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

April 17, 2018

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

```
const int N = 1000000 + 9;
bool p[N];
int a[N];
int main()
{
  int n;
  cin >> n;
  int cnt = 0;
  for (int i = 0; i <= n; i++) p[i] = true;
  for (int i = 2; i <= n; i++) {
```

```
    if (p[i]) a[cnt++] = i;
    for (int j = 0; j < cnt; j++) {
        if (i * a[j] > n) break;
        p[i * a[j]] = false;
        if (i % a[j] == 0) break;
    }
  }
  cout << cnt << endl;
}
```

1.6 素数函数

```
#include <bits/stdc++.h>
typedef long long ll;
using namespace std;
ll f[340000],g[340000], n;
void init(){
    ll i, j, m;
    for(m = 1; m * m <= n; ++m)f[m]=n/m-1;
    for(i=1;i<=m;++i)g[i]=i-1;
    for(i=2;i<=m;++i){
        if(g[i]==g[i-1])continue;
        for(j=1;j<=min(m-1,n/i/i);++j){
            if(i*j<m)f[j]-=f[i*j]-g[i-1];
            else f[j]-=g[n/i/j]-g[i-1];
        }
        for(j=m;j>=i*i;--j)g[j]-=g[j/i]-g[i-1];
    }
}
int main()
{
    while (cin >> n) {
      init();
      cout << f[1] << endl;
    }
}
```

1.7 欧拉函数

1.7.1 递推求

```
// 傻逼写的
int phi[MAXN];
```

```cpp
void init() {
    memset(phi, 0, sizeof(phi));
    phi[1] = 1;
    for(int i = 2; i < MAXN; i++) if(!phi[i])
        for(int j = i; j < MAXN; j += i) {
            if(!phi[j]) phi[j] = j;
            phi[j] = phi[j] / i * (i - 1);
        }
}
// 聪明人写的
void init()
{
  for (int i = 2; i < maxb; i++)
    if (f[i] == 0)
      for (int j = i; j < maxb; j += i)
      {
        if (f[j] == 0) f[j] = j;
        f[j] = (f[j] / i) * (i - 1);
      }
  for (int i = 1; i < maxb; i++)
    f[i] += f[i - 1];
}
```

1.7.2 单个求

```cpp
ll phi(ll n){
    ll ans = n,a = n;
    for(ll i = 2; i * i <= a; i++){
        if(a % i == 0){
            ans = ans / i * (i - 1);
            while(a % i == 0) a /= i;
        }
    }
    if(a > 1) ans = ans / a * (a - 1);
    return ans;
}
```

1.8 莫比乌斯函数

```cpp
int mu[MAXN], prm[MAXN], vis[MAXN];
void getmu(int n) {
    int sz = 0;
    mu[1] = 1;
```

```
    for(int i = 2; i <= n; i++) {
        if(!vis[i]) prm[++sz] = i, mu[i] = -1;
        for(int j = 1; j <= sz && prm[j] * i <= n; j++) {
            vis[i * prm[j]] = 1;
            if(i % prm[j] == 0) {mu[i * prm[j]] = 0;
               break;}
            mu[i * prm[j]] = -mu[i];
        }
    }
}
```

1.9 逆元

1.9.1 递推求

```
ll mul_mod(ll x, ll y, ll mod = MOD)
{
    return x * y % MOD;
}
m[1] = 1;
inv[1] = 1;
for (n = 2; ;n++) {
    s[n] = s[n - 1] + n;
    m[n] = mul_mod(m[n - 1], n);
    inv[n] = (MOD - MOD / n) * inv[MOD % n] % MOD;
    if (s[n] > MAXX) break;
}
```

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);
```

```
26      for (int i = 0; i < A.n; i++)
27          for (int j = 0; j < B.m; j++)
28              for (int k = 0; k < A.m; k++)
29                  t.a[i][j] = (t.a[i][j] + A.a[i][k] * B.a[
                        k][j] % MOD) % MOD;
30      return t;
31  }
32  Mat pow(Mat A, int n)
33  {
34      Mat t(A.n, A.m);
35      t.eye();
36      while (n > 0) {
37          if (n % 2) t = mul(t, A);
38          A = mul(A, A);
39          n /= 2;
40      }
41      return t;
42  }
43  int det(Mat A)
44  {
45    int cnt = 0,ans = 1, n = A.n;
46    for(int i = 0; i < n; i++) {
47      for(int j = i+1; j < n; j++) {
48        int x = i, y = j;
49        while(A.a[y][i]) {
50          int t = A.a[x][i] / A.a[y][i];
51          for(int k = 0; k < n; k++) {
52            A.a[x][k] = A.a[x][k] - A.a[y][k]*t;
53          }
54          swap(x, y);
55        }
56        if(x != i) {
57          for(int k = 0; k < n; k++) {
58            swap(A.a[x][k], A.a[y][k]);
59          }
60          cnt ^= 1;
61        }
62      }
63      if(A.a[i][i] == 0) return 0;
64      else ans *= A.a[i][i];
65    }
66    if(cnt) ans *= -1;
67      return ans;
68  }
```

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;
                }
```

```
20                  if (j < k) j += k;
21          }
22  }
23
24  void fft(complex *y, long long len, long long on) {
25          change(y, len);
26          for (int h = 2; h <= len; h <<= 1) {
27                  complex wn(cos(-on * 2 * pi / h), sin(-on
                        * 2 * pi / h));
28                  for (int j = 0; j < len; j += h) {
29                          complex w(1, 0);
30                          for (int k = j; k < j + h / 2; k
                                ++) {
31                                  complex u = y[k];
32                                  complex t = w * y[k + h /
                                        2];
33                                  y[k] = u + t;
34                                  y[k + h / 2] = u - t;
35                                  w = w * wn;
36                          }
37                  }
38          }
39          if (on == -1)
40                  for (int i = 0; i < len; i++)
41                          y[i].a /= len;
42  }
```

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 最大团

```
/*
最大团搜索算法
Si = {vi, vi+1, ... , vn}
mc[i] = MC(Si) = 点集合 Si 的最大团
MC(V) = mc[1]
mc[i] = mc[i+1] || mc[i] = mc[i+1] + 1 因为最多增加一个点
后一种情况发生是在 Si 中找到一个包含vi的团
所以只要搜 是否在 Si 中有一个包含vi且比当前最大团还大的团
*/
#include <bits/stdc++.h>
#define rep(i, l, r) for (int (i) = (l); (i) <= (r); (i)
    ++)
#define log(x) cout << #x << " = " << x << endl
#define mem(x, y) memset((x), (y), sizeof(x))
using namespace std;
int n, ans;
bool found;
const int N = 50 + 9;
int g[N][N];
int len[N], mc[N], li[N][N];
void dfs(int sz)
{
    if (len[sz] == 0) {
        if (sz > ans) {
            ans = sz;
            found = true;
        }
    }
```

```
28      for (int k = 0; k < len[sz] && !found; k++) {
29          if (sz + len[sz] - k <= ans) break;
30          int i = li[sz][k];
31          if (sz + mc[i] <= ans) break;
32          len[sz+1] = 0;
33          for (int j = k + 1; j < len[sz]; j++) {
34              if (g[i][li[sz][j]])
35                  li[sz+1][len[sz+1]++] = li[sz][j];
36          }
37          dfs(sz+1);
38      }
39  }
40  int max_cluster()
41  {
42      mc[n] = ans = 1;
43      for (int i = n - 1; i; i--) {
44          found = false;
45          len[1] = 0;
46          rep(j, i+1, n) {
47              if (g[i][j]) li[1][len[1]++] = j;
48          }
49          dfs(1);
50          mc[i] = ans;
51      }
52      return ans;
53  }
54  int solve()
55  {
56      rep(i, 1, n) rep(j, 1, n) scanf("%d", &g[i][j]);
57      return max_cluster();
58  }
59  int main()
60  {
61      for (;;) {
62          scanf("%d", &n);
63          if (n == 0) break;
64          printf("%d\n", solve());
65      }
66  }
```

4.2 图的遍历和连通性

4.2.1 割点和桥

```
1  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;
}
```

4.2.2 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);
    }
}
```

4.2.3 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);
    }
}
```

4.2.4 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

4.2.5 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;}
        }
    }
}
```

4.3 路径

4.3.1 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();
        }
    }
}
```

4.4 匹配

4.4.1 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;
}
```

4.4.2 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);
```

```
71      rep(i, 1, n) rep(j, 1, n) {
72          int x;
73          scanf("%d", &x);
74          solver.add_edge(i, j, x);
75      }
76      return solver.calc();
77  }
78  int main()
79  {
80      KM solver;
81      int n;
82      while (~scanf("%d", &n)) printf("%d\n", solve(n,
            solver));
83  }
```

4.5 树

4.5.1 prim

```
1  int cnt = 1, ans = 0;
2  priority_queue<edge> pq;
3  for(int i = 0; i < g[1].size(); i++) pq.push(g[1][i]);
4  while(!pq.empty()) {
5      edge t = pq.top(); pq.pop();
6      int v = t.v;
7      if(p[v] != 1) {
8          cnt++; ans += t.w;
9          p[v] = 1;
10 for(int i = 0; i < g[v].size(); i++) if(p[g[v][i].v] !=
       1) pq.push(g[v][i]);
11     }
12     if(cnt == n) break;
13 }
```

4.5.2 曼哈顿最小距离生成树

```
1  struct Edge {
2      int u, v, w;
3      bool operator < (const Edge& rhs) const {
4          return w < rhs.w;
5      }
6  }edges[8 * MAXN];
7  struct node {
8      int x, y, id;
```

```
9       bool operator < (const node& rhs) const {
10          return x == rhs.x ? y > rhs.y : x > rhs.x;
11      }
12  }nd[MAXN];
13  int ecnt, n, k, sz;
14  int sq[MAXN];
15  int minv[MAXN], pos[MAXN];
16  int p[MAXN];
17  int lowbit(int x) {return x & -x;}
18  void update(int x, int val, int id) {
19      while(x) {
20          if(val < minv[x]) {
21              minv[x] = val; pos[x] = id;
22          }
23          x -= lowbit(x);
24      }
25  }
26  int query(int x) {
27      int ans = 1 << 30, ret = 0;
28      while(x <= sz) {
29          if(minv[x] < ans) {
30              ans = minv[x]; ret = pos[x];
31          }
32          x += lowbit(x);
33      }
34      //cout << ans << endl;
35      return ret;
36  }
37  int fp(int x) {return p[x] == x ? x : p[x] = fp(p[x]);}
38  void addEdge(int u, int v, int w) {
39      edges[ecnt++] = (Edge) {u, v, w};
40  }
41  void solve() {
42      for(int dir = 0; dir < 4; dir++) {
43          if(dir == 1 || dir == 3)
44              for(int i = 1; i <= n; i++) swap(nd[i].x, nd[
                    i].y);
45          else if(dir == 2)
46              for(int i = 1; i <= n; i++) nd[i].x = -nd[i].
                    x;
47          sort(nd + 1, nd + n + 1);
48          for(int i = 1; i <= n; i++) sq[i] = nd[i].y - nd[
                i].x;
49          sort(sq + 1, sq + n + 1);
50          sz = unique(sq + 1, sq + n + 1) - (sq + 1);
```

```
51          for(int i = 1; i <= sz; i++) {minv[i] = (1 << 30)
                ; pos[i] = 0;}
52          for(int i = 1; i <= n; i++) {
53              int p = lower_bound(sq + 1, sq + sz + 1, nd[i
                    ].y - nd[i].x) - sq;
54              int v = query(p);
55              //cout << p << ' ' << v << endl;
56              if(v) addEdge(nd[i].id, nd[v].id, abs(nd[i].x
                     - nd[v].x) + abs(nd[i].y - nd[v].y));
57              update(p, nd[i].x + nd[i].y, i);
58          }
59      }
60      sort(edges, edges + ecnt);
61      int cnt = 0;
62      for(int i = 1; i <= n; i++) p[i] = i;
63      for(int i = 0; i < ecnt; i++) {
64          Edge e = edges[i];
65          int x = fp(e.u), y = fp(e.v);
66          if(x != y) {
67              p[x] = y;
68              if(++cnt == n - k) {cout << e.w << endl;
                    break;}
69              //cout << cnt << endl;
70          }
71      }
72  }
```

4.6 网络流

4.6.1 最大流 Dinic

```
1   int bfs(int s, int t) {
2       memset(vis, false, sizeof(vis));
3       queue<int> q;
4       q.push(s);
5       vis[s] = true; d[s] = 0;
6       while(!q.empty()) {
7           int u = q.front(); q.pop();
8           for(int i = 0; i < g[u].size(); i++) {
9               Edge e = edges[g[u][i]];
10              if(vis[e.v]) continue;//不能再走访问过的点
11              if(e.cap > e.flow) {
12                  d[e.v] = d[u] + 1;
13                  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;
}
```

4.6.2 **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;
```

```
12      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
13      bool vis[MAXN];         // BFS使用
14      int d[MAXN];            // 从起点到i的距离
15      int cur[MAXN];         // 当前弧指针
16      int p[MAXN];           // 可增广路上的上一条弧
17      int num[MAXN];         // 距离标号计数
18
19      void add_edge(int from, int to, int cap) {
20        edges.push_back((Edge){from, to, cap, 0});
21        edges.push_back((Edge){to, from, 0, 0});
22        m = edges.size();
23        g[from].push_back(m-2);
24        g[to].push_back(m-1);
25      }
26
27      bool BFS() {
28        memset(vis, 0, sizeof(vis));
29        queue<int> Q;
30        Q.push(t);
31        vis[t] = 1;
32        d[t] = 0;
33        while(!Q.empty()) {
34          int x = Q.front(); Q.pop();
35          for(int i = 0; i < g[x].size(); i++) {
36            Edge& e = edges[g[x][i]^1];
37            if(!vis[e.from] && e.cap > e.flow) {
38              vis[e.from] = 1;
39              d[e.from] = d[x] + 1;
40              Q.push(e.from);
41            }
42          }
43        }
44        return vis[s];
45      }
46
47      void init(int nn) {
48        n = nn;
49          mem(g, 0);
50        edges.clear();
51      }
52
53      void ClearFlow() {
54        for(int i = 0; i < edges.size(); i++) edges[i].
              flow = 0;
```

```
55        }
56
57        int Augment() {
58          int x = t, a = inf;
59          while(x != s) {
60            Edge& e = edges[p[x]];
61            a = min(a, e.cap-e.flow);
62            x = edges[p[x]].from;
63          }
64          x = t;
65          while(x != s) {
66            edges[p[x]].flow += a;
67            edges[p[x]^1].flow -= a;
68            x = edges[p[x]].from;
69          }
70          return a;
71        }
72
73        int max_flow(int s, int t, int need) {
74          this->s = s; this->t = t;
75          int flow = 0;
76          BFS();
77          memset(num, 0, sizeof(num));
78          for(int i = 0; i < n; i++) num[d[i]]++;
79          int x = s;
80          memset(cur, 0, sizeof(cur));
81          while(d[s] < n) {
82            if(x == t) {
83              flow += Augment();
84              if(flow >= need) return flow;
85              x = s;
86            }
87            int ok = 0;
88            for(int i = cur[x]; i < g[x].size(); i++) {
89              Edge& e = edges[g[x][i]];
90              if(e.cap > e.flow && d[x] == d[e.to] + 1) {
                    // Advance
91                ok = 1;
92                p[e.to] = g[x][i];
93                cur[x] = i; // 注意
94                x = e.to;
95                break;
96              }
97            }
98            if(!ok) { // Retreat
```

```
99              int m = n-1; // 初值注意
100             for(int i = 0; i < g[x].size(); i++) {
101               Edge& e = edges[g[x][i]];
102               if(e.cap > e.flow) m = min(m, d[e.to]);
103             }
104             if(--num[d[x]] == 0) break;
105             num[d[x] = m+1]++;
106             cur[x] = 0; // 注意
107             if(x != s) x = edges[p[x]].from;
108           }
109         }
110         return flow;
111       }
112
113       vector<int> Mincut() { // call this after maxflow
114         BFS();
115         vector<int> ans;
116         for(int i = 0; i < edges.size(); i++) {
117           Edge& e = edges[i];
118           if(!vis[e.from] && vis[e.to] && e.cap > 0) ans.
                push_back(i);
119         }
120         return ans;
121       }
122
123       void Reduce() {
124         for(int i = 0; i < edges.size(); i++) edges[i].
              cap -= edges[i].flow;
125       }
126
127       void print() {
128         printf("Graph:\n");
129         for(int i = 0; i < edges.size(); i++)
130           printf("%d->%d, %d, %d\n", edges[i].from, edges
                [i].to , edges[i].cap, edges[i].flow);
131       }
132     };
```

4.6.3 费用流

4.6.4 无源无汇有容量下界网络的可行流

```
1 #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) {
```

```
47              vis[e.to] = true;
48              d[e.to] = d[x] + 1;
49              q.push(e.to);
50            }
51          }
52        }
53        return vis[t];
54      }
55      int DFS(int x, int a)
56      {
57        if (x == t || a == 0) return a;
58        int flow = 0;
59        int f;
60        for (int& i = cur[x]; i < g[x].size(); i++) {
61          Edge& e = edges[g[x][i]];
62          if (d[x] + 1 == d[e.to] && (f = DFS(e.to, min(a,
                e.cap - e.flow))) > 0) {
63            e.flow += f;
64            edges[g[x][i]^1].flow -= f;
65            flow += f;
66            a -= f;
67            if (a == 0) break;
68          }
69        }
70        return flow;
71      }
72      int max_flow(int ss, int tt)
73      {
74          s = ss, t = tt;
75        int flow = 0;
76        while (BFS()) {
77          memset(cur, 0, sizeof(cur));
78          flow += DFS(s, INF);
79        }
80        return flow;
81      }
82      void solve(int source, int sink)
83      {
84          max_flow(source, sink);
85          int sz = g[source].size() - 1;
86          bool flag = true;
87          rep(i, 0, sz) {
88              if (edges[g[source][i]].flow < edges[g[source
                    ][i]].cap) {
89                  puts("NO");
90                  flag = false;
```

```
91              }
92          }
93          if (flag) {
94              puts("YES");
95              rep(i, 1, m) {
96                 //log(low[i]);
97                  printf("%d\n", edges[i*2+1].cap-edges[i
                        *2+1].flow + low[i]);
98              }
99          }
100     }
101 };
102 void solve(Dinic &solver)
103 {
104     solver.init();
105     scanf("%d", &m);
106     int source = n + 1, sink = n + 2;
107     // sink -> source cap : inf
108     solver.add_edge(sink, source, INF);
109     int in[N];
110     mem(in, 0);
111     mem(low, 0);
112     rep(i, 1, m) {
113         int u, v, b, c;
114         scanf("%d%d%d%d", &u, &v, &b, &c);
115         low[i] = b; // 流量下界
116         in[u] -= b;
117         in[v] += b;
118         solver.add_edge(u, v, c - b);
119     }
120     rep(i, 1, n) {
121         if (in[i] < 0) {
122             solver.add_edge(i, sink, -in[i]);
123         } else if (in[i] > 0) {
124             solver.add_edge(source, i, in[i]);
125         }
126     }
127     solver.solve(source, sink);
128 }
129 int main()
130 {
131     Dinic solver;
132     while (~scanf("%d", &n)) solve(solver);
133 }
```

4.7 其他

4.7.1 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

4.7.2 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 并将紧跟的 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++)
```

```
27        if (ccw(a[i], a[i + 1], a[i + 2]) * ccw(a[i + 1], a
              [i + 2], a[i + 3]) < 0)
28        {
29          flag = false;
30          break;
31        }
32      if (flag) puts("convex"); else puts("concave");
33    }
34  }
```

5.2 判断线段是否相交

```
1  struct P
2  {
3    double x, y;
4  };
5  struct Segment
6  {
7    P p, q;
8  };
9  double ccw(P a, P b, P c)
10 {
11    return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y
          - a.y);
12 }
13 bool intersects(Segment a, Segment b)
14 {
15    if (ccw(a.p, a.q, b.p) * ccw(a.p, a.q, b.q) > 0)
          return false;
16    if (ccw(b.p, b.q, a.p) * ccw(b.p, b.q, a.q) > 0)
          return false;
17    return true;
18 }
```

5.3 求对称点求交点

```
1  using namespace std;
2  const double eps = 1e-10;
3  double add (double a, double b)
4  {
5    if (abs(a + b) < eps * (abs(a) + abs(b))) return 0;
6    return a + b;
7  }
```

```
8  struct P
9  {
10   double x, y;
11   P(){}
12   P(double x, double y):x(x),y(y){}
13   P operator + (P p)
14   {
15     return P(add(x, p.x), add(y, p.y));
16   }
17   P operator - (P p)
18   {
19     return P(add(x, -p.x), add(y, -p.y));
20   }
21   P operator * (double d)
22   {
23     return P(x * d, y * d);
24   }
25 };
26 double det(P p1, P p2)
27 {
28   return add(p1.x * p2.y, -p1.y * p2.x);
29 }
30 bool g_equal(double a, double b)
31 {
32   if (a > b - eps && a < b + eps) return true;
33   return false;
34 }
35 //求p关于p1, p2所成直线的对称点
36 P symmetric_point(P p, P p1, P p2)
37 {
38   //直线与x轴垂直
39         if (g_equal(p1.x, p2.x)) return P(2 * p1.x - p.x,
               p.y);
40   double k = (p1.y - p2.y ) / (p1.x - p2.x);
41   if (g_equal(k, 0)) return P(p.x, 2 * p1.y - p.y);
42   double x = (2*k*k*p1.x + 2*k*p.y - 2*k*p1.y - k*k*p.x +
         p.x) / (1 + k * k);
43   double y = p.y - (x - p.x) / k;
44   return P(x, y);
45 }
46 P intersection(P p1, P p2, P q1, P q2)
47 {
48   return p1 + (p2 - p1) * (det(q2 - q1, q1 - p1) / det(q2
         - q1, p2 - p1));
49 }
```

5.4 终极模板

```
#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; }
```

```
38
39 //向量长度\share\CodeBlocks\templates\wizard\console\cpp
40 double Length(Vector A) { return sqrt(Dot(A, A)); }
41
42 //向量夹角
43 double Angle(Vector A, Vector B) { return acos(Dot(A, B)
       / Length(A) / Length(B)); }
44
45 //向量叉积
46 double Cross(Vector A, Vector B) { return A.x*B.y - A.y*B
       .x; }
47
48 //三角形有向面积的二倍
49 double Area2(Point A, Point B, Point C) { return Cross(B-
       A, C-A); }
50
51 //向量逆时针旋转rad度(弧度)
52 Vector Rotate(Vector A, double rad) {
53     return Vector(A.x*cos(rad)-A.y*sin(rad), A.x*sin(rad)
          +A.y*cos(rad));
54 }
55
56 //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。
57 Vector Normal(Vector A) {
58     double L = Length(A);
59     return Vector(-A.y/L, A.x/L);
60 }
61
62 /*
       ************************************************************************
63 使用复数类实现点及向量的简单操作
64
65 #include <complex>
66 typedef complex<double> Point;
67 typedef Point Vector;
68
69 double Dot(Vector A, Vector B) { return real(conj(A)*B)}
70 double Cross(Vector A, Vector B) { return imag(conj(A)*B)
       ;}
71 Vector Rotate(Vector A, double rad) { return A*exp(Point
       (0, rad)); }
72
```

```
73  ************************************************************************
        */
74
75  /*
        ************************************************************************
76  * 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;
77  * 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;
78  * 当t 无限制时， 该参数方程表示直线。
79  * 当t > 0时， 该参数方程表示射线。
80  * 当 0 < t < 1时， 该参数方程表示线段。
81  ************************************************************************
        */
82
83  //直线交点,须确保两直线有唯一交点。
84  Point GetLineIntersection(Point P, Vector v, Point Q,
        Vector w) {
85      Vector u = P - Q;
86      double t = Cross(w, u)/Cross(v, w);
87      return P+v*t;
88  }
89
90  //点到直线距离
91  double DistanceToLine(Point P, Point A, Point B) {
92      Vector v1 = B - A, v2 = P - A;
93      return fabs(Cross(v1, v2) / Length(v1)); //不取绝对
            值，得到的是有向距离
94  }
95
96  //点到线段的距离
97  double DistanceToSegmentS(Point P, Point A, Point B) {
98      if(A == B) return Length(P-A);
99      Vector v1 = B-A, v2 = P-A, v3 = P-B;
100     if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
101     else if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
102     else return fabs(Cross(v1, v2)) / Length(v1);
103 }
104
105 //点在直线上的投影
106 Point GetLineProjection(Point P, Point A, Point B) {
107     Vector v = B - A;
108     return A+v*(Dot(v, P-A)/Dot(v, v));
109 }
110
```

```
111 //线段相交判定，交点不在一条线段的端点
112 bool SegmentProperIntersection(Point a1, Point a2, Point
        b1, Point b2) {
113     double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2
            -a1);
114     double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2
            -b1);
115     return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) <
            0;
116 }
117
118 //判断点是否在点段上，不包含端点
119 bool OnSegment(Point P, Point a1, Point a2) {
120     return dcmp(Cross(a1-P, a2-P) == 0 && dcmp((Dot(a1-P,
            a2-P)) < 0));
121 }
122
123 //计算凸多边形面积
124 double ConvexPolygonArea(Point *p, int n) {
125     double area = 0;
126     for(int i = 1; i < n-1; i++)
127         area += Cross(p[i] - p[0], p[i+1] - p[0]);
128     return area/2;
129 }
130
131 //计算多边形的有向面积
132 double PolygonArea(Point *p, int n) {
133     double area = 0;
134     for(int i = 1; i < n-1; i++)
135         area += Cross(p[i] - p[0], p[i+1] - p[0]);
136     return area/2;
137 }
138
139 /*
        ************************************************************************

140 * 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。
141 * 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;
142 ************************************************************************
        */
143
144 struct Circle {
145     Point c;
```

```
146     double r;
147 
148     Circle(Point c, double r) : c(c), r(r) {}
149     //通过圆心角确定圆上坐标
150     Point point(double a) {
151         return Point(c.x + cos(a)*r, c.y + sin(a)*r);
152     }
153 };
154 
155 struct Line {
156     Point p;
157     Vector v;
158     double ang;
159     Line() {}
160     Line(Point p, Vector v) : p(p), v(v) {}
161     bool operator < (const Line& L) const {
162         return ang < L.ang;
163     }
164 };
165 
166 //直线和圆的交点，返回交点个数，结果存在sol中。
167 //该代码没有清空sol。
168 int getLineCircleIntersecion(Line L, Circle C, double& t1
        , double& t2, vector<Point>& sol) {
169     double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L
            .p.y - C.c.y;
170     double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*
            d - C.r*C.r;
171     double delta = f*f - 4*e*g;
172     if(dcmp(delta) < 0) return 0; //相离
173     if(dcmp(delta) == 0) {        //相切
174         t1 = t2 = -f / (2*e);
175         sol.push_back(C.point(t1));
176         return 1;
177     }
178     //相交
179     t1 = (-f - sqrt(delta)) / (2*e); sol.push_back(C.
            point(t1));
180     t2 = (-f + sqrt(delta)) / (2*e); sol.push_back(C.
            point(t2));
181     return 2;
182 }
183 
184 //两圆相交
```

```
185 int getCircleCircleIntersection(Circle C1, Circle C2,
        vector<Point>& sol) {
186     double d = Length(C1.c - C2.c);
187     if(dcmp(d) == 0) {
188         if(dcmp(C1.r - C2.r == 0)) return -1;    //两圆完
                全重合
189         return 0;                                //同心
                圆，半径不一样
190     }
191     if(dcmp(C1.r + C2.r - d) < 0) return 0;
192     if(dcmp(fabs(C1.r - C2.r) == 0)) return -1;
193
194     double a = Angle(C2.c - C1.c);               //向量
            C1C2的极角
195     double da = acos((C1.r*C1.r + d*d - C2.r*C2.r) / (2*
            C1.r*d));
196     //C1C2到C1P1的角
197     Point p1 = C1.point(a-da), p2 = C1.point(a+da);
198     sol.push_back(p1);
199     if(p1 == p2) return 1;
200     sol.push_back(p2);
201     return 2;
202 }
203
204 const double PI = acos(-1);
205 //过定点做圆的切线
206 //过点p做圆C的切线，返回切线个数。v[i]表示第i条切线
207 int getTangents(Point p, Circle C, Vector* v) {
208     Vector u = C.c - p;
209     double dist = Length(u);
210     if(dist < C.r) return 0;
211     else if(dcmp(dist - C.r) == 0) {
212         v[0] = Rotate(u, PI/2);
213         return 1;
214     } else {
215         double ang = asin(C.r / dist);
216         v[0] = Rotate(u, -ang);
217         v[1] = Rotate(u, +ang);
218         return 2;
219     }
220 }
221
222 //两圆的公切线
223 //返回切线的个数，-1表示有无数条公切线。
224 //a[i], b[i] 表示第i条切线在圆A，圆B上的切点
```

```
225 int getTangents(Circle A, Circle B, Point *a, Point *b) {
226     int cnt = 0;
227     if(A.r < B.r) {
228         swap(A, B); swap(a, b);
229     }
230     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);
231     int rdiff = A.r - B.r;
232     int rsum = A.r + B.r;
233     if(d2 < rdiff*rdiff) return 0;   //内含
234     double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
235     if(d2 == 0 && A.r == B.r) return -1;   //无限多条切线
236     if(d2 == rdiff*rdiff) {         //内切一条切线
237         a[cnt] = A.point(base);
238         b[cnt] = B.point(base);
239         cnt++;
240         return 1;
241     }
242     //有外共切线
243     double ang = acos((A.r-B.r) / sqrt(d2));
244     a[cnt] = A.point(base+ang); b[cnt] = B.point(base+ang
        ); cnt++;
245     a[cnt] = A.point(base-ang); b[cnt] = B.point(base-ang
        ); cnt++;
246     if(d2 == rsum*rsum) {  //一条公切线
247         a[cnt] = A.point(base);
248         b[cnt] = B.point(PI+base);
249         cnt++;
250     } else if(d2 > rsum*rsum) {   //两条公切线
251         double ang = acos((A.r + B.r) / sqrt(d2));
252         a[cnt] = A.point(base+ang); b[cnt] = B.point(PI+
            base+ang); cnt++;
253         a[cnt] = A.point(base-ang); b[cnt] = B.point(PI+
            base-ang); cnt++;
254     }
255     return cnt;
256 }
257
258 typedef vector<Point> Polygon;
259
260 //点在多边形内的判定
261 int isPointInPolygon(Point p, Polygon poly) {
262     int wn = 0;
263     int n = poly.size();
264     for(int i = 0; i < n; i++) {
```

```
265         if(OnSegment(p, poly[i], poly[(i+1)%n])) return
                -1; //在边界上
266         int k = dcmp(Cross(poly[(i+1)%n]-poly[i], p-poly[
                i]));
267         int d1 = dcmp(poly[i].y - p.y);
268         int d2 = dcmp(poly[(i+1)%n].y - p.y);
269         if(k > 0 && d1 <= 0 && d2 > 0) wn++;
270         if(k < 0 && d2 <= 0 && d1 > 0) wn++;
271     }
272     if(wn != 0) return 1;       //内部
273     return 0;                   //外部
274 }
275
276 //凸包
277 /*
    ************************************************************

278 * 输入点数组p, 个数为p, 输出点数组ch。 返回凸包顶点数
279 * 不希望凸包的边上有输入点，把两个<= 改成 <
280 * 高精度要求时建议用dcmp比较
281 * 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
282 ************************************************************
    */
283 int ConvexHull(Point *p, int n, Point* ch) {
284     sort(p, p+n);      //先比较x坐标，再比较y坐标
285     int m = 0;
286     for(int i = 0; i < n; i++) {
287         while(m > 1 && Cross(ch[m-1] - ch[m-2], p[i]-ch[m
                -2]) <= 0) m--;
288         ch[m++] = p[i];
289     }
290     int k = m;
291     for(int i = n-2; i >= 0; i++) {
292         while(m > k && Cross(ch[m-1] - ch[m-2], p[i]-ch[m
                -2]) <= 0) m--;
293         ch[m++] = p[i];
294     }
295     if(n > 1) m--;
296     return m;
297 }
298
299 //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
300 //复杂度O(n2);
301 Polygon CutPolygon(Polygon poly, Point A, Point B) {
```

```
302     Polygon newpoly;
303     int n = poly.size();
304     for(int i = 0; i < n; i++) {
305         Point C = poly[i];
306         Point D = poly[(i+1)%n];
307         if(dcmp(Cross(B-A, C-A)) >= 0) newpoly.push_back(
                C);
308         if(dcmp(Cross(B-A, C-D)) != 0) {
309             Point ip = GetLineIntersection(A, B-A, C, D-C
                    );
310             if(OnSegment(ip, C, D)) newpoly.push_back(ip)
                    ;
311         }
312     }
313     return newpoly;
314 }
315
316 //半平面交
317
318 //点p再有向直线L的左边。(线上不算)
319 bool Onleft(Line L, Point p) {
320     return Cross(L.v, p-L.p) > 0;
321 }
322
323 //两直线交点，假定交点唯一存在
324 Point GetIntersection(Line a, Line b) {
325     Vector u = a.p - b.p;
326     double t = Cross(b.v, u) / Cross(a.v, b.v);
327     return a.p+a.v*t;
328 }
329
330 int HalfplaneIntersection(Line* L, int n, Point* poly) {
331     sort(L, L+n);                //按极角排序
332
333     int first, last;             //双端队列的第一个元素和
            最后一个元素
334     Point *p = new Point[n];     //p[i]为q[i]和q[i+1]的交
            点
335     Line *q = new Line[n];       //双端队列
336     q[first = last = 0] = L[0]; //队列初始化为只有一个半
            平面L[0]
337     for(int i = 0; i < n; i++) {
338         while(first < last && !Onleft(L[i], p[last-1]))
                last--;
```

```
339         while(first < last && !Onleft(L[i], p[first]))
                first++;
340         q[++last] = L[i];
341         if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
342             last--;
343             if(Onleft(q[last], L[i].p)) q[last] = L[i];
344         }
345         if(first < last) p[last-1] = GetIntersection(q[
                last-1], q[last]);
346     }
347     while(first < last && !Onleft(q[first], p[last-1]))
            last--;
348     //删除无用平面
349     if(last-first <= 1) return 0;   //空集
350     p[last] = GetIntersection(q[last], q[first]);
351
352     //从deque复制到输出中
353     int m = 0;
354     for(int i = first; i <= last; i++) poly[m++] = p[i];
355     return m;
356 }
357 int main() {
358
359     return 0;
360 }
```

5.5 K 次圆

```
1  #include <bits/stdc++.h>
2  #define rep(i, l, r) for (int(i) = (l); (i) <= (r); (i)
       ++)
3  #define mem(x, y) memset((x), (y), sizeof(x))
4  using namespace std;
5  const double eps = 1e-8;
6  const int N = 1000 + 9;
7  const double pi = 3.1415926535;
8  double sqr(double x)
9  {
10     return x * x;
11 }
12 int dcmp(double x)
13 {
14     if (fabs(x) < eps)
15         return 0;
```

```
16      else
17          return x < 0 ? -1 : 1;
18  }
19  struct Circle
20  {
21      double x, y, r, angle;
22      int d;
23      Circle() {}
24      Circle(double xx, double yy, double ang = 0, int t =
            0)
25      {
26          x = xx;
27          y = yy;
28          angle = ang;
29          d = t;
30      }
31      void read()
32      {
33          scanf("%lf%lf%lf", &x, &y, &r);
34          d = 1;
35      }
36  };
37
38  Circle cir[N], tp[N * 2];
39  double area[N];
40
41  double dis(Circle a, Circle b)
42  {
43      return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
44  }
45
46  double cross(Circle p0, Circle p1, Circle p2)
47  {
48      return (p1.x - p0.x) * (p2.y - p0.y) - (p1.y - p0.y)
            * (p2.x - p0.x);
49  }
50  //圆相交
51  int CirCrossCir(Circle p1, double r1, Circle p2, double
        r2, Circle &cp1, Circle &cp2)
52  {
53      double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx *
            mx;
54      double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my *
            my;
55      double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq
            - sqr(r1 + r2));
```

```
56      if (d + eps < 0)
57          return 0;
58      if (d < eps)
59          d = 0;
60      else
61          d = sqrt(d);
62      double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) +
            sx * my2;
63      double y = my * ((r1 + r2) * (r1 - r2) + my * sy) +
            sy * mx2;
64      double dx = mx * d, dy = my * d;
65      sq *= 2;
66      cp1.x = (x - dy) / sq;
67      cp1.y = (y + dx) / sq;
68      cp2.x = (x + dy) / sq;
69      cp2.y = (y - dx) / sq;
70      if (d > eps)
71          return 2;
72      else
73          return 1;
74  }
75  bool circmp(const Circle &u, const Circle &v)
76  {
77      return dcmp(u.r - v.r) < 0;
78  }
79
80  bool cmp(const Circle &u, const Circle &v)
81  {
82      if (dcmp(u.angle - v.angle))
83          return u.angle < v.angle;
84      return u.d > v.d;
85  }
86  //0.5*r*r*(K-sin(K))
87  double calc(Circle cir, Circle cp1, Circle cp2)
88  {
89      double ans = (cp2.angle - cp1.angle) * sqr(cir.r) -
            cross(cir, cp1, cp2) + cross(Circle(0, 0), cp1,
            cp2);
90      return ans / 2;
91  }
92
93  void CirUnion(Circle cir[], int n)
94  {
95      Circle cp1, cp2;
96      sort(cir, cir + n, circmp);
97      for (int i = 0; i < n; ++i)
```

```
98          for (int j = i + 1; j < n; ++j)
99              if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir
                    [j].r) <= 0)
100                 cir[i].d++;
101     for (int i = 0; i < n; ++i)
102     {
103         int tn = 0, cnt = 0;
104         for (int j = 0; j < n; ++j)
105         {
106             if (i == j)
107                 continue;
108             if (CirCrossCir(cir[i], cir[i].r, cir[j], cir
                    [j].r,
109                             cp2, cp1) < 2)
110                 continue;
111             cp1.angle = atan2(cp1.y - cir[i].y, cp1.x -
                    cir[i].x);
112             cp2.angle = atan2(cp2.y - cir[i].y, cp2.x -
                    cir[i].x);
113             cp1.d = 1;
114             tp[tn++] = cp1;
115             cp2.d = -1;
116             tp[tn++] = cp2;
117             if (dcmp(cp1.angle - cp2.angle) > 0)
118                 cnt++;
119         }
120         tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y,
                pi, -cnt);
121         tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y,
                -pi, cnt);
122         sort(tp, tp + tn, cmp);
123         int p, s = cir[i].d + tp[0].d;
124         for (int j = 1; j < tn; ++j)
125         {
126             p = s;
127             s += tp[j].d;
128             area[p] += calc(cir[i], tp[j - 1], tp[j]);
129         }
130     }
131 }
132 int n;
133 void solve()
134 {
135     for (int i = 0; i < n; i++) cir[i].read(); // read x
            y r
136     mem(area, 0);
```

```
137     CirUnion(cir, n);
138     for (int i = 1; i <= n; i++)
139     {
140         area[i] -= area[i + 1];
141         printf("[%d] = %.3f\n", i, area[i]);
142     }
143 }
144 
145 int main()
146 {
147     while (~scanf("%d", &n)) solve();
148 }
```

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];
```

```
29          swap(a[heap[p].tag],a[heap[p/2].tag]);
30          swap(heap[p],heap[p/2]);
31      }
32  }
```

6.2 左偏树

```
1   #include <bits/stdc++.h>
2   #define rep(i, l, r) for (int (i) = (l); (i) <= (r); (i)
        ++)
3   using namespace std;
4   int n,m,ans;
5   struct Data{int fa,x,dis,l,r;};
6   const int MAXN = 100001;
7   Data q[MAXN];
8   int find(int x)
9   {
10      return x==q[x].fa?x:q[x].fa=find(q[x].fa);
11  }
12  int merge(int x,int y)
13  {
14      if(!x) return y;
15      if(!y) return x;
16      if(q[x].x<q[y].x) swap(x,y);
17      q[x].r=merge(q[x].r,y);
18      q[q[x].r].fa=x;
19      if(q[q[x].l].dis<q[q[x].r].dis) swap(q[x].l,q[x].r);
20      if(!q[x].r) q[x].dis=0;
21      else q[x].dis=q[q[x].r].dis+1;
22      return x;
23  }
24  int pop(int x)
25  {
26      int l=q[x].l,r=q[x].r;
27      q[l].fa=l,q[r].fa=r;
28      q[x].l=q[x].r=q[x].dis=0;
29      return merge(l,r);
30  }
31  void slove()
32  {
33      int x,y,u,v,l1,l2;
34      rep(i, 1, n) {
35          scanf("%d", &q[i].x);
36          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;
```

```
31              tree[rc].addv += tree[o].addv;
32              tree[o].addv = 0;
33          }
34      }
35      void maintain(int o, int l, int r)
36      {
37          int lc(o*2), rc(o*2+1);
38          if (r > l) {
39              tree[o].sumv = tree[lc].sumv + tree[rc].sumv;
40              tree[o].minv = min(tree[lc].minv,tree[rc].
                    minv);
41              tree[o].maxv = max(tree[lc].maxv,tree[rc].
                    maxv);
42          }
43          if (tree[o].setv >= 0) {
44              tree[o].minv = tree[o].maxv = tree[o].setv;
45              tree[o].sumv = (r-l+1)*tree[o].setv;
46          }
47          if (tree[o].addv) {
48              tree[o].minv += tree[o].addv;
49              tree[o].maxv += tree[o].addv;
50              tree[o].sumv += (r-l+1) * tree[o].addv;
51          }
52      }
53      void update(int o, int l, int r, int op, int ql, int
            qr, int v)
54      {
55          int lc(o*2), rc(o*2+1);
56          if (ql <= l && qr >= r) {
57              if (op == 1) tree[o].addv += v;
58              else {
59                  tree[o].setv = v;
60                  tree[o].addv = 0;
61              }
62          } else {
63              pushdown(o);
64              int m = mid(l, r);
65              if (ql <= m) update(lc, l, m, op, ql, qr, v);
                     else maintain(lc, l, m);
66              if (qr > m) update(rc, m+1, r, op, ql, qr, v)
                    ; else maintain(rc, m+1, r);
67          }
68          maintain(o, l, r);
69      }
70      Node query(int o, int l, int r, int ql, int qr)
71      {
```

```
72          Node res;
73          int lc(o*2), rc(o*2+1), m(mid(l, r));
74          maintain(o, l, r);
75          if (ql <= l && qr >= r) {
76              res.sumv = tree[o].sumv;
77              res.minv = tree[o].minv;
78              res.maxv = tree[o].maxv;
79          } else {
80              pushdown(o);
81              Node lres;
82              lres.minv = INF, lres.maxv = -INF;
83              Node rres(lres);
84              if (ql <= m) lres = query(lc, l, m, ql, qr);
                    else maintain(lc, l, m);
85              if (qr > m) rres = query(rc, m+1, r, ql, qr);
                     else maintain(rc, m+1, r);
86              res.sumv = lres.sumv + rres.sumv;
87              res.minv = min(lres.minv, rres.minv);
88              res.maxv = max(lres.maxv, rres.maxv);
89          }
90          return res;
91      }
92  };
93  Tree tree[MAXR];
94  void solve(int r, int c, int m)
95  {
96      for (int i(1); i <= r; i++) tree[i].tree[1].setv = 0;
97      while (m--) {
98          int op, x1, y1, x2, y2;
99          scanf("%d%d%d%d%d", &op, &x1, &y1, &x2, &y2);
100         if (op < 3) {
101             int v;
102             scanf("%d", &v);
103             for (int i(x1); i <= x2; i++) tree[i].update
                    (1, 1, c, op, y1, y2, v);
104         } else {
105             Node ans;
106             ans.sumv = 0, ans.minv = INF, ans.maxv = -INF
                    ;
107             for (int i(x1); i <= x2; i++) {
108                 Node res(tree[i].query(1, 1, c, y1, y2));
109                 ans.sumv += res.sumv;
110                 ans.minv = min(ans.minv, res.minv);
111                 ans.maxv = max(ans.maxv, res.maxv);
112             }
```

```
113             printf("%d␣%d␣%d\n", ans.sumv, ans.minv, ans.
                    maxv);
114         }
115     }
116 }
117 int main()
118 {
119     //freopen("in", "r", stdin);
120     int r, c, m;
121     while (scanf("%d%d%d", &r, &c, &m) != EOF) {
122         solve(r, c, m);
123     }
124 }
```

6.5 二维线段树

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  const int MAXN = 800 * 4 + 10;
4  int mx[MAXN][MAXN], mn[MAXN][MAXN], g[810][810];
5  int n;
6  void buildy(int xo, int o, int l, int r, int x) {
7      if(l == r) {
8          if(x != -1) mx[xo][o] = mn[xo][o] = g[x][l];
9          else {
10             int xlch = xo * 2, xrch = xlch + 1;
11             mx[xo][o] = max(mx[xlch][o], mx[xrch][o]);
12             mn[xo][o] = min(mn[xlch][o], mn[xrch][o]);
13         }
14         return;
15     }
16     int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
17     buildy(xo, lch, l, mid, x);
18     buildy(xo, rch, mid + 1, r, x);
19     mx[xo][o] = max(mx[xo][lch], mx[xo][rch]);
20     mn[xo][o] = min(mn[xo][lch], mn[xo][rch]);
21 }
22 void buildx(int o, int l, int r) {
23     if(l == r) {
24         buildy(o, 1, 1, n, l);
25         return;
26     }
27     int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
28     buildx(lch, l, mid);
```

```
29      buildx(rch, mid + 1, r);
30      buildy(o, 1, 1, n, -1);
31  }
32  void updatey(int xo, int o, int l, int r, int y, int v) {
33      if(l == r) {
34          if(v != -1) mn[xo][o] = mx[xo][o] = v;
35          else {
36              int xlch = xo * 2, xrch = xlch + 1;
37              mx[xo][o] = max(mx[xlch][o], mx[xrch][o]);
38              mn[xo][o] = min(mn[xlch][o], mn[xrch][o]);
39          }
40          return;
41      }
42      int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
43      if(y <= mid) updatey(xo, lch, l, mid, y, v);
44      else updatey(xo, rch, mid +1, r, y, v);
45      mx[xo][o] = max(mx[xo][lch], mx[xo][rch]);
46      mn[xo][o] = min(mn[xo][lch], mn[xo][rch]);
47  }
48  void updatex(int o, int l, int r, int x, int y, int v) {
49      if(l == r) {
50          updatey(o, 1, 1, n, y, v);
51          return;
52      }
53      int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
54      if(mid >= x) updatex(lch, l, mid, x, y, v);
55      else updatex(rch, mid + 1, r, x, y, v);
56      updatey(o, 1, 1, n, y, -1);
57  }
58  int minv, maxv;
59  void queryy(int xo, int o, int l, int r, int yl, int yr)
        {
60      if(yl <= l && r <= yr) {
61          minv = min(minv, mn[xo][o]);
62          maxv = max(maxv, mx[xo][o]);
63          return;
64      }
65      int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
66      if(mid >= yl) queryy(xo, lch, l, mid, yl, yr);
67      if(mid < yr) queryy(xo, rch, mid + 1, r, yl, yr);
68  }
69  void queryx(int o, int l, int r, int xl, int xr, int yl,
        int yr) {
70      if(xl <= l && r <= xr) {
71          queryy(o, 1, 1, n, yl, yr);
72          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) {
```

```
11      node* k = o->ch[d^1]; o->ch[d^1] = k->ch[d]; k->ch[d]
            = o;
12      //一定先维护o，因为o是k的子节点
13      o->maintain(); k->maintain();
14          o = k;
15      }
16      void ins(node*& o, int x) {
17          if(o == null) {
18              o = &nd[ncnt++];
19              *o = *null;
20              o->v = x; o->k = rand();
21              o->w = 1; o->maintain();
22          }
23          else {
24              int d = o->cmp(x);
25              if(d == -1) {o->w++; o->sum++;}
26              else {
27                  ins(o->ch[d], x);
28                  o->maintain();
29                  if(o < o->ch[d]) rot(o, d^1);
30              }
31          }
32      }
33      void del(node*& o, int x) {
34          if(o == null) return;
35          int d = o->cmp(x);
36          if(d == -1) {
37              if(o->w > 1) {o->w--; o->sum--;}
38              else {
39                  if(o->ch[0] != null && o->ch[1] != null)
                        {
40                      int dd = o->ch[0] > o->ch[1];
41                      rot(o, dd);
42                      del(o->ch[dd], x);
43                  }
44                  else o = o->ch[0] == null ? o->ch[1] : o
                        ->ch[0];
45              }
46          }
47          else del(o->ch[d], x);
48          o->maintain();
49      }
50      int getrk(node* o, int x) {
51          if(o == null) return 0;
52          int d = o->cmp(x);
53          if(d == -1) return o->ch[0]->sum + 1;
```

```
54          else {
55              if(d == 1) return o->ch[0]->sum + o->w +
                    getrk(o->ch[1], x);
56              return getrk(o->ch[0], x);
57          }
58      }
59      int getkth(node* o, int x) {
60          if(o == null) return 0;
61          int ls = o->ch[0]->sum, cs = o->w;
62          if(x <= ls) return getkth(o->ch[0], x);
63          else if(x > ls && x <= ls + cs) return o->v;
64          else return getkth(o->ch[1], x - ls - cs);
65      }
66      int ans;
67      //找第一个小于x的数
68      void getpre(node* o, int x) {
69          if(o == null) return;
70          if(x > o->v) {
71              ans = o->v;
72              getpre(o->ch[1], x);
73          }
74          else getpre(o->ch[0], x);
75      }
76      //第一个大于x的数
77      void getnxt(node* o, int x) {
78          if(o == null) return;
79          if(x < o->v) {
80              ans = o->v;
81              getnxt(o->ch[0], x);
82          }
83          else getnxt(o->ch[1], x);
84      }
```

6.7 splay

```
1  int ch[MAXN][2], fa[MAXN], val[MAXN], sz[MAXN], cnt[MAXN
       ];
2  int rt, ncnt;
3  void update(int r) {sz[r] = sz[ch[r][0]] + sz[ch[r][1]] +
       cnt[r];}
4  void rot(int x) {
5      int y = fa[x], z = fa[y], d = ch[y][1] == x;
6      ch[y][d] = ch[x][d^1], fa[ch[y][d]] = y;
7      ch[x][d^1] = y; fa[y] = x;
```

```
8       fa[x] = z;
9       if(z) ch[z][ch[z][1] == y] = x;
10      update(y);
11  }
12  void splay(int r, int tp) {
13      for(int y, z; (y = fa[r]) != tp; rot(r)) {
14          z = fa[y];
15          if(z == tp) continue;
16          if((ch[z][0] == y) == (ch[y][0] == r)) rot(y);
17          else rot(r);
18      }
19      if(!tp) rt = r;
20      update(r);
21  }
22  void ins(int r, int x) {
23      int y = 0;
24      while(r && val[r] != x) {y = r; r = ch[r][val[r] < x
            ];}
25      if(r) ++cnt[r];
26      else {
27          r = ++ncnt;
28          sz[r] = cnt[r] = 1;
29          val[r] = x; fa[r] = y;
30          ch[r][0] = ch[r][1] = 0;
31          if(y) ch[y][val[y] < x] = r;
32      }
33      splay(r, 0);
34  }
35  void get(int v) {
36      int x = rt; if(!x) return;
37      while(ch[x][val[x] < v] && val[x] != v) x = ch[x][val
            [x] < v];
38      splay(x, 0);
39  }
40  int getrk(int v) {
41      get(v);
42      return sz[ch[rt][0]];
43  }
44  int getkth(int x) {
45      int y = rt, p;
46      if(x > sz[rt]) return 0;
47      while(1) {
48          p = ch[y][0];
49          if(sz[p] + cnt[y] < x) {
50              x -= sz[p] + cnt[y];
51              y = ch[y][1];
```

```
52          }
53          else if(sz[p] >= x) y = p;
54          else return val[y];
55      }
56  }
57  int nxt(int x, bool op) {
58      get(x);
59      if((val[rt] > x && op) || (val[rt] < x && !op))
            return rt;
60      int p = ch[rt][op];
61      while(ch[p][op^1]) p = ch[p][op^1];
62      return p;
63  }
64  void del(int v) {
65      int p = nxt(v, 0), s = nxt(v, 1);
66      splay(p, 0);
67      splay(s, p);
68      p = ch[s][0];
69      if(cnt[p] > 1) -- cnt[p], splay(p, 0);
70      else ch[s][0] = 0;
71  }
```

6.8 **倍增** LCA

```
1   void initp() {
2       for(int j = 1; (1 << j) <= n; j++)
3           for(int i = 1; i <= n; i++) if(p[i][j - 1])//一定
                要有这个if
4               p[i][j] = p[p[i][j - 1]][j - 1];
5   }
6   int LCA(int u, int v) {
7       if(d[u] < d[v]) swap(u, v);
8   int lim;
9   //确定最大的2^lim不超过d[u]
10      for(lim = 0; (1 << lim) <= d[u]; lim++); lim--;
11  int ret = 0;
12  //把u上升到v相同的高度
13      for(int i = lim; i >= 0; i--) if(d[u] - (1 << i) >= d
            [v]) u = p[u][i];
14      if(u == v) return u; //一定要有这个判断
15      for(int i = lim; i >= 0; i--) if(p[u][i] != p[v][i])
            {u = p[u][i]; v = p[v][i];}
16      return p[u][0];
17  }
```

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;
```

```
27      int ret = getdp(rt); //分治的结果
28      for(int i = head[rt]; i != -1; i = ed[i].next) {
29          int v = ed[i].v;
30          if(!vis[v]) ret = max(ret, solve(v));
31      }
32      return ret;
33  }
```

6.12 RMQ

6.13 整体二分

```
1   Divide_Conquer(Q, AL, AR)
2   //Q是当前处理的操作序列
3   //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)
4   //[AL, AR]是询问的答案范围区间
5   if AL = AR then
6       将Q中所有是询问操作的答案设为AL
7   end if
8   //我们二分答案,AM为当前的判定答案
9   AM = (AL+AR) / 2
10  //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
11  Solve(Q, AL, AM)
12  //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
13  //Q1,Q2为了两个临时数组，用于划分操作序列
14  for i = 1 to Length(Q) do
15      if (Q[i].WANT <= Q[i].CURRENT + ANS[i]) then
16          //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
17          向数组Q1末尾添加Q[i]
18      else
19          //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
20          //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;
    }
```

```
66      int t = tar.x[a[m].d] - a[m].x[a[m].d];
67      // 要调查的点和 这个节点的距离
68      if(t > 0) { // 右子树
69        fd(m+1, r, tar); //先在右子树
70        if(pq.size() < num) {
71          pq.push(make_pair(d, m));
72          fd(l,m-1,tar); // 再找左子树
73        } else {
74          if(d < pq.top().first) {
75            pq.pop();
76            pq.push(make_pair(d, m));
77          }
78          if(pq.top().first > t*t) fd(l,m-1,tar);
79        }
80      } else { //
81        fd(l,m-1,tar);
82        if(pq.size() < num) {
83          pq.push(make_pair(d, m));
84          fd(m+1,r,tar);
85        } else {
86          if(pq.top().first > d) {
87              pq.pop();
88              pq.push(make_pair(d, m));
89          }
90          if (pq.top().first > t*t) fd(m+1,r,tar);
91        }
92      }
93  }
94  void solve(int n, int k)
95  {
96    Node::k = k;
97    for (int i = 0; i < n; i++) a[i].read();
98    build(0, n-1, 0);
99    int t;
100   cin >> t;
101   while (t--) {
102     Node q;
103     q.read();
104     while (!pq.empty()) pq.pop();
105     cin >> num; // 题目中需要找与q距离最近的num个点
106     fd(0, n-1, q);
107     vector<int> ans;
108     while (!pq.empty()) {
109       ans.push_back(pq.top().second);
110       pq.pop();
111     }
```

```
112     printf("the␣closest␣%d␣points␣are:\n",num);
113     for(int j = num - 1; j >= 0; j--) {
114       for(int kk = 0;kk < k;kk++) {
115         kk == 0 ? cout << a[ans[j]].x[kk] : cout << '␣'
              << a[ans[j]].x[kk];
116       }
117       cout << endl;
118     }
119   }
120 }
121 int main()
122 {
123   int n, k;
124   while(cin >> n >> k) solve(n, k);
125 }
```

6.16 优先队列可删除任意元素的堆

6.17 动态树带

6.18 动态树维护后缀自动机 parent 树

6.19 虚树

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;
        }
```

```
29          if (ss[i + k] > ss[j + k])
30            if (i + k + 1 > j) i = i + k + 1; else i = j +
                  1;
31          else if (j + k + 1 > i) j = j + k + 1;
32          else j = i + 1;
33        }
34        if (!flag)
35        if (i < j) p = i; else p = j;
36        s[ii] = ss.substr(p, l);
37      }
38      sort(s, s + n);
39      int ans = 1;
40      for (int i = 1; i < n; i++)
41        if (s[i] != s[i - 1]) ans++;
42      printf("%d\n", ans);
43    }
44  }
```

7.2 KMP

```
1  void getfail() {
2      f[0] = f[1] = 0;
3      for(int i = 1; i < n; i++) {
4          int j = f[i];
5                  //找到与当前后缀匹配的最靠右的前缀的位置
6          while(j && c[i] != c[j]) j = f[j];
7              f[i + 1] = c[i] == c[j] ? j + 1 : 0;
8      }
9  }
10 void findp(char *T, char *P, int* f) {
11         int n = strlen(T), m = strlen(P);
12         int j = 0;
13         for(int i = 0; i < n; i++) {
14                 //如果不匹配就往前找
15         while(j && P[j] != T[i]) j = f[j];
16         if(P[j] == T[i]) j++;
17         if(j == m) printf("%d\n", i - m + 1);
18         }
19 }
```

7.3 Manacher

```
1  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 {
```

```
21              int nq = ++ncnt;
22              memcpy(ch[nq], ch[q], sizeof(ch[q]));
23              val[nq] = val[p] + 1;
24              par[nq] = par[q];
25              par[q] = nq;
26              par[np] = nq;
27              while(p && ch[p][x] == q) {
28                  ch[p][x] = nq;
29                  p = par[p];
30              }
31          }
32      }
33      last = np;
34  }
35  void tsort() {
36      memset(c, 0, sizeof(c));
37      for(int i = 1; i <= ncnt; i++) c[val[i]]++;
38      for(int i = 1; i <= ncnt; i++) c[i] += c[i - 1];
39      for(int i = 1; i <= ncnt; i++) rk[c[val[i]]--] = i;
40  }
```

7.7 shift-and

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);
```

```
10              else res += dp(pos - 1, sum + 1, 0, lim && i
                    == bd);
11          }
12          else res += dp(pos - 1, sum - 1, 0, lim && i ==
                bd);
13      }
14      if(!lim && !lz) d[pos][sum] = res;
15      return res;
16  }
17  int solve(int x) {
18      int cnt = 0;
19      while(x) {
20          if(x & 1) s[++cnt] = 1;
21          else s[++cnt] = 0;
22          x /= 2;
23      }
24      return dp(cnt, 33, 1, 1);
25  }
```

8.6 大数

```
1   #include <cstdio>
2   #include <cstring>
3   #include <cstdlib>
4   #include <iostream>
5   #include <algorithm>
6   using namespace std;
7
8   #define MAXN 9999
9   #define MAXSIZE 10
10  #define DLEN 4
11
12  class BigNum
13  {
14  private:
15      int a[2000];    //可以控制大数的位数
16      int len;       //大数长度
17  public:
18      BigNum(){ len = 1;memset(a,0,sizeof(a)); }   //构造函
            数
19      BigNum(const int);       //将一个int类型的变量转化为
            大数
20      BigNum(const char*);     //将一个字符串类型的变量转化
            为大数
```

```
21      BigNum(const BigNum &);  //拷贝构造函数
22      BigNum &operator=(const BigNum &);   //重载赋值运算
            符，大数之间进行赋值运算
23
24      friend istream& operator>>(istream&,  BigNum&);   //
            重载输入运算符
25      friend ostream& operator<<(ostream&,  BigNum&);   //
            重载输出运算符
26
27      BigNum operator+(const BigNum &) const;   //重载加法
            运算符，两个大数之间的相加运算
28      BigNum operator-(const BigNum &) const;   //重载减法
            运算符，两个大数之间的相减运算
29      BigNum operator*(const BigNum &) const;   //重载乘法
            运算符，两个大数之间的相乘运算
30      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
31
32      BigNum operator^(const int  &) const;    //大数的n次
            方运算
33      int    operator%(const int  &) const;    //大数对一个
            int类型的变量进行取模运算
34      bool   operator>(const BigNum & T)const;   //大数和另
            一个大数的大小比较
35      bool   operator<(const BigNum & T) const;
36      bool   operator==(const BigNum & T) const;
37      bool   operator>(const int & t)const;      //大数和一
            个int类型的变量的大小比较
38      bool   operator<(const int &t) const;
39      bool   operator==(const int &t) const;
40
41      void print();       //输出大数
42  };
43
44  bool BigNum::operator==(const BigNum & T) const {
45      return !(*this > T) && !(T > *this);
46  }
47  bool BigNum::operator==(const int &t) const {
48      BigNum T = BigNum(t);
49      return *this == T;
50  }
51  bool BigNum::operator<(const BigNum & T) const {
52      return T > *this;
53  }
54  bool BigNum::operator<(const int &t) const {
55      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];
}
```

```
97  BigNum & BigNum::operator=(const BigNum & n)   //重载赋值
        运算符，大数之间进行赋值运算
98  {
99      int i;
100     len = n.len;
101     memset(a,0,sizeof(a));
102     for(i = 0 ; i < len ; i++)
103         a[i] = n.a[i];
104     return *this;
105 }
106 istream& operator>>(istream & in,  BigNum & b)   //重载输
        入运算符
107 {
108     char ch[MAXSIZE*4];
109     int i = -1;
110     in>>ch;
111     int l=strlen(ch);
112     int count=0,sum=0;
113     for(i=l-1;i>=0;)
114     {
115         sum = 0;
116         int t=1;
117         for(int j=0;j<4&&i>=0;j++,i--,t*=10)
118         {
119             sum+=(ch[i]-'0')*t;
120         }
121         b.a[count]=sum;
122         count++;
123     }
124     b.len =count++;
125     return in;
126
127 }
128 ostream& operator<<(ostream& out,  BigNum& b)   //重载输
        出运算符
129 {
130     int i;
131     cout << b.a[b.len - 1];
132     for(i = b.len - 2 ; i >= 0 ; i--)
133     {
134         cout.width(DLEN);
135         cout.fill('0');
136         cout << b.a[i];
137     }
138     return out;
139 }
```

```
140
141 BigNum BigNum::operator+(const BigNum & T) const   //两个
    大数之间的相加运算
142 {
143     BigNum t(*this);
144     int i,big;      //位数
145     big = T.len > len ? T.len : len;
146     for(i = 0 ; i < big ; i++)
147     {
148         t.a[i] +=T.a[i];
149         if(t.a[i] > MAXN)
150         {
151             t.a[i + 1]++;
152             t.a[i] -=MAXN+1;
153         }
154     }
155     if(t.a[big] != 0)
156         t.len = big + 1;
157     else
158         t.len = big;
159     return t;
160 }
161 BigNum BigNum::operator-(const BigNum & T) const   //两个
    大数之间的相减运算
162 {
163     int i,j,big;
164     bool flag;
165     BigNum t1,t2;
166     if(*this>T)
167     {
168         t1=*this;
169         t2=T;
170         flag=0;
171     }
172     else
173     {
174         t1=T;
175         t2=*this;
176         flag=1;
177     }
178     big=t1.len;
179     for(i = 0 ; i < big ; i++)
180     {
181         if(t1.a[i] < t2.a[i])
182         {
183             j = i + 1;
```

```
184             while(t1.a[j] == 0)
185                 j++;
186             t1.a[j--]--;
187             while(j > i)
188                 t1.a[j--] += MAXN;
189             t1.a[i] += MAXN + 1 - t2.a[i];
190         }
191         else
192             t1.a[i] -= t2.a[i];
193     }
194     t1.len = big;
195     while(t1.a[t1.len - 1] == 0 && t1.len > 1)
196     {
197         t1.len--;
198         big--;
199     }
200     if(flag)
201         t1.a[big-1]=0-t1.a[big-1];
202     return t1;
203 }
204
205 BigNum BigNum::operator*(const BigNum & T) const   //两个
    大数之间的相乘运算
206 {
207     BigNum ret;
208     int i,j,up;
209     int temp,temp1;
210     for(i = 0 ; i < len ; i++)
211     {
212         up = 0;
213         for(j = 0 ; j < T.len ; j++)
214         {
215             temp = a[i] * T.a[j] + ret.a[i + j] + up;
216             if(temp > MAXN)
217             {
218                 temp1 = temp - temp / (MAXN + 1) * (MAXN
                        + 1);
219                 up = temp / (MAXN + 1);
220                 ret.a[i + j] = temp1;
221             }
222             else
223             {
224                 up = 0;
225                 ret.a[i + j] = temp;
226             }
227         }
```

```
228             if(up != 0)
229                 ret.a[i + j] = up;
230         }
231         ret.len = i + j;
232         while(ret.a[ret.len - 1] == 0 && ret.len > 1)
233             ret.len--;
234         return ret;
235     }
236     BigNum BigNum::operator/(const int & b) const   //大数对
        一个整数进行相除运算
237     {
238         BigNum ret;
239         int i,down = 0;
240         for(i = len - 1 ; i >= 0 ; i--)
241         {
242             ret.a[i] = (a[i] + down * (MAXN + 1)) / b;
243             down = a[i] + down * (MAXN + 1) - ret.a[i] * b;
244         }
245         ret.len = len;
246         while(ret.a[ret.len - 1] == 0 && ret.len > 1)
247             ret.len--;
248         return ret;
249     }
250     int BigNum::operator %(const int & b) const    //大数对一
        个int类型的变量进行取模运算
251     {
252         int i,d=0;
253         for (i = len-1; i>=0; i--)
254         {
255             d = ((d * (MAXN+1))% b + a[i])% b;
256         }
257         return d;
258     }
259     BigNum BigNum::operator^(const int & n) const    //大数的
        n次方运算
260     {
261         BigNum t,ret(1);
262         int i;
263         if(n<0)
264             exit(-1);
265         if(n==0)
266             return 1;
267         if(n==1)
268             return *this;
269         int m=n;
270         while(m>1)
```

```
271     {
272         t=*this;
273         for( i=1;i<<1<=m;i<<=1)
274         {
275             t=t*t;
276         }
277         m-=i;
278         ret=ret*t;
279         if(m==1)
280             ret=ret*(*this);
281     }
282     return ret;
283 }
284 bool BigNum::operator>(const BigNum & T) const   //大数和
    另一个大数的大小比较
285 {
286     int ln;
287     if(len > T.len)
288         return true;
289     else if(len == T.len)
290     {
291         ln = len - 1;
292         while(a[ln] == T.a[ln] && ln >= 0)
293             ln--;
294         if(ln >= 0 && a[ln] > T.a[ln])
295             return true;
296         else
297             return false;
298     }
299     else
300         return false;
301 }
302 bool BigNum::operator >(const int & t) const    //大数和
    一个int类型的变量的大小比较
303 {
304     BigNum b(t);
305     return *this>b;
306 }
307
308 void BigNum::print()    //输出大数
309 {
310     int i;
311     printf("%d", a[len-1]);
312     for (int i = len-2; i >= 0; --i) {
313         printf("%04d", a[i]);
314     }
```

```
315     puts("");
316 }
```

8.7 可以重复走的异或路径

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  typedef long long ll;
4  const int MAXN = 1e5 + 10;
5  vector<pair<int, ll> > g[MAXN];
6  vector<ll> a;
7  int vis[MAXN];
8  ll dis[MAXN];
9  ll b[60];
10 int n, m;
11 void dfs(int u, ll d) {
12     vis[u] = 1;
13     dis[u] = d;
14     for(int i = 0; i < g[u].size(); i++) {
15         int v = g[u][i].first;
16         if(vis[v]) a.push_back(d^g[u][i].second^dis[v]);
17         else dfs(v, dis[u]^g[u][i].second);
18     }
19 }
20 int main() {
21     cin >> n >> m;
22     for(int i = 1; i <= m; i++) {
23         int u, v;
24         ll w;
25         scanf("%d%d%lld", &u, &v, &w);
26         g[u].push_back(make_pair(v, w));
27         g[v].push_back(make_pair(u, w));
28     }
29     dfs(1, 0);
30     for(int i = 0; i < a.size(); i++) {
31         for(int j = 40; j >= 0; j--) {
32             if(a[i] & (1LL << j)) {
33                 if(!b[j]) {b[j] = a[i]; break;}
34                 else a[i]^=b[j];
35             }
36         }
37     }
38     ll ans = dis[n];
```

```
39      for(int i = 40; i >= 0; i--) ans = min(ans, ans^b[i])
            ;
40      cout << ans << endl;
41  }
```

8.8 最简分数逼近