# 华理小男孩模板

| | | |
|---|---|---|
| 王亦凡 | 姚远 | 王泽宸 |
| 主代码手 | 队长 | 解题核心 |

October 9, 2017

# Contents

# Chapter 1

# 数论

## 1.1 指数降幂公式

$$A^x \equiv A^{x \bmod \phi(p)+\phi(p)} \pmod p (x \geq \phi(p))$$

## 1.2 威尔逊定理

$$(p-1)! \equiv -1 \pmod p$$

## 1.3 费马小定理

$$a^p \equiv a \pmod p$$

## 1.4 欧拉定理

$$a^\phi(n) \equiv 1 \pmod n$$

## 1.5 质数表

## 1.6 素数函数

## 1.7 欧拉函数

### 1.7.1 递推求

### 1.7.2 单个求

## 1.8 莫比乌斯函数

## 1.9 逆元

### 1.9.1 递推求

### 1.9.2 单个求

用费马小定理

# Chapter 2

# 概率论

## 2.1 超几何分布

超几何分布是统计学上一种离散概率分布。它描述了由有限个物件中抽出 n 个物件，成功抽出指定种类的物件的个数（不归还）

例如在有 N 个样本，其中 m 个是不及格的。超几何分布描述了在该 N 个样本中抽出 n 个，其中 k 个是不及格的概率：

$$f(k;n,m,N)=\frac{\binom{m}{k}\binom{N-m}{n-k}}{\binom{N}{n}}$$

# Chapter 3

# 数学

## 3.1 矩阵

### 3.1.1 矩阵类

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1000, MOD = 1E9 + 7;
struct Mat
{
    int a[N][N];
    int n, m;
    Mat(int n, int m) : n(n), m(m) {memset(a, 0, sizeof(a));}
    void eye()
    {
        memset(a, 0, sizeof(a));
        for (int i = 0; i < n; i++) a[i][i] = 1;
    }
    void print()
    {
        for (int i = 0; i < n; i++) {
            cout << endl;
            for (int j = 0; j < m; j++)
                cout << '␣' << a[i][j];
        }
    }
};
Mat mul(Mat A, Mat B)
{
    Mat t(A.n, B.m);
    for (int i = 0; i < A.n; i++)
        for (int j = 0; j < B.m; j++)
            for (int k = 0; k < A.m; k++)
                t.a[i][j] = (t.a[i][j] + A.a[i][k] * B.a[k][j] % MOD) % MOD;
    return t;
}
Mat pow(Mat A, int n)
{
    Mat t(A.n, A.m);
    t.eye();
    while (n > 0) {
        if (n % 2) t = mul(t, A);
        A = mul(A, A);
        n /= 2;
    }
    return t;
}
int det(Mat A)
{
  int cnt = 0,ans = 1, n = A.n;
  for(int i = 0; i < n; i++) {
    for(int j = i+1; j < n; j++) {
      int x = i, y = j;
      while(A.a[y][i]) {
        int t = A.a[x][i] / A.a[y][i];
        for(int k = 0; k < n; k++) {
          A.a[x][k] = A.a[x][k] - A.a[y][k]*t;
        }
        swap(x, y);
      }
      if(x != i) {
        for(int k = 0; k < n; k++) {
          swap(A.a[x][k], A.a[y][k]);
        }
        cnt ^= 1;
      }
    }
    if(A.a[i][i] == 0) return 0;
    else ans *= A.a[i][i];
  }
  if(cnt) ans *= -1;
    return ans;
}
```

### 3.1.2 高斯消元

```
double a[MAXN][MAXN];
double ans[MAXN];
bool f[MAXN];//自由变量
int sgn(double x) {return (x > eps) - (x < -eps);}
//x 0~equ - 1, y 0~var
int gauss(int equ, int var) {
    int k = 0, col = 0;
    memset(f, true, sizeof(f));
    for(k = 0; k < equ && col < var; k++, col++) {
        int r = k;
        for(int i = k + 1; i < equ; i++)
            if(fabs(a[i][col]) > fabs(a[r][col])) r = i;
        if(r != k) for(int j = k; j <= var; j++) swap(a[r][j], a[k][j]);
        if(a[k][col] == 0) {k--; continue;}
        for(int i = k + 1; i < equ; i++) if(a[i][col]) {
            for(int j = var; j >= col; j--) a[i][j] -= a[i][col] / a[k][col] * a[k][j];
        }
    }
    for(int i = k; i < equ; i++) if(sgn(a[i][col]) != 0) return 0;
    if(k < var) {
        for(int i = k - 1; i >= 0; i--) {
            int cnt = 0, p;
            for(int j = 0; j < var; j++)
                if(sgn(a[i][j]) && f[j])
                    cnt++, p = j;
            if(cnt > 1) continue;
            double t = a[i][var];
            for(int j = 0; j < var; j++)
                if(sgn(a[i][j]) && j != p)
                    t -= a[i][j] * ans[j];
            ans[p] = t / a[i][p];
            f[p] = 0;
        }
    }
    for(int i = var - 1; i >= 0; i--) {
        double t = a[i][var];
        for(int j = i + 1; j < var; j++)
            if(sgn(a[i][j]))
                t -= a[i][j] * ans[j];
        ans[i] = t / a[i][i];
    }
    return 1;
}
```

## 3.2 整除与剩余

### 3.2.1 扩展欧几里得逆元

```
扩展欧几里得
//ax + by = d, d = gcd(a, b)
void gcd(ll a , ll b ,ll &d, ll &x,ll &y){
    if(!b) {d = a; x = 1; y = 0; return;}
    else{
        gcd(b , a % b ,d , y , x);
        y -= x * (a / b);
        return;
    }
}

逆元
//a在模n下的逆元，a和n互素才有逆元
ll inv(ll a, ll n) {
    ll d, x, y;
    gcd(a, n, d, x, y);
    return d == 1 ? (x + n) % n : -1;
}
```

### 3.2.2 中国剩余定理

```
// n个方程 x   a[i](mod m[i]) i = 0..n-1
ll china(int n, ll *a, ll *m)
{
    ll M = m[0], R = a[0];
    for (int i = 1; i < n; i++) {
        ll d = __gcd(M, m[i]);
        ll c = a[i] - R;
        if (c % d) return -1;
        ll k1, k2;
        extgcd(M,m[i],d,k1, k2);
        k1 = (c /d * k1) % (m[i]/d);
        R = R + k1 * M;
        M = M /d * m[i];
        R %= M;
    }
    if (R < 0) R += M;
    return R;
}
```

## 3.3 数值计算

## 3.4 其他

### 3.4.1 lucas 定理

```
ll qPow (ll a, ll k) {
    ll ans = 1;
    while (k) {
        if (k&1)
            ans = (ans * a) % p;
        a = (a * a) % p;
        k /= 2;
    }
    return ans;
}

ll C (ll a, ll b, ll p) {
    if (a < b)
        return 0;
    if (b > a - b)
        b = a - b;

    ll up = 1, down = 1;
    for (ll i = 0; i < b; i++) {
        up = up * (a-i) % p;
        down = down * (i+1) % p;
    }
    return up * qPow(down, p-2) % p; // 逆元
}
ll lucas (ll a, ll b, ll p) {
    if (b == 0)
        return 1;
    return C(a%p, b%p, p) * lucas(a/p, b/p, p) % p;
}
```

### 3.4.2 递推求组合数

```
void calc()
{
    for (int i = 0; i < N; i++) {
        c[i][0] = c[i][i] = 1;
        for (int j = 1; j < i; j++) {
            c[i][j] = c[i-1][j] + c[i-1][j-1];
        }
    }
}
```

### 3.4.3 单个求组合数

```
ll C(ll n, ll m)
{
        ll c = 1;
        for (int i = 1; i <= m; i++) {
                c *= (n - m + i);
                if (c % i == 0) c /= i;
        }
        return c;
}
```

### 3.4.4 威佐夫博弈

```
double gold = (1 + sqrt(5)) / 2;
if (m == floor(((n - m) * gold))) cout << 'G' << endl; else cout << 'B' << endl;
```

### 3.4.5 FWT

```
//位运算多项式卷积
void FWT(int *a, int n) {
    for(int d = 1; d < n; d<<=1)
        for(int m = d<<1, i = 0; i < n; i += m)
            for(int j = 0; j < d; j++) {
                int x = a[i + j], y = a[i + j + d];
                a[i + j] = (x + y) % MOD, a[i + j + d] = (x - y + MOD) % MOD;
                //xor:a[i+j]=x+y,a[i+j+d]=(x-y+mod)%mod;
                //and:a[i+j]=x+y;
                //or:a[i+j+d]=x+y;
            }
}
void UFWT(int *a, int n) {
    for(int d = 1; d < n; d <<= 1)
    for(int m = d<<1, i = 0; i < n; i += m)
      for(int j = 0; j < d; j++) {
        int x = a[i + j], y = a[i + j + d];
        a[i + j] = (ll)(x + y) * inv2 % MOD, a[i + j + d] = ((ll)(x - y) * inv2 % MOD + MOD) % MOD;
        //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
        //and:a[i+j]=x-y;
        //or:a[i+j+d]=y-x;
      }
}
void conv(int *a, int *b, int n) {
  FWT(a, n);
  FWT(b, n);
  for(int i = 0; i < n; i++) a[i] = (ll)a[i] * b[i] % MOD;
  UFWT(a, n);
}
```

### 3.4.6 FFT

```
const double pi = atan(1.0) * 4;

struct complex {
    double a, b;
    complex(double aa = 0.0, double bb = 0.0) { a = aa; b = bb; }
    complex operator +(const complex &e) { return complex(a + e.a, b + e.b); }
    complex operator -(const complex &e) { return complex(a - e.a, b - e.b); }
    complex operator *(const complex &e) { return complex(a * e.a - b * e.b, a * e.b + b * e.a); }
};

void change(complex * y, long long len) {
    long long i, j, k;
    for (i = 1, j = len / 2; i < len - 1; i++) {
        if (i < j) swap(y[i], y[j]);
        k = len / 2;
        while (j >= k) {
            j -= k;
            k /= 2;
        }
```

```
                if (j < k) j += k;
        }
}

void fft(complex *y, long long len, long long on) {
        change(y, len);
        for (int h = 2; h <= len; h <<= 1) {
                complex wn(cos(-on * 2 * pi / h), sin(-on
                    * 2 * pi / h));
                for (int j = 0; j < len; j += h) {
                        complex w(1, 0);
                        for (int k = j; k < j + h / 2; k
                           ++) {
                                complex u = y[k];
                                complex t = w * y[k + h /
                                    2];
                                y[k] = u + t;
                                y[k + h / 2] = u - t;
                                w = w * wn;
                        }
                }
        }
        if (on == -1)
                for (int i = 0; i < len; i++)
                        y[i].a /= len;
}
```

### 3.4.7 hell 方程

$x^2 - ny^2 = 1$ $x[i+1] = x[1] * x[i] + n * y[1] * y[i]$; $y[i+1] = x[1] * y[i] + y[1] * x[i]$

# Chapter 4

# 图论

## 4.1 图的遍历和连通性

### 4.1.1 割点和桥

```
int pre[MAXN], iscut[MAXN], dfs_clock = 0;
int dfs(int u, int fa) {
int lowu = pre[u] = ++dfs_clock;
int child = 0;
for(int i = 0; i < g[u].size(); i++) {
int v = g[u][i];
  if(!pre[v]) {
child++;
        int lowv = dfs(v, u);
      lowu = min(lowu, lowv);
      if(lowv >= pre[u]) iscut[u] = true;
//if lowv > pre[u] (u, v) is bridge

        }
        else if(pre[v] < pre[u] && v != fa) lowu = min(
           lowu, pre[v]);
}
if(fa < 0 && child == 1) iscut[u] = false;
    return lowu;
}
```

### 4.1.2 双连通分量

```
int dfs(int u, int fa) {
    int lowu = pre[u] = ++dfs_clock;
    int child = 0;
    for(int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if(!pre[v]) {
            s.push(node(u, v));
            child++;
            int lowv = dfs(v, u);
            lowu = min(lowu, lowv);
            if(lowv >= pre[u]) {
                iscut[u] = true;
                bcc[++bcc_cnt].clear();
                for(;;) {
                    node temp = s.top(); s.pop();
                        //注意割顶可能包含在多个bcc中，bccno不是唯一的标准
                    if(bccno[temp.u] != bcc_cnt)
                {bcc[bcc_cnt].push_back(temp.u); bccno[temp.u] = bcc_cnt;}
                    if(bccno[temp.v] != bcc_cnt)
                {bcc[bcc_cnt].push_back(temp.v); bccno[temp.v] = bcc_cnt;}
                    if(temp.u == u && temp.v == v) break;
                }
            }
        }
        else if(pre[v] < pre[u] && v != fa) {
            s.push(node(u, v));
            lowu = min(lowu, pre[v]);
        }
    }
    if(fa < 0 && child == 1) iscut[u] = false;
    return lowu;
}
void find_bcc(int n) {
    memset(pre, 0, sizeof(pre));
    memset(iscut, false, sizeof(iscut));
    memset(bccno, 0, sizeof(bccno));
    dfs_clock = bcc_cnt = 0;
    for(int i = 1; i <= n; i++) {
        if(!pre[i]) dfs(i, -1);
    }
}
```

### 4.1.3 强连通分量

```
int dfs(int u) {
    pre[u] = low[u] = ++dfs_clock;
    s.push(u);
    for(int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if(!pre[v]) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
        else if(!sccno[v]) {
            low[u] = min(low[u], pre[v]);
        }
}
//如果low[u] == pre[u]，那么它就是这个scc的第一个点
    if(low[u] == pre[u]) {
        scc_cnt++;
        for(;;) {
            int temp = s.top(); s.pop();
            sccno[temp] = scc_cnt;
            if(temp == u) break;
        }
    }
}
void find_scc(int n) {
    memset(pre, 0, sizeof(pre));
    memset(low, 0, sizeof(low));
    memset(sccno, 0, sizeof(sccno));
    dfs_clock = scc_cnt = 0;
    for(int i = 1; i <= n; i++) {
        if(!pre[i]) dfs(i);
    }
}
```

### 4.1.4 拓扑排序

**BFS**

**判断是否成环**

拓扑排序形成的 ans 的 sz != n 则成环

### 4.1.5 2SAT

```
//2-sat中不能走的决策往另一个决策连一条边
int dfs(int u) {
    if(vis[u^1]) return false;
    if(vis[u]) return true;
    s[cnt++] = u;
    vis[u] = 1;
    for(int i = 0; i < g[u].size(); i++) if(!dfs(g[u][i])) return false;
    return true;
}
bool flag = true;
//一定记得+=2
for(int i = 2; i <= 2 * n; i+= 2) {
    if(!vis[i] && !vis[i + 1]) {
        cnt = 0;
        if(!dfs(i)) {
            while(cnt) vis[s[--cnt]] = 0;
            if(!dfs(i + 1)) {flag = false; break;}
        }
    }
}
```

## 4.2 路径

### 4.2.1 非递归欧拉回路

```
void euler(int u) {
    stack<int> st;
    st.push(u);
    nxt[st.size()] = -1;
    while(!st.empty()) {
        int a = st.top();
        int i;
        for(i = last[a]; i < 26; i++) if(!vis[a][i]) {
            vis[a][i] = 1;
            st.push(g[a][i]);
            nxt[st.size()] = i;
            last[a] = i + 1;
            break;
        }
        if(i == 26) {
            if(nxt[st.size()] != -1) ans.push_back((char)(nxt[st.size()] + 'a'));
            st.pop();
        }
    }
}
```

## 4.3 匹配

### 4.3.1 二分图最大匹配

最小点覆盖的点数 = 二分图最大匹配
最大独立集的点数 = 总点数 - 二分图最大匹配

```
int uN,vN;
vector<int> g[MAXN];
int linker[MAXN];
bool used[MAXN];
bool dfs(int u) {
    for(int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if(!used[v]) {
            used[v]=true;
            if(linker[v] == -1 || dfs(linker[v])) {
                linker[v]=u;
                return true;
            }
        }
    }
    return false;
}
int hungary() {
    int res=0;
    memset(linker, -1, sizeof(linker));
    for(int u=0; u < uN; u++)
    {
        memset(used, 0, sizeof(used));
        if(dfs(u)) res++;
    }
    return res;
}
```

### 4.3.2 二分图最优匹配

```
struct KM // 二分图最优匹配
{
    int n; //总点数
    vector<int> g[N];
    int g2[N][N];
    void init(int nn)
    {
        n = nn;
        mem(g, 0), mem(g2, 0);
    }
    void add_edge(int u, int v, int w)
    {
        g[u].push_back(v);
        g2[u][v] = w;
    }
    int lx[N], ly[N], match[N], lcheck[N], rcheck[N];
    const int INF = INT_MAX;
    bool dfs(int u)
    {
        lcheck[u] = true;
        for (int v : g[u]) {
            if (lx[u] + ly[v] == g2[u][v] && !rcheck[v]) {
                rcheck[v] = true;
                if (match[v] == -1 || dfs(match[v])) {
                    match[v] = u;
                    return true;
                }
            }
        }
        return false;
    }
    void update()
    {
        int a = INF;
        rep(u, 1, n) {
            if (lcheck[u]) {
                for (int v : g[u]) {
                    if (!rcheck[v]) {
                        a = min(a, lx[u] + ly[v] - g2[u][v]);
                    }
                }
            }
        }
        rep(i, 1, n) {
            if (lcheck[i]) lx[i] -= a;
            if (rcheck[i]) ly[i] += a;
        }
    }
    int calc()
    {
        rep(i, 1, n) {
            lx[i] = *max_element(g2[i]+1, g2[i]+n+1);
            ly[i] = 0;
            match[i] = -1;
        }
        rep(i, 1, n) {
            for (;;) {
                mem(lcheck, 0);
                mem(rcheck, 0);
                if (dfs(i)) break; else update();
            }
        }
        int ans = 0;
        rep(i, 1, n) if (~match[i]) ans += lx[match[i]] + ly[i];
        return ans;
    }
};
int solve(int n, KM &solver)
{
    solver.init(n);
    rep(i, 1, n) rep(j, 1, n) {
        int x;
        scanf("%d", &x);
        solver.add_edge(i, j, x);
    }
    return solver.calc();
}
int main()
{
    KM solver;
    int n;
    while (~scanf("%d", &n)) printf("%d\n", solve(n, solver));
}
```

## 4.4 树

### 4.4.1 prim

```
int cnt = 1, ans = 0;
priority_queue<edge> pq;
for(int i = 0; i < g[1].size(); i++) pq.push(g[1][i]);
while(!pq.empty()) {
    edge t = pq.top(); pq.pop();
    int v = t.v;
    if(p[v] != 1) {
        cnt++; ans += t.w;
        p[v] = 1;
for(int i = 0; i < g[v].size(); i++) if(p[g[v][i].v] != 1) pq.push(g[v][i]);
    }
    if(cnt == n) break;
}
```

### 4.4.2 曼哈顿最小距离生成树

```
struct Edge {
    int u, v, w;
    bool operator < (const Edge& rhs) const {
        return w < rhs.w;
    }
}edges[8 * MAXN];
struct node {
    int x, y, id;
    bool operator < (const node& rhs) const {
        return x == rhs.x ? y > rhs.y : x > rhs.x;
    }
}nd[MAXN];
int ecnt, n, k, sz;
int sq[MAXN];
int minv[MAXN], pos[MAXN];
int p[MAXN];
int lowbit(int x) {return x & -x;}
void update(int x, int val, int id) {
    while(x) {
        if(val < minv[x]) {
            minv[x] = val; pos[x] = id;
        }
        x -= lowbit(x);
    }
}
int query(int x) {
    int ans = 1 << 30, ret = 0;
    while(x <= sz) {
        if(minv[x] < ans) {
            ans = minv[x]; ret = pos[x];
        }
        x += lowbit(x);
    }
    //cout << ans << endl;
    return ret;
}
int fp(int x) {return p[x] == x ? x : p[x] = fp(p[x]);}
void addEdge(int u, int v, int w) {
    edges[ecnt++] = (Edge) {u, v, w};
}
void solve() {
    for(int dir = 0; dir < 4; dir++) {
        if(dir == 1 || dir == 3)
            for(int i = 1; i <= n; i++) swap(nd[i].x, nd[i].y);
        else if(dir == 2)
            for(int i = 1; i <= n; i++) nd[i].x = -nd[i].x;
        sort(nd + 1, nd + n + 1);
        for(int i = 1; i <= n; i++) sq[i] = nd[i].y - nd[i].x;
        sort(sq + 1, sq + n + 1);
        sz = unique(sq + 1, sq + n + 1) - (sq + 1);
        for(int i = 1; i <= sz; i++) {minv[i] = (1 << 30); pos[i] = 0;}
        for(int i = 1; i <= n; i++) {
            int p = lower_bound(sq + 1, sq + sz + 1, nd[i].y - nd[i].x) - sq;
            int v = query(p);
            //cout << p << ' ' << v << endl;
            if(v) addEdge(nd[i].id, nd[v].id, abs(nd[i].x - nd[v].x) + abs(nd[i].y - nd[v].y));
            update(p, nd[i].x + nd[i].y, i);
        }
    }
    sort(edges, edges + ecnt);
    int cnt = 0;
    for(int i = 1; i <= n; i++) p[i] = i;
    for(int i = 0; i < ecnt; i++) {
        Edge e = edges[i];
        int x = fp(e.u), y = fp(e.v);
        if(x != y) {
            p[x] = y;
            if(++cnt == n - k) {cout << e.w << endl; break;}
            //cout << cnt << endl;
        }
    }
}
```

## 4.5 网络流

### 4.5.1 最大流 Dinic

```
int bfs(int s, int t) {
    memset(vis, false, sizeof(vis));
    queue<int> q;
    q.push(s);
    vis[s] = true; d[s] = 0;
    while(!q.empty()) {
        int u = q.front(); q.pop();
        for(int i = 0; i < g[u].size(); i++) {
            Edge e = edges[g[u][i]];
            if(vis[e.v]) continue;//不能再走访问过的点
            if(e.cap > e.flow) {
                d[e.v] = d[u] + 1;
                vis[e.v] = true;
                q.push(e.v);
            }
        }
    }
    return vis[t];
}
int dfs(int u, int a, int t) {
    if(a == 0 || u == t) return a;
    int flow = 0, f;
    for(int &i = cur[u]; i < g[u].size(); i++) {
        Edge &e = edges[g[u][i]];
        if(d[e.v] == d[u] + 1 && (f = dfs(e.v, min(a, e.cap - e.flow), t)) > 0) {//如果从v走还可以增广
            e.flow += f;
            edges[g[u][i]^1].flow -= f;
            flow += f;
            a -= f;
            if(a == 0) break;
        }
    }
    return flow;
}
int dinic(int s, int t) {
    int flow = 0;
    while(bfs(s, t)) {
        memset(cur, 0, sizeof(cur));
        flow += dfs(s, INF, t);
    }
    return flow;
}
```

### 4.5.2 最大流 ISAP

```
struct ISAP {
    struct Edge {
      int from, to, cap, flow;
      bool operator < (const Edge& b) {
        return from < b.from || (from == b.from && to < b.to);
      }
    };


      int  n, m, s, t;
      vector<Edge> edges;
      vector<int> g[MAXN];   // 邻接表，g[i][j]表示结点i的第j条边在e数组中的序号
      bool vis[MAXN];        // BFS使用
      int d[MAXN];           // 从起点到i的距离
      int cur[MAXN];        // 当前弧指针
      int p[MAXN];          // 可增广路上的上一条弧
      int num[MAXN];        // 距离标号计数

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

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

  void init(int nn) {
    n = nn;
      mem(g, 0);
    edges.clear();
  }

  void ClearFlow() {
    for(int i = 0; i < edges.size(); i++) edges[i].flow = 0;
  }

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

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

  vector<int> Mincut() { // call this after maxflow
    BFS();
    vector<int> ans;
    for(int i = 0; i < edges.size(); i++) {
      Edge& e = edges[i];
      if(!vis[e.from] && vis[e.to] && e.cap > 0) ans.push_back(i);
    }
    return ans;
  }

  void Reduce() {
    for(int i = 0; i < edges.size(); i++) edges[i].cap -= edges[i].flow;
  }
```

```

      void print() {
        printf("Graph:\n");
        for(int i = 0; i < edges.size(); i++)
          printf("%d->%d,␣%d,␣%d\n", edges[i].from, edges[i].to , edges[i].cap, edges[i].flow);
      }
    };
```

### 4.5.3 费用流

### 4.5.4 无源无汇有容量下界网络的可行流

```
#include <bits/stdc++.h>
#define log(x) cout << #x << "␣=␣" << (x) << endl
#define mem(x, y) memset((x), (y), sizeof((x)))
#define rep(i, l, r) for (int (i) = (l); (i) <= (r); (i)++)
using namespace std;
typedef long long ll;
/*---金牌---*/
const int N = 200 + 9;
const int INF = 0x3f3f3f3f;
const int MAXN = 10*N;
int low[MAXN*10];
int n, m;
struct Dinic
{
    struct Edge
    {
      int from, to, cap, flow;
    };
    vector<Edge> edges;
    vector<int> g[MAXN];
    void init() {
        mem(g, 0);
        edges.clear();
    }
    void add_edge(int from, int to, int cap) {
      edges.push_back((Edge){from, to, cap, 0});
      edges.push_back((Edge){to, from, 0, 0});
      int m = edges.size();
      g[from].push_back(m - 2);
```

```
  g[to].push_back(m - 1);
}
int s, t;
bool vis[MAXN];
int d[MAXN], cur[MAXN];
bool BFS()
{
  memset(vis, 0, sizeof(vis));
  queue<int> q;
  q.push(s);
  d[s] = 0;
  vis[s] = true;
  while (!q.empty()) {
    int x = q.front(); q.pop();
    for (int i = 0; i < g[x].size(); i++) {
      Edge& e = edges[g[x][i]];
      if (!vis[e.to] && e.cap > e.flow) {
        vis[e.to] = true;
        d[e.to] = d[x] + 1;
        q.push(e.to);
      }
    }
  }
  return vis[t];
}
int DFS(int x, int a)
{
  if (x == t || a == 0) return a;
  int flow = 0;
  int f;
  for (int& i = cur[x]; i < g[x].size(); i++) {
    Edge& e = edges[g[x][i]];
    if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a,
       e.cap - e.flow))) > 0) {
      e.flow += f;
      edges[g[x][i]^1].flow -= f;
      flow += f;
      a -= f;
      if (a == 0) break;
    }
  }
  return flow;
}
int max_flow(int ss, int tt)
{
    s = ss, t = tt;
```

```
      int flow = 0;
      while (BFS()) {
        memset(cur, 0, sizeof(cur));
        flow += DFS(s, INF);
      }
      return flow;
    }
    void solve(int source, int sink)
    {
        max_flow(source, sink);
        int sz = g[source].size() - 1;
        bool flag = true;
        rep(i, 0, sz) {
            if (edges[g[source][i]].flow < edges[g[source][i]].cap) {
                puts("NO");
                flag = false;
            }
        }
        if (flag) {
            puts("YES");
            rep(i, 1, m) {
                //log(low[i]);
                printf("%d\n", edges[i*2+1].cap-edges[i*2+1].flow + low[i]);
            }
        }
    }
};
void solve(Dinic &solver)
{
    solver.init();
    scanf("%d", &m);
    int source = n + 1, sink = n + 2;
    // sink -> source cap : inf
    solver.add_edge(sink, source, INF);
    int in[N];
    mem(in, 0);
    mem(low, 0);
    rep(i, 1, m) {
        int u, v, b, c;
        scanf("%d%d%d%d", &u, &v, &b, &c);
        low[i] = b; // 流量下界
        in[u] -= b;
        in[v] += b;
```

```
            solver.add_edge(u, v, c - b);
    }
    rep(i, 1, n) {
        if (in[i] < 0) {
            solver.add_edge(i, sink, -in[i]);
        } else if (in[i] > 0) {
            solver.add_edge(source, i, in[i]);
        }
    }
    solver.solve(source, sink);
}
int main()
{
    Dinic solver;
    while (~scanf("%d", &n)) solve(solver);
}
```

## 4.6 其他

### 4.6.1 层次遍历

每次记录队列当前的 sz 然后在一次循环中只出队 sz 次

### 4.6.2 图解序列

图解序列：一系列非负整数可以构成一个简单图的度序列
Havel 定理：对于 $n>1$, 长度为 n 的整数序列是图解序列当且仅当 d' 是图解序列，d' 是的删除 d 中最大元素 $\Delta$ 并将紧跟的 $\Delta$ 个元素减 1 后序列

# Chapter 5

# 计算几何

## 5.1 判断凸包

```
#include <cstdio>
#include <iostream>
using namespace std;
const int maxn = 1e8;
struct P
{
  int x, y;
};
int ccw(P a, P b, P c)
{
  return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y -
      a.y);
}
P a[maxn];
int main()
{
  for (;;)
  {
    int n;
    cin >> n;
    if (n == 0) break;
    for (int i = 0; i < n; i++)
      cin >> a[i].x >> a[i].y;
    bool flag = true;
    a[n++] = a[0];
    a[n++] = a[1];
    for (int i = 0; i < n - 3; i++)
```

```
      if (ccw(a[i], a[i + 1], a[i + 2]) * ccw(a[i + 1], a[i + 2], a[i + 3]) < 0)
      {
        flag = false;
        break;
      }
    if (flag) puts("convex"); else puts("concave");
  }
}
```

## 5.2 判断线段是否相交

```
struct P
{
  double x, y;
};
struct Segment
{
  P p, q;
};
double ccw(P a, P b, P c)
{
   return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y - a.y);
}
bool intersects(Segment a, Segment b)
{
   if (ccw(a.p, a.q, b.p) * ccw(a.p, a.q, b.q) > 0) return false;
   if (ccw(b.p, b.q, a.p) * ccw(b.p, b.q, a.q) > 0) return false;
   return true;
}
```

## 5.3 求对称点求交点

```
using namespace std;
const double eps = 1e-10;
double add (double a, double b)
{
  if (abs(a + b) < eps * (abs(a) + abs(b))) return 0;
  return a + b;
}
```

```
struct P
{
  double x, y;
  P(){}
  P(double x, double y):x(x),y(y){}
  P operator + (P p)
  {
    return P(add(x, p.x), add(y, p.y));
  }
  P operator - (P p)
  {
    return P(add(x, -p.x), add(y, -p.y));
  }
  P operator * (double d)
  {
    return P(x * d, y * d);
  }
};
double det(P p1, P p2)
{
  return add(p1.x * p2.y, -p1.y * p2.x);
}
bool g_equal(double a, double b)
{
  if (a > b - eps && a < b + eps) return true;
  return false;
}
//求p关于p1, p2所成直线的对称点
P symmetric_point(P p, P p1, P p2)
{
  //直线与x轴垂直
        if (g_equal(p1.x, p2.x)) return P(2 * p1.x - p.x,
            p.y);
  double k = (p1.y - p2.y ) / (p1.x - p2.x);
  if (g_equal(k, 0)) return P(p.x, 2 * p1.y - p.y);
  double x = (2*k*k*p1.x + 2*k*p.y - 2*k*p1.y - k*k*p.x +
      p.x) / (1 + k * k);
  double y = p.y - (x - p.x) / k;
  return P(x, y);
}
P intersection(P p1, P p2, P q1, P q2)
{
  return p1 + (p2 - p1) * (det(q2 - q1, q1 - p1) / det(q2
      - q1, p2 - p1));
}
```

## 5.4 终极模板

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

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

typedef Point Vector;

Vector operator + (Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
Vector operator - (Vector A, Vector B) { return Vector(A.x - B.x, A.y - B.y); }
Vector operator * (Vector A, double p) { return Vector(A.x*p, A.x*p); }
Vector operator / (Vector A, double p) { return Vector(A.x/p, A.x/p); }

bool operator < (const Point& a, const Point b) {
    return a.x < b.x || (a.x == b.x && a.y < b.y);
}

const double EPS = 1e-10;

int dcmp(double x) {
    if(fabs(x) < EPS) return 0;
    else return x < 0 ? -1 : 1;
}

bool operator == (const Point& a, const Point& b) {
    return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y);
}

//向量a的极角
double Angle(const Vector& v) {
    return atan2(v.y, v.x);\share\CodeBlocks\templates\wizard\console\cpp
}

//向量点积
double Dot(Vector A, Vector B) { return A.x*B.x + A.y*B.y; }
```

```

//向量长度\share\CodeBlocks\templates\wizard\console\cpp
double Length(Vector A) { return sqrt(Dot(A, A)); }

//向量夹角
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }

//向量叉积
double Cross(Vector A, Vector B) { return A.x*B.y - A.y*B.x; }

//三角形有向面积的二倍
double Area2(Point A, Point B, Point C) { return Cross(B-A, C-A); }

//向量逆时针旋转rad度(弧度)
Vector Rotate(Vector A, double rad) {
    return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)+A.y*cos(rad));
}

//计算向量A的单位法向量。左转90°，把长度归一。调用前确保A不是零向量。
Vector Normal(Vector A) {
    double L = Length(A);
    return Vector(-A.y/L, A.x/L);
}

/*************************************************************************
使用复数类实现点及向量的简单操作

#include <complex>
typedef complex<double> Point;
typedef Point Vector;

double Dot(Vector A, Vector B) { return real(conj(A)*B)}
double Cross(Vector A, Vector B) { return imag(conj(A)*B);}
Vector Rotate(Vector A, double rad) { return A*exp(Point(0, rad)); }

```

```
************************************************************************ */

/* ***************************************************************************
* 用直线上的一点p0和方向向量v表示一条指向。直线上的所有点P满足P = P0+t*v;
* 如果知道直线上的两个点则方向向量为B-A, 所以参数方程为A+(B-A)*t;
* 当t 无限制时， 该参数方程表示直线。
* 当t > 0时， 该参数方程表示射线。
* 当 0 < t < 1时， 该参数方程表示线段。
**************************************************************************** */

//直线交点,须确保两直线有唯一交点。
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w) {
    Vector u = P - Q;
    double t = Cross(w, u)/Cross(v, w);
    return P+v*t;
}

//点到直线距离
double DistanceToLine(Point P, Point A, Point B) {
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2) / Length(v1)); //不取绝对值，得到的是有向距离
}

//点到线段的距离
double DistanceToSegmentS(Point P, Point A, Point B) {
    if(A == B) return Length(P-A);
    Vector v1 = B-A, v2 = P-A, v3 = P-B;
    if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
    else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
    else return fabs(Cross(v1, v2)) / Length(v1);
}

//点在直线上的投影
Point GetLineProjection(Point P, Point A, Point B) {
    Vector v = B - A;
    return A+v*(Dot(v, P-A)/Dot(v, v));
}

```

```
//线段相交判定，交点不在一条线段的端点
bool SegmentProperIntersection(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, a1-b1), c4 = Cross(b2-b1, a2-b1);
    return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
}

//判断点是否在点段上，不包含端点
bool OnSegment(Point P, Point a1, Point a2) {
    return dcmp(Cross(a1-P, a2-P) == 0 && dcmp((Dot(a1-P, a2-P)) < 0));
}

//计算凸多边形面积
double ConvexPolygonArea(Point *p, int n) {
    double area = 0;
    for(int i = 1; i < n-1; i++)
        area += Cross(p[i] - p[0], p[i+1] - p[0]);
    return area/2;
}

//计算多边形的有向面积
double PolygonArea(Point *p, int n) {
    double area = 0;
    for(int i = 1; i < n-1; i++)
        area += Cross(p[i] - p[0], p[i+1] - p[0]);
    return area/2;
}

/* **********************************************************************
* Morley定理：三角形每个内角的三等分线，相交成的三角形是等边三角形。
* 欧拉定理：设平面图的定点数，边数和面数分别为V,E,F。则V+F-E = 2;
***********************************************************************/

struct Circle {
    Point c;
    double r;

    Circle(Point c, double r) : c(c), r(r) {}
    //通过圆心角确定圆上坐标
    Point point(double a) {
        return Point(c.x + cos(a)*r, c.y + sin(a)*r);
    }
};

struct Line {
    Point p;
    Vector v;
    double ang;
    Line() {}
    Line(Point p, Vector v) : p(p), v(v) {}
    bool operator < (const Line& L) const {
        return ang < L.ang;
    }
};

//直线和圆的交点，返回交点个数，结果存在sol中。
//该代码没有清空sol。
int getLineCircleIntersecion(Line L, Circle C, double& t1, double& t2, vector<Point>& sol) {
    double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
    double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
    double delta = f*f - 4*e*g;
    if(dcmp(delta) < 0) return 0; //相离
    if(dcmp(delta) == 0) {        //相切
        t1 = t2 = -f / (2*e);
        sol.push_back(C.point(t1));
        return 1;
    }
    //相交
    t1 = (-f - sqrt(delta)) / (2*e); sol.push_back(C.point(t1));
    t2 = (-f + sqrt(delta)) / (2*e); sol.push_back(C.point(t2));
    return 2;
}

//两圆相交
```

```
int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol) {
    double d = Length(C1.c - C2.c);
    if(dcmp(d) == 0) {
        if(dcmp(C1.r - C2.r == 0)) return -1;    //两圆完全重合
        return 0;                                //同心圆，半径不一样
    }
    if(dcmp(C1.r + C2.r - d) < 0) return 0;
    if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;

    double a = Angle(C2.c - C1.c);               //向量C1C2的极角
    double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*C1.r*d));
    //C1C2到C1P1的角
    Point p1 = C1.point(a-da), p2 = C1.point(a+da);
    sol.push_back(p1);
    if(p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
}

const double PI = acos(-1);
//过定点做圆的切线
//过点p做圆C的切线，返回切线个数。v[i]表示第i条切线
int getTangents(Point p, Circle C, Vector* v) {
    Vector u = C.c - p;
    double dist = Length(u);
    if(dist < C.r) return 0;
    else if(dcmp(dist - C.r) == 0) {
        v[0] = Rotate(u, PI/2);
        return 1;
    } else {
        double ang = asin(C.r / dist);
        v[0] = Rotate(u, -ang);
        v[1] = Rotate(u, +ang);
        return 2;
    }
}

//两圆的公切线
//返回切线的个数，-1表示有无数条公切线。
//a[i], b[i] 表示第i条切线在圆A，圆B上的切点
```

```
int getTangents(Circle A, Circle B, Point *a, Point *b) {
    int cnt = 0;
    if(A.r < B.r) {
        swap(A, B); swap(a, b);
    }
    int d2 = (A.c.x - B.c.x)*(A.c.x - B.c.x) + (A.c.y - B.c.y)*(A.c.y - B.c.y);
    int rdiff = A.r - B.r;
    int rsum = A.r + B.r;
    if(d2 < rdiff*rdiff) return 0;   //内含
    double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
    if(d2 == 0 && A.r == B.r) return -1;   //无限多条切线
    if(d2 == rdiff*rdiff) {          //内切一条切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(base);
        cnt++;
        return 1;
    }
    //有外共切线
    double ang = acos((A.r-B.r) / sqrt(d2));
    a[cnt] = A.point(base+ang); b[cnt] = B.point(base+ang); cnt++;
    a[cnt] = A.point(base-ang); b[cnt] = B.point(base-ang); cnt++;
    if(d2 == rsum*rsum) {  //一条公切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(PI+base);
        cnt++;
    } else if(d2 > rsum*rsum) {   //两条公切线
        double ang = acos((A.r + B.r) / sqrt(d2));
        a[cnt] = A.point(base+ang); b[cnt] = B.point(PI+base+ang); cnt++;
        a[cnt] = A.point(base-ang); b[cnt] = B.point(PI+base-ang); cnt++;
    }
    return cnt;
}

typedef vector<Point> Polygon;

//点在多边形内的判定
int isPointInPolygon(Point p, Polygon poly) {
    int wn = 0;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1; //在边界上
        int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[i]));
        int d1 = dcmp(poly[i].y - p.y);
        int d2 = dcmp(poly[(i+1)%n].y - p.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn++;
    }
    if(wn != 0) return 1;       //内部
    return 0;                   //外部
}

//凸包
/* **************************************************************
* 输入点数组p， 个数为p， 输出点数组ch。 返回凸包顶点数
* 不希望凸包的边上有输入点，把两个<= 改成 <
* 高精度要求时建议用dcmp比较
* 输入点不能有重复点。函数执行完以后输入点的顺序被破坏
*************************************************************** */
int ConvexHull(Point *p, int n, Point* ch) {
    sort(p, p+n);      //先比较x坐标，再比较y坐标
    int m = 0;
    for(int i = 0; i < n; i++) {
        while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    int k = m;
    for(int i = n-2; i >= 0; i++) {
        while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m-2]) <= 0) m--;
        ch[m++] = p[i];
    }
    if(n > 1) m--;
    return m;
}

//用有向直线A->B切割多边形poly， 返回“左侧”。 如果退化，可能会返回一个单点或者线段
//复杂度O(n2);
Polygon CutPolygon(Polygon poly, Point A, Point B) {
    Polygon newpoly;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        Point C = poly[i];
        Point D = poly[(i+1)%n];
        if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(C);
        if(dcmp(Cross(B-A, C-D)) != 0) {
            Point ip = GetLineIntersection(A, B-A, C, D-C);
            if(OnSegment(ip, C, D)) newpoly.push_back(ip);
        }
    }
    return newpoly;
}

//半平面交

//点p再有向直线L的左边。（线上不算）
bool Onleft(Line L, Point p) {
    return Cross(L.v, p-L.p) > 0;
}

//两直线交点，假定交点唯一存在
Point GetIntersection(Line a, Line b) {
    Vector u = a.p - b.p;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.p+a.v*t;
}

int HalfplaneIntersection(Line* L, int n, Point* poly) {
    sort(L, L+n);               //按极角排序

    int first, last;            //双端队列的第一个元素和最后一个元素
    Point *p = new Point[n];    //p[i]为q[i]和q[i+1]的交点
    Line *q = new Line[n];      //双端队列
    q[first = last = 0] = L[0]; //队列初始化为只有一个半平面L[0]
    for(int i = 0; i < n; i++) {
        while(first < last && !Onleft(L[i], p[last-1])) last--;
        while(first < last && !Onleft(L[i], p[first]))
           first++;
        q[++last] = L[i];
        if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
            last--;
            if(Onleft(q[last], L[i].p)) q[last] = L[i];
        }
        if(first < last) p[last-1] = GetIntersection(q[
           last-1], q[last]);
    }
    while(first < last && !Onleft(q[first], p[last-1]))
       last--;
    //删除无用平面
    if(last-first <= 1) return 0;   //空集
    p[last] = GetIntersection(q[last], q[first]);

    //从deque复制到输出中
    int m = 0;
    for(int i = first; i <= last; i++) poly[m++] = p[i];
    return m;
}
int main() {

    return 0;
}
```

## 5.5 K 次圆

# Chapter 6

# 数据结构

## 6.1 手写堆

```
struct Node
{
    int index;
    int tag;
    int l;
    bool operator<(Node ano) const
    {
        return l!=ano.l?l<ano.l:index>ano.index;
    }
};
Node heap[51111];
int heapsz;
int a[2<<21]; // 编号为i的在heap的位置
void swim(int p)
{
        while (p > 1 && heap[p / 2] < heap[p])
        {
            swap(a[heap[p].tag],a[heap[p/2].tag]);
            swap(heap[p],heap[p/2]);
            p/=2;
        }
}
void sink(int p)
{
    while(p * 2 <= heapsz)
    {
        p *= 2;
        if (p+1<=heapsz) p|=heap[p]<heap[p+1];
        swap(a[heap[p].tag],a[heap[p/2].tag]);
        swap(heap[p],heap[p/2]);
    }
}
```

## 6.2 左偏树

```
#include <bits/stdc++.h>
#define rep(i, l, r) for (int (i) = (l); (i) <= (r); (i)
    ++)
using namespace std;
int n,m,ans;
struct Data{int fa,x,dis,l,r;};
const int MAXN = 100001;
Data q[MAXN];
int find(int x)
{
    return x==q[x].fa?x:q[x].fa=find(q[x].fa);
}
int merge(int x,int y)
{
    if(!x) return y;
    if(!y) return x;
    if(q[x].x<q[y].x) swap(x,y);
    q[x].r=merge(q[x].r,y);
    q[q[x].r].fa=x;
    if(q[q[x].l].dis<q[q[x].r].dis) swap(q[x].l,q[x].r);
    if(!q[x].r) q[x].dis=0;
    else q[x].dis=q[q[x].r].dis+1;
    return x;
}
int pop(int x)
{
    int l=q[x].l,r=q[x].r;
    q[l].fa=l,q[r].fa=r;
    q[x].l=q[x].r=q[x].dis=0;
    return merge(l,r);
}
void slove()
{
    int x,y,u,v,l1,l2;
    rep(i, 1, n) {
        scanf("%d", &q[i].x);
        q[i].fa=i;
        q[i].l=q[i].r=q[i].dis=0;
     }
    scanf("%d", &m);
    rep(i, 1, m) {
        scanf("%d%d", &x, &y);
        l1=find(x), l2=find(y);
        if(l1 == l2){printf("-1\n");continue;}
        q[l1].x/=2;u=pop(l1);u=merge(u,l1);
        q[l2].x/=2;v=pop(l2);v=merge(v,l2);
        ans=merge(u,v);
        printf("%d\n",q[ans].x);
    }
}
int main()
{
    while(~scanf("%d",&n)) slove();
}
```

## 6.3 两优先队列模拟堆

```

最简分数实数逼近
a1 = 100000, a2 = 1, ans = 100000;
ll xl = 0, yl = 1, xr = 1, yr = 0, xm, ym;
while(yl + yr <= 100000) {
    xm = xl + xr;
    ym = yl + yr;
    if(check(xm, ym, a1, a2, kk)) {a1 = xm, a2 = ym;}
    if(get(xm, ym, kk)) {xr = xm, yr = ym;}
    else {xl = xm, yl = ym;}
 }
struct heap {
    priority_queue<int> p, q;
    void init() {
        while(!p.empty()) p.pop();
        while(!q.empty()) q.pop();
    }
    void add(int x) {p.push(x);}
    void del(int x) {q.push(x);}
    int top() {
        while(1) {
            if(p.empty()) return -INF;
            else if(!q.empty() && p.top() == q.top()) {p.pop(); q.pop();}
            else return p.top();
        }
    }
    int toptwo() {
        int a = top(); del(a);
        int b = top(); add(a);
        if(b == -INF) return a == -INF ? a : 0;
        else return max(a + b, 0);
    }
}
```

## 6.4 线段树

```
#include <bits/stdc++.h>
using namespace std;
const int MAXM = 1E6 + 9;
const int MAXR = 20 + 9;
const int INF = INT_MAX / 10;
struct Node
{
    int setv, addv, sumv, minv, maxv;
    Node()
    {
        setv = -1, addv = 0, sumv = 0, minv = 0, maxv = 0;
    }
};
struct Tree
{
    Node tree[4 * MAXM];
    int mid(int l, int r)
    {
        return l + (r-l) / 2;
    }
    void pushdown(int o)
    {
        int lc(o*2), rc(o*2+1);
        if (tree[o].setv >= 0) {
            tree[lc].setv = tree[rc].setv = tree[o].setv;
            tree[lc].addv = tree[rc].addv = 0;
            tree[o].setv = -1;
        }
        if (tree[o].addv) {
            tree[lc].addv += tree[o].addv;
            tree[rc].addv += tree[o].addv;
            tree[o].addv = 0;
        }
    }
    void maintain(int o, int l, int r)
    {
        int lc(o*2), rc(o*2+1);
        if (r > l) {
            tree[o].sumv = tree[lc].sumv + tree[rc].sumv;
            tree[o].minv = min(tree[lc].minv,tree[rc].minv);
            tree[o].maxv = max(tree[lc].maxv,tree[rc].maxv);
        }
        if (tree[o].setv >= 0) {
            tree[o].minv = tree[o].maxv = tree[o].setv;
            tree[o].sumv = (r-l+1)*tree[o].setv;
        }
        if (tree[o].addv) {
            tree[o].minv += tree[o].addv;
            tree[o].maxv += tree[o].addv;
            tree[o].sumv += (r-l+1) * tree[o].addv;
        }
    }
    void update(int o, int l, int r, int op, int ql, int qr, int v)
    {
        int lc(o*2), rc(o*2+1);
        if (ql <= l && qr >= r) {
            if (op == 1) tree[o].addv += v;
            else {
                tree[o].setv = v;
                tree[o].addv = 0;
            }
        } else {
            pushdown(o);
            int m = mid(l, r);
            if (ql <= m) update(lc, l, m, op, ql, qr, v); else maintain(lc, l, m);
            if (qr > m) update(rc, m+1, r, op, ql, qr, v); else maintain(rc, m+1, r);
        }
        maintain(o, l, r);
    }
    Node query(int o, int l, int r, int ql, int qr)
    {
        Node res;
        int lc(o*2), rc(o*2+1), m(mid(l, r));
        maintain(o, l, r);
        if (ql <= l && qr >= r) {
            res.sumv = tree[o].sumv;
            res.minv = tree[o].minv;
            res.maxv = tree[o].maxv;
        } else {
            pushdown(o);
            Node lres;
            lres.minv = INF, lres.maxv = -INF;
            Node rres(lres);
            if (ql <= m) lres = query(lc, l, m, ql, qr); else maintain(lc, l, m);
            if (qr > m) rres = query(rc, m+1, r, ql, qr); else maintain(rc, m+1, r);
            res.sumv = lres.sumv + rres.sumv;
            res.minv = min(lres.minv, rres.minv);
            res.maxv = max(lres.maxv, rres.maxv);
        }
        return res;
    }
};
Tree tree[MAXR];
void solve(int r, int c, int m)
{
    for (int i(1); i <= r; i++) tree[i].tree[1].setv = 0;
    while (m--) {
        int op, x1, y1, x2, y2;
        scanf("%d%d%d%d%d", &op, &x1, &y1, &x2, &y2);
        if (op < 3) {
            int v;
            scanf("%d", &v);
            for (int i(x1); i <= x2; i++) tree[i].update(1, 1, c, op, y1, y2, v);
        } else {
            Node ans;
            ans.sumv = 0, ans.minv = INF, ans.maxv = -INF;
            for (int i(x1); i <= x2; i++) {
                Node res(tree[i].query(1, 1, c, y1, y2));
                ans.sumv += res.sumv;
                ans.minv = min(ans.minv, res.minv);
                ans.maxv = max(ans.maxv, res.maxv);
            }
            printf("%d %d %d\n", ans.sumv, ans.minv, ans.maxv);
        }
    }
}
int main()
{
    //freopen("in", "r", stdin);
    int r, c, m;
    while (scanf("%d%d%d", &r, &c, &m) != EOF) {
        solve(r, c, m);
    }
}
```

## 6.5 二维线段树

```
#include <bits/stdc++.h>
using namespace std;
const int MAXN = 800 * 4 + 10;
int mx[MAXN][MAXN], mn[MAXN][MAXN], g[810][810];
int n;
void buildy(int xo, int o, int l, int r, int x) {
    if(l == r) {
        if(x != -1) mx[xo][o] = mn[xo][o] = g[x][l];
        else {
            int xlch = xo * 2, xrch = xlch + 1;
            mx[xo][o] = max(mx[xlch][o], mx[xrch][o]);
            mn[xo][o] = min(mn[xlch][o], mn[xrch][o]);
        }
        return;
    }
    int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
    buildy(xo, lch, l, mid, x);
    buildy(xo, rch, mid + 1, r, x);
    mx[xo][o] = max(mx[xo][lch], mx[xo][rch]);
    mn[xo][o] = min(mn[xo][lch], mn[xo][rch]);
}
void buildx(int o, int l, int r) {
    if(l == r) {
        buildy(o, 1, 1, n, l);
        return;
    }
    int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
    buildx(lch, l, mid);
    buildx(rch, mid + 1, r);
    buildy(o, 1, 1, n, -1);
}
void updatey(int xo, int o, int l, int r, int y, int v) {
    if(l == r) {
        if(v != -1) mn[xo][o] = mx[xo][o] = v;
        else {
            int xlch = xo * 2, xrch = xlch + 1;
            mx[xo][o] = max(mx[xlch][o], mx[xrch][o]);
            mn[xo][o] = min(mn[xlch][o], mn[xrch][o]);
        }
        return;
    }
    int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
    if(y <= mid) updatey(xo, lch, l, mid, y, v);
    else updatey(xo, rch, mid +1, r, y, v);
    mx[xo][o] = max(mx[xo][lch], mx[xo][rch]);
    mn[xo][o] = min(mn[xo][lch], mn[xo][rch]);
}
void updatex(int o, int l, int r, int x, int y, int v) {
    if(l == r) {
        updatey(o, 1, 1, n, y, v);
        return;
    }
    int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
    if(mid >= x) updatex(lch, l, mid, x, y, v);
    else updatex(rch, mid + 1, r, x, y, v);
    updatey(o, 1, 1, n, y, -1);
}
int minv, maxv;
void queryy(int xo, int o, int l, int r, int yl, int yr) {
    if(yl <= l && r <= yr) {
        minv = min(minv, mn[xo][o]);
        maxv = max(maxv, mx[xo][o]);
        return;
    }
    int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
    if(mid >= yl) queryy(xo, lch, l, mid, yl, yr);
    if(mid < yr) queryy(xo, rch, mid + 1, r, yl, yr);
}
void queryx(int o, int l, int r, int xl, int xr, int yl, int yr) {
    if(xl <= l && r <= xr) {
        queryy(o, 1, 1, n, yl, yr);
        return;
    }
    int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
    if(mid >= xl) queryx(lch, l, mid, xl, xr, yl, yr);
    if(mid < xr) queryx(rch, mid + 1, r, xl, xr, yl, yr);
}
int main() {
    int T; scanf("%d", &T);
    int cas = 1;
    while(T--) {
        scanf("%d", &n);
        for(int i = 1; i <= n; i++)
            for(int j = 1; j <= n; j++)
                scanf("%d", &g[i][j]);
        buildx(1, 1, n);
        int q; scanf("%d", &q);
        printf("Case␣#%d:\n", cas++);
        while(q--) {
            int x, y, len; scanf("%d%d%d", &x, &y, &len);
            int xl = max(1, x - len / 2), xr = min(n, x + len / 2), yl = max(1, y - len / 2), yr = min(n, y + len / 2);
            minv = 1 << 30, maxv = -minv;
            queryx(1, 1, n, xl, xr, yl, yr);
            int ans = (maxv + minv) / 2;
            updatex(1, 1, n, x, y, ans);
            printf("%d\n", ans);
        }
    }
}
```

## 6.6 Treap

```
struct node {
        node* ch[2];
        int w, sum, v, k;
        void maintain() {sum = w + ch[0]->sum + ch[1]->sum;}
        int cmp(int vv) {return vv == v ? -1 : vv > v;}
        bool operator < (const node& rhs) {return k < rhs.k;}
    }nd[MAXN *10];
    node* null, *rt;
    int ncnt;
    void rot(node*& o, int d) {
```

```
node* k = o->ch[d^1]; o->ch[d^1] = k->ch[d]; k->ch[d] = o;
//一定先维护o，因为o是k的子节点
o->maintain(); k->maintain();
    o = k;
}
void ins(node*& o, int x) {
    if(o == null) {
        o = &nd[ncnt++];
        *o = *null;
        o->v = x; o->k = rand();
        o->w = 1; o->maintain();
    }
    else {
        int d = o->cmp(x);
        if(d == -1) {o->w++; o->sum++;}
        else {
            ins(o->ch[d], x);
            o->maintain();
            if(o < o->ch[d]) rot(o, d^1);
        }
    }
}
void del(node*& o, int x) {
    if(o == null) return;
    int d = o->cmp(x);
    if(d == -1) {
        if(o->w > 1) {o->w--; o->sum--;}
        else {
            if(o->ch[0] != null && o->ch[1] != null) {
                int dd = o->ch[0] > o->ch[1];
                rot(o, dd);
                del(o->ch[dd], x);
            }
            else o = o->ch[0] == null ? o->ch[1] : o->ch[0];
        }
    }
    else del(o->ch[d], x);
    o->maintain();
}
int getrk(node* o, int x) {
    if(o == null) return 0;
    int d = o->cmp(x);
    if(d == -1) return o->ch[0]->sum + 1;
```

```
        else {
            if(d == 1) return o->ch[0]->sum + o->w + getrk(o->ch[1], x);
            return getrk(o->ch[0], x);
        }
    }
    int getkth(node* o, int x) {
        if(o == null) return 0;
        int ls = o->ch[0]->sum, cs = o->w;
        if(x <= ls) return getkth(o->ch[0], x);
        else if(x > ls && x <= ls + cs) return o->v;
        else return getkth(o->ch[1], x - ls - cs);
    }
    int ans;
    //找第一个小于x的数
    void getpre(node* o, int x) {
        if(o == null) return;
        if(x > o->v) {
            ans = o->v;
            getpre(o->ch[1], x);
        }
        else getpre(o->ch[0], x);
    }
    //第一个大于x的数
    void getnxt(node* o, int x) {
        if(o == null) return;
        if(x < o->v) {
            ans = o->v;
            getnxt(o->ch[0], x);
        }
        else getnxt(o->ch[1], x);
    }
```

## 6.7 splay

```
int ch[MAXN][2], fa[MAXN], val[MAXN], sz[MAXN], cnt[MAXN];
int rt, ncnt;
void update(int r) {sz[r] = sz[ch[r][0]] + sz[ch[r][1]] + cnt[r];}
void rot(int x) {
    int y = fa[x], z = fa[y], d = ch[y][1] == x;
    ch[y][d] = ch[x][d^1], fa[ch[y][d]] = y;
    ch[x][d^1] = y; fa[y] = x;
    fa[x] = z;
    if(z) ch[z][ch[z][1] == y] = x;
    update(y);
}
void splay(int r, int tp) {
    for(int y, z; (y = fa[r]) != tp; rot(r)) {
        z = fa[y];
        if(z == tp) continue;
        if((ch[z][0] == y) == (ch[y][0] == r)) rot(y);
        else rot(r);
    }
    if(!tp) rt = r;
    update(r);
}
void ins(int r, int x) {
    int y = 0;
    while(r && val[r] != x) {y = r; r = ch[r][val[r] < x];}
    if(r) ++cnt[r];
    else {
        r = ++ncnt;
        sz[r] = cnt[r] = 1;
        val[r] = x; fa[r] = y;
        ch[r][0] = ch[r][1] = 0;
        if(y) ch[y][val[y] < x] = r;
    }
    splay(r, 0);
}
void get(int v) {
    int x = rt; if(!x) return;
    while(ch[x][val[x] < v] && val[x] != v) x = ch[x][val[x] < v];
    splay(x, 0);
}
int getrk(int v) {
    get(v);
    return sz[ch[rt][0]];
}
int getkth(int x) {
    int y = rt, p;
    if(x > sz[rt]) return 0;
    while(1) {
        p = ch[y][0];
        if(sz[p] + cnt[y] < x) {
            x -= sz[p] + cnt[y];
            y = ch[y][1];
        }
        else if(sz[p] >= x) y = p;
        else return val[y];
    }
}
int nxt(int x, bool op) {
    get(x);
    if((val[rt] > x && op) || (val[rt] < x && !op)) return rt;
    int p = ch[rt][op];
    while(ch[p][op^1]) p = ch[p][op^1];
    return p;
}
void del(int v) {
    int p = nxt(v, 0), s = nxt(v, 1);
    splay(p, 0);
    splay(s, p);
    p = ch[s][0];
    if(cnt[p] > 1) -- cnt[p], splay(p, 0);
    else ch[s][0] = 0;
}
```

## 6.8 倍增 LCA

```
void initp() {
    for(int j = 1; (1 << j) <= n; j++)
        for(int i = 1; i <= n; i++) if(p[i][j - 1])//一定要有这个if
            p[i][j] = p[p[i][j - 1]][j - 1];
}
int LCA(int u, int v) {
    if(d[u] < d[v]) swap(u, v);
int lim;
//确定最大的2^lim不超过d[u]
    for(lim = 0; (1 << lim) <= d[u]; lim++); lim--;
int ret = 0;
//把u上升到v相同的高度
    for(int i = lim; i >= 0; i--) if(d[u] - (1 << i) >= d[v]) u = p[u][i];
    if(u == v) return u; //一定要有这个判断
    for(int i = lim; i >= 0; i--) if(p[u][i] != p[v][i]) {u = p[u][i]; v = p[v][i];}
    return p[u][0];
}
```

## 6.9 主席树

```
void build(node* &now, node* &pre, int l, int r, int x) {
    now = &tn[ncnt++];
    *now = *null;
    int mid = (l + r) / 2;
    if(l == r) {
        *now = *pre;
        now->val++;
        return;
    }
    if(x <= sq[mid]) {
        build(now->ch[0], pre->ch[0], l, mid, x);
        now->ch[1] = pre->ch[1];
        now->maintain();
    }
    else {
        build(now->ch[1], pre->ch[1], mid + 1, r, x);
        now->ch[0] = pre->ch[0];
        now->maintain();
    }
}
```

## 6.10 树剖

```
void dfs1(int u, int fa, int dep) {
    sz[u] = 1; d[u] = dep; ch[u] = 0; p[u] = fa;
    for(int i = head[u]; i != -1; i = ed[i].next) {
        int v = ed[i].v;
        if(v == fa) continue;
        dfs1(v, u, dep + 1);
        sz[u] += sz[v];
        if(sz[v] > sz[ch[u]]) ch[u] = v;
    }
}
void dfs2(int u, int rt) {
    idx[u] = id++;
    top[u] = rt;
    if(ch[u]) dfs2(ch[u], rt);
    for(int i = head[u]; i != -1; i = ed[i].next) {
        int v = ed[i].v;
        if(v == p[u] || v == ch[u]) continue;
        dfs2(v, v);
    }
}
int ask(int u, int v) {
    int ret = 0;
while(top[u] != top[v]) {
    //一定是top[u]的深度大于等于top[v]
        if(d[top[u]] < d[top[v]]) swap(u, v);
        ret = max(ret, query(1, 1, n, idx[top[u]], idx[u]]));
        u = p[top[u]];
    }
    if(d[u] < d[v]) swap(u, v);
    if(u != v) ret = max(ret, query(1, 1, n, idx[ch[v]], idx[u]));
    return ret;
}
```

## 6.11 点分治

```
void getsz(int u, int fa) {
    sz[u] = 1; f[u] = 0;
    for(int i = head[u]; i != -1; i = ed[i].next) {
        int v = ed[i].v;
        if(v == fa || vis[v]) continue;
        getsz(v, u);
        sz[u] += sz[v];
        f[u] = max(f[u], sz[v]);
    }
}
//找到最大子树最小的点作为分治的中心
void getrt(int r, int u, int fa) {
    //用父边所连的子树更新f
    f[u] = max(f[u], sz[r] - sz[u]);
    if(f[u] < minf) {minf = f[u]; rt = u;}
    for(int i = head[u]; i != -1; i = ed[i].next) {
        int v = ed[i].v;
        if(v == fa || vis[v]) continue;
        getrt(r, v, u);
    }
}
int solve(int u) {
    minf = n;
    getsz(u, 0);
    getrt(u, u, 0);
    vis[rt] = 1;
    int ret = getdp(rt); //分治的结果
    for(int i = head[rt]; i != -1; i = ed[i].next) {
        int v = ed[i].v;
        if(!vis[v]) ret = max(ret, solve(v));
    }
    return ret;
}
```

## 6.12 RMQ

## 6.13 整体二分

```
Divide_Conquer(Q, AL, AR)
//Q是当前处理的操作序列
//WANT是要求的贡献，CURRENT为已经累计的贡献(记录的是1~AL-1内所有修改的贡献)
//[AL, AR]是询问的答案范围区间
if AL = AR then
    将Q中所有是询问操作的答案设为AL
end if
//我们二分答案,AM为当前的判定答案
AM = (AL+AR) / 2
//Solve是主处理函数,只考虑参数满足判定标准[AL, AM]的修改的贡献,因为CURRENT域中已经记录了[1,AL-1]的修改的贡献了,这一步是保证时间复杂度的关键,因为SOLVE只于当前Q的长度有关,而不与整个操作序列的长度有线性关系,这保证了主定理解出来只多一个log
Solve(Q, AL, AM)
//Solve之后Q中各个参数满足判定标准的修改对询问的贡献被存储在ANS数组
//Q1,Q2为了两个临时数组，用于划分操作序列
for i = 1 to Length(Q) do
    if (Q[i].WANT <= Q[i].CURRENT + ANS[i]) then
        //当前已有贡献不小于要求贡献,说明最终答案应当不大于判定答案
        向数组Q1末尾添加Q[i]
    else
        //当前已有贡献小于要求贡献,说明最终答案应当大于判定答案
        //这里是整体二分的关键,把当前贡献累计入总贡献,以后不再重复统计!
        Q[i].CURRENT = Q[i].CURRENT + ANS[i]
        向数组Q2末尾添加Q[i]
    end if
end for
//分治,递归处理
Divide_Conquer(Q1, AL, AM)
Divide_Conquer(Q2, AM+1, AR)
以上别人的伪代码 非常清楚

带修改的整体二分
void cal(int ql, int qr, int l, int mid) {
    for(int i = ql; i <= qr; i++) {
        if(nd[i].k) nd[i].cnt = query(nd[i].r) - query(nd[i].l - 1);
        else if(nd[i].r <= mid) update(nd[i].l, nd[i].cnt);
    }
    for(int i = ql; i <= qr; i++) if(nd[i].r <= mid && (!nd[i].k)) update(nd[i].l, -nd[i].cnt);
}
void divide(int ql, int qr, int l, int r) {
    if(l == r) {
        for(int i = ql; i <= qr; i++) if(nd[i].k) ans[nd[i].id] = l;
        return;
    }
    int mid = (l + r) / 2;
    cal(ql, qr, l, mid);
    int p1 = 0, p2 = 0;
    for(int i = ql; i <= qr; i++) {
        if(nd[i].k) {
            if(nd[i].cnt >= nd[i].k) t1[++p1] = nd[i];
            else {
                nd[i].k -= nd[i].cnt;
                t2[++p2] = nd[i];
            }
        }
        else {
            if(nd[i].r <= mid) t1[++p1] = nd[i];
            else t2[++p2] = nd[i];
        }
    }
    for(int i = 1; i <= p1; i++) nd[ql - 1 + i] = t1[i];
    for(int i = 1; i <= p2; i++) nd[ql + p1 - 1 + i] = t2[i];
    if(p1) divide(ql, ql + p1 - 1, l, mid);
    if(p2) divide(ql + p1, qr, mid + 1, r);
}
```

## 6.14 莫队

```
struct qnode {
    int l, r, id;
    ll a, b;
    bool operator < (const qnode& rhs) const {
        return dv[l] == dv[rhs.l] ? dv[r] < dv[rhs.r] :
            dv[l] < dv[rhs.l];
    }
}qn[MAXN];
int bd = (int)sqrt(n);
for(int i = 1; i <=n; i++) {
    dv[i] = (i - 1) / bd + 1;
}
```

## 6.15 KDtree

给 N 个 K 维点，找这 K 维点里面最近的 M 个点

```
#include <bits/stdc++.h>
using namespace std;
const int N = 50080;
const int K = 5;
typedef pair<int, int> Pair;
struct Node
{
    int x[K], d;
    bool f;
    static int k, cd;
    void read() {
      for (int i = 0; i < k; i++) {
        cin >> x[i];
      }
      f = 0;
    }
    bool operator < (Node node) const {
      return x[cd] < node.x[cd];
    }
};
int Node::k = 0, Node::cd = 0;
int mid(int l, int r)
```

```
{
  return l + (r - l) / 2;
}
int dist(Node a, Node b) //算k维的距离
{
    int res = 0;
    for(int i = 0;i < Node::k; i++)
        res += (a.x[i] - b.x[i])*(a.x[i] - b.x[i]);
    return res;
}
Node a[N];
priority_queue<Pair> pq;  // 距离 下标
void build(int l, int r, int d)
{
    if (l > r) return;
    int m = mid(l, r);
    Node::cd = d;
    nth_element(a+l,a+m,a+r+1); //保证第n大的在第n的位置,类似快排的基准
    //这样就可以保证a[m]是中间的 这样树就可以建的非常平均
    a[m].d = d; //这个点的维度是d
    if(l == r) //到了叶节点了
    {
        a[m].f = true; //应是是不是叶节点的意思
        return;
    }
    build(l, m-1,(d+1)%Node::k); //递归建树
    build(m+1,r, (d+1)%Node::k);
}
int num;
void fd(int l, int r, Node tar)
{
    if (l > r) return;
    int m = mid(l, r);
    int d = dist(a[m], tar);
    if(a[m].f) {   //如果是叶子
      if(pq.size() < num) { // 还每找满m个
        pq.push(make_pair(d, m));
      } else if(d < pq.top().first) { // 已经找了m个了,要删了 并且是比当前最远点小
        pq.pop();
        pq.push(make_pair(d, m));
      }
      return;
    }
    int t = tar.x[a[m].d] - a[m].x[a[m].d];
    // 要调查的点和 这个节点的距离
    if(t > 0) { // 右子树
      fd(m+1, r, tar); //先在右子树
      if(pq.size() < num) {
        pq.push(make_pair(d, m));
        fd(l,m-1,tar); // 再找左子树
      } else {
        if(d < pq.top().first) {
          pq.pop();
          pq.push(make_pair(d, m));
        }
        if(pq.top().first > t*t) fd(l,m-1,tar);
      }
    } else { //
      fd(l,m-1,tar);
      if(pq.size() < num) {
        pq.push(make_pair(d, m));
        fd(m+1,r,tar);
      } else {
        if(pq.top().first > d) {
            pq.pop();
            pq.push(make_pair(d, m));
        }
        if (pq.top().first > t*t) fd(m+1,r,tar);
      }
    }
}
void solve(int n, int k)
{
  Node::k = k;
  for (int i = 0; i < n; i++) a[i].read();
  build(0, n-1, 0);
  int t;
  cin >> t;
  while (t--) {
    Node q;
    q.read();
    while (!pq.empty()) pq.pop();
    cin >> num; // 题目中需要找与q距离最近的num个点
    fd(0, n-1, q);
    vector<int> ans;
    while (!pq.empty()) {
      ans.push_back(pq.top().second);
      pq.pop();
    }
    printf("the␣closest␣%d␣points␣are:\n",num);
    for(int j = num - 1; j >= 0; j--) {
      for(int kk = 0;kk < k;kk++) {
        kk == 0 ? cout << a[ans[j]].x[kk] : cout << '␣'
           << a[ans[j]].x[kk];
      }
      cout << endl;
    }
  }
}
int main()
{
  int n, k;
  while(cin >> n >> k) solve(n, k);
}
```

# Chapter 7

# 字符串

## 7.1 最小表示法

```
#include <cstdio>
#include <algorithm>
#include <iostream>
using namespace std;
const int maxn = 100010;
int main()
{
  int n;
  while (scanf("%d", &n) != EOF)
  {
    string s[maxn];
    for (int ii = 0; ii < n; ii++)
    {
      string ss;
      cin >> ss;
      ss = ss + ss;
      bool flag = false;
      int i = 0, j = 1, k = 0, l = ss.size() / 2, p = 0;
      while (i < l && j < l)
      {
        k = 0;
        while (ss[i + k] == ss[j + k] && k < l) k++;
        if (k == l)
        {
          p = i;
          flag = true;
          break;
        }
    if (ss[i + k] > ss[j + k])
     if (i + k + 1 > j) i = i + k + 1; else i = j + 1;
    else if (j + k + 1 > i) j = j + k + 1;
    else j = i + 1;
   }
   if (!flag)
   if (i < j) p = i; else p = j;
   s[ii] = ss.substr(p, l);
  }
  sort(s, s + n);
  int ans = 1;
  for (int i = 1; i < n; i++)
   if (s[i] != s[i - 1]) ans++;
  printf("%d\n", ans);
 }
}
```

## 7.2 KMP

```
void getfail() {
  f[0] = f[1] = 0;
  for(int i = 1; i < n; i++) {
    int j = f[i];
        //找到与当前后缀匹配的最靠右的前缀的位置
    while(j && c[i] != c[j]) j = f[j];
      f[i + 1] = c[i] == c[j] ? j + 1 : 0;
  }
}
void findp(char *T, char *P, int* f) {
    int n = strlen(T), m = strlen(P);
    int j = 0;
    for(int i = 0; i < n; i++) {
        //如果不匹配就往前找
    while(j && P[j] != T[i]) j = f[j];
    if(P[j] == T[i]) j++;
    if(j == m) printf("%d\n", i - m + 1);
    }
}
```

## 7.3 Manacher

```
int len = strlen(s);
int n = 1, pre = 0, ans = 0;
ss[0] = '$';
for(int i = 0; i < len; i++) {
    ss[n++] = '#';
    ss[n++] = s[i];
}
ss[n] = '#';
for(int i = 1; i < n; i++) {
//pre是在i之前半径延伸地最远的点
//如果pre的区间包含了i，那么i延伸的区间是i向右和i关于pre的对称点向左
延伸的较小值
if(i < pre + p[pre])p[i] = min(p[pre − (i − pre)], p[pre] + pre - i);
else p[i] = 1;
//继续扩展i的回文区间
while(ss[i - p[i]] == ss[i + p[i]]) ++p[i];
//用i更新pre
    if(pre + p[pre] < i + p[i]) pre = i;
}
```

## 7.4 AC 自动机

```
void getfail() {
    queue<int> q;
    for(int i = 0; i < 26; i++) if(ch[0][i]) q.push(ch[0][i]);
    while(!q.empty()) {
        int rt = q.front(); q.pop();
        for(int i = 0; i < 26; i++) {
            int u = ch[rt][i];
            if(!u) {ch[rt][i] = ch[fail[rt]][i]; continue;}
                        //如果本来没有u节点 那么就走失配边 从而形成了图
            q.push(u);
            int v = fail[rt];
            while(v && !ch[v][i]) v = fail[v];
            fail[u] = ch[v][i];
                        //Trie中一个节点可能匹配了多个 所以还要记录上一个匹配的后缀
            last[u] = val[fail[u]] ? fail[u] : last[fail[u]];
        }
    }
}
void print(int j) {
    if(j) {
        printf("%d:␣%d", j, val[j]);
        print(last[j]);
    }
}
void findp(char * T) {
    int len = strlen(T);
    int j = 0;
    for(int i = 0; i < n; i++) {
        int x = T[i] - 'a';
        while(j && !ch[j][x]) j = fail[j];
        j = ch[j][x];
        if(val[j]) print(j);
        else if(last[j]) print(last[j]);
    }
}
```

## 7.5 后缀数组

```
int sa[MAXN], rk[MAXN], ht[MAXN], cnt[MAXN], t1[MAXN], t2[MAXN];
void getsa(int m) {
    int *x = t1, *y = t2;
    for(int i = 0; i < m; i++) cnt[i] = 0;
    for(int i = 0; i < n; i++) cnt[x[i] = s[i]]++;
    for(int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
for(int i = n - 1; i >= 0; i--) sa[--cnt[x[i]]] = i;
//sa数组从0到n-1
    for(int k = 1; ; k *= 2) {
        int p = 0;
        //2nd
        for(int i = n - k; i < n; i++) y[p++] = i;
        for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
        //1st
        for(int i = 0; i < m; i++) cnt[i] = 0;
        for(int i = 0; i < n; i++) cnt[x[y[i]]]++;
        for(int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
        for(int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]]]] = y[i];
        swap(x, y);
        p = 1; x[sa[0]] = 0;
        for(int i = 1; i < n; i++)
            x[sa[i]] = (y[sa[i]] == y[sa[i - 1]] && y[sa[i] + k] == y[sa[i - 1] + k] && sa[i] + k < n && sa[i - 1] + k < n) ? p - 1 : p++;
        if(p >= n) break;
        m = p;
    }
}
void getheight() {
    int k = 0;
    for(int i = 0; i < n; i++) rk[sa[i]] = i;
for(int i = 0; i < n; i++) {
                if(rk[i] == 0) continue;
        if(k) k--;
        int j = sa[rk[i] - 1];
        while(s[i + k] == s[j + k] && i + k < n && j + k < n) k++;
        ht[rk[i]] = k;
    }
}
```

## 7.6 后缀自动机

```
//rt = 1
int ncnt, last, ch[MAXN][26], val[MAXN], par[MAXN];
int c[MAXN], rk[MAXN];
void init(int x, int v) {
    memset(ch[x], 0, sizeof(ch[x]));
    par[x] = 0; val[x] = v;
}
void add(int x) {
    int p = last, np = ++ncnt;
    memset(ch[np], 0, sizeof(ch[np]));
    init(np, val[p] + 1);
    while(p && !ch[p][x]) {
        ch[p][x] = np;
        p = par[p];
    }
    if(p == 0) par[np] = 1;
    else {
        int q = ch[p][x];
        if(val[p] + 1 == val[q]) par[np] = q;
        else {
            int nq = ++ncnt;
            memcpy(ch[nq], ch[q], sizeof(ch[q]));
            val[nq] = val[p] + 1;
            par[nq] = par[q];
            par[q] = nq;
            par[np] = nq;
            while(p && ch[p][x] == q) {
                ch[p][x] = nq;
                p = par[p];
            }
        }
    }
    last = np;
}
void tsort() {
    memset(c, 0, sizeof(c));
    for(int i = 1; i <= ncnt; i++) c[val[i]]++;
    for(int i = 1; i <= ncnt; i++) c[i] += c[i - 1];
    for(int i = 1; i <= ncnt; i++) rk[c[val[i]]--] = i;
}
```

# Chapter 8

# 其他

## 8.1 蔡勒公式

```
int calc(int y,int m,int d)
{
    if (m < 3) m += 12, y--;
    int w = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4
        - y / 100 + y / 400) % 7;
    return w;
}
bool isRun(int YYYY)
{
    return (YYYY % 4 == 0 && YYYY % 100 != 0) || (YYYY %
       400 == 0);
}
```

## 8.2 斜率 DP

## 8.3 最长子序列

```
bool solve()
{
    int n, k;
    scanf("%d%d", &n, &k);
    int a[N];
    int dp[N];
    int g[N];
    rep(i, 1, n) scanf("%d", &a[i]), g[i] = inf;
    int ans1 = 0;
    rep(i, 1, n) {
        int k = upper_bound(g+1, g+1+n, a[i]) - g;
        dp[i] = k;
        g[k] = a[i];
        ans1 = max(ans1, k);
    }
```

## 8.4 四边形不等式

```
for(int i=2;i<=m+1;++i)
{
    s[i][n+1]=n;
    for(int j=n;j>i;--j)
    {
        for(int k=s[i-1][j];k<=s[i][j+1];++k)
        {
            LL tmp=dp[i-1][k]+w[k+1][j];
    //      cout<<i<<' '<<j<<' '<<k<<' '<<tmp<<endl;
            if(tmp<dp[i][j])
            {
                dp[i][j]=tmp;
                s[i][j]=k;
            }
        }
    }
}
```

## 8.5 数位 DP

```
//基本上都是这个模板，但是有点慢
int dp(int pos, int sum, int lz, int lim) {
    if(!lim && !lz && d[pos][sum] != -1) return d[pos][sum];
    if(pos == 0) return sum >= 33;
    int bd = lim ? s[pos] : 1;
    int res = 0;
    for(int i = 0; i <= bd; i++) {
        if(i == 0) {
            if(lz) res += dp(pos - 1, sum, lz, lim && i == bd);
            else res += dp(pos - 1, sum + 1, 0, lim && i == bd);
        }
        else res += dp(pos - 1, sum - 1, 0, lim && i == bd);
    }
    if(!lim && !lz) d[pos][sum] = res;
    return res;
}
int solve(int x) {
    int cnt = 0;
    while(x) {
        if(x & 1) s[++cnt] = 1;
        else s[++cnt] = 0;
        x /= 2;
    }
    return dp(cnt, 33, 1, 1);
}
```

## 8.6 大数

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

#define MAXN 9999
#define MAXSIZE 10
#define DLEN 4

class BigNum
{
private:
    int a[2000];    //可以控制大数的位数
    int len;       //大数长度
public:
    BigNum(){ len = 1;memset(a,0,sizeof(a)); }   //构造函数
    BigNum(const int);       //将一个int类型的变量转化为大数
    BigNum(const char*);     //将一个字符串类型的变量转化为大数
    BigNum(const BigNum &);  //拷贝构造函数
    BigNum &operator=(const BigNum &);   //重载赋值运算符，大数之间进行赋值运算

    friend istream& operator>>(istream&,  BigNum&);   //重载输入运算符
    friend ostream& operator<<(ostream&,  BigNum&);   //重载输出运算符

    BigNum operator+(const BigNum &) const;   //重载加法运算符，两个大数之间的相加运算
    BigNum operator-(const BigNum &) const;   //重载减法运算符，两个大数之间的相减运算
    BigNum operator*(const BigNum &) const;   //重载乘法运算符，两个大数之间的相乘运算
    BigNum operator/(const int   &) const;    //重载除法运算符，大数对一个整数进行相除运算

    BigNum operator^(const int  &) const;    //大数的n次方运算
    int    operator%(const int  &) const;    //大数对一个int类型的变量进行取模运算
    bool   operator>(const BigNum & T)const;   //大数和另一个大数的大小比较
    bool   operator<(const BigNum & T) const;
    bool   operator==(const BigNum & T) const;
    bool   operator>(const int & t)const;      //大数和一个int类型的变量的大小比较
    bool   operator<(const int &t) const;
    bool   operator==(const int &t) const;

    void print();       //输出大数
};

bool BigNum::operator==(const BigNum & T) const {
    return !(*this > T) && !(T > *this);
}
bool BigNum::operator==(const int &t) const {
    BigNum T = BigNum(t);
    return *this == T;
}
bool BigNum::operator<(const BigNum & T) const {
    return T > *this;
}
bool BigNum::operator<(const int &t) const {
    return BigNum(t) > *this;
```

```
}
BigNum::BigNum(const int b)     //将一个int类型的变量转化
    为大数
{
    int c,d = b;
    len = 0;
    memset(a,0,sizeof(a));
    while(d > MAXN)
    {
        c = d - (d / (MAXN + 1)) * (MAXN + 1);
        d = d / (MAXN + 1);
        a[len++] = c;
    }
    a[len++] = d;
}
BigNum::BigNum(const char*s)     //将一个字符串类型的变量
    转化为大数
{
    int t,k,index,l,i;
    memset(a,0,sizeof(a));
    l=strlen(s);
    len=l/DLEN;
    if(l%DLEN)
        len++;
    index=0;
    for(i=l-1;i>=0;i-=DLEN)
    {
        t=0;
        k=i-DLEN+1;
        if(k<0)
            k=0;
        for(int j=k;j<=i;j++)
            t=t*10+s[j]-'0';
        a[index++]=t;
    }
}
BigNum::BigNum(const BigNum & T) : len(T.len)  //拷贝构造
    函数
{
    int i;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)
        a[i] = T.a[i];
}
```

```
BigNum & BigNum::operator=(const BigNum & n)   //重载赋值运算符，大数之间进行赋值运算
{
    int i;
    len = n.len;
    memset(a,0,sizeof(a));
    for(i = 0 ; i < len ; i++)
        a[i] = n.a[i];
    return *this;
}
istream& operator>>(istream & in,  BigNum & b)   //重载输入运算符
{
    char ch[MAXSIZE*4];
    int i = -1;
    in>>ch;
    int l=strlen(ch);
    int count=0,sum=0;
    for(i=l-1;i>=0;)
    {
        sum = 0;
        int t=1;
        for(int j=0;j<4&&i>=0;j++,i--,t*=10)
        {
            sum+=(ch[i]-'0')*t;
        }
        b.a[count]=sum;
        count++;
    }
    b.len =count++;
    return in;

}
ostream& operator<<(ostream& out,  BigNum& b)   //重载输出运算符
{
    int i;
    cout << b.a[b.len - 1];
    for(i = b.len - 2 ; i >= 0 ; i--)
    {
        cout.width(DLEN);
        cout.fill('0');
        cout << b.a[i];
    }
    return out;
}
```

```

BigNum BigNum::operator+(const BigNum & T) const   //两个大数之间的相加运算
{
    BigNum t(*this);
    int i,big;      //位数
    big = T.len > len ? T.len : len;
    for(i = 0 ; i < big ; i++)
    {
        t.a[i] +=T.a[i];
        if(t.a[i] > MAXN)
        {
            t.a[i + 1]++;
            t.a[i] -=MAXN+1;
        }
    }
    if(t.a[big] != 0)
        t.len = big + 1;
    else
        t.len = big;
    return t;
}
BigNum BigNum::operator-(const BigNum & T) const   //两个大数之间的相减运算
{
    int i,j,big;
    bool flag;
    BigNum t1,t2;
    if(*this>T)
    {
        t1=*this;
        t2=T;
        flag=0;
    }
    else
    {
        t1=T;
        t2=*this;
        flag=1;
    }
    big=t1.len;
    for(i = 0 ; i < big ; i++)
    {
        if(t1.a[i] < t2.a[i])
        {
            j = i + 1;
            while(t1.a[j] == 0)
                j++;
            t1.a[j--]--;
            while(j > i)
                t1.a[j--] += MAXN;
            t1.a[i] += MAXN + 1 - t2.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    }
    t1.len = big;
    while(t1.a[t1.len - 1] == 0 && t1.len > 1)
    {
        t1.len--;
        big--;
    }
    if(flag)
        t1.a[big-1]=0-t1.a[big-1];
    return t1;
}

BigNum BigNum::operator*(const BigNum & T) const   //两个大数之间的相乘运算
{
    BigNum ret;
    int i,j,up;
    int temp,temp1;
    for(i = 0 ; i < len ; i++)
    {
        up = 0;
        for(j = 0 ; j < T.len ; j++)
        {
            temp = a[i] * T.a[j] + ret.a[i + j] + up;
            if(temp > MAXN)
            {
                temp1 = temp - temp / (MAXN + 1) * (MAXN + 1);
                up = temp / (MAXN + 1);
                ret.a[i + j] = temp1;
            }
            else
            {
                up = 0;
                ret.a[i + j] = temp;
            }
        }
        if(up != 0)
            ret.a[i + j] = up;
    }
    ret.len = i + j;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
BigNum BigNum::operator/(const int & b) const   //大数对一个整数进行相除运算
{
    BigNum ret;
    int i,down = 0;
    for(i = len - 1 ; i >= 0 ; i--)
    {
        ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
        down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
    }
    ret.len = len;
    while(ret.a[ret.len - 1] == 0 && ret.len > 1)
        ret.len--;
    return ret;
}
int BigNum::operator %(const int & b) const    //大数对一个int类型的变量进行取模运算
{
    int i,d=0;
    for (i = len-1; i>=0; i--)
    {
        d = ((d * (MAXN+1))% b + a[i])% b;
    }
    return d;
}
BigNum BigNum::operator^(const int & n) const    //大数的n次方运算
{
    BigNum t,ret(1);
    int i;
    if(n<0)
        exit(-1);
    if(n==0)
        return 1;
    if(n==1)
        return *this;
    int m=n;
    while(m>1)
    {
        t=*this;
        for( i=1;i<<1<=m;i<<=1)
        {
            t=t*t;
        }
        m-=i;
        ret=ret*t;
        if(m==1)
            ret=ret*(*this);
    }
    return ret;
}
bool BigNum::operator>(const BigNum & T) const   //大数和另一个大数的大小比较
{
    int ln;
    if(len > T.len)
        return true;
    else if(len == T.len)
    {
        ln = len - 1;
        while(a[ln] == T.a[ln] && ln >= 0)
            ln--;
        if(ln >= 0 && a[ln] > T.a[ln])
            return true;
        else
            return false;
    }
    else
        return false;
}
bool BigNum::operator >(const int & t) const    //大数和一个int类型的变量的大小比较
{
    BigNum b(t);
    return *this>b;
}

void BigNum::print()    //输出大数
{
    int i;
    printf("%d", a[len-1]);
    for (int i = len-2; i >= 0; --i) {
        printf("%04d", a[i]);
    }
    puts("");
}
```

## 8.7 可以重复走的异或路径

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int MAXN = 1e5 + 10;
vector<pair<int, ll> > g[MAXN];
vector<ll> a;
int vis[MAXN];
ll dis[MAXN];
ll b[60];
int n, m;
void dfs(int u, ll d) {
    vis[u] = 1;
    dis[u] = d;
    for(int i = 0; i < g[u].size(); i++) {
        int v = g[u][i].first;
        if(vis[v]) a.push_back(d^g[u][i].second^dis[v]);
        else dfs(v, dis[u]^g[u][i].second);
    }
}
int main() {
    cin >> n >> m;
    for(int i = 1; i <= m; i++) {
        int u, v;
        ll w;
        scanf("%d%d%lld", &u, &v, &w);
        g[u].push_back(make_pair(v, w));
        g[v].push_back(make_pair(u, w));
    }
    dfs(1, 0);
    for(int i = 0; i < a.size(); i++) {
        for(int j = 40; j >= 0; j--) {
            if(a[i] & (1LL << j)) {
                if(!b[j]) {b[j] = a[i]; break;}
                else a[i]^=b[j];
            }
        }
    }
    ll ans = dis[n];
    for(int i = 40; i >= 0; i--) ans = min(ans, ans^b[i]);
    cout << ans << endl;
}
```

# Bibliography

[1] 余勇. *ACM 国际大学生程序设计竞赛: 算法与实现*. 北京, 清华大学出版社, 2012