1 src\1_基本算法

基本算法

1 位运算

1.1 快速幂

```
int fpow(int a, int b, int c) {
    int res = 1 % c;
    for (; b; b >>= 1) {
        if (b & 1) res = (long long)res * a % c;
        a = (long long)a * a % c;
    }
    return res;
}
```

1.2 龟速乘

```
LL mul(LL a, LL b, LL c) {
    LL res = 0;
    for (; b; b >>= 1) {
        if (b & 1) res = (res + a) % c;
        a = (a + a) % c;
    }
    return res;
}
```

1.3 快速乘

```
ull mul(ull a, ull b, ull p) {
    a %= p, b %= p;
    ull c = (long double)a * b / p;
    ull x = a * b, y = c * p;
    ll res = (ll)(x % p) - (ll)(y % p);
    if (res < 0) res += p;
    return res;
}
```

2 排序算法

2.1 1快速排序

```
void q_sort(int l, int r) {
    if (l >= r) return;
    int i = l - 1, j = r + 1, x = a[l + r >> 1];
    while (i < j) {
        do i ++; while (x > a[i]);
        do j --; while (x < a[j]);
        if (i < j) swap(a[i], a[j]);
    }
    q_sort(l, j), q_sort(j + 1, r); //此处边界不能换为(l,i),(i+1,r) 否则会死循环
}
```

2.2 2归并排序

```
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 = l;
    while (i <= mid && j <= r)
        if (a[i] < a[j]) b[k ++] = a[i ++];
        else b[k ++] = a[j ++];
    while (i <= mid) b[k ++] = a[i ++];
    while (j <= r) b[k ++] = a[j ++];
    for (int i = l; i <= r; i ++) a[i] = b[i];
}
```

3 二分

3.1 整数二分算法模板

```
bool check(int x) {/* ... */} // 检查x是否满足某种性质
```

3.2 1区间[l,r]被划分成[l,mid]和[mid+1,r]时使用

```
int bsearch_1(int l, int r) {
    while (l < r) {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check()判断mid是否满足性质
        else l = mid + 1;
    }
    return l;
}
```

3.3 2区间[l,r]被划分成[l,mid-1]和[mid,r]时使用

```
int bsearch_2(int l, int r) {
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

3.4 3浮点数二分算法模板

```
bool check(double x) {/* ... */} // 检查x是否满足某种性质

double bsearch_3(double l, double r) {
    const double eps = 1e-6;   // eps 表示精度，取决于题目对精度的要求
    while (r - l > eps) {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

4 高精度

4.1 1加法

```
vector<int> add(vector<int> &a,vector<int> &b){
    vector<int> c;
    int t=0;//进位
    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;//进位权重下降
    }
    if(t) c.push_back(1);
    return c;
}
```

4.2 2减法

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

vector<int> sub(vector<int> &a,vector<int> &b){
    vector<int> c;
    int t=0;//借位
    for(int i=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;//t<0 表示借位了
        else t=0;//否则就是没借位
    }
    while(c.size()>1 && c.back()==0) c.pop_back();
    return c;
}
```

4.3 3乘法

1

```
vector<int> mul(vector<int> &a,int &b){
    vector<int> c;
    for(int i=0,t=0;i<a.size() || t;i++){//进位存在或没乘完
        if(i<a.size()) t+=a[i]*b;
        c.push_back(t%10);
        t/=10;
    }
    while(c.size()>1 && c.back()==0) c.pop_back();//去除前导零
    return c;
}
```

2

```
// 高精乘高精
```

```cpp
vector<int> mul(vector<int> &a, vector<int> &b) {
    vector<int> c;
    c.resize(a.size() + b.size());
    for (int i = 0; i < a.size(); ++ i) {
        int t = 0;
        for (int j = 0; j < b.size(); ++ j) {
            c[i + j] += t + a[i] * b[j];
            t = c[i + j] / 10;
            c[i + j] %= 10;
        }
        c[i + b.size()] = t;
    }
    while (c.size() > 1 && c.back() == 0) c.pop_back();
    return c;
}
```

4.4 4除法

```cpp
// 高精除低精
vector<int> div(vector<int> &a,int &b,int &r){
    vector<int> c;
    r=0;
    for(int i=a.size()-1;i>=0;i--){
        r=r*10+a[i];
        c.push_back(r/b);
        r%=b;
    }
    reverse(c.begin(),c.end());
    while(c.size()>1 && c.back()==0) c.pop_back();//去除前导零
    return c;
}
```

5 前缀和差分

$$S[i]=\sum_{j=1}^{i}A[j]$$

5.1 1一维前缀和

```cpp
#include<iostream>
#include<cstdio>
using namespace std;
const int M=1e5+10;
int a[M],b[M];
int main(){
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    for(int i=1;i<=n;i++) b[i]=a[i]+b[i-1];
    while(m--){
        int l,r;
```

```
        scanf("%d%d",&l,&r);
        printf("%d\n",b[r]-b[l-1]);
    }
}
```

5.2 2二维前缀和

```
#include<iostream>
#include<cstdio>
using namespace std;

const int M=1e3+10;
int a[M][M],b[M][M];
int n,m,q,x1,y1,x2,y2;

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++)
            b[i][j]=b[i-1][j]+b[i][j-1]-b[i-1][j-1]+a[i][j];
    while(q--){
        scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
        printf("%d\n",b[x2][y2]-b[x1-1][y2]-b[x2][y1-1]+b[x1-1][y1-1]);
    }
}
```

5.3 3一维差分

```
#include<iostream>
#include<cstdio>
using namespace std;

const int M=1e5+10;
int a[M],b[M];
int n,m;
int l,r,c;

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

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++) b[i]=a[i]-a[i-1];//构造差分数组
    for(int i=1;i<=m;i++){
        scanf("%d%d%d",&l,&r,&c);
        in(l,r,c);
    }
    for(int i=1;i<=n;i++) b[i]=b[i]+b[i-1];//还原数组
    for(int i=1;i<=n;i++) printf("%d ",b[i]);
}
```

5.4 4二维差分

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

using namespace std;

const int M=1000+10;
int n,m,q;
int a[M][M],s[M][M];
int x1,y1,x2,y2,c;

void in(int x1,int y1,int x2,int y2,int c){//插入
    s[x1][y1]+=c;
    s[x2+1][y1]-=c;
    s[x1][y2+1]-=c;
    s[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++)
            in(i,j,i,j,a[i][j]);//构造
    for(int i=1;i<=q;i++){
        scanf("%d%d%d%d%d",&x1,&y1,&x2,&y2,&c);
        in(x1,y1,x2,y2,c);
    }
    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];//还原
    for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++){
            printf("%d ",s[i][j]);
            if(j==m) puts("");
        }
}
```

6 双指针算法

```cpp
for (int i = 0, j = 0; i < n; i ++ ) {
    while (j < i && check(i, j)) j ++ ;
    // 具体问题的逻辑
}
```

常见问题分类：

- (1) 对于一个序列，用两个指针维护一段区间

- (2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作

7 离散化与逆序对

7.1 1哈希

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

using namespace std;

typedef long long LL;

const int N = 5e5 + 10;
int c[N], n;
int a[N], b[N];
unordered_map<int, int> m;

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

void add(int x, int v) {
    for (int i = x; i < N; i += lowbit(i))
        c[i] += v;
}

int query(int x) {
    int res = 0;
    for (int i = x; i; i -= lowbit(i))
        res += c[i];
    return res;
}

int main() {
    scanf("%d", &n);
    for (int i = 1; i <= n; ++ i)
        scanf("%d", &a[i]), b[i] = a[i];
    sort(b + 1, b + 1 + n);
    int cnt = 0;
    for (int i = 1, j = 1; i <= n; ++ i) {
        while (j <= n && b[j] == b[i]) ++ j;
        m[b[i]] = ++ cnt;
        i = j - 1;
    }
    for (int i = 1; i <= n; ++ i)
        a[i] = m[a[i]];
    LL ans = 0;
    for (int i = 1; i <= n; ++ i) {
```

```
        add(a[i], 1);
        ans += i - query(a[i]);
    }
    cout << ans << endl;
}
```

7.2 2二分

```
sort(b + 1, b + 1 + n);
int k = unique(b + 1, b + 1 + n) - b - 1;
for (int i = 1; i <= n; ++ i)
    a[i] = lower_bound(b + 1, b + 1 + k, a[i]) - b;
```

8 模拟退火

使用范围: 最优化问题，比如DP，贪心，计算几何

如果函数连续性较强 (即轻微的扰动对函数值的影响较小)，退火的出解率较高

序列的邻项考虑随机交换，如 2424. 保龄球 - AcWing题库

```
#include <bits/stdc++.h> // https://www.acwing.com/problem/content/3170/

using namespace std;

#define x first
#define y second
#define toMin(a, b) a > b ? a = b: 0

typedef pair<double, double> PDD;

const int N = 110;
double ans = 1e8;
PDD a[N];
int n;

double rand(int l, int r) {
    return (double)rand() / RAND_MAX * (r - l) + l;
}

double dist(PDD a, PDD b) {
    auto x = a.x - b.x;
    auto y = a.y - b.y;
    return sqrt(x * x + y * y);
}

double calc(PDD p) {
    double res = 0;
    for (int i = 1; i <= n; ++ i) res += dist(p, a[i]);
    toMin(ans, res);
    return res;
}

```

```cpp
void SA() {
    PDD cur(rand(0, 10000), rand(0, 10000));
    for (double T = 1e4; T > 1e-4; T *= 0.99) { // 不同的参数出解率不同
        PDD np(rand(cur.x - T, cur.x + T), rand(cur.y - T, cur.y + T));
        double dt = calc(np) - calc(cur);
        if (exp(-dt / T) > rand(0, 1)) cur = np;
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);
    srand(time(0));

    cin >> n;
    for (int i = 1; i <= n; ++ i) cin >> a[i].x >> a[i].y;
    for (int i = 1; i <= 100; ++ i) SA(); // 考场里可以考虑卡时
    cout << int(ans + 0.5) << '\n';
}
```

9 文件读写

```cpp
// #include <cstdlib>
// freopen("P2058_2.in", "r", stdin);
// freopen("my_ans.out", "w", stdout);
// fclose(stdin);
// fclose(stdout);
```

10 快读

1cin加速:

根据个人评测经验，这种方法会比 `scanf()` 还要快

```cpp
cin.tie(0)->sync_with_stdio(0)
```

2使用 `getchar()`

```cpp
template<typename T> void read(T &x) {
    char ch = getchar(); bool flag = 0; x = 0;
    for (; ch < '0' || ch > '9'; ch = getchar())
        flag |= (ch == '-');
    for (; ch >= '0' && ch <= '9'; ch = getchar())
        x = (x << 1) + (x << 3) + ch - '0';
    if (flag) x = -x;
}
```

3将字符先读到 `buff` 数组中再从 `buff` 中读入

```
inline char GET_CHAR() {
    static char buf[maxn], *p1 = buf, *p2 = buf;
    return p1 == p2 && (p2 = (p1 = buf) +
        fread(buf, 1, maxn, stdin), p1 == p2) ? EOF: *p1 ++;
}

template<class T> inline void read(T &x) {
    x = 0; int f = 0; char ch = GET_CHAR();
    for (; ch < '0' || ch > '9'; ch = GET_CHAR()) flag |= (ch == '-');
    for (; ch >= '0' && ch <= '9'; ch = GET_CHAR()) x = (x << 1) + (x << 3) + (ch ^
48);
    x = f ? -x: x;
}
```

数据结构

1 链表

1.1 1单链表

```
#include <iostream>
using namespace std;
const int N = 100010;
int head, e[N], ne[N], idx;
void init() {
    head = -1;
    idx = 0;
}
void add_to_head(int x) {
    e[idx] = x, ne[idx] = head, head = idx ++ ;
}
void add(int k, int x) {
    e[idx] = x, ne[idx] = ne[k], ne[k] = idx ++ ;
}
void remove(int k) {
    ne[k] = ne[ne[k]];
}
int main() {
    int m; cin >> m;
    init();
    while (m -- ) {
        int k, x; char op;
        cin >> op;
        if (op == 'H') {
            cin >> x;
            add_to_head(x);
        } else if (op == 'D') {
            cin >> k;
            if (!k) head = ne[head];
            else remove(k - 1);
        } else {
            cin >> k >> x;
            add(k - 1, x);
        }
    }
```

```
    for (int i = head; i != -1; i = ne[i]) cout << e[i] << ' ';
    cout << endl;
    return 0;
}
```

1.2 2双链表

```
#include <iostream>
using namespace std;
const int N = 100010;
int m;
int e[N], l[N], r[N], idx;
// 在节点a的右边插入一个数x
void insert(int a, int x) {
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx ++ ;
}
// 删除节点a
void remove(int a) {
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}
int main() {
    cin >> m;
    // 0是左端点，1是右端点
    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] << ' ';
    cout << endl;
    return 0;
}
```

2 栈

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int N = 1e6 + 10;
int stk[N], tt = -1;
string op;
int m, k;
int main() {
    cin >> m;
    while (m --) {
        cin >> op;
        if (op == "push") {
            cin >> k;
            stk[++ tt] = k;
        } else if (op == "query") {
            cout << stk[tt] << endl;
        } else if (op == "pop") tt --;
        else cout << (tt < 0 ? "YES" : "NO") << endl;
    }
    return 0;
}
```

2.1 单调栈

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

using namespace std;

const int N = 3e6 + 10;
int stk[N], tt, n;
int a[N], ans[N];

void push(int x) { stk[++tt] = x; }

int pop() { return stk[tt--]; }

int top() { return stk[tt]; }

int empty()
{
    if (tt > 0)
        return false;
    return true;
}

void out()
{
    for (int i = 1; i <= tt; i++)
        cout << stk[i] << ' ';
    cout << endl;
}

int main()
```

```
{
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
        scanf("%d", &a[i]);
    for (int i = n; i > 0; i--)
    {
        while (!empty() && a[top()] <= a[i]) pop();
        ans[i] = empty() ? 0 : top();
        push(i);
    }
    for (int i = 1; i <= n; i++) printf("%d ", ans[i]);
    //system("pause");
    return 0;
}
```

3 队列

```
#include <iostream>
using namespace std;
const int N = 100010;
int m;
int q[N], hh, tt = -1;
int main() {
    cin >> m;
    while (m -- ) {
        string op;
        int x;
        cin >> op;
        if (op == "push") {
            cin >> x;
            q[ ++ tt] = x;
        }
        else if (op == "pop") hh ++ ;
        else if (op == "empty") cout << (hh <= tt ? "NO" : "YES") << endl;
        else cout << q[hh] << endl;
    }
    return 0;
}
```

3.1 单调队列

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

using namespace std;

const int N = 1e6 + 10;
int q[N], hh, tt = -1;
int n, k;
int a[N];

int main()
{
    scanf("%d%d", &n, &k);
    for (int i = 0; i < n; i++)
    {
```

```
        scanf("%d", &a[i]);
        if (i - k + 1 > q[hh])
            ++hh;
        while (hh <= tt && a[i] <= a[q[tt]])
            --tt;
        q[++tt] = i;
        if (i + 1 >= k)
            printf("%d ", a[q[hh]]);
    }
    puts("");
    hh = 0, tt = -1;
    for (int i = 0; i < n; i++)
    {
        if (i - k + 1 > q[hh])
            ++hh;
        while (hh <= tt && a[i] >= a[q[tt]])
            --tt;
        q[++tt] = i;
        if (i + 1 >= k)
            printf("%d ", a[q[hh]]);
    }
    //system("pause");
    return 0;
}
```

4 KMP

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

using namespace std;

const int N = 1e6 + 10;
int n, m;
char p[N] = "0", s[N] = "0";
int ne[N];

int main()
{
    cin >> s + 1 >> p + 1;
    n = strlen(p) - 1;
    m = strlen(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\n", i - n + 1);
            j = ne[j];
        }
    }

    for (int i = 1; i <= n; i++)
        printf("%d ", ne[i]);
    system("pause");
    return 0;
}
```

5 Trie树

```
const int M = 1e6 + 10;
int son[M][26], cnt[M], idx;
// 插入
void insert(char str[])
{
    int p = 0;
    for (int i = 0; str[i]; i++)
    {
        int u = str[i] - 'a';
        if (!son[p][u]) son[p][u] = ++ idx;
        p = son[p][u];
    }
    cnt[p] ++;
}
// 查询
int query(char str[])
{
    int p = 0;
    for (int i = 0; str[i]; i++)
    {
        int u = str[i] - 'a';
        if (!son[p][u]) return 0;
        p = son[p][u];
    }
    return cnt[p];
}
```

6 并查集

```
void init(int n) {
    for (int i = 1; i <= n; i ++) fa[i] = i;
}

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

void merge(int x, int y) {
    fa[find(x)] = find(y);
}
```

7 堆

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

using namespace std;

const int N = 1e6 + 10;
int h[N], cnt, n;

void up(int k) {
    while (k >> 1 && h[k >> 1] > h[k]) {
        swap(h[k], h[k >> 1]);
        k >>= 1;
    }
}

void insert(int x) {
    h[++ cnt] = x;
    up(cnt);
}

void down(int x) {
    int t = x;
    if (x * 2 <= cnt && h[t] > h[x * 2]) t = 2 * x;
    if (2 * x + 1 <= cnt && h[t] > h[2 * x + 1]) t = 2 * x + 1;
    if (t != x) swap(h[t], h[x]), down(t);
}

void remove() {
    swap(h[1], h[cnt--]);
    down(1);
}

int main() {
    scanf("%d", &n);
    int op, x;
    while (n --) {
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d", &x);
```

```
            insert(x);
        }
        else if (op == 2) printf("%d\n", h[1]);
        else if (op == 3) remove();
    }
    return 0;
}
```

8 hash表

8.1 1开放寻址法

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

const int N = 2e5 + 10, INF = 0x3f3f3f3f;
int h[N];
int find(int x) {
    int t = (x % N + N) % N;
    while (h[t] != INF && h[t] != x) {
        t ++;
        if (t == N) t = 0;
    }
    return t;
}

int main() {
    int n;
    scanf("%d", &n);
    memset(h, 0x3f, sizeof h);
    while (n --) {
        char op[2]; int x;
        scanf("%s%d", op, &x);
        int k = find(x);
        if (op[0] == 'I') h[k] = x;
        else {
            if (h[k] == INF) puts("No");
            else puts("Yes");
        }
    }
}
```

8.2 2拉链法

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

using namespace std;

const int N = 1e5 + 3;

```

```cpp
int h[N], e[N], ne[N], idx;

void insert(int x) {
    int k = (x % N + N) % N;
    e[idx] = x, ne[idx] = h[k], h[k] = idx ++;
}

bool find(int x) {
    int k = (x % N + N) % N;
    for (int i = h[k]; i != -1; i = ne[i]) if (e[i] == x) return true;
    return false;
}

int main() {
    int n;
    scanf("%d", &n);
    memset(h, -1, sizeof h);
    while (n --) {
        char op[2]; int x;
        scanf("%s%d", op, &x);
        if (op[0] == 'I') insert(x);
        else {
            if (find(x)) puts("Yes");
            else puts("No");
        }
    }
}
```

8.3 3字符串哈希

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10, P = 131;
#define u unsigned long long
u n, m, h[N], p[N];
char str[N];
u f(int l, int r) {
    return h[r] - h[l - 1] * p[r - l + 1];
}
int main() {
    cin >> 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;
    }
    while(m --) {
        u a, b,c, d;
        cin >> a >> b >> c >> d;
        if(f(a, b) == f(c, d)) puts("Yes");
        else puts("No");
    }
    return 0;
}
```

随机模数，防止 hack。

```cpp
using LL = long long;
using i128 = __int128; // gcc win 64 only

std::mt19937 rng(std::chrono::steady_clock::now().time_since_epoch().count());

LL fpow(LL a, LL b, LL p) {
    LL res = 1;
    while (b) {
        if (b & 1) res = res * a % p;
        a = a * a % p;
        b >>= 1;
    }
    return res;
}

bool millerRabin(LL x) {
    if (x < 2) return false;
    if (x == 2) return true;
    if (x % 2 == 0) return false;
    LL p = x - 1, q = 0;
    while (!(p & 1)) p >>= 1, q++;
    for (int i = 0; i < 10; i++) {
        LL a = rng() % (x - 2) + 2;
        LL v = fpow(a, p, x);
        if (v == 1 || v == x - 1) continue;
        for (int j = 0; j < q; j++) {
            v = v * v % x;
            if (v == x - 1) break;
            if (j == q - 1) return false;
        }
    }
    return true;
}

template<typename T1 = int, typename T2 = LL> // T1用来存储数据 T2用来进行预算(防止溢出)
struct strHash {
    int base, n;
    std::vector<T1> hash, pow;
    T1 mod;

    strHash(int N, int base = 131) : base(base), pow(N) {
        mod = std::uniform_int_distribution<T1>(1e9, 2e9)(rng);
        while (!millerRabin(mod)) mod++;
        pow[0] = 1;
        for (int i = 1; i < N; i++) pow[i] = (T2)pow[i - 1] * base % mod;
    }

    void init(std::string &s, int n) { // s下标从1开始
        this->n = n;
        hash.assign(n + 1, 0);
        for (int i = 1; i <= n; i++) hash[i] = ((T2)hash[i - 1] * base + s[i]) % mod;
    }

    T1 get(int l, int r) { // [l, r]
        return (hash[r] - (T2)hash[l - 1] * pow[r - l + 1] % mod + mod) % mod;
    }
```

```
};
```

9 C++STL

```
unique，将数组中重复的元素放到了最后
sort，将数组中的元素排序
vector，变长数组，倍增的思想
    size()  返回元素个数
    empty()  返回是否为空
    clear()  清空
    front()/back()
    push_back()/pop_back()
    begin()/end()
    []
    支持比较运算，按字典序
pair<int, int>
    first，第一个元素
    second，第二个元素
    支持比较运算，以first为第一关键字，以second为第二关键字（字典序）
string，字符串
    size()/length()  返回字符串长度
    empty()
    clear()
    substr(起始下标，(子串长度))  返回子串
    c_str()  返回字符串所在字符数组的起始地址
queue，队列
    size()
    empty()
    push()  向队尾插入一个元素
    front()  返回队头元素
    back()  返回队尾元素
    pop()  弹出队头元素
priority_queue，优先队列，默认是大根堆
    push()  插入一个元素
    top()  返回堆顶元素
    pop()  弹出堆顶元素
    定义成小根堆的方式：priority_queue<int, vector<int>, greater<int>> q;
stack，栈
    size()
    empty()
    push()  向栈顶插入一个元素
    top()  返回栈顶元素
    pop()  弹出栈顶元素
deque，双端队列
    size()
    empty()
    clear()
    front()/back()
    push_back()/pop_back()
    push_front()/pop_front()
    begin()/end()
    []
set, map, multiset, multimap, 基于平衡二叉树（红黑树），动态维护有序序列
    size()
    empty()
```

```
    clear()
    begin()/end()
    ++, -- 返回前驱和后继，时间复杂度 O(logn)

    set/multiset
        insert()  插入一个数
        find()  查找一个数
        count()  返回某一个数的个数
        erase()
            (1) 输入是一个数x，删除所有x   O(k + logn)
            (2) 输入一个迭代器，删除这个迭代器
        lower_bound()/upper_bound()
            lower_bound(x)  返回大于等于x的最小的数的迭代器
            upper_bound(x)  返回大于x的最小的数的迭代器
    map/multimap
        insert()  插入的数是一个pair
        erase()  输入的参数是pair或者迭代器
        find()
        []  注意multimap不支持此操作。 时间复杂度是 O(logn)
        lower_bound()/upper_bound()
unordered_set, unordered_map, unordered_multiset, unordered_multimap, 哈希表
    和上面类似，增删改查的时间复杂度是 O(1)
    不支持 lower_bound()/upper_bound(),  迭代器的++, --
bitset, 压位
    bitset<10000> s;
    ~, &, |, ^
    >>, <<
    ==, !=
    []

    count()  返回有多少个1

    any()  判断是否至少有一个1
    none()  判断是否全为0

    set()  把所有位置成1
    set(k, v)  将第k位变成v
    reset()  把所有位变成0
    flip()  等价于~
    flip(k) 把第k位取反
```

10 C++pb_ds

洛谷 P3369 【模板】普通平衡树

```cpp
#include <iostream>
#include <ext/pb_ds/assoc_container.hpp> // 引入树的头文件
#include <ext/pb_ds/tree_policy.hpp>

/**
 * 赛时可以写万能头
 * #include <bits/stdc++.h>
 * #include <bits/extc++.h>
 */

namespace pbds = __gnu_pbds;
```

```cpp

template<typename T, typename cmp = std::less<T>>
using rbtree = pbds::tree<T, pbds::null_type, cmp, pbds::rb_tree_tag,
pbds::tree_order_statistics_node_update>;
using PII = std::pair<int, int>;

void solve() {
    int n;
    std::cin >> n;

    rbtree<PII> tree;
    int idx = 0;

    while (n --) {
        int op, x;
        std::cin >> op >> x;

        if (op == 1) {
            tree.insert({x, ++ idx});
        } else if (op == 2) {
            tree.erase(tree.lower_bound({x, 0}));
        } else if (op == 3) {
            std::cout << tree.order_of_key({x, 0}) + 1 << '\n';
        } else if (op == 4) {
            std::cout << tree.find_by_order(x - 1)->first << '\n';
        } else if (op == 5) {
            std::cout << (--tree.lower_bound({x, 0}))->first << '\n';
        } else {
            std::cout << tree.upper_bound({x, idx})->first << '\n';
        }
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);

    solve();
}
```

11 线段树

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

using LL = long long;

struct node {
    int l, r;
    LL val, add;
};

struct segmentTree {
    #define ls(x) (x << 1)
    #define rs(x) (x << 1 | 1)
```

```
    #define p tr[u]
    #define pl tr[ls(u)]
    #define pr tr[rs(u)]

    std::vector<node> tr;
    segmentTree(int n) : tr(n << 2) {}

    void pushup(int u) {
        p.val = pl.val + pr.val;
    }

    void pushdown(int u) {
        if (p.add) {
            pl.val += p.add * (pl.r - pl.l + 1);
            pr.val += p.add * (pr.r - pr.l + 1);
            pl.add += p.add;
            pr.add += p.add;
            p.add = 0;
        }
    }

    void build(int u, int l, int r, std::vector<LL> &a) {
        p.l = l, p.r = r;
        if (l == r) {
            p.val = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(ls(u), l, mid, a);
        build(rs(u), mid + 1, r, a);
        pushup(u);
    }

    void modify(int u, int l, int r, LL d) {
        if (p.l >= l && p.r <= r) {
            p.val += d * (p.r - p.l + 1);
            p.add += d;
            return;
        }
        pushdown(u);
        int mid = (p.l + p.r) >> 1;
        if (l <= mid) modify(ls(u), l, r, d);
        if (r > mid) modify(rs(u), l, r, d);
        pushup(u);
    }

    LL query(int u, int l, int r) {
        if (p.l >= l && p.r <= r) return p.val;
        pushdown(u);
        int mid = (p.l + p.r) >> 1;
        LL res = 0;
        if (l <= mid) res += query(ls(u), l, r);
        if (r > mid) res += query(rs(u), l, r);
        return res;
    }

```

```
    #undef ls
    #undef rs
    #undef p
    #undef pl
    #undef pr
};

void solve() {
    int n, m;
    std::cin >> n >> m;
    std::vector<LL> a(n + 1);
    for (int i = 1; i <= n; ++ i) std::cin >> a[i];
    segmentTree tree(n + 10);
    tree.build(1, 1, n, a);
    while (m --) {
        int op, x, y, k;
        std::cin >> op >> x >> y;
        if (op == 1) {
            std::cin >> k;
            tree.modify(1, x, y, k);
        } else {
            std::cout << tree.query(1, x, y) << '\n';
        }
    }
}

int main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);

    solve();
}
```

12 树状数组

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

using namespace std;

const int N = 5e5 + 10;
int n, m, tr[N];

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

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

int query(int x) {
    int res = 0;
    for (; x; x -= lowbit(x)) res += tr[x];
    return res;
```

```
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++ i) {
        int t; scanf("%d", &t);
        add(i, t);
    }
    while (m --) {
        int op, x, y;
        scanf("%d%d%d", &op, &x, &y);
        if (op == 1) add(x, y);
        else printf("%d\n", query(y) - query(x - 1));
    }
}
```

13 回文串

马拉车 d1: 奇数长度最长回文半径 d2: 偶数长度最长回文半径

```
void calc(string &s, int n, bool *pre) { // 判断前缀是否回文 字符串下标从0开始
    vector<int> d1(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 1 : min(d1[l + r - i], r - i + 1);
        while (0 <= i - k && i + k < n && s[i - k] == s[i + k]) {
            k++;
        }
        d1[i] = k--;
        if (i + k > r) {
            l = i - k;
            r = i + k;
        }
    }
    vector<int> d2(n);
    for (int i = 0, l = 0, r = -1; i < n; i++) {
        int k = (i > r) ? 0 : min(d2[l + r - i + 1], r - i + 1);
        while (0 <= i - k - 1 && i + k < n && s[i - k - 1] == s[i + k]) {
            k++;
        }
        d2[i] = k--;
        if (i + k > r) {
            l = i - k - 1;
            r = i + k;
        }
    }
    for (int i = 0; i < n; ++ i) {
        int l = i - d1[i] + 1, r = i + d1[i] - 1;
        if (!l && !pre[r]) pre[r] = true;

        l = i - d2[i], r = i + d2[i] - 1;
        if (!l && !pre[r]) pre[r] = true;
    }
}
```

搜索

1 DFS

1.1 枚举

1.1.1 指数型

```
void calc(int x) {
    if (x > n) {
        for (int i: choose) cout << i << ' ';
        cout << endl;
        return;
    }
    calc(x + 1); // choose
    choose.push_back(x); // not choose
    calc(x + 1);
    choose.pop_back();
};
```

1.1.2 组合型

```
void calc(int x) {
    if (choose.size() > m || choose.size() + n - x + 1 < m)
        return;
    if (x > n) {
        for (int i: choose) cout << i << ' ';
        cout << endl;
        return;
    }
    choose.push_back(x); // not choose
    calc(x + 1);
    choose.pop_back();
    calc(x + 1); // choose
};
```

1.1.3 排列型

```
void calc(int x) {
    if (x > n) {
        for (int i = 1; i <= n; ++ i)
            printf("%d%c", a[i], " \n"[i == n]);
        return;
    }
    for (int i = 1; i <= n; ++ i)
        if (!st[i]) {
            a[x] = i, st[i] = true;
            calc(x + 1);
            a[x] = 0, st[i] = false;
        }
}
```

2 BFS

2.1 Flood-Fill

可以在线性时间复杂度内, 找到某个点所在的连通块.

[1097. 池塘计数 - AcWing题库]

```cpp
#include <iostream>
#include <cstring>
#include <algorithm>
#define x first
#define y second

using namespace std;

typedef pair<int, int> PII;

const int N = 1005, M = N * N;
char g[N][N];
bool st[N][N];
int n, m, cnt;
PII q[M];

void bfs(int x, int y) {
    int hh = 0, tt = 0;
    q[tt] = {x, y};
    st[x][y] = true;

    while (hh <= tt) {
        auto [x, y] = q[hh ++];
        for (int i = x - 1; i <= x + 1; ++ i)
            for (int j = y - 1; j <= y + 1; ++ j) {
                if (i == x && j == y) continue;
                if (i < 0 || i >= n || j < 0 || j >= m) continue;
                if (g[i][j] == '.' || st[i][j]) continue;

                q[++ tt] = {i, j};
                st[i][j] = true;
            }
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);

    cin >> n >> m;
    for (int i = 0; i < n; ++ i) cin >> g[i];

    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j)
            if (g[i][j] == 'W' && !st[i][j]) {
```

```
                bfs(i, j);
                ++ cnt;
            }

    cout << cnt << '\n';
}
```

2.2 最短距离

要满足所有边的权重都相同

[1076. 迷宫问题 - AcWing题库]

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

#define x first
#define y second

using namespace std;

typedef pair<int, int> PII;

const int N = 1005;
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};
bool g[N][N];
bool st[N][N];
PII pre[N][N];
int n, idx;
PII q[N * N];

bool isin(int x, int y) {
    return 0 <= x && x < n && 0 <= y && y < n;
}

void bfs(int x, int y) {
    int hh = 0, tt = 0;
    q[0] = {x, y};
    st[x][y] = true;

    while (hh <= tt) {
        auto [x, y] = q[hh ++];
        if (x == n - 1 && y == n - 1) return;

        for (int i = 0; i < 4; ++ i) {
            int nx = x + dx[i], ny = y + dy[i];
            if (isin(nx, ny) && !g[nx][ny] && !st[nx][ny]) {
                q[++ tt] = {nx, ny};
                pre[nx][ny] = {x, y};
                st[nx][ny] = true;
            }
        }
    }
```

```
}

void printpath(int x, int y, int cnt) {
    if (!x && !y) { cout << "0 0\n"; return; }
    auto [nx, ny] = pre[x][y];
    printpath(nx, ny, cnt + 1);
    cout << x << ' ' << y << '\n';
}

int main() {
    cin.tie(0)->sync_with_stdio(0);

    cin >> n;
    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < n; ++ j)
            cin >> g[i][j];

    bfs(0, 0);

    printpath(n - 1, n - 1, 0);
}
```

2.3 多源BFS

增加一个虚拟源点

173. 矩阵距离 - AcWing题库

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

using namespace std;

typedef pair<int, int> PII;

const int N = 1010;
const int dx[] = {0, -1, 0, 1};
const int dy[] = {-1, 0, 1, 0};
char g[N][N];
int dist[N][N];
int n, m;
PII q[N * N];

bool isin(int x, int y) {
    return 0 <= x && x < n && 0 <= y && y < m;
}

void bfs() {
    memset(dist, -1, sizeof dist);
    int hh = 0, tt = -1;

    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j)
            if (g[i][j] == '1') {
                q[++ tt] = {i, j};
```

```
                dist[i][j] = 0;
            }

    while (hh <= tt) {
        auto [x, y] = q[hh ++];
        int d = dist[x][y];

        for (int i = 0; i < 4; ++ i) {
            int nx = x + dx[i], ny = y + dy[i];
            if (isin(nx, ny) && dist[nx][ny] == -1) {
                dist[nx][ny] = d + 1;
                q[++ tt] = {nx, ny};
            }
        }
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);

    cin >> n >> m;
    for (int i = 0; i < n; ++ i) cin >> g[i];

    bfs();

    for (int i = 0; i < n; ++ i)
        for (int j = 0; j < m; ++ j)
            cout << dist[i][j] << " \n"[j == m - 1];
}
```

2.4 最小步数

AcWing 1107. 魔板 - AcWing

AcWing 845. 八数码 - AcWing

```
// https://www.acwing.com/activity/content/problem/content/1475/
#include <iostream>
#include <cstring>
#include <algorithm>
#include <unordered_map>
#include <queue>

using namespace std;

string start = "12345678";
unordered_map<string, int> dist;
unordered_map<string, pair<char, string>> path;
const int change[3][8] = {
    {7, 6, 5, 4, 3, 2, 1, 0},
    {3, 0, 1, 2, 5, 6, 7, 4},
    {0, 6, 1, 3, 4, 2, 5, 7}
};

```

```cpp
string move(string s, int k) {
    string res;
    for (int i = 0; i < 8; ++ i) res += s[change[k][i]];
    return res;
}

int bfs(string start, string end) {
    queue<string> q;
    q.push(start);
    dist[start] = 0;

    while (q.size()) {
        string t = q.front();
        int d = dist[t];
        q.pop();

        if (t == end) return d;

        for (int i = 0; i < 3; ++ i) {
            string u = move(t, i);
            if (!dist.count(u)) {
                q.push(u);
                dist[u] = d + 1;
                path[u] = {'A' + i, t};
            }
        }
    }

    return -1;
}

void printpath(string s) {
    if (s == start) return;
    auto [op, str] = path[s];
    printpath(str);
    cout << op;
}

int main() {
    cin.tie(0)->sync_with_stdio(0);

    string end;
    for (int i = 0; i < 8; ++ i) {
        int x; cin >> x;
        end += '0' + x;
    }

    cout << bfs(start, end) << '\n';
    printpath(end);
}
```

2.5 双端队列BFS

AcWing 175. 电路维修 - AcWing

```cpp
#include <iostream>     /*如果边权只有0和1就可以将bfs转化为dijkstra*/
#include <cstring>      /*在本题中将联通的两点看成边权为0*/
#include <deque>        /*如果两点中间的标准件需要旋转才能联通则看为1*/
#include <algorithm>    /*即转化为最短路问题*/
                        /*容易知道奇点无法到达所以在bfs时不会出现某个点被操作两次*/
using namespace std;    /*该题点的坐标和标准件坐标并不同,注意变换*/

typedef pair<int, int> PII;

const int N = 510;
char g[N][N];
int dist[N][N];
bool st[N][N];
int n, m;

int bfs() {
    memset(dist, 0x3f, sizeof dist);
    memset(st, false, sizeof st);
    deque<PII> q;
    q.push_back({0, 0});
    dist[0][0] = 0;

    char cs[] = "\\/\\/";
    int dx[] = {-1, 1, 1, -1}, dy[] = {-1, -1, 1, 1};
    int ix[] = {-1, 0, 0, -1}, iy[] = {-1, -1, 0, 0};

    while (q.size()) {
        auto [x, y] = q.front();
        int d = dist[x][y];
        q.pop_front();

        if (st[x][y]) continue;
        st[x][y] = true;

        for (int i = 0; i < 4; ++ i) {
            int nx = x + dx[i], ny = y + dy[i];
            if (nx < 0 || nx > n || ny < 0 || ny > m) continue;

            int gx = x + ix[i], gy = y + iy[i];
            int w = g[gx][gy] != cs[i];

            if (d + w < dist[nx][ny]) {
                dist[nx][ny] = d + w;
                if (w) q.push_back({nx, ny});
                else q.push_front({nx, ny});
            }
        }
    }

    return dist[n][m];
}

```

```cpp
int main() {
    cin.tie(0)->sync_with_stdio(0);

    int T; cin >> T;
    while (T --) {
        cin >> n >> m;
        for (int i = 0; i < n; ++ i) cin >> g[i];

        if (n + m & 1) cout << "NO SOLUTION\n";
        else cout << bfs() << '\n';
    }
}
```

3 背包问题

3.1 01背包问题

每件物品最多可以选一次

起点	$i \in [0,m], f(0,i)=0$
转移方程	$f(i,j)=max(f(i-1,j),f(i-1,j-v_i)+w_i)$
终点	$f(n,m)$

- 朴素版本

```cpp
for (int i = 1; i <= n; ++ i)
    for (int j = 0; j <= m; ++ j) {
        f[i][j] = f[i - 1][j];
        if (v[i] <= j) f[i][j] = max(f[i][j], f[i - 1][j - v[i]] + w[i]);
    }

printf("%d", f[n][m]);
```

- 优化为一维

```cpp
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]);

printf("%d", f[m]);
```

3.2 完全背包

每个物品可以选无限个

起点	$f(0,j)=0$
转移方程	$f(i,j)=Max_{k=0}^{v_i\times k\le j}f(i-1,j-v_i\times k)+w_i\times k$
终点	$f(n,m)$

- 朴素版本

```
for (int i = 1; i <= n; ++ i)
    for (int j = 0; j <= m; ++ j)
        for (int k = 0; k * v[i] <= j; ++ k)
            f[i][j] = max(f[i][j], f[i- 1][j - k * v[i]] + k * w[i]);

cout << f[n][m] << endl;
```

- 一维优化

```
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;
```

3.3 多重背包

每个物品只能选有限个

起点	$f(0,j)=0$
转移方程	$f(i,j)=Max_{k=0}^{s_i}f(i-1,j-v_i\times k)+w_i\times k$
终点	$f(n,m)$

- 朴素版本

```
for (int i = 1; i <= n; ++ i)
    for (int j = 0; j <= m; ++ j)
        for (int k = 0; k <= s[i] && k * v[i] <= j; ++ k)
            f[i][j] = max(f[i][j], f[i - 1][j - k * v[i]] + w[i] * k);

cout << f[n][m] << endl;
```

- 二进制优化

```
for (int i = 1; i <= n; ++ i) {
    int v, w, s;
    scanf("%d%d%d", &v, &w, &s);
    for (int k = 1; k <= s; s -= k, k <<= 1)
        goods.push_back({k * v, k * w});
    if (s) goods.push_back({s * v, s * w});
}
for (auto [v, w]: goods)
    for (int j = m; j >= v; -- j)
        f[j] = max(f[j], f[j - v] + w);
cout << f[m] << endl;
```

3.4 分组背包

每组物品有若干个，同一组内的物品最多只能选一个。

起点	$f(0,j)=0$
转移方程	$f(i,j)=Max(f(i-1,j),Max_{k=1}^{s_i}f(i-1,j-v_{i,k})+w_{i,k})$
终点	$f(n,m)$

```
for (int i = 1; i <= n; ++ i)
    for (int j = m; j >= 0; -- j)
        for (int k = 1; k <= s[i]; ++ k)
            if (v[i][k] <= j)
                f[j] = max(f[j], f[j - v[i][k]] + w[i][k]);
printf("%d\n", f[m]);
```

4 数位DP

- 计数问题

```
LL f(int pos, int cnt, bool islimit, bool isnum, int digit) {
    if (pos == s.size()) return isnum ? cnt: 0;
    LL& u = dp[pos][cnt];
    if (!islimit && isnum && u >= 0) return u;
    LL res = isnum ? 0: f(pos + 1, cnt, false, false, digit);
    int up = islimit ? s[pos] - '0': 9;
    for (int d = 1 - isnum; d <= up; ++ d)
        res += f(pos + 1, cnt + (d == digit), islimit && d == up, true, digit);
    if (!islimit && isnum) u = res;
    return res;
}

LL solve(LL x, int digit) {
    memset(dp, -1, sizeof dp);
    s = to_string(x);
    return f(0, 0, true, false, digit);
}
```

5 线性DP

5.1 数字三角形

```
for (int i = 0; i <= r; ++ i)
    for (int j = 0; j <= i + 1; ++ j)
        f[i][j] = -INF;

f[1][1] = a[1][1];
for (int i = 2; i <= r; ++ 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]);
// 可以是从上向下走也可以是从左向右走
int res = -INF;
for (int i = 1; i <= r; ++ i) res = max(res, f[r][i]);
printf("%d\n", res);
```

5.2 最长上升子序列

- $O(n^2)$ 做法

```
for (int i = 1; i <= n; ++ i) {
    f[i] = 1;
    for (int j = 1; j <= i; ++ j)
        if (w[i] > w[j]) f[i] = max(f[i], f[j] + 1);
    ans = max(ans, f[i]);
}
```

- $O(nlog(n))$ 做法

```
int len = 0;
for (int i = 0; i < n; ++ i) {
    int l = 0, r = len;
    while (l < r) {
        int mid = l + r + 1 >> 1;
        if (q[mid] < a[i]) l = mid;
        else r = mid - 1;
    }
    len = max(len, r + 1);
    q[r + 1] = a[i];
}
printf("%d\n", len);
```

5.3 最长公共子序列

```
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);
    }
printf("%d\n", f[n][m]);
```

5.4 区间DP

- 石子合并

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

using namespace std;

const int N = 1010;
int n, m, f[N][N], s[N];

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

    f[0][0] = 0;

    for (int len = 2; len <= n; ++ len)
```

```
        for (int i = 1; i + len - 1 <= n; ++ i)
        {
            int j = i + len - 1;
            f[i][j] = 1e9;
            for (int k = i; k < j; ++ k)
                f[i][j] = min(f[i][j], f[i][k] + f[k + 1][j] + s[j] - s[i - 1]);
        }

    printf("%d\n", f[1][n]);
}
```

- 环形石子合并

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

#define max(x, y) x < y ? x = y : 0
#define min(x, y) x > y ? x = y : 0

using namespace std;

const int N = 205 << 1;
int n, w[N], f[N][N], s[N], g[N][N];

int main() {
    cin.tie(0)->sync_with_stdio(0), cout.tie(0);
    cin >> n;
    for (int i = 1; i <= n; ++ i) cin >> w[i], w[i + n] = w[i];

    for (int i = 1; i <= n * 2; ++ i) s[i] = s[i - 1] + w[i];

    memset(f, 0x3f, sizeof f);
    for (int i = 0; i <= 2 * n; ++ i) f[i][i] = 0;

    for (int len = 2; len <= n; ++ len)
        for (int i = 1; i + len - 1 <= n * 2; ++ i) {
            int j = i + len - 1;
            for (int k = i; k < j; ++ k) {
                min(f[i][j], f[i][k] + f[k + 1][j] + s[j] - s[i - 1]);
                max(g[i][j], g[i][k] + g[k + 1][j] + s[j] - s[i - 1]);
            }
        }

    int mv = 0x3f3f3f3f, mi = -mv;
    for (int i = 1; i <= n; ++ i) {
        min(mv, f[i][i + n - 1]);
        max(mi, g[i][i + n - 1]);
    }
    cout << mv << endl;
    cout << mi << endl;
}
```

图论

1 存图

邻接表

```
int h[N], e[M], ne[M], idx;
void add(int a, int b) { // 加边
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}
```

2 拓扑排序

有向图才有拓扑序列（有向无环图又叫拓扑图）

性质：一个有向无环图一定至少存在一个入度为0的点

- 度数

入度：有向图中某点作为图中边的终点的次数之和

出度：顶点的出边条数称为该顶点的出度

```
bool topsort() {
    int tt = -1, hh = 0;
    for (int i = 1; i <= n; ++ i) if (!d[i]) q[++ tt] = i;
    while (hh <= tt) {
        int t = q[hh ++];
        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];
            d[j] --;
            if (!d[j]) q[++ tt] = j;
        }
    }
    return tt == n - 1;
}
```

3 最短路

n：定点数；m：边数

3.1 单源最短路

3.1.1 所有边权都是正数

3.1.1.1 朴素Dijkstra算法

O(n^2)　　(m, n <= 1e5；即稠密图)

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

using namespace std;

const int N = 510;
int n, m, a, b, c;
int g[N][N]; // 稠密图用邻接矩阵存
```

```cpp
int dist[N]; // 距离起点的距离
bool st[N]; // 该点距离有无确定

int dijkstra() {
    memset(dist, 0x3f, sizeof dist); // 所有点的距离初始化为+∞
    dist[1] = 0;
    for (int i = 0; i < n - 1; ++ i) {
        int t = -1;
        // 在所有没有确定最短距离的点中找到最近的点t
        for (int j = 1; j <= n; ++ j)
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;
        // 用t点更新其他点的最短路
        for (int j = 1; j <= n; ++ j)
            dist[j] = min(dist[j], dist[t] + g[t][j]);
        st[t] = true;
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}

int main() {
    scanf("%d%d", &n, &m);
    memset(g, 0x3f, sizeof g);
    while (m --) {
        scanf("%d%d%d", &a, &b, &c);
        g[a][b] = min(g[a][b], c);
        // 重边取最小即可
    }
    printf("%d\n", dijkstra());
    return 0;
}
```

3.1.1.2 堆优化的Dijkstra算法

O(mlogn) (m,n > 1e5；即稀疏图)

```cpp
// 堆优化dijkstra算法 --- 稀疏图
#include <iostream>
#include <queue>
#include <cstring>

using namespace std;

typedef pair<int, int> PII;

const int N = 1e6 + 10;
int n, m, a, b, c;
int h[N], w[N], e[N], ne[N], idx; // 稀疏图用邻接表存
int dist[N];
bool st[N];

void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

```

```cpp
int dijkstra() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    priority_queue<PII, vector<PII>, greater<PII>> heap;
    // 定义小根堆 first---距离 second---点的编号
    heap.push({0, 1});
    while (heap.size()) {
        // 每次取出距离最短的点
        auto t = heap.top();
        heap.pop();
        int ver = t.second, distance = t.first;
        // 如果ver点的最短路已经确定那么continue
        if (st[ver]) continue;
        st[ver] = true;
        // 更新ver点连的边的最短路
        for (int i = h[ver]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[ver] + w[i]) {
                dist[j] = dist[ver] + w[i];
                heap.push({dist[j], j});
            }
        }
    }
    if (dist[n] == 0x3f3f3f3f) return -1;
    return dist[n];
}

int main() {
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m --) {
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
        // 邻接表不用考虑重边
    }
    printf("%d\n", dijkstra());
    return 0;
}
```

3.1.2 存在负权边

3.1.2.1 Bellman-Ford

O(nm)

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

using namespace std;

const int N = 510, M = 1e4 + 10;
struct Edge {
    int a, b, c;
} edge[M];
int n, m, k, a, b, c;
int dist[N];
int last[N]; // 上次的最短路
```

```

void bellman_ford() {
   memset(dist, 0x3f, sizeof dist);
   dist[1] = 0;
   for (int i = 0; i < k; ++ i) { // 走k次
       memcpy(last, dist, sizeof dist); // 防止出现串联
       for (int j = 0; j < m; j ++) {
           auto e = edge[j];
           dist[e.b] = min(dist[e.b], last[e.a] + e.c);
           // 松弛操作
       } // 三角不等式---dist[b]<=dist[a]+c
   }
}

int main() {
    scanf("%d%d%d", &n, &m, &k);
    for (int i = 0; i < m; ++ i) {
        scanf("%d%d%d", &a, &b, &c);
        edge[i] = {a, b, c};
    }
    bellman_ford();
    // 因为存在负权边可能存在dist[x]==INF-y(y比较小)
    // 所有只要判断>一个比较大的数即可
    if (dist[n] > 0x3f3f3f3f / 2) puts("impossible");
    else printf("%d\n", dist[n]);
    return 0;
}
```

3.1.2.2 SPFA

一般：O(nm)；最坏：O(nm)

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

using namespace std;

const int N = 1e6 + 10;
int n, m, a, b, c;
int h[N], e[N], ne[N], idx, w[N];
int dist[N];
bool st[N];

void add(int a, int b, int c) {
    e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx ++;
}

int spfa() {
    memset(dist, 0x3f, sizeof dist);
    dist[1] = 0;
    queue<int> q;
    q.push(1);
    st[1] = true;
    while (q.size()) {
```

```
        int t = q.front();
        q.pop();
        st[t] = false;
        for (int i = h[t]; i != -1; i = ne[i]) {
            int j = e[i];
            if (dist[j] > dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                if (!st[j]) {
                    q.push(j);
                    st[j] = true;
                }
            }
        }
    }
    return dist[n];
}

int main() {
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m --) {
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
    }
    int t = spfa();
    if (t == 0x3f3f3f3f) puts("impossible");
    else printf("%d\n", t);
    return 0;
}
```

3.2 多源汇最短路

3.2.1 Floyd算法

O(n^3)

源点即起点

汇点即终点

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

using namespace std;

const int N = 210, INF = 1e9;
int n, m, Q;
int d[N][N];
int a, b, c;

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

```

```cpp
int main() {
    scanf("%d%d%d", &n, &m, &Q);
    for (int i = 1; i <= n; ++ i)
        for (int j = 1; j <= n; ++ j)
            if (i == j) d[i][j] = 0;
            else d[i][j] = INF;
    while (m --) {
        scanf("%d%d%d", &a, &b, &c);
        d[a][b] = min(d[a][b], c);
    }
    floyd();
    while (Q --) {
        scanf("%d%d", &a, &b);
        int t = d[a][b];
        if (t > INF / 2) puts("impossible");
        else printf("%d\n", t);
    }
    return 0;
}
```

4 最小生成树

4.1 朴素版Prim

O(n^2)

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

using namespace std;

const int N = 510, INF = 0x3f3f3f3f;
int n, m;
int g[N][N];
int dist[N];
bool st[N]; // 表示一个点是否在集合里
int a, b, c;

int prim() {
    memset(dist, 0x3f, sizeof dist);
    // 初始化成+∞
    int res = 0;
    for (int i = 0; i < n; ++ i) {
        // 找到集合外距离最近的点
        int t = -1;
        for (int j = 1; j <= n; ++ j)
            if (!st[j] && (t == -1 || dist[t] > dist[j]))
                t = j;
        // 如果不能到达集合说明不存在最小生成树
        if (i && dist[t] == INF) return INF;
        // 此句写在31行前避免自环
        if (i) res += dist[t];
        st[t] = true; // 将t加入集合
```

```
        // 用t更新其他点到集合的距离
        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 --) {
        scanf("%d%d%d", &a, &b, &c);
        g[a][b] = g[b][a] = min(g[a][b], c);
    }
    int t = prim();
    if (t == INF) puts("impossible");
    else printf("%d\n", t);
}
```

4.2 堆优化Prim

O(mlogn)

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

using namespace std;

typedef pair<int, int> PII;

const int N = 5e3 + 5, M = 4e5 + 5, INF = 0x3f3f3f3f;
int st[N], dist[N];
int h[N], e[M], ne[M], w[M], idx;
int n, m;

void add(int a, int b, int c) {
    e[++ idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx;
}

int prim() {
    priority_queue<PII, vector<PII>, greater<PII>> q;
    memset(dist, 0x3f, sizeof dist);
    int res = 0, cnt = 0;
    dist[1] = 0;
    q.push({dist[1], 1});

    while (q.size() && cnt < n) {
        int d = q.top().first, t = q.top().second;
        q.pop();
        if (st[t]) continue;
        cnt ++;
        res += d;
        st[t] = 1;
        for (int i = h[t]; i; i = ne[i]) {
```

```
            int j = e[i];
            if (w[i] < dist[j]) {
                dist[j] = w[i];
                q.push({dist[j], j});
            }
        }
    }

    return cnt == n ? res: INF;
}

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

    int t = prim();

    if (t == INF) puts("orz");
    else printf("%d", t);
}
```

4.3 Kruskal

O(mlogm)

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

using namespace std;

const int N = 1e6 + 10, M = 2e6 + 10, INF = 0x3f3f3f3f;
int n, m;
int p[N]; // 并查集数组

struct Edge {
    int a, b, w;
    bool operator < (const Edge &W) const {
        return w < W.w;
    } // 运算符重载，写了可以不用写cmp
} edges[M];

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

int kruskal() {
    sort(edges, edges + m);
    // 将所有点按权重从小到大排序 O(mlogm)
    // 排序算法常数小
    // 所以能跑的比较快
```

```
    for (int i = 1; i <= n; ++ i) p[i] = i;
    // 初始化并查集
    int res = 0, cnt = 0;
    // res表示生成树中所有边权之和
    // cnt表示加入了多少边
    for (int i = 0; i < m; ++ i) {
        // 枚举每条边
        int a = edges[i].a, b = edges[i].b, w = edges[i].w;
        // 如果a, b不连通
        a = find(a), b = find(b);
        if (a != b) {
            p[a] = b; // 将这条边加入集合中
            res += w;
            cnt ++;
        }
    }
    // 如果有边未加入集合说明最小生成树不存在
    if (cnt < n - 1) return INF;
    return res;
}

int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; ++ i) {
        int a, b, w;
        scanf("%d%d%d", &a, &b, &w);
        edges[i] = {a, b, w};
    }
    int t = kruskal();
    if (t == INF) puts("impossible");
    else printf("%d\n", t);
}
```

5 二分图

二分图：节点由两个集合组成，且两个集合内部没有边的图

性质：一个图是二分图当且仅当图中不含奇数环

5.1 染色法

O(n+m)

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

using namespace std;

const int N = 1e6 + 10, M = 2e6 + 10;
int n, m;
int h[N], e[M], ne[M], idx;
int color[N];
int a, b;

```

```cpp
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

bool dfs(int u, int c) {
    color[u] = c;
    // 将u染成c颜色
    for (int i = h[u]; i != -1; i = ne[i]) {
        int j = e[i];
        if (!color[j]) { // 如果没有染色就染色
            if (!dfs(j, 3 - c)) return false; // 1->2, 2->1
        } // 相邻两点颜色不能相同
        else if (color[j] == c) return false;
    }
    return true;
}

int main() {
    scanf("%d%d", &n, &m);
    memset(h, -1, sizeof h);
    while (m --) {
        scanf("%d%d", &a, &b);
        add(a, b), add(b, a); // 无向图
    }
    bool flag = true;
    // 遍历所有点，如果i未染色，将其所在的连通块染色
    // 如果一个点染色，那么这个点所在的连通块的颜色就已确定
    for (int i = 1; i <= n && flag; ++ i)
        if (!color[i] && !dfs(i, 1)) flag = false; // 如果产生矛盾则不是二分图
    puts(flag ? "Yes" : "No");
}
```

5.2 匈牙利算法

O(mn), 实际运行时间一般远小于O(mn) [可能是线性的也说不定]

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

using namespace std;
// y总的奇妙比喻（男生表示左边点，女生表示右边点）
const int N = 510, M = 1e6 + 10;
int n1, n2, m;
int h[N], e[M], ne[M], idx; // 数组越界可能发生任何错误，不只是段错误
int match[N]; // 记录每个女生配对的男生
bool st[N]; // 记录每个女生有没有遍历过
int a, b;

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

bool find(int x) {
    for (int i = h[x]; i != -1; i = ne[i]) {
        int j = e[i];
```

```
        if (!st[j]) {
            st[j] = true;
            if (match[j] == 0 || find(match[j])) { // 如果女生没有配对或能匹配别的男生则匹配
                match[j] = x;
                return true;
            }
        }
    }
    return false;
}

int main() {
    scanf("%d%d%d", &n1, &n2, &m);
    memset(h, -1, sizeof h);
    while (m --) {
        scanf("%d%d", &a, &b);
        add(a, b); // 存男生指向女生即可
    }
    int res = 0;
    for (int i = 1; i <= n1; ++ i) {
        memset(st, false, sizeof st);
        if (find(i)) res ++;
    }
    printf("%d\n", res);
}
```

6 SPFA差分约束与判负环

```
const int N = 1e5 + 10, M = N * 3;
int n, m;
int h[N], e[M], ne[M], w[M], idx;
int dist[N], cnt[N], q[N];
bool st[N];

void add(int a, int b, int c) {
    e[idx] = b, ne[idx] = h[a], w[idx] = c, h[a] = idx ++;
}

bool spfa(int s) {
    memset(dist, 0xcf, sizeof dist);
    int hh = 0, tt = 1;
    q[0] = s;
    dist[s] = 0;
    st[s] = true;

    while (hh != tt) {
        int t = q[-- tt];
        st[t] = false;
        for (int i = h[t]; ~i; i = ne[i]) {
            int j = e[i];
            if (dist[j] < dist[t] + w[i]) {
                dist[j] = dist[t] + w[i];
                cnt[j] = cnt[t] + 1;
                if (cnt[j] >= n + 1) return false;
```

```
                if (!st[j]) {
                    q[tt ++] = j;
                    st[j] = true;
                }
            }
        }
    }

    return true;
}
```

7 LCA

7.1 倍增

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

using namespace std;

const int N = 5e5 + 10, M = N << 1;
int n, m, s;
int x, y;
int h[N], e[M], ne[M], idx;
int d[N], f[N][22], lg[N];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

void dfs(int now, int fa) {
    f[now][0] = fa, d[now] = d[fa] + 1;
    for (int i = 1; i <= lg[d[now]]; ++ i)
        f[now][i] = f[f[now][i - 1]][i - 1];

    for (int i = h[now]; ~i; i = ne[i]) {
        int j = e[i];
        if (j != fa) dfs(j, now);
    }
}

int lca(int x, int y) {
    if (d[x] < d[y]) swap(x, y);
    while (d[x] > d[y])
        x = f[x][lg[d[x] - d[y]] - 1];

    if (x == y) return x;

    for (int k = lg[d[x]] - 1; k >= 0; -- k)
        if (f[x][k] != f[y][k])
            x = f[x][k], y = f[y][k];
    return f[x][0];
}

int main() {
```

```
    memset(h , -1, sizeof h);

    scanf("%d%d%d", &n, &m, &s);

    for (int i = 1; i <= n - 1; ++ i) {
        scanf("%d%d", &x, &y);
        add(x, y), add(y, x);
    }

    for (int i = 1; i <= n; ++ i)
        lg[i] = lg[i - 1] + (1 << lg[i - 1] == i);

    dfs(s, 0);

    while (m --) {
        scanf("%d%d", &x, &y);
        printf("%d\n",lca(x, y));
    }
}
```

7.2 树链剖分

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

using namespace std;

const int N = 4e4 + 10, M = N << 1;
int h[N], e[M], ne[M], idx;
int fa[N], dep[N], siz[N], son[N], top[N], dfn[N], rnk[N], tot;
int n, m, s;

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

void dfs1(int p, int depth, int father) {
    fa[p] = father;
    dep[p] = depth;
    siz[p] = 1;
    int mmax = -1;

    for (int i = h[p]; ~i; i = ne[i]) {
        int j = e[i];
        if (j == father) continue;
        dfs1(j, depth + 1, p);
        if (siz[j] > mmax) {
            mmax = siz[j];
            son[p] = j;
        }
        siz[p] += siz[j];
    }
}

```

```cpp
void dfs2(int p, int tp) {
    top[p] = tp;
    dfn[p] = ++ tot;
    rnk[tot] = p;

    if (!son[p]) return;

    for (int i = h[p]; ~i; i = ne[i]) {
        int j = e[i];
        if (j != fa[p]) {
            if (j != son[p]) dfs2(j, j);
            else dfs2(j, tp);
        }
    }
}

int lca(int x, int y) {
    while (top[x] != top[y]) {
        if (dep[top[x]] >= dep[top[y]]) x = fa[top[x]];
        else y = fa[top[y]];
    }
    return dep[x] < dep[y] ? x : y;
}

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

    scanf("%d", &n);

    for (int i = 1; i <= n; ++ i) {
        int u, v;
        scanf("%d%d", &u, &v);
        if (~v) add(u, v), add(v, u);
        else s = u;
    }

    dfs1(s, 1, s);
    dfs2(s, s);

    scanf("%d", &m);
    while (m --) {
        int a, b;
        scanf("%d%d", &a, &b);
        int Lca = lca(a, b), ans = 0;
        if (Lca == a) ans = 1;
        else if (Lca == b) ans = 2;
        printf("%d\n", ans);
    }
}
```

8 有向图强连通分量SCC

tarjan之后不需要拓扑排序，按照强连通分量逆序即为一个拓扑序

```
const int N = 1e4 + 10, M = 5e5 + 10;
int n, m;
int h[N], e[M], ne[M], idx;
int stk[N], top;
bool in_stk[N];
int dfn[N], low[N], timestamp;
int dout[N];
int id[N], scc_cnt, siz[N];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

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

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

9 无向图的双连通分量

9.1 边双连通分量E-DCC

极大的不含有桥的一个连通区域

```
int h[N], e[M], ne[M], idx;
int n, m;
int dfn[N], low[N], timestamp;
int stk[N], top;
int id[N], dcc_cnt;
bool is_bridge[M];

```

```
void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = 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 = e[i];
        if (!dfn[j]) {
            tarjan(j, i);
            low[u] = min(low[u], low[j]);
            if (dfn[u] < low[j])
                is_bridge[i] = is_bridge[i ^ 1] = true;
        } else if (i != (from ^ 1))
            low[u] = min(low[u], dfn[j]);
    }

    if (dfn[u] == low[u]) {
        ++ dcc_cnt;
        int y;
        do {
            y = stk[top --];
            id[y] = dcc_cnt;
        } while (u != y);
    }
}
```

9.2 点双连通分量V-DCC

极大的不含有割点的一个连通区域

```
const int N = 1010, M = 1010;
int h[N], e[M], ne[M], idx;
int dfn[N], low[N], timestamp;
int n, m, dcc_cnt, root;
int stk[N], top;
vector<int> dcc[N];
bool cut[N]; /*判断i是否为割点*/

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

void tarjan(int u) {
    dfn[u] = low[u] = ++ timestamp;
    stk[++ top] = u;

    if (u == root && h[u] == -1) {
        dcc[++ dcc_cnt].push_back(u);
        return;
    }

    int cnt = 0;
    for (int i = h[u]; ~i; i = ne[i]) {
```

```
        int j = e[i];
        if (!dfn[j]) {
            tarjan(j);
            low[u] = min(low[u], low[j]);
            if (low[j] >= dfn[u]) {
                ++ cnt;
                if (u != root || cnt > 1) cut[u] = true;
                ++ dcc_cnt;
                int y;
                do {
                    y = stk[top --];
                    dcc[dcc_cnt].push_back(y);
                } while (y != j);
                dcc[dcc_cnt].push_back(u);
            }
        } else low[u] = min(low[u], dfn[j]);
    }
}
```

10 欧拉回路与欧拉路径

邻接表

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

using namespace std;

const int N = 1e5 + 10, M = 4e5 + 10;
int h[N], e[M], ne[M], idx;
int n, m, type, din[N], dout[N];
int ans[M >> 1], cnt;
bool used[M];

void add(int a, int b) {
    e[idx] = b, ne[idx] = h[a], h[a] = idx ++;
}

void dfs(int u) {
    for (int &i = h[u]; ~i;) { /*防止被自环图卡*/
        if (used[i]) {
            i = ne[i]; continue;
        }

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

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

```

```
        int j = e[i];
        i = ne[i];
        dfs(j);

        ans[++ cnt] = t;
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);
    memset(h, -1, sizeof h);

    cin >> type >> n >> m;
    for (int i = 1; i <= m; ++ i) {
        int a, b;
        cin >> a >> b;
        add(a, b);
        if (type == 1) add(b, a);
        ++ dout[a], ++ din[b];
    }

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

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

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

    cout << "YES\n";
    for (int i = cnt; i; -- i) cout << ans[i] << ' ';
}
```

邻接矩阵

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

using namespace std;

const int N = 510, M = 1110;
```

```cpp
int n = 500, m;
int g[N][N];
int ans[M], cnt;
int d[N];

void dfs(int u) {
    for (int i = 1; i <= n; ++ i)
        if (g[u][i]) {
            -- g[u][i], -- g[i][u];
            dfs(i);
        }
    ans[++ cnt] = u;
}

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin >> m;
    while (m --) {
        int a, b;
        cin >> a >> b;
        ++ g[a][b], ++ g[b][a];
        ++ d[a], ++ d[b];
    }

    int start = 1;
    for (int i = 1; i <= n; ++ i)
        if (d[i] && d[i] & 1) {
            start = i; break;
        }

    dfs(start);

    for (int i = cnt; i; -- i) cout << ans[i] << '\n';
}
```

11 网络流初步

11.1 EK求最大流

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

using namespace std;

const int N = 1000 + 10, M = 20000 + 10, INF = 1 << 29;
int n, m, S, T;
int h[N], e[M], ne[M], f[M], idx;
int q[N], d[N], pre[N];
bool st[N];

void add(int a, int b, int c) {
    e[idx] = b, f[idx] = c, ne[idx] = h[a], h[a] = idx ++;
    e[idx] = a, f[idx] = 0, ne[idx] = h[b], h[b] = idx ++;
```

```
}

bool bfs() {
    int hh = 0, tt = -1;
    memset(st, false, sizeof st);
    q[++ tt] = S;
    st[S] = true;
    d[S] = INF;

    while (hh <= tt) {
        int t = q[hh ++];
        for (int i = h[t]; ~i; i = ne[i]) {
            int ver = e[i];
            if (!st[ver] && f[i]) {
                st[ver] = true;
                d[ver] = min(d[t], f[i]);
                pre[ver] = i;
                if (ver == T) return true;
                q[++ tt] = ver;
            }
        }
    }
    return false;
}

int ek() {
    int maxflow = 0;
    while (bfs()) {
        maxflow += d[T];
        for (int i = T; i != S; i = e[pre[i] ^ 1]) {
            f[pre[i]] -= d[T];
            f[pre[i] ^ 1] += d[T];
        }
    }
    return maxflow;
}

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

11.2 dinic求最大流

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

using namespace std;

const int N = 1e5 + 10, M = N << 1, INF = 0x3f3f3f3f;

```

```cpp
int n, m, S, T;
int h[N], e[M], ne[M], f[M], idx;
int q[N], d[N], cur[N];

void add(int a, int b, int c) {
    e[idx] = b, ne[idx] = h[a], f[idx] = c, h[a] = idx ++;
    e[idx] = a, ne[idx] = h[b], f[idx] = 0, h[b] = idx ++;
}

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 = e[i];
            if (d[ver] == -1 && f[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 = e[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 r = 0, flow;
    while (bfs()) while (flow = find(S, INF)) r += flow;
    return r;
}

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

```
    }
    printf("%d\n", dinic());
    return 0;
}
```

11.3 点分裂

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

#define debug

using namespace std;

const int N = 55 << 1, M = (N * N) << 1, INF = 0x3f3f3f3f;
int n, m, S, T;
int h[N], e[M], ne[M], f[M], fs[M], idx;
bool g[N][N];
int q[N], cur[N], d[N];

void add(int a, int b, int c) {
    e[idx] = b, ne[idx] = h[a], f[idx] = c, h[a] = idx ++;
    e[idx] = a, ne[idx] = h[b], f[idx] = 0, h[b] = idx ++;
}

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 = e[i];
            if (d[ver] == -1 && f[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 = e[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 r = 0, flow;
    while (bfs()) while (flow = find(S, INF)) r += flow;
    return r;
}

void init() {
    memset(h, -1, sizeof h);
    memset(g, 0, sizeof g);
    memset(f, 0, sizeof f);
    idx = 0;
}

int main() {
    while (scanf("%d%d", &n, &m) != EOF) {
        init();

        for (int i = 1; i <= m; ++ i) {
            int a, b;
            scanf(" (%d,%d)", &a, &b);
            a ++, b ++;
            add(a + n, b, INF);
            add(b + n, a, INF);
            g[a][b] = g[b][a] = true;
        }
        for (int i = 1; i <= n; ++ i) add(i, i + n, 1);
        memcpy(fs, f, sizeof f);

        int res = n;
        for (S = n + 1; S <= 2 * n; ++ S)
            for (T = S - n + 1; T <= n; ++ T)
                if (!g[S - n][T]) {
                    memcpy(f, fs, sizeof f);
                    res = min(res, dinic());
                }

        printf("%d\n", res);
    }
}
```# 数学知识

数论

试除法判定质数

```c++
#include <iostream>

using namespace std;

int n, a;

```

```cpp
bool is_prime(int n) {
    if (n <= 1) return false;
    for (int i = 2; i <= n / i; ++ i) // 此处不要写成i*i<=n或者i<=sqrt(n)
        if (n % i == 0) return false; // 前者会溢出后者较慢
    return true;
}

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d", &a);
        puts(is_prime(a) ? "Yes" : "No");
    }
}
```

11.4 分解质因数

```cpp
#include <iostream>

using namespace std;

int n, a;

void divide(int n) {
    for (int i = 2; i <= n / i; ++ i) {
        if (n % i == 0) {
            int s = 0;
            while (n % i == 0) n /= i, s ++;
            printf("%d %d\n", i, s);
        }
    } // n至多有一个大于根号n的质因子
    if (n > 1) printf("%d 1\n", n);
    puts("");
}

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d", &a);
        divide(a);
    }
}
```

11.5 筛质数

```cpp
#include <iostream>

using namespace std;

const int N = 1e6 + 10;
int n;
int prime[N], cnt;
bool st[N];

void get_primes(int n) {
    for (int i = 2; i <= n; ++ i) {
```

```
        if (!st[i]) prime[cnt ++] = i;
        for (int j = 0; prime[j] <= n / i; ++ j) {
            st[prime[j] * i] = true;
            if (i % prime[j] == 0) break;
        }
    }
} // 一个数只用其最小质因数筛掉

int main() {
    scanf("%d", &n);
    get_primes(n);
    printf("%d\n", cnt);
}
```

11.6 试除法求约数

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

using namespace std;

int n, a;

vector<int> get_divisors(int n) {
    vector<int> res;
    for (int i = 1; i <= n / i; ++ i) {
        if (n % i == 0) {
            res.push_back(i);
            if (i != n / i) res.push_back(n / i);
        }
    }
    sort(res.begin(), res.end());
    return res;
}

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d", &a);
        auto res = get_divisors(a);
        for (auto i : res) printf("%d ", i);
        puts("");
    }
}
```

11.7 约数个数

```
#include <iostream>
#include <unordered_map>

using namespace std;

typedef long long LL;

const int MOD = 1e9 + 7;
```

```
int n, a;
unordered_map<int, int> primes;
LL res = 1;

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d", &a);
        for (int i = 2; i <= a / i; ++ i)
            while (a % i == 0) a /= i, primes[i] ++;
        if (a > 1) primes[a] ++;
    }
    for (auto i : primes) res = res * (i.second + 1) % MOD; // 约数个数其实就是质因数的组合数
    printf("%lld\n", res);
}
```

11.8 约数之和

```
#include <iostream>
#include <unordered_map>

using namespace std;

typedef long long LL;

const int MOD = 1e9 + 7;
int n, a;
unordered_map<int, int> primes;
LL res = 1;

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d", &a);
        for (int i = 2; i <= a / i; ++ i)
            while (a % i == 0) a /= i, primes[i] ++;
        if (a > 1) primes[a] ++;
    }
    for (auto i : primes) {
        int fi = i.first, se = i.second;
        LL t = 1;
        while (se --) t = (t * fi + 1) % MOD;
        res = res * t % MOD;
    }
    printf("%lld\n", res);
}
```

11.9 最大公约数

```
#include <iostream>

using namespace std;

int n, a, b;

```

```cpp
int gcd(int a, int b) {
    return b ? gcd(b, a % b) : a;
}

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d%d", &a, &b);
        printf("%d\n", gcd(a, b));
    }
}
```

12 快速幂

```cpp
int q_pow(int a, int b, int c) {
    int ans = 1;
    while (b) {
        if (b & 1) ans = (LL)ans * a % c;
        a = (LL)a * a % c;
        b >>= 1;
    }
    return ans;
}
```

12.1 快速幂求逆元

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

int n, a, p;

int q_pow(int a, int b, int k) {
    int ans = 1;
    while (b) {
        if (b & 1) ans = (LL)ans * a % k;
        a = (LL)a * a % k;
        b >>= 1;
    }
    return ans;
}

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%d%d", &a, &p);
        int res = q_pow(a, p - 2, p);
        if (a % p) printf("%d\n", res);
        else puts("impossible");
    }
}
```

12.2 线性逆元

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

using namespace std;

typedef long long LL;

const int N = 20000528 + 10;
int n, p;
int inv[N];

int main() {
    scanf("%d%d", &n, &p);
    inv[1] = 1;
    for (int i = 2; i <= n; ++ i)
        inv[i] = (LL)(p - p / i) * inv[p % i] % p;
    for (int i = 1; i <= n; ++ i)
        printf("%d\n", inv[i]);
    return 0;
}
```

13 拓展欧几里得

```cpp
int exgcb(int a, int b, int &x, int &y) {
    if (!b) {
        x = 1, y = 0;
        return a;
    }
    int d = exgcb(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

13.1 线性同余方程

```cpp
#include <iostream>

using namespace std;

int n, a, b, m, x, y;

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() {
    scanf("%d", &n);
    while (n --) {
```

```
        scanf("%d%d%d", &a, &b, &m);
        int d = exgcd(a, m, x, y);
        if (b % d == 0) printf("%d\n", x * 1ll * b / d % m);
        else printf("impossible\n");
    }
}
```

14 中国剩余定理

1

```
#include <iostream>

using namespace std;

typedef long long LL;

int n;
bool has_answer = true;

LL exgcd(LL a, LL b, LL &x, LL &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() {
    cin >> n;
    LL a1, m1;
    cin >> a1 >> m1;

    for (int i = 1; i < n; ++ i) {
        LL a2, m2;
        cin >> a2 >> m2;

        LL k1, k2;
        LL d = exgcd(a1, a2, k1, k2);

        if ((m2 - m1) % d) {
            has_answer = false;
            break;
        }

        k1 *= (m2 - m1) / d;
        LL t = a2 / d;
        k1 = (k1 % t + t) % t;

        m1 = a1 * k1 + m1;
        a1 = abs(a1 / d * a2);
    }
```

```
    cout << (has_answer ? (m1 % a1 + a1) % a1 : -1) << endl;
}
```

2

```
#include <iostream>
#include <cmath>

using namespace std;

typedef long long LL;

int n, p, e, i, d;
LL a[4]; int w[] = {0, 23, 28, 33};

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 ChRe() {
    int n = 1;
    for (int i = 1; i <= 3; ++ i) a[i] %= w[i], n *= w[i];
    LL ans = 0;
    for (int i = 1; i <= 3; ++ i) {
        int m = n / w[i];
        LL x, y;
        exgcd(m, w[i], x, y);
        ans = (ans + m * x * a[i]) % n;
    }
    ans = (ans % n + n) % n;
    ans -= d;
    while (ans <= 0) ans += n;
    return ans;
}

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

    while (n --) {
        LL Case = 0;
        while (scanf("%lld%lld%lld%lld", &a[1], &a[2], &a[3], &d) != EOF) {
            if (a[1] == -1 && a[2] == -1 && a[3] == -1 && d == -1) break;
            LL ans = ChRe();
            printf("Case %lld: the next triple peak occurs in %lld days.\n", ++ Case,
ans);
        }
    }
}
```

15 组合计数

公式：(a 选 b)

$$C_a^b = \frac{a!}{(a-b)!b!}$$

递推式：

$$C_a^b = C_a^{b-1} + C_{a-1}^{b-1}$$
$$C_a^0 = 1$$
$$C_a^b = 0(a < b)$$
$$\sum_{i=0}^{n} C_i^n = 2^n$$

性质：

$$C_m^n = C_m^{m-n}$$

数据范围较小时：a,b <= 2e3

```cpp
#include <iostream>

using namespace std;

const int N = 2e3 + 10, MOD = 1e9 + 7;
int c[N][N];
int n, a, b;

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();
    scanf("%d", &n);
    while (n --) {
        scanf("%d%d", &a, &b);
        printf("%d\n", c[a][b]);
    }
}
```

a,b较大时：a,b <= 1e7

```cpp
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 1e6 + 10, MOD = 1e9 + 7;
int fact[N], infact[N];
```

```cpp
int n, a, b;

int q_pow(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1) res = (LL)res * a % MOD;
        a = (LL)a * a % MOD;
        b >>= 1;
    }
    return res;
}

int C(int a, int b) {
    return (LL)fact[a] * infact[b] % MOD * infact[a - b] % MOD;
}

int main() {
    fact[0] = infact[0] = 1;
    for (int i = 1; i < N; ++ i) {
        fact[i] = (LL)fact[i - 1] * i % MOD;
        infact[i] = (LL)infact[i - 1] * q_pow(i, MOD - 2) % MOD;
    }

    scanf("%d", &n);
    while (n --) {
        scanf("%d%d", &a, &b);
        printf("%d\n", C(a, b));
    }
}
```

a,b 巨大时：a,b<=1e18 p <=1e5

```cpp
#include <iostream> // 卢卡斯定理

using namespace std;

typedef long long LL;
int n, p;
LL a, b;

int q_pow(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 C(int a, int b, int p) {
    if (b > a) return 0;

    int res = 1;
    for (int i = 1, j = a; i <= b; ++ i, -- j) {
        res = (LL)res * j % p;
```

```
        res = (LL)res * q_pow(i, p - 2, p) % p;
    }
    return res;
}

int lucas(LL a, LL b, int p) {
    if (a < p && b < p) return C(a, b, p);
    return (LL)C(a % p, b % p, p) * lucas(a / p, b / p, p) % p;
}

int main() {
    scanf("%d", &n);
    while (n --) {
        scanf("%lld%lld%d", &a, &b, &p);
        printf("%d\n", lucas(a, b, p));
    }
}
```

朴素做法：高精

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

using namespace std;

const int N = 5010;
int primes[N], cnt;
bool st[N];
int sum[N];
int a, b;
vector<int> res = {1};

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() || t; ++ i) {
        if (i < a.size()) t += a[i] * b;
```

```
        c.push_back(t % 10);
        t /= 10;
    }
    while (c.size() > 1 && c.back() == 0) c.pop_back();
    return c;
}

int main() {
    scanf("%d%d", &a, &b);
    get_primes(a);
    for (int i = 0; i < cnt; ++ i) {
        int p = primes[i];
        sum[i] = get(a, p) - get(a - b, p) - get(b, p);
    }

    for (int i = 0; i < cnt; ++ i)
        for (int j = 0; j < sum[i]; ++ j)
            res = mul(res, primes[i]);

    for (int i = res.size() - 1; i >= 0; -- i) printf("%d", res[i]);
    puts("");
}
```

卡特兰数

$$ktl_n = \sum_{i=0}^{n-1} ktl_i * ktl_{n-1-i}$$

$$ktl_1 = ktl_0 = 1$$

$$ktl_n = \frac{C_{2n}^n}{n+1}$$

```
#include <iostream>

using namespace std;

typedef long long LL;

const int MOD = 1e9 + 7;

int q_pow(int a, int b) {
    int res = 1;
    while (b) {
        if (b & 1) res = (LL)res * a % MOD;
        a = (LL)a * a % MOD;
        b >>= 1;
    }
    return res;
}

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

    int a = 2 * n, 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 * q_pow(i, MOD - 2) % MOD;
    res = (LL)res * q_pow(n + 1, MOD - 2) % MOD;

    printf("%d\n", res);
}
```

16 高斯消元

```
#include <iostream>
#include <cmath>

using namespace std;

const int N = 110;
const double eps = 1e-6;
int n;
double a[N][N];

bool lf_is_same(double a, double b) {
    return fabs(a - b) < eps;
}

int gauss() {
    int c, r; // c--列 r--行
    for (c = 0, r = 0; c < n; ++ c) {
        int t = r; // t--绝对值最大的行
        for (int i = r; i < n; ++ i)
            if (fabs(a[i][c]) > fabs(a[t][c])) t = i;
        // 如果是0就continue
        if (lf_is_same(a[t][c], 0)) continue;
        // 将绝对值最大行放在最上方
        for (int i = c; i < n + 1; ++ i) swap(a[t][i], a[r][i]);
        for (int i = n; i >= c; -- i) a[r][i] /= a[r][c]; // 将改行第一个系数变为1
        // 将下面所有行的第一个系数变为0
        for (int i = r + 1; i < n; ++ i)
            if (!lf_is_same(a[i][c], 0))
                for (int j = n; j >= c; -- j) a[i][j] -= a[r][j] * a[i][c];
        // 行数++
        r ++;
    }
    // 如果行数<n
    if (r < n) {
        for (int i = r; i < n; ++ i)
            if (!lf_is_same(0, a[i][n])) return 2; // 如果出现0=非0则无解
        return 1; // 如果出现0=0
    }

    for (int i = n - 1; i >= 0; -- i)
        for (int j = i + 1; j < n; ++ j) a[i][n] -= a[j][n] * a[i][j]; // 将a[i][n]转换成方程的解
    return 0;
}

```

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

    int t = gauss();
    if (!t) for (int i = 0; i < n; ++ i) printf("%.2lf\n", a[i][n]);
    else if (t == 1) puts("Infinite group solutions");
    else puts("No solution");
}
```

17 简单博弈论

Nim游戏

台阶-Nim游戏

集合-Nim游戏

拆分-Nim游戏

18 容斥原理

```
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 20;
int n, m;
int p[N];

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

    int res = 0;
    for (int i = 1; i < 1 << m; ++ i) { // 不能从i=0开始因为一定要选一个集合
        int t = 1, s = 0; // t表示所选质数乘积 s表示所选集合个数
        for (int j = 0; j < m; ++ j)
            if (i >> j & 1) {
                s ++;
                if ((LL)t * p[j] > n) {
                    t = -1; break;
                }
                t *= p[j];
            }
        if (t == -1) continue;
        // 奇数个+ 偶数个-
        if (s & 1) res += n / t;
```

```
        else res -= n / t;
    }

    printf("%d\n", res);
}
```

19 拓展欧拉定理

P5091 【模板】扩展欧拉定理

```
#include <iostream>

using namespace std;

typedef long long LL;

const int N = 20000000 + 10;
char b[N];

LL mul(LL a, LL b, LL m) { // 龟速乘
    LL res = 0;
    while (b) {
        if (b & 1) res = (res + a) % m;
        a = (a + a) % m;
        b >>= 1;
    }
    return res;
}

LL q_pow(LL a, LL b, LL MOD) {
    LL res = 1;
    while (b) {
        if (b & 1) res = mul(res, a, MOD);
        a = mul(a, a, MOD);
        b >>= 1;
    }
    return res;
}

LL phi(LL x) {
    LL res = x;
    for (LL i = 2; i <= x / i; i ++ )
        if (x % i == 0)
        {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);

    return res;
}

int main() {
    LL a, m; bool falg = false;
    cin >> a >> m >> b + 1;
    LL p = phi(m);
```

```
    LL ans = 0;
    for (LL i = 1; b[i]; ++ i) {
        ans = ans * 10 + b[i] - '0';
        if (ans >= p) {
            falg = true;
            ans %= p;
        }
    }
    if (falg) ans += p;
    cout << q_pow(a, ans, m) << endl;
}
```