vim配置

```
set backspace=2

syntax on

set nu

set autoindent
set cindent
set tabstop=4
set shiftwidth=4
set softtabstop=4

inoremap {<CR> {}<ESC>i<CR><ESC>O

```

基础算法

快速排序

```
#include <iostream>

using namespace std;

const int N = 100010;

int q[N];

void quick_sort(int q[], int l, int r)
{
    if (l >= r) return;

    int i = l - 1, j = r + 1, x = q[l + r >> 1];
    while (i < j)
    {
        do i ++ ; while (q[i] < x);
        do j -- ; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }

    quick_sort(q, l, j);
    quick_sort(q, j + 1, r);
}

int main()
{
    int n;
    scanf("%d", &n);
```

```
    for (int i = 0; i < n; i ++ ) scanf("%d", &q[i]);

    quick_sort(q, 0, n - 1);

    for (int i = 0; i < n; i ++ ) printf("%d ", q[i]);

    return 0;
}
```

归并排序

```
#include <iostream>

using namespace std;

const int N = 1e5 + 10;

int a[N], tmp[N];

void merge_sort(int q[], int l, int r)
{
    if (l >= r) return;

    int mid = l + r >> 1;

    merge_sort(q, l, mid), merge_sort(q, mid + 1, r);

    int k = 0, i = l, j = mid + 1;
    while (i <= mid && j <= r)
        if (q[i] <= q[j]) tmp[k ++ ] = q[i ++ ];
        else tmp[k ++ ] = q[j ++ ];
    while (i <= mid) tmp[k ++ ] = q[i ++ ];
    while (j <= r) tmp[k ++ ] = q[j ++ ];

    for (i = l, j = 0; i <= r; i ++, j ++ ) q[i] = tmp[j];
}

int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i ++ ) scanf("%d", &a[i]);

    merge_sort(a, 0, n - 1);

    for (int i = 0; i < n; i ++ ) printf("%d ", a[i]);

    return 0;
}
```

整数二分

模板一

```
while(l < r){
    int mid = (l + r) >> 1;
    if(check(mid)) r = mid;
    else l = mid + 1;
}
```

模板二

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

实数二分

```
while(l + eps < r){
    double mid = (l + r) / 2;
    if(check(mid)) r = mid;
    else l = mid;
}
```

高精度加法

```
#include<iostream>
#include<cstring>
#include<vector>

using namespace std;

vector<int> add(vector<int>& A, vector<int>& B) {
    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size() || i < B.size(); i++) {
        if (i < A.size()) 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;
}

int main() {
    string a, b;
    cin >> a >> b;
    vector<int> A, B;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
```

```
    for (int i = b.length() - 1; i >= 0; i--) B.push_back(b[i] - '0');
    vector<int> ans = add(A, B);
    for (int i = ans.size() - 1; i >= 0; i--) cout << ans[i];
    return 0;
}
```

高精度减法

```
#include<iostream>
#include<cstring>
#include<vector>

using namespace std;

bool cmp(vector<int>& A, vector<int>& B) { //判断A>=B
    if (A.size() != B.size()) return A.size() > B.size();
    for (int i = A.size() - 1; 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() {
    string a, b;
    cin >> a >> b;
    vector<int> A, B;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    for (int i = b.length() - 1; i >= 0; i--) B.push_back(b[i] - '0');
    vector<int> ans;
    if (cmp(A, B)) ans = sub(A, B);
    else { ans = sub(B, A); cout << "-"; }
    for (int i = ans.size() - 1; i >= 0; i--) cout << ans[i];
    return 0;
}
```

高精度乘法

```cpp
#include<iostream>
#include<cstring>
#include<vector>

using namespace std;

vector<int> mul(vector<int>& A, int B) {
    vector<int> C;
    for (int i = 0, t = 0; i < A.size() || t; i++) {
        t += A[i] * B;
        C.push_back(t % 10);
        t /= 10;
    }
    return C;
}

int main() {
    string a;
    vector<int> A;
    int B;
    cin >> a >> B;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
    vector<int> ans = mul(A, B);
    for (int i = ans.size() - 1; i >= 0; i--) cout << ans[i];
    return 0;
}
```

高精度除法

```cpp
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

vector<int> div(vector<int>& A, int b, int& r) { //返回余数r
    vector<int> C;
    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() {
    string a;
    int b;
    cin >> a >> b;
    vector<int> A;
    for (int i = a.length() - 1; i >= 0; i--) A.push_back(a[i] - '0');
```

```
    int r;
    vector<int> ans = div(A, b, r);
    for (int i = ans.size() - 1; i >= 0; i--) cout << ans[i];
    cout << endl << r;
    return 0;
}
```

快速幂

```
#include<iostream>

using namespace std;
typedef long long ll;

int qmi(int a, int b, int p) {
    int res = 1;
    while (b) {
        if (b & 1) res = (ll)res * a % p;
        a = (ll)a * a % p;
        b >>= 1;
    }
    return res;
}
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        int a, b, p;
        cin >> a >> b >> p;
        cout << qmi(a, b, p) << endl;;
    }
    return 0;
}
```

龟速乘

```
#include<iostream>
#include<cstdio>
#include<cstring>

using namespace std;
typedef long long ll;

ll qmul(ll a, ll k, ll b) {
    ll res = 0;
    while (k) {
        if (k & 1) res = (res + a) % b;
        a = (a + a) % b;
        k >>= 1;
    }
    return res;
}

```

```cpp
int main() {
    ll a, b, p;
    cin >> a >> b >> p;
    cout << qmul(a, b, p) << endl;
    return 0;
}
```

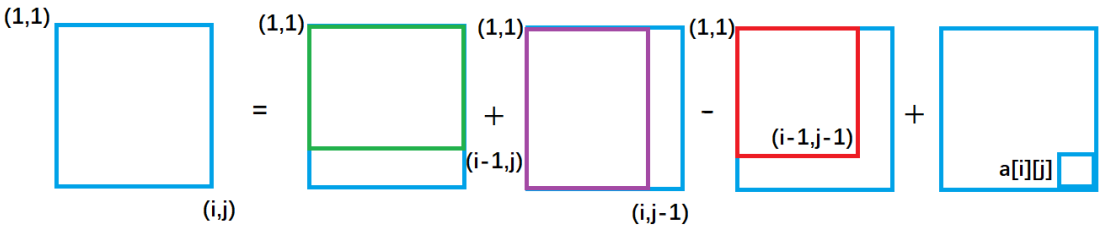
一维前缀和

在 $O(1)$ 内求出 $[l,r]$ 的和，预处理 $O(n)$

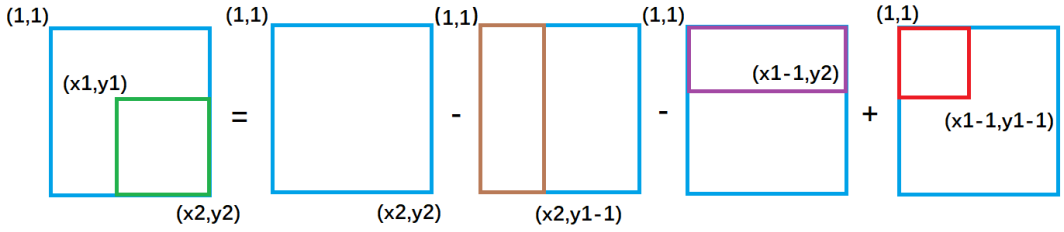
```cpp
for(int i = 1;i <= n;++i) s[i]=s[i-1]+a[i]; //预处理
int sum = s[r] - s[l-1]; //[l,r]的和
```

二位前缀和

预处理$O(nm)$，公式: $s_{i,j} = s_{i-1,j} + s_{i,j-1} + a_{i,j} - s_{i-1,j-1}$



查询$O(1)$, 公式: $sum = s_{x_2,y_2} - s_{x_1-1,y_2} - s_{x_2,y_1-1} + s_{x_1-1,y_1-1}$




```cpp
#include<cstdio>

using namespace std;

const int N = 1010;

int s[N][N], a[N][N];

int main() {
    int n, m, q;
    scanf("%d%d%d", &n, &m, &q);
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            scanf("%d", &a[i][j]);
```

```
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + a[i][j];
    while (q--) {
        int x1, x2, y1, y2;
        scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
        printf("%d\n", s[x2][y2] - s[x1 - 1][y2] - s[x2][y1 - 1] + s[x1 - 1][y1 - 1]);
    }
    return 0;
}
```

一维差分

$O(n)$预处理, $O(1)$的时间在$[l, r]$上加c

性质： 差分的前缀和是原数组

```
#include<iostream>

using namespace std;

const int N = 100010;

int a[N], b[N];

void insert(int l, int r, int c) {
    b[l] += c;
    b[r + 1] -= c;
}

int main() {
    int n, q;
    cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
        insert(i, i, a[i]);
    }
    while (q--) {
        int a, b, c;
        cin >> a >> b >> c;
        insert(a, b, c);
    }
    for (int i = 1; i <= n; i++) {
        b[i] += b[i - 1];
        cout << b[i] << " ";
    }
    return 0;
}
```

二维差分

```cpp
#include <iostream>

using namespace std;

const int N = 1010;

int n, m, q;
int a[N][N], b[N][N];

void insert(int x1, int y1, int x2, int y2, int c)
{
    b[x1][y1] += c;
    b[x2 + 1][y1] -= c;
    b[x1][y2 + 1] -= c;
    b[x2 + 1][y2 + 1] += c;
}

int main()
{
    scanf("%d%d%d", &n, &m, &q);

    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= m; j ++ )
            scanf("%d", &a[i][j]);

    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= m; j ++ )
            insert(i, j, i, j, a[i][j]);

    while (q -- )
    {
        int x1, y1, x2, y2, c;
        cin >> x1 >> y1 >> x2 >> y2 >> c;
        insert(x1, y1, x2, y2, c);
    }

    for (int i = 1; i <= n; i ++ )
        for (int j = 1; j <= m; j ++ )
            b[i][j] += b[i - 1][j] + b[i][j - 1] - b[i - 1][j - 1];

    for (int i = 1; i <= n; i ++ )
    {
        for (int j = 1; j <= m; j ++ ) printf("%d ", b[i][j]);
        puts("");
    }

    return 0;
}
```

双指针算法

1.两个指针从头开始一起走

2.一个指针从头开始，一个指针从尾开始

时间复杂度： $O(n)$ 或$O(nlogn)$

位运算

左移: 在二进制表示下把数字同时向左移动，高位越界舍弃，低位补 0

算术右移: 在二进制补码表示下把数字同时向右移动，高位以符号补充，低位越界舍弃

逻辑右移: 在二进制补码表示下把数字同时向右移动，高位以 0 补充，低位越界舍弃

与: 二进制表示下两位同时为 1 则为 1 ，否则为 0

或: 二进制表示下只要有一位为 1 则为 1 ，否则为 0

异或: 二进制表示下相应位不同则为 1 ，否则为 0

状态压缩

```
取出整数n在二进制表示下的第k位: (n >> k) & 1
取出整数n在二进制表示下的第0~k-1位(后k位): n & ((1 << k) - 1)
把整数n在二进制表示下的第k位取反: n ^ (1 << k)
对整数n在二进制表示下的第k位赋值为1: n | (1 << k)
对整数n在二进制表示下的第k位复制为0: n & (~(1 << k))
```

成对变换(场用在图的存储中):

1.当 n 为偶数时， n ^ i 等于 $n+1$

2.当 n 为奇数时， n ^ i 等于 $n-1$

lowbit运算

$lowbit(n)$ 定义为非负整数 n 在二进制下最后一个 1 后 0 的个数

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

离散化

排序+去重

```
void discreate(){
    sort(a + 1, a + n + 1);
    for(int i = 1; i <= n; ++i)
        if(i == 1 || a[i] != a[i-1])
            b[++m]=a[i];
}
```

```
vector<int> a;
void discreate(){
    sort(a.begin(),a.end());
    a.erase(unique(a.begin(), a.end()), a.end());
}
```

通过二分查找元素

```
int query(int x){
    return lower_bound(b + 1, b + m + 1, x) - b;
}
```

```
int query(int x){
    return lower_bound(a.begin(), a.end(), x) - a.begin();
}
```

ST表

解决RMQ问题，ST算法在 $O(nlogn)$ 预处理后可以 $O(1)$ 查询

递推公式: $f[i,j]=max(f[i,j-1],f[i+2^{j-1},j-1])$

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<algorithm>

using namespace std;

const int N = 1e6 + 10;

int n, m, t;
int f[N][21];
int a[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
}

void ST_init() {
    for (int i = 1; i <= n; ++i) f[i][0] = a[i];
    int t = log(n) / log(2) + 1;
```

```
    for (int j = 1; j < t; ++j)
        for (int i = 1; i <= n - (1 << j) + 1; ++i)
            f[i][j] = max(f[i][j - 1], f[i + (1 << (j - 1))][j - 1]);
}

int query(int l, int r) {
    int k = log(r - l + 1) / log(2);
    return max(f[l][k], f[r - (1 << k) + 1][k]);
}

int main() {
    read(n), read(m);
    for (int i = 1; i <= n; ++i) scanf("%d", &a[i]);

    ST_init();

    while (m--) {
        int l, r;
        read(l), read(r);
        printf("%d\n", query(l, r));
    }
}
```

区间合并

把一堆有交集的区间合并

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

vector<pair<int, int>> segs;

void merge(vector<pair<int, int>>& segs) {
    vector<pair<int, int>> res;
    sort(segs.begin(), segs.end());
    int start = -2e9, end = -2e9;
    for (int i = 0; i < segs.size(); i++) {
        if (end < segs[i].first) {
            if (start != -2e9) res.push_back({ start,end });
            start = segs[i].first, end = segs[i].second;
        }
        else end = max(end, segs[i].second);
    }
    if (start != -2e9) res.push_back({ start,end });
    segs = res;
}

int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        int l, r;
```

```
        cin >> l >> r;
        segs.push_back({ l,r });
    }
    merge(segs);
    cout << segs.size() << endl; // 合并后的区间数
    return 0;
}
```

启发式合并

把"权重小"的合并到"权重大"的上面，从而达到减少操作次数的目的

平均时间复杂度: $O(nlogn)$

并查集启发式合并

```
// HNOI 梦幻布丁
// 题意：n个布丁，m个颜色
// op1：要把一个颜色的布丁全部变为另一种颜色
// op2：询问连续的颜色段个数
// 表示方法：用数组表示所有颜色，用一个单链表表示颜色编号，让他们来合并布丁
// 启发示合并：元素少的合并到元素多的集合
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 100010, M = 1000010;

int n, m;
int h[M], to[N], ne[N], tot;
int color[N], sz[M], p[M];
int ans;

inline void add(int u, int v) {
    to[tot] = v, ne[tot] = h[u], h[u] = tot++;
    sz[u]++;
}

void merge(int& x, int& y) {
    if (x == y) return;
    if (sz[x] > sz[y]) swap(x, y);
    for (int i = h[x]; ~i; i = ne[i]) {
        int j = to[i];
        ans -= (color[j - 1] == y) + (color[j + 1] == y);
    }
    for (int i = h[x]; ~i; i = ne[i]) {
        int j = to[i];
        color[j] = y;
        if (ne[i] == -1) {
            ne[i] = h[y], h[y] = h[x];
            break;
```

```
        }
    }
    h[x] = -1;
    sz[y] += sz[x], sz[x] = 0;
}

int main() {
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &color[i]);
        if (color[i] != color[i - 1]) ans++;
        add(color[i], i);
    }

    for (int i = 0; i < M; ++i) p[i] = i;

    while (m--) {
        int op;
        scanf("%d", &op);
        if (op == 2) printf("%d\n", ans);
        else {
            int x, y;
            scanf("%d%d", &x, &y);
            merge(p[x], p[y]);
        }
    }
    return 0;
}
```

dsu on tree(树上并查集)

```
// CF600E Lomsat gelral
// 给定一棵由 n 个节点组成的树
// 树的节点编号为 1~n，其中 1 号节点为树的根节点
// 每个节点上都标有某种颜色,第 i 个节点上的颜色为 c[i]
// 如果在以节点 v 为根节点的子树中，没有任何颜色的出现次数超过颜色 c 的出现次数，那么我们称颜色 c 为该子树的主要颜色。一些子树的主要颜色可能不止一种。
// 以节点 v 为根节点的子树由节点 v 以及到达根节点的路径中包含节点 v 的其它所有节点共同组成
// 对于每个节点 v,请你求出以该节点为根节点的子树的所有主要颜色之和
// 启发式合并：重儿子合并到轻儿子
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;
typedef long long ll;

const int N = 100010, M = N << 1;

int n;
int h[N], to[M], ne[M], tot;
int color[N], cnt[N], sz[N], son[N];
ll ans[N], sum;
```

```cpp
int mx;

inline void add(int u, int v) {
    to[tot] = v, ne[tot] = h[u], h[u] = tot++;
}

int dfs_son(int u, int father) {
    sz[u] = 1;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == father) continue;
        sz[u] += dfs_son(j, u);
        if (sz[j] > sz[son[u]]) son[u] = j;
    }
    return sz[u];
}

void update(int u, int father, int sign, int pson) {
    int c = color[u];
    cnt[c] += sign;
    if (cnt[c] > mx) mx = cnt[c], sum = c;
    else if (cnt[c] == mx) sum += c;

    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == father || j == pson) continue;
        update(j, u, sign, pson);
    }
}

void dfs(int u, int father, int op) {
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == father || j == son[u]) continue;
        dfs(j, u, 0);
    }

    if (son[u]) dfs(son[u], u, 1);
    update(u, father, 1, son[u]);

    ans[u] = sum;

    if (!op) update(u, father, -1, 0), sum = mx = 0;
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) scanf("%d", &color[i]);
    for (int i = 0; i < n - 1; ++i) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }

    dfs_son(1, -1);
    dfs(1, -1, 1);

```

```
    for (int i = 1; i <= n; ++i) printf("%lld ", ans[i]);
    return 0;
}

```

最小表示法

1.初始化 $i=1, j=2$

2.通过直接向后扫描的方法，比较 b_i 和 b_j 两个循环同构串

(1) 如果扫描了 n 个字符后仍然相等，说明 S 只又一种字符串构成，任意 B_i 都是它的最小表示

(2) 如果 $i+k$ 与 $j+k$ 处发现不相等：

若 $SS[i+k] > SS[j+k]$，令 $i=i+k+1$。若此时 $i=j$，再令 $i=i+1$

若 $SS[i+k] < SS[j+k]$，令 $j=+k+1$。若此时 $i=j$，再令 $j=j+1$

3.若 $i>n$， B_j 为最小表示；若 $j>n$，$B[i]$ 为最小表示； 否则重复第二步

```cpp
// 给两个长度相等由0~9构成的串
// 问这两串能否表示为同一串
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 2000010;

int n;
char a[N], b[N];

int get_min(char s[]) {
    int i = 0, j = 1;
    while (i < n && j < n) {
        int k = 0;
        while (k < n && s[i + k] == s[j + k]) k++;
        if (k == n) break;
        if (s[i + k] > s[j + k]) i += k + 1;
        else j += k + 1;
        if (i == j) j++;
    }
    int k = min(i, j);
    s[k + n] = 0;
    return k;
}

int main() {
    scanf("%s%s", a, b);
    n = strlen(a);
    memcpy(a + n, a, n);
    memcpy(b + n, b, n);

```

```
    int x = get_min(a), y = get_min(b);
    if (strcmp(a + x, b + y)) puts("No");
    else {
        puts("Yes");
        puts(a + x); //输出字典序最小表示
    }
    return 0;
}
```

贪心

常用证明方法: 微扰，范围缩放，决策包容性，反证法，数学归纳法

通常贪心会用到排序或者堆等数据结构维护某些性质

打表

当数据范围较小或数据有规律时，可以通过搜索求出所有答案，然后放到一个数组中，直接输出

数据结构

单链表

单链表支持以下操作:

> 1.向链表头插入一个数
>
> 2.删除第 k 个插入的数后面的数
>
> 3.在第 k 个插入的数后面插入一个数

```
#include<iostream>

using namespace std;

const int N = 100010;

int ne[N], to[N], head, tot;

void init() { head = -1; }

void add_to_head(int x) {
    to[tot] = x, ne[tot] = head, head = tot++;
}

void add(int k, int x) {
    to[tot] = x, ne[tot] = ne[k], ne[k] = tot++;
}

```

```cpp
void del(int k) {
    ne[k] = ne[ne[k]];
}

int main() {
    int m;
    cin >> m;
    init();
    for (int i = 0; i < m; i++) {
        char g;
        cin >> g;
        if (g == 'H') {
            int x;
            cin >> x;
            add_to_head(x);
        }
        else if (g == 'D') {
            int k;
            cin >> k;
            if (!k) head = ne[head];
            else del(k - 1);
        }
        else if (g == 'I') {
            int k, x;
            cin >> k >> x;
            add(k - 1, x);
        }
    }
    for (int i = head; i != -1; i = ne[i]) cout << to[i] << " ";
    return 0;
}
```

双链表

双链表支持以下操作:

> 1.在最左侧插入一个数
>
> 2.在最右侧插入一个数
>
> 3.将第 k 个插入的数删除
>
> 4.在第 k 个插入的数的左侧插入一个数
>
> 5.在第 k 个插入的数的右侧插入一个数

```cpp
#include <iostream>

using namespace std;

const int N = 100010;

int m;
int e[N], l[N], r[N], idx;

```

```
void insert(int a, int x)
{
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx++;
}

void remove(int a)
{
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}

int main()
{
    cin >> m;
    r[0] = 1, l[1] = 0;
    idx = 2;
    while (m--)
    {
        string op;
        cin >> op;
        int k, x;
        if (op == "L")
        {
            cin >> x;
            insert(0, x);
        }
        else if (op == "R")
        {
            cin >> x;
            insert(l[1], x);
        }
        else if (op == "D")
        {
            cin >> k;
            remove(k + 1);
        }
        else if (op == "IL")
        {
            cin >> k >> x;
            insert(l[k + 1], x);
        }
        else
        {
            cin >> k >> x;
            insert(k + 1, x);
        }
    }
    for (int i = r[0]; i != 1; i = r[i]) cout << e[i] << ' ';
    return 0;
}

```

栈

这是一个**后进先出**(LIFO)的数据结构

```
#include<iostream>

using namespace std;

const int N = 100010;

int stack[N], top;

int main() {
    int m;
    cin >> m;
    while (m--) {
        string g;
        cin >> g;
        if (g == "push") {
            int x;
            cin >> x;
            stack[++top] = x;
        }
        else if (g == "pop") --top;
        else if (g == "empty") cout << (top ? "NO" : "YES") << endl;
        else if (g == "query") cout << stack[top] << endl;
    }
    return 0;
}
```

队列

这个一个**先进先出**(FIFO)的数据结构

```
#include<iostream>

using namespace std;

const int N = 100010;

int q[N], head, tail = -1;

int main() {
    int m;
    cin >> m;
    while (m--) {
        string g;
        cin >> g;
        if (g == "push") {
            int x;
            cin >> x;
            q[++tail] = x;
        }
        else if (g == "pop") ++head;
        else if (g == "empty") cout << (head <= tail ? "NO" : "YES") << endl;
```

```
        else if (g == "query") cout << q[head] << endl;
    }
    return 0;
}
```

单调栈

利用栈这种数据结构的性质来维护某种单调性

```
// 给定一个长度为 N 的整数数列，输出每个数左边第一个比它小的数，如果不存在则输出 -1
#include<iostream>

using namespace std;

const int N = 100010;

int stack[N], top;

int main() {
    int n, x;
    cin >> n;
    while (n--) {
        cin >> x;
        while (top && stack[top] >= x) --top;
        if (!top) cout << -1 << " ";
        else cout << stack[top] << " ";
        stack[++top] = x;
    }
    return 0;
}
```

单调队列(滑动窗口)

确定每个窗口中的 最大值 和 最小值

用队列的性质来维护一个单调队列，让对列中的数据严格单调，那么对头就是最值

```
#include<iostream>

using namespace std;

const int N = 1000010;

int q[N], a[N], head, tail = -1;

bool not_empty() {
    if (head <= tail) return true;
    else return false;
}

int main() {
    int n, k; // k为窗口大小
    cin >> n >> k;
```

```
    for (int i = 0; i < n; i++) cin >> a[i];
    // 最小值
    for (int i = 0; i < n; i++) {
        if (head <= tail && i - k + 1 > q[head]) head++;
        while (head <= tail && a[q[tail]] >= a[i]) tail--;
        q[++tail] = i;
        if (i - k + 1 >= 0) cout << a[q[head]] << " ";
    }
    cout << endl;
    // 最大值
    head = 0, tail = -1;
    for (int i = 0; i < n; i++) {
        if (head <= tail && i - k + 1 > q[head]) head++;
        while (head <= tail && a[q[tail]] <= a[i]) tail--;
        q[++tail] = i;
        if (i - k + 1 >= 0) cout << a[q[head]] << " ";
    }
    return 0;
}
```

并查集

一种支持集合合并与查询的数据结构

通常使用并查集来维护连通性

并查集路径压缩：在查找的时候把在路径上的每个节点都连接到根上，这样就可以减少未来查询次数

```
#include<iostream>
#include<cstring>

using namespace std;

const int N = 100010;

int fa[N];

int find(int x) {
    return fa[x] == x ? fa[x] : fa[x] = find(fa[x]);
}

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) fa[i] = i; // 初始化,各自构成一个集合
    while (m--) {
        string a;
        int b, c;
        cin >> a >> b >> c;
        if (a == "M") fa[find(b)] = fa[find(c)];
        else {
            if (find(b) == find(c)) cout << "Yes" << endl;
            else cout << "No" << endl;
        }
    }
    return 0;
```

```
}
```

堆

该数据结构支持一下操作:

> 1.插入一个数 x
>
> 2.输出当前集合中的最值(堆顶元素)
>
> 3.删除当前集合中的最值
>
> 4.删除第 k 个插入的数
>
> 5.修改第 k 个插入的数，将其变为 x

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 100010;

int heap[N], cnt, fheap[N], heapf[N];

void heap_swap(int a, int b) {
    swap(fheap[heapf[a]], fheap[heapf[b]]);
    swap(heapf[a], heapf[b]);
    swap(heap[a], heap[b]);
}

void down(int u) {
    int t = u;
    if (u * 2 <= cnt && heap[u * 2] < heap[t]) t = u * 2;
    if (u * 2 + 1 <= cnt && heap[u * 2 + 1] < heap[t]) t = u * 2 + 1;
    if (u != t) {
        heap_swap(u, t);
        down(t);
    }
}

void up(int u) {
    while (u / 2 && heap[u] < heap[u / 2]) {
        heap_swap(u, u / 2);
        u >>= 1;
    }
}

int main() {
    int n, m = 0, x, k;
    cin >> n;
    while (n--) {
        string g;
        cin >> g;
        if (g == "I") {
```

```
            cin >> x;
            cnt++, m++;
            fheap[m] = cnt, heapf[cnt] = m;
            heap[cnt] = x;
            up(cnt);
        }
        else if (g == "PM") cout << heap[1] << endl;
        else if (g == "DM") {
            heap_swap(1, cnt--);
            down(1);
        }
        else if (g == "D") {
            cin >> k;
            k = fheap[k];
            heap_swap(k, cnt--);
            up(k);
            down(k);
        }
        else if (g == "C") {
            cin >> k >> x;
            k = fheap[k];
            heap[k] = x;
            up(k);
            down(k);
        }
    }
    return 0;
}
```

哈希表

维护一个集合，支持一下操作：

> 1.插入一个数 x
>
> 2.查询数 x 是否在集合中出现过

拉链法

```
#include<iostream>
#include<cstring>

using namespace std;

const int N = 100003; // 一般使用质数

int to[N], ne[N], head[N], tot;

void insert(int x) {
    int k = (x % N + N) % N;
    to[tot] = x, ne[tot] = head[k], head[k] = tot++;
}

bool find(int x) {
    int k = (x % N + N) % N;
```

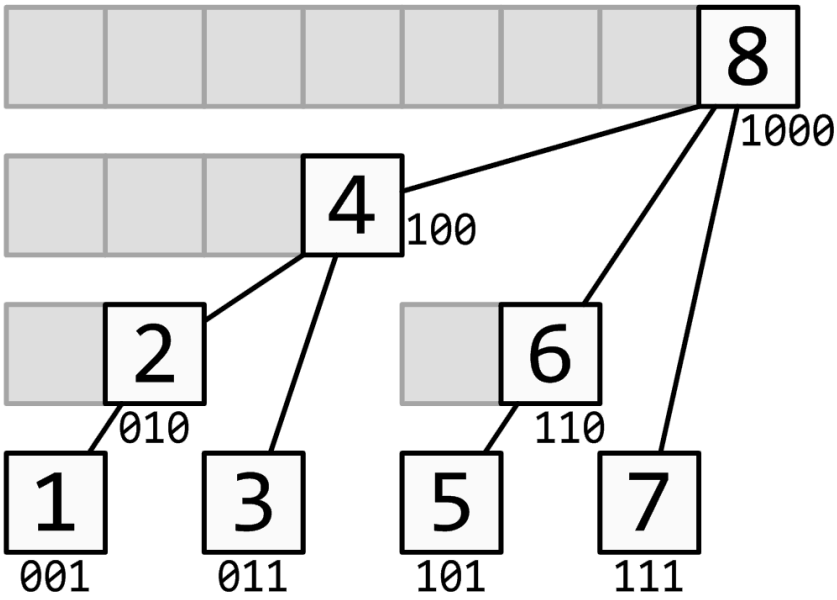
```
    for (int i = head[k]; i != -1; i = ne[i]) {
        if (to[i] == x) return true;
    }
    return false;
}

int main() {
    memset(head, -1, sizeof head);
    int n, x;
    cin >> n;
    while (n--) {
        string g;
        cin >> g;
        if (g == "I") {
            cin >> x;
            insert(x);
        }
        else {
            cin >> x;
            if (find(x)) cout << "Yes" << endl;
            else cout << "No" << endl;
        }
    }
    return 0;
}
```

开放与寻址法(推荐)

```
#include<iostream>
#include<cstring>

using namespace std;

const int N = 200003, null = 0x3f3f3f3f;

int h[N];

int find(int x) {
    int t = (x % N + N) % N;
    while (h[t] != null && h[t] != x) {
        t++;
        if (t == N) t - 0;
    }
    return t;
}

int main() {
    memset(h, 0x3f, sizeof h);
    int n, x;
    cin >> n;
    while (n--) {
        string g;
        cin >> g >> x;
        if (g == "I") h[find(x)] = x;
        else {
```

```
            if (h[find(x)] == null) cout << "No" << endl;
            else cout << "Yes" << endl;
        }
    }
    return 0;
}
```

树状数组



给定一个数列，树状数组支持以下操作：

> 1.区间查询
>
> 2.单点修改
>
> 3.区间修改

```
// 区间修改+区间求和
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 100010;

int n, m;
int a[N];
long long tr1[N], tr2[N];

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

void add(long long tr[], int x, long long c) {
    for (; x <= n; x += lowbit(x)) tr[x] += c;
}

long long sum(long long tr[], int x) {
    long long res = 0;
```

```
    for (; x; x -= lowbit(x)) res += tr[x];
    return res;
}

long long ask(int x) {
    return sum(tr1, x) * (x + 1) - sum(tr2, x);
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) {
        int b = a[i] - a[i - 1];
        add(tr1, i, b);
        add(tr2, i, (long long)b * i);
    }
    while (m--) {
        char op[2];
        int l, r, d;
        scanf("%s%d%d", op, &l, &r);
        if (*op == 'C') {
            scanf("%d", &d);
            add(tr1, l, d), add(tr2, l, l * d);
            add(tr1, r + 1, -d), add(tr2, r + 1, (r + 1) * -d);
        }
        else {
            printf("%lld\n", ask(r) - ask(l - 1));
        }
    }
}
```

线段树

把数列分成很多段，每一段是一个叶节点，线段树要开 $4N$ 的空间

给定一个数列，线段树可以支持以下操作：

> 1.区间查询
>
> 2.区间修改
>
> 3.单点修改

```
// 区间修改+区间求和
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 100010;

int n, m;
int w[N];

struct SegTree {
```

```
    int l, r;
    long long sum, add;
}tr[N << 2];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

void pushdown(int u) { // 懒标记
    SegTree& root = tr[u], & left = tr[u << 1], & right = tr[u << 1 | 1];
    if (root.add) {
        left.add += root.add, left.sum += (long long)(left.r - left.l + 1) *
root.add;
        right.add += root.add, right.sum += (long long)(right.r - right.l + 1) *
root.add;
        root.add = 0;
    }
}

void pushup(int u) { // 向上更新信息
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

void build(int u, int l, int r) {
    if (l == r) tr[u] = { l,r,w[r],0 };
    else {
        tr[u] = { l,r };
        int mid = (l + r) >> 1;
        build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
        pushup(u);
    }
}

void modify(int u, int l, int r, int d) {
    if (tr[u].l >= l && tr[u].r <= r) {
        tr[u].sum += (long long)(tr[u].r - tr[u].l + 1) * d;
        tr[u].add += d;
    }
    else {
        pushdown(u);
        int mid = (tr[u].l + tr[u].r) >> 1;
        if (l <= mid) modify(u << 1, l, r, d);
        if (r > mid) modify(u << 1 | 1, l, r, d);
        pushup(u);
    }
}

long long query(int u, int l, int r) {
    if (tr[u].l >= l && tr[u].r <= r) return tr[u].sum;
    pushdown(u); // 先向下更新信息
    int mid = (tr[u].l + tr[u].r) >> 1;
    long long sum = 0;
    if (l <= mid) sum = query(u << 1, l, r);
    if (r > mid) sum += query(u << 1 | 1, l, r);
    return sum;
```

```
}

int main() {
    read(n), read(m);
    for (int i = 1; i <= n; i++) read(w[i]);
    build(1, 1, n);
    char op[2];
    int l, r, d;
    while (m--) {
        scanf("%s", op);
        read(l), read(r);
        if (*op == 'C') {
            read(d);
            modify(1, l, r, d);
        }
        else {
            printf("%lld\n", query(1, l, r));
        }
    }
    return 0;
}
```

可持久化数据结构

部分可持久化: 所有版本都可以查询，但是只有最新版本可以修改

完全可持久化: 所有版本都可以查询和修改

可持久化线段树(主席树)

```
// 查询A[l]~A[r]中的第 k 小数
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<vector>

using namespace std;
const int N = 100010;

int n, m;
int a[N];
int root[N], tot;
vector<int> nums;

struct SegTree {
    int l, r;
    int cnt;
}tr[N * 4 + N * 17];

int find(int x) { // 整体二分
    return lower_bound(nums.begin(), nums.end(), x) - nums.begin();
}

int build(int l, int r) {
```

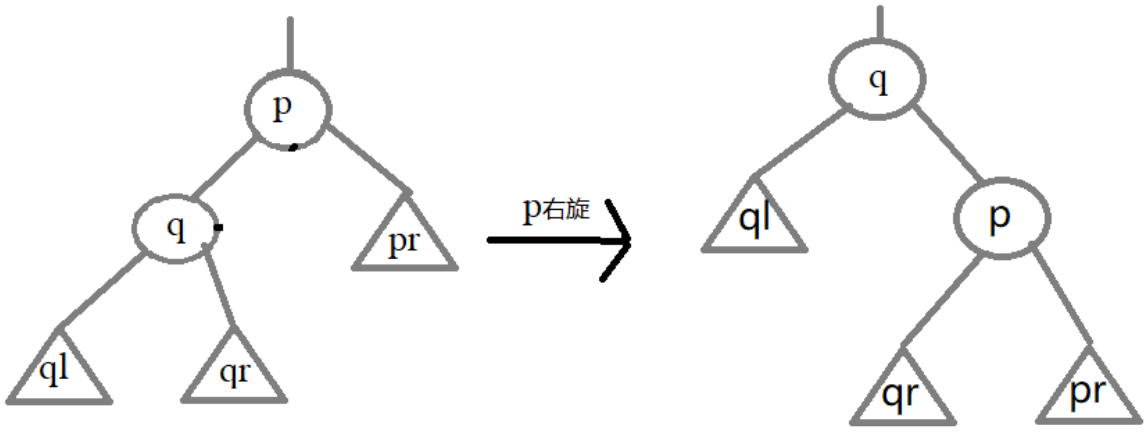
```
    int p = ++tot;
    if (l == r) return p;
    int mid = (l + r) >> 1;
    tr[p].l = build(l, mid), tr[p].r = build(mid + 1, r);
    return p;
}

int insert(int p, int l, int r, int x) {
    int q = ++tot;
    tr[q] = tr[p];
    if (l == r) {
        tr[q].cnt++;
        return q;
    }
    int mid = (l + r) >> 1;
    if (x <= mid) tr[q].l = insert(tr[p].l, l, mid, x);
    else tr[q].r = insert(tr[p].r, mid + 1, r, x);
    tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt;
    return q;
}

int query(int q, int p, int l, int r, int k) {
    if (l == r) return r;
    int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;
    int mid = (l + r) >> 1;
    if (k <= cnt) return query(tr[q].l, tr[p].l, l, mid, k);
    else return query(tr[q].r, tr[p].r, mid + 1, r, k - cnt);

}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) {
        scanf("%d", &a[i]);
        nums.push_back(a[i]);
    }

    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());

    root[0] = build(0, nums.size() - 1);

    for (int i = 1; i <= n; ++i)
        root[i] = insert(root[i - 1], 0, nums.size() - 1, find(a[i]));

    while (m--) {
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", nums[query(root[r], root[l - 1], 0, nums.size() - 1, k)]);
    }
    return 0;
}
```

平衡树

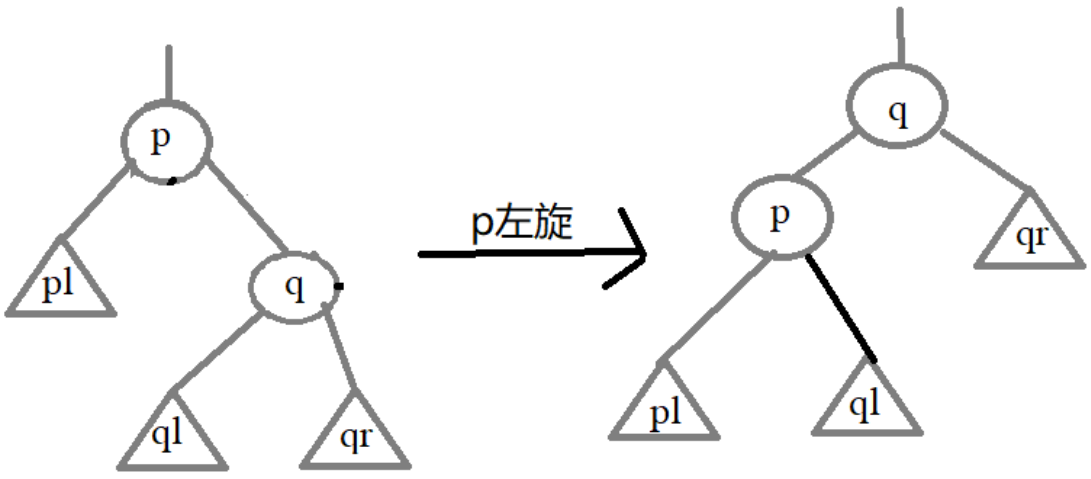
该数据结构可以实现以下操作:

> 1.插入数值 x
>
> 2.删除数值 x (若有多个相同的数，应只删除一个)
>
> 3.查询数值 x 的排名(若有多个相同的数，应输出最小的排名)
>
> 4.查询排名为 x 的数值
>
> 5.求数值 x 的前驱(前驱定义为小于 x 的最大的数)
>
> 6.求数值 x 的后继(后继定义为大于 x 的最小的数)

右旋(zig): 节点 p 右旋时，携带自己的右子树，向右旋转到 q 的右子树， q 的右子树游离出来，此时 p 右旋后左子树空出来，然后将游离的 q 的右子树放在 p 的左子树位置，旋转后的根节点为 q .



左旋(zag): 节点 p 旋转时，携带自己的左子树，向左旋转到 q 的左子树位置，q 的左子树游离出来，此时 p 左旋后右子树空出来，然后将游离的 q 的左子树放在 p 的右子树位置，旋转后的根为 q .




```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstdlib>

using namespace std;

const int N = 100010, INF = 1e8;
```

```cpp

int n;
int root, idx;

struct Node {
    int l, r; //左右子节点在数组中的下标
    int key, val; //节点关键码,权值
    int cnt, size; //副本数,子树大小
}tr[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

int get_node(int key) {
    tr[++idx].key = key;
    tr[idx].val = rand();
    tr[idx].cnt = tr[idx].size = 1;
    return idx;
}

void pushup(int p) {
    tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + tr[p].cnt;
}

void zig(int& p) { //右旋
    int q = tr[p].l;
    tr[p].l = tr[q].r;
    tr[q].r = p;
    p = q;
    pushup(tr[p].r), pushup(p);
}

void zag(int& p) { //左旋
    int q = tr[p].r;
    tr[p].r = tr[q].l;
    tr[q].l = p;
    p = q;
    pushup(tr[p].l), pushup(p);
}

void build() {
    get_node(-INF), get_node(INF);
    root = 1, tr[1].r = 2;
    pushup(root);
    if (tr[1].val < tr[2].val) zag(root);
}

void insert(int& p, int key) {
    if (!p) p = get_node(key);
    else if (tr[p].key == key) tr[p].cnt++;
    else if (tr[p].key > key) {
        insert(tr[p].l, key);
        if (tr[tr[p].l].val > tr[p].val) zig(p);
    }
```

```
    else {
        insert(tr[p].r, key);
        if (tr[tr[p].r].val > tr[p].val) zag(p);
    }
    pushup(p);
}

void del(int& p, int key) {
    if (!p) return;
    if (tr[p].key == key) {
        if (tr[p].cnt > 1) tr[p].cnt--;
        else if (tr[p].l || tr[p].r) { //左子树或右子树
            if (!tr[p].r || tr[tr[p].l].val > tr[tr[p].r].val) {
                zig(p);
                del(tr[p].r, key);
            }
            else {
                zag(p);
                del(tr[p].l, key);
            }
        }
        else p = 0;
    }
    else if (tr[p].key > key) del(tr[p].l, key);
    else del(tr[p].r, key);
    pushup(p);
}

int get_rank_by_key(int p, int key) { //通过数值找排名
    if (!p) return 0;
    if (tr[p].key == key) return tr[tr[p].l].size + 1;
    if (tr[p].key > key) return get_rank_by_key(tr[p].l, key);
    return tr[tr[p].l].size + tr[p].cnt + get_rank_by_key(tr[p].r, key);
}

int get_key_by_rank(int p, int rank) { //通过数值找排名
    if (!p) return INF;
    if (tr[tr[p].l].size >= rank) return get_key_by_rank(tr[p].l, rank); //在左边,
跑到左边找
    if (tr[tr[p].l].size + tr[p].cnt >= rank) return tr[p].key;
    return get_key_by_rank(tr[p].r, rank - tr[tr[p].l].size - tr[p].cnt);
}

int get_prev(int p, int key) { //找到严格小于key的最大值
    if (!p) return -INF;
    if (tr[p].key >= key) return get_prev(tr[p].l, key);
    return max(tr[p].key, get_prev(tr[p].r, key));
}

int get_next(int p, int key) { //找到严格大于key的最小值
    if (!p) return INF;
    if (tr[p].key <= key) return get_next(tr[p].r, key); //在右边,跑到右边找
    return min(tr[p].key, get_next(tr[p].l, key));
}

int main() {
    build();

```

```
    read(n);
    while (n--) {
        int opt, x;
        read(opt), read(x);
        if (opt == 1) insert(root, x);
        else if (opt == 2) del(root, x);
        else if (opt == 3) printf("%d\n", get_rank_by_key(root, x) - 1); //因为多了一个正无穷的节点
        else if (opt == 4) printf("%d\n", get_key_by_rank(root, x + 1)); //理由同上
        else if (opt == 5) printf("%d\n", get_prev(root, x));
        else printf("%d\n", get_next(root, x));
    }
    return 0;
}
```

splay

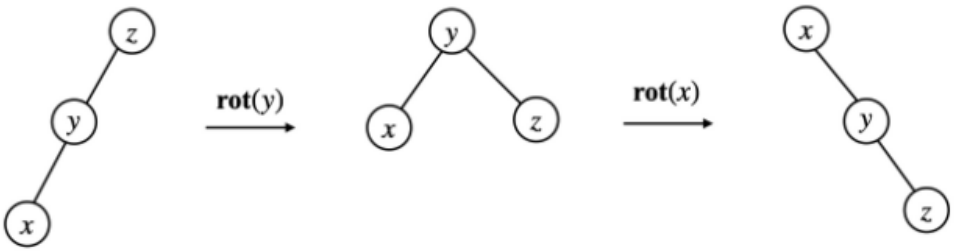
splay是一种二叉搜索树，它通过不断将某个节点旋转到根节点，使整棵树满足二叉搜索树的性质

旋转(rotate): 假设需要旋转的节点为 x，其父节点为 y，y 的父节点为 z，以左旋为例，对称过去即为右旋。如果 y 的右子树是 x，y 是 z 的右子树，那么将 z 的右子树变成 x，此时 x 的父节点是 z，y 的右子树为 x 的左子树，此时 x 的左子树的根节点为 y，然后把 x 的左子树变为 y，此时 y 的根节点是 x，然后先更新 y，再更新 x.

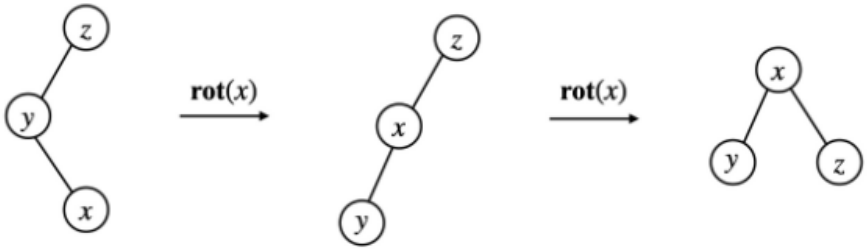
splay: 将节点 x 旋转至节点 k 下面

splay旋转需要分类讨论，如下:

- **链状**



- **锯齿状**




```
// 翻转子序列
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

```

```cpp
const int N = 100010;
int n, m;
int root, idx;

struct Tree {
    int s[2], p, v;
    int size, flag;
    void init(int _v, int _p) {
        v = _v, p = _p;
        size = 1;
    }
}tr[N];

void pushup(int u) {
    tr[u].size = tr[tr[u].s[0]].size + tr[tr[u].s[1]].size + 1;
}

void pushdown(int u) {
    if (tr[u].flag) { //需要交换
        swap(tr[u].s[0], tr[u].s[1]);
        tr[tr[u].s[0]].flag ^= 1;
        tr[tr[u].s[1]].flag ^= 1;
        tr[u].flag = 0;
    }
}

void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}

void splay(int x, int k) { //把x转到k处
    while (tr[x].p != k) {
        int y = tr[x].p, z = tr[y].p;
        if (z != k)
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        rotate(x);
    }
    if (!k) root = x;
}

int get_rank(int k) {
    int u = root;
    while (1) {
        pushdown(u);
        if (tr[tr[u].s[0]].size >= k) u = tr[u].s[0];
        else if (tr[tr[u].s[0]].size + 1 == k) return u;
        else k -= tr[tr[u].s[0]].size + 1, u = tr[u].s[1];
    }
    return -1;
}

void out(int u) {
```

```
    pushdown(u);
    if (tr[u].s[0]) out(tr[u].s[0]);
    if (tr[u].v >= 1 && tr[u].v <= n) printf("%d ", tr[u].v);
    if (tr[u].s[1]) out(tr[u].s[1]);
}

void insert(int v) {
    int u = root, p = 0;
    while (u) p = u, u = tr[u].s[v > tr[u].v];
    u = ++idx;
    if (p) tr[p].s[v > tr[p].v] = u;
    tr[u].init(v, p);
    splay(u, 0);
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n + 1; i++) insert(i);
    while (m--) {
        int l, r;
        scanf("%d%d", &l, &r);
        l = get_rank(l), r = get_rank(r + 2);
        splay(l, 0), splay(r, l);
        tr[tr[r].s[0]].flag ^= 1;
    }
    out(root);
    return 0;
}
```

树套树

树套树是一种思想，有两个维度，当题目要求实现的功能无法使用一种树形结构实现时，但可以用一种树形结构维护一部分，用另一种树形结构维护另一部分时，可以考虑使用树套树。一棵树维护一部分性质，树中的节点是另一棵树，用来维护其它性质.

线段树套平衡树

> 1.查询整数 x 在区间 $[l,r]$ 内的排名.
>
> 2.查询区间 $[l,r]$ 内排名为 k 的值.
>
> 3.将 pos 位置的数修改为 x .
>
> 4.查询整数 x 在区间 $[l,r]$ 内的前驱(前驱定义为小于 x ，且最大的数).
>
> 5.查询整数 x 在区间 $[l,r]$ 内的后继(后继定义为大于 x ，且最小的数).

```
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 1500010, INF = 0x3f3f3f3f;

int n, m;
```

```
int L[N], R[N], T[N], idx;
int w[N];

struct Tree {
    int s[2], p, v;
    int size;
    void init(int _v, int _p) {
        v = _v, p = _p;
        size = 1;
    }
}tr[N];

void pushup(int x) {
    tr[x].size = tr[tr[x].s[0]].size + tr[tr[x].s[1]].size + 1;
}

void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    tr[z].s[tr[z].s[1] == y] = x, tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}

void splay(int& root, int x, int k) {
    while (tr[x].p != k) {
        int y = tr[x].p, z = tr[y].p;
        if (z != k)
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        rotate(x);
    }
    if (!k) root = x;
}

void insert(int& root, int v) {
    int u = root, p = 0;
    while (u) p = u, u = tr[u].s[v > tr[u].v];
    u = ++idx;
    if (p) tr[p].s[v > tr[p].v] = u;
    tr[u].init(v, p);
    splay(root, u, 0);
}

int get_k(int root, int v) {
    int u = root, res = 0;
    while (u) {
        if (tr[u].v < v) res += tr[tr[u].s[0]].size + 1, u = tr[u].s[1];
        else u = tr[u].s[0];
    }
    return res;
}

void update(int& root, int x, int y) {
    int u = root;
    while (u) {
        if (tr[u].v == x) break;
```

```
        if (tr[u].v < x) u = tr[u].s[1];
        else u = tr[u].s[0];
    }
    splay(root, u, 0);
    int l = tr[u].s[0], r = tr[u].s[1];
    while (tr[l].s[1]) l = tr[l].s[1];
    while (tr[r].s[0]) r = tr[r].s[0];
    splay(root, l, 0), splay(root, r, l);
    tr[r].s[0] = 0;
    pushup(r), pushup(l);
    insert(root, y);
}

void build(int u, int l, int r) {
    L[u] = l, R[u] = r;
    insert(T[u], -INF), insert(T[u], INF);
    for (int i = l; i <= r; ++i) insert(T[u], w[i]);
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

int query(int u, int l, int r, int x) {
    if (L[u] >= l && R[u] <= r) return get_k(T[u], x) - 1;
    int mid = (L[u] + R[u]) >> 1, res = 0;
    if (l <= mid) res += query(u << 1, l, r, x);
    if (r > mid) res += query(u << 1 | 1, l, r, x);
    return res;
}

void modify(int u, int p, int x) {
    update(T[u], w[p], x);
    if (L[u] == R[u]) return;
    int mid = (L[u] + R[u]) >> 1;
    if (p <= mid) modify(u << 1, p, x);
    else modify(u << 1 | 1, p, x);
}

int get_pre(int root, int v) {
    int u = root, res = -INF;
    while (u) {
        if (tr[u].v < v) res = max(res, tr[u].v), u = tr[u].s[1];
        else u = tr[u].s[0];
    }
    return res;
}

int get_suc(int root, int v) {
    int u = root, res = INF;
    while (u) {
        if (tr[u].v > v) res = min(res, tr[u].v), u = tr[u].s[0];
        else u = tr[u].s[1];
    }
    return res;
}

int query_pre(int u, int l, int r, int  x) {
    if (L[u] >= l && R[u] <= r) return get_pre(T[u], x);
```

```
    int mid = (L[u] + R[u]) >> 1, res = -INF;
    if (l <= mid) res = max(res, query_pre(u << 1, l, r, x));
    if (r > mid) res = max(res, query_pre(u << 1 | 1, l, r, x));
    return res;
}

int query_suc(int u, int l, int r, int x) {
    if (L[u] >= l && R[u] <= r) return get_suc(T[u], x);
    int mid = (L[u] + R[u]) >> 1, res = INF;
    if (l <= mid) res = min(res, query_suc(u << 1, l, r, x));
    if (r > mid) res = min(res, query_suc(u << 1 | 1, l, r, x));
    return res;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) scanf("%d", &w[i]);
    build(1, 1, n);

    while (m--) {
        int op, l, r, x;
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d%d%d", &l, &r, &x);
            printf("%d\n", query(1, l, r, x) + 1);
        }
        else if (op == 2) {
            int a, b;
            scanf("%d%d%d", &a, &b, &x);
            int l = 0, r = 1e8;
            while (l < r) {
                int mid = (l + r + 1) >> 1;
                if (query(1, a, b, mid) + 1 <= x) l = mid;
                else r = mid - 1;
            }
            printf("%d\n", r);
        }
        else if (op == 3) {
            scanf("%d%d", &l, &x);
            modify(1, l, x);
            w[l] = x;
        }
        else if (op == 4) {
            scanf("%d%d%d", &l, &r, &x);
            printf("%d\n", query_pre(1, l, r, x));
        }
        else {
            scanf("%d%d%d", &l, &r, &x);
            printf("%d\n", query_suc(1, l, r, x));
        }
    }
    return 0;
}
```

线段树套线段树

如下立例题(K大数查询):

> 两种操作:
>
> - 如果是 `1 a b c` 的形式，表示在第 a 个位置到第 b 个位置，每个位置加入一个数 c。
> - 如果是 `2 a b c` 的形式，表示询问从第 a 个位置到第 b个位置，第 c 大的数是多少。

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 50010, P = N * 17 * 17, M = N << 2;

int n, m;
struct Tree {
    int l, r, sum, add;
}tr[P];
int L[M], R[M], T[M], idx;
struct Query {
    int op, a, b, c;
}q[N];
vector<int> nums;

int get(int x) { // 整体二分
    return lower_bound(nums.begin(), nums.end(), x) - nums.begin();
}

void build(int u, int l, int r) {
    L[u] = l, R[u] = r, T[u] = ++idx;
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
}

int intersection(int a, int b, int c, int d) {
    return min(b, d) - max(a, c) + 1;
}

void update(int u, int l, int r, int pl, int pr) {
    tr[u].sum += intersection(l, r, pl, pr);
    if (l >= pl && r <= pr) {
        tr[u].add++;
        return;
    }
    int mid = (l + r) >> 1;
    if (pl <= mid) {
        if (!tr[u].l) tr[u].l = ++idx;
        update(tr[u].l, l, mid, pl, pr);
    }
    if (pr > mid) {
        if (!tr[u].r) tr[u].r = ++idx;
        update(tr[u].r, mid + 1, r, pl, pr);
    }
}

```

```cpp
void change(int u, int a, int b, int c) {
    update(T[u], 1, n, a, b);
    if (L[u] == R[u]) return;
    int mid = (L[u] + R[u]) >> 1;
    if (c <= mid) change(u << 1, a, b, c);
    else change(u << 1 | 1, a, b, c);
}

int get_sum(int u, int l, int r, int pl, int pr, int add) {
    if (l >= pl && r <= pr) return tr[u].sum + (r - l + 1) * add;
    int mid = (l + r) >> 1, res = 0;
    add += tr[u].add;
    if (pl <= mid) {
        if (tr[u].l) res += get_sum(tr[u].l, l, mid, pl, pr, add);
        else res += intersection(l, mid, pl, pr) * add;
    }
    if (pr > mid) {
        if (tr[u].r) res += get_sum(tr[u].r, mid + 1, r, pl, pr, add);
        else res += intersection(mid + 1, r, pl, pr) * add;
    }
    return res;
}

int query(int u, int a, int b, int c) {
    if (L[u] == R[u]) return R[u];
    int mid = (L[u] + R[u]) >> 1;
    int k = get_sum(T[u << 1 | 1], 1, n, a, b, 0);
    if (k >= c) return query(u << 1 | 1, a, b, c);
    return query(u << 1, a, b, c - k);
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; ++i) {
        scanf("%d%d%d%d", &q[i].op, &q[i].a, &q[i].b, &q[i].c);
        if (q[i].op == 1) nums.push_back(q[i].c);
    }
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());

    build(1, 0, nums.size() - 1);

    for (int i = 0; i < m; ++i) {
        int op = q[i].op, a = q[i].a, b = q[i].b, c = q[i].c;
        if (op == 1) change(1, a, b, get(c));
        else printf("%d\n", nums[query(1, a, b, c)]);
    }
    return 0;
}
```

分块

时间复杂度: $O(n\sqrt{n})$

思想: 段内暴力维护，整段直接维护，即“大段维护，局部朴素”.

```cpp
// 区间修改+区间求和
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

using namespace std;
typedef long long ll;

const int N = 100010;

ll a[N], sum[N], add[N];
int L[N], R[N];
int pos[N];
int n, m, len;

void init() { // 预处理,分块
    for (int i = 1; i <= len; ++i) {
        L[i] = (i - 1) * len + 1;
        R[i] = i * len;
    }
    if (R[len] < n) len++, L[len] = R[len - 1] + 1, R[len] = n;

    for (int i = 1; i <= len; ++i)
        for (int j = L[i]; j <= R[i]; ++j) {
            pos[j] = i;
            sum[i] += a[j];
        }
}

void change(int l, int r, int d) {
    int p = pos[l], q = pos[r];
    if (p == q) { // 段内,暴力维护
        for (int i = l; i <= r; ++i) a[i] += d;
        sum[p] += d * (r - l + 1);
    }
    else { // 整段得直接维护,不是整段得暴力维护
        for (int i = p + 1; i <= q - 1; ++i) add[i] += d;
        for (int i = l; i <= R[p]; ++i) a[i] += d;
        sum[p] += d * (R[p] - l + 1);
        for (int i = L[q]; i <= r; ++i) a[i] += d;
        sum[q] += d * (r - L[q] + 1);
    }
}

ll ask(int l, int r) {
    int p = pos[l], q = pos[r];
    ll res = 0;
    if (p == q) {
        for (int i = l; i <= r; ++i) res += a[i];
        res += add[p] * (r - l + 1);
    }
```

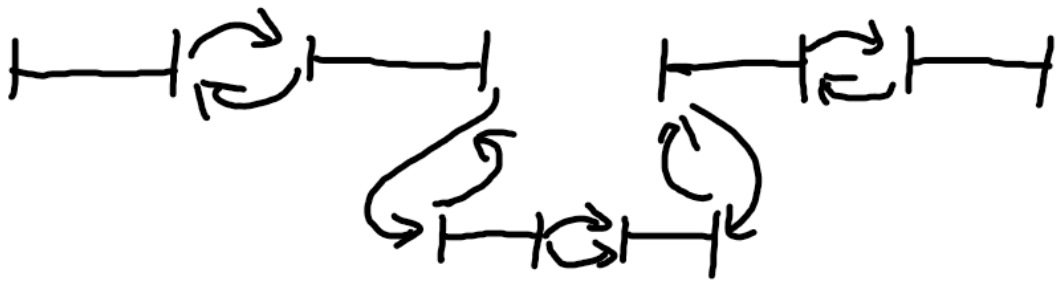
```
    else {
        for (int i = p + 1; i <= q - 1; ++i)
            res += sum[i] + add[i] * (R[i] - L[i] + 1);
        for (int i = l; i <= R[p]; ++i) res += a[i];
        res += add[p] * (R[p] - l + 1);
        for (int i = L[q]; i <= r; ++i) res += a[i];
        res += add[q] * (r - L[q] + 1);
    }
    return res;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) scanf("%lld", &a[i]);

    len = sqrt(n);
    init();

    while (m--) {
        char op[3];
        int l, r, d;
        scanf("%s%d%d", op, &l, &r);
        if (*op == 'C') {
            scanf("%d", &d);
            change(l, r, d);
        }
        else printf("%lld\n", ask(l, r));
    }
    return 0;
}
```

块状链表

块状链表支持合并、插入、删除，查询操作.

插入一段

> 1.分裂节点 $O(\sqrt{n})$
>
> 2.在分裂节点处插入序列 $O(\sqrt{n})$



删除一段

> 1.删除开头节点得后半部分 $O(\sqrt{n})$
>
> 2.删除中间完整节点 $O(\sqrt{2})$

> 3.删除结尾节点的前半部分 $O(\sqrt{n})$

合并

> 遍历整个链表，若下一个点可以合并到当前点，则合并
>
> 合并操作可以保证块状链表的时间复杂度，否则块状链表可能退化成普通数组

如下例题(NOI 2003 文本编辑器)

操作名称	输入文件中的格式	功能
MOVE(k)	Move k	将光标移动到第 k 个字符之后，如果 k=0，将光标移到文本开头
INSERT(n, s)	Insert n s	在光标处插入长度为 n 的字符串 s，光标位置不变，$n \geq 1$
DELETE(n)	Delete n	删除光标后的 n 个字符，光标位置不变，$n \geq 1$
GET(n)	Get n	输出光标后的 n 个字符，光标位置不变，$n \geq 1$
PREV()	Prev	光标前移一个字符
NEXT()	Next	光标后移一个字符

> 任务：
>
> - 建立一个空的文本编辑器。
> - 从输入文件中读入一些操作并执行。
> - 对所有执行过的 `GET` 操作，将指定的内容写入输出文件。

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 2000, M = 2010;

int n, x, y; // x表示当前光标在哪个块,y表示光标在第几个位置
char str[2000010];
int q[M], tt; // 内存回收机制

struct Node {
    char s[N + 1];
    int c, l, r; // c表示块长,l,r分别为链表的左右指针
}p[M];

inline void move(int k) { // 将光标移动到第k个字符后面
    x = p[0].r;
    while (k > p[x].c) k -= p[x].c, x = p[x].r;
    y = k - 1;
}

inline void add(int x, int u) { // 将节点u插入到节点x的右边
    p[u].r = p[x].r, p[p[u].r].l = u;
    p[u].l = x, p[x].r = u;
```

```
}

inline void del(int u) { // 删除节点u
    p[p[u].l].r = p[u].r;
    p[p[u].r].l = p[u].l;
    p[u].c = p[u].l = p[u].r = 0; // 清空节点u
    q[++tt] = u; // 回收节点
}

void insert(int k) { // 在光标后插入k个字符
    if (y < p[x].c - 1) { // 从光标处分裂
        int u = q[tt--]; // 新建一个节点
        for (int i = y + 1; i < p[x].c; ++i)
            p[u].s[p[u].c++] = p[x].s[i]; // 复制节点x光标后的字符
        p[x].c = y + 1; //调整节点x的长度
        add(x, u);
    }
    int cur = x;
    for (int i = 0; i < k;) {
        int u = q[tt--]; // 新建一个节点
        while (p[u].c < N && i < k) {
            p[u].s[p[u].c++] = str[i++];
        }
        add(cur, u);
        cur = u;
    }
}

void remove(int k) { // 删除光标后k个字符
    if (p[x].c - 1 - y >= k) { // 块内删除
        for (int i = y + k + 1, j = y + 1; i < p[x].c; ++i, ++j) p[x].s[j] =
p[x].s[i];
        p[x].c -= k;
    }
    else {
        k -= p[x].c - y - 1; // 删除当前节点的剩余部分
        p[x].c = y + 1;
        while (p[x].r && k >= p[p[x].r].c) {
            int u = p[x].r;
            k -= p[u].c;
            del(u);
        }
        int u = p[x].r;
        for (int i = 0, j = k; j < p[u].c; ++i, ++j) p[u].s[i] = p[u].s[j]; // 删
除结尾的前面
        p[u].c -= k;
    }
}

void get(int k) { // 输出从光标开始的k字符
    if (p[x].c - 1 - y >= k) { //块内输出
        for (int i = 0, j = y + 1; i < k; ++i, ++j) putchar(p[x].s[j]);
    }
    else {
        k -= p[x].c - y - 1;
        for (int i = y + 1; i < p[x].c; ++i) putchar(p[x].s[i]);
        int cur = x;
        while (p[cur].r && k >= p[p[cur].r].c) {
```

```
            int u = p[cur].r;
            for (int i = 0; i < p[u].c; ++i) putchar(p[u].s[i]);
            k -= p[u].c;
            cur = u;
        }
        int u = p[cur].r;
        for (int i = 0; i < k; ++i) putchar(p[u].s[i]);
    }
    puts("");
}

inline void prev() { // 将光标向前移动一位
    if (!y) {
        x = p[x].l;
        y = p[x].c - 1;
    }
    else y--;
}

inline void next() { //将光标向后移动一位
    if (y < p[x].c - 1) y++;
    else {
        x = p[x].r;
        y = 0;
    }
}

void merge() { // 将长度较短的相邻节点合并
    for (int i = p[0].r; i; i = p[i].r) { // 遍历整个链表
        while (p[i].r && p[i].c + p[p[i].r].c < N) {
            int r = p[i].r;
            for (int j = p[i].c, k = 0; k < p[r].c; ++j, ++k)
                p[i].s[j] = p[r].s[k];
            if (x == r) x = i, y += p[i].c; // 更新光标位置
            p[i].c += p[r].c;
            del(r);
        }
    }
}

int main() {
    for (int i = 1; i < M; ++i) q[++tt] = i;
    scanf("%d", &n);
    char op[10];

    str[0] = '>'; // 哨兵
    insert(1);
    move(1);

    while (n--) {
        int a;
        scanf("%s", op);
        if (!strcmp(op, "Move")) {
            scanf("%d", &a);
            move(a + 1);
        }
        else if (!strcmp(op, "Insert")) {
            scanf("%d", &a);
```

```
            int i = 0, k = a;
            while (a) {
                str[i] = getchar();
                if (str[i] >= 32 && str[i] <= 126) i++, a--;
            }
            insert(k);
            merge();
        }
        else if (!strcmp(op, "Delete")) {
            scanf("%d", &a);
            remove(a);
            merge();
        }
        else if (!strcmp(op, "Get")) {
            scanf("%d", &a);
            get(a);
        }
        else if (!strcmp(op, "Prev")) prev();
        else next();
    }
    return 0;
}
```

莫队

莫对可以维护这么一个问题：在一段数列中，有多少种不同的数

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>

using namespace std;
const int N = 50010, M = 200010, S = 2000010;

int n, m, len;
int w[N], ans[M], cnt[S];
struct Query {
    int id, l, r;
}q[M];

void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

int get(int x) {
    return x / len;
}

bool cmp(const Query& a, const Query& b) { // 玄学奇偶排序
    int i = get(a.l), j = get(b.l);
```

```
    if (i != j) return i < j;
    return a.r < b.r;
}

void add(int x, int& res) {
    if (!cnt[x]) res++;
    cnt[x]++;
}

void del(int x, int& res) {
    cnt[x]--;
    if (!cnt[x]) res--;
}

int main() {
    read(n);
    for (int i = 1; i <= n; i++) read(w[i]);
    read(m);

    len = sqrt((double)n * n / m); // 分块

    for (int i = 0; i < m; i++) {
        int l, r;
        read(l), read(r);
        q[i] = { i,l,r };
    }
    sort(q, q + m, cmp);

    for (int k = 0, i = 0, j = 1, res = 0; k < m; k++) { // j在左,i在右
        int id = q[k].id, l = q[k].l, r = q[k].r;
        while (i < r) add(w[++i], res); // i小于r,向右添加(移动)
        while (i > r) del(w[i--], res); // i大于r,向左删除(移动)
        while (j < l) del(w[j++], res); // j小于l,向右删除(移动)
        while (j > l) add(w[--j], res); // j大于l,向左添加(移动)
        ans[id] = res;
    }
    for (int i = 0; i < m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

回滚莫队

回滚莫队支持删除，把删除操作转换成回滚添加

```
// 历史的研究
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>

using namespace std;

const int N = 100010;
```

```cpp

int n, m, len;
int w[N], cnt[N];
long long ans[N];
vector<int> nums; // 离散化

struct Query {
    int id, l, r;
}q[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

int get(int x) {
    return x / len;
}

bool cmp(const Query& a, const Query& b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    return a.r < b.r;
}

void add(int x, long long& res) {
    cnt[x]++;
    res = max(res, (long long)cnt[x] * nums[x]);
}

int main() {
    scanf("%d%d", &n, &m);
    len = sqrt(n);
    for (int i = 1; i <= n; i++) read(w[i]), nums.push_back(w[i]);
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    for (int i = 1; i <= n; i++) w[i] = lower_bound(nums.begin(), nums.end(), w[i])
- nums.begin();

    for (int i = 0; i < m; i++) {
        int l, r;
        read(l), read(r);
        q[i] = { i,l,r };
    }
    sort(q, q + m, cmp);

    for (int x = 0; x < m;) {
        int y = x;
        while (y < m && get(q[y].l) == get(q[x].l)) y++; // 此时[x,y]左端点在同一块内
        int right = get(q[x].l) * len + len - 1;
        // 暴力求解块内询问
        while (x < y && q[x].r <= right) {
            long long res = 0;
            int id = q[x].id, l = q[x].l, r = q[x].r;
            for (int k = l; k <= r; k++) add(w[k], res);
            ans[id] = res;
```

```
            for (int k = l; k <= r; k++) cnt[w[k]]--; // 回滚
            x++;
        }
        // 求解块外询问
        long long res = 0;
        int i = right, j = right + 1;
        while (x < y) {
            int id = q[x].id, l = q[x].l, r = q[x].r;
            while (i < r) add(w[++i], res);
            long long backup = res;
            while (j > l) add(w[--j], res);
            ans[id] = res;
            while (j < right + 1) cnt[w[j++]]--; // 回滚
            res = backup;
            x++;
        }
        memset(cnt, 0, sizeof cnt);
    }
    for (int i = 0; i < m; i++) printf("%lld\n", ans[i]);
    return 0;
}
```

树上莫队

树上莫队可以查询一棵树上 (u, v) 有多少种权值不同的点

欧拉序: 一种对树进行 `dfs` 的一种序列

欧拉序求法: 对树 `dfs` 每次访问到某个节点的时候记录一下，当离开这个节点的时候在记录一下

first[u] : u 第一次出现的位置 `last[u] :` u 最后一次出现的位置

> 1.求欧拉序列
>
> 2.求 lca
>
> 3.将树上路径询问转化为序列区间询问
>
> 4.普通莫队

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
#include<cmath>

using namespace std;

const int N = 100010;

int  head[N], to[N], ne[N], w[N], tot;
int n, m, len;
int depth[N], f[N][16];
int seq[N], top, first[N], last[N];
int cnt[N], st[N], ans[N];
```

```cpp
int que[N];
vector<int> nums;

struct Query {
    int id, l, r, p;
}q[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

inline void add_edge(int u, int v) {
    to[tot] = v, ne[tot] = head[u], head[u] = tot++;
}

void dfs(int u, int father) {
    seq[++top] = u;
    first[u] = top;
    for (int i = head[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j != father) dfs(j, u);
    }
    seq[++top] = u;
    last[u] = top;
}

int get(int x) {
    return x / len;
}

bool cmp(const Query& a, const Query& b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    return a.r < b.r;
}

void add(int x, int& res) {
    st[x] ^= 1;
    if (st[x] == 0) {
        cnt[w[x]]--;
        if (!cnt[w[x]]) res--;
    }
    else {
        if (!cnt[w[x]]) res++;
        cnt[w[x]]++;
    }
}

void bfs() {
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[1] = 1;
    int hh = 0, tt = 0;
    que[0] = 1;
    while (hh <= tt) {
        int t = que[hh++];
```

```
        for (int i = head[t]; ~i; i = ne[i]) {
            int j = to[i];
            if (depth[j] > depth[t] + 1) {
                depth[j] = depth[t] + 1;
                f[j][0] = t;
                for (int k = 1; k <= 15; k++) {
                    f[j][k] = f[f[j][k - 1]][k - 1];
                }
                que[++tt] = j;
            }
        }
    }
}

int lca(int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 15; k >= 0; k--) {
        if (depth[f[a][k]] >= depth[b])
            a = f[a][k];
    }
    if (a == b) return a;
    for (int k = 15; k >= 0; k--) {
        if (f[a][k] != f[b][k]) {
            a = f[a][k];
            b = f[b][k];
        }
    }
    return f[a][0];
}

int main() {
    read(n), read(m);

    for (int i = 1; i <= n; i++) read(w[i]), nums.push_back(w[i]);
    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    for (int i = 1; i <= n; i++)
        w[i] = lower_bound(nums.begin(), nums.end(), w[i]) - nums.begin();

    memset(head, -1, sizeof head);
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        read(a), read(b);
        add_edge(a, b), add_edge(b, a);
    }

    dfs(1, -1); // 求欧拉序列
    bfs();

    for (int i = 0; i < m; i++) {
        int a, b;
        read(a), read(b);
        if (first[a] > first[b]) swap(a, b);
        int p = lca(a, b);
        if (a == p) q[i] = { i,first[a],first[b] };
        else q[i] = { i,last[a],first[b],p };
    }

```

```
    len = sqrt(top);
    sort(q, q + m, cmp);

    for (int i = 0, L = 1, R = 0, res = 0; i < m; i++) {
        int id = q[i].id, l = q[i].l, r = q[i].r, p = q[i].p;
        while (R < r) add(seq[++R], res);
        while (R > r) add(seq[R--], res);
        while (L < l) add(seq[L++], res);
        while (L > l) add(seq[--L], res);
        if (p) add(p, res);
        ans[id] = res;
        if (p) add(p, res);
    }

    for (int i = 0; i < m; i++) printf("%d\n", ans[i]);
    return 0;
}
```

二次离线莫队

> 给定一个长度为 n 的序列 $a_1, a_2, \ldots, a_n$ 以及一个整数 k。
>
> 现在要进行 m 次询问，每次询问给定一个区间 $[l, r]$，请你求出共有多少个数对 $(i, j)(i, j)$ 满足 $l \leq i < j \leq r$ 且 $a_i \oplus a_j$ 的结果在二进制表示下恰好有 k 个 1 。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
#include<vector>
using namespace std;

typedef long long ll;
const int N = 100010;

int n, m, k, len;
int w[N];
ll ans[N];

struct  Query {
    int id, l, r;
    ll res;
}q[N];

struct Range {
    int id, l, r, t;
};

vector<Range> range[N];
int f[N], g[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
```

```
    x *= f;
}

inline int get_count(int x) {
    int res = 0;
    while (x) res += x & 1, x >>= 1;
    return res;
}

inline int get(int x) {
    return x / len;
}

bool cmp(const Query& a, const Query& b) {
    int i = get(a.l), j = get(b.l);
    if (i != j) return i < j;
    return a.r < b.r;

}

int main() {
    read(n), read(m), read(k);
    for (int i = 1; i <= n; i++) read(w[i]);
    vector<int> nums;
    for (int i = 0; i < 1 << 14; ++i) {
        if (get_count(i) == k)
            nums.push_back(i);
    }
    for (int i = 1; i <= n; i++) {
        for (int y : nums) ++g[w[i] ^ y];
        f[i] = g[w[i + 1]];
    }

    for (int i = 0; i < m; i++) {
        int l, r;
        read(l), read(r);
        q[i] = { i,l,r };
    }

    len = sqrt(n);
    sort(q, q + m, cmp);

    for (int i = 0, L = 1, R = 0; i < m; ++i) {
        int l = q[i].l, r = q[i].r;

        if (R < r) range[L - 1].push_back({ i,R + 1,r,-1 });
        while (R < r) q[i].res += f[R++];

        if (R > r) range[L - 1].push_back({ i,r + 1,R,1 });
        while (R > r) q[i].res -= f[--R];

        if (L < l) range[R].push_back({ i,L,l - 1,-1 });
        while (L < l) q[i].res += f[L - 1] + (!k), L++;

        if (L > l) range[R].push_back({ i,l,L - 1,1 });
        while (L > l) q[i].res -= f[L - 2] + (!k), L--;
    }

```

```
    memset(g, 0, sizeof g);
    for (int i = 1; i <= n; ++i) {
        for (int y : nums) ++g[w[i] ^ y];
        for (Range& rg : range[i]) {
            int id = rg.id, l = rg.l, r = rg.r, t = rg.t;
            for (int x = l; x <= r; ++x) {
                q[id].res += g[w[x]] * t;
            }
        }
    }

    for (int i = 1; i < m; ++i) q[i].res += q[i - 1].res;
    for (int i = 0; i < m; ++i) ans[q[i].id] = q[i].res;
    for (int i = 0; i < m; ++i) printf("%lld\n", ans[i]);
    return 0;
}
```

树链剖分

> - `1 u v k`，修改路径上节点权值，将节点 u 和节点 v 之间路径上的所有节点（包括这两个节点）的权值增加 k 。
> - `2 u k`，修改子树上节点权值，将以节点 u 为根的子树上的所有节点的权值增加 k 。
> - `3 u v`，询问路径，询问节点 u 和节点 v 之间路径上的所有节点（包括这两个节点）的权值和。
> - `4 u`，询问子树，询问以节点 u 为根的子树上的所有节点的权值和。

核心：

1.将一棵树转化成一个序列

2.将树中的路径转化成 $logn$ 段连续区间

重要性质：树中任意一条路径均可拆分成 $O(logn)$ 个连续区间，即 $O(logn)$ 条重链

`dfs` 序：优先遍历重儿子 $\Longrightarrow$ 重链编号是连续的

重儿子：子树节点数目最多的节点

轻儿子：除重儿子外的节点

重边：重儿子和它父节点之间的边

轻边：除重边外的其余边

重链：极大的由重边构成的路径

轻链：极大的由轻边构成的路径

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

using namespace std;
typedef long long ll;

const int N = 100010, M = N * 2;

```

```cpp
int n, m;
int h[N], w[N], ne[M], to[M], tot;
int id[N], nw[N], cnt;
int dep[N], sz[N], top[N], fa[N], son[N];

struct Segment {
    int l, r;
    ll add, sum;
}tr[N << 2];

inline void add(int u, int v) {
    to[tot] = v, ne[tot] = h[u], h[u] = tot++;
}

void dfs1(int u, int father, int depth) {
    dep[u] = depth, fa[u] = father, sz[u] = 1;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == father) continue;
        dfs1(j, u, depth + 1);
        sz[u] += sz[j];
        if (sz[son[u]] < sz[j]) son[u] = j;
    }
}

void dfs2(int u, int t) {
    id[u] = ++cnt, nw[cnt] = w[u], top[u] = t;
    if (!son[u]) return;
    dfs2(son[u], t);
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == fa[u] || j == son[u]) continue;
        dfs2(j, j);
    }
}

inline void pushup(int u) {
    tr[u].sum = tr[u << 1].sum + tr[u << 1 | 1].sum;
}

inline void pushdown(int u) {
    Segment& root = tr[u], & left = tr[u << 1], & right = tr[u << 1 | 1];
    if (root.add) {
        left.add += root.add, left.sum += root.add * (left.r - left.l + 1);
        right.add += root.add, right.sum += root.add * (right.r - right.l + 1);
        root.add = 0;
    }
}

void build(int u, int l, int r) {
    tr[u] = { l,r,0,nw[r] };
    if (l == r) return;
    int mid = (l + r) >> 1;
    build(u << 1, l, mid), build(u << 1 | 1, mid + 1, r);
    pushup(u);
}

void update(int u, int l, int r, int k) {
```

```
    if (l <= tr[u].l && r >= tr[u].r) {
        tr[u].add += k;
        tr[u].sum += k * (tr[u].r - tr[u].l + 1);
        return;
    }
    pushdown(u);
    int mid = (tr[u].l + tr[u].r) >> 1;
    if (l <= mid) update(u << 1, l, r, k);
    if (r > mid) update(u << 1 | 1, l, r, k);
    pushup(u);
}

ll query(int u, int l, int r) {
    if (l <= tr[u].l && r >= tr[u].r) return tr[u].sum;
    pushdown(u);
    int mid = (tr[u].l + tr[u].r) >> 1;
    ll res = 0;
    if (l <= mid) res += query(u << 1, l, r);
    if (r > mid) res += query(u << 1 | 1, l, r);
    return res;
}

void update_path(int u, int v, int k) {
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        update(1, id[top[u]], id[u], k);
        u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    update(1, id[v], id[u], k);
}

ll query_path(int u, int v) {
    ll res = 0;
    while (top[u] != top[v]) {
        if (dep[top[u]] < dep[top[v]]) swap(u, v);
        res += query(1, id[top[u]], id[u]);
        u = fa[top[u]];
    }
    if (dep[u] < dep[v]) swap(u, v);
    res += query(1, id[v], id[u]);
    return res;
}

void update_tree(int u, int k) {
    update(1, id[u], id[u] + sz[u] - 1, k);
}

ll query_tree(int u) {
    return query(1, id[u], id[u] + sz[u] - 1);
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) scanf("%d", &w[i]);
    memset(h, -1, sizeof h);
    for (int i = 0; i < n - 1; ++i) {
        int a, b;
```

```
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }

    dfs1(1, -1, 1);
    dfs2(1, 1);
    build(1, 1, n);

    scanf("%d", &m);
    while (m--) {
        int t, u, v, k;
        scanf("%d%d", &t, &u);
        if (t == 1) {
            scanf("%d%d", &v, &k);
            update_path(u, v, k);
        }
        else if (t == 2) {
            scanf("%d", &k);
            update_tree(u, k);
        }
        else if (t == 3) {
            scanf("%d", &v);
            printf("%lld\n", query_path(u, v));
        }
        else printf("%lld\n", query_tree(u));
    }
    return 0;
}
```

动态树(LCT)

> - `0 x y`，表示询问点 x 到点 y 之间的路径上的所有点（包括两端点）的权值的异或和。保证 x 和 y 之间存在连通路径。
> - `1 x y`，表示在点 x 和点 y 之间增加一条边 $(x,y)(x,y)$ 。注意：**如果两点已经处于连通状态，则无视该操作。**
> - `2 x y`，表示删除边 $(x,y)(x,y)$ 。注意：**如果该边不存在，则无视该操作。**
> - `3 x w`，表示将点 x 的权值修改为 w 。

实边：子，父节点均知道

虚边：子节点知道父节点，父节点不知道子节点

`splay` 维护所有实边路径

`splay` 中的后继前驱维护树中的父子关系

虚边用 `splay` 根节点来维护

相关操作：

`access(x)` ： 建立一条从根节点到 x 的实边路径

`made_root(x)` ： 将 x 变成根节点

`findroot(x)` ： 找到 x 所在树的根节点(判断 x,y 是不是在同一棵树)

`split(x,y)` ： 将 x 到 y 的路径变成实边路径

`link(x,y)`： 若 x,y 不连通，则加入 (x,y) 这条边

`cut(x,y)`： 若 x,y 之间有边，则删除

```
#include<iostream>
#include<cstdio>
#include<algorithm>

using namespace std;

const int N = 100010;

int n, m;
int stk[N];

struct Node {
    int s[2], p, v;
    int sum, rev;
}tr[N];

inline void pushrev(int x) {
    swap(tr[x].s[0], tr[x].s[1]);
    tr[x].rev ^= 1;
}

inline void pushup(int x) {
    tr[x].sum = tr[tr[x].s[0]].sum ^ tr[x].v ^ tr[tr[x].s[1]].sum;
}

inline void pushdown(int x) {
    if (tr[x].rev) {
        pushrev(tr[x].s[0]), pushrev(tr[x].s[1]);
        tr[x].rev = 0;
    }
}

inline bool isroot(int x) {
    return tr[tr[x].p].s[0] != x && tr[tr[x].p].s[1] != x;
}

void rotate(int x) {
    int y = tr[x].p, z = tr[y].p;
    int k = tr[y].s[1] == x;
    if (!isroot(y)) tr[z].s[tr[z].s[1] == y] = x;
    tr[x].p = z;
    tr[y].s[k] = tr[x].s[k ^ 1], tr[tr[x].s[k ^ 1]].p = y;
    tr[x].s[k ^ 1] = y, tr[y].p = x;
    pushup(y), pushup(x);
}

void splay(int x) {
    int top = 0, r = x;
    stk[++top] = r;
    while (!isroot(r)) stk[++top] = r = tr[r].p;
    while (top) pushdown(stk[top--]);
    while (!isroot(x)) {
        int y = tr[x].p, z = tr[y].p;
```

```
        if (!isroot(y))
            if ((tr[y].s[1] == x) ^ (tr[z].s[1] == y)) rotate(x);
            else rotate(y);
        rotate(x);
    }
}

void access(int x) { // 建立一条从根到x的路径,同时将x变成splay的根节点
    int z = x;
    for (int y = 0; x; y = x, x = tr[x].p) {
        splay(x);
        tr[x].s[1] = y, pushup(x);
    }
    splay(z);
}

void makeroot(int x) { // 将x变成原树的根节点
    access(x);
    pushrev(x);
}

int findroot(int x) { // 找到x所在原树的根节点,再将原树的根节点旋转到splay的根节点
    access(x);
    while (tr[x].s[0]) pushdown(x), x = tr[x].s[0];
    splay(x);
    return x;
}

void split(int x, int y) { // x和y之间的路径建立一个splay,其根节点是y
    makeroot(x);
    access(y);
}

void link(int x, int y) { // 若x和y不连通,则在x和y之间连一条边
    makeroot(x);
    if (findroot(y) != x) tr[x].p = y;
}

void cut(int x, int y) { // 若x和y之间存在边,则删除该边
    makeroot(x);
    if (findroot(y) == x && tr[y].p == x && !tr[y].s[0]) {
        tr[x].s[1] = tr[y].p = 0;
        pushup(x);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) scanf("%d", &tr[i].v);
    while (m--) {
        int t, x, y;
        scanf("%d%d%d", &t, &x, &y);
        if (t == 0) {
            split(x, y);
            printf("%d\n", tr[y].sum);
        }
        else if (t == 1) link(x, y);
        else if (t == 2) cut(x, y);
```

```
        else {
            splay(x);
            tr[x].v = y;
            pushup(x);
        }
    }
    return 0;
}
```

DLX之精确覆盖问题

> 在01矩阵中找到一个行的集合，使得这些行中，每一列都有且仅有一个数字 1 。

```
#include<iostream>
#include<cstdio>

using namespace std;
const int N = 5520;

int n, m;
int l[N], r[N], u[N], d[N], col[N], row[N], s[N], idx;
int ans[N], top;

void init() {
    for (int i = 0; i <= m; ++i) {
        l[i] = i - 1, r[i] = i + 1;
        u[i] = d[i] = i;
    }
    l[0] = m, r[m] = 0;
    idx = m + 1;
}

inline void add(int& hh, int& tt, int x, int y) { // 十字链表
    row[idx] = x, col[idx] = y, s[y]++;
    u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
    r[hh] = l[tt] = idx;
    r[idx] = tt, l[idx] = hh;
    tt = idx++;
}

void remove(int p) {
    r[l[p]] = r[p], l[r[p]] = l[p];
    for (int i = d[p]; i != p; i = d[i])
        for (int j = r[i]; j != i; j = r[j]) {
            s[col[j]]--;
            u[d[j]] = u[j], d[u[j]] = d[j];
        }

}

void resume(int p) {
    for (int i = u[p]; i != p; i = u[i])
        for (int j = l[i]; j != i; j = l[j]) {
            u[d[j]] = j, d[u[j]] = j;
            s[col[j]]++;
```

```
        }
    r[l[p]] = p, l[r[p]] = p;
}

bool dfs() {
    if (!r[0]) return true;
    int p = r[0];
    for (int i = r[0]; i; i = r[i])
        if (s[i] < s[p])
            p = i;
    remove(p);
    for (int i = d[p]; i != p; i = d[i]) {
        ans[++top] = row[i];
        for (int j = r[i]; j != i; j = r[j]) remove(col[j]);
        if (dfs()) return true;
        for (int j = l[i]; j != i; j = l[j]) resume(col[j]);
        top--;
    }
    resume(p);
    return false;
}

int main() {
    scanf("%d%d", &n, &m);
    init();
    for (int i = 1; i <= n; ++i) {
        int hh = idx, tt = idx;
        for (int j = 1; j <= m; ++j) {
            int x;
            scanf("%d", &x);
            if (x) add(hh, tt, i, j);
        }
    }

    if (dfs()) {
        for (int i = 1; i <= top; ++i) printf("%d ", ans[i]);
    }
    else puts("No Solution!");
    return 0;

}
```

DLX之重复覆盖问题

> 请你01在矩阵中找到一个行的集合，使得这些行中，每一列都包含数字 1 ，并且集合中包含的行数尽可能少。

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

using namespace std;

const int N = 10010;
```

```
9
10  int n, m;
11  int l[N], r[N], u[N], d[N], row[N], col[N], s[N], idx;
12  int ans[N];
13  bool vis[110];
14
15  void init() {
16      for (int i = 0; i <= m; ++i) {
17          l[i] = i - 1, r[i] = i + 1;
18          col[i] = u[i] = d[i] = i;
19          s[i] = 0;
20      }
21      l[0] = m, r[m] = 0;
22      idx = m + 1;
23  }
24
25  inline void add(int& hh, int& tt, int x, int y) {
26      row[idx] = x, col[idx] = y, s[y]++;
27      u[idx] = y, d[idx] = d[y], u[d[y]] = idx, d[y] = idx;
28      r[hh] = l[tt] = idx, r[idx] = tt, l[idx] = hh;
29      tt = idx++;
30  }
31
32  inline void remove(int p) {
33      for (int i = d[p]; i != p; i = d[i]) {
34          r[l[i]] = r[i];
35          l[r[i]] = l[i];
36      }
37  }
38
39  inline void resume(int p) {
40      for (int i = u[p]; i != p; i = u[i]) {
41          r[l[i]] = i;
42          l[r[i]] = i;
43      }
44  }
45
46  int h() {
47      int cnt = 0;
48      memset(vis, 0, sizeof vis);
49      for (int i = r[0]; i; i = r[i]) {
50          if (vis[col[i]]) continue;
51          cnt++;
52          vis[col[i]] = true;
53          for (int j = d[i]; j != i; j = d[j])
54              for (int k = r[j]; k != j; k = r[k])
55                  vis[col[k]] = true;
56      }
57      return cnt;
58  }
59
60  bool dfs(int k, int depth) {
61      if (k + h() > depth) return false;
62      if (!r[0]) return true;
63      int p = r[0];
64      for (int i = r[0]; i; i = r[i])
65          if (s[p] > s[i])
66              p = i;
```

```
    for (int i = d[p]; i != p; i = d[i]) {
        ans[k] = row[i];
        remove(i);
        for (int j = r[i]; j != i; j = r[j]) remove(j);
        if (dfs(k + 1, depth)) return true;
        for (int j = l[i]; j != i; j = l[j]) resume(j);
        resume(i);
    }
    return false;
}

int main() {
    scanf("%d%d", &n, &m);
    init();
    for (int i = 1; i <= n; ++i) {
        int hh = idx, tt = idx;
        for (int j = 1; j <= m; ++j) {
            int x;
            scanf("%d", &x);
            if (x) add(hh, tt, i, j);
        }
    }

    int depth = 0;
    while (!dfs(0, depth)) depth++;
    printf("%d\n", depth);
    for (int i = 0; i < depth; ++i) printf("%d ", ans[i]);
    return 0;
}
```

左偏树(可并堆)

> 1. `1 a`，在集合中插入一个新堆，堆中只包含一个数 a 。
> 2. `2 x y`，将第 x 个插入的数和第 yy 个插入的数所在的小根堆合并。数据保证两个数均未被删除。若两数已在同一堆中，则忽略此操作。
> 3. `3 x`，输出第 x 个插入的数所在小根堆的最小值。数据保证该数未被删除。
> 4. `4 x`，删除第 x 个插入的数所在小根堆的最小值（若最小值不唯一，则优先删除先插入的数）。数据保证该数未被删除。

操作:

1.插入一个数 $O(logn)$

2.求最小值 $O(1)$

3.删除最小值 $O(logn)$

4.合并两棵树 $O(logn)$

`f(x)`： 距离为 k 的子树至少包含多少个点

`merge(a,b)`： 合并子树 a 与 b ，并返回合并后的节点。并查集换根

```
#include<iostream>
#include<cstdio>
#include<algorithm>
```

```cpp

using namespace std;

const int N = 200010;

int n;
int v[N], dist[N], l[N], r[N], idx;
int fa[N];

int find(int x) {
    return fa[x] == x ? fa[x] : fa[x] = find(fa[x]);
}

inline bool cmp(int x, int y) {
    if (v[x] != v[y]) return v[x] < v[y];
    return x < y;
}

int merge(int x, int y) {
    if (!x || !y) return x + y;
    if (cmp(y, x)) swap(x, y);
    r[x] = merge(r[x], y);
    if (dist[r[x]] > dist[l[x]]) swap(l[x], r[x]);
    dist[x] = dist[r[x]] + 1;
    return x;
}

int main() {
    scanf("%d", &n);
    v[0] = 2e9;
    while (n--) {
        int t, x, y;
        scanf("%d%d", &t, &x);
        if (t == 1) {
            v[++idx] = x;
            dist[idx] = 1;
            fa[idx] = idx;
        }
        else if (t == 2) {
            scanf("%d", &y);
            x = find(x), y = find(y);
            if (x != y) {
                if (cmp(y, x)) swap(x, y);
                fa[y] = x;
                merge(x, y);
            }
        }
        else if (t == 3) {
            printf("%d\n", v[find(x)]);
        }
        else {
            x = find(x);
            if (cmp(r[x], l[x])) swap(l[x], r[x]);
            fa[x] = l[x], fa[l[x]] = l[x];
            merge(l[x], r[x]);
        }
    }
    return 0;
```

```
}
```

点分治

> 树上两个节点 x 与 y 之间的路径长度就是路径上各边的权值和。求长度不超过 K 的路径有多少条。

即统计长度小于等于 k 的路径有多少条，设 u 为根，那么满足条件的路径可以分为以下两种:

1.经过 u　2.不经过 u

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 10010, M = N * 2;

int n, m;
int h[N], to[M], w[M], ne[M], idx;
bool vis[N];
int p[N], q[N];

inline void add(int u, int v, int c) {
    to[idx] = v, w[idx] = c, ne[idx] = h[u], h[u] = idx++;
}

int get_size(int u, int fa) {
    if (vis[u]) return 0;
    int res = 1;
    for (int i = h[u]; ~i; i = ne[i])
        if (to[i] != fa)
            res += get_size(to[i], u);
    return res;
}

int get_wc(int u, int fa, int tot, int& wc) {
    if (vis[u]) return 0;
    int sum = 1, ms = 0;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == fa) continue;
        int t = get_wc(j, u, tot, wc);
        ms = max(ms, t);
        sum += t;
    }
    ms = max(ms, tot - sum);
    if (ms <= tot / 2) wc = u;
    return sum;
}

void get_dist(int u, int fa, int dist, int& qt) {
    if (vis[u]) return;
    q[qt++] = dist;
```

```
    for (int i = h[u]; ~i; i = ne[i])
        if (to[i] != fa)
            get_dist(to[i], u, dist + w[i], qt);
}

int get(int a[], int k) {
    sort(a, a + k);
    int res = 0;
    for (int i = k - 1, j = -1; i >= 0; --i) {
        while (j + 1 < i && a[j + 1] + a[i] <= m) j++;
        j = min(j, i - 1);
        res += j + 1;
    }
    return res;
}

int calc(int u) {
    if (vis[u]) return 0;
    int res = 0;
    get_wc(u, -1, get_size(u, -1), u);
    vis[u] = true;

    int pt = 0;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i], qt = 0;
        get_dist(j, -1, w[i], qt);
        res -= get(q, qt);
        for (int k = 0; k < qt; ++k) {
            if (q[k] <= m) res++;
            p[pt++] = q[k];
        }
    }
    res += get(p, pt);

    for (int i = h[u]; ~i; i = ne[i]) res += calc(to[i]);
    return res;
}

int main() {
    while (scanf("%d%d", &n, &m), n || m) {
        memset(h, -1, sizeof h);
        memset(vis, 0, sizeof vis);
        idx = 0;
        for (int i = 0; i < n - 1; ++i) {
            int a, b, c;
            scanf("%d%d%d", &a, &b, &c);
            add(a, b, c), add(b, a, c);
        }
        printf("%d\n", calc(0));
    }
    return 0;
}
```

CDQ分治

解决三维偏序问题

```
#include<iostream>
#include<cstdio>
#include<algorithm>

using namespace std;

const int N = 100010, M = 200010;

int n, m;
int tr[M], ans[N];

struct Data {
    int a, b, c, s, res;

    bool operator<(const Data& t)const {
        if (a != t.a) return a < t.a;
        if (b != t.b) return b < t.b;
        return c < t.c;
    }
    bool operator==(const Data& t)const {
        return a == t.a && b == t.b && c == t.c;
    }
}q[N], w[N];

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

inline void add(int x, int v) {
    for (; x < M; x += lowbit(x)) tr[x] += v;
}

inline int sum(int x) {
    int res = 0;
    for (; x; x -= lowbit(x)) res += tr[x];
    return res;
}

void merge_sort(int l, int r) {
    if (l >= r) return;
    int mid = (l + r) >> 1;
    merge_sort(l, mid), merge_sort(mid + 1, r);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r)
        if (q[i].b <= q[j].b) add(q[i].c, q[i].s), w[k++] = q[i++];
        else q[j].res += sum(q[j].c), w[k++] = q[j++];
    while (i <= mid) add(q[i].c, q[i].s), w[k++] = q[i++];
    while (j <= r) q[j].res += sum(q[j].c), w[k++] = q[j++];
    for (i = l; i <= mid; ++i) add(q[i].c, -q[i].s);
    for (i = l, j = 0; j < k; i++, j++) q[i] = w[j];
}

int main() {
```

```
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; ++i) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        q[i] = { a,b,c,1 };
    }

    sort(q, q + n);

    int k = 1;
    for (int i = 1; i <= n; ++i)
        if (q[i] == q[k - 1]) q[k - 1].s++;
        else q[k++] = q[i];

    merge_sort(0, k - 1);
    for (int i = 0; i < k; ++i)
        ans[q[i].res + q[i].s - 1] += q[i].s;

    for (int i = 0; i < n; ++i) printf("%d\n", ans[i]);
    return 0;
}
```

仙人掌

把仙人掌图转化为圆方树

圆点：仙人掌原有的点

方点：新加进去的点

`tarjan` 找环，建树

```
// 求仙人掌最短路
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 12010, M = N * 3;

int n, m, new_n, Q;
int h1[N], h2[N], to[M], w[M], ne[M], tot;
int dfn[N], low[N], cnt;
int s[N], stot[N], fu[N], fw[N];
int fa[N][14], depth[N], d[N];
int A, B;

inline void add(int h[], int u, int v, int c) {
    to[tot] = v, w[tot] = c, ne[tot] = h[u], h[u] = tot++;
}

void build_circle(int x, int y, int z) {
    int sum = z;
    for (int k = y; k != x; k = fu[k]) {
```

```
        s[k] = sum;
        sum += fw[k];
    }
    s[x] = stot[x] = sum;
    add(h2, x, ++new_n, 0); //square
    for (int k = y; k != x; k = fu[k]) {
        stot[k] = sum;
        add(h2, new_n, k, min(s[k], sum - s[k]));
    }
}

void tarjan(int u, int from) {
    dfn[u] = low[u] = ++cnt;
    for (int i = h1[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!dfn[j]) {
            fu[j] = u, fw[j] = w[i];
            tarjan(j, i);
            low[u] = min(low[u], low[j]);
            if (dfn[u] < low[j]) add(h2, u, j, w[i]);
        }
        else if (i != (from ^ 1)) low[u] = min(low[u], dfn[j]);
    }

    for (int i = h1[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (dfn[u] < dfn[j] && fu[j] != u)
            build_circle(u, j, w[i]);
    }
}

void dfs(int u, int father) {
    depth[u] = depth[father] + 1;
    fa[u][0] = father;
    for (int k = 1; k <= 13; ++k)
        fa[u][k] = fa[fa[u][k - 1]][k - 1];
    for (int i = h2[u]; ~i; i = ne[i]) {
        int j = to[i];
        d[j] = d[u] + w[i];
        dfs(j, u);
    }
}

int lca(int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    for (int k = 13; k >= 0; --k)
        if (depth[fa[a][k]] >= depth[b]) a = fa[a][k];
    if (a == b) return a;
    for (int k = 13; k >= 0; --k)
        if (fa[a][k] != fa[b][k])
            a = fa[a][k], b = fa[b][k];
    A = a, B = b;
    return fa[a][0];
}

int main() {
    scanf("%d%d%d", &n, &m, &Q);
    new_n = n;
```

```
    memset(h1, -1, sizeof h1);
    memset(h2, -1, sizeof h2);
    while (m--) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(h1, a, b, c), add(h1, b, a, c);
    }
    tarjan(1, -1);
    dfs(1, 0);

    while (Q--) {
        int a, b;
        scanf("%d%d", &a, &b);
        int p = lca(a, b);
        if (p <= n) printf("%d\n", d[a] + d[b] - 2 * d[p]);
        else {
            int da = d[a] - d[A], db = d[b] - d[B];
            int l = abs(s[A] - s[B]);
            int dm = min(l, stot[A] - l);
            printf("%d\n", da + dm + db);
        }
    }
    return 0;
}
```

图论

链式前向星

```
int h[N], to[M], ne[M], w[M], idx;

inline void add(int u, int v, int c){
    to[idx] = v, w[idx] = c, ne[idx] = h[u], h[u] = idx++;
}
```

拓扑排序

DAG(有向无环图)一定存在拓扑排序

1.寻找入度为 0 的点，放入队列中

2.弹出队头，访问该点的邻点，并把邻点入读减 1 ，若邻点入读变为 0 ，则把该点放入队列中

3.若最后队列中有 n 个点，则存在拓扑序

```
#include<cstring>
#include<iostream>
#include<queue>

using namespace std;

const int N = 100000 + 10;
```

```cpp

int to[N], ne[N], head[N], deg[N], tot, cnt, n, m, a[N];

void add(int u, int v) {
    to[tot] = v, ne[tot] = head[u], head[u] = tot++, deg[v]++;//deg记录一个点的入度
}

bool topsort() {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (deg[i] == 0) q.push(i); //寻找一个入度为零的点
    }
    while (q.size()) {
        int t = q.front(); q.pop();
        a[++cnt] = t; //记录下拓扑序，cnt为拓扑序中有多少个点
        for (int i = head[t]; i != -1; i = ne[i]) {
            int j = to[i];
            if (--deg[j] == 0) q.push(j); //删除后入度为零，入队
        }
    }
    return cnt == n; //判断是否存在拓扑序
}

int main() {
    memset(head, -1, sizeof head);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int a, b;
        cin >> a >> b;
        add(a, b);
    }
    if (topsort()) {
        for (int i = 1; i <= n; i++)
            cout << a[i] << " ";
    }
    else cout << -1 << endl;
    return 0;
}
```

dijkstra

时间复杂度: $O(n^2 + m)$，适合用在稠密图

1.`dist`置成正无穷，令第一个点的距离为 0

2.遍历所有点，找一条最短的路

3.更新邻点到当前点的最短距离

```cpp
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 510;
```

```cpp
const int INF = 0x3f3f3f3f;

int n, m;
int g[N][N], dist[N];
bool st[N];

int dijkstra() {
    memset(dist, INF, sizeof dist);
    dist[1] = 0;
    for (int i = 1; i <= n; i++) {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;
        st[t] = true;
        for (int j = 1; j <= n; j++)
            dist[j] = min(dist[j], dist[t] + g[t][j]); //更新当前节点到其他点的距离
    }
    if (dist[n] == INF) return -1;
    else return dist[n];
}

int main() {
    memset(g, INF, sizeof g);
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        g[u][v] = min(g[u][v], w); //稠密图用邻接矩阵存储
    }
    int ans = dijkstra();
    cout << ans;
    return 0;
}
```

dijkstra(堆优化)

时间复杂度: $O(mlogn)$，用于稀疏图

1.使用优先队列(小根堆)

2.把 `dist` 置成正无穷，起点距离为 0，把起点放入堆中

3.每次用堆顶来更新距离

```cpp
#include<queue>
#include<iostream>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 100000 + 10, INF = 0x3f3f3f3f;

int to[N], ne[N], w[N], dist[N], head[N], tot, n, m;
bool st[N];
```

```cpp

void add(int u, int v, int c) {
    to[tot] = v, w[tot] = c, ne[tot] = head[u], head[u] = tot++;
}

int dijkstra() {
    memset(dist, 0x3f, sizeof dist);
    priority_queue<pair<int, int>> heap;
    dist[1] = 0;
    heap.push(make_pair(0, 1));
    while (heap.size()) {
        int t = heap.top().second; heap.pop();
        if (st[t]) continue;
        st[t] = true;
        for (int i = head[t]; i != -1; i = ne[i]) {
            int j = to[i], k = w[i];
            if (dist[j] > dist[t] + k) {
                dist[j] = dist[t] + k;
                heap.push(make_pair(-dist[j], j)); //取相反数，让大的小，小的大
            }
        }
    }
    if (dist[n] == INF) return -1; //判断从1能否走到n
    else return dist[n];
}

int main() {
    cin >> n >> m;
    memset(head, -1, sizeof head);
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    int ans = dijkstra();
    cout << ans;
    return 0;
}
```

Bellman-Ford

时间复杂度: $O(nm)$，可以求有边数限制的最短路，可以处理负权

1.把 `dist` 置成正无穷，起点距离为 0 ，循环 k 次 (k为边数限制)，做 k 次松弛操作

2.每次备份，用上一次的结果来更新，避免被当前最短路更新

```cpp
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 510, M = 10010;

struct Edge { int u, v, w; }edge[M];
```

```cpp
int dist[N], backup[N], n, m, k;
bool st[N];

int bellman_ford() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    for (int i = 1; i <= k; i++) {
        memcpy(backup, dist, sizeof dist);
        for (int j = 0; j < m; j++) {
            Edge a = edge[j];
            dist[a.v] = min(dist[a.v], backup[a.u] + a.w);
        }
    }
    return dist[n];
}

int main() {
    cin >> n >> m >> k;
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        edge[i] = { a,b,c };
    }
    int ans = bellman_ford();
    if (ans > 0x3f3f3f3f / 2) cout << "impossible"; //一个数加上负数会比原来小，所以大于一个很大的数就认为无法到达
    else cout << ans;
    return 0;
}
```

SPFA

时间复杂度:$O(kn)$，k 是一个很小的常数，~~spfa可被特殊图卡~~

1.把 `dist` 置成正无穷，让起点距离为 0，把起点放进队列里

2.循环整个队列，直到队列为空

3.每次取出对头，把对头元素的访问状态变成 `false`

4.遍历对头的邻点，进行松弛操作，如果邻点不在队列中，则将邻点放入队列中

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 100000 + 10;

int dist[N], to[N], ne[N], w[N], head[N], tot;
int n, m;
int q[N];
bool st[N];

void add(int u, int v, int c) {
    to[tot] = v, w[tot] = c, ne[tot] = head[u], head[u] = tot++;
```

```
}

int spfa() {
    memset(dist, 0x3f, sizeof dist);
    int hh = 0, tt = 1;
    dist[1] = 0, q[0] = 1;
    while (hh != tt) {
        int t = q[hh++];
        if (hh == N) hh = 0;
        st[t] = false;
        for (int i = head[t]; i != -1; i = ne[i]) {
            int j = to[i], k = w[i];
            if (dist[j] > dist[t] + k) {
                dist[j] = dist[t] + k;
                q[tt++] = j;
                if (tt == N) tt = 0;
                st[j] = true;
            }
        }
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    else return dist[n];
}

int main() {
    cin >> n >> m;
    memset(head, -1, sizeof head);
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    int ans = spfa();
    if (ans == -1) cout << "impossible";
    else cout << ans;
    return 0;
}
```

Floyd

时间复杂度: $O(n^3)$

状态转移方程: $g[i][j] = min(g[i][j], g[i][k] + g[k][j])$

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 300, INF = 0x3f3f3f3f;
int g[N][N], n, m, k;

void floyd() {
    for (int k = 1; k <= n; k++)
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++)
```

```
                g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
}

int main() {
    cin >> n >> m >> k;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (i == j) g[i][j] = 0;
            else g[i][j] = INF;
        }
    }
    for (int i = 1; i <= m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        g[a][b] = min(g[a][b], c);
    }
    floyd();
    while (k--) {
        int u, v;
        cin >> u >> v;
        if (g[u][v] > INF / 2) cout << "impossible" << endl; //无解，从u无法走到v
        else cout << g[u][v] << endl;
    }
    return 0;
}
```

传递闭包

状态转移方程: $d[i][j] |= d[i][k] \&\& d[k][j]$

```
// 不等式排序
#include<iostream>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 30;

bool g[N][N], d[N][N];
int n, m, t;
bool st[N];

void floyd() {
    memcpy(d, g, sizeof d);

    for (int k = 0; k < n; ++k)
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                d[i][j] |= d[i][k] && d[k][j];
}

int check() {
    for (int i = 0; i < n; ++i) if (d[i][i]) return 2;

```

```
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (i != j && (!d[i][j]) && (!d[j][i])) return 0;

    return 1;
}

char get_min() {
    for (int i = 0; i < n; ++i)
        if (!st[i]) {
            bool flag = 1;
            for (int j = 0; j < n; ++j)
                if (!st[j] && d[j][i]) {
                    flag = 0;
                    break;
                }

            if (flag) {
                st[i] = 1;
                return 'A' + i;
            }
        }
}

int main() {
    while (cin >> n >> m, n || m) {
        memset(g, 0, sizeof g);
        int type = 0;
        for (int i = 1; i <= m; ++i) {
            char str[5];
            cin >> str;
            int a = str[0] - 'A', b = str[2] - 'A';
            if (!type) {
                g[a][b] = 1;
                floyd();
                type = check();
                if (type) t = i;
            }

        }

        if (!type) puts("Sorted sequence cannot be determined.");
        else if (type == 2) cout << "Inconsistency found after " << t << "
relations." << endl;
        else {
            memset(st, 0, sizeof st);
            cout << "Sorted sequence determined after " << t << " relations: ";
            for (int i = 0; i < n; ++i) cout << get_min();
            cout << '.' << endl;
        }
    }
    return 0;
}
```

SPFA判负环

如果某个点被经过了 n 次，那么图中一定存在负环

tips: 如果使用队列会超时，可以换成栈

```cpp
#include<iostream>
#include<cstring>
#include<queue>

using namespace std;
const int N = 100000 + 10;

int dist[N], to[N], ne[N], w[N], head[N], tot, n, m, cnt[N];
bool st[N];

void add(int u, int v, int c) {
    to[tot] = v, w[tot] = c, ne[tot] = head[u], head[u] = tot++;
}

int spfa() {  //存在负环就返回true，不存在返回false
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    queue<int> q;
    q.push(1);
    st[1] = true;
    for (int i = 1; i <= n; i++) {
        st[i] = true;
        q.push(i);
    }
    while (q.size()) {
        int t = q.front(); q.pop();
        st[t] = false;
        for (int i = head[t]; i != -1; i = ne[i]) {
            int j = to[i], k = w[i];
            if (dist[j] > dist[t] + k) {
                dist[j] = dist[t] + k;
                cnt[j] = cnt[t] + 1;
                if (cnt[j] >= n) return true;
                if (!st[j]) {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }
    return false;
}

int main() {
    cin >> n >> m;
    memset(head, -1, sizeof head);
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        add(a, b, c);
    }
    if (spfa()) cout << "Yes";
```

```
    else cout << "No";
    return 0;
}
```

Prim

时间复杂度: $O(n^2 + m)$， 用于稠密图

1.把dist置成正无穷

2.循环所有点，做松弛操作，不管更新所有点到当前连通块(已经确定的最小生成树)的距离

```
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 510, INF = 0x3f3f3f3f;

int n, m;
int g[N][N];
int dist[N];
bool st[N];

int prim()
{
    memset(dist, 0x3f, sizeof dist);
    int res = 0;
    for (int i = 0; i < n; i++)
    {
        int t = -1;
        for (int j = 1; j <= n; j++)
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;
        if (i && dist[t] == INF) return INF;
        if (i) res += dist[t];
        st[t] = true;
        for (int j = 1; j <= n; j++) dist[j] = min(dist[j], g[t][j]);
    }
    return res;
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(g, 0x3f, sizeof g);
    while (m--)
    {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        g[a][b] = g[b][a] = min(g[a][b], c); //解决重边问题
    }
    int t = prim();
    if (t == INF) puts("impossible");
    else printf("%d\n", t);
```

```
    return 0;
}
```

Kruskal

时间复杂度: $O(mlogn)$ ，用于稀疏图

1.按边权排序

2.用并查集维护连通性，把不在集合中的点合并进来，形成一棵最小生成树

```
#include<iostream>
#include<algorithm>

using namespace std;
const int N = 1e6, M = 200010;
int fa[N], cnt, n, m, ans;

struct Edge {
    int u, v, w;
    bool operator <(const Edge& W)const {
        return w < W.w;
    }
}edge[M];

int find(int x) { //并查集核心操作
    return fa[x] == x ? fa[x] : fa[x] = find(fa[x]);
}

bool kruskal() {
    for (int i = 1; i <= n; i++) fa[i] = i;
    sort(edge, edge + m);
    for (int i = 0; i < m; i++) {
        int a = find(edge[i].u), b = find(edge[i].v), c = edge[i].w;
        if (a != b) {
            fa[a] = b;
            ans += c;
            cnt++;
        }
    }
    if (cnt < n - 1) return false; //无法连通
    else return true;
}

int main() {
    cin >> n >> m;
    for (int i = 0; i < m; i++) {
        int a, b, c;
        cin >> a >> b >> c;
        edge[i] = { a,b,c };
    }
    if (kruskal()) cout << ans;
    else cout << "impossible";
    return 0;
}
```

染色法判二分图

若某个点染色失败，即相邻的两个点被染了同一种颜色，那么该图不是二分图

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 100010, M = 200010;
int head[N], to[M], ne[M], tot, color[N];

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

bool dfs(int u, int c) {
    color[u] = c;
    for (int i = head[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!color[j]) {
            if (!dfs(j, 3 - c)) return false;
        }
        else if (color[j] == c) return false;
    }
    return true;
}

int main() {
    memset(head, -1, sizeof head);
    int n, m;
    scanf("%d%d", &n, &m);
    while (m--) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }
    bool flag = true;
    for (int i = 1; i <= n; i++) {
        if (!color[i]) {
            if (!dfs(i, 1)) {
                flag = false;
                break;
            }
        }
    }
    if (flag) puts("Yes");
    else puts("No");
    return 0;
}
```

匈牙利算法

解决二分图最大匹配问题

术语：

`匹配`： 一个匹配(独立边集)是一个边的集合，其中任意两条边都没有公共点

`最大匹配`： 一个图的所有匹配中，所含匹配边数最多的匹配，称为最大匹配

`完美匹配`： 若一个图的匹配中，所有顶点都是匹配点，则称该匹配为完美匹配

`交替路`： 从一个未匹配点出发，依次经过非匹配边，匹配边形成的路径称为交替路

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 510, M = 100010;

int head[N], to[M], ne[M], tot, match[N];
bool st[N];

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

bool find(int x) {
    for (int i = head[x]; ~i; i = ne[i]) {
        int j = to[i];
        if (!st[j]) {
            st[j] = true;
            if (match[j] == 0 || find(match[j])) {
                match[j] = x;
                return true;
            }
        }
    }
    return false;
}

int main() {
    memset(head, -1, sizeof head);
    int n1, n2, m;
    scanf("%d%d%d", &n1, &n2, &m);
    while (m--) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }
    int res = 0;
    for (int i = 1; i <= n1; i++) {
        memset(st, false, sizeof st);
        if (find(i)) res++;
    }
    printf("%d", res);
    return 0;
```

```
}
```

倍增求LCA

时间复杂度: 预处理$O(nlogn)$，询问$O(logn)$

设 $F[x,k]$ 表示 x 的 2^k 级祖先，$F[x,k]=F[F[x,k-1]][k-1]$

1.设 `d[x]` 表示 x 的深度。不妨设 $d[x]>=d[y]$

2.用二进制拆分思想，把 x 向上调到与 y 同一深度

3.若此时 $x=y$，则说明已经找到 `LCA`，`LCA` 就是 x，否则 x,y 一起继续往上跳，直到相遇

```
#include<iostream>
#include<cstring>
#include<algorithm>
#include<cstdio>
#include<cmath>
using namespace std;
const int N = 40010, M = N * 2;
int head[N], ne[M], to[M], tot;
int depth[N], fa[N][20];
int n, m, root, t;
int q[N];

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

void bfs() {
    memset(depth, 0x3f, sizeof depth);
    depth[0] = 0, depth[root] = 1;
    int hh = 0, tt = 0;
    q[0] = root;
    while (hh <= tt) {
        int x = q[hh++];
        for (int i = head[x]; ~i; i = ne[i]) {
            int y = to[i];
            if (depth[y] > depth[x] + 1) {
                depth[y] = depth[x] + 1;
                q[++tt] = y;
                fa[y][0] = x;
                for (int k = 1; k <= 15; k++)
                    fa[y][k] = fa[fa[y][k - 1]][k - 1];
            }
        }
    }
}

int lca(int x, int y) {
    if (depth[x] < depth[y]) swap(x, y);
    for (int k = t; k >= 0; k--) {
        if (depth[fa[x][k]] >= depth[y]) x = fa[x][k];
    }
    if (x == y) return x;
    for (int k = t; k >= 0; k--) {
```

```
        if (fa[x][k] != fa[y][k])
            x = fa[x][k], y = fa[y][k];
    }
    return fa[x][0];
}

int main() {
    memset(head, -1, sizeof head);
    t = 15;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        if (b == -1) root = a;
        else add(a, b), add(b, a);
    }
    bfs();
    scanf("%d", &m);
    while (m--)
    {
        int a, b;
        scanf("%d%d", &a, &b);
        int p = lca(a, b);
        if (p == a) puts("1");
        else if (p == b) puts("2");
        else puts("0");
    }
    return 0;
}
```

tarjan求LCA

离线算法，时间复杂度: $O(m+n)$

在深度优先遍历的任意时刻，树中节点分为三类:

1. 已经访问完毕并且回溯的节点，标记为 2
2. 已经开始递归，但尚未回溯的节点。这些节点就是当前正在访问的节点 x 以及 x 的祖先。标记为 1
3. 没有访问的节点。没有标记

考虑使用 `并查集` 优化，当一个节点被标记 2 时，合并到它的父节点(合并时它的父节点标记一定为1，且单独构成一个集合)

```
// 求树上任意两点距离
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<vector>

using namespace std;

const int N = 10010, M = N * 2;

int head[N], to[M], ne[M], w[M], tot;
int n, m;
```

```cpp
int fa[N];
int dist[N], vis[N];
int res[M];
vector<pair<int, int> > query[N];

void add(int u, int v, int c) {
    to[tot] = v, w[tot] = c, ne[tot] = head[u], head[u] = tot++;
}

int find(int x) {
    return fa[x] == x ? fa[x] : fa[x] = find(fa[x]);
}

void dfs(int u, int fa) {
    for (int i = head[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (j == fa) continue;
        dist[j] = dist[u] + w[i];
        dfs(j, u);
    }
}

void tarjan(int u) {
    vis[u] = 1;
    for (int i = head[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!vis[j]) {
            tarjan(j);
            fa[j] = u;
        }
    }

    for (pair<int, int> item : query[u]) {
        int y = item.first, id = item.second;
        if (vis[y] == 2) {
            int anc = find(y);
            res[id] = dist[u] + dist[y] - 2 * dist[anc];
        }
    }
    vis[u] = 2;
}

int main() {
    memset(head, -1, sizeof head);
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n - 1; i++) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c), add(b, a, c);
    }
    for (int i = 0; i < m; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        if (a != b) {
            query[a].push_back({ b,i });
            query[b].push_back({ a,i });
        }
    }
```

```
    for (int i = 1; i <= n; i++) fa[i] = i;

    dfs(1, -1); //预处理距离
    tarjan(1);

    for (int i = 0; i < m; i++) printf("%d\n", res[i]);
    return 0;
}
```

有向图强联通分量

强连通图： 一张有向图 G .若对于图中任意两个节点 x,y ，既存在从 x 到 y 的路径，也存在从 y 到 x 的路径，则称该图是强连通图.

有向图的极大连通子图被称为 `强联通分量`，记为 SCC

`tarjan`算法计算"追溯值"：

1.当前节点 x 第一次被访问时，把 x 加入栈中，初始化`low[x]=dfn[x]`

2.扫描 x 的出边 (x,y)

(1) 若 y 没被访问过，则说明 (x,y) 是树枝边，递归访问 y ，从 y 回溯之后，令 `low[u]=min(low[x],low[y])`

(2) 若 y 被访问过且 y 在栈中，则令 `low[u]=min(low[x],dfn[y])`

3.若 y 回溯之前，判断是否有 `low[x]=dfn[x]`，若有则不断从栈中淡出结点，直到 x 出栈

强联通分量判定法则：在追溯值得计算过程中，若从 x 回溯前，有 `low[x]=dfn[x]` 成立，则栈中从 x 到栈顶得所有节点构成一个强联通分量

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

using namespace std;

const int N = 10010, M = 50010;

int n, m;
int h[N], to[M], ne[M], idx;
int dfn[N], low[N], cnt;
int stk[N], top;
bool ins[N];
int id[N], scc_cnt, sz[N];
int dout[N];

inline void add(int u, int v) {
    to[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u) {
    dfn[u] = low[u] = ++cnt;
    stk[++top] = u, ins[u] = true;
```

```
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!dfn[j]) {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (ins[j]) low[u] = min(low[u], dfn[j]);
    }

    if (dfn[u] == low[u]) {
        int y;
        ++scc_cnt;
        do {
            y = stk[top--];
            ins[y] = false;
            id[y] = scc_cnt;
            sz[scc_cnt]++;
        } while (y != u);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m--) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
    }

    for (int i = 1; i <= n; ++i)
        if (!dfn[i])
            tarjan(i);

    for (int i = 1; i <= n; ++i)
        for (int k = h[i]; ~k; k = ne[k]) {
            int j = to[k];
            int a = id[i], b = id[j];
            if (a != b) dout[a]++;
        }

    int zeros = 0, sum = 0;
    for (int i = 1; i <= scc_cnt; ++i)
        if (!dout[i]) {
            zeros++;
            sum += sz[i];
            if (zeros > 1) {
                sum = 0;
                break;
            }
        }
    printf("%d", sum);
    return 0;
}
```

无向图的双连通分量

边的双连通分量($e-dcc$)： 极大的不包含桥的连通块

点的双连通分量($v-dcc$)： 极大的不包含割点的连通块

`割点`： 给定一张无向连通图 G .若对于 $x \in V$ ，从图中删去节点 x 以及所有与 x 关联的边后，G 分裂成两个或两个以上不相连的子图，则称 x 为 G 的割点

`桥`： 给定一张无向连通图 G .若对于 $e \in E$ ，从图中删去边 e 之后， G 分裂成两个不相连的子图，则称 e 为 G 的桥或割边

割边判定法则：无向边 (x,y) 是桥，当且仅当搜索树中存在 x 的一个子节点 y ，满足 `dfn[x]<low[y]` .

割点判定法则： 若 x 不是搜索树的父节点，则 x 是割点当且仅当搜索树上存在 x 的一个子节点 y ，满足 `dfn[x] <= low[y]`

```cpp
// e-dcc
// Acwing395 冗余路径
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 5010, M = 20010;

int n, m;
int h[N], to[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int stk[N], top;
int id[N], dcc_cnt;
bool bridge[N];
int d[N];

inline void add(int u, int v) {
    to[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u, int from) {
    dfn[u] = low[u] = ++timestamp;
    stk[++top] = u;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!dfn[j]) {
            tarjan(j, i);
            low[u] = min(low[u], low[j]);
            if (dfn[u] < low[j])
                bridge[i] = bridge[i ^ 1] = true;
        }
        else if (i != (from ^ 1)) low[u] = min(low[u], dfn[j]);
    }

    if (low[u] == dfn[u]) {
        ++dcc_cnt;
        int y;
        do {
```

```
            y = stk[top--];
            id[y] = dcc_cnt;
        } while (y != u);
    }
}

int main() {
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m--) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a);
    }

    tarjan(1, -1);

    for (int i = 0; i < idx; ++i)
        if (bridge[i])
            d[id[to[i]]]++;

    int cnt = 0;
    for (int i = 1; i <= dcc_cnt; ++i)
        if (d[i] == 1)
            cnt++;
    printf("%d", (cnt + 1) / 2);
    return 0;
}
```

```
// v-dcc
// Acwing1183 电力
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 10010, M = 30010;

int n, m;
int h[N], to[M], ne[M], idx;
int low[N], dfn[N], timestamp;
int root, ans;

inline void add(int u, int v) {
    to[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void tarjan(int u) {
    low[u] = dfn[u] = ++timestamp;
    int cnt = 0;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!dfn[j]) {
```

```
            tarjan(j);
            low[u] = min(low[u], low[j]);
            if (low[j] >= dfn[u]) cnt++;
        }
        else low[u] = min(low[u], dfn[j]);
    }
    if (u != root) cnt++;
    ans = max(ans, cnt);
}

int main() {
    while (scanf("%d%d", &n, &m), n || m) {
        memset(h, -1, sizeof h);
        memset(dfn, 0, sizeof dfn);
        idx = timestamp = 0;

        while (m--) {
            int a, b;
            scanf("%d%d", &a, &b);
            add(a, b), add(b, a);
        }

        ans = 0;
        int cnt = 0;
        for (root = 0; root < n; ++root)
            if (!dfn[root]) {
                cnt++;
                tarjan(root);
            }
        printf("%d\n", ans + cnt - 1);
    }
    return 0;
}
```

欧拉路问题

欧拉路： 给定一张无向图，若存在一条从节点 S 到节点 T 的路径，恰好不重不漏的经过每一条边一次，则称该路径为欧拉路

欧拉回路： 若存在一条从节点 S 出发，恰好不重不漏地经过每条边一次，最终回到 S，则称该回路为欧拉回路

欧拉图的判定：一张无向图为欧拉图，当且仅当无向图连通，并且每个点的度数都是偶数

```
// 欧拉回路
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 100010, M = 400010;

int t, n, m;
int h[N], to[M], ne[M], idx;
```

```cpp
bool used[M];
int ans[M / 2], cnt;
int din[N], dout[N];

inline void add(int u, int v) {
    to[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void dfs(int u) {
    for (int& i = h[u]; ~i;) {
        if (used[i]) {
            i = ne[i];
            continue;
        }

        used[i] = true;
        if (t == 1) used[i ^ 1] = true;

        int res;
        if (t == 1) {
            res = i / 2 + 1;
            if (i & 1) res = -res;
        }
        else res = i + 1;

        int j = to[i];
        i = ne[i];
        dfs(j);

        ans[++cnt] = res;
    }
}

int main() {
    scanf("%d%d%d", &t, &n, &m);
    memset(h, -1, sizeof h);

    for (int i = 0; i < m; ++i) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(a, b);
        if (t == 1) add(b, a);
        din[b]++, dout[a]++;
    }

    if (t == 1) {
        for (int i = 1; i <= n; ++i)
            if (din[i] + dout[i] & 1) {
                puts("NO");
                return 0;
            }
    }
    else {
        for (int i = 1; i <= n; ++i)
            if (din[i] != dout[i]) {
                puts("NO");
                return 0;
            }
```

```
    }

    for (int i = 1; i <= n; ++i)
        if (h[i] != -1) {
            dfs(i);
            break;
        }

    if (cnt < m) {
        puts("NO");
        return 0;
    }

    puts("YES");
    for (int i = cnt; i; --i) printf("%d ", ans[i]);
    return 0;
}
```

最大流

一个流是可行流满足以下两个条件:

1.容量限制，$0 \le f(u,v) \le c(u,v)$

2.流量守恒，$\forall x \in V/\{x,t\}$, $\sum\limits_{(v,x)\in E} f(v,x) = \sum\limits_{x,v)\in E} f(x,v)$

残留网络 f' + 原图的可行流 f = 原图的另一个可行流

(1) $|f'+f| = |f'|+|f|$

(2)$|f'|$ 可能是负数

割：流网络 $G=(V,E)$ 中的一个割 (S,T) 将节点集合 V 划分为 S 和 $T=V-S$ 两个集合，使得 $s \in S, t \in T$

割的容量：$c(S,T) = \sum\limits_{u \int S} \sum\limits_{v \in T} c(u,v)$

割的流量：$f(S,T) = \sum\limits_{u\in S}\sum\limits_{v\in T} f(u,v) - \sum\limits_{u\in T}\sum\limits_{v\in S} f(u,v)$， $f(S,T) = \sum\limits_{u\in S}\sum\limits_{v\in T} f(u,v) - \sum\limits_{u\in S}\sum\limits_{v\in T} f(v,u)$

最大流最小割定理： 设 f 为流网络 $G=(V,E)$ 中的一个流，该流网络的源点为 s， 汇点为 t， 则以下条件是等价的:

1.f 是 G 的一个最大流

2.残留网络 G_f 不包含任何增广路径

3.$|f| = c(S,T)$，其中 (S,T) 是流网络的某个割

Edmond-Karp动能算法(EK算法)

时间复杂度： $O(nm^2)$

使用 `bfs` 寻找增广路，然后对其增广，计算路径经过各边的剩余流量的最小值 $minf$， 则网络的流量增加 $minf$

```
#include<iostream>
```

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

using namespace std;

const int N = 1010, M = 20010, INF = 1e9;

int n, m, S, T;
int h[N], to[M], c[M], ne[M], tot;
int q[N], d[N], pre[N];
bool vis[N];

void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = h[u], h[u] = tot++; //建正向边
    to[tot] = u, c[tot] = 0, ne[tot] = h[v], h[v] = tot++; //建反向边
}

bool bfs() {
    memset(vis, false, sizeof vis);
    int head = -1, tail = 0;
    q[++head] = S, vis[S] = true, d[S] = INF;
    while (head <= tail) {
        int t = q[head++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (!vis[ver] && c[i]) { //容量不为0,将其展开
                vis[ver] = true;
                d[ver] = min(d[t], c[i]);
                pre[ver] = i;
                if (ver == T) return true;
                q[++tail] = ver;
            }
        }
    }
    return false;
}

int EK() { //Edmonds-Karp
    int res = 0;
    while (bfs()) {
        res += d[T];
        for (int i = T; i != S; i = to[pre[i] ^ 1])
            c[pre[i]] -= d[T], c[pre[i] ^ 1] += d[T];
    }
    return res;
}
int main() {
    memset(h, -1, sizeof h);
    scanf("%d%d%d%d", &n, &m, &S, &T);
    while (m--) {
        int u, v, w;
        scanf("%d%d%d", &u, &v, &w);
        add(u, v, w);
    }
    printf("%d", EK());
    return 0;
}
```

Dinic算法

时间复杂度： 玄学的 $O(n^2m)$

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;
const int N = 100010, M = 200010, INF = 0x3f3f3f3f;

int n, m, S, T;
int h[N], ne[M], to[M], c[M], f[M], tot;
bool vis[N];
int d[N], cur[N], q[N];

inline void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = h[v], h[v] = tot++;
}

bool bfs() { // 分层
    memset(d, -1, sizeof d);
    int head = 0, tail = 0;
    q[0] = S, d[S] = 0, cur[S] = h[S];
    while (head <= tail) {
        int t = q[head++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                if (ver == T) return true;
                q[++tail] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) { // 找增广路
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i; // 当前弧优化
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}
```

```

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d%d%d%d", &n, &m, &S, &T);
    while (m--) {
        int a, b, w;
        scanf("%d%d%d", &a, &b, &w);
        add(a, b, w);
    }
    printf("%d", dinic());
    return 0;
}
```

最大流二分图匹配

二分图的最大流即为二分图最大匹配

```
// dinic二分图匹配
// 从源点向1~m连一条长度为1的边
// 从m+1~n向汇点连一条长度为1的边
// 然后读入数据,从u到v连一条长度为1的边
// 求最大流即为ans
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int N = 110, M = 5210, INF = 1e8;
int head[N], to[M], ne[M], c[M], tot;
int q[N], d[N], cur[N];
int n, m, S, T;

static char buf[1 << 20], * p1, * p2;
size_t fread(void* buffer, size_t size, size_t count, FILE* stream);
#define gc() (p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<20,stdin),p1==p2)?EOF:*p1++)
template <typename T>
inline T read(T& x) {
    x = 0; T f = 1; char c = gc();
    while (!isdigit(c)) { if (c == '-')f = -1; c = gc(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = gc(); }
    x *= f;
}

inline void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = head[u], head[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = head[v], head[v] = tot++;
}

bool bfs() {
```

```
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = head[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = head[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = head[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;

}

int main() {
    read(m), read(n);
    S = 0, T = n + 1;
    memset(head, -1, sizeof head);
    for (int i = 1; i <= m; i++) add(S, i, 1);
    for (int i = m + 1; i <= n; i++) add(i, T, 1);
    int a, b;
    while (read(a), read(b), a != -1) {
        add(a, b, 1);
    }
    printf("%d\n", dinic());
    for (int i = 0; i < tot; i += 2) {
        if (to[i] > m && to[i] <= n && !c[i])
            printf("%d %d\n", to[i ^ 1], to[i]);
    }
    return 0;
}
```

无源汇上下界可行流(循环流)

给定一个网络，求一个流满足： 每条边 i 流量在 $[low(i), up(i)]$ 之间，每个点 u 都要满足流量守恒

思想： 把一个不满足流量守恒的初始流调整成满足流量守恒的流

设 $a(u) = \sum_{to_{i=u}} f(i) - \sum_{from_{i=u}} f(i)$ ，即点 u 的流入量-留出量

当 $-a(u) < 0 (a(u) > 0)$ 时，需要让 u 的流入量增加 $a(u)$ ，通过建立超级源点 S'，并增加 $S' -> u$ ，容量为 $a(u)$ 的边

当 $-a(u) > 0 (a(u) < 0)$ 时，需要让 u 的流出量增加 $a(u)$ ，通过建立超级汇点 T'，并增加 $T' -> u$ ，容量为 $-a(u)$ 的边

原图中的边 i ,$up(i) - low(i)$ 的容量也要加上

```cpp
// 无源汇上下界可行可行流
// 0<=可行流-下界可行流<=上界可行流-下界可行流

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 210, M = (10200 + N) * 2, INF = 1e8;

int head[N], f[M], ne[M], to[M], l[M], idx;
int n, m, S, T;
int q[N], d[N], cur[N], A[N];

static char buf[1 << 25], * p1, * p2;
size_t fread(void* buffer, size_t size, size_t count, FILE* stream);
#define gc() (p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<20,stdin),p1==p2)?EOF:*p1++)

void read(int& x) {
    x = 0; int f = 1; char c = gc();
    while (!isdigit(c)) { if (f == '-')f = -1; c = gc(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = gc(); }
    x *= f;
}

void add(int u, int v, int c, int d) {
    to[idx] = v, f[idx] = d - c, l[idx] = c, ne[idx] = head[u], head[u] = idx++;
    to[idx] = u, f[idx] = 0, ne[idx] = head[v], head[v] = idx++;
}

bool bfs() {
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = head[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = head[t]; ~i; i = ne[i]) {
            int ver = to[i];
```

```
            if (d[ver] == -1 && f[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = head[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && f[i]) {
            int t = find(ver, min(f[i], limit - flow));
            if (!t) d[ver] = -1;
            f[i] -= t, f[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    memset(head, -1, sizeof head);
    read(n), read(m);
    S = 0, T = n + 1;
    for (int i = 0; i < m; i++) {
        int a, b, c, d;
        read(a), read(b), read(c), read(d);
        add(a, b, c, d);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) add(S, i, 0, A[i]), tot += A[i];
        else if (A[i] < 0) add(i, T, 0, -A[i]);
    }
    if (dinic() != tot) printf("NO");
    else {
        printf("YES\n");
        for (int i = 0; i < m * 2; i += 2) { // 输出原网络的可行流
            printf("%d\n", f[i ^ 1] + l[i]);
        }
    }
    return 0;
}
```

有源汇上下界最大流

在有源汇的网络流图中，源点 rS 、汇点 rT 不满足流量守恒，无法直接跑循环流。

但只要加一条 $rT \to rS$,下界为0上界为正无穷的边，就能保证流量守恒。跑一下超源SS到TT的最大流，得到可行流，可行流中 $rT \to rS$ 的流量即为 $rS \to rT$ 的真正流量，记为 $f1$.

删掉 $rT \to rS$ 的边，再跑一次 $rS \to rT$ 的最大流 $f2$,答案即为 $f1 + f2$

```cpp
// 有源汇上下界最大流
#include<cstdio>
#include<algorithm>
#include<iostream>
#include<cstring>

using namespace std;

const int N = 210, M = (N + 10000) * 2, INF = 0x3f3f3f3f;

int head[N], to[M], ne[M], c[M], idx;
int q[N], d[N], cur[N], A[N];
int n, m, S, T, s, t;

static char buf[1 << 25], * p1, * p2;
size_t fread(size_t* buffer, size_t size, size_t sount, FILE* stream);
#define gc() (p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<20,stdin),p1==p2)?EOF:*p1++)
template <typename T> inline void read(T& x) {
    x = 0; T f = 1; char c = gc();
    while (!isdigit(c)) { if (c == '-')f = -1; c = gc(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = gc(); }
    x *= f;
}

inline void add(int u, int v, int w) {
    to[idx] = v, c[idx] = w, ne[idx] = head[u], head[u] = idx++;
    to[idx] = u, c[idx] = 0, ne[idx] = head[v], head[v] = idx++;
}

bool bfs() {
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = head[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = head[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = head[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}
```

```

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    memset(head, -1, sizeof head);
    read(n), read(m), read(s), read(t);
    S = 0, T = n + 1;
    while (m--) {
        int a, b, c, d;
        read(a), read(b), read(c), read(d);
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) add(S, i, A[i]), tot += A[i];
        else if (A[i] < 0) add(i, T, -A[i]);
    }
    add(t, s, INF);
    if (dinic() < tot) printf("No Solution");
    else {
        int res = c[idx - 1];
        S = s, T = t;
        c[idx - 1] = c[idx - 2] = 0; // 删去虚拟边
        printf("%d", res + dinic());
    }
    return 0;
}
```

有源汇上下界最小流

用 $f1$ 减去 rT 流向 rS 的最大流 $f3$ 即为答案

```
// 有源汇上下界最小流
// 在有源汇上下界最大流中求的是s->t的最大流,那么要求最小只要求t->s的最大即可
#include<iostream>
```

```cpp
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int N = 50010, M = (125005 + N) * 2, INF = 0x3f3f3f3f;
int head[N], ne[M], to[M], c[M], idx;
int q[N], d[N], cur[N], A[N];
int n, m, S, T, s, t;

static char buf[1 << 25], * p1, * p2;
size_t fread(void* buffer, size_t size, size_t count, FILE* stream);
#define gc() (p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<20,stdin),p1==p2)?EOF:*p1++)
template <typename T> inline void read(T& x) {
    x = 0; T f = 1; char c = gc();
    while (!isdigit(c)) { if (c == '-')f = -1; c = gc(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = gc(); }
    x *= f;
}

inline void add(int u, int v, int w) {
    to[idx] = v, c[idx] = w, ne[idx] = head[u], head[u] = idx++;
    to[idx] = u, c[idx] = 0, ne[idx] = head[v], head[v] = idx++;
}

bool bfs() {
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = head[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = head[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = head[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

```

```
int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    memset(head, -1, sizeof head);
    read(n), read(m), read(s), read(t);
    S = 0, T = n + 1;
    while (m--) {
        int a, b, c, d;
        read(a), read(b), read(c), read(d);
        add(a, b, d - c);
        A[a] -= c, A[b] += c;
    }
    int tot = 0;
    for (int i = 1; i <= n; i++) {
        if (A[i] > 0) add(S, i, A[i]), tot += A[i];
        else if (A[i] < 0) add(i, T, -A[i]);
    }
    add(t, s, INF);
    if (dinic() < tot) printf("No Solution");
    else {
        int res = c[idx - 1];
        S = t, T = s;
        c[idx - 1] = c[idx - 2] = 0;
        printf("%d", res - dinic());
    }
    return 0;
}
```

多源汇最大流

建立一个超级源点 S，对于所有源点 s_i，建立一条容量为 $+\infty$ 的边，即 $c(S, s_i) = +\infty$

建立一个超级汇点 T，对于所有汇点 t_i，建立一条容量为 $+\infty$ 的边，即 $c(t_i, T) = +\infty$

原网络的最大流等于 $S->T$ 的最大流

```
// 从超级源点向每个源点连一条容量为INF的边
// 从每个汇点向超级汇点连一条容量为INF的边
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int N = 10005, M = (100000 + N) * 2, INF = 0x3f3f3f3f;
int head[N], to[M], ne[M], c[M], tot;
int q[N], cur[N], d[N];
int n, m, S, T;

static char buf[1 << 25], * p1, * p2;
size_t fread(void* buffer, size_t size, size_t count, FILE* stream);
#define gc() (p1==p2&&(p2=(p1=buf)+fread(buf,1,1<<20,stdin),p1==p2)?EOF:*p1++)
```

```cpp
template <typename T> inline void read(T& x) {
    x = 0; T f = 1; char c = gc();
    while (!isdigit(c)) { if (c == '-')f = -1; c = gc(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = gc(); }
    x *= f;
}

void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = head[u], head[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = head[v], head[v] = tot++;
}

bool bfs() {
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = head[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = head[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = head[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    memset(head, -1, sizeof head);
    int sc, tc;
    read(n), read(m), read(sc), read(tc);
    S = 0, T = n + 1;
    while (sc--) {
```

```
        int s;
        read(s);
        add(S, s, INF);
    }
    while (tc--) {
        int t;
        read(t);
        add(t, T, INF);
    }
    while (m--) {
        int a, b, w;
        read(a), read(b), read(w);
        add(a, b, w);
    }
    printf("%d", dinic());
    return 0;
}
```

最小割

由最大流最小割定理可得，最小割即最大流

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

using namespace std;
const int N = 10010, M = 200010, INF = 0x3f3f3f3f;

int n, m, S, T;
int h[N], to[M], c[M], ne[M], tot;
int q[N], d[N], cur[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

inline void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = h[v], h[v] = tot++;
}

bool bfs() {
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = h[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
```

```
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    read(n), read(m), read(S), read(T);
    memset(h, -1, sizeof h);
    while (m--) {
        int a, b, w;
        read(a), read(b), read(w);
        add(a, b, w);
    }
    printf("%d", dinic());
    return 0;
}
```

最大权闭合图

最大权闭合图时一个点权之和最大的闭合图，即最大化 $\sum_{v\in V} w_v$

构造：建立源点 S 和汇点 T，从源点向点连一条容量为 $|w|$ 的边，点与点之间的边容量为 $+\infty$

```
// NOI2006 最大获利
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

```

```cpp
using namespace std;

const int N = 55010, M = (50000 * 3 + 5000) * 2 + 10, INF = 0x3f3f3f3f;

int n, m, S, T;
int h[N], c[M], to[M], ne[M], tot;
int q[N], d[N], cur[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

inline void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = h[v], h[v] = tot++;
}

bool bfs() {
    memset(d, -1, sizeof d);
    int hh = 0, tt = 0;
    q[0] = S, d[S] = 0, cur[S] = h[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
```

```cpp
}

int main() {
    memset(h, -1, sizeof h);
    read(n), read(m);
    S = 0, T = n + m + 1;
    for (int i = 1; i <= n; ++i) {
        int p;
        read(p);
        add(m + i, T, p);
    }

    int tot = 0;
    for (int i = 1; i <= m; ++i) {
        int a, b, c;
        read(a), read(b), read(c);
        add(S, i, c);
        add(i, m + a, INF);
        add(i, m + b, INF);
        tot += c;
    }

    printf("%d", tot - dinic());
    return 0;
}
```

最大密度子图

`密度`： 一个无向图 $G=(V,E)$ 的密度 D 为该图的边数 $|E|$ 与该图的点数 $|V|$ 的比值，即 $D=\frac{|E|}{|V|}$

`最大密度子图`： 给定一个无向图 $G=(V,E)$，具有最大密度的子图 $G'=(V',E')$ 称为最大密度子图，即 $D_{max}=\frac{|E'|}{|V'|}$

使用 `01分数规划` 求解

最小点权覆盖集

`点覆盖集`： 点覆盖集一个无向图 G 的一个点集，使得该图中所有边都至少由一个端点在该集合内

`最小点覆盖集`： 点数最少的点覆盖集

`最小点权覆盖集`： 带点权 无向图 G 中，点权之和最小的点覆盖集

```cpp
// 有向图破坏
// 拆点 最小点权覆盖集
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 210, M = 5210 * 2, INF = 0x3f3f3f3f;

```

```cpp
int n, m, S, T;
int h[N], to[M], c[M], ne[M], tot;
int q[N], d[N], cur[N];
bool st[N];

inline void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = h[v], h[v] = tot++;
}

bool bfs() {
    memset(d, -1, sizeof d);
    int hh = 0, tt = 0;
    q[0] = S, d[S] = 0, cur[S] = h[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(c[i], limit - flow));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

void dfs(int u) {
    st[u] = true;
    for (int i = h[u]; ~i; i = ne[i])
        if (c[i] && !st[to[i]])
            dfs(to[i]);
}

int main() {
```

```
    memset(h, -1, sizeof h);
    scanf("%d%d", &n, &m);
    S = 0, T = 2 * n + 1;
    for (int i = 1; i <= n; ++i) {
        int w;
        scanf("%d", &w);
        add(S, i, w);
    }
    for (int i = 1; i <= n; ++i) {
        int w;
        scanf("%d", &w);
        add(n + i, T, w);
    }
    while (m--) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(b, n + a, INF);
    }

    printf("%d\n", dinic());
    dfs(S);

    int cnt = 0;
    for (int i = 0; i < tot; i += 2) {
        int a = to[i ^ 1], b = to[i];
        if (st[a] && !st[b]) cnt++;
    }

    printf("%d\n", cnt);

    for (int i = 0; i < tot; i += 2) {
        int a = to[i ^ 1], b = to[i];
        if (st[a] && !st[b])
            if (a == S) printf("%d +\n", b);
    }
    for (int i = 2; i < tot; i += 2) {
        int a = to[i ^ 1], b = to[i];
        if (st[a] && !st[b])
            if (b == T) printf("%d -\n", a - n);
    }
    return 0;
}
```

最大点权独立集

点独立集： 点独立集是一个无向图 G 的一个点集，使得任意两个在该集合中的点在原图中都不相邻

最大点独立集： 在无向图 G 中，点数最多的点独立集

最大点权独立集： 在带点权无向图 G 中，点权之和最大的点独立集

覆盖集与独立集互补定理： 若 $\overline{V'}$ 为不含孤立点的任意图的一个点覆盖集当且仅当 V' 是该图的一个点独立集

```
// 王者之剑
// 拆点 构造二分图
```

```cpp
// 求二分图最大权值和的点独立集
// ans=sum-最小点权覆盖集
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 10010, M = 60010, INF = 0x3f3f3f3f;

int n, m, S, T;
int h[N], to[M], ne[M], c[M], tot;
int q[N], d[N], cur[N];
int dx[] = { 0,1,0,-1 }, dy[] = { -1,0,1,0 };

inline int get(int x, int y) {
    return (x - 1) * m + y;
}

inline void add(int u, int v, int w) {
    to[tot] = v, c[tot] = w, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, c[tot] = 0, ne[tot] = h[v], h[v] = tot++;
}

bool bfs() {
    int hh = 0, tt = 0;
    memset(d, -1, sizeof d);
    q[0] = S, d[S] = 0, cur[S] = h[S];
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (d[ver] == -1 && c[i]) {
                d[ver] = d[t] + 1;
                cur[ver] = h[ver];
                if (ver == T) return true;
                q[++tt] = ver;
            }
        }
    }
    return false;
}

int find(int u, int limit) {
    if (u == T) return limit;
    int flow = 0;
    for (int i = cur[u]; ~i && flow < limit; i = ne[i]) {
        cur[u] = i;
        int ver = to[i];
        if (d[ver] == d[u] + 1 && c[i]) {
            int t = find(ver, min(limit - flow, c[i]));
            if (!t) d[ver] = -1;
            c[i] -= t, c[i ^ 1] += t, flow += t;
        }
    }
    return flow;
}
```

```

int dinic() {
    int res = 0, flow;
    while (bfs()) while (flow = find(S, INF)) res += flow;
    return res;
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d%d", &n, &m);
    S = 0, T = n * m + 1;

    int tot = 0;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            int w;
            scanf("%d", &w);
            if (i + j & 1) {
                add(S, get(i, j), w);
                for (int k = 0; k < 4; ++k) {
                    int x = i + dx[k], y = j + dy[k];
                    if (x >= 1 && x <= n && y >= 1 && y <= m)
                        add(get(i, j), get(x, y), INF);
                }
            }
            else
                add(get(i, j), T, w);
            tot += w;
        }

    printf("%d", tot - dinic());
    return 0;
}
```

费用流

EK+SPFA求费用流

```
// 最小费用流
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 5010, M = 100010, INF = 0x3f3f3f3f;

int n, m, S, T;
int h[N], ne[M], to[M], c[M], w[M], tot;
int q[N], d[N], pre[N], incf[N];
bool vis[N];

inline void add(int u, int v, int x, int y) {
    to[tot] = v, c[tot] = x, w[tot] = y, ne[tot] = h[u], h[u] = tot++;
```

```
    to[tot] = u, c[tot] = 0, w[tot] = -y, ne[tot] = h[v], h[v] = tot++;
}

bool spfa() {
    int hh = 0, tt = 1;
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    q[0] = S, d[S] = 0, incf[S] = INF;
    while (hh != tt) {
        int t = q[hh++];
        if (hh == N) hh = 0;
        vis[t] = false;

        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (c[i] && d[ver] > d[t] + w[i]) {
                d[ver] = d[t] + w[i];
                pre[ver] = i;
                incf[ver] = min(incf[t], c[i]);
                if (!vis[ver]) {
                    q[tt++] = ver;
                    if (tt == N) tt = 0;
                    vis[ver] = true;
                }
            }
        }
    }
    return incf[T] > 0;
}

void EK(int& flow, int& cost) {
    flow = cost = 0;
    while (spfa()) {
        int t = incf[T];
        flow += t, cost += t * d[T];
        for (int i = T; i != S; i = to[pre[i] ^ 1]) {
            c[pre[i]] -= t;
            c[pre[i] ^ 1] += t;
        }
    }
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d%d%d%d", &n, &m, &S, &T);
    while (m--) {
        int a, b, c, d;
        scanf("%d%d%d%d", &a, &b, &c, &d);
        add(a, b, c, d);
    }

    int flow, cost;
    EK(flow, cost);

    printf("%d %d", flow, cost);
    return 0;
}
```

二分图最优匹配

用费用流解决

```cpp
// 网络流24题 分配问题
// 二分图最优匹配
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 110, M = 5210, INF = 0x3f3f3f3f;

int h[N], f[M], w[M], to[M], ne[M], tot;
int q[N], d[N], pre[N], incf[N];
bool vis[N];
int n, S, T;

inline void add(int u, int v, int c, int d) {
    to[tot] = v, f[tot] = c, w[tot] = d, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, f[tot] = 0, w[tot] = -d, ne[tot] = h[v], h[v] = tot++;
}

bool spfa() {
    int hh = 0, tt = 1;
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    q[0] = S, d[S] = 0, incf[S] = INF;
    while (hh != tt) {
        int t = q[hh++];
        if (hh == N) hh = 0;
        vis[t] = false;
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (f[i] && d[ver] > d[t] + w[i]) {
                d[ver] = d[t] + w[i];
                pre[ver] = i;
                incf[ver] = min(incf[t], f[i]);
                if (!vis[ver]) {
                    q[tt++] = ver;
                    if (tt == N) tt = 0;
                    vis[ver] = true;
                }
            }
        }
    }
    return incf[T] > 0;
}

int EK() {
    int cost = 0;
    while (spfa()) {
        int t = incf[T];
        cost += t * d[T];
```

```
        for (int i = T; i != S; i = to[pre[i] ^ 1]) {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
    return cost;
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d", &n);
    S = 0, T = 2 * n + 1;

    for (int i = 1; i <= n; ++i) {
        add(S, i, 1, 0);
        add(n + i, T, 1, 0);
    }
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j) {
            int c;
            scanf("%d", &c);
            add(i, n + j, 1, c);
        }
    printf("%d\n", EK());

    for (int i = 0; i < tot; i += 2) {
        f[i] += f[i ^ 1], f[i ^ 1] = 0;
        w[i] = -w[i], w[i ^ 1] = -w[i ^ 1];
    }

    printf("%d", -EK());
    return 0;
}
```

最大权不相交路径

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

using namespace std;

const int N = 1200, M = 4000, INF = 0x3f3f3f3f;

int m, n, S, T;
int h[N], f[M], w[M], to[M], ne[M], tot;
int q[N], d[N], incf[N], pre[N];
bool vis[N];
int id[40][40], cost[40][40];

inline void add(int u, int v, int c, int d) {
    to[tot] = v, f[tot] = c, w[tot] = d, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, f[tot] = 0, w[tot] = -d, ne[tot] = h[v], h[v] = tot++;
}
```

```
20
21  bool spfa() {
22      int hh = 0, tt = 1;
23      memset(d, -0x3f, sizeof d);
24      memset(incf, 0, sizeof incf);
25      q[0] = S, d[S] = 0, incf[S] = INF;
26      while (hh != tt) {
27          int t = q[hh++];
28          if (hh == N) hh = 0;
29          vis[t] = false;
30          for (int i = h[t]; ~i; i = ne[i]) {
31              int ver = to[i];
32              if (f[i] && d[ver] < d[t] + w[i]) {
33                  d[ver] = d[t] + w[i];
34                  pre[ver] = i;
35                  incf[ver] = min(incf[t], f[i]);
36                  if (!vis[ver]) {
37                      q[tt++] = ver;
38                      if (tt == N) tt = 0;
39                      vis[ver] = true;
40                  }
41              }
42          }
43      }
44      return incf[T] > 0;
45  }
46
47  int EK() {
48      int cost = 0;
49      while (spfa()) {
50          int t = incf[T];
51          cost += t * d[T];
52          for (int i = T; i != S; i = to[pre[i] ^ 1]) {
53              f[pre[i]] -= t;
54              f[pre[i] ^ 1] += t;
55          }
56      }
57      return cost;
58  }
59
60  int main() {
61      int cnt = 0;
62      scanf("%d%d", &m, &n);
63      S = ++cnt;
64      T = ++cnt;
65      for (int i = 1; i <= n; ++i)
66          for (int j = 1; j <= m + i - 1; ++j) {
67              scanf("%d", &cost[i][j]);
68              id[i][j] = ++cnt;
69          }
70
71      //rule1
72      memset(h, -1, sizeof h), tot = 0;
73      for (int i = 1; i <= n; ++i)
74          for (int j = 1; j <= m + i - 1; ++j) {
75              add(id[i][j] * 2, id[i][j] * 2 + 1, 1, cost[i][j]);
76              if (i == 1) add(S, id[i][j] * 2, 1, 0);
77              if (i == n) add(id[i][j] * 2 + 1, T, 1, 0);
```

```
            if (i < n) {
                add(id[i][j] * 2 + 1, id[i + 1][j] * 2, 1, 0);
                add(id[i][j] * 2 + 1, id[i + 1][j + 1] * 2, 1, 0);
            }
        }
    printf("%d\n", EK());

    //rule2
    memset(h, -1, sizeof h), tot = 0;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m + i - 1; ++j) {
            add(id[i][j] * 2, id[i][j] * 2 + 1, INF, cost[i][j]);
            if (i == 1) add(S, id[i][j] * 2, 1, 0);
            if (i == n) add(id[i][j] * 2 + 1, T, INF, 0);
            if (i < n) {
                add(id[i][j] * 2 + 1, id[i + 1][j] * 2, 1, 0);
                add(id[i][j] * 2 + 1, id[i + 1][j + 1] * 2, 1, 0);
            }
        }
    printf("%d\n", EK());

    //rule3
    memset(h, -1, sizeof h), tot = 0;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m + i - 1; ++j) {
            add(id[i][j] * 2, id[i][j] * 2 + 1, INF, cost[i][j]);
            if (i == 1) add(S, id[i][j] * 2, 1, 0);
            if (i == n) add(id[i][j] * 2 + 1, T, INF, 0);
            if (i < n) {
                add(id[i][j] * 2 + 1, id[i + 1][j] * 2, INF, 0);
                add(id[i][j] * 2 + 1, id[i + 1][j + 1] * 2, INF, 0);
            }
        }
    printf("%d\n", EK());
    return 0;
}
```

费用流之上下界可行流

```
// NOI2008 志愿者招募
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 1010, M = 24010, INF = 0x3f3f3f3f;

int n, m, S, T;
int h[N], f[M], w[M], to[M], ne[M], tot;
int q[N], d[N], pre[N], incf[N];
bool vis[N];

inline void add(int u, int v, int c, int d) {
```

```
    to[tot] = v, f[tot] = c, w[tot] = d, ne[tot] = h[u], h[u] = tot++;
    to[tot] = u, f[tot] = 0, w[tot] = -d, ne[tot] = h[v], h[v] = tot++;
}

bool spfa() {
    int hh = 0, tt = 1;
    memset(d, 0x3f, sizeof d);
    memset(incf, 0, sizeof incf);
    q[0] = S, d[S] = 0, incf[S] = INF;
    while (hh != tt) {
        int t = q[hh++];
        if (hh == N) hh = 0;
        vis[t] = false;
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = to[i];
            if (f[i] && d[ver] > d[t] + w[i]) {
                d[ver] = d[t] + w[i];
                pre[ver] = i;
                incf[ver] = min(incf[t], f[i]);
                if (!vis[ver]) {
                    q[tt++] = ver;
                    if (tt == N) tt = 0;
                    vis[ver] = true;
                }
            }
        }
    }
    return incf[T] > 0;
}

int EK() {
    int cost = 0;
    while (spfa()) {
        int t = incf[T];
        cost += t * d[T];
        for (int i = T; i != S; i = to[pre[i] ^ 1]) {
            f[pre[i]] -= t;
            f[pre[i] ^ 1] += t;
        }
    }
    return cost;
}

int main() {
    scanf("%d%d", &n, &m);
    S = 0, T = n + 2;
    memset(h, -1, sizeof h);

    int last = 0;
    for (int i = 1; i <= n; ++i) {
        int cur;
        scanf("%d", &cur);
        if (last > cur) add(S, i, last - cur, 0);
        else if (cur > last) add(i, T, cur - last, 0);
        add(i, i + 1, INF - cur, 0);
        last = cur;
    }
    add(S, n + 1, last, 0);
```

```

    while (m--) {
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(b + 1, a, INF, c);
    }

    printf("%d", EK());
    return 0;
}
```

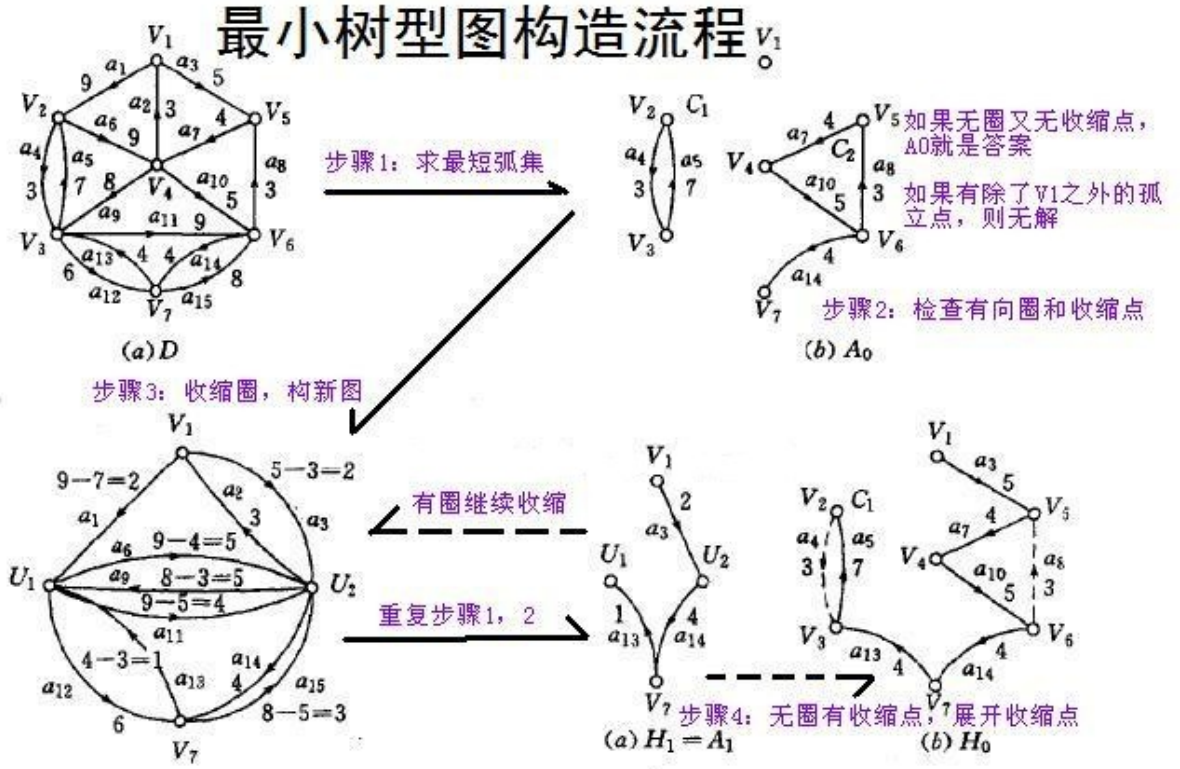
2-SAT问题

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

using namespace std;

const int N = 2000010, M = 2000010;

int n, m;
int h[N], to[M], ne[M], tot;
int dfn[N], low[N], ts, stk[N], top;
int id[N], cnt;
int ins[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

inline void add(int u, int v) {
    to[tot] = v, ne[tot] = h[u], h[u] = tot++;
}

void tarjan(int u) {
    dfn[u] = low[u] = ++ts;
    stk[++top] = u, ins[u] = true;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!dfn[j]) {
            tarjan(j);
            low[u] = min(low[u], low[j]);
        }
        else if (ins[j]) low[u] = min(low[u], dfn[j]);
    }

    if (low[u] == dfn[u]) {
        int y;
        cnt++;
        do {
```

```
            y = stk[top--], ins[y] = false, id[y] = cnt;
        } while (y != u);
    }
}

int main() {
    read(n), read(m);
    memset(h, -1, sizeof h);

    while (m--) {
        int i, a, j, b;
        read(i), read(a), read(j), read(b);
        i--, j--;
        add(2 * i + !a, 2 * j + b);
        add(2 * j + !b, 2 * i + a);
    }

    for (int i = 0; i < n * 2; ++i)
        if (!dfn[i])
            tarjan(i);

    for (int i = 0; i < n; ++i)
        if (id[i * 2] == id[i * 2 + 1]) {
            puts("IMPOSSIBLE");
            return 0;
        }
    puts("POSSIBLE");
    for (int i = 0; i < n; ++i)
        if (id[i * 2] < id[i * 2 + 1]) printf("0 ");
        else printf("1 ");
    return 0;
}
```

朱刘算法

`最小树形图`： 最小树形图，就是给有向带权图中指定一个特殊的点`root`，求一棵以`root`为根的有向生成树T，并且T中所有边的总权值最小。




```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<cstring>

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;

const int N = 110, INF = 1e8;

int n, m;
pdd q[N];
bool g[N][N];
double d[N][N], bd[N][N];
int pre[N], dpre[N];
int dfn[N], low[N], ts, stk[N], top;
int id[N], cnt;
bool vis[N], ins[N];

void dfs(int u) {
    vis[u] = true;
    for (int i = 1; i <= n; ++i)
        if (g[u][i] && !vis[i])
            dfs(i);
}

bool check_con() {
```

```
    memset(vis, 0, sizeof vis);
    dfs(1);
    for (int i = 1; i <= n; ++i)
        if (!vis[i])
            return false;
    return true;
}

double get_dist(int a, int b) {
    double dx = q[a].x - q[b].x;
    double dy = q[a].y - q[b].y;
    return sqrt(dx * dx + dy * dy);
}

void tarjan(int u) {
    dfn[u] = low[u] = ++ts;
    stk[++top] = u, ins[u] = true;

    int j = pre[u];
    if (!dfn[j]) {
        tarjan(j);
        low[u] = min(low[u], low[j]);
    }
    else if (ins[j]) low[u] = min(low[u], dfn[j]);

    if (low[u] == dfn[u]) {
        int y;
        ++cnt;
        do {
            y = stk[top--], ins[y] = false, id[y] = cnt;
        } while (y != u);
    }
}

double work() {
    double res = 0;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= n; ++j)
            if (g[i][j]) d[i][j] = get_dist(i, j);
            else d[i][j] = INF;

    while (true) {
        for (int i = 1; i <= n; ++i) {
            pre[i] = i;
            for (int j = 1; j <= n; ++j)
                if (d[pre[i]][i] > d[j][i])
                    pre[i] = j;
        }

        memset(dfn, 0, sizeof dfn);
        ts = cnt = 0;
        for (int i = 1; i <= n; ++i)
            if (!dfn[i])
                tarjan(i);

        if (cnt == n) {
            for (int i = 2; i <= n; ++i) res += d[pre[i]][i];
            break;
```

```
        }

        for (int i = 2; i <= n; ++i) {
            if (id[pre[i]] == id[i])
                res += d[pre[i]][i];
        }

        for (int i = 1; i <= cnt; ++i)
            for (int j = 1; j <= cnt; ++j)
                bd[i][j] = INF;

        for (int i = 1; i <= n; ++i)
            for (int j = 1; j <= n; ++j)
                if (d[i][j] < INF && id[i] != id[j]) {
                    int a = id[i], b = id[j];
                    if (id[pre[j]] == id[j]) bd[a][b] = min(bd[a][b], d[i][j] -
d[pre[j]][j]);
                    else bd[a][b] = min(bd[a][b], d[i][j]);
                }
        n = cnt;
        memcpy(d, bd, sizeof d);
    }
    return res;
}

int main() {
    while (~scanf("%d%d", &n, &m)) {
        for (int i = 1; i <= n; ++i) scanf("%lf%lf", &q[i].x, &q[i].y);

        memset(g, 0, sizeof g);
        while (m--) {
            int a, b;
            scanf("%d%d", &a, &b);
            if (a != b && b != 1) g[a][b] = true;
        }

        if (!check_con()) puts("poor snoopy");
        else printf("%.2lf\n", work());
    }
    return 0;
}
```

prufer序列

```
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 100010;

int n, m;
int f[N], d[N], p[N];

void tree2prufer() {
```

```
    for (int i = 1; i < n; ++i) {
        scanf("%d", &f[i]);
        d[f[i]]++;
    }

    for (int i = 0, j = 1; i < n - 2; j++) {
        while (d[j]) j++;
        p[i++] = f[j];
        while (i < n - 2 && --d[p[i - 1]] == 0 && p[i - 1] < j) p[i++] = f[p[i - 1]];
    }

    for (int i = 0; i < n - 2; ++i) printf("%d ", p[i]);
}

void prufer2tree() {
    for (int i = 1; i <= n - 2; ++i) {
        scanf("%d ", &p[i]);
        d[p[i]]++;
    }
    p[n - 1] = n;

    for (int i = 1, j = 1; i < n; ++i, ++j) {
        while (d[j]) j++;
        f[j] = p[i];
        while (i < n - 1 && --d[p[i]] == 0 && p[i] < j) f[p[i]] = p[i + 1], i++;
    }

    for (int i = 1; i <= n - 1; ++i) printf("%d ", f[i]);
}

int main() {
    scanf("%d%d", &n, &m);
    if (m == 1) tree2prufer();
    else prufer2tree();
    return 0;
}
```

数学

试除法判质数

```
#include<iostream>

using namespace std;

bool is_prime(int n) {
    if (n < 2) return false;
    for (int i = 2; i < n / i; ++i) {
        if (n % i == 0) return false;
    }
    return true;
}
```

```
12
13 int main() {
14     int n;
15     cin >> n;
16     for (int i = 0; i < n; ++i) {
17         int a;
18         cin >> a;
19         if (is_prime(a)) cout << "Yes" << endl;
20         else cout << "No" << endl;
21     }
22 }
```

分解质因数

由算术基本定理，一个整数 N 可分解成： $N = p_1^{a1} P_2^{a2} \cdots p_k^a k$

```
1  #include<iostream>
2
3  using namespace std;
4
5  void divide(int x) {
6      for (int i = 2; i <= x / i; i++) {
7          if (x % i == 0) {
8              int s = 0;
9              while (x % i == 0) x /= i, s++;
10             cout << i << ' ' << s << endl;
11         }
12     }
13     if (x > 1) cout << x << ' ' << 1 << endl;
14     cout << endl;
15 }
16
17 int main() {
18     int n;
19     cin >> n;
20     for (int i = 0; i < n; i++) {
21         int x;
22         cin >> x;
23         divide(x);
24     }
25     return 0;
26 }
```

线性筛

```
1  #include<iostream>
2
3  using namespace std;
4
5  const int N = 1000010;
6
7  int prime[N], cnt;
8  bool not_prime[N];
```

```
void get_prime(int n) {
    for (int i = 2; i <= n; i++) {
        if (!not_prime[i]) prime[cnt++] = i;
        for (int j = 0; prime[j] <= n / i; j++) {
            not_prime[prime[j] * i] = true;
            if (i % prime[j] == 0) break;
        }
    }
}

int main() {
    int n;
    cin >> n;
    get_prime(n);
    cout << cnt;
    return 0;
}
```

试除法求约数

```
#include<iostream>
#include<algorithm>
#include<vector>
using namespace std;
vector<int> get_divisors(int x){
    vector<int> res;
    for(int i=1;i<=x/i;i++){
        if(x%i==0){
            res.push_back(i);
            if(i!=x/i) res.push_back(x/i);
        }
    }
    sort(res.begin(),res.end());
    return res;
}
int main(){
    int  n,x;
    cin>>n;
    while(n--){
        cin>>x;
        vector<int> ans=get_divisors(x);
        for(int i=0;i<ans.size();i++) cout<<ans[i]<<" ";
        cout<<'\n';
    }
    return 0;
}
```

约数个数

N 的正约数个数为：$\prod_{i=1}^{m}(c_i+1)$

```cpp
#include<iostream>
#include<unordered_map>

using namespace std;

const int mod = 1e9 + 7;

int main() {
    int n, x;
    cin >> n;
    unordered_map<int, int> prime;
    while (n--) {
        cin >> x;
        for (int i = 2; i <= x / i; i++) {
            while (x % i == 0) {
                x /= i;
                prime[i]++;
            }
        }
        if (x > 1) prime[x]++;
    }
    long long ans = 1;
    for (unordered_map<int, int>::iterator i = prime.begin(); i != prime.end();
i++) {
        ans = ans * ((*i).second + 1) % mod;
    }
    cout << ans;
    return 0;
}
```

约数之和

N 的所有正约数和为：$\prod_{i=1}^{m}(\sum_{j=0}^{c_i}(p_i)^j)$

```cpp
#include<iostream>
#include<map>

using namespace std;

const int mod = 1e9 + 7;

int main() {
    int n, x;
    cin >> n;
    map<int, int> prime;
    while (n--) {
        cin >> x;
        for (int i = 2; i <= x / i; i++) {
            while (x % i == 0) {
```

```
                x /= i;
                prime[i]++;
            }
        }
        if (x > 1) prime[x]++;
    }

    long long ans = 1;
    for (map<int, int>::iterator i = prime.begin(); i != prime.end(); i++) {
        long long a = (*i).first, b = (*i).second;
        long long t = 1;
        while (b--) t = (t * a + 1) % mod;
        ans = ans * t % mod;
    }
    cout << ans;
    return 0;
}
```

欧几里得算法

```
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}
```

欧拉函数

$1 \sim N$ 中互质的数的个数被称为欧拉函数，记为 $\phi(N)$

$\phi(N) = N * \prod_{\text{质数}\ p|N} (1 - \frac{1}{p})$

```
#include<iostream>

using namespace std;

int phi(int n) {
    long long res = n;
    for (int i = 2; i <= n / i; i++) {
        if (n % i == 0) {
            res = res / i * (i - 1);
            while (n % i == 0) n /= i;
        }
    }
    if (n > 1) res = res / n * (n - 1);
    return res;
}

int main() {
    int n, x;
    cin >> n;
    while (n--) {
        cin >> x;
        cout << phi(x) << endl;
    }
```

```
    return 0;
}
```

筛法求欧拉函数

```
#include<iostream>

using namespace std;

const int N = 1000010;

int prime[N], phi[N], cnt;
bool not_prime[N];

int slove(int n) {
    phi[1] = 1;
    for (int i = 2; i <= n; i++) {
        if (!not_prime[i]) prime[cnt++] = i, phi[i] = i - 1;
        for (int j = 0; prime[j] <= n / i; j++) {
            not_prime[prime[j] * i] = true;
            if (i % prime[j] == 0) {
                phi[prime[j] * i] = phi[i] * prime[j];
                break;
            }
            phi[prime[j] * i] = phi[i] * (prime[j] - 1);
        }
    }
}

int main() {
    int n;
    cin >> n;
    slove(n);
    long long res = 0;
    for (int i = 1; i <= n; i++) res += phi[i];
    cout << res << endl;
    return 0;
}
```

扩展欧几里得算法

```
#include<iostream>
using namespace std;
int exgcd(int a,int b,int &x,int &y){
    if(b==0){x=1,y=0;return a;}
    int d=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}
int main(){
    int n;
    scanf("%d",&n);
    while(n--){
```

```
        int a,b,x,y;
        scanf("%d%d",&a,&b);
        exgcd(a,b,x,y);
        printf("%d %d\n",x,y);
    }
}
```

线性同余方程

$a*x\equiv b\pmod{m}$，在有解时，先用欧几里得算法求出一组整数 x_0,y_0，满足 $a*x_0+m*y_0=gcd(a,m)$，然后 $x=x_0*\frac{b}{gcd(a,m)}$ 就是原线性同余方程的一个解

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;

int exgcd(int a, int b, int& x, int& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

int main() {
    int n;
    scanf("%d", &n);
    while (n--) {
        int a, b, m, x, y;
        scanf("%d%d%d", &a, &b, &m);
        int d = exgcd(a, m, x, y);
        if (b % d) printf("impossible\n");
        else printf("%d\n", (long long)x * (b / d) % m);
    }
    return 0;
}
```

中国剩余定理

设 $m_1,m_2,\cdots,m_n$ 是两两互质的整数，$m=\prod_{i=1}^{n}m_i,M_i=\frac{m}{m_i}$，$t_i$ 是线性同余方程 $M_it_i\equiv 1(\pmod{m_i})$ 的一个解。对于任意的 n 个整数 $a_1,a_2,\cdots,a_n$，方程组

$$\begin{cases} x\equiv a_1(\mod m_1) \\ x\equiv a_2(\mod m_2) \\ . \\ . \\ . \\ x\equiv a_n(\mod m_n) \end{cases}$$

有整数解，解为 $x=\sum_{i=1}^{n}a_iM_it_i$

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>

using namespace std;

const int N = 15;
long long n, a[N], m[N], w[N], M = 1, ans;

inline void exgcd(long long a, long long b, long long& x, long long& y) { //扩展欧几里得求乘法逆元
    if (b == 0) {
        x = 1, y = 0;
        return;
    }
    exgcd(b, a % b, x, y);
    int z = x; x = y, y = z - y * (a / b);
}

int main() {
    scanf("%lld", &n);
    for (int i = 1; i <= n; i++) {
        int b;
        scanf("%d", &b);
        m[i] = b, M *= b;
        scanf("%lld", &a[i]);
    }
    for (int i = 1; i <= n; i++) {
        w[i] = M / m[i];
        long long x = 0, y = 0;
        exgcd(w[i], m[i], x, y);
        ans += a[i] * w[i] * (x < 0 ? x + m[i] : x);
    }
    printf("%lld", ans % M);
    return 0;
}
```

扩展中国剩余定理

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>

using namespace std;
typedef long long ll;
const int N = 100010;

ll a[N], b[N];
ll ans, m;
int n;

inline ll exgcd(ll a, ll b, ll& x, ll& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
```

```
    }
    ll d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

ll mul(ll a, ll b, ll p) {
    ll res = 0;
    while (b) {
        if (b & 1) res = (res + a) % p;
        a = (a + a) % p;
        b >>= 1;
    }
    return res;
}

ll exCRT() {
    m = b[1], ans = a[1];
    for (int i = 2; i <= n; ++i) {
        ll p = ((a[i] - ans) % b[i] + b[i]) % b[i];
        ll x, y;
        ll d = exgcd(m, b[i], x, y);
        x = mul(x, p / d, b[i] / d);
        ans += x * m;
        m *= b[i] / d;
        ans = (ans % m + m) % m;
    }
    return (ans % m + m) % m;
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        scanf("%lld%lld", &b[i], &a[i]);
    }
    printf("%lld", exCRT());
    return 0;
}
```

BSGS

解决高次同余方程

给定正整数 a,p,b ， a,p 互质，求满足 $a^x \equiv b(\mod p)$ 的最小非负整数 x

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<unordered_map>
#include<cmath>

using namespace std;
typedef long long ll;

int  bsgs(int a, int b, int p) {
    if (1 % p == b % p) return 0;
```

```
    int k = sqrt(p) + 1;
    unordered_map<int, int> hash;
    for (int i = 0, j = b % p; i < k; ++i) {
        hash[j] = i;
        j = (ll)j * a % p;
    }
    int ak = 1;
    for (int i = 0; i < k; ++i) ak = (ll)ak * a % p;
    for (int i = 1, j = ak; i <= k; ++i) {
        if (hash.count(j)) return (ll)i * k - hash[j];
        j = (ll)j * ak % p;
    }
    return -1;
}

int main() {
    int a, b, p;
    while (cin >> a >> p >> b, a || b || p) {
        int res = bsgs(a, b, p);
        if (res == -1) puts("No Solution");
        else cout << res << endl;
    }
    return 0;
}
```

扩展BSGS

当 a, p 不互质时的做法

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<map>

using namespace std;
typedef long long ll;
const int INF = 0x3f3f3f3f;

int exgcd(int a, int b, int& x, int& y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}

int bsgs(int a, int b, int p) {
    if (1 % p == b % p) return 0;
    int k = sqrt(p) + 1;
    map<int, int> hash;
    for (int i = 0, j = b % p; i < k; ++i) {
        hash[j] = i;
        j = (ll)j * a % p;
```

```
    }
    int ak = 1;
    for (int i = 0; i < k; ++i) ak = (ll)ak * a % p;
    for (int i = 1, j = ak; i <= k; ++i) {
        if (hash.count(j)) return i * k - hash[j];
        j = (ll)j * ak % p;
    }
    return -INF;
}

int exbsgs(int a, int b, int p) {
    b = (b % p + p) % p;
    if (1 % p == b % p) return 0;
    int x, y;
    int d = exgcd(a, p, x, y);
    if (d > 1) {
        if (b % d) return -INF;
        exgcd(a / d, p / d, x, y);
        return exbsgs(a, (ll)b / d * x % (p / d), p / d) + 1;
    }
    return bsgs(a, b, p);
}

int main() {
    int a, p, b;
    while (cin >> a >> p >> b, a || p || b) {
        int res = exbsgs(a, b, p);
        if (res < 0) puts("No Solution");
        else cout << res << endl;
    }
    return 0;
}
```

组合数

$\binom{n}{m} = \frac{n!}{m!(n-m)!}$

$\binom{n}{m} = \binom{n}{n-m}$

$\binom{n}{m} = \binom{n-1}{m}\binom{n-1}{m-1}$

求法1

```
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 2010;
const int mod = 1e9 + 7;

int c[N][N];

void init() {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j <= i; j++) {
```

```
            if (!j) c[i][j] = 1;
            else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
        }
    }
}

int main() {
    init();
    int n;
    scanf("%d", &n);
    while (n--) {
        int a, b;
        scanf("%d%d", &a, &b);
        printf("%d\n", c[a][b]);
    }
    return 0;
}
```

求法2

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
const int N=100005,mod=1e9+7;
int fact[N],infact[N];

int qmi(int a,int k,int p){
    int res=1;
    while(k){
        if(k&1) res=(long long)res*a%p;
        a=(long long)a*a%mod;
        k>>=1;
    }
    return res;
}

void init(){
    fact[0]=infact[0]=1;
    for(int i=1;i<N;i++){
        fact[i]=(long long)fact[i-1]*i%mod;
        infact[i]=(long long)infact[i-1]*qmi(i,mod-2,mod)%mod;
    }
}

int main(){
    int n;
    scanf("%d",&n);
    init();
    while(n--){
        int a,b;
        scanf("%d%d",&a,&b);
        printf("%d\n",(long long)fact[a]*infact[a-b]%mod*infact[b]%mod);
    }
```

```
    return 0;
}
```

求法3(高精度)

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<vector>

using namespace std;

const int N = 5010;

int primes[N], cnt, sum[N];
bool st[N];

void get_primes(int n) {
    for (int i = 2; i <= n; i++) {
        if (!st[i]) primes[cnt++] = i;
        for (int j = 0; primes[j] <= n / i; j++) {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int get(int n, int p) {
    int res = 0;
    while (n) {
        res += n / p;
        n /= p;
    }
    return res;
}

vector<int> mul(vector<int> a, int b) {
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i++) {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }
    while (t) {
        c.push_back(t % 10);
        t /= 10;
    }
    return c;
}

int main() {
    int a, b;
    cin >> a >> b;
    get_primes(a);
```

```
51
52      for (int i = 0; i < cnt; i++) {
53          int p = primes[i];
54          sum[i] = get(a, p) - get(a - b, p) - get(b, p);
55      }
56
57      vector<int> res;
58      res.push_back(1);
59      for (int i = 0; i < cnt; i++)
60          for (int j = 0; j < sum[i]; j++)
61              res = mul(res, primes[i]);
62      for (int i = res.size() - 1; i >= 0; i--) printf("%d", res[i]);
63      return 0;
64  }
```

Lucas定理

$$\binom{n}{m} \bmod p = \binom{\lfloor n/p \rfloor}{\lfloor m/p \rfloor} \cdot \binom{n \bmod p}{m \bmod p} \bmod p$$

```
1   #include<iostream>
2   #include<cstdio>
3
4   using namespace std;
5   typedef long long LL;
6
7   int qmi(int a, int k, int p) {
8       int res = 1;
9       while (k) {
10          if (k & 1) res = (LL)res * a % p;
11          a = (LL)a * a % p;
12          k >>= 1;
13      }
14      return res;
15  }
16
17  int C(int a, int b, int p) {
18      if (b > a) return 0;
19      int res = 1;
20      for (int i = 1, j = a; i <= b; i++, j--) {
21          res = (LL)res * j % p;
22          res = (LL)res * qmi(i, p - 2, p) % p;
23      }
24      return res;
25  }
26
27  int lucas(LL a, LL b, int p) {
28      if (a < p && b < p) return C(a, b, p);
29      return (LL)C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
30  }
31
32  int main() {
33      int T;
34      cin >> T;
35      while (T--) {
```

```
        LL a, b;
        int p;
        cin >> a >> b >> p;
        cout << lucas(a, b, p) << endl;
    }
    return 0;
}
```

Catalan数

$$\binom{2n}{n} - \binom{2n}{n-1} = \frac{\binom{2n}{n}}{n+1}$$

```
// 给定 n 个 0 和 n 个 1，它们将按照某种顺序排成长度为 2n 的序列
// 求它们能排列成的所有序列中，能够满足任意前缀序列中 0 的个数都不少于 1 的个数的序列有多少个
#include<iostream>
#include<cstdio>

using namespace std;
typedef long long ll;

const int mod = 1e9 + 7;

inline int qmi(int a, int b, int p) {
    int res = 1;
    while (b) {
        if (b & 1) res = (ll)res * a % p;
        a = (ll)a * a % p;
        b >>= 1;
    }
    return res;
}

int main() {
    int n;
    cin >> n;

    int a = n * 2, b = n;
    int res = 1;
    for (int i = a; i > a - b; --i) res = (ll)res * i % mod;
    for (int i = 1; i <= b; ++i) res = (ll)res * qmi(i, mod - 2, mod) % mod;
    res = (ll)res * qmi(n + 1, mod - 2, mod) % mod;

    cout << res;
    return 0;
}
```

第一类Stirling数(斯特林轮换数)

将 n 个两两不同的元素，划分为 k 个非空圆排列的方案数

$$\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1) \begin{bmatrix} n-1 \\ k \end{bmatrix}$$

```cpp
#include<iostream>
#include<cstdio>

using namespace std;
typedef long long ll;

const int N = 1010, mod = 1e9 + 7;

int n, m;
int f[N][N];

int main() {
    scanf("%d%d", &n, &m);
    f[0][0] = 1;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            f[i][j] = (f[i - 1][j - 1] + (ll)(i - 1) * f[i - 1][j]) % mod;
    cout << f[n][m];
    return 0;
}
```

第二类Stirling数(斯特林子集数)

将 n 个两两不同的元素，划分为 k 个非空子集的方案数

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k \begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

```cpp
#include<iostream>
#include<cstdio>

using namespace std;
typedef long long ll;

const int N = 1010, mod = 1e9 + 7;

int n, m;
int f[N][N];

int main() {
    scanf("%d%d", &n, &m);
    f[0][0] = 1;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
            f[i][j] = (f[i - 1][j - 1] + (ll)j * f[i - 1][j]) % mod;
    cout << f[n][m];
    return 0;
}
```

高斯消元

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>

using namespace std;

const int N = 105;
const double eps = 1e-6;

int n;
double a[N][N];

int gauss() {
    int c, r;
    for (c = 0, r = 0; c < n; c++) { //枚举每一行
        int t = r;
        for (int i = r; i < n; i++)
            if (fabs(a[i][c]) > fabs(a[t][c])) //找到绝对值最大的一行
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i < n + 1; i++) swap(a[t][i], a[r][i]); //将该行换上去
        for (int i = n; i >= c; i--) a[r][i] /= a[r][c]; //将该行的第一个系数变为1

        for (int i = r + 1; i < n; i++)
            if (fabs(a[i][c]) > eps)
                for (int j = n; j >= c; j--)
                    a[i][j] -= a[r][j] * a[i][c]; //将下面所有行的第c列消成0
        r++;
    }

    if (r < n) {
        for (int i = r; i < n; i++) {
            if (fabs(a[i][n]) > eps)
                return 2; //无解
        }
        return 1; //有无穷多组解
    }

    for (int i = n; i >= 0; i--)
        for (int j = i + 1; j < n; j++)
            a[i][n] -= a[j][n] * a[i][j];

    return 0;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n + 1; j++)
```

```
            scanf("%lf", &a[i][j]);

    int t = gauss();

    if (t == 0) {
        for (int i = 0; i < n; i++) printf("%.2lf\n", a[i][n]);
    }
    else if (t == 1) printf("Infinite group solutions");
    else printf("No solution");
    return 0;
}
```

Nim博弈

Nim博弈先手必胜，当且仅当 $A_1 xor A_2 xor \cdots xor A_n \neq 0$

```
#include<iostream>
#include<cstdio>

using namespace std;

int main() {
    int n;
    cin >> n;
    int res = 0;
    while (n--) {
        int x;
        cin >> x;
        res ^= x;
    }
    if (res) puts("Yes");
    else puts("No");
    return 0;
}
```

莫比乌斯反演

若 $F(n) = \sum_{d|n}(d)$ ， 则 $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$

burnside引理

每个置换的不动点个数的平均值就是不同的方案数

Pólya定理

旋转置换：一共n个置换，第i个置换的循环节的个数为 $gcd(n, i)$

则： $\sum_{k=0}^{m} m^{(n,k)}$

旋转置换：

n 为奇数：$n * m^{\frac{n+1}{2}}$

n 为偶数：$\frac{(n*m^{\frac{n}{2}+1}+n*m^{\frac{n}{2}})}{\frac{n}{2}}$

```cpp
#include<iostream>
#include<cstdio>

using namespace std;
typedef long long ll;

int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

ll power(int a, int k) {
    ll res = 1;
    while (k) {
        if (k & 1) res *= a;
        a *= a;
        k >>= 1;
    }
    return res;
}

int main() {
    int m, n;
    while (cin >> m >> n, m || n) {
        ll res = 0;
        for (int i = 0; i < n; ++i)
            res += power(m, gcd(n, i));
        if (n % 2)
            res += n * power(m, (n + 1) / 2);
        else
            res += n / 2 * (power(m, n / 2 + 1) + power(m, n / 2));
        cout << res / n / 2 << endl;
    }
    return 0;
}
```

FFT

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

using namespace std;
const int N = 300010;
const double pi = acos(-1);

int n, m;
int rev[N], bit, tot;

struct Complex {
```

```
    double x, y;
    Complex operator+ (Complex const& t)const {
        return { x + t.x,y + t.y };
    }
    Complex operator- (Complex const& t)const {
        return { x - t.x,y - t.y };
    }
    Complex operator* (Complex const& t)const {
        return { x * t.x - y * t.y,x * t.y + y * t.x };
    }
}a[N], b[N];

inline void FFT(Complex a[], int inv) {
    for (int i = 0; i < tot; ++i)
        if (i < rev[i])
            swap(a[i], a[rev[i]]);
    for (int mid = 1; mid < tot; mid <<= 1) {
        Complex w1 = Complex({ cos(pi / mid),inv * sin(pi / mid) });
        for (int i = 0; i < tot; i += mid * 2) {
            Complex wk = Complex({ 1,0 });
            for (int j = 0; j < mid; ++j, wk = wk * w1) {
                Complex x = a[i + j], y = wk * a[i + j + mid];
                a[i + j] = x + y, a[i + j + mid] = x - y;
            }
        }
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i <= n; ++i) scanf("%lf", &a[i].x);
    for (int i = 0; i <= m; ++i) scanf("%lf", &b[i].x);
    while ((1 << bit) < n + m + 1) bit++;
    tot = 1 << bit;
    for (int i = 0; i < tot; ++i)
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));
    FFT(a, 1), FFT(b, 1);
    for (int i = 0; i < tot; ++i) a[i] = a[i] * b[i];
    FFT(a, -1);
    for (int i = 0; i <= n + m; ++i)
        printf("%d ", (int)(a[i].x / tot + 0.5));
    return 0;
}
```

计算几何

知识点：

1.前置知识

(1) `pi=acos(-1)`

(2) 余弦定理： $c^2 = a^2 + b^2 - 2abcos\theta$

(3) 正弦定理： $\frac{A}{sinA} = \frac{B}{sinB} = \frac{C}{sinC} = 2R$

2.浮点数比较

```
const double eps = 1e-8;
int sign(double x) { // 符号函数
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}
int cmp(double x, double y) {  // 比较函数
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}
```

3.向量

(1)向量的加减法和数乘运算

(2)内积(点积) $\vec{A} \cdot \vec{B} = |\vec{A}||\vec{B}|cos\theta$

几何意义： 向量 $\vec{A}$ 在向量 $\vec{B}$ 上的投影与 B 的长度的乘积

```
double dot(Point a, Point b) {
    return a.x * b.x + a.y * b.y;
}
```

(3)外积(叉积) $\vec{A} \times \vec{B} = |\vec{A}||\vec{B}|sin\theta$

几何意义： 向量 $\vec{A}$ 与向量 $\vec{B}$ 张成的平行四边形的有向面积

```
double cross(Point a, Point b) {
    return a.x * b.y - b.x * a.y;
}
```

(4)常用函数

```
// 取模
double get_lenth(Point a) {
    return sqrt(dot(a, a));
}
// 计算向量夹角
double get_angle(Point a,Point b) {
    return acos(dot(a, b) / get_lenth(a) / get_lenth(b));
}
// 计算两个向量构成的平行四边形的有向面积
double area(Point a,Point b,Point c) {
    return cross(b - a, c - a);
}
// 向量A顺时针旋转theta角度
Point rotate(Point a,double theta) {
    return Point(a.x * cos(theta) + a.y * sin(theta), -a.x * sin(theta) + a.y *
cos(theta));
}
```

4.点与线

(1)直线的形式

一般式： $ax + by + c = 0$

点向式： $p_0 + vt$

斜截式： $y = kx + b$

(2)常用操作

```
// 判断点在直线上 A x B = 0
// 两直线相交
// cross(v, w) == 0则两直线平行或者重合
Point get_line_intersection(Point p, Vector v, Point q, vector w) {
    vector u = p - q;
    double t = cross(w, u) / cross(v, w);
    return p + v * t;
}
// 点到直线的距离
double distance_to_line(Point p, Point a, Point b) {
    vector v1 = b - a, v2 = p - a;
    return fabs(cross(v1, v2) / get_length(v1));
}
// 点到线段的距离
double distance_to_segment(Point p, Point a, Point b) {
    if (a == b) return get_length(p - a);
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if (sign(dot(v1, v2)) < 0) return get_length(v2);
    if (sign(dot(v1, v3)) > 0) return get_length(v3);
        return distance_to_line(p, a, b);
}
// 点在直线上的投影
double get_line_projection(Point p, Point a, Point b) {
    Vector v = b - a;
    return a + v * (dot(v, p - a) / dot(v, v));
}
// 点是否在线段上
bool on_segment(Point p, Point a, Point b) {
    return sign(cross(p - a, p - b)) == 0 && sign(dot(p - a, p - b)) <= 0;
}
// 判断两线段是否相交
bool segment_intersection(Point a1, Point a2, Point b1, Point b2) {
    double c1 = cross(a2 - a1, b1 - a1), c2 = cross(a2 - a1, b2 - a1);
    double c3 = cross(b2 - b1, a2 - b1), c4 = cross(b2 - b1, a1 - b1);
    return sign(c1) * sign(c2) <= 0 && sign(c3) * sign(c4) <= 0;
}
```

5.多边形

(1)三角形

(2)面积

叉积

海伦公式： $S = \sqrt{p(p-a)(p-b)(p-c)}, p = \frac{a+b+c}{2}$

(3)三角形四心

外心： 外接圆圆心。三边中垂线交点，到三角形三个顶点距离相等

内心： 内切圆圆心。角平分线交点，到三边距离相等

垂心： 三条垂线交点

重心： 三条中线交点。到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点

(4)普通多边形

多边形： 由在同一平面且不再同一直线上的多条线段首尾顺次连接且不相交所组成的图形叫多边形

简单多边形： 简单多边形是除相邻边外其它边不相交的多边形

凸多边形：

过多边形的任意一边做一条直线，如果其他各个顶点都在这条直线的同侧，则把这个多边形叫做凸多边形

任意凸多边形外角和均为 360°

任意凸多边形内角和为 $(n-2)180^\circ$

(5)常用函数

```
// (1)求多边形面积（不一定是凸多边形）
// 可以从第一个顶点除法把凸多边形分成n - 2个三角形，然后把面积加起来。
double polygon_area(Point p[], int n) {
    double s = 0;
    for (int i = 1; i + 1 < n; i ++ )
        s += cross(p[i] - p[0], p[i + 1] - p[i]);
    return s / 2;
}
(2) 判断点是否在多边形内（不一定是凸多边形）
    a. 射线法，从该点任意做一条和所有边都不平行的射线。交点个数为偶数，则在多边形外，为奇数，则在多边形内。
    b. 转角法
(3) 判断点是否在凸多边形内
    只需判断点是否在所有边的左边（逆时针存储多边形）。
```

(6)皮克定理

皮克定理是指一个计算点阵中顶点在格点上的多边形面积公式该公式可以表示为: $S = a + \frac{b}{2} - 1$，其中 a 表示多边形内部的点数，b 表示多边形边界上的点数，S 表示多边形的面积。

6. 圆
   (1) 圆与直线交点
   (2) 两圆交点
   (3) 点到圆的切线
   (4) 两圆公切线
   (5) 两圆相交面积

凸包

把所有点以横坐标为第一关键字，纵坐标为第二关键字排序

排序后最小的元素和最大的元素一定在凸包上。而且因为是凸多边形，我们如果从一个点出发逆时针走，轨迹总是“左拐”的，一旦出现右拐，就说明这一段不在凸包上。因此我们可以用一个单调栈来维护上下凸壳

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
```

```cpp

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;
const int N = 10010;

pdd q[N];
int n;
int stack[N];
bool used[N];

pdd operator-(pdd a, pdd b) {
    return { a.x - b.x,a.y - b.y };
}

double get_dist(pdd a, pdd b) {
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

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

double area(pdd a, pdd b, pdd c) {
    return cross(b - a, c - a);
}

double Andrew() {
    sort(q, q + n);
    int top = 0;
    for (int i = 0; i < n; ++i) {
        while (top >= 2 && area(q[stack[top - 1]], q[stack[top]], q[i]) <= 0) {
            if (area(q[stack[top - 1]], q[stack[top]], q[i]) < 0)
                used[stack[top--]] = false;
            else top--;
        }
        stack[++top] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; --i) {
        if (used[i]) continue;
        while (top >= 2 && area(q[stack[top - 1]], q[stack[top]], q[i]) <= 0)
            used[stack[top--]] = false;
        stack[++top] = i;
    }

    double res = 0;
    for (int i = 2; i <= top; ++i) {
        res += get_dist(q[stack[i - 1]], q[stack[i]]);
    }
    return res;
}

```

```
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%lf%lf", &q[i].x, &q[i].y);

    double ans = Andrew();
    printf("%.2lf", ans);
    return 0;
}
```

半平面交

半平面交是指多个半平面的交集。因为半平面是点集，所以点集的交集仍然是点集。在平面直角坐标系围成一个区域。

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;
const int N = 510;
const double eps = 1e-10;

int cnt;
int q[N];
pdd pg[N], ans[N];

struct Line {
    pdd st, ed;
}line[N];

pdd operator-(pdd a, pdd b) {
    return { a.x - b.x,a.y - b.y };
}

int sign(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y) {
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

double get_angle(Line a) {
    return atan2(a.ed.y - a.st.y, a.ed.x - a.st.x);
}
```

```
41
42  double cross(pdd a, pdd b) {
43      return a.x * b.y - a.y * b.x;
44  }
45
46  double area(pdd a, pdd b, pdd c) {
47      return cross(b - a, c - a);
48  }
49
50  bool cmp(const Line& a, const Line& b) {
51      double A = get_angle(a), B = get_angle(b);
52      if (!dcmp(A, B)) return area(a.st, a.ed, b.ed) < 0;
53      return A < B;
54  }
55
56  pdd Get_line_intersection(pdd p, pdd v, pdd q, pdd w) {
57      pdd u = p - q;
58      double t = cross(w, u) / cross(v, w);
59      return { p.x + t * v.x, p.y + t * v.y };
60  }
61
62  pdd get_line_intersection(Line a, Line b) { //获取两直线交点
63      return Get_line_intersection(a.st, a.ed - a.st, b.st, b.ed - b.st);
64  }
65
66  bool on_right(Line a, Line b, Line c) { // 判断bc的交点是否在a的右侧
67      pdd o = get_line_intersection(b, c);
68      return sign(area(a.st, a.ed, o)) <= 0;
69  }
70
71  double half_plane_intersection() {
72      sort(line, line + cnt, cmp);
73      int hh = 0, tt = -1;
74      for (int i = 0; i < cnt; ++i) {
75          if (i && !dcmp(get_angle(line[i]), get_angle(line[i - 1]))) continue;
76          while (hh + 1 <= tt && on_right(line[i], line[q[tt - 1]], line[q[tt]]))
    tt--;
77          while (hh + 1 <= tt && on_right(line[i], line[q[hh]], line[q[hh + 1]]))
    hh++;
78          q[++tt] = i;
79      }
80
81      while (hh + 1 <= tt && on_right(line[q[hh]], line[q[tt - 1]], line[q[tt]]))
    tt--;
82      while (hh + 1 <= tt && on_right(line[q[tt]], line[q[hh]], line[q[hh + 1]]))
    hh++;
83      q[++tt] = q[hh];
84
85      int k = 0;
86      for (int i = hh; i < tt; ++i) {
87          ans[k++] = get_line_intersection(line[q[i]], line[q[i + 1]]);
88      }
89
90      double res = 0;
91      for (int i = 1; i + 1 < k; ++i) {
92          res += area(ans[0], ans[i], ans[i + 1]);
93      }
94      return res / 2;
```

```
}

int main() {
    int n, m;
    scanf("%d", &n);
    while (n--) {
        scanf("%d", &m);
        for (int i = 0; i < m; ++i)
            scanf("%lf%lf", &pg[i].x, &pg[i].y);
        for (int i = 0; i < m; ++i)
            line[cnt++] = { pg[i],pg[(i + 1) % m] };
    }

    double ans = half_plane_intersection();
    printf("%.3lf", ans);
    return 0;
}
```

最小圆覆盖

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;
const int N = 100010;
const double eps = 1e-12;
const double pi = acos(-1);

int n;
pdd q[N];

struct Circle {
    pdd p;
    double r;
};

pdd operator-(pdd a, pdd b) {
    return { a.x - b.x,a.y - b.y };
}

pdd operator+(pdd a, pdd b) {
    return { a.x + b.x,a.y + b.y };
}

pdd operator*(pdd a, double t) { // 数乘
    return { t * a.x,t * a.y };
}

double operator*(pdd a, pdd b) { //叉乘
```

```
    return a.x * b.y - a.y * b.x;
}

pdd operator/(pdd a, double t) {
    return { a.x / t,a.y / t };
}

int sign(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y) {
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

double get_dist(pdd a, pdd b) {
    double dx = a.x - b.x;
    double dy = a.y - b.y;
    return sqrt(dx * dx + dy * dy);
}

pdd rotate(pdd a, double theta) {
    return { a.x * cos(theta) + a.y * sin(theta),-a.x * sin(theta) + a.y *
cos(theta) };
}

pdd get_line_intersection(pdd p, pdd v, pdd q, pdd w) {
    pdd u = p - q;
    double t = w * u / (v * w);
    return p + v * t;
}

pair<pdd, pdd> get_line(pdd a, pdd b) {
    return { (a + b) / 2,rotate(b - a,pi / 2) };
}

Circle get_circle(pdd a, pdd b, pdd c) {
    pair<pdd, pdd> u = get_line(a, b), v = get_line(a, c);
    pdd p = get_line_intersection(u.x, u.y, v.x, v.y);
    return { p,get_dist(p,a) };
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%lf%lf", &q[i].x, &q[i].y);
    random_shuffle(q, q + n);

    Circle c({ q[0],0 });
    for (int i = 1; i < n; ++i) {
        if (dcmp(c.r, get_dist(c.p, q[i])) < 0) {
            c = { q[i],0 };
            for (int j = 0; j < i; ++j)
                if (dcmp(c.r, get_dist(c.p, q[j])) < 0) {
```

```
                    c = { (q[i] + q[j]) / 2,get_dist(q[i],q[j]) / 2 };
                    for (int k = 0; k < j; ++k)
                        if (dcmp(c.r, get_dist(c.p, q[k])) < 0)
                            c = get_circle(q[i], q[j], q[k]);
                }
            }
        }
        printf("%.10lf\n%.10lf %.10lf", c.r, c.p.x, c.p.y);
        return 0;
}
```

三维凸包

```
// 求三维凸包面积
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;
const int N = 210;
const double eps = 1e-12;

int n, m;
bool g[N][N];

double rand_eps() {
    return ((double)rand() / RAND_MAX - 0.5) * eps;
}

struct Point {
    double x, y, z;
    void shake() {
        x += rand_eps(), y += rand_eps(), z += rand_eps();
    }
    Point operator-(Point t) {
        return { x - t.x,y - t.y,z - t.z };
    }
    double operator&(Point t) {
        return x * t.x + y * t.y + z * t.z;
    }
    Point operator*(Point t) {
        return { y * t.z - z * t.y,z * t.x - x * t.z,x * t.y - y * t.x };
    }
    double len() {
        return sqrt(x * x + y * y + z * z);
    }
}q[N];

struct Plane {
    int v[3];
    Point norm() {
```

```
        return (q[v[1]] - q[v[0]]) * (q[v[2]] - q[v[0]]);
    }
    double area() {
        return norm().len() / 2;
    }
    bool above(Point a) {
        return ((a - q[v[0]]) & norm()) >= 0;
    }
}plane[N], np[N];

void get_convex_3d() {
    plane[m++] = { 0,1,2 };
    plane[m++] = { 2,1,0 };

    for (int i = 3; i < n; ++i) {
        int cnt = 0;
        for (int j = 0; j < m; ++j) {
            bool t = plane[j].above(q[i]);
            if (!t) np[cnt++] = plane[j];
            for (int k = 0; k < 3; ++k)
                g[plane[j].v[k]][plane[j].v[(k + 1) % 3]] = t;
        }
        for (int j = 0; j < m; ++j) {
            for (int k = 0; k < 3; ++k) {
                int a = plane[j].v[k], b = plane[j].v[(k + 1) % 3];
                if (g[a][b] && !g[b][a])
                    np[cnt++] = { a,b,i };
            }
        }
        m = cnt;
        for (int j = 0; j < m; ++j) plane[j] = np[j];
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i) {
        scanf("%lf%lf%lf", &q[i].x, &q[i].y, &q[i].z);
        q[i].shake();
    }

    get_convex_3d();

    double res = 0;
    for (int i = 0; i < m; ++i) {
        res += plane[i].area();
    }
    printf("%.6lf", res);
    return 0;
}
```

旋转卡壳

给定一个二维平面，平面上有 N 个点。每个点的位置可以用一对整数坐标 (x, y) 表示，求出平面上距离最远的点对之间的距离

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;
typedef pair<int, int> pii;
const int N = 50010;

int n, top;
pii q[N];
int stk[N];
bool used[N];

pii operator-(pii a, pii b) {
    return { a.x - b.x,a.y - b.y };
}

int operator*(pii a, pii b) {
    return a.x * b.y - a.y * b.x;
}

int area(pii a, pii b, pii c) {
    return (b - a) * (c - a);
}

int get_dist(pii a, pii b) {
    int dx = a.x - b.x;
    int dy = a.y - b.y;
    return dx * dx + dy * dy;
}

void get_convex() {
    sort(q, q + n);
    for (int i = 0; i < n; ++i) {
        while (top >= 2 && area(q[stk[top - 2]], q[stk[top - 1]], q[i]) <= 0) {
            if (area(q[stk[top - 2]], q[stk[top - 1]], q[i]) < 0)
                used[stk[--top]] = false;
            else top--;
        }
        stk[top++] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; --i) {
        if (used[i]) continue;
        while (top >= 2 && area(q[stk[top - 2]], q[stk[top - 1]], q[i]) <= 0)
            top--;
        stk[top++] = i;
```

```
    }
    top--;
}

int rotating_calipers() {
    if (top <= 2) return get_dist(q[0], q[n - 1]);

    int res = 0;
    for (int i = 0, j = 2; i < top; ++i) {
        pii d = q[stk[i]], e = q[stk[i + 1]];
        while (area(d, e, q[stk[j]]) < area(d, e, q[stk[j + 1]])) j = (j + 1) %
top;
        res = max(res, max(get_dist(d, q[stk[j]]), get_dist(e, q[stk[j]])));
    }
    return res;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%d%d", &q[i].x, &q[i].y);

    get_convex();
    printf("%d", rotating_calipers());
    return 0;
}
```

最小矩形覆盖

用一个面积最下的矩形覆盖一堆点，用旋转卡壳解决

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;
const int N = 50010;
const double eps = 1e-12, INF = 1e20;
const double pi = acos(-1);

int n;
pdd q[N];
int stk[N], top;
bool used[N];
pdd ans[N];
double area_min = INF;

pdd operator+(pdd a, pdd b) {
    return { a.x + b.x,a.y + b.y };
}
```

```
25
26  pdd operator-(pdd a, pdd b) {
27      return { a.x - b.x,a.y - b.y };
28  }
29
30  pdd operator*(pdd a, double t) {
31      return { a.x * t,a.y * t };
32  }
33
34  pdd operator/(pdd a, double t) {
35      return { a.x / t,a.y / t };
36  }
37
38  double operator*(pdd a, pdd b) {
39      return a.x * b.y - a.y * b.x;
40  }
41
42  double operator&(pdd a, pdd b) {
43      return a.x * b.x + a.y * b.y;
44  }
45
46  int sign(double x) {
47      if (fabs(x) < eps) return 0;
48      if (x < 0) return -1;
49      return 1;
50  }
51
52  int dcmp(double x, double y) {
53      if (fabs(x - y) < eps) return 0;
54      if (x < y) return -1;
55      return 1;
56  }
57
58  double area(pdd a, pdd b, pdd c) {
59      return (b - a) * (c - a);
60  }
61
62  double get_len(pdd a) {
63      return sqrt(a & a);
64  }
65
66  pdd norm(pdd a) {
67      return a / get_len(a);
68  }
69
70  double project(pdd a, pdd b, pdd c) {
71      return ((b - a) & (c - a)) / get_len(b - a);
72  }
73
74  pdd rotate(pdd a, double theta) {
75      return { a.x * cos(theta) + a.y * sin(theta),-a.x * sin(theta) + a.y *
    cos(theta) };
76  }
77
78  void get_convex() {
79      sort(q, q + n);
80      for (int i = 0; i < n; ++i) {
```

```
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) >= 0)
            used[stk[--top]] = false;
        stk[top++] = i;
        used[i] = true;
    }
    used[0] = false;
    for (int i = n - 1; i >= 0; --i) {
        if (used[i]) continue;
        while (top >= 2 && sign(area(q[stk[top - 2]], q[stk[top - 1]], q[i])) >= 0)
            top--;
        stk[top++] = i;
    }
    reverse(stk, stk + top);
    top--;
}

void rotating_calipers() {
    for (int i = 0, a = 2, b = 2, c = 2; i < top; ++i) {
        pdd d = q[stk[i]], e = q[stk[i + 1]];
        while (dcmp(area(d, e, q[stk[a]]), area(d, e, q[stk[a + 1]])) < 0) a = (a + 1) % top;
        while (dcmp(project(d, e, q[stk[b]]), project(d, e, q[stk[b + 1]])) < 0) b = (b + 1) % top;
        if (!i) c = a;
        while (dcmp(project(d, e, q[stk[c]]), project(d, e, q[stk[c + 1]])) > 0) c = (c + 1) % top;
        pdd x = q[stk[a]], y = q[stk[b]], z = q[stk[c]];
        double h = area(d, e, x) / get_len(e - d);
        double w = ((y - z) & (e - d)) / get_len(e - d);
        if (h * w < area_min) {
            area_min = h * w;
            ans[0] = d + norm(e - d) * project(d, e, y);
            ans[3] = d + norm(e - d) * project(d, e, z);
            pdd u = norm(rotate(e - d, -pi / 2));
            ans[1] = ans[0] + u * h;
            ans[2] = ans[3] + u * h;
        }
    }
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; ++i)
        scanf("%lf%lf", &q[i].x, &q[i].y);

    get_convex();
    rotating_calipers();

    int k = 0;
    for (int i = 1; i < 4; ++i) {
        if (dcmp(ans[i].y, ans[k].y) < 0 || !dcmp(ans[i].y, ans[k].y) && dcmp(ans[i].x, ans[k].x))
            k = i;
    }

    printf("%.5lf\n", area_min);
```

```
    for (int i = 0; i < 4; ++i, ++k) {
        double x = ans[k % 4].x, y = ans[k % 4].y;
        if (!sign(x)) x = 0; // 防止-0.00000这种神奇情况发生
        if (!sign(y)) y = 0;
        printf("%.5lf %.5lf\n", x, y);
    }
    return 0;
}
```

三角剖分

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#include<cmath>

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;

const int N = 55;
const double eps = 1e-8;
const double pi = acos(-1);

double R;
int n;
pdd q[N], r;

int sign(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y) {
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    else return 1;
}

pdd operator+(pdd a, pdd b) {
    return { a.x + b.x,a.y + b.y };
}

pdd operator-(pdd a, pdd b) {
    return { a.x - b.x,a.y - b.y };
}

pdd operator*(pdd a, double t) {
    return { a.x * t,a.y * t };
}

```

```
pdd operator/(pdd a, double t) {
    return { a.x / t,a.y / t };
}

double operator*(pdd a, pdd b) {
    return a.x * b.y - a.y * b.x;
}

double operator&(pdd a, pdd b) {
    return a.x * b.x + a.y * b.y;
}

double area(pdd a, pdd b, pdd c) {
    return (b - a) * (c - a);
}

double get_len(pdd a) {
    return sqrt(a & a);
}

double get_dist(pdd a, pdd b) {
    return get_len(b - a);
}

double project(pdd a, pdd b, pdd c) {
    return ((c - a) & (b - a)) / get_len(b - a);
}

pdd rotate(pdd a, double theta) {
    return { a.x * cos(theta) + a.y * sin(theta),-a.x * sin(theta) + a.y *
cos(theta) };
}

pdd norm(pdd a) {
    return a / get_len(a);
}

bool on_segment(pdd p, pdd a, pdd b) {
    return !sign((p - a) * (p - b)) && sign((p - a) & (p - b)) <= 0;
}

pdd get_line_intersection(pdd p, pdd v, pdd q, pdd w) {
    pdd u = p - q;
    double t = w * u / (v * w);
    return p + v * t;
}

double get_circle_line_intersection(pdd a, pdd b, pdd& pa, pdd& pb) {
    pdd e = get_line_intersection(a, b - a, r, rotate(b - a, pi / 2));
    double mind = get_dist(r, e);
    if (!on_segment(e, a, b)) mind = min(get_dist(r, a), get_dist(r, b));
    if (dcmp(R, mind) <= 0) return mind;
    double len = sqrt(R * R - get_dist(r, e) * get_dist(r, e));
    pa = e + norm(a - b) * len;
    pb = e + norm(b - a) * len;
    return mind;
}

```

```
double get_sector(pdd a, pdd b) {
    double angle = acos((a & b) / get_len(a) / get_len(b));
    if (sign(a * b) < 0) angle = -angle;
    return R * R * angle / 2;
}

double get_circle_triangle_area(pdd a, pdd b) {
    double da = get_dist(r, a), db = get_dist(r, b);
    if (dcmp(R, da) >= 0 && dcmp(R, db) >= 0) return a * b / 2;
    if (!sign(a * b)) return 0;
    pdd pa, pb;
    double mind = get_circle_line_intersection(a, b, pa, pb);
    if (dcmp(R, mind) <= 0) return get_sector(a, b);
    if (dcmp(R, da) >= 0) return a * pb / 2 + get_sector(pb, b);
    if (dcmp(R, db) >= 0) return get_sector(a, pa) + pa * b / 2;
    return get_sector(a, pa) + pa * pb / 2 + get_sector(pb, b);
}

double work() {
    double res = 0;
    for (int i = 0; i < n; ++i)
        res += get_circle_triangle_area(q[i], q[(i + 1) % n]);
    return fabs(res);
}

int main() {
    while (~scanf("%lf%d", &R, &n)) {
        for (int i = 0; i < n; ++i) scanf("%lf%lf", &q[i].x, &q[i].y);
        printf("%.2lf\n", work());
    }
    return 0;
}
```

扫描线

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<vector>

#define x first
#define y second

using namespace std;
typedef long long ll;
typedef pair<int, int> PII;

const int N = 1010;

int n;
PII l[N], r[N];
PII q[N];

ll range_area(int a, int b) {
    int cnt = 0;
```

```
    for (int i = 0; i < n; ++i)
        if (l[i].x <= a && r[i].x >= b)
            q[cnt++] = { l[i].y,r[i].y };
    if (!cnt) return 0;
    sort(q, q + cnt);
    ll res = 0;
    int st = q[0].x, ed = q[0].y;
    for (int i = 1; i < cnt; ++i)
        if (q[i].x <= ed) ed = max(ed, q[i].y);
        else {
            res += ed - st;
            st = q[i].x, ed = q[i].y;
        }
    res += ed - st;
    return res * (b - a);
}

int main() {
    scanf("%d", &n);
    vector<int> xs;
    for (int i = 0; i < n; ++i) {
        scanf("%d%d%d%d", &l[i].x, &l[i].y, &r[i].x, &r[i].y);
        xs.push_back(l[i].x), xs.push_back(r[i].x);
    }
    sort(xs.begin(), xs.end());
    ll res = 0;
    for (int i = 0; i < xs.size() - 1; ++i) {
        if (xs[i] != xs[i + 1])
            res += range_area(xs[i], xs[i + 1]);
    }
    printf("%lld", res);
    return 0;
}
```

三角形面积并

使用扫描线解决，讲三角形划分成不同的图形后计算

```
#include<iostream>
#include<cstdio>
#include<vector>
#include<algorithm>
#include<cmath>

#define x first
#define y second

using namespace std;
typedef pair<double, double> pdd;

const int N = 110;
const double eps = 1e-8, INF = 1e6;

int n;
pdd tr[N][3];
```

```
pdd q[N];

int sign(double x) {
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

int dcmp(double x, double y) {
    if (fabs(x - y) < eps) return 0;
    if (x < y) return -1;
    return 1;
}

pdd operator+(pdd a, pdd b) {
    return { a.x + b.x,a.y + b.y };
}

pdd operator-(pdd a, pdd b) {
    return { a.x - b.x,a.y - b.y };
}

pdd operator*(pdd a, double t) {
    return { a.x * t,a.y * t };
}

double operator*(pdd a, pdd b) {
    return a.x * b.y - a.y * b.x;
}

double operator&(pdd a, pdd b) {
    return a.x * b.x + a.y * b.y;
}

bool on_segment(pdd p, pdd a, pdd b) {
    return sign((p - a) & (p - b)) <= 0;
}

pdd get_line_intersection(pdd p, pdd v, pdd q, pdd w) {
    if (!sign(v * w)) return { INF,INF };
    pdd u = p - q;
    double t = w * u / (v * w);
    pdd o = p + v * t;
    if (!on_segment(o, p, p + v) || !on_segment(o, q, q + w))
        return { INF,INF };
    return o;
}

double line_area(double a, int side) {
    int cnt = 0;
    for (int i = 0; i < n; ++i) {
        auto t = tr[i];
        if (dcmp(t[0].x, a) > 0 || dcmp(t[2].x, a) < 0) continue;
        if (!dcmp(t[0].x, a) && !dcmp(t[1].x, a)) {
            if (side) q[cnt++] = { t[0].y,t[1].y };
        }
        else if (!dcmp(t[2].x, a) && !dcmp(t[1].x, a)) {
            if (!side) q[cnt++] = { t[2].y,t[1].y };
```

```
        }
        else {
            double d[3];
            int u = 0;
            for (int j = 0; j < 3; ++j) {
                pdd o = get_line_intersection(t[j], t[(j + 1) % 3] - t[j], { a,-
INF }, { 0,2 * INF });
                if (dcmp(o.x, INF))
                    d[u++] = o.y;
            }
            if (u) {
                sort(d, d + u);
                q[cnt++] = { d[0],d[u - 1] };
            }
        }
    }
    if (!cnt) return 0;
    for (int i = 0; i < cnt; ++i)
        if (q[i].x > q[i].y)
            swap(q[i].x, q[i].y);
    sort(q, q + cnt);
    double res = 0, st = q[0].x, ed = q[0].y;
    for (int i = 1; i < cnt; ++i)
        if (q[i].x <= ed) ed = max(ed, q[i].y);
        else {
            res += ed - st;
            st = q[i].x, ed = q[i].y;
        }
    res += ed - st;
    return res;
}

double range_area(double a, double b) {
    return (line_area(a, 1) + line_area(b, 0)) * (b - a) / 2;
}

int main() {
    scanf("%d", &n);
    vector<double> xs;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 3; ++j) {
            scanf("%lf%lf", &tr[i][j].x, &tr[i][j].y);
            xs.push_back(tr[i][j].x);
        }
        sort(tr[i], tr[i] + 3);
    }

    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            for (int x = 0; x < 3; ++x)
                for (int y = 0; y < 3; ++y) {
                    pdd o = get_line_intersection(tr[i][x], tr[i][(x + 1) % 3] -
tr[i][x], tr[j][y], tr[j][(y + 1) % 3] - tr[j][y]);
                    if (dcmp(o.x, INF))
                        xs.push_back(o.x);
                }

    sort(xs.begin(), xs.end());
```

```
    double res = 0;
    for (int i = 0; i < xs.size() - 1; ++i)
        if (dcmp(xs[i], xs[i + 1]))
            res += range_area(xs[i], xs[i + 1]);
    printf("%.2lf", res);
    return 0;
}
```

辛普森积分

$$\int_a^b f(x) = \int_a^b F(x) = \frac{(b-a)(f(a)+4f(\frac{a+b}{2})+f(b))}{6}$$

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

using namespace std;

const double eps = 1e-12;

double f(double x) {
    return sin(x) / x;
}

double simpson(double l, double r) {
    double mid = (l + r) / 2;
    return (r - l) * (f(l) + f(r) + 4 * f(mid)) / 6;
}

double asr(double l, double r, double s) {
    double mid = (l + r) / 2;
    double left = simpson(l, mid), right = simpson(mid, r);
    if (fabs(left + right - s) < eps) return left + right;
    return asr(l, mid, left) + asr(mid, r, right);
}

int main() {
    double l, r;
    scanf("%lf%lf", &l, &r);
    printf("%lf", asr(l, r, simpson(l, r)));
    return 0;
}
```

动态规划

由于动态规划没有固定的套路，这里只摘录经典题型

动态规划一般步骤

1.确定维度

2.状态表示

3.状态转移

线性dp

数字三角形

状态表示：$f[i][j]$ 表示从左上角走到第 i 行第 j 列，和最大是多少

转移方程：$f[i][j] = max\{[i-1][j], f[i-1][j-1]\} + a[i][j]$

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 510, INF = 0x3f3f3f3f;

int f[N][N], a[N][N];

int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= i; j++)
            scanf("%d", &a[i][j]);
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= i + 1; j++) f[i][j] = -INF;
    f[1][1] = a[1][1];
    for (int i = 2; i <= n; i++)
        for (int j = 1; j <= i; j++)
            f[i][j] = max(f[i - 1][j - 1] + a[i][j], f[i - 1][j] + a[i][j]);
    int ans = 0;
    for (int i = 0; i <= n; i++) ans = max(ans, f[n][i]);
    printf("%d", ans);
    return 0;
}
```

最长上升子序列(LIS)

状态表示：$f[i]$ 表示以 $A[i]$ 结尾的 `LIS` 的长度

转移方程：$f[j] = \max\limits_{0<=j<i, A[j]<A[i]}\{f[j]+1\}$

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 1010;

int a[N], f[N], ans;

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

```
    for (int i = 1; i <= n; i++) {
        f[i] = 1;
        for (int j = 1; j < i; j++) {
            if (a[j] < a[i]) f[i] = max(f[i], f[j] + 1);
        }
        ans = max(ans, f[i]);
    }
    cout << ans;
    return 0;
}
```

最长公共子序列(LCS)

状态表示：$f[i][j]$ 表示前缀子串 `A[1~i]` 与 `B[1~j]` 的 `LCS` 的长度

转移方程 $f[i][j] = max\{f[i-1][j], f[i][j-1], f[i-1][j-1]+1\}$

```
#include<iostream>
#include<algorithm>
#include<cstdio>

using namespace std;

const int N = 1010;

int f[N][N];
char a[N], b[N];

int main() {
    int n, m;
    cin >> n >> m;
    scanf("%s%s", a + 1, b + 1);

    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j) {
            f[i][j] = max(f[i - 1][j], f[i][j - 1]);
            if (a[i] == b[j]) f[i][j] = max(f[i][j], f[i - 1][j - 1] + 1);
        }

    cout << f[n][m] << endl;
    return 0;
}
```

最长公共上升子序列(LCIS)

```
#include<iostream>
#include<algorithm>
#include<cstdio>

using namespace std;

const int N = 3010;

```

```
int a[N], b[N], f[N][N];

int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &a[i]);
    for (int i = 1; i <= n; i++) scanf("%d", &b[i]);
    for (int i = 1; i <= n; i++) {
        int val = 0;
        if (b[0] < a[i]) val = f[i - 1][0];
        for (int j = 1; j <= n; j++) {
            if (a[i] == b[j]) f[i][j] = val + 1;
            else f[i][j] = f[i - 1][j];
            if (b[j] < a[i]) val = max(val, f[i - 1][j]);
        }
    }
    int res = 0;
    for (int i = 1; i <= n; i++) res = max(res, f[n][i]);
    printf("%d", res);
    return 0;
}
```

背包问题

01背包

状态表示： $f[i][j]$ 表示从前 i 个物品中选，容量不超过 j 的最大值

状态转移： $f[i][j] = max\{f[i-1][j], f[i-1][j-v] + w[i]\}$

采用倒推的方式优化成一维

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 1010;

int f[N], v[N], w[N];

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 0; i < n; i++) cin >> v[i] >> w[i];
    for (int i = 0; i < n; i++) {
        for (int j = m; j >= v[i]; --j)
            f[j] = max(f[j], f[j - v[i]] + w[i]);
    }
    cout << f[m];
    return 0;
}
```

完全背包

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 1010;

int f[N], v[N], w[N];

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i];
    for (int i = 1; i <= n; i++) {
        for (int j = v[i]; j <= m; j++) {
            f[j] = max(f[j], f[j - v[i]] + w[i]);
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

多重背包

朴素版

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 1100;

int f[N][N], v[N], w[N], s[N];

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) cin >> v[i] >> w[i] >> s[i];
    for (int i = 1; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            for (int k = 0; k * v[i] <= j && k <= s[i]; k++)
                f[i][j] = max(f[i][j], f[i - 1][j - k * v[i]] + k * w[i]);
        }
    }
    cout << f[n][m];
    return 0;
}
```

01背包+二进制拆分优化

```cpp
#include<iostream>
#include<algorithm>
using namespace std;
```

```cpp

const int N = 12010, M = 2010;

int f[M], w[N], v[N];

int main() {
    int n, m;
    cin >> n >> m;
    int cnt = 0;
    for (int i = 1; i <= n; ++i) {
        int a, b, s;
        cin >> a >> b >> s;
        int k = 1;
        while (k <= s) {
            ++cnt;
            v[cnt] = k * a;
            w[cnt] = k * b;
            s -= k;
            k *= 2;
        }
        if (s) {
            ++cnt;
            v[cnt] = a * s;
            w[cnt] = b * s;
        }
    }
    n = cnt;
    for (int i = 1; i <= n; ++i)
        for (int j = m; j >= v[i]; --j)
            f[j] = max(f[j], f[j - v[i]] + w[i]);

    cout << f[m] << endl;
    return 0;
}
```

单调队列优化

```cpp
#include<iostream>
#include<algorithm>
#include<cstring>
using namespace std;

const int N = 20010;

int f[N], g[N], q[N];
int n, m;
int v, w, s;

int main() {
    cin >> n >> m;

    for (int i = 0; i < n; ++i) {
        cin >> v >> w >> s;
        memcpy(g, f, sizeof f);
        for (int j = 0; j < v; ++j) {
            int hh = 0, tt = -1;
            for (int k = j; k <= m; k += v) {
```

```
                if (hh <= tt && k - s * v > q[hh]) hh++;
                while (hh <= tt && g[q[tt]] - (q[tt] - j) / v * w <= g[k] - (k - j) / v * w) tt--;
                q[++tt] = k;
                f[k] = g[q[hh]] + (k - q[hh]) / v * w;
            }
        }
    }
    cout << f[m] << endl;

    return 0;
}
```

混合背包

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 1010;

int n, m;
int f[N];
int v, w, s;

int main() {
    cin >> n >> m;
    for (int i = 0; i < n; ++i) {
        cin >> v >> w >> s;
        if (!s)
            for (int j = v; j <= m; j++)
                f[j] = max(f[j], f[j - v] + w);
        else {
            if (s == -1) s = 1;
            for (int k = 1; k <= s; k *= 2) {
                for (int j = m; j >= k * v; j--)
                    f[j] = max(f[j], f[j - k * v] + k * w);
                s -= k;
            }

            if (s)
                for (int j = m; j >= s * v; j--)
                    f[j] = max(f[j], f[j - s * v] + s * w);
        }
    }
    cout << f[m] << endl;
    return 0;
}
```

二维费用背包

状态表示： $f[i][j][k]$ 表示从前 i 个物品中选，体积不超过 j，费用不超过 k 的最大值

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 1001, M = 101;

int f[N][M][M], v[N], w[N], m[N];

int main() {
    int a, b, c;
    cin >> a >> b >> c;
    for (int i = 1; i <= a; i++) cin >> v[i] >> m[i] >> w[i];
    for (int i = 1; i <= a; i++) {
        for (int j = 0; j <= b; j++) {
            for (int k = 0; k <= c; k++) {
                f[i][j][k] = f[i - 1][j][k];
                if (v[i] <= j && m[i] <= k)
                    f[i][j][k] = max(f[i][j][k], f[i - 1][j - v[i]][k - m[i]] + w[i]);
            }
        }
    }
    cout << f[a][b][c];
    return 0;
}
```

分组背包

```cpp
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 110;

int f[N], v[N][N], w[N][N], s[N];

int main() {
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; ++i) {
        cin >> s[i];
        for (int j = 0; j < s[i]; ++j) cin >> v[i][j] >> w[i][j];
    }

    for (int i = 1; i <= n; ++i)
        for (int j = m; j >= 0; --j)
            for (int k = 0; k < s[i]; ++k)
                if (j >= v[i][k])
                    f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);

    cout << f[m] << endl;
    return 0;
}
```

有依赖的背包

```cpp
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 110;

int v[N], w[N], to[N], head[N], ne[N], tot;
int f[N][N], n, m, root;

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

void dfs(int u) {
    for (int i = head[u]; i != -1; i = ne[i]) {
        dfs(to[i]);
        for (int j = m - v[u]; j >= 0; j--) for (int k = 0; k <= j; k++) f[u][j] = max(f[u][j], f[u][j - k] + f[to[i]][k]);
    }
    for (int i = m; i >= v[u]; i--) f[u][i] = f[u][i - v[u]] + w[u];
    for (int i = 0; i < v[u]; i++) f[u][i] = 0;
}

int main() {
    memset(head, -1, sizeof head);
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) {
        int t;
        scanf("%d%d%d", &v[i], &w[i], &t);
        if (t == -1) root = i;
        else add(t, i);
    }
    dfs(root);
    printf("%d", f[root][m]);
    return 0;
}
```

01背包求方案数

```cpp
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 1010, mod = 1e9 + 7;

```

```
int f[N], g[N];

int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    memset(f, -0x3f, sizeof f);
    f[0] = 0;
    g[0] = 1;

    for (int i = 0; i < n; i++) {
        int v, w;
        scanf("%d%d", &v, &w);
        for (int j = m; j >= v; j--) {
            int maxv = max(f[j], f[j - v] + w);
            int cnt = 0;
            if (f[j] == maxv) cnt += g[j];
            if (f[j - v] + w == maxv) cnt += g[j - v];
            g[j] = cnt % mod;
            f[j] = maxv;
        }
    }

    int res = 0;
    for (int i = 0; i <= m; i++) res = max(res, f[i]);
    int cnt = 0;
    for (int i = 0; i <= m; i++) {
        if (f[i] == res)
            cnt = (cnt + g[i]) % mod;
    }
    printf("%d", cnt);
    return 0;
}
```

背包求具体方案

```
#include<iostream>
#include<algorithm>
#include<cstdio>
using namespace std;
const int N = 1010;
int f[N][N], v[N], w[N];
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%d%d", &v[i], &w[i]);
    for (int i = n; i >= 1; i--) {
        for (int j = 0; j <= m; j++) {
            f[i][j] = f[i + 1][j];
            if (j >= v[i])
                f[i][j] = max(f[i][j], f[i + 1][j - v[i]] + w[i]);
        }
    }

    int j = m;
    for (int i = 1; i <= n; i++) {
```

```
        if (j >= v[i] && f[i][j] == f[i + 1][j - v[i]] + w[i]) {
            printf("%d ", i);
            j -= v[i];
        }
    }
    return 0;
}
```

区间dp

石子合并

状态表示：$f[i][j]$ 表示将 i 和 j 合并成一堆的方案集合

状态转移：$f[i][j] = \min_{i<j=k<=j-1}\{f[i][j], f[i][k] + f[k+1][j] + s[j] - s[i-1]\}$

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 310;

int s[N], f[N][N];

int main() {
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) { cin >> s[i]; s[i] += s[i - 1]; }
    for (int len = 2; len <= n; len++) {
        for (int i = 1; i + len - 1 <= n; i++) {
            int l = i, r = i + len - 1;
            f[l][r] = 1e9;
            for (int k = l; k < r; k++)
                f[l][r] = min(f[l][r], f[l][k] + f[k + 1][r] + s[r] - s[l - 1]);
        }
    }
    printf("%d", f[1][n]);
    return 0;
}
```

能量项链

对于环形dp问题，可以破环成链，把链复制一份接到后面枚举即可

```
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 210;

int n;
```

```cpp
int w[N];
int f[N][N];

int main() {
    cin >> n;

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

    for (int len = 3; len <= n + 1; len++)
        for (int l = 1; l + len - 1 <= 2 * n; l++) {
            int r = l + len - 1;
            for (int k = l + 1; k < r; k++)
                f[l][r] = max(f[l][r], f[l][k] + f[k][r] + w[l] * w[r] * w[k]);
        }

    int res = 0;
    for (int i = 1; i <= n; i++) res = max(res, f[i][i + n]);
    cout << res << endl;

    return 0;
}
```

树形dp

没有上司的舞会

状态表示:

$f[i][0]$ 表示不选当前节点

$f[i][1]$ 表示选当前节点

```cpp
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 6010;

int to[N], ne[N], head[N], tot, happy[N], f[N][2];
int n;
bool has_father[N];

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

void dp(int u) {
    f[u][1] = happy[u];
    for (int i = head[u]; i != -1; i = ne[i]) {
        int j = to[i];
```

```
        dp(j);
        f[u][1] += f[j][0];
        f[u][0] += max(f[j][0], f[j][1]);
    }
}

int main() {
    memset(head, -1, sizeof head);
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) scanf("%d", &happy[i]);
    for (int i = 0; i < n - 1; i++) {
        int a, b;
        scanf("%d%d", &a, &b);
        has_father[a] = true;
        add(b, a);
    }
    int root = 1;
    while (has_father[root]) ++root;
    dp(root);
    printf("%d", max(f[root][0], f[root][1]));
    return 0;
}
```

状压dp

蒙德里安的梦想

状态表示：$f[i][j]$ 表示当前在第 i 行，状态为 j

已经完成上一行，用上一行来确定当前行

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

using namespace std;
typedef long long ll;

const int N = 12, M = 1 << 11;

int n, m;
ll f[N][M];
bool in[M];

int main() {
    while (cin >> n >> m, n || m) {
        for (int i = 0; i < 1 << m; ++i) { // 统计每一个状态中是否有一段连续的偶数个1
            bool odd = false, cnt = false;
            for (int j = 0; j < m; ++j)
                if (i >> j & 1) odd |= cnt, cnt = 0;
                else cnt ^= 1;
            in[i] = odd | cnt ? 0 : 1;
        }

        f[0][0] = 1;
```

```
        for (int i = 1; i <= n; ++i)
            for (int j = 0; j < 1 << m; ++j) {
                f[i][j] = 0;
                for (int k = 0; k < 1 << m; ++k)
                    if ((j & k) == 0 && in[j | k])
                        f[i][j] += f[i - 1][k];
            }
        cout << f[n][0] << endl;
    }
    return 0;
}
```

已经完成当前行，用当前行来生成下一行

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>

using namespace std;
typedef long long ll;

const int N = 12, M = 1 << 11;

int n, m;
ll f[N][M];
vector<int> ne;

void dfs(int state, int x, int ns) {
    if (x == m) {
        ne.push_back(ns);
        return;
    }

    if (state >> x & 1) {
        dfs(state, x + 1, ns);
        return;
    }

    dfs(state, x + 1, ns | 1 << x);

    if (x < m - 1 && (state >> (x + 1) & 1) == 0)
        dfs(state, x + 2, ns);
}

int main() {
    while (cin >> n >> m, n || m) {
        memset(f, 0, sizeof f);
        f[0][0] = 1;

        for (int i = 0; i < n; ++i)
            for (int j = 0; j < 1 << m; ++j)
                if (f[i][j] > 0) {
                    ne.clear();
                    dfs(j, 0, 0);
                    for (int k = 0; k < ne.size(); ++k)
```

```
                        f[i + 1][ne[k]] += f[i][j];
                }
        cout << f[n][0] << endl;
    }
    return 0;
}
```

数位dp

计数问题

画树形图分析，从高到低考虑每一位填什么数

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<vector>

using namespace std;

int get(vector<int> num, int l, int r) {
    int res = 0;
    for (int i = l; i >= r; --i) res = res * 10 + num[i];
    return res;
}

int power10(int k) {
    int res = 1;
    while (k--) res *= 10;
    return res;
}

int count(int n, int x) {
    if (!n) return 0;

    vector<int> num;
    while (n) {
        num.push_back(n % 10);
        n /= 10;
    }

    n = num.size();
    int res = 0;
    for (int i = n - 1 - (!x); i >= 0; --i) {
        if (i < n - 1) {
            res += get(num, n - 1, i + 1) * power10(i);
            if (!x) res -= power10(i);
        }
        if (num[i] == x) res += get(num, i - 1, 0) + 1;
        else if (num[i] > x) res += power10(i);
    }
    return res;
}

int main() {
    int a, b;
```

```
    while (cin >> a >> b, a || b) {
        if (a > b) swap(a, b);
        for (int i = 0; i <= 9; ++i)
            cout << count(b, i) - count(a - 1, i) << " ";
        cout << endl;
    }
    return 0;
}
```

基环树dp

岛屿

先找环，然后把环断开，用类似于处理环形区间dp的方法处理

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

using namespace std;
typedef long long ll;
const int N = 1000010, M = N * 2;

int n;
int h[N], to[M], ne[M], w[M], tot;
int fa[N], fw[N], q[N];
int cir[N], ed[N], cnt;
ll s[N], d[M], sum[M];
bool vis[N], ins[N];
ll ans;

inline void add(int u, int v, int c) {
    to[tot] = v, w[tot] = c, ne[tot] = h[u], h[u] = tot++;
}

void dfs_cir(int u, int from) {
    vis[u] = ins[u] = true;
    for (int i = h[u]; ~i; i = ne[i]) {
        if (i == (from ^ 1)) continue;
        int j = to[i];
        fa[j] = u, fw[j] = w[i];
        if (!vis[j]) dfs_cir(j, i);
        else if (ins[j]) {
            cnt++;
            ed[cnt] = ed[cnt - 1];
            ll sum = w[i];
            for (int k = u; k != j; k = fa[k]) {
                s[k] = sum;
                sum += fw[k];
                cir[++ed[cnt]] = k;
            }
            s[j] = sum, cir[++ed[cnt]] = j;
        }
    }
    ins[u] = false;
```

```cpp
}

ll dfs_d(int u) {
    vis[u] = true;
    ll d1 = 0, d2 = 0;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (vis[j]) continue;
        ll dist = dfs_d(j) + w[i];
        if (dist >= d1) d2 = d1, d1 = dist;
        else if (dist > d2) d2 = dist;
    }
    ans = max(ans, d1 + d2);
    return d1;
}

int main() {
    memset(h, -1, sizeof h);
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i) {
        int a, b;
        scanf("%d%d", &a, &b);
        add(i, a, b), add(a, i, b);
    }

    for (int i = 1; i <= n; ++i)
        if (!vis[i])
            dfs_cir(i, -1);

    memset(vis, 0, sizeof vis);
    for (int i = 1; i <= ed[cnt]; ++i) vis[cir[i]] = true;

    ll res = 0;
    for (int i = 1; i <= cnt; ++i) {
        ans = 0;
        int sz = 0;
        for (int j = ed[i - 1] + 1; j <= ed[i]; ++j) {
            int k = cir[j];
            d[sz] = dfs_d(k);
            sum[sz] = s[k];
            sz++;
        }
        for (int j = 0; j < sz; ++j)
            d[sz + j] = d[j], sum[sz + j] = sum[j] + sum[sz - 1];
        int hh = 0, tt = -1;
        for (int j = 0; j < sz * 2; ++j) {
            if (hh <= tt && j - q[hh] >= sz) hh++;
            if (hh <= tt) ans = max(ans, d[j] + sum[j] + d[q[hh]] - sum[q[hh]]);
            while (hh <= tt && d[q[tt]] - sum[q[tt]] <= d[j] - sum[j]) tt--;
            q[++tt] = j;
        }
        res += ans;
    }
    printf("%lld", res);
    return 0;
}
```

创世纪

基环树森林，做法类似 `没有上司的舞会`

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 1000010, INF = 0x3f3f3f3f;

int n;
int h[N], to[N], rm[N], ne[N], idx;
int f1[N][2], f2[N][2];
bool vis[N], ins[N];
int ans;

inline void add(int u, int v) {
    to[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void dfs_f(int u, int ap, int f[][2]) {
    for (int i = h[u]; ~i; i = ne[i]) {
        if (rm[i]) continue;
        int j = to[i];
        dfs_f(j, ap, f);
        f[u][0] += max(f[j][0], f[j][1]);
    }
    if (u == ap) f[u][1] = f[u][0] + 1, f[u][0] = -INF;
    else {
        f[u][1] = -INF;
        for (int i = h[u]; ~i; i = ne[i]) {
            if (rm[i]) continue;
            int j = to[i];
            f[u][1] = max(f[u][1], f[u][0] - max(f[j][0], f[j][1]) + f[j][0] + 1);
        }
    }
}

void dfs_c(int u, int from) {
    vis[u] = ins[u] = true;
    for (int i = h[u]; ~i; i = ne[i]) {
        int j = to[i];
        if (!vis[j]) dfs_c(j, i);
        else if (ins[j]) {
            rm[i] = 1;
            dfs_f(j, -1, f1);
            dfs_f(j, u, f2);
            ans += max(max(f1[j][0], f1[j][1]), f2[j][0]);
        }
    }
    ins[u] = false;
}

```

```
int main() {
    scanf("%d", &n);
    memset(h, -1, sizeof h);
    for (int i = 1; i <= n; ++i) {
        int a;
        scanf("%d", &a);
        add(a, i);
    }

    for (int i = 1; i <= n; ++i)
        if (!vis[i])
            dfs_c(i, -1);

    printf("%d", ans);
    return 0;
}
```

插头dp

详见陈丹琦《基于连通性状态压缩的动态规划问题》

本质是状压dp，常用的表示法是最小表示法和括号表示法

对于不同的插头，分类讨论即可

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

using namespace std;
typedef long long ll;
typedef long long LL;

const int N = 50000, M = N * 2 + 7;

int n, m, end_x, end_y;
int g[20][20], q[2][N], cnt[N];
int h[2][M];
ll v[2][M];

inline int find(int cur, int x) {
    int t = x % M;
    while (h[cur][t] != -1 && h[cur][t] != x)
        if (++t == M)
            t = 0;
    return t;
}

inline void insert(int cur, int state, ll w) {
    int t = find(cur, state);
    if (h[cur][t] == -1) {
        h[cur][t] = state, v[cur][t] = w;
        q[cur][++cnt[cur]] = t;
    }
    else v[cur][t] += w;
```

```
}

inline int get(int state, int k) {
    return state >> k * 2 & 3;
}

inline int set(int k, int v) {
    return v * (1 << k * 2);
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i) {
        char str[20];
        scanf("%s", str + 1);
        for (int j = 1; j <= m; ++j)
            if (str[j] == '.') {
                g[i][j] = 1;
                end_x = i, end_y = j;
            }
    }

    ll res = 0;
    memset(h, -1, sizeof h);
    int cur = 0;
    insert(cur, 0, 1);
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= cnt[cur]; j++)
            h[cur][q[cur][j]] <<= 2;
        for (int j = 1; j <= m; ++j) {
            //cout<<i<<" "<<j<<endl;
            int last = cur;
            cur ^= 1, cnt[cur] = 0;
            memset(h[cur], -1, sizeof h[cur]);
            for (int k = 1; k <= cnt[last]; ++k) {
                int state = h[last][q[last][k]];
                ll w = v[last][q[last][k]];
                int x = get(state, j - 1), y = get(state, j);
                if (!g[i][j]) {
                    if (!x && !y) insert(cur, state, w);
                }
                else if (!x && !y) {
                    if (g[i + 1][j] && g[i][j + 1])
                        insert(cur, state + set(j - 1, 1) + set(j, 2), w);
                }
                else if (!x && y) {
                    if (g[i][j + 1]) insert(cur, state, w);
                    if (g[i + 1][j]) insert(cur, state + set(j - 1, y) - set(j,
y), w);
                }
                else if (x && !y) {
                    if (g[i][j + 1]) insert(cur, state - set(j - 1, x) + set(j,
x), w);
                    if (g[i + 1][j]) insert(cur, state, w);
                }
                else if (x == 1 && y == 1) {
                    for (int u = j + 1, s = 1;; u++) {
                        int z = get(state, u);
```

```
                        if (z == 1) s++;
                        else if (z == 2) {
                            if (--s == 0) {
                                insert(cur, state - set(j - 1, x) - set(j, y) - set(u, 1), w);
                                break;
                            }
                        }
                    }
                }
                else if (x == 2 && y == 2) {
                    for (int u = j - 2, s = 1;; u--) {
                        int z = get(state, u);
                        if (z == 2) s++;
                        else if (z == 1) {
                            if (--s == 0) {
                                insert(cur, state - set(j - 1, x) - set(j, y) + set(u, 1), w);
                                break;
                            }
                        }
                    }
                }
                else if (x == 2 && y == 1) {
                    insert(cur, state - set(j - 1, x) - set(j, y), w);
                }
                else if (i == end_x && j == end_y) {
                    res += w;
                }
            }
        }
    }
    cout << res;
    return 0;
}
```

dp的优化方法

1.数据结构优化，一般用单调队列

2.斜率优化

3.四边形不等式优化

搜索

Flood Fill

求连通块

```
#include<iostream>
#include<cstdio>

using namespace std;

```

```cpp
#define x first
#define y second

const int N = 1010;

char g[N][N];
bool vis[N][N];
int n, m;
pair<int, int> q[N * N];

void bfs(int sx, int sy) {
    int head = 0, tail = 0;
    q[0] = { sx,sy };
    vis[sx][sy] = true;
    while (head <= tail) {
        pair<int, int> t = q[head++];
        for (int i = t.x - 1; i <= t.x + 1; i++)
            for (int j = t.y - 1; j <= t.y + 1; j++) {
                if (i == t.x && j == t.y) continue;
                if (i < 0 || i >= n || j < 0 || j >= m) continue;
                if (g[i][j] == '.' || vis[i][j]) continue;
                vis[i][j] = true;
                q[++tail] = { i,j };
            }
    }
}

int main() {
    int cnt = 0;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) scanf("%s", g[i]);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (g[i][j] == 'W' && !vis[i][j]) {
                bfs(i, j);
                cnt++;
            }
    printf("%d", cnt);
    return 0;
}
```

最短路模型

`bfs` 第一次搜到的就是最短路

多源bfs

```cpp
// 矩阵距离
#include<iostream>
#include<cstdio>
#include<cstring>

#define x first
#define y second
```

```cpp

using namespace std;

const int N = 1010;

pair<int, int> q[N * N];
char g[N][N];
int n, m, dist[N][N];
int dx[] = { -1,0,1,0 }, dy[] = { 0,1,0,-1 };

void bfs() {
    memset(dist, -1, sizeof dist);
    int head = 0, tail = -1;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= m; j++)
            if (g[i][j] == '1') dist[i][j] = 0, q[++tail] = { i,j };
    while (head <= tail) {
        pair<int, int> t = q[head++];
        for (int i = 0; i < 4; i++) {
            int x = t.x + dx[i], y = t.y + dy[i];
            if (x<1 || x>n || y<1 || y>m) continue;
            if (dist[x][y] != -1) continue;
            dist[x][y] = dist[t.x][t.y] + 1;
            q[++tail] = { x,y };
        }
    }
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; i++) scanf("%s", g[i] + 1);
    bfs();
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) printf("%d ", dist[i][j]);
        printf("\n");
    }
    return 0;
}
```

双端队列广搜

```cpp
// 电路维修
#include<iostream>
#include<cstdio>
#include<cstring>
#include<deque>

#define x first
#define y second

using namespace std;
typedef pair<int, int> PII;

const int N = 510;

```

```cpp
int n, m;
char g[N][N];
int dist[N][N];
bool vis[N][N];

int bfs() {
    memset(dist, 0x3f, sizeof dist);
    memset(vis, 0, sizeof vis);
    dist[0][0] = 0;
    deque<PII> q;
    q.push_back({ 0,0 });

    char cs[] = "\\/\\/";
    int dx[] = { -1,-1,1,1 }, dy[] = { -1,1,1,-1 };
    int ix[] = { -1,-1,0,0 }, iy[] = { -1,0,0,-1 };

    while (q.size()) {
        PII t = q.front();
        q.pop_front();

        if (vis[t.x][t.y]) continue;
        vis[t.x][t.y] = true;

        for (int i = 0; i < 4; ++i) {
            int a = t.x + dx[i], b = t.y + dy[i];
            if (a<0 || a>n || b<0 || b>m) continue;

            int ca = t.x + ix[i], cb = t.y + iy[i];
            int d = dist[t.x][t.y] + (g[ca][cb] != cs[i]);

            if (d < dist[a][b]) {
                dist[a][b] = d;
                if (g[ca][cb] != cs[i]) q.push_back({ a,b });
                else q.push_front({ a,b });
            }
        }
    }
    return dist[n][m];
}

int main() {
    int T;
    scanf("%d", &T);
    while (T--) {
        scanf("%d%d", &n, &m);
        for (int i = 0; i < n; ++i) scanf("%s", g[i]);

        int res = bfs();

        if (res == 0x3f3f3f3f) puts("NO SOLUTION");
        else printf("%d\n", res);
    }
    return 0;
}
```

双向广搜

分别从起点和终点开始搜索

```cpp
// 字串变换
#include<iostream>
#include<cstdio>
#include<cstring>
#include<queue>
#include<map>

using namespace std;

const int N = 6;

int n;
string a[N], b[N];

int extend(queue<string>& q, map<string, int>& da, map<string, int>& db, string a[], string b[]) {
    for (int k = 0, sk = q.size(); k < sk; ++k) {
        string t = q.front();
        q.pop();

        for (int i = 0; i < t.size(); ++i)
            for (int j = 0; j < n; ++j) {
                if (t.substr(i, a[j].size()) == a[j]) {
                    string state = t.substr(0, i) + b[j] + t.substr(i + a[j].size());
                    if (da.count(state)) continue;
                    if (db.count(state)) return da[t] + 1 + db[state];
                    da[state] = da[t] + 1;
                    q.push(state);
                }
            }
    }
    return 11;
}

int bfs(string A, string B) {
    queue<string> qa, qb;
    map<string, int> da, db;
    qa.push(A), qb.push(B);
    da[A] = 0, db[B] = 0;

    while (qa.size() && qb.size()) {
        int t;
        if (qa.size() <= qb.size()) t = extend(qa, da, db, a, b);
        else t = extend(qb, db, da, b, a);
        if (t <= 10) return t;
    }
    return 11;
}

int main() {
    string A, B;
    cin >> A >> B;
```

```
    while (cin >> a[n] >> b[n]) n++;

    int step = bfs(A, B);
    if (step > 10) puts("NO ANSWER!");
    else printf("%d", step);
    return 0;
}
```

A*

带有估价函数的广搜

```
// 第k短路
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>

#define x first
#define y second

using namespace std;
typedef pair<int, int> PII;
typedef pair<int, PII> PIII;

const int N = 1010, M = 200010;

int n, m, S, T, K;
int h[N], rh[N], to[M], ne[M], w[M], idx;
int dist[N], cnt[N];
bool vis[N];

inline void read(int& x) {
    x = 0; int f = 1; char c = getchar();
    while (!isdigit(c)) { if (c == '-')f = -1; c = getchar(); }
    while (isdigit(c)) { x = (x << 1) + (x << 3) + (c ^ 48); c = getchar(); }
    x *= f;
}

inline void add(int h[], int u, int v, int c) {
    to[idx] = v, w[idx] = c, ne[idx] = h[u], h[u] = idx++;
}

void dijkstra() {
    priority_queue<PII, vector<PII>, greater<PII> > heap;
    memset(dist, 0x3f, sizeof dist);
    heap.push({ 0,T });
    dist[T] = 0;

    while (heap.size()) {
        PII t = heap.top();
        heap.pop();

        int ver = t.y;
```

```
        if (vis[ver]) continue;
        vis[ver] = true;

        for (int i = rh[ver]; ~i; i = ne[i]) {
            int j = to[i];
            if (dist[j] > dist[ver] + w[i]) {
                dist[j] = dist[ver] + w[i];
                heap.push({ dist[j],j });
            }
        }
    }
}

int Astar() {
    priority_queue<PIII, vector<PIII>, greater<PIII> > heap;
    heap.push({ dist[S],{0,S} });

    while (heap.size()) {
        PIII t = heap.top();
        heap.pop();

        int ver = t.y.y, dis = t.y.x;
        cnt[ver]++;
        if (cnt[T] == K) return dis;

        for (int i = h[ver]; ~i; i = ne[i]) {
            int j = to[i];
            if (cnt[j] < K)
                heap.push({ dis + w[i] + dist[j],{dis + w[i],j} });
        }
    }
    return -1;
}

int main() {
    memset(h, -1, sizeof h);
    memset(rh, -1, sizeof rh);
    read(n), read(m);
    while (m--) {
        int a, b, c;
        read(a), read(b), read(c);
        add(h, a, b, c);
        add(rh, b, a, c);
    }
    read(S), read(T), read(K);
    if (S == T) K++;

    dijkstra();
    printf("%d", Astar());
    return 0;
}
```

IDA*

带有估价函数的深搜

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

using namespace std;

const int N=15;

int n;
int q[N],w[5][N];

inline int f(){
    int cnt=0;
    for(int i=0;i+1<n;++i)
        if(q[i+1] != q[i]+1)
            cnt++;
    return (cnt+2)/3;
}

inline bool check(){
    for(int i=0;i+1<n;++i)
        if(q[i+1] != q[i]+1)
            return false;
    return true;
}

bool dfs(int dep,int max_dep){
    if(dep+f() > max_dep) return false;
    if(check()) return true;

    for(int len=1;len<=n;++len)
        for(int l=0;l+len-1<n;++l){
            int r=l+len-1;
            for(int k=r+1;k<n;++k){
                memcpy(w[dep],q,sizeof q);
                int x,y;
                for(x=r+1,y=l;x<=k;++x,++y) q[y]=w[dep][x];
                for(x=l;x<=r;++x,++y) q[y]=w[dep][x];
                if(dfs(dep+1,max_dep)) return true;
                memcpy(q,w[dep],sizeof q);
            }
        }
    return false;
}

int main(){
    int T;
    cin>>T;
    while(T--){
        cin>>n;
        for(int i=0;i<n;++i) cin>>q[i];

```

```
        int depth=0;
        while(depth<5 && !dfs(0,depth)) depth++;
        if(depth >= 5) puts("5 or more");
        else cout<<depth<<endl;
    }
    return 0;
}
```

模拟退火

模拟退火状态分为两种：

1.新状态比原来状态更优，那么接受新状态

2.新状态更差，那么以一定概率接受该状态

最小值最优化： $exp(-\frac{\Delta f}{kT})$

最大值最优化： $exp(\frac{\Delta f}{kT})$

```
// 平衡点
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstdlib>
#include<cmath>
#include<ctime>

using namespace std;

const int N = 1010;
const double dT = 0.98618;

int n;
double ansx, ansy, answ;
int xsum, ysum;

struct Node {
    int x, y, w;
}node[N];

double energy(double x, double y) { // 计算势能
    double res = 0, dx, dy;
    for (int i = 1; i <= n; i++)
    {
        dx = x - node[i].x;
        dy = y - node[i].y;
        res += sqrt(dx * dx + dy * dy) * node[i].w;
    }
    return res;
}

void sa() {
    double T = 3000;
    while (T > 1e-15) {
        double ex = ansx + (rand() * 2 - RAND_MAX) * T;
```

```
        double ey = ansy + (rand() * 2 - RAND_MAX) * T;
        double ew = energy(ex, ey);
        double dE = ew - answ;
        if (dE < 0) {
            ansx = ex;
            ansy = ey;
            answ = ew;
        }
        else if (exp(-dE / T) * RAND_MAX > rand()) { // 以一个多项式概率接受该答案
            ansx = ex;
            ansy = ey;
        }
        T *= dT; // 降温
    }

}
void solve() { // 玄学保佑
    sa(); sa(); sa(); sa(); sa();
}
int main() {
    srand(19260817); // 钦定随机数种子
    scanf("%d", &n);
    for (int i = 1; i <= n; i++) {
        scanf("%d%d%d", &node[i].x, &node[i].y, &node[i].w);
        xsum += node[i].x, ysum += node[i].y;
    }
    ansx = xsum / 2, ansy = ysum / n; // 钦定原始坐标
    answ = energy(ansx, ansy); // 钦定一个原始能量
    solve();
    printf("%.3lf %.3lf", ansx, ansy);
    return 0;
}
```

```
// 均分数据
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<cstring>

using namespace std;
const int N = 25, M = 10;

int n, m;
int w[N], s[M];
double ans = 1e8, dT = 0.9618;

double clac() {
    memset(s, 0, sizeof s);
    for (int i = 0; i < n; ++i) {
        int k = 0;
        for (int j = 0; j < m; ++j)
            if (s[j] < s[k])
                k = j;
        s[k] += w[i];
```

```
    }

    double avg = 0;
    for (int i = 0; i < m; ++i) avg += (double)s[i] / m;
    double res = 0;
    for (int i = 0; i < m; ++i)
        res += (s[i] - avg) * (s[i] - avg);
    res = sqrt(res / m);
    ans = min(ans, res);
    return res;
}

void sa() {
    random_shuffle(w, w + n);

    double T = 1e6;
    while (T > 1e-6) {
        int a = rand() % n, b = rand() % n;
        double x = clac();
        swap(w[a], w[b]);
        double y = clac();
        double delta = y - x;
        if (exp(-delta / T) < (double)rand() / RAND_MAX)
            swap(w[a], w[b]);
        T *= dT;
    }
}

int main() {
    srand(time(NULL));
    cin >> n >> m;
    for (int i = 0; i < n; ++i) cin >> w[i];
    sa();
    printf("%.2lf", ans);
    return 0;
}
```

爬山法

梯度下降

```
// 球形空间产生器
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>

using namespace std;

const int N = 15;

int n;
double d[N][N];
double ans[N], dist[N], delta[N];

```

```cpp
void calc() {
    double avg = 0;
    for (int i = 0; i < n + 1; ++i) {
        dist[i] = delta[i] = 0;
        for (int j = 0; j < n; ++j)
            dist[i] += (d[i][j] - ans[j]) * (d[i][j] - ans[j]);
        dist[i] = sqrt(dist[i]);
        avg += dist[i] / (n + 1);
    }
    for (int i = 0; i < n + 1; ++i)
        for (int j = 0; j < n; ++j)
            delta[j] += (dist[i] - avg) * (d[i][j] - ans[j]) / avg;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n + 1; ++i)
        for (int j = 0; j < n; ++j) {
            scanf("%lf", &d[i][j]);
            ans[j] += d[i][j] / (n + 1);
        }
    for (double T = 1e4; T > 1e-6; T *= 0.99995) {
        calc();
        for (int i = 0; i < n; ++i)
            ans[i] += delta[i] * T;
    }
    for (int i = 0; i < n; ++i) printf("%.3lf ", ans[i]);
    return 0;
}
```

字符串

字符串哈希

$$hash(s) = \sum_{i=1}^{l} s[i] * b^{l-i} (\mod P)$$

P 一般取 131 或 13331

```cpp
#include<iostream>

using namespace std;
typedef unsigned long long ull;

const int N = 1e5 + 10, P = 131;

ull h[N], p[N];
char str[N];

ull find(int l, int r) {
    return h[r] - h[l - 1] * p[r - l + 1];
}

int main() {
    int n, m;
```

```
    scanf("%d%d%s", &n, &m, str + 1);
    p[0] = 1;
    for (int i = 1; i <= n; ++i) {
        h[i] = h[i - 1] * P + str[i];
        p[i] = p[i - 1] * P;
    }

    int l1, l2, r1, r2;
    while (m--) {
        scanf("%d%d%d%d", &l1, &r1, &l2, &r2);
        if (find(l1, r1) == find(l2, r2)) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

KMP

```
#include<iostream>
#include<cstdio>

using namespace std;

const int N = 1e6 + 10;

int n, m, ne[N];
char p[N], s[N];

int main() {
    scanf("%d%s%d%s", &n, p + 1, &m, s + 1);

    for (int i = 2, j = 0; i <= n; ++i) {
        while (j && p[i] != p[j + 1]) j = ne[j];
        if (p[i] == p[j + 1]) j++;
        ne[i] = j;
    }

    for (int i = 1, j = 0; i <= m; ++i) {
        while (j && s[i] != p[j + 1]) j = ne[j];
        if (s[i] == p[j + 1]) j++;
        if (j == n) {
            printf("%d ", i - n);
            j = ne[j];
        }
    }
    return 0;
}
```

Trie

```cpp
#include<iostream>
#include<cstdio>
#include<cstring>

using namespace std;

const int N = 1e5 + 10;

int t[N][26], cnt[N], idx;

void insert(char a[]) {
    int p = 1;
    for (int i = 0; a[i]; i++) {
        if (!t[p][a[i] - 'a']) t[p][a[i] - 'a'] = ++idx;
        p = t[p][a[i] - 'a'];
    }
    cnt[p]++;

}

int find(char a[]) {
    int p = 1;
    for (int i = 0; a[i]; ++i) {
        if (!t[p][a[i] - 'a']) return 0;
        p = t[p][a[i] - 'a'];
    }
    return cnt[p];
}

int main() {
    int m;
    cin >> m;
    while (m--) {
        char op, s[N];
        cin >> op;
        if (op == 'I') {
            cin >> s;
            insert(s);
        }
        else {
            cin >> s;
            cout << find(s) << endl;
        }
    }
    return 0;
}
```

AC自动机

trie+KMP

```cpp
#include<iostream>
#include<cstdio>
```

```cpp
#include<cstring>
#include<algorithm>

using namespace std;

const int N = 1000010;

int n;
int tr[N][26], cnt[N], idx;
char str[N];
int q[N], ne[N];

void insert() {
    int p = 0;
    for (int i = 0; str[i]; i++) {
        int t = str[i] - 'a';
        if (!tr[p][t]) tr[p][t] = ++idx;
        p = tr[p][t];
    }
    cnt[p]++;
}

void build() {
    int hh = 0, tt = -1;
    for (int i = 0; i < 26; i++) {
        if (tr[0][i]) q[++tt] = tr[0][i];
    }
    while (hh <= tt) {
        int t = q[hh++];
        for (int i = 0; i < 26; i++) {
            int p = tr[t][i];
            if (!p) tr[t][i] = tr[ne[t]][i];
            else {
                ne[p] = tr[ne[t]][i];
                q[++tt] = p;
            }
        }
    }
}

int query(char s[]) {
    int res = 0;
    for (int i = 0, j = 0; str[i]; ++i) {
        int t = str[i] - 'a';
        j = tr[j][t];
        for (int p = j; p && ~cnt[p]; p = ne[p]) {
            res += cnt[p];
            cnt[p] = -1;
        }
    }
    return res;
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%s", str);
        insert();
```

```
    }

    build();

    scanf("%s", str);
    int res = query(str);
    printf("%d\n", res);
    return 0;
}
```

后缀数组

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

using namespace std;
const int N = 1000010;

int n, m;
char s[N];
int sa[N], c[N], x[N], y[N], rk[N], height[N];

void get_sa() {
    for (int i = 1; i <= n; ++i) c[x[i] = s[i]]++;
    for (int i = 2; i <= m; ++i) c[i] += c[i - 1];
    for (int i = n; i; --i) sa[c[x[i]]--] = i;

    for (int k = 1; k <= n; k <<= 1) {
        int num = 0;
        for (int i = n - k + 1; i <= n; ++i) y[++num] = i;
        for (int i = 1; i <= n; ++i)
            if (sa[i] > k)
                y[++num] = sa[i] - k;
        for (int i = 1; i <= m; ++i) c[i] = 0;
        for (int i = 1; i <= n; ++i) c[x[i]]++;
        for (int i = 2; i <= m; ++i) c[i] += c[i - 1];
        for (int i = n; i; --i) sa[c[x[y[i]]]--] = y[i], y[i] = 0;
        swap(x, y);
        x[sa[1]] = 1, num = 1;
        for (int i = 2; i <= n; ++i)
            x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] +
k]) ? num : ++num;
        if (num == n) break;
        m = num;
    }
}

void get_height() {
    for (int i = 1; i <= n; ++i) rk[sa[i]] = i;
    for (int i = 1, k = 0; i <= n; ++i) {
        if (rk[i] == 1) continue;
        if (k) k--;
        int j = sa[rk[i] - 1];
```

```
        while (i + k <= n && j + k <= n && s[i + k] == s[j + k]) k++; // 枚举LCP
        height[rk[i]] = k; // LCP的值
    }
}

int main() {
    scanf("%s", s + 1);
    n = strlen(s + 1), m = 122;
    get_sa();
    get_height();

    for (int i = 1; i <= n; ++i) printf("%d ", sa[i]);
    puts("");
    for (int i = 1; i <= n; ++i) printf("%d ", height[i]);
    return 0;
}
```

后缀自动机

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

using namespace std;
typedef long long ll;

const int N = 2000010;

int tot = 1, last = 1;
struct Node {
    int len, fa;
    int ch[26];
}node[N];

char str[N];
ll f[N], ans;
int h[N], to[N], ne[N], idx;

void extend(int c)
{
    int p = last, np = last = ++tot;
    f[tot] = 1;
    node[np].len = node[p].len + 1;
    for (; p && !node[p].ch[c]; p = node[p].fa) node[p].ch[c] = np;
    if (!p) node[np].fa = 1;
    else
    {
        int q = node[p].ch[c];
        if (node[q].len == node[p].len + 1) node[np].fa = q;
        else
        {
            int nq = ++tot;
            node[nq] = node[q], node[nq].len = node[p].len + 1;
            node[q].fa = node[np].fa = nq;
```

```
            for (; p && node[p].ch[c] == q; p = node[p].fa) node[p].ch[c] = nq;
        }
    }
}


inline void add(int u, int v) {
    to[idx] = v, ne[idx] = h[u], h[u] = idx++;
}

void dfs(int u) {
    for (int i = h[u]; ~i; i = ne[i]) {
        dfs(to[i]);
        f[u] += f[to[i]];
    }
    if (f[u] > 1) ans = max(ans, f[u] * node[u].len);
}

int main() {
    scanf("%s", str);
    for (int i = 0; str[i]; ++i) extend(str[i] - 'a');
    memset(h, -1, sizeof h);
    for (int i = 2; i <= tot; ++i) add(node[i].fa, i);
    dfs(1);
    printf("%lld\n", ans);
    return 0;
}
```