {
        std::cin >> (s + 1);
        ACM.Insert(s, i);
    }
    ACM.Build();
    std::cin >> (t + 1);
    ACM.Query(t);
    ACM.FailTree();
    for (int i = 1; i <= n; i++)
        std::cout << cnt[i] << '\n';
    return 0;
}
```

## 2.5 Manacher 算法

```
void Manacher(char s[], int len)
{ // 原字符串和串长
    int l = 0;
    String[l++] = '$'; // 0 下标存储为其他字符,防止越界
    String[l++] = '#';
    for (int i = 0; i < len; i++)
    {
        String[l++] = s[i];
        String[l++] = '#';
    }
    String[l] = 0; // 空字符
    int MaxR = 0;
    int flag = 0;
    for (int i = 0; i < l; i++)
    {
         cnt[i] = MaxR > i ? min(cnt[2 * flag - i], MaxR - i) : 1; //
2*flag-i 是 i 点关于 flag 的对称点
        while (String[i + cnt[i]] == String[i - cnt[i]])
            cnt[i]++;
        if (i + cnt[i] > MaxR)
        {
            MaxR = i + cnt[i];
            flag = i;
        }
    }
}
```

## 2.6 字符串哈希

```
int count_unique_substrings(string const &s)
{
    int n = s.size();
    const int b = 31;
    const int m = 1e9 + 9;
    vector<long long> b_pow(n);
    b_pow[0] = 1;
    for (int i = 1; i < n; i++)
        b_pow[i] = (b_pow[i - 1] * b) % m;

    vector<long long> h(n + 1, 0);
    for (int i = 0; i < n; i++)
        h[i + 1] = (h[i] + (s[i] - 'a' + 1) * b_pow[i]) % m;

    int cnt = 0;
    for (int l = 1; l <= n; l++)
    {
        set<long long> hs;
        for (int i = 0; i <= n - l; i++)
        {
            long cur_h = (h[i + l] + m - h[i]) % m;
            cur_h = (cur_h * b_pow[n - i - 1]) % m;
            hs.insert(cur_h);
        }
        cnt += hs.size();
    }
    return cnt;
}
```

## 2.7 后缀自动机

```
void sam_extend(char c)
{
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c))
    {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1)
    {
        st[cur].link = 0;
    }
    else
    {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len)
        {
            st[cur].link = q;
        }
        else
        {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q)
            {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
```

# 3 数据结构 Data Structure

## 3.1 并查集

```
#include <bits/stdc++.h>

using namespace std;

struct dsu
{
    vector<size_t> pa, size, sum;

    explicit dsu(size_t size_)
        : pa(size_ * 2), size(size_ * 2, 1), sum(size_ * 2)
    {
        // size 与 sum 的前半段其实没有使用，只是为了让下标计算更简单
        iota(pa.begin(), pa.begin() + size_, size_);
        iota(pa.begin() + size_, pa.end(), size_);
        iota(sum.begin() + size_, sum.end(), 0);
    }

    void unite(size_t x, size_t y)
    {
        x = find(x), y = find(y);
        if (x == y)
            return;
        if (size[x] < size[y])
            swap(x, y);
        pa[y] = x;
        size[x] += size[y];
        sum[x] += sum[y];
    }

    void move(size_t x, size_t y)
    {
        auto fx = find(x), fy = find(y);
        if (fx == fy)
            return;
        pa[x] = fy;
        --size[fx], ++size[fy];
        sum[fx] -= x, sum[fy] += x;
    }

    size_t find(size_t x) { return pa[x] == x ? x : pa[x] = find(pa[x]); }
};

int main()
{
    size_t n, m, op, x, y;
    while (cin >> n >> m)
    {
        dsu dsu(n + 1); // 元素范围是 1..n
        while (m--)
        {
            cin >> op;
            switch (op)
            {
            case 1:
                cin >> x >> y;
                dsu.unite(x, y);
                break;
            case 2:
                cin >> x >> y;
                dsu.move(x, y);
                break;
            case 3:
                cin >> x;
                x = dsu.find(x);
                cout << dsu.size[x] << ' ' << dsu.sum[x] << '\n';
                break;
            default:
                assert(false); // not reachable
            }
        }
    }
}
```

## 3.2 线段树

### 3.2.1 区间加/求和线段树

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

template <typename T>
class SegTreeLazyRangeAdd
{
    vector<T> tree, lazy;
    vector<T> *arr;
    int n, root, n4, end;

    void maintain(int cl, int cr, int p)
    {
        int cm = cl + (cr - cl) / 2;
        if (cl != cr && lazy[p])
        {
            lazy[p * 2] += lazy[p];
            lazy[p * 2 + 1] += lazy[p];
            tree[p * 2] += lazy[p] * (cm - cl + 1);
            tree[p * 2 + 1] += lazy[p] * (cr - cm);
            lazy[p] = 0;
        }
    }

    T range_sum(int l, int r, int cl, int cr, int p)
    {
        if (l <= cl && cr <= r)
            return tree[p];
        int m = cl + (cr - cl) / 2;
        T sum = 0;
        maintain(cl, cr, p);
        if (l <= m)
            sum += range_sum(l, r, cl, m, p * 2);
        if (r > m)
            sum += range_sum(l, r, m + 1, cr, p * 2 + 1);
        return sum;
    }

    void range_add(int l, int r, T val, int cl, int cr, int p)
    {
        if (l <= cl && cr <= r)
        {
            lazy[p] += val;
            tree[p] += (cr - cl + 1) * val;
            return;
        }
        int m = cl + (cr - cl) / 2;
        maintain(cl, cr, p);
        if (l <= m)
            range_add(l, r, val, cl, m, p * 2);
        if (r > m)
            range_add(l, r, val, m + 1, cr, p * 2 + 1);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
    }

    void build(int s, int t, int p)
    {
        if (s == t)
        {
            tree[p] = (*arr)[s];
            return;
        }
        int m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
    }
```

```
public:
    explicit SegTreeLazyRangeAdd<T>(vector<T> v)
    {
        n = v.size();
        n4 = n * 4;
        tree = vector<T>(n4, 0);
        lazy = vector<T>(n4, 0);
        arr = &v;
        end = n - 1;
        root = 1;
        build(0, end, 1);
        arr = nullptr;
    }

    void show(int p, int depth = 0)
    {
        if (p > n4 || tree[p] == 0)
            return;
        show(p * 2, depth + 1);
        for (int i = 0; i < depth; ++i)
            putchar('\t');
        printf("%d:%d\n", tree[p], lazy[p]);
        show(p * 2 + 1, depth + 1);
    }

    T range_sum(int l, int r) { return range_sum(l, r, 0, end, root); }

    void range_add(int l, int r, int val) { range_add(l, r, val, 0, end,
root); }
};
```

### 3.2.2 区间修改/求和线段树

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

template <typename T>
class SegTreeLazyRangeSet
{
    vector<T> tree, lazy;
    vector<T> *arr;
    int n, root, n4, end;

    void maintain(int cl, int cr, int p)
    {
        int cm = cl + (cr - cl) / 2;
        if (cl != cr && lazy[p])
        {
            lazy[p * 2] = lazy[p];
            lazy[p * 2 + 1] = lazy[p];
            tree[p * 2] = lazy[p] * (cm - cl + 1);
            tree[p * 2 + 1] = lazy[p] * (cr - cm);
            lazy[p] = 0;
        }
    }
    T range_sum(int l, int r, int cl, int cr, int p)
    {
        if (l <= cl && cr <= r)
            return tree[p];
        int m = cl + (cr - cl) / 2;
        T sum = 0;
        maintain(cl, cr, p);
        if (l <= m)
            sum += range_sum(l, r, cl, m, p * 2);
        if (r > m)
            sum += range_sum(l, r, m + 1, cr, p * 2 + 1);
        return sum;
    }

    void range_set(int l, int r, T val, int cl, int cr, int p)
    {
        if (l <= cl && cr <= r)
        {
            lazy[p] = val;
            tree[p] = (cr - cl + 1) * val;
            return;
        }
        int m = cl + (cr - cl) / 2;
        maintain(cl, cr, p);
        if (l <= m)
            range_set(l, r, val, cl, m, p * 2);
        if (r > m)
            range_set(l, r, val, m + 1, cr, p * 2 + 1);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
    }

    void build(int s, int t, int p)
    {
        if (s == t)
        {
            tree[p] = (*arr)[s];
            return;
        }
        int m = s + (t - s) / 2;
        build(s, m, p * 2);
        build(m + 1, t, p * 2 + 1);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
    }

public:
    explicit SegTreeLazyRangeSet<T>(vector<T> v)
    {
        n = v.size();
        n4 = n * 4;
        tree = vector<T>(n4, 0);
        lazy = vector<T>(n4, 0);
        arr = &v;
        end = n - 1;
        root = 1;
        build(0, end, 1);
        arr = nullptr;
    }

    void show(int p, int depth = 0)
    {
        if (p > n4 || tree[p] == 0)
            return;
        show(p * 2, depth + 1);
        for (int i = 0; i < depth; ++i)
            putchar('\t');
        printf("%d:%d\n", tree[p], lazy[p]);
        show(p * 2 + 1, depth + 1);
    }

    T range_sum(int l, int r) { return range_sum(l, r, 0, end, root); }

    void range_set(int l, int r, int val) { range_set(l, r, val, 0, end,
root); }
};
```

## 3.3 主席树

```
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
const int maxn = 1e5; // 数据范围
int tot, n, m;
int sum[(maxn << 5) + 10], rt[maxn + 10], ls[(maxn << 5) + 10],
    rs[(maxn << 5) + 10];
int a[maxn + 10], ind[maxn + 10], len;

inline int getid(const int &val)
{ // 离散化
    return lower_bound(ind + 1, ind + len + 1, val) - ind;
}

int build(int l, int r)
{ // 建树
    int root = ++tot;
    if (l == r)
        return root;
    int mid = l + r >> 1;
    ls[root] = build(l, mid);
    rs[root] = build(mid + 1, r);
    return root; // 返回该子树的根节点
}

int update(int k, int l, int r, int root)
{ // 插入操作
    int dir = ++tot;
    ls[dir] = ls[root], rs[dir] = rs[root], sum[dir] = sum[root] + 1;
    if (l == r)
```

```
        return dir;
    int mid = l + r >> 1;
    if (k <= mid)
        ls[dir] = update(k, l, mid, ls[dir]);
    else
        rs[dir] = update(k, mid + 1, r, rs[dir]);
    return dir;
}

int query(int u, int v, int l, int r, int k)
{ // 查询操作
    int mid = l + r >> 1,
        x = sum[ls[v]] - sum[ls[u]]; // 通过区间减法得到左儿子中所存储的数
值个数
    if (l == r)
        return l;
    if (k <= x) // 若 k 小于等于 x ，则说明第 k 小的数字存储在在左儿子中
        return query(ls[u], ls[v], l, mid, k);
    else // 否则说明在右儿子中
        return query(rs[u], rs[v], mid + 1, r, k - x);
}

inline void init()
{
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d", a + i);
    memcpy(ind, a, sizeof ind);
    sort(ind + 1, ind + n + 1);
    len = unique(ind + 1, ind + n + 1) - ind - 1;
    rt[0] = build(1, len);
    for (int i = 1; i <= n; ++i)
        rt[i] = update(getid(a[i]), 1, len, rt[i - 1]);
}

int l, r, k;

inline void work()
{
    while (m--)
    {
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", ind[query(rt[l - 1], rt[r], 1, len, k)]); // 回答
询问
    }
}

int main()
{
    init();
    work();
    return 0;
}
```

## 3.4 划分树

```
#define _mid(a, b) ((a + b) / 2)

using namespace std;
typedef long long ll;
const int maxn = 1e5 + 10;
const int INF = 0x3f3f3f3f;
int sorted[maxn];
int cnt[20][maxn];
int tree[20][maxn];
void build(int l, int r, int k)
{
    if (r == l) // 如果区间内只有一个数，返回
        return;
    int mid = _mid(l, r), flag = mid - l + 1; // 求出flag
    for (int i = l; i <= r; i++)
        if (tree[k][i] < sorted[mid]) // sorted代表排序好了的数组
            flag--;
    int bufl = l, bufr = mid + 1;
    for (int i = l; i <= r; i++)
    {
        cnt[k][i] = (i == l) ? 0 : cnt[k][i - 1]; // 初始化
        if (tree[k][i] < sorted[mid] || tree[k][i] == sorted[mid] && flag > 0)
        { // 如果有多个中值
            tree[k + 1][bufl++] = tree[k][i];
            cnt[k][i]++; // 进入左子树
            if (tree[k][i] == sorted[mid])
                flag--;
        }
        else // 进入右子树
            tree[k + 1][bufr++] = tree[k][i];
    }
    build(l, mid, k + 1);
    build(mid + 1, r, k + 1);
}

int ask(int k, int sl, int sr, int l, int r, int x)
{
    if (sl == sr)
        return tree[k][sl];
    int cntl;
    cntl = (l == sl) ? 0 : cnt[k][l - 1]; // 是否和查询区间重合
    int cntl2r = cnt[k][r] - cntl;        // 计算l到r有cntl2r个数进入左子树
    if (cntl2r >= x)                      // 如果大于当前查询的k则进入左子树（因为左子树中最大的数大于第k大的数）
        return ask(k + 1, sl, _mid(sl, sr), sl + cntl, sl + cnt[k][r] - 1, x);
    else
    { // 否则进入右子树
        int lr = _mid(sl, sr) + 1 + (l - sl - cntl);
        return ask(k + 1, _mid(sl, sr) + 1, sr, lr, lr + r - l - cntl2r,
x - cntl2r);
    }
}
```

## 3.5 单调栈

```
const int N = 100010;

stack<int> s;
int a[N], n;

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

    for (int i = 1; i <= n; i++) {
        while(!s.empty() && a[s.top()] >= a[i]) s.pop();
        if(!s.empty()) cout << a[s.top()] << ' ';
        else cout << -1 << ' ';
        s.push(i);
    }
    return 0;
}
```

## 3.6 伸展树

```
#include <cstdio>
const int N = 100005;
int rt, tot, fa[N], ch[N][2], val[N], cnt[N], sz[N];

struct Splay
{
    void maintain(int x) { sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + cnt[x]; }

    bool get(int x) { return x == ch[fa[x]][1]; }

    void clear(int x)
    {
        ch[x][0] = ch[x][1] = fa[x] = val[x] = sz[x] = cnt[x] = 0;
    }

    void rotate(int x)
    {
        int y = fa[x], z = fa[y], chk = get(x);
        ch[y][chk] = ch[x][chk ^ 1];
        if (ch[x][chk ^ 1])
            fa[ch[x][chk ^ 1]] = y;
        ch[x][chk ^ 1] = y;
        fa[y] = x;
        fa[x] = z;
        if (z)
            ch[z][y == ch[z][1]] = x;
        maintain(y);
        maintain(x);
    }

    void splay(int x)
    {
        for (int f = fa[x]; f = fa[x], f; rotate(x))
            if (fa[f])
                rotate(get(x) == get(f) ? f : x);
        rt = x;
    }

    void ins(int k)
    {
        if (!rt)
        {
            val[++tot] = k;
            cnt[tot]++;
            rt = tot;
            maintain(rt);
            return;
        }
        int cur = rt, f = 0;
        while (1)
        {
            if (val[cur] == k)
            {
                cnt[cur]++;
                maintain(cur);
                maintain(f);
                splay(cur);
                break;
            }
            f = cur;
            cur = ch[cur][val[cur] < k];
            if (!cur)
            {
                val[++tot] = k;
                cnt[tot]++;
                fa[tot] = f;
                ch[f][val[f] < k] = tot;
                maintain(tot);
                maintain(f);
                splay(tot);
                break;
            }
        }
    }

    int rk(int k)
    {
        int res = 0, cur = rt;
        while (1)
        {
            if (k < val[cur])
            {
                cur = ch[cur][0];
            }
            else
            {
                res += sz[ch[cur][0]];
                if (k == val[cur])
                {
                    splay(cur);
                    return res + 1;
                }
                res += cnt[cur];
                cur = ch[cur][1];
            }
        }
    }

    int kth(int k)
    {
        int cur = rt;
        while (1)
        {
            if (ch[cur][0] && k <= sz[ch[cur][0]])
            {
                cur = ch[cur][0];
            }
            else
            {
                k -= cnt[cur] + sz[ch[cur][0]];
                if (k <= 0)
                {
                    splay(cur);
                    return val[cur];
                }
                cur = ch[cur][1];
            }
        }
    }

    int pre()
    {
        int cur = ch[rt][0];
        if (!cur)
            return cur;
        while (ch[cur][1])
            cur = ch[cur][1];
        splay(cur);
        return cur;
    }

    int nxt()
    {
        int cur = ch[rt][1];
        if (!cur)
            return cur;
        while (ch[cur][0])
            cur = ch[cur][0];
        splay(cur);
        return cur;
    }

    void del(int k)
    {
        rk(k);
        if (cnt[rt] > 1)
        {
            cnt[rt]--;
            maintain(rt);
            return;
        }
        if (!ch[rt][0] && !ch[rt][1])
        {
            clear(rt);
            rt = 0;
            return;
        }
        if (!ch[rt][0])
        {
            int cur = rt;
            rt = ch[rt][1];
            fa[rt] = 0;
            clear(cur);
            return;
        }
        if (!ch[rt][1])
        {
            int cur = rt;
            rt = ch[rt][0];
            fa[rt] = 0;
            clear(cur);
            return;
        }
        int cur = rt;
        int x = pre();
        fa[ch[cur][1]] = x;
        ch[x][1] = ch[cur][1];
        clear(cur);
        maintain(rt);
    }
} tree;

int main()
{
    int n, opt, x;
    for (scanf("%d", &n); n; --n)
    {
        scanf("%d%d", &opt, &x);
        if (opt == 1)
            tree.ins(x);
        else if (opt == 2)
            tree.del(x);
        else if (opt == 3)
            printf("%d\n", tree.rk(x));
        else if (opt == 4)
            printf("%d\n", tree.kth(x));
        else if (opt == 5)
            tree.ins(x), printf("%d\n", val[tree.pre()]), tree.del(x);
        else
            tree.ins(x), printf("%d\n", val[tree.nxt()]), tree.del(x);
    }
    return 0;
}
```

## 3.7 替罪羊树

```
const int MAXN = 1500005, INF = 0x7f7f7f7f;
const double ALPHA = 0.7;
int L[MAXN], R[MAXN], N[MAXN], val[MAXN], size[MAXN], cnt = 1, FV[MAXN],
FN[MAXN];
int flatten(int pos, int *fv, int *fn) // 拉平
{
    int l = 0, r = 0, unempty = N[pos] != 0;
    if (L[pos])
        l = flatten(L[pos], fv, fn);
    if (unempty)
    {
        fv[l] = val[pos];
        fn[l] = N[pos];
    }
    if (R[pos])
        r = flatten(R[pos], fv + l + unempty, fn + l + unempty);
    return l + r + unempty;
}
void rebuild(int pos, int l, int r) // 重建
{
    int mid = (l + r) / 2, sz1 = 0, sz2 = 0;
    if (l < mid)
    {
        L[pos] = ++cnt;
        rebuild(L[pos], l, mid - 1);
        sz1 = size[L[pos]];
    }
    else
        L[pos] = 0;
    if (mid < r)
    {
        R[pos] = ++cnt;
        rebuild(R[pos], mid + 1, r);
        sz2 = size[R[pos]];
    }
    else
        R[pos] = 0;
    val[pos] = FV[mid];
    N[pos] = FN[mid];
    size[pos] = sz1 + sz2 + N[pos];
}
void try_restructure(int pos) // 尝试重构当前子树
{
    double k = max(size[L[pos]], size[R[pos]]) / double(size[pos]);
    if (k > ALPHA)
    {
        int sz = flatten(pos, FV, FN);
        rebuild(pos, 0, sz - 1);
    }
}
void insert(int v, int pos = 1) // 插入
{
    size[pos]++;
    if (N[pos] == 0 && L[pos] == 0 && R[pos] == 0)
    {
        val[pos] = v;
        N[pos] = 1;
    }
    else if (v < val[pos])
    {
        if (L[pos] == 0)
            L[pos] = ++cnt;
        insert(v, L[pos]);
    }
    else if (v > val[pos])
    {
        if (R[pos] == 0)
            R[pos] = ++cnt;
        insert(v, R[pos]);
    }
    else
        N[pos]++;
    try_restructure(pos);
}
void remove(int v, int pos = 1) // 删除
{
    size[pos]--;
    if (v < val[pos])
        remove(v, L[pos]);
    else if (v > val[pos])
        remove(v, R[pos]);
    else
        N[pos]--;
    try_restructure(pos);
}
int countl(int v, int pos = 1) // 统计比 v 小的数的个数
{
    if (v < val[pos])
        return L[pos] ? countl(v, L[pos]) : 0;
    else if (v > val[pos])
        return size[L[pos]] + N[pos] + (R[pos] ? countl(v, R[pos]) : 0);
    else
        return size[L[pos]];
```

```
}
int countg(int v, int pos = 1) // 统计比v大的数的个数
{
    if (v > val[pos])
        return R[pos] ? countg(v, R[pos]) : 0;
    else if (v < val[pos])
        return size[R[pos]] + N[pos] + (L[pos] ? countg(v, L[pos]) : 0);
    else
        return size[R[pos]];
}
int rank(int v) // 求排名
{
    return countl(v) + 1;
}
int kth(int k, int pos = 1) // 求指定排名的数
{
    if (size[L[pos]] + 1 > k)
        return kth(k, L[pos]);
    else if (size[L[pos]] + N[pos] < k)
        return kth(k - size[L[pos]] - N[pos], R[pos]);
    else
        return val[pos];
}
int pre(int v) // 求前驱
{
    int r = countl(v);
    return kth(r);
}
int suc(int v) // 求后继
{
    int r = size[1] - countg(v) + 1;
    return kth(r);
}
```

## 3.8 一维树状数组

### 3.8.1 单点修改 + 区间查询

```
inline int lowbit(int x) // 求二进制下最低位的1
{
    return x & -x;
}

inline void add(int x, int d) //单点修改  给第x号元素 + d
{
    while (x <= n) {
        aa[x] += d;
        x += lowbit(x);
    }
}
```

```
inline int ask(int x) // 求从第一个元素 到 第x个元素的总和
{
    int res = 0;
    while (x) {
        res += aa[x];
        x -= lowbit(x);
    }
    return res;
}
inline int getsum(int x, int y) // 求区间[l, r]内的元素总和
{
    return ask(r) - ask(l - 1);
}
```

### 3.8.2 区间修改 + 单点查询

```
inline int lowbit(int x) // 求二进制下最低位的1
{
    return x & -x;
}

void add(int x, int d) // 修改点
{
    while (x <= n)
    {
        p[x] += d;
        x += lowbit(x);
    }
}

void range_add(int l, int r, int d) // 经过差分处理后， p数组修改只要在第l个元素的 +d 和第r + 1个元素-d即可完成区间修改
{
    add(l, d);
    add(r + 1, -d);
}

int ask(int x) // 求第x个位置的元素值
{
    int res = 0;
    while (x)
    {
        res += p[x];
        x -= lowbit(x);
    }
    return res;
}
```

### 3.8.3 区间修改 + 区间查询

```
typedef long long ll;
```

```
ll sum1[N], sum2[N];

inline int lowbit(int x) // 求二进制下最低位的1
{
    return x & -x;
}

inline void add(ll x, ll d) // 对sum1数组和sum2数组进行维护
{
    for (int i = x; i <= n; i += lowbit(i))
    {
        sum1[i] += d;
        sum2[i] += x * d;
    }
}
inline void range_add(ll l, ll r, ll d) // 差分思想进行区间维护
{
    add(l, d);
    add(r + 1, -d);
}

inline ll ask(ll x) // 求[1, x]内的所有元素之和
{
    ll res = 0;
    for (int i = x; i > 0; i -= lowbit(i))
    {
        res += (x + 1) * sum1[i] - sum2[i];
    }
    return res;
}

inline ll range_ask(ll l, ll r) // 求区间[l, r]内所有元素之和
{
    return ask(r) - ask(l - 1);
}
```

### 3.8.4 求逆序对数量

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

typedef long long ll;
const int N = 5e5 + 7;

int n;
int tr[N], a[N];
vector<int> mp;
ll ans;

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

```
void add(int x, int d)
{
    while (x <= n)
    {
        tr[x] += d;
        x += lowbit(x);
    }
}

int ask(int x)
{
    int res = 0;
    while (x)
    {
        res += tr[x];
        x -= lowbit(x);
    }
    return res;
}

inline int get_id(int x)
{
    return lower_bound(mp.begin(), mp.end(), x) - mp.begin() + 1;
}

int main()
{
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]), mp.push_back(a[i]);
    // 离散化
    sort(mp.begin(), mp.end());
    mp.erase(unique(mp.begin(), mp.end()), mp.end());
    // 先序遍历找这个点前有多少个点比它大
    for (int i = 1; i <= n; ++i)
    {
        int u = get_id(a[i]);
        add(u, 1);
        ans = ans + i - ask(u);
    }
    /*
    //后序遍历，看在这个点后有多少个点比它小
    for (int i = n; i > 0; --i) {
        int u = get_id(a[i]);
        add(u, 1);
        ans += ask(u - 1);
    */
    printf("%lld\n", ans);
    return 0;
}
```

## 3.9 二维树状数组

### 3.9.1 单点修改 + 二维区间查询

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

void add(int x, int y, int d)
{
    for (int i = x; i <= n; i += lowbit(i))
        for (int j = y; j <= n; j += lowbit(j))
            a[i][j] += d;
}

int ask(int x, int y)
{
    int res = 0;
    for (int i = x; i > 0; i -= lowbit(i))
        for (int j = y; j > 0; j -= lowbit(j))
            res += a[i][j];
    return res;
}

int range_ask(int x1, int y1, int x2, int y2)
{
    return ask(x2, y2) - ask(x1 - 1, y2) - ask(x2, y1 - 1) + ask(x1 - 1,
y1 - 1);
}
```

### 3.9.2 二维区间修改 + 单点查询

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

void update(int x, int y,  int d) //单点修改
{
    for (int i = x; i <= n; i += lowbit(i))
        for (int j = y; j <= m; j += lowbit(j));
            tree[i][j] += d;
}

void ranger_update(int x1, int y1, int x2, int y2, int d) //二维区间修改
{
    update(x1, y1, d);
    update(x1, y2 + 1, -d);
    update(x2 + 1, y1, -d);
    update(x2 + 1, y2 + 1, d);
}

void ask(int x, int y)
{
    int res = 0;
    for (int i = x; i > 0; i -= lowbit(i))
        for (int j = y; j > 0; j -= lowbit(j))
            res += tree[i][j];
}
```

### 3.9.3 二维区间修改 + 二维区间查询

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

void update(int x, int y,  int d) // 单点修改
{
    for (int i = x; i <= n; i += lowbit(i))
        for (int j = y; j <= m; j += lowbit(j));
    tree[i][j] += d;
}

void ranger_update(int x1, int y1, int x2, int y2, int d) // 二维区间修改
{
    update(x1, y1, d);
    update(x1, y2 + 1, -d);
    update(x2 + 1, y1, -d);
    update(x2 + 1, y2 + 1, d);
}

int ask(int x, int y) // 单点查询
{
    int res = 0;
    for (int i = x; i > 0; i -= lowbit(i))
        for (int j = y; j > 0; j -= lowbit(j))
            res += tree[i][j];
    return res;
}

int ranger_ask(int x1, int y1, int x2, int y2) // 利用二维差分进行二维区间
查询
{
    return ask(x2, y2) - ask(x2, y1 - 1) - ask(x1 - 1, y2) + ask(x1 - 1,
y1 - 1);
}
```

# 4 数学 Number Theory

## 4.1 Euler 筛法

```
void Euler(int n)
{
    for (int i = 2; i <= n; ++i)
    {
        if (!vis[i])
        {
            pri[cnt++] = i;
        }
        for (int j = 0; j < cnt; ++j)
        {
            if (1ll * i * pri[j] > n) break;
            vis[i * pri[j]] = 1;
            if (i % pri[j] == 0) break;
        }
    }
}
```

## 4.2 分解质因数

```
void divide(int x)
{
    for (int i=2;i<=x/i;i++) { //找当前的x的大于等于i的质数
        if (x%i==0) {//i从2开始，满足条件x%i为0的i一定都是质数，合数肯定
不符合条件
            int s=0;
            while (x%i==0) s++,x/=i; //去除掉x的质因子i，所以x不再包含2~i
之间的质因子了
            printf("%d %d\n",i,s); //输出i^s
        }
    }
    if (x>1) printf("%d %d\n",x,1);//最后剩下的x要么是个质数，要么是个1
    puts("");
}
```

## 4.3 快速幂

```
long long qpow(double x, long long n) {
    long long ans = 1.0;
    long long y = x;
    while (n > 0) {
        if (n & 1) ans = ans * y % MOD;
        y = y * y % MOD;
        n >>= 1;
    }
    return ans;
}
```

## 4.4 牛顿迭代法

```
double sqrt_newton(double n)
```

```
{
    const double eps = 1E-15;
    double x = 1;
    while (true)
    {
        double nx = (x + n / x) / 2;
        if (abs(x - nx) < eps)
            break;
        x = nx;
    }
    return x;
}
```

## 4.5 快速数论变换

```
//大数相乘
inline int read()
{
    int x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9')
    {
        if (ch == '-')
            f = -1;
        ch = getchar();
    }
    while (ch <= '9' && ch >= '0')
    {
        x = 10 * x + ch - '0';
        ch = getchar();
    }
    return x * f;
}

void print(int x)
{
    if (x < 0)
        putchar('-'), x = -x;
    if (x >= 10)
        print(x / 10);
    putchar(x % 10 + '0');
}

const int N = 300100, P = 998244353;

inline int qpow(int x, int y)
{
    int res(1);
    while (y)
    {
        if (y & 1)
            res = 1ll * res * x % P;
        x = 1ll * x * x % P;
        y >>= 1;
    }
    return res;
}

int r[N];

void ntt(int *x, int lim, int opt)
{
    int i, j, k, m, gn, g, tmp;
    for (i = 0; i < lim; ++i)
        if (r[i] < i)
            swap(x[i], x[r[i]]);
    for (m = 2; m <= lim; m <<= 1)
    {
        k = m >> 1;
        gn = qpow(3, (P - 1) / m);
        for (i = 0; i < lim; i += m)
        {
            g = 1;
            for (j = 0; j < k; ++j, g = 1ll * g * gn % P)
            {
                tmp = 1ll * x[i + j + k] * g % P;
                x[i + j + k] = (x[i + j] - tmp + P) % P;
                x[i + j] = (x[i + j] + tmp) % P;
            }
        }
    }
    if (opt == -1)
    {
        reverse(x + 1, x + lim);
        int inv = qpow(lim, P - 2);
        for (i = 0; i < lim; ++i)
            x[i] = 1ll * x[i] * inv % P;
    }
}

int A[N], B[N], C[N];

char a[N], b[N];

int main()
{
    int i, lim(1), n;
    scanf("%s", &a);
    n = strlen(a);
    for (i = 0; i < n; ++i)
        A[i] = a[n - i - 1] - '0';
    while (lim < (n << 1))
        lim <<= 1;
    scanf("%s", &b);
    n = strlen(b);
    for (i = 0; i < n; ++i)
        B[i] = b[n - i - 1] - '0';
    while (lim < (n << 1))
        lim <<= 1;
    for (i = 0; i < lim; ++i)
```

```
        r[i] = (i & 1) * (lim >> 1) + (r[i >> 1] >> 1);
    ntt(A, lim, 1);
    ntt(B, lim, 1);
    for (i = 0; i < lim; ++i)
        C[i] = 1ll * A[i] * B[i] % P;
    ntt(C, lim, -1);
    int len(0);
    for (i = 0; i < lim; ++i)
    {
        if (C[i] >= 10)
            len = i + 1, C[i + 1] += C[i] / 10, C[i] %= 10;
        if (C[i])
            len = max(len, i);
    }
    while (C[len] >= 10)
        C[len + 1] += C[len] / 10, C[len] %= 10, len++;
    for (i = len; ~i; --i)
        putchar(C[i] + '0');
    puts("");
    return 0;
}
```

## 4.6 线性同余方程

```
int 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;
}

bool liEu(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;
}
```

## 4.7 中国剩余定理

```
LL CRT(int k, LL *a, LL *r)
{
    LL n = 1, ans = 0;
    for (int i = 1; i <= k; i++)
        n = n * r[i];
    for (int i = 1; i <= k; i++)
    {
        LL m = n / r[i], b, y;
        exgcd(m, r[i], b, y); // b * m mod r[i] = 1
        ans = (ans + a[i] * m * b % n) % n;
    }
    return (ans % n + n) % n;
}
```

## 4.8 扩展欧几里得算法

```
void exgcd(int a, int b, int& x, int& y) {      //求逆元
  if (b == 0) {
    x = 1, y = 0;
    return;
  }
  exgcd(b, a % b, y, x);
  y -= a / b * x;
}
```

## 4.9 高斯消元

```
std::bitset<1010> matrix[2010];  // matrix[1~n]：增广矩阵，0 位置为常数

std::vector<bool> GaussElimination(
    int n, int m)  // n 为未知数个数，m 为方程个数，返回方程组的解
                   //  （多解 / 无解返回一个空的 vector）
{
  for (int i = 1; i <= n; i++) {
    int cur = i;
    while (cur <= m && !matrix[cur].test(i)) cur++;
    if (cur > m) return std::vector<bool>(0);
    if (cur != i) swap(matrix[cur], matrix[i]);
    for (int j = 1; j <= m; j++)
      if (i != j && matrix[j].test(i)) matrix[j] ^= matrix[i];
  }
  std::vector<bool> ans(n + 1, 0);
  for (int i = 1; i <= n; i++) ans[i] = matrix[i].test(0);
  return ans;
}
```

## 4.10 高精度计算

```
#include <cstdio>
```

```
#include <cstring>

static const int LEN = 1004;

int a[LEN], b[LEN], c[LEN], d[LEN];

void clear(int a[])
{
    for (int i = 0; i < LEN; ++i)
        a[i] = 0;
}

void read(int a[])
{
    static char s[LEN + 1];
    scanf("%s", s);

    clear(a);

    int len = strlen(s);
    for (int i = 0; i < len; ++i)
        a[len - i - 1] = s[i] - '0';
}

void print(int a[])
{
    int i;
    for (i = LEN - 1; i >= 1; --i)
        if (a[i] != 0)
            break;
    for (; i >= 0; --i)
        putchar(a[i] + '0');
    putchar('\n');
}

void add(int a[], int b[], int c[])
{
    clear(c);

    for (int i = 0; i < LEN - 1; ++i)
    {
        c[i] += a[i] + b[i];
        if (c[i] >= 10)
        {
            c[i + 1] += 1;
            c[i] -= 10;
        }
    }
}

void sub(int a[], int b[], int c[])
{
    clear(c);

    for (int i = 0; i < LEN - 1; ++i)
    {
        c[i] += a[i] - b[i];
        if (c[i] < 0)
        {
            c[i + 1] -= 1;
            c[i] += 10;
        }
    }
}

void mul(int a[], int b[], int c[])
{
    clear(c);

    for (int i = 0; i < LEN - 1; ++i)
    {
        for (int j = 0; j <= i; ++j)
            c[i] += a[j] * b[i - j];

        if (c[i] >= 10)
        {
            c[i + 1] += c[i] / 10;
            c[i] %= 10;
        }
    }
}

inline bool greater_eq(int a[], int b[], int last_dg, int len)
{
    if (a[last_dg + len] != 0)
        return true;
    for (int i = len - 1; i >= 0; --i)
    {
        if (a[last_dg + i] > b[i])
            return true;
        if (a[last_dg + i] < b[i])
            return false;
    }
    return true;
}

void div(int a[], int b[], int c[], int d[])
{
    clear(c);
    clear(d);

    int la, lb;
    for (la = LEN - 1; la > 0; --la)
        if (a[la - 1] != 0)
            break;
    for (lb = LEN - 1; lb > 0; --lb)
        if (b[lb - 1] != 0)
            break;
    if (lb == 0)
    {
        puts("> <");
        return;
    }

    for (int i = 0; i < la; ++i)
        d[i] = a[i];
    for (int i = la - lb; i >= 0; --i)
    {
        while (greater_eq(d, b, i, lb))
        {
            for (int j = 0; j < lb; ++j)
            {
                d[i + j] -= b[j];
                if (d[i + j] < 0)
                {
                    d[i + j + 1] -= 1;
                    d[i + j] += 10;
                }
            }
            c[i] += 1;
        }
    }
}

int main()
{
    read(a);

    char op[4];
    scanf("%s", op);

    read(b);

    switch (op[0])
    {
    case '+':
        add(a, b, c);
        print(c);
        break;
    case '-':
        sub(a, b, c);
        print(c);
        break;
    case '*':
        mul(a, b, c);
        print(c);
        break;
    case '/':
        div(a, b, c, d);
        print(c);
        print(d);
        break;
    default:
        puts("> <");
    }

    return 0;
}
```

# 5 动态规划 Dynamic Programming

## 5.1 背包 DP

```
// 0-1 背包问题母代码(二维)
void bags()
{
    vector<int> weight = {1, 3, 4};   //各个物品的重量
    vector<int> value = {15, 20, 30}; //对应的价值
    int bagWeight = 4;                //背包最大能放下多少重的物品

    // 二维数组：状态定义:dp[i][j]表示从 0-i 个物品中选择不超过 j 重量的物品的最
大价值
    vector<vector<int>> dp(weight.size() + 1, vector<int>(bagWeight + 1,
0));

    // 初始化:第一列都是 0，第一行表示只选取 0 号物品最大价值
    for (int j = bagWeight; j >= weight[0]; j--)
        dp[0][j] = dp[0][j - weight[0]] + value[0];

    // weight 数组的大小 就是物品个数
    for (int i = 1; i < weight.size(); i++) // 遍历物品(第 0 个物品已经初始
化)
    {
        for (int j = 0; j <= bagWeight; j++) // 遍历背包容量
        {
            if (j < weight[i])           //背包容量已经不足以拿第 i 个物品了
                dp[i][j] = dp[i - 1][j]; //最大价值就是拿第 i-1 个物品的最大
价值
             //背包容量足够拿第 i 个物品,可拿可不拿：拿了最大价值是前 i-1 个物品
扣除第 i 个物品的 重量的最大价值加上 i 个物品的价值
            //不拿就是前 i-1 个物品的最大价值,两者进行比较取较大的
            else
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weight[i]] +
value[i]);
        }
    }
    cout << dp[weight.size() - 1][bagWeight] << endl;
}
// 首先是背包分类的模板：
// 1、0/1 背包：外循环 nums,内循环 target,target 倒序且 target>=nums[i];
// 2、完全背包：外循环 nums,内循环 target,target 正序且 target>=nums[i];
// 3、组合背包：外循环 target,内循环 nums,target 正序且 target>=nums[i];
// 4、分组背包：这个比较特殊，需要三重循环：外循环背包 bags,内部两层循环根据题目
的要求转化为 1,2,3 三种背包类型的模板

// 然后是问题分类的模板：
// 1、最值问题：dp[i] = max/min(dp[i], dp[i-nums]+1)或dp[i] = max/min(dp[i],
dp[i-num]+nums);
// 2、存在问题(bool)：dp[i]=dp[i]||dp[i-num];
// 3、组合问题：dp[i]+=dp[i-num];
```

## 5.2 数位 DP

```
//给定两个正整数 a,b，求在 [a,b] 中的所有整数中，每个数码 (digit) 各出现了多少
次。
#include <bits/stdc++.h>
using namespace std;
const int N = 15;
typedef long long ll;
ll l, r, dp[N], sum[N], mi[N];
ll ans1[N], ans2[N];
int a[N];

inline void solve(ll n, ll *ans) {
  ll tmp = n;
  int len = 0;
  while (n) a[++len] = n % 10, n /= 10;
  for (int i = len; i >= 1; --i) {
    for (int j = 0; j < 10; j++) ans[j] += dp[i - 1] * a[i];
    for (int j = 0; j < a[i]; j++) ans[j] += mi[i - 1];
    tmp -= mi[i - 1] * a[i], ans[a[i]] += tmp + 1;
    ans[0] -= mi[i - 1];
  }
}

int main() {
  scanf("%lld%lld", &l, &r);
  mi[0] = 1ll;
  for (int i = 1; i <= 13; ++i) {
    dp[i] = dp[i - 1] * 10 + mi[i - 1];
    mi[i] = 10ll * mi[i - 1];
  }
  solve(r, ans1), solve(l - 1, ans2);
  for (int i = 0; i < 10; ++i) printf("%lld ", ans1[i] - ans2[i]);
  return 0;
}
```

## 5.3 插头 DP

```
//在 N * M 的棋盘内铺满 1 * 2 或 2 * 1 的多米诺骨牌，求方案数。
#include <bits/stdc++.h>
using namespace std;
const int N = 11;
long long f[2][1 << N], *f0, *f1;
int n, m;

int main() {
  while (cin >> n >> m && n) {
    f0 = f[0];
    f1 = f[1];
    fill(f1, f1 + (1 << m), 0);
    f1[0] = 1;
    for (int i = 0; i < n; ++i) {
      for (int j = 0; j < m; ++j) {
        swap(f0, f1);
        fill(f1, f1 + (1 << m), 0);
#define u f0[s]
        for (int s = 0; s < 1 << m; ++s)
          if (u) {
             if (j != m - 1 && (!(s >> j & 3))) f1[s ^ 1 << j + 1] +=
u;  // 横放
            f1[s ^ 1 << j] += u;  // 竖放或不放
          }
      }
    }
    cout << f1[0] << endl;
  }
}
```

## 5.4 线性 DP

```
//最长上升子序列 LIS
const int N = 1e4 + 1;
int a[N];
int d[N];
int main() {
  int n;
  cin >> n;
  for (int i = 1;i <= n;i++)
    cin >> a[i];
  d[1] = a[1];//将第一个元素存入
  int len = 1;
  for (int i = 2;i <= n;i++) {
      if (a[i] > d[len]) d[++len] = a[i];
      else {
        int j = lower_bound(d+1,d + len + 1,a[i]) - d;
        d[j] = a[i];//替换序列中比他第一个大的元素
      }
  }
  cout << len;
  return 0;
}
```

# 6 计算几何 Computational Geometry

## 6.1 常数定义相关

```
//long double 的输入输出
scanf("%Lf" , &a);
printf("%.10Lf" , a);
//常用函数:fabsl(a),cosl(a).....
//即在末尾加上了字母 l
//常数定义
const double eps = 1e-8;
const double PI = acos(-1.0);

int sgn(double x)//符号函数，eps 使用最多的地方
{
    if (fabs(x) < eps)
        return 0;
    if (x < 0)
        return -1;
    else
        return 1;
}
```

## 6.2 点类

```
struct Point        //点类
{
    double x, y;
    Point() {}
    Point(double _x, double _y) : x(_x), y(_y) {}
     Point operator-(const Point &b) const { return Point(x - b.x, y -
b.y); }
     Point operator+(const Point &b) const { return Point(x + b.x, y +
b.y); }

    double operator^(const Point &b) const { return x * b.y - y * b.x; }
//叉积
    double operator*(const Point &b) const { return x * b.x + y * b.y; }
//点积

    bool operator<(const Point &b) const { return x < b.x || (x == b.x &&
y < b.y); }
    bool operator==(const Point &b) const { return sgn(x - b.x) == 0 &&
sgn(y - b.y) == 0; }

    Point Rotate(double B, Point P) //绕着点P，逆时针旋转角度B(弧度)
    {
        Point tmp;
        tmp.x = (x - P.x) * cos(B) - (y - P.y) * sin(B) + P.x;
        tmp.y = (x - P.x) * sin(B) + (y - P.y) * cos(B) + P.y;
        return tmp;
    }
};

double dist(Point a, Point b) { return sqrt((a - b) * (a - b)); } //两点
```

```
间距离
double len(Point a){return sqrt(a.x * a.x + a.y * a.y);}//向量的长度
```

## 6.3 直线类

```
struct Line     //直线类
{
    Point s, e;
    Line() {}
    Line(Point _s, Point _e) : s(_s), e(_e) {}

    //两直线相交求交点
    //第一个值为0表示直线重合，为1表示平行,为2是相交
    //只有第一个值为2时，交点才有意义

    pair<int, Point> operator&(const Line &b) const
    {
        Point res = s;
        if (sgn((s - e) ^ (b.s - b.e)) == 0)
        {
            if (sgn((s - b.e) ^ (b.s - b.e)) == 0)
                return make_pair(0, res); //重合
            else
                return make_pair(1, res); //平行
        }
        double t = ((s - b.s) ^ (b.s - b.e)) / ((s - e) ^ (b.s - b.e));
        res.x += (e.x - s.x) * t;
        res.y += (e.y - s.y) * t;
        return make_pair(2, res);
    }
};
```

## 6.4 相交关系判断

```
bool inter(Line l1, Line l2)       //判断线段是否相交
{
    return max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
            max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
            max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
            max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
              sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^
(l1.s - l1.e)) <= 0 &&
              sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e - l2.e) ^
(l2.s - l2.e)) <= 0;
}
bool Seg_inter_line(Line l1, Line l2)       //判断直线和线段是否相交
{
    return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s
- l1.e)) <= 0;
}
```

## 6.5 距离相关计算

```
Point PointToLine(Point P, Line L)      //求点到直线的距离
{
    Point result;
    double t = ((P - L.s) * (L.e - L.s)) / ((L.e - L.s) * (L.e - L.s));
    result.x = L.s.x + (L.e.x - L.s.x) * t;
    result.y = L.s.y + (L.e.y - L.s.y) * t;
    return result;
}
Point NearestPointToLineSeg(Point P, Line L)    //求点到线段的距离
{
    Point result;
    double t = ((P - L.s) * (L.e - L.s)) / ((L.e - L.s) * (L.e - L.s));
    if (t >= 0 && t <= 1)
    {
        result.x = L.s.x + (L.e.x - L.s.x) * t;
        result.y = L.s.y + (L.e.y - L.s.y) * t;
    }
    else
    {
        if (dist(P, L.s) < dist(P, L.e))
            result = L.s;
        else
            result = L.e;
    }
    return result;
}
```

## 6.6 点和直线相关

```
//计算多边形面积,点的编号从 0~n-1
double CalcArea(Point p[], int n)
{
    double res = 0;
    for (int i = 0; i < n; i++)
        res += (p[i] ^ p[(i + 1) % n]) / 2;
    return fabs(res);
}
//*判断点在线段上
bool OnSeg(Point P, Line L)
{
    return sgn((L.s - P) ^ (L.e - P)) == 0 &&
            sgn((P.x - L.s.x) * (P.x - L.e.x)) <= 0 &&
            sgn((P.y - L.s.y) * (P.y - L.e.y)) <= 0;
}
```

## 6.7 凸包相关

```
int ConvexHull(Point *p, int n, Point *ch) //求凸包Andrew算法
{
    sort(p, p + n);
    n = unique(p, p + n) - p; //去重
    int m = 0;
    for (int i = 0; i < n; ++i)
    {
        while (m > 1 && sgn((ch[m - 1] - ch[m - 2]) ^ (p[i] - ch[m - 1])) 
<= 0)
            --m;
        ch[m++] = p[i];
    }
    int k = m;
    for (int i = n - 2; i >= 0; i--)
    {
        while (m > k && sgn((ch[m - 1] - ch[m - 2]) ^ (p[i] - ch[m - 1])) 
<= 0)
            --m;
        ch[m++] = p[i];
    }
    if (n > 1)
        m--;
    return m;
}
```

## 6.8 极角排序

```
// 叉积: 对于 tmp = a x b
// 如果b在a的逆时针(左边):tmp > 0
// 顺时针(右边): tmp < 0
// 同向: tmp = 0
// 相对于原点的极角排序
// 如果是相对于某一点x,只需要把x当作原点即可
bool mycmp(Point a, Point b)
{
    if (atan2(a.y, a.x) != atan2(b.y, b.x))
        return atan2(a.y, a.x) < atan2(b.y, b.x);
    else
        return a.x < b.x;
}
```

## 6.9 点和多边形的位置关系

```
// 点形成一个凸包，而且按逆时针排序
// 如果是顺时针把里面的<0改为>0
// 点的编号:0~n-1
// 返回值:
// -1:点在凸多边形外
// 0:点在凸多边形边界上
// 1:点在凸多边形内
int inConvexPoly(Point a, Point p[], int n)
{
    for (int i = 0; i < n; i++)
    {
        if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
            return -1;
        else if (OnSeg(a, Line(p[i], p[(i + 1) % n])))
            return 0;
    }
    return 1;
}

//判断点是否在凸包内
bool inConvex(Point A, Point *p, int tot)
{
    int l = 1, r = tot - 2, mid;
    while (l <= r)
    {
        mid = (l + r) >> 1;
        double a1 = (p[mid] - p[0]) ^ (A - p[0]);
        double a2 = (p[mid + 1] - p[0]) ^ (A - p[0]);
        if (a1 >= 0 && a2 <= 0)
        {
            if (((p[mid + 1] - p[mid]) ^ (A - p[mid])) >= 0)
                return true;
            return false;
        }
        else if (a1 < 0)
            r = mid - 1;
        else
            l = mid + 1;
    }
    return false;
}

// 判断点在任意多边形内
// 射线法，poly[]的顶点数要大于等于 3,点的编号 0~n-1
// 返回值
// -1:点在凸多边形外
// 0:点在凸多边形边界上
// 1:点在凸多边形内
int inPoly(Point p, Point poly[], int n)
{
    int cnt;
    Line ray, side;
    cnt = 0;
    ray.s = p;
    ray.e.y = p.y;
    ray.e.x = -100000000000.0; //-INF,注意取值防止越界

    for (int i = 0; i < n; i++)
    {
        side.s = poly[i];
        side.e = poly[(i + 1) % n];

        if (OnSeg(p, side))
            return 0;

        //如果平行轴则不考虑
        if (sgn(side.s.y - side.e.y) == 0)
            continue;

        if (OnSeg(side.s, ray))
        {
            if (sgn(side.s.y - side.e.y) > 0)
                cnt++;
        }
        else if (OnSeg(side.e, ray))
        {
            if (sgn(side.e.y - side.s.y) > 0)
                cnt++;
        }
        else if (inter(ray, side))
            cnt++;
    }
    if (cnt % 2 == 1)
        return 1;
    else
        return -1;
}

// 判断凸多边形
bool isconvex(Point poly[], int n)
{
    bool s[3];
    memset(s, false, sizeof(s));
    for (int i = 0; i < n; i++)
    {
        s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] -
poly[i])) + 1] = true;
        if (s[0] && s[2])
            return false;
    }
    return true;
}

//判断凸包是否相离 bool isConvexHullSeparate(int n, int m, Point a[], Point
b[])
{
    for (int i = 0; i < n; i++)
        if (inPoly(a[i], b, m) != -1)
            return false;

    for (int i = 0; i < m; i++)
        if (inPoly(b[i], a, n) != -1)
            return false;

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {
            Line l1 = Line(a[i], a[(i + 1) % n]);
            Line l2 = Line(b[j], b[(j + 1) % m]);
            if (inter(l1, l2))
                return false;
        }
    }
    return true;
}
```

### 6.10 闵可夫斯基和

```
const int MAX = 2e5 + 5;
Point s1[MAX], s2[MAX];

int Minkowski(Point A[], int n, Point B[], int m, Point M[])
{
    int tot = 0;
    for (int i = 0; i < n; i++)
        s1[i] = A[(i + 1) % n] - A[i];

    for (int i = 0; i < m; i++)
        s2[i] = B[(i + 1) % m] - B[i];

    M[tot] = A[0] + B[0];
    int p1 = 0, p2 = 0;

    while (p1 < n && p2 < m)
        ++tot, M[tot] = M[tot - 1] + ((s1[p1] ^ s2[p2]) >= 0 ? s1[p1++] :
s2[p2++]);
    while (p1 < n)
        ++tot, M[tot] = M[tot - 1] + s1[p1++];
    while (p2 < m)
        ++tot, M[tot] = M[tot - 1] + s2[p2++];
    return tot + 1;
}
```

# 7 图论 Graph Theory

### 7.1 拓扑排序

```
int n, m;
vector<int> G[MAXN];
int in[MAXN]; // 存储每个结点的入度

bool toposort()
{
    vector<int> L;
    queue<int> S;
    for (int i = 1; i <= n; i++)
        if (in[i] == 0)
            S.push(i);
    while (!S.empty())
    {
        int u = S.front();
        S.pop();
        L.push_back(u);
        for (auto v : G[u])
        {
            if (--in[v] == 0)
            {
                S.push(v);
            }
        }
    }
    if (L.size() == n)
    {
        for (auto i : L)
            cout << i << ' ';
        return true;
    }
    else
    {
        return false;
    }
}
```

## 7.2 最短路

### 7.2.1 Bellman-Ford 算法

```
struct edge
{ // 可求有负权的图
    int v, w;
};

vector<edge> e[maxn];
int dis[maxn];
const int inf = 0x3f3f3f3f;

bool bellmanford(int n, int s)
{
    memset(dis, 63, sizeof(dis));
    dis[s] = 0;
    bool flag; // 判断一轮循环过程中是否发生松弛操作
    for (int i = 1; i <= n; i++)
    {
        flag = false;
        for (int u = 1; u <= n; u++)
        {
            if (dis[u] == inf)
                continue;
            // 无穷大与常数加减仍然为无穷大
            // 因此最短路长度为 inf 的点引出的边不可能发生松弛操作
            for (auto ed : e[u])
            {
                int v = ed.v, w = ed.w;
                if (dis[v] > dis[u] + w)
                {
                    dis[v] = dis[u] + w;
                    flag = true;
                }
            }
        }
        // 没有可以松弛的边时就停止算法
        if (!flag)
            break;
    }
    // 第 n 轮循环仍然可以松弛时说明 s 点可以抵达一个负环
    return flag;
}
```

### 7.2.2 Dijkstra 算法

```
struct edge
{
    int v, w;
};

struct node
{
    int dis, u;

    bool operator>(const node &a) const { return dis > a.dis; }
};

vector<edge> e[maxn];
int dis[maxn], vis[maxn];
priority_queue<node, vector<node>, greater<node>> q;

void dijkstra(int n, int s)
{
    memset(dis, 63, sizeof(dis));
    dis[s] = 0;
    q.push({0, s});
    while (!q.empty())
    {
        int u = q.top().u;
        q.pop();
        if (vis[u])
            continue;
        vis[u] = 1;
        for (auto ed : e[u])
        {
            int v = ed.v, w = ed.w;
            if (dis[v] > dis[u] + w)
            {
                dis[v] = dis[u] + w;
                q.push({dis[v], v});
            }
        }
    }
}
```

## 7.3 强连通分量

```
int dfn[N], low[N], dfncnt, s[N], in_stack[N], tp;
int scc[N], sc; // 结点 i 所在 SCC 的编号
int sz[N];      // 强连通 i 的大小

void tarjan(int u)
{
    low[u] = dfn[u] = ++dfncnt, s[++tp] = u, in_stack[u] = 1;
    for (int i = h[u]; i; i = e[i].nex)
    {
        const int &v = e[i].t;
        if (!dfn[v])
        {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        }
        else if (in_stack[v])
        {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u])
    {
        ++sc;
        while (s[tp] != u)
        {
            scc[s[tp]] = sc;
            sz[sc]++;
            in_stack[s[tp]] = 0;
            --tp;
        }
        scc[s[tp]] = sc;
        sz[sc]++;
        in_stack[s[tp]] = 0;
        --tp;
    }
}
```

## 7.4 割点和桥

### 7.4.1 割点

```
#include <bits/stdc++.h>
using namespace std;
int n, m; // n： 点数 m： 边数
int dfn[100001], low[100001], inde, res;
// dfn： 记录每个点的时间戳
// low： 能不经过父亲到达最小的编号，inde： 时间戳，res： 答案数量
bool vis[100001], flag[100001]; // flag： 答案 vis： 标记是否重复
vector<int> edge[100001];       // 存图用的

void Tarjan(int u, int father)
{                             // u 当前点的编号，father 自己爸爸的编号
    vis[u] = true;            // 标记
    low[u] = dfn[u] = ++inde; // 打上时间戳
    int child = 0;            // 每一个点儿子数量
    for (auto v : edge[u])
    { // 访问这个点的所有邻居 （C++11）

        if (!vis[v])
        {
            child++;                      // 多了一个儿子
            Tarjan(v, u);                 // 继续
            low[u] = min(low[u], low[v]); // 更新能到的最小节点编号
            if (father != u && low[v] >= dfn[u] &&
                !flag
                    [u]) // 主要代码
                          // 如果不是自己，且不通过父亲返回的最小点符合割点的
要求，并且没有被标记过
                         // 要求即为：删了父亲连不上去了，即为最多连到父亲
            {
                flag[u] = true;
                res++; // 记录答案
            }
        }
        else if (v != father)
            low[u] =
                min(low[u], dfn[v]); // 如果这个点不是自己，更新能到的最小节
点编号
    }
    if (father == u && child >= 2 &&
        !flag[u])
    { // 主要代码，自己的话需要 2 个儿子才可以
        flag[u] = true;
        res++; // 记录答案
    }
}

int main()
```

```
{
    cin >> n >> m; // 读入数据
    for (int i = 1; i <= m; i++)
    { // 注意点是从 1 开始的
        int x, y;
        cin >> x >> y;
        edge[x].push_back(y);
        edge[y].push_back(x);
    }                            // 使用 vector 存图
    for (int i = 1; i <= n; i++) // 因为 Tarjan 图不一定连通
        if (!vis[i])
        {
            inde = 0;     // 时间戳初始为 0
            Tarjan(i, i); // 从第 i 个点开始，父亲为自己
        }
    cout << res << endl;
    for (int i = 1; i <= n; i++)
        if (flag[i])
            cout << i << " "; // 输出结果
    return 0;
}
```

### 7.4.2 割边

```
int low[MAXN], dfn[MAXN], dfs_clock;
bool isbridge[MAXN];
vector<int> G[MAXN];
int cnt_bridge;
int father[MAXN];

void tarjan(int u, int fa)
{
    father[u] = fa;
    low[u] = dfn[u] = ++dfs_clock;
    for (int i = 0; i < G[u].size(); i++)
    {
        int v = G[u][i];
        if (!dfn[v])
        {
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            if (low[v] > dfn[u])
            {
                isbridge[v] = true;
                ++cnt_bridge;
            }
        }
        else if (dfn[v] < dfn[u] && v != fa)
        {
            low[u] = min(low[u], dfn[v]);
        }
    }
}
```

## 7.5 欧拉图

```
// 欧拉回路：通过图中每条边恰好一次的回路
// 欧拉通路：通过图中每条边恰好一次的通路
// 欧拉图：具有欧拉回路的图
// 半欧拉图：具有欧拉通路但不具有欧拉回路的图

// 给定一张有 500 个顶点的无向图,求这张图的一条欧拉路或欧拉回路。如果有多组解，输出最小的那一组。
// 在本题中，欧拉路或欧拉回路不需要经过所有顶点。
#include <algorithm>
#include <cstdio>
#include <stack>
#include <vector>
using namespace std;

struct edge {
  int to;
  bool exists;
  int revref;

  bool operator<(const edge& b) const { return to < b.to; }
};

vector<edge> beg[505];
int cnt[505];

const int dn = 500;
stack<int> ans;

void Hierholzer(int x) {  // 关键函数
  for (int& i = cnt[x]; i < (int)beg[x].size();) {
    if (beg[x][i].exists) {
      edge e = beg[x][i];
      beg[x][i].exists = 0;
      beg[e.to][e.revref].exists = 0;
      ++i;
      Hierholzer(e.to);
    } else {
      ++i;
    }
  }
  ans.push(x);
}

int deg[505];
int reftop[505];

int main() {
  for (int i = 1; i <= dn; ++i) {
    beg[i].reserve(1050);  // vector 用 reserve 避免动态分配空间，加快速度
  }
  int m;
  scanf("%d", &m);
  for (int i = 1; i <= m; ++i) {
    int a, b;
    scanf("%d%d", &a, &b);
    beg[a].push_back((edge){b, 1, 0});
    beg[b].push_back((edge){a, 1, 0});
    ++deg[a];
    ++deg[b];
  }

  for (int i = 1; i <= dn; ++i) {
    if (!beg[i].empty()) {
      sort(beg[i].begin(), beg[i].end());  // 为了要按字典序贪心，必须排序
    }
  }

  for (int i = 1; i <= dn; ++i) {
    for (int j = 0; j < (int)beg[i].size(); ++j) {
      beg[i][j].revref = reftop[beg[i][j].to]++;
    }
  }

  int bv = 0;
  for (int i = 1; i <= dn; ++i) {
    if (!deg[bv] && deg[i]) {
      bv = i;
    } else if (!(deg[bv] & 1) && (deg[i] & 1)) {
      bv = i;
    }
  }

  Hierholzer(bv);

  while (!ans.empty()) {
    printf("%d\n", ans.top());
    ans.pop();
  }
}
```

## 7.6 最小环

```
// 给出一个图，问其中的有 n 个节点构成的边权和最小的环 (n >= 3) 是多大。
int val[maxn + 1][maxn + 1]; // 原图的邻接矩阵

inline int floyd(const int &n)
{
    static int dis[maxn + 1][maxn + 1]; // 最短路矩阵
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            dis[i][j] = val[i][j]; // 初始化最短路矩阵
    int ans = inf;
    for (int k = 1; k <= n; ++k)
    {
        for (int i = 1; i < k; ++i)
            for (int j = 1; j < i; ++j)
                ans = std::min(ans, dis[i][j] + val[i][k] + val[k][j]);
// 更新答案
        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
               dis[i][j] = std::min(
                    dis[i][j], dis[i][k] + dis[k][j]); // 正常的 floyd 更
新最短路矩阵
    }
    return ans;
}
```

## 7.7 树哈希

```
// 给定一棵以点 1 为根的树，你需要输出这棵树中最多能选出多少个互不同构的子树。
// 两棵有根树 T1,T2 同构当且仅当他们的大小相等，且存在一个顶点排列 σ 使得在 T1 中
i 是 j 的祖先当且仅当在 T2 中 σi 是 σj 的祖先。
#include <cctype>
#include <chrono>
#include <cstdio>
#include <random>
#include <set>
#include <vector>

typedef unsigned long long ull;

const ull mask =
std::chrono::steady_clock::now().time_since_epoch().count();

ull shift(ull x)
{
    x ^= mask;
    x ^= x << 13;
    x ^= x >> 7;
    x ^= x << 17;
    x ^= mask;
    return x;
}

const int N = 1e6 + 10;

int n;
ull hash[N];
std::vector<int> edge[N];
std::set<ull> trees;

void getHash(int x, int p)
{
    hash[x] = 1;
    for (int i : edge[x])
    {
        if (i == p)
        {
            continue;
        }
        getHash(i, x);
        hash[x] += shift(hash[i]);
    }
    trees.insert(hash[x]);
}

int main()
{
    scanf("%d", &n);
    for (int i = 1; i < n; i++)
    {
        int u, v;
        scanf("%d%d", &u, &v);
        edge[u].push_back(v);
        edge[v].push_back(u);
    }
    getHash(1, 0);
    printf("%lu", trees.size());
}
```

## 7.8 最小树形图

```
bool solve()
{
    ans = 0;
    int u, v, root = 0;
    for (;;)
    {
        f(i, 0, n) in[i] = 1e100;
        f(i, 0, m)
        {
            u = e[i].s;
            v = e[i].t;
            if (u != v && e[i].w < in[v])
            {
                in[v] = e[i].w;
                pre[v] = u;
            }
        }
        f(i, 0, m) if (i != root && in[i] > 1e50) return 0;
        int tn = 0;
        memset(id, -1, sizeof id);
        memset(vis, -1, sizeof vis);
        in[root] = 0;
        f(i, 0, n)
        {
            ans += in[i];
            v = i;
            while (vis[v] != i && id[v] == -1 && v != root)
            {
                vis[v] = i;
                v = pre[v];
            }
            if (v != root && id[v] == -1)
            {
                for (int u = pre[v]; u != v; u = pre[u])
                    id[u] = tn;
                id[v] = tn++;
            }
        }
        if (tn == 0)
            break;
        f(i, 0, n) if (id[i] == -1) id[i] = tn++;
        f(i, 0, m)
        {
            u = e[i].s;
            v = e[i].t;
            e[i].s = id[u];
            e[i].t = id[v];
            if (e[i].s != e[i].t)
                e[i].w -= in[v];
        }
        n = tn;
        root = id[root];
    }
    return ans;
}
```

## 7.9 最小直径生成树

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 502;
typedef long long ll;
typedef pair<int, int> pii;
ll d[MAXN][MAXN], dd[MAXN][MAXN], rk[MAXN][MAXN], val[MAXN];
const ll INF = 1e17;
int n, m;

bool cmp(int a, int b) { return val[a] < val[b]; }

void floyd()
{
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
                d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
}

struct node
{
    ll u, v, w;
} a[MAXN * (MAXN - 1) / 2];
```

```
void solve()
{
    // 求图的绝对中心
    floyd();
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            rk[i][j] = j;
            val[j] = d[i][j];
        }
        sort(rk[i] + 1, rk[i] + 1 + n, cmp);
    }
    ll P = 0, ansP = INF;
    // 在点上
    for (int i = 1; i <= n; i++)
    {
        if (d[i][rk[i][n]] * 2 < ansP)
        {
            ansP = d[i][rk[i][n]] * 2;
            P = i;
        }
    }
    // 在边上
    int f1 = 0, f2 = 0;
    ll disu = INT_MIN, disv = INT_MIN, ansL = INF;
    for (int i = 1; i <= m; i++)
    {
        ll u = a[i].u, v = a[i].v, w = a[i].w;
        for (int p = n, i = n - 1; i >= 1; i--)
        {
            if (d[v][rk[u][i]] > d[v][rk[u][p]])
            {
                if (d[u][rk[u][i]] + d[v][rk[u][p]] + w < ansL)
                {
                    ansL = d[u][rk[u][i]] + d[v][rk[u][p]] + w;
                    f1 = u, f2 = v;
                    disu = (d[u][rk[u][i]] + d[v][rk[u][p]] + w) / 2 -
d[u][rk[u][i]];
                    disv = w - disu;
                }
                p = i;
            }
        }
    }
    cout << min(ansP, ansL) / 2 << '\n';
    // 最小路径生成树
    vector<pii> pp;
    for (int i = 1; i <= 501; ++i)
        for (int j = 1; j <= 501; ++j)
            dd[i][j] = INF;
    for (int i = 1; i <= 501; ++i)
        dd[i][i] = 0;
    if (ansP <= ansL)
    {
        for (int j = 1; j <= n; j++)
        {
            for (int i = 1; i <= m; ++i)
            {
                ll u = a[i].u, v = a[i].v, w = a[i].w;
                if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v])
                {
                    dd[P][v] = dd[P][u] + w;
                    pp.push_back({u, v});
                }
                u = a[i].v, v = a[i].u, w = a[i].w;
                if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v])
                {
                    dd[P][v] = dd[P][u] + w;
                    pp.push_back({u, v});
                }
            }
        }
        for (auto [x, y] : pp)
            cout << x << ' ' << y << '\n';
    }
    else
    {
        d[n + 1][f1] = disu;
        d[f1][n + 1] = disu;
        d[n + 1][f2] = disv;
        d[f2][n + 1] = disv;
        a[m + 1].u = n + 1, a[m + 1].v = f1, a[m + 1].w = disu;
        a[m + 2].u = n + 1, a[m + 2].v = f2, a[m + 2].w = disv;
        n += 1;
        m += 2;
        floyd();
        P = n;
        for (int j = 1; j <= n; j++)
        {
            for (int i = 1; i <= m; ++i)
            {
                ll u = a[i].u, v = a[i].v, w = a[i].w;
                if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v])
                {
                    dd[P][v] = dd[P][u] + w;
                    pp.push_back({u, v});
                }
                u = a[i].v, v = a[i].u, w = a[i].w;
                if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v])
                {
                    dd[P][v] = dd[P][u] + w;
                    pp.push_back({u, v});
                }
            }
        }
        cout << f1 << ' ' << f2 << '\n';
        for (auto [x, y] : pp)
            if (x != n && y != n)
                cout << x << ' ' << y << '\n';
    }
}

void init()
{
    for (int i = 1; i <= 501; ++i)
        for (int j = 1; j <= 501; ++j)
            d[i][j] = INF;
    for (int i = 1; i <= 501; ++i)
        d[i][i] = 0;
}

int main()
{
    init();
    cin >> n >> m;
    for (int i = 1; i <= m; ++i)
    {
        ll u, v, w;
        cin >> u >> v >> w;
        w *= 2;
        d[u][v] = w, d[v][u] = w;
        a[i].u = u, a[i].v = v, a[i].w = w;
    }
    solve();
    return 0;
}
```

# 8 网络流 Flow Network

## 8.1 最大流

### 8.1.1 Edmonds-Karp 算法

```
#define maxn 250
#define INF 0x3f3f3f3f

struct Edge
{
    int from, to, cap, flow;

    Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
};

struct EK
{
    int n, m;             // n: 点数, m: 边数
    vector<Edge> edges;   // edges: 所有边的集合
    vector<int> G[maxn];  // G: 点 x -> x 的所有边在 edges 中的下标
    int a[maxn], p[maxn]; // a: 点 x -> BFS 过程中最近接近点 x 的边给它的最
大流
                          // p: 点 x -> BFS 过程中最近接近点 x 的边

    void init(int n)
    {
        for (int i = 0; i < n; i++)
            G[i].clear();
        edges.clear();
    }

    void AddEdge(int from, int to, int cap)
    {
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0));
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    int Maxflow(int s, int t)
    {
        int flow = 0;
        for (;;)
        {
            memset(a, 0, sizeof(a));
            queue<int> Q;
            Q.push(s);
            a[s] = INF;
            while (!Q.empty())
            {
                int x = Q.front();
                Q.pop();
                for (int i = 0; i < G[x].size(); i++)
                { // 遍历以 x 作为起点的边
                    Edge &e = edges[G[x][i]];
                    if (!a[e.to] && e.cap > e.flow)
                    {
                        p[e.to] = G[x][i]; // G[x][i] 是最近接近点 e.to 的
边

                        a[e.to] =
                            min(a[x], e.cap - e.flow); // 最近接近点 e.to
的边赋给它的流
                        Q.push(e.to);
                    }
                }
                if (a[t])
                    break; // 如果汇点接受到了流，就退出 BFS
            }
            if (!a[t])
                 break; // 如果汇点没有接受到流，说明源点和汇点不在同一个连通
分量上
            for (int u = t; u != s;
                 u = edges[p[u]].from)
            {                                 // 通过 u 追寻 BFS 过程中 s ->
t 的路径
                edges[p[u]].flow += a[t];     // 增加路径上边的 flow 值
                edges[p[u] ^ 1].flow -= a[t]; // 减小反向路径的 flow 值
            }
            flow += a[t];
        }
        return flow;
    }
};
```

### 8.1.2 Dinic 算法

```
struct MF
{
    struct edge
    {
        int v, nxt, cap, flow;
    } e[N];

    int fir[N], cnt = 0;

    int n, S, T;
    ll maxflow = 0;
    int dep[N], cur[N];

    void init()
    {
        memset(fir, -1, sizeof fir);
        cnt = 0;
    }

    void addedge(int u, int v, int w)
    {
        e[cnt] = {v, fir[u], w, 0};
        fir[u] = cnt++;
        e[cnt] = {u, fir[v], 0, 0};
        fir[v] = cnt++;
    }

    bool bfs()
    {
        queue<int> q;
        memset(dep, 0, sizeof(int) * (n + 1));

        dep[S] = 1;
        q.push(S);
        while (q.size())
        {
            int u = q.front();
            q.pop();
            for (int i = fir[u]; ~i; i = e[i].nxt)
            {
                int v = e[i].v;
                if ((!dep[v]) && (e[i].cap > e[i].flow))
                {
                    dep[v] = dep[u] + 1;
                    q.push(v);
                }
            }
        }
        return dep[T];
    }

    int dfs(int u, int flow)
    {
        if ((u == T) || (!flow))
            return flow;

        int ret = 0;
        for (int &i = cur[u]; ~i; i = e[i].nxt)
        {
            int v = e[i].v, d;
            if ((dep[v] == dep[u] + 1) &&
                (d = dfs(v, min(flow - ret, e[i].cap - e[i].flow))))
            {
                ret += d;
                e[i].flow += d;
                e[i ^ 1].flow -= d;
                if (ret == flow)
                    return ret;
            }
        }
        return ret;
    }

    void dinic()
    {
        while (bfs())
        {
            memcpy(cur, fir, sizeof(int) * (n + 1));
            maxflow += dfs(S, INF);
        }
    }
} mf;
```

### 8.1.3 MPM 算法

```
struct MPM
{
    struct FlowEdge
    {
        int v, u;
        long long cap, flow;

        FlowEdge() {}
        FlowEdge(int _v, int _u, long long _cap, long long _flow)
            : v(_v), u(_u), cap(_cap), flow(_flow) {}

        FlowEdge(int _v, int _u, long long _cap)
            : v(_v), u(_u), cap(_cap), flow(0ll) {}
    };

    const long long flow_inf = 1e18;
    vector<FlowEdge> edges;
    vector<char> alive;
    vector<long long> pin, pout;
    vector<list<int>> in, out;
    vector<vector<int>> adj;
    vector<long long> ex;
    int n, m = 0;
    int s, t;
    vector<int> level;
    vector<int> q;
    int qh, qt;

    void resize(int _n)
    {
        n = _n;
        ex.resize(n);
        q.resize(n);
        pin.resize(n);
        pout.resize(n);
        adj.resize(n);
        level.resize(n);
        in.resize(n);
        out.resize(n);
    }

    MPM() {}

    MPM(int _n, int _s, int _t)
    {
        resize(_n);
        s = _s;
        t = _t;
    }

    void add_edge(int v, int u, long long cap)
    {
        edges.push_back(FlowEdge(v, u, cap));
        edges.push_back(FlowEdge(u, v, 0));
        adj[v].push_back(m);
        adj[u].push_back(m + 1);
        m += 2;
    }

    bool bfs()
    {
        while (qh < qt)
        {
            int v = q[qh++];
            for (int id : adj[v])
            {
                if (edges[id].cap - edges[id].flow < 1)
                    continue;
                if (level[edges[id].u] != -1)
                    continue;
                level[edges[id].u] = level[v] + 1;
                q[qt++] = edges[id].u;
            }
        }
        return level[t] != -1;
    }

    long long pot(int v) { return min(pin[v], pout[v]); }

    void remove_node(int v)
    {
        for (int i : in[v])
        {
            int u = edges[i].v;
            auto it = find(out[u].begin(), out[u].end(), i);
            out[u].erase(it);
            pout[u] -= edges[i].cap - edges[i].flow;
        }
        for (int i : out[v])
        {
            int u = edges[i].u;
            auto it = find(in[u].begin(), in[u].end(), i);
            in[u].erase(it);
            pin[u] -= edges[i].cap - edges[i].flow;
        }
    }

    void push(int from, int to, long long f, bool forw)
    {
        qh = qt = 0;
        ex.assign(n, 0);
        ex[from] = f;
        q[qt++] = from;
        while (qh < qt)
        {
            int v = q[qh++];
            if (v == to)
                break;
            long long must = ex[v];
            auto it = forw ? out[v].begin() : in[v].begin();
            while (true)
            {
                int u = forw ? edges[*it].u : edges[*it].v;
                       long  long  pushed  =  min(must,  edges[*it].cap  -
edges[*it].flow);
                if (pushed == 0)
                    break;
                if (forw)
                {
                    pout[v] -= pushed;
                    pin[u] -= pushed;
                }
                else
                {
                    pin[v] -= pushed;
                    pout[u] -= pushed;
                }
                if (ex[u] == 0)
                    q[qt++] = u;
                ex[u] += pushed;
                edges[*it].flow += pushed;
                edges[(*it) ^ 1].flow -= pushed;
                must -= pushed;
                if (edges[*it].cap - edges[*it].flow == 0)
                {
                    auto jt = it;
                    ++jt;
                    if (forw)
                    {
                            in[u].erase(find(in[u].begin(), in[u].end(), 
*it));
                        out[v].erase(it);
                    }
                    else
                    {
                         out[u].erase(find(out[u].begin(), out[u].end(), 
*it));
                        in[v].erase(it);
                    }
                    it = jt;
                }
                else
                    break;
                if (!must)
                    break;
            }
        }
    }

    long long flow()
    {
        long long ans = 0;
        while (true)
        {
            pin.assign(n, 0);
            pout.assign(n, 0);
            level.assign(n, -1);
            alive.assign(n, true);
            level[s] = 0;
            qh = 0;
            qt = 1;
            q[0] = s;
            if (!bfs())
                break;
            for (int i = 0; i < n; i++)
            {
                out[i].clear();
                in[i].clear();
            }
            for (int i = 0; i < m; i++)
            {
                if (edges[i].cap - edges[i].flow == 0)
                    continue;
                int v = edges[i].v, u = edges[i].u;
                if (level[v] + 1 == level[u] && (level[u] < level[t] || u
== t))
                {
                    in[u].push_back(i);
                    out[v].push_back(i);
                    pin[u] += edges[i].cap - edges[i].flow;
                    pout[v] += edges[i].cap - edges[i].flow;
                }
            }
            pin[s] = pout[t] = flow_inf;
            while (true)
            {
                int v = -1;
                for (int i = 0; i < n; i++)
                {
                    if (!alive[i])
                        continue;
                    if (v == -1 || pot(i) < pot(v))
                        v = i;
                }
                if (v == -1)
                    break;
                if (pot(v) == 0)
                {
                    alive[v] = false;
                    remove_node(v);
                    continue;
                }
                long long f = pot(v);
                ans += f;
                push(v, s, f, false);
                push(v, t, f, true);
                alive[v] = false;
                remove_node(v);
            }
        }
        return ans;
    }
};
```

### 8.1.4 ISAP 算法

```
struct Edge
{
    int from, to, cap, flow;

    Edge(int u, int v, int c, int f) : from(u), to(v), cap(c), flow(f) {}
```

```
};

bool operator<(const Edge &a, const Edge &b)
{
    return a.from < b.from || (a.from == b.from && a.to < b.to);
}

struct ISAP
{
    int n, m, s, t;
    vector<Edge> edges;
    vector<int> G[maxn];
    bool vis[maxn];
    int d[maxn];
    int cur[maxn];
    int p[maxn];
    int num[maxn];

    void AddEdge(int from, int to, int cap)
    {
        edges.push_back(Edge(from, to, cap, 0));
        edges.push_back(Edge(to, from, 0, 0));
        m = edges.size();
        G[from].push_back(m - 2);
        G[to].push_back(m - 1);
    }

    bool BFS()
    {
        memset(vis, 0, sizeof(vis));
        queue<int> Q;
        Q.push(t);
        vis[t] = 1;
        d[t] = 0;
        while (!Q.empty())
        {
            int x = Q.front();
            Q.pop();
            for (int i = 0; i < G[x].size(); i++)
            {
                Edge &e = edges[G[x][i] ^ 1];
                if (!vis[e.from] && e.cap > e.flow)
                {
                    vis[e.from] = 1;
                    d[e.from] = d[x] + 1;
                    Q.push(e.from);
                }
            }
        }
        return vis[s];
    }

    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)
```

```
            G[i].clear();
        edges.clear();
    }

    int Augment()
    {
        int x = t, a = INF;
        while (x != s)
        {
            Edge &e = edges[p[x]];
            a = min(a, e.cap - e.flow);
            x = edges[p[x]].from;
        }
        x = t;
        while (x != s)
        {
            edges[p[x]].flow += a;
            edges[p[x] ^ 1].flow -= a;
            x = edges[p[x]].from;
        }
        return a;
    }

    int Maxflow(int s, int t)
    {
        this->s = s;
        this->t = t;
        int flow = 0;
        BFS();
        memset(num, 0, sizeof(num));
        for (int i = 0; i < n; i++)
            num[d[i]]++;
        int x = s;
        memset(cur, 0, sizeof(cur));
        while (d[s] < n)
        {
            if (x == t)
            {
                flow += Augment();
                x = s;
            }
            int ok = 0;
            for (int i = cur[x]; i < G[x].size(); i++)
            {
                Edge &e = edges[G[x][i]];
                if (e.cap > e.flow && d[x] == d[e.to] + 1)
                {
                    ok = 1;
                    p[e.to] = G[x][i];
                    cur[x] = i;
                    x = e.to;
                    break;
                }
            }
            if (!ok)
            {
                int m = n - 1;
                for (int i = 0; i < G[x].size(); i++)
                {
                    Edge &e = edges[G[x][i]];
                    if (e.cap > e.flow)
                        m = min(m, d[e.to]);
                }
                if (--num[d[x]] == 0)
                    break;
                num[d[x] = m + 1]++;
                cur[x] = 0;
                if (x != s)
                    x = edges[p[x]].from;
            }
        }
        return flow;
    }
};
```

## 8.2 最小割

```
//最大流最小割定理
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>

const int N = 1e4 + 5, M = 2e5 + 5;
int n, m, s, t, tot = 1, lnk[N], ter[M], nxt[M], val[M], dep[N], cur[N];

void add(int u, int v, int w) {
  ter[++tot] = v, nxt[tot] = lnk[u], lnk[u] = tot, val[tot] = w;
}

void addedge(int u, int v, int w) { add(u, v, w), add(v, u, 0); }

int bfs(int s, int t) {
  memset(dep, 0, sizeof(dep));
  memcpy(cur, lnk, sizeof(lnk));
  std::queue<int> q;
  q.push(s), dep[s] = 1;
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    for (int i = lnk[u]; i; i = nxt[i]) {
      int v = ter[i];
      if (val[i] && !dep[v]) q.push(v), dep[v] = dep[u] + 1;
    }
  }
  return dep[t];
}

int dfs(int u, int t, int flow) {
  if (u == t) return flow;
  int ans = 0;
  for (int &i = cur[u]; i && ans < flow; i = nxt[i]) {
    int v = ter[i];
    if (val[i] && dep[v] == dep[u] + 1) {
      int x = dfs(v, t, std::min(val[i], flow - ans));
      if (x) val[i] -= x, val[i ^ 1] += x, ans += x;
    }
  }
  if (ans < flow) dep[u] = -1;
  return ans;
}

int dinic(int s, int t) {
  int ans = 0;
  while (bfs(s, t)) {
    int x;
    while ((x = dfs(s, t, 1 << 30))) ans += x;
  }
  return ans;
}

int main() {
  scanf("%d%d%d%d", &n, &m, &s, &t);
  while (m--) {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    addedge(u, v, w);
  }
  printf("%d\n", dinic(s, t));
  return 0;
}
```

## 8.3 费用流

### 8.3.1 基于 EK 算法

```
struct qxx
{
    int nex, t, v, c;
};

qxx e[M];
int h[N], cnt = 1;

void add_path(int f, int t, int v, int c)
{
    e[++cnt] = (qxx){h[f], t, v, c}, h[f] = cnt;
}

void add_flow(int f, int t, int v, int c)
{
    add_path(f, t, v, c);
    add_path(t, f, 0, -c);
}

int dis[N], pre[N], incf[N];
bool vis[N];

bool spfa()
{
    memset(dis, 0x3f, sizeof(dis));
    queue<int> q;
    q.push(s), dis[s] = 0, incf[s] = INF, incf[t] = 0;
    while (q.size())
    {
        int u = q.front();
        q.pop();
        vis[u] = 0;
        for (int i = h[u]; i; i = e[i].nex)
        {
            const int &v = e[i].t, &w = e[i].v, &c = e[i].c;
            if (!w || dis[v] <= dis[u] + c)
                continue;
            dis[v] = dis[u] + c, incf[v] = min(w, incf[u]), pre[v] = i;
            if (!vis[v])
                q.push(v), vis[v] = 1;
        }
    }
    return incf[t];
}

int maxflow, mincost;

void update()
{
    maxflow += incf[t];
    for (int u = t; u != s; u = e[pre[u] ^ 1].t)
    {
        e[pre[u]].v -= incf[t], e[pre[u] ^ 1].v += incf[t];
        mincost += incf[t] * e[pre[u]].c;
    }
}
// 调用: while(spfa())update();
```

### 8.3.2 基于 Dinic 算法

```
#include <algorithm>
#include <cstdio>
#include <cstring>
#include <queue>

const int N = 5e3 + 5, M = 1e5 + 5;
const int INF = 0x3f3f3f3f;
int n, m, tot = 1, lnk[N], cur[N], ter[M], nxt[M], cap[M], cost[M], dis[N],
ret;
bool vis[N];
```

```
void add(int u, int v, int w, int c)
{
    ter[++tot] = v, nxt[tot] = lnk[u], lnk[u] = tot, cap[tot] = w,
cost[tot] = c;
}

void addedge(int u, int v, int w, int c) { add(u, v, w, c), add(v, u, 0,
-c); }

bool spfa(int s, int t)
{
    memset(dis, 0x3f, sizeof(dis));
    memcpy(cur, lnk, sizeof(lnk));
    std::queue<int> q;
    q.push(s), dis[s] = 0, vis[s] = 1;
    while (!q.empty())
    {
        int u = q.front();
        q.pop(), vis[u] = 0;
        for (int i = lnk[u]; i; i = nxt[i])
        {
            int v = ter[i];
            if (cap[i] && dis[v] > dis[u] + cost[i])
            {
                dis[v] = dis[u] + cost[i];
                if (!vis[v])
                    q.push(v), vis[v] = 1;
            }
        }
    }
    return dis[t] != INF;
}

int dfs(int u, int t, int flow)
{
    if (u == t)
        return flow;
    vis[u] = 1;
    int ans = 0;
    for (int &i = cur[u]; i && ans < flow; i = nxt[i])
    {
        int v = ter[i];
        if (!vis[v] && cap[i] && dis[v] == dis[u] + cost[i])
        {
            int x = dfs(v, t, std::min(cap[i], flow - ans));
            if (x)
                ret += x * cost[i], cap[i] -= x, cap[i ^ 1] += x, ans +=
x;
        }
    }
    vis[u] = 0;
    return ans;
}

int mcmf(int s, int t)
{
    int ans = 0;
    while (spfa(s, t))
    {
        int x;
        while ((x = dfs(s, t, INF)))
            ans += x;
    }
    return ans;
}

int main()
{
    int s, t;
    scanf("%d%d%d%d", &n, &m, &s, &t);
    while (m--)
    {
        int u, v, w, c;
        scanf("%d%d%d%d", &u, &v, &w, &c);
        addedge(u, v, w, c);
    }
    int ans = mcmf(s, t);
    printf("%d %d\n", ans, ret);
    return 0;
}
```

# 9 STL 标准模板库 Standard Template Library

## 9.1 Priority_queue

```
top()   访问队首元素
push()  入队
pop()   堆顶（队首）元素出队
size()  队列元素个数
empty()是否为空
注意没有 clear()!   不提供该方法
优先队列只能通过 top()访问队首元素（优先级最高的元素）
```

## 9.2 Deque

```
push_back(x)/push_front(x)    把 x 压入后/前端
back()/front() 访问(不删除)后/前端元素
pop_back() pop_front()    删除后/前端元素
erase(iterator it)删除双端队列中的某一个元素
erase(iterator first,iterator last) 删除双端队列中[first,last) 中的元素
empty()判断 deque 是否空
size() 返回 deque 的元素数量
clear()清空 deque
```

# 10 其他 Other

## 10.1 竞赛模板

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

const int INF = 0x3f3f3f3f;
const int MOD = 1e9 + 7;
const int MX = 1e5 + 10;
const vector<pair<int, int>> DIRS = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

template <typename T>
istream& operator>>(istream& is, vector<T>& v) {
    for (auto& x : v) cin >> x;
    return is;
}

template <typename T>
ostream &operator<<(ostream& os, vector<T>& v) {
    for (auto& x : v) cout << x << ' ';
    return os;
}

void solution() {

}

signed main() {
    cin.tie(nullptr);
    ios::sync_with_stdio(false);
    int t{1};
    cin >> t;
    while (t--) solution();
    return 0;
}
```