

# 华理小男孩模板

王亦凡  
主代码手

姚远  
队长

王泽宸  
解题核心

October 9, 2017

# Contents

<b>1</b>	<b>数论</b>	<b>4</b>
1.1	指数降幂公式	4
1.2	威尔逊定理	4
1.3	费马小定理	4
1.4	欧拉定理	4
1.5	质数表	5
1.6	素数函数	5
1.7	欧拉函数	5
1.7.1	递推求	5
1.7.2	单个求	5
1.8	莫比乌斯函数	5
1.9	逆元	5
1.9.1	递推求	5
1.9.2	单个求	5
<b>2</b>	<b>概率论</b>	<b>6</b>
2.1	超几何分布	6
<b>3</b>	<b>数学</b>	<b>7</b>
3.1	矩阵	7
3.1.1	矩阵类	7
3.1.2	高斯消元	9
3.2	整除与剩余	10
3.2.1	扩展欧几里得逆元	10
3.2.2	中国剩余定理	10
3.3	数值计算	11
3.4	其他	11
3.4.1	lucas 定理	11
3.4.2	递推求组合数	11
3.4.3	单个求组合数	12
3.4.4	威佐夫博弈	12
3.4.5	FWT	12
3.4.6	FFT	13
3.4.7	hell 方程	14

<b>4</b>	<b>图论</b>	<b>15</b>
4.1	图的遍历和连通性	15
4.1.1	割点和桥	15
4.1.2	双连通分量	15
4.1.3	强连通分量	16
4.1.4	拓扑排序	17
4.1.5	2SAT	17
4.2	路径	18
4.2.1	非递归欧拉回路	18
4.3	匹配	19
4.3.1	二分图最大匹配	19
4.3.2	二分图最优匹配	19
4.4	树	21
4.4.1	prim	21
4.4.2	曼哈顿最小距离生成树	22
4.5	网络流	24
4.5.1	最大流 Dinic	24
4.5.2	最大流 ISAP	25
4.5.3	费用流	28
4.5.4	无源无汇有容量下界网络的可行流	28
4.6	其他	31
4.6.1	层次遍历	31
4.6.2	图解序列	31
<b>5</b>	<b>计算几何</b>	<b>32</b>
5.1	判断凸包	32
5.2	判断线段是否相交	33
5.3	求对称点求交点	33
5.4	终极模板	35
5.5	K 次圆	44
<b>6</b>	<b>数据结构</b>	<b>45</b>
6.1	手写堆	45
6.2	左偏树	46
6.3	两优先队列模拟堆	47
6.4	线段树	48
6.5	二维线段树	51
6.6	Treap	53
6.7	splay	55
6.8	倍增 LCA	57
6.9	主席树	58
6.10	树剖	58
6.11	点分治	59
6.12	RMQ	60
6.13	整体二分	60
6.14	莫队	62

6.15	KDtree	62
<b>7</b>	<b>字符串</b>	<b>66</b>
7.1	最小表示法	66
7.2	KMP	67
7.3	Manacher	67
7.4	AC 自动机	68
7.5	后缀数组	69
7.6	后缀自动机	70
<b>8</b>	<b>其他</b>	<b>72</b>
8.1	蔡勒公式	72
8.2	斜率 DP	72
8.3	最长子序列	72
8.4	四边形不等式	73
8.5	数位 DP	73
8.6	大数	74
8.7	可以重复走的异或路径	82

# Chapter 1

## 数论

### 1.1 指数降幂公式

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

### 1.2 威尔逊定理

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

### 1.3 费马小定理

$$a^p \equiv a \pmod{p}$$

### 1.4 欧拉定理

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

## 1.5 质数表

## 1.6 素数函数

## 1.7 欧拉函数

### 1.7.1 递推求

### 1.7.2 单个求

## 1.8 莫比乌斯函数

## 1.9 逆元

### 1.9.1 递推求

### 1.9.2 单个求

用费马小定理

## Chapter 2

# 概率论

### 2.1 超几何分布

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

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

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

## Chapter 3

# 数学

### 3.1 矩阵

#### 3.1.1 矩阵类

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 const int N = 1000, MOD = 1E9 + 7;
4 struct Mat
5 {
6     int a[N][N];
7     int n, m;
8     Mat(int n, int m) : n(n), m(m) {memset(a, 0, sizeof(a));}
9     void eye()
10    {
11        memset(a, 0, sizeof(a));
12        for (int i = 0; i < n; i++) a[i][i] = 1;
13    }
14    void print()
15    {
16        for (int i = 0; i < n; i++) {
17            cout << endl;
18            for (int j = 0; j < m; j++)
19                cout << ' ' << a[i][j];
20        }
21    }
22 };
23 Mat mul(Mat A, Mat B)
24 {
25     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

```
1 double a[MAXN][MAXN];
2 double ans[MAXN];
3 bool f[MAXN]; //自由变量
4 int sgn(double x) {return (x > eps) - (x < -eps);}
5 //x 0~equ - 1, y 0~var
6 int gauss(int equ, int var) {
7     int k = 0, col = 0;
8     memset(f, true, sizeof(f));
9     for(k = 0; k < equ && col < var; k++, col++) {
10         int r = k;
11         for(int i = k + 1; i < equ; i++)
12             if(fabs(a[i][col]) > fabs(a[r][col])) r = i;
13         if(r != k) for(int j = k; j <= var; j++) swap(a[r
14             ][j], a[k][j]);
15         if(a[k][col] == 0) {k--; continue;}
16         for(int i = k + 1; i < equ; i++) if(a[i][col]) {
17             for(int j = var; j >= col; j--) a[i][j] -= a[
18                 i][col] / a[k][col] * a[k][j];
19         }
20         for(int i = k; i < equ; i++) if(sgn(a[i][col]) != 0)
21             return 0;
22         if(k < var) {
23             for(int i = k - 1; i >= 0; i--) {
24                 int cnt = 0, p;
25                 for(int j = 0; j < var; j++)
26                     if(sgn(a[i][j]) && f[j])
27                         cnt++, p = j;
28                 if(cnt > 1) continue;
29                 double t = a[i][var];
30                 for(int j = 0; j < var; j++)
31                     if(sgn(a[i][j]) && j != p)
32                         t -= a[i][j] * ans[j];
33                 ans[p] = t / a[i][p];
34                 f[p] = 0;
35             }
36         }
37         for(int i = var - 1; i >= 0; i--) {
38             double t = a[i][var];
39             for(int j = i + 1; j < var; j++)
40                 if(sgn(a[i][j]))
41                     t -= a[i][j] * ans[j];
42             ans[i] = t / a[i][i];
43         }
44     }
```

```

42     return 1;
43 }

```

## 3.2 整除与剩余

### 3.2.1 扩展欧几里得逆元

```

1  扩展欧几里得
2  //ax + by = d, d = gcd(a, b)
3  void gcd(ll a, ll b, ll &d, ll &x, ll &y){
4      if(!b) {d = a; x = 1; y = 0; return;}
5      else{
6          gcd(b, a % b, d, y, x);
7          y -= x * (a / b);
8          return;
9      }
10 }
11
12 逆元
13 //a在模n下的逆元, a和n互素才有逆元
14 ll inv(ll a, ll n) {
15     ll d, x, y;
16     gcd(a, n, d, x, y);
17     return d == 1 ? (x + n) % n : -1;
18 }

```

### 3.2.2 中国剩余定理

```

1  // n个方程 x ≡ a[i] (mod m[i]) i = 0..n-1
2  ll china(int n, ll *a, ll *m)
3  {
4      ll M = m[0], R = a[0];
5      for (int i = 1; i < n; i++) {
6          ll d = __gcd(M, m[i]);
7          ll c = a[i] - R;
8          if (c % d) return -1;
9          ll k1, k2;
10         extgcd(M, m[i], d, k1, k2);
11         k1 = (c / d * k1) % (m[i] / d);
12         R = R + k1 * M;
13         M = M / d * m[i];
14         R %= M;
15     }
16     if (R < 0) R += M;

```

```

17     return R;
18 }

```

### 3.3 数值计算

### 3.4 其他

#### 3.4.1 lucas 定理

```

1 ll qPow (ll a, ll k) {
2     ll ans = 1;
3     while (k) {
4         if (k&1)
5             ans = (ans * a) % p;
6         a = (a * a) % p;
7         k /= 2;
8     }
9     return ans;
10 }
11
12 ll C (ll a, ll b, ll p) {
13     if (a < b)
14         return 0;
15     if (b > a - b)
16         b = a - b;
17
18     ll up = 1, down = 1;
19     for (ll i = 0; i < b; i++) {
20         up = up * (a-i) % p;
21         down = down * (i+1) % p;
22     }
23     return up * qPow(down, p-2) % p; // 逆元
24 }
25 ll lucas (ll a, ll b, ll p) {
26     if (b == 0)
27         return 1;
28     return C(a%p, b%p, p) * lucas(a/p, b/p, p) % p;
29 }

```

#### 3.4.2 递推求组合数

```

1 void calc()
2 {

```

```

3     for (int i = 0; i < N; i++) {
4         c[i][0] = c[i][i] = 1;
5         for (int j = 1; j < i; j++) {
6             c[i][j] = c[i-1][j] + c[i-1][j-1];
7         }
8     }
9 }

```

### 3.4.3 单个求组合数

```

1 ll C(ll n, ll m)
2 {
3     ll c = 1;
4     for (int i = 1; i <= m; i++) {
5         c *= (n - m + i);
6         if (c % i == 0) c /= i;
7     }
8     return c;
9 }

```

### 3.4.4 威佐夫博弈

```

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

```

### 3.4.5 FWT

```

1 //位运算多项式卷积
2 void FWT(int *a, int n) {
3     for(int d = 1; d < n; d<<=1)
4         for(int m = d<<1, i = 0; i < n; i += m)
5             for(int j = 0; j < d; j++) {
6                 int x = a[i + j], y = a[i + j + d];
7                 a[i + j] = (x + y) % MOD, a[i + j + d] =
                    (x - y + MOD) % MOD;
8                 //xor:a[i+j]=x+y,a[i+j+d]=(x-y+mod)%mod;
9                 //and:a[i+j]=x+y;
10                //or:a[i+j+d]=x+y;
11            }
12 }
13 void UFWT(int *a, int n) {
14     for(int d = 1; d < n; d <<= 1)

```

```

15     for(int m = d<<1, i = 0; i < n; i += m)
16         for(int j = 0; j < d; j++) {
17             int x = a[i + j], y = a[i + j + d];
18             a[i + j] = (ll)(x + y) * inv2 % MOD, a[i
                + j + d] = ((ll)(x - y) * inv2 % MOD +
                    MOD) % MOD;
19             //xor:a[i+j]=(x+y)/2,a[i+j+d]=(x-y)/2;
20             //and:a[i+j]=x-y;
21             //or:a[i+j+d]=y-x;
22         }
23     }
24 void conv(int *a, int *b, int n) {
25     FWT(a, n);
26     FWT(b, n);
27     for(int i = 0; i < n; i++) a[i] = (ll)a[i] * b[i] %
        MOD;
28     UFWT(a, n);
29 }

```

### 3.4.6 FFT

```

1  const double pi = atan(1.0) * 4;
2
3  struct complex {
4      double a, b;
5      complex(double aa = 0.0, double bb = 0.0) { a =
        aa; b = bb; }
6      complex operator +(const complex &e) { return
        complex(a + e.a, b + e.b); }
7      complex operator -(const complex &e) { return
        complex(a - e.a, b - e.b); }
8      complex operator *(const complex &e) { return
        complex(a * e.a - b * e.b, a * e.b + b * e.a);
        }
9  };
10
11 void change(complex * y, long long len) {
12     long long i, j, k;
13     for (i = 1, j = len / 2; i < len - 1; i++) {
14         if (i < j) swap(y[i], y[j]);
15         k = len / 2;
16         while (j >= k) {
17             j -= k;
18             k /= 2;
19         }

```

```

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
28             * 2 * pi / h));
29         for (int j = 0; j < len; j += h) {
30             complex w(1, 0);
31             for (int k = j; k < j + h / 2; k
32                 ++){
33                 complex u = y[k];
34                 complex t = w * y[k + h /
35                     2];
36                 y[k] = u + t;
37                 y[k + h / 2] = u - t;
38                 w = w * wn;
39             }
40         }
41     }
42     if (on == -1)
43         for (int i = 0; i < len; i++)
44             y[i].a /= len;
45 }

```

### 3.4.7 hell 方程

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

## Chapter 4

# 图论

### 4.1 图的遍历和连通性

#### 4.1.1 割点和桥

```
1 int pre[MAXN], iscut[MAXN], dfs_clock = 0;
2 int dfs(int u, int fa) {
3     int lowu = pre[u] = ++dfs_clock;
4     int child = 0;
5     for(int i = 0; i < g[u].size(); i++) {
6         int v = g[u][i];
7         if(!pre[v]) {
8             child++;
9             int lowv = dfs(v, u);
10            lowu = min(lowu, lowv);
11            if(lowv >= pre[u]) iscut[u] = true;
12            //if lowu > pre[u] (u, v) is bridge
13
14            }
15            else if(pre[v] < pre[u] && v != fa) lowu = min(
16                lowu, pre[v]);
17        }
18        if(fa < 0 && child == 1) iscut[u] = false;
19        return lowu;
20    }
```

#### 4.1.2 双连通分量

```
1 int dfs(int u, int fa) {
2     int lowu = pre[u] = ++dfs_clock;
3     int child = 0;
```



```

4     for(int i = 0; i < g[u].size(); i++) {
5         int v = g[u][i];
6         if(!pre[v]) {
7             s.push(node(u, v));
8             child++;
9             int lowv = dfs(v, u);
10            lowu = min(lowu, lowv);
11            if(lowv >= pre[u]) {
12                iscut[u] = true;
13                bcc[++bcc_cnt].clear();
14                for(;;) {
15                    node temp = s.top(); s.pop();
16                    //注意割顶可能包含在多个bcc中,
17                    //bccno不是唯一的标准
18                    if(bccno[temp.u] != bcc_cnt)
19                        {bcc[bcc_cnt].push_back(temp.u); bccno[
20                        temp.u] = bcc_cnt;}
21                    if(bccno[temp.v] != bcc_cnt)
22                        {bcc[bcc_cnt].push_back(temp.v); bccno[
23                        temp.v] = bcc_cnt;}
24                    if(temp.u == u && temp.v == v) break;
25                }
26            }
27        }
28        else if(pre[v] < pre[u] && v != fa) {
29            s.push(node(u, v));
30            lowu = min(lowu, pre[v]);
31        }
32    }
33    if(fa < 0 && child == 1) iscut[u] = false;
34    return lowu;
35 }
36 void find_bcc(int n) {
37     memset(pre, 0, sizeof(pre));
38     memset(iscut, false, sizeof(iscut));
39     memset(bccno, 0, sizeof(bccno));
40     dfs_clock = bcc_cnt = 0;
41     for(int i = 1; i <= n; i++) {
42         if(!pre[i]) dfs(i, -1);
43     }
44 }

```

### 4.1.3 强连通分量

```

1 int dfs(int u) {

```

```

2     pre[u] = low[u] = ++dfs_clock;
3     s.push(u);
4     for(int i = 0; i < g[u].size(); i++) {
5         int v = g[u][i];
6         if(!pre[v]) {
7             dfs(v);
8             low[u] = min(low[u], low[v]);
9         }
10        else if(!sccno[v]) {
11            low[u] = min(low[u], pre[v]);
12        }
13    }
14    //如果 low[u] == pre[u], 那么它就是这个 scc 的第一个点
15    if(low[u] == pre[u]) {
16        scc_cnt++;
17        for(;;) {
18            int temp = s.top(); s.pop();
19            sccno[temp] = scc_cnt;
20            if(temp == u) break;
21        }
22    }
23 }
24 void find_scc(int n) {
25     memset(pre, 0, sizeof(pre));
26     memset(low, 0, sizeof(low));
27     memset(sccno, 0, sizeof(sccno));
28     dfs_clock = scc_cnt = 0;
29     for(int i = 1; i <= n; i++) {
30         if(!pre[i]) dfs(i);
31     }
32 }

```

#### 4.1.4 拓扑排序

BFS

**判断是否成环**

拓扑排序形成的 ans 的 sz != n 则成环

#### 4.1.5 2SAT

```

1 //2-sat中不能走的决策往另一个决策连一条边
2 int dfs(int u) {
3     if(vis[u^1]) return false;
4     if(vis[u]) return true;

```

```

5     s[cnt++] = u;
6     vis[u] = 1;
7     for(int i = 0; i < g[u].size(); i++) if(!dfs(g[u][i])
8         ) return false;
9     return true;
10 }
11 bool flag = true;
12 //一定记得+=2
13 for(int i = 2; i <= 2 * n; i+= 2) {
14     if(!vis[i] && !vis[i + 1]) {
15         cnt = 0;
16         if(!dfs(i)) {
17             while(cnt) vis[s[--cnt]] = 0;
18             if(!dfs(i + 1)) {flag = false; break;}
19         }
20     }

```

## 4.2 路径

### 4.2.1 非递归欧拉回路

```

1 void euler(int u) {
2     stack<int> st;
3     st.push(u);
4     nxt[st.size()] = -1;
5     while(!st.empty()) {
6         int a = st.top();
7         int i;
8         for(i = last[a]; i < 26; i++) if(!vis[a][i]) {
9             vis[a][i] = 1;
10            st.push(g[a][i]);
11            nxt[st.size()] = i;
12            last[a] = i + 1;
13            break;
14        }
15        if(i == 26) {
16            if(nxt[st.size()] != -1) ans.push_back((char)
17                (nxt[st.size()] + 'a'));
18            st.pop();
19        }
20    }

```

## 4.3 匹配

### 4.3.1 二分图最大匹配

最小点覆盖的点数 = 二分图最大匹配

最大独立集的点数 = 总点数 - 二分图最大匹配

```
1 int uN,vN;
2 vector<int> g[MAXN];
3 int linker[MAXN];
4 bool used[MAXN];
5 bool dfs(int u) {
6     for(int i = 0; i < g[u].size(); i++) {
7         int v = g[u][i];
8         if(!used[v]) {
9             used[v]=true;
10            if(linker[v] == -1 || dfs(linker[v])) {
11                linker[v]=u;
12                return true;
13            }
14        }
15    }
16    return false;
17 }
18 int hungary() {
19     int res=0;
20     memset(linker, -1, sizeof(linker));
21     for(int u=0; u < uN; u++)
22     {
23         memset(used, 0, sizeof(used));
24         if(dfs(u)) res++;
25     }
26     return res;
27 }
```

### 4.3.2 二分图最优匹配

```
1 struct KM // 二分图最优匹配
2 {
3     int n; //总点数
4     vector<int> g[N];
5     int g2[N][N];
6     void init(int nn)
7     {
8         n = nn;
9         mem(g, 0), mem(g2, 0);
```

```

10     }
11     void add_edge(int u, int v, int w)
12     {
13         g[u].push_back(v);
14         g2[u][v] = w;
15     }
16     int lx[N], ly[N], match[N], lcheck[N], rcheck[N];
17     const int INF = INT_MAX;
18     bool dfs(int u)
19     {
20         lcheck[u] = true;
21         for (int v : g[u]) {
22             if (lx[u] + ly[v] == g2[u][v] && !rcheck[v])
23             {
24                 rcheck[v] = true;
25                 if (match[v] == -1 || dfs(match[v])) {
26                     match[v] = u;
27                     return true;
28                 }
29             }
30         }
31         return false;
32     }
33     void update()
34     {
35         int a = INF;
36         rep(u, 1, n) {
37             if (lcheck[u]) {
38                 for (int v : g[u]) {
39                     if (!rcheck[v]) {
40                         a = min(a, lx[u] + ly[v] - g2[u][v]);
41                     }
42                 }
43             }
44         }
45         rep(i, 1, n) {
46             if (lcheck[i]) lx[i] -= a;
47             if (rcheck[i]) ly[i] += a;
48         }
49     }
50     int calc()
51     {
52         rep(i, 1, n) {
53             lx[i] = *max_element(g2[i]+1, g2[i]+n+1);
54             ly[i] = 0;

```

```

54         match[i] = -1;
55     }
56     rep(i, 1, n) {
57         for (;;) {
58             mem(lcheck, 0);
59             mem(rcheck, 0);
60             if (dfs(i)) break; else update();
61         }
62     }
63     int ans = 0;
64     rep(i, 1, n) if (~match[i]) ans += lx[match[i]] +
        ly[i];
65     return ans;
66 }
67 };
68 int solve(int n, KM &solver)
69 {
70     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 }

```

## 4.4 树

### 4.4.1 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 }

```

#### 4.4.2 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[
45                 i].y);
46         else if(dir == 2)
47             for(int i = 1; i <= n; i++) nd[i].x = -nd[i].
48                 x;
49         sort(nd + 1, nd + n + 1);
50         for(int i = 1; i <= n; i++) sq[i] = nd[i].y - nd[
51             i].x;
52         sort(sq + 1, sq + n + 1);
53         sz = unique(sq + 1, sq + n + 1) - (sq + 1);
54         for(int i = 1; i <= sz; i++) {minv[i] = (1 << 30)
55             ; pos[i] = 0;}
56         for(int i = 1; i <= n; i++) {
57             int p = lower_bound(sq + 1, sq + sz + 1, nd[i
58                 ].y - nd[i].x) - sq;
59             int v = query(p);
60             //cout << p << ' ' << v << endl;
61             if(v) addEdge(nd[i].id, nd[v].id, abs(nd[i].x
62                 - nd[v].x) + abs(nd[i].y - nd[v].y));
63             update(p, nd[i].x + nd[i].y, i);
64         }
65     }
66     sort(edges, edges + ecnt);
67     int cnt = 0;
68     for(int i = 1; i <= n; i++) p[i] = i;
69     for(int i = 0; i < ecnt; i++) {
70         Edge e = edges[i];
71         int x = fp(e.u), y = fp(e.v);
72         if(x != y) {
73             p[x] = y;
74             if(++cnt == n - k) {cout << e.w << endl;
75                 break;}
76             //cout << cnt << endl;
77         }
78     }
79 }

```



## 4.5 网络流

### 4.5.1 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;
14                q.push(e.v);
15            }
16        }
17    }
18    return vis[t];
19 }
20 int dfs(int u, int a, int t) {
21     if(a == 0 || u == t) return a;
22     int flow = 0, f;
23     for(int &i = cur[u]; i < g[u].size(); i++) {
24         Edge &e = edges[g[u][i]];
25         if(d[e.v] == d[u] + 1 && (f = dfs(e.v, min(a, e.
26             cap - e.flow), t)) > 0) { //如果从v走还可以增广
27             e.flow += f;
28             edges[g[u][i]^1].flow -= f;
29             flow += f;
30             a -= f;
31             if(a == 0) break;
32         }
33     }
34     return flow;
35 }
36 int dinic(int s, int t) {
37     int flow = 0;
38     while(bfs(s, t)) {
39         memset(cur, 0, sizeof(cur));
40         flow += dfs(s, INF, t);
41     }
42 }
```

```

41     return flow;
42 }

```

## 4.5.2 最大流 ISAP

```

1  struct ISAP {
2      struct Edge {
3          int from, to, cap, flow;
4          bool operator < (const Edge& b) {
5              return from < b.from || (from == b.from && to < b
6                  .to);
7          }
8      };
9
10     int n, m, s, t;
11     vector<Edge> edges;
12     vector<int> g[MAXN]; // 邻接表, g[i][j]表示结点i
13                          // 的第j条边在e数组中的序号
14     bool vis[MAXN];      // BFS使用
15     int d[MAXN];         // 从起点到i的距离
16     int cur[MAXN];       // 当前弧指针
17     int p[MAXN];         // 可增广路上的上一条弧
18     int num[MAXN];       // 距离标号计数
19
20     void add_edge(int from, int to, int cap) {
21         edges.push_back((Edge){from, to, cap, 0});
22         edges.push_back((Edge){to, from, 0, 0});
23         m = edges.size();
24         g[from].push_back(m-2);
25         g[to].push_back(m-1);
26     }
27
28     bool BFS() {
29         memset(vis, 0, sizeof(vis));
30         queue<int> Q;
31         Q.push(t);
32         vis[t] = 1;
33         d[t] = 0;
34         while(!Q.empty()) {
35             int x = Q.front(); Q.pop();
36             for(int i = 0; i < g[x].size(); i++) {
37                 Edge& e = edges[g[x][i]^1];
38                 if(!vis[e.from] && e.cap > e.flow) {
39                     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].
55         flow = 0;
56 }
57
58 int Augment() {
59     int x = t, a = inf;
60     while(x != s) {
61         Edge& e = edges[p[x]];
62         a = min(a, e.cap-e.flow);
63         x = edges[p[x]].from;
64     }
65     x = t;
66     while(x != s) {
67         edges[p[x]].flow += a;
68         edges[p[x]^1].flow -= a;
69         x = edges[p[x]].from;
70     }
71     return a;
72 }
73
74 int max_flow(int s, int t, int need) {
75     this->s = s; this->t = t;
76     int flow = 0;
77     BFS();
78     memset(num, 0, sizeof(num));
79     for(int i = 0; i < n; i++) num[d[i]]++;
80     int x = s;
81     memset(cur, 0, sizeof(cur));
82     while(d[s] < n) {
83         if(x == t) {
84             flow += Augment();
85         }
86     }
87 }

```

```

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) {
91             // Advance
92             ok = 1;
93             p[e.to] = g[x][i];
94             cur[x] = i; // 注意
95             x = e.to;
96             break;
97         }
98     }
99     if(!ok) { // Retreat
100         int m = n-1; // 初值注意
101         for(int i = 0; i < g[x].size(); i++) {
102             Edge& e = edges[g[x][i]];
103             if(e.cap > e.flow) m = min(m, d[e.to]);
104         }
105         if(--num[d[x]] == 0) break;
106         num[d[x] = m+1]++;
107         cur[x] = 0; // 注意
108         if(x != s) x = edges[p[x]].from;
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.
119             push_back(i);
120     }
121     return ans;
122 }
123
124 void Reduce() {
125     for(int i = 0; i < edges.size(); i++) edges[i].
126         cap -= edges[i].flow;
127 }

```

```

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
131                 [i].to, edges[i].cap, edges[i].flow);
132     };

```

### 4.5.3 费用流

#### 4.5.4 无源无汇有容量下界网络的可行流

```

1  #include <bits/stdc++.h>
2  #define log(x) cout << #x << " = " << (x) << endl
3  #define mem(x, y) memset((x), (y), sizeof((x)))
4  #define rep(i, l, r) for (int (i) = (l); (i) <= (r); (i)
    ++)
5  using namespace std;
6  typedef long long ll;
7  /*---金牌---*/
8  const int N = 200 + 9;
9  const int INF = 0x3f3f3f3f;
10 const int MAXN = 10*N;
11 int low[MAXN*10];
12 int n, m;
13 struct Dinic
14 {
15     struct Edge
16     {
17         int from, to, cap, flow;
18     };
19     vector<Edge> edges;
20     vector<int> g[MAXN];
21     void init() {
22         mem(g, 0);
23         edges.clear();
24     }
25     void add_edge(int from, int to, int cap) {
26         edges.push_back((Edge){from, to, cap, 0});
27         edges.push_back((Edge){to, from, 0, 0});
28         int m = edges.size();
29         g[from].push_back(m - 2);

```

```

30     g[to].push_back(m - 1);
31 }
32 int s, t;
33 bool vis[MAXN];
34 int d[MAXN], cur[MAXN];
35 bool BFS()
36 {
37     memset(vis, 0, sizeof(vis));
38     queue<int> q;
39     q.push(s);
40     d[s] = 0;
41     vis[s] = true;
42     while (!q.empty()) {
43         int x = q.front(); q.pop();
44         for (int i = 0; i < g[x].size(); i++) {
45             Edge& e = edges[g[x][i]];
46             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,
63             e.cap - e.flow))) > 0) {
64             e.flow += f;
65             edges[g[x][i]^1].flow -= f;
66             flow += f;
67             a -= f;
68             if (a == 0) break;
69         }
70     }
71     return flow;
72 }
73 int max_flow(int ss, int tt)
74 {
75     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]
89             ][i].cap) {
90             puts("NO");
91             flag = false;
92         }
93     }
94     if (flag) {
95         puts("YES");
96         rep(i, 1, m) {
97             //log(low[i]);
98             printf("%d\n", edges[i*2+1].cap-edges[i
99                 *2+1].flow + low[i]);
100         }
101     }
102 };
103 void solve(Dinic &solver)
104 {
105     solver.init();
106     scanf("%d", &m);
107     int source = n + 1, sink = n + 2;
108     // sink -> source cap : inf
109     solver.add_edge(sink, source, INF);
110     int in[N];
111     mem(in, 0);
112     mem(low, 0);
113     rep(i, 1, m) {
114         int u, v, b, c;
115         scanf("%d%d%d%d", &u, &v, &b, &c);
116         low[i] = b; // 流量下界
117         in[u] -= b;
118         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 }

```

## 4.6 其他

### 4.6.1 层次遍历

每次记录队列当前的  $sz$  然后在一次循环中只出队  $sz$  次

### 4.6.2 图解序列

图解序列：一系列非负整数可以构成一个简单图的度序列

Havel 定理：对于  $n > 1$ , 长度为  $n$  的整数序列是图解序列当且仅当  $d'$  是图解序列,  $d'$  是删除  $d$  中最大元素  $\Delta$  并将紧跟的  $\Delta$  个元素减 1 后序列



## Chapter 5

# 计算几何

### 5.1 判断凸包

```
1  #include <cstdio>
2  #include <iostream>
3  using namespace std;
4  const int maxn = 1e8;
5  struct P
6  {
7      int x, y;
8  };
9  int ccw(P a, P b, P c)
10 {
11     return (b.x - a.x) * (c.y - a.y) - (c.x - a.x) * (b.y -
12         a.y);
13 }
14 P a[maxn];
15 int main()
16 {
17     for (;;)
18     {
19         int n;
20         cin >> n;
21         if (n == 0) break;
22         for (int i = 0; i < n; i++)
23             cin >> a[i].x >> a[i].y;
24         bool flag = true;
25         a[n++] = a[0];
26         a[n++] = a[1];
27         for (int i = 0; i < n - 3; i++)
```

```

27         if (ccw(a[i], a[i + 1], a[i + 2]) * ccw(a[i + 1], a
28             [i + 2], a[i + 3]) < 0)
29         {
30             flag = false;
31             break;
32         }
33     if (flag) puts("convex"); else puts("concave");
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
12         - a.y);
13 }
14 bool intersects(Segment a, Segment b)
15 {
16     if (ccw(a.p, a.q, b.p) * ccw(a.p, a.q, b.q) > 0)
17         return false;
18     if (ccw(b.p, b.q, a.p) * ccw(b.p, b.q, a.q) > 0)
19         return false;
20     return true;
21 }

```

## 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,
40                                         p.y);
41     double k = (p1.y - p2.y) / (p1.x - p2.x);
42     if (g_equal(k, 0)) return P(p.x, 2 * p1.y - p.y);
43     double x = (2*k*k*p1.x + 2*k*p.y - 2*k*p1.y - k*k*p.x +
44                 p.x) / (1 + k * k);
45     double y = p.y - (x - p.x) / k;
46     return P(x, y);
47 }
48 P intersection(P p1, P p2, P q1, P q2)
49 {
50     return p1 + (p2 - p1) * (det(q2 - q1, q1 - p1) / det(q2
51         - q1, p2 - p1));
52 }

```

## 5.4 终极模板

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  struct Point {
5      double x, y;
6      Point(double x = 0, double y = 0) : x(x), y(y) {}
7  };
8
9  typedef Point Vector;
10
11 Vector operator + (Vector A, Vector B) { return Vector(A.
    x + B.x, A.y + B.y); }
12 Vector operator - (Vector A, Vector B) { return Vector(A.
    x - B.x, A.y - B.y); }
13 Vector operator * (Vector A, double p) { return Vector(A.
    x*p, A.y*p); }
14 Vector operator / (Vector A, double p) { return Vector(A.
    x/p, A.y/p); }
15
16 bool operator < (const Point& a, const Point b) {
17     return a.x < b.x || (a.x == b.x && a.y < b.y);
18 }
19
20 const double EPS = 1e-10;
21
22 int dcmp(double x) {
23     if(fabs(x) < EPS) return 0;
24     else return x < 0 ? -1 : 1;
25 }
26
27 bool operator == (const Point& a, const Point& b) {
28     return dcmp(a.x-b.x) == 0 && dcmp(a.y-b.y);
29 }
30
31 //向量a的极角
32 double Angle(const Vector& v) {
33     return atan2(v.y, v.x);\share\CodeBlocks\templates\
    wizard\console\cpp
34 }
35
36 //向量点积
37 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  $p_0$  和方向向量  $v$  表示一条指向。直线上的所有点
     $P$  满足  $P = P_0 + t * v$ ;
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 getLineCircleInterseccion(Line L, Circle C, double& t1
169 , double& t2, vector<Point>& sol) {
170     double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L
171         .p.y - C.c.y;
172     double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*
173         d - C.r*C.r;
174     double delta = f*f - 4*e*g;
175     if(dcmp(delta) < 0) return 0; //相离
176     if(dcmp(delta) == 0) { //相切
177         t1 = t2 = -f / (2*e);
178         sol.push_back(C.point(t1));
179         return 1;
180     }
181     //相交
182     t1 = (-f - sqrt(delta)) / (2*e); sol.push_back(C.
183         point(t1));
184     t2 = (-f + sqrt(delta)) / (2*e); sol.push_back(C.
185         point(t2));
186     return 2;
187 }
188
189 //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 rdifff = A.r - B.r;
232     int rsum = A.r + B.r;
233     if(d2 < rdifff*rdifff) 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 == rdifff*rdifff) {          //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(n^2);
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]))
340             first++;
341         q[++last] = L[i];
342         if(fabs(Cross(q[last].v, q[last-1].v)) < EPS) {
343             last--;
344             if(Onleft(q[last], L[i].p)) q[last] = L[i];
345         }
346         if(first < last) p[last-1] = GetIntersection(q[
347             last-1], q[last]);
348     }
349     while(first < last && !Onleft(q[first], p[last-1]))
350         last--;
351     //删除无用平面
352     if(last-first <= 1) return 0;    //空集
353     p[last] = GetIntersection(q[last], q[first]);
354     //从 deque 复制到输出中
355     int m = 0;
356     for(int i = first; i <= last; i++) poly[m++] = p[i];
357     return m;
358 }
359 int main() {
360     return 0;
361 }

```

## 5.5 K 次圆

## Chapter 6

# 数据结构

### 6.1 手写堆

```
1 struct Node
2 {
3     int index;
4     int tag;
5     int l;
6     bool operator<(Node ano) const
7     {
8         return l!=ano.l?l<ano.l:index>ano.index;
9     }
10 };
11 Node heap[51111];
12 int heapsz;
13 int a[2<<21]; // 编号为i的在heap的位置
14 void swim(int p)
15 {
16     while (p > 1 && heap[p / 2] < heap[p])
17     {
18         swap(a[heap[p].tag], a[heap[p/2].tag]);
19         swap(heap[p], heap[p/2]);
20         p/=2;
21     }
22 }
23 void sink(int p)
24 {
25     while(p * 2 <= heapsz)
26     {
27         p *= 2;
28         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;

```

```

37         q[i].l=q[i].r=q[i].dis=0;
38     }
39     scanf("%d", &m);
40     rep(i, 1, m) {
41         scanf("%d%d", &x, &y);
42         l1=find(x), l2=find(y);
43         if(l1 == l2){printf("-1\n");continue;}
44         q[l1].x/=2;u=pop(l1);u=merge(u,l1);
45         q[l2].x/=2;v=pop(l2);v=merge(v,l2);
46         ans=merge(u,v);
47         printf("%d\n",q[ans].x);
48     }
49 }
50 int main()
51 {
52     while(~scanf("%d",&n)) slove();
53 }

```

## 6.3 两优先队列模拟堆

```

1
2 最简分数实数逼近
3 a1 = 100000, a2 = 1, ans = 100000;
4 ll xl = 0, yl = 1, xr = 1, yr = 0, xm, ym;
5 while(yl + yr <= 100000) {
6     xm = xl + xr;
7     ym = yl + yr;
8     if(check(xm, ym, a1, a2, kk)) {a1 = xm, a2 = ym;}
9     if(get(xm, ym, kk)) {xr = xm, yr = ym;}
10    else {xl = xm, yl = ym;}
11 }
12 struct heap {
13     priority_queue<int> p, q;
14     void init() {
15         while(!p.empty()) p.pop();
16         while(!q.empty()) q.pop();
17     }
18     void add(int x) {p.push(x);}
19     void del(int x) {q.push(x);}
20     int top() {
21         while(1) {
22             if(p.empty()) return -INF;
23             else if(!q.empty() && p.top() == q.top()) {p.
                pop(); q.pop();}

```



```

24         else return p.top();
25     }
26 }
27 int toptwo() {
28     int a = top(); del(a);
29     int b = top(); add(a);
30     if(b == -INF) return a == -INF ? a : 0;
31     else return max(a + b, 0);
32 }
33 }

```

## 6.4 线段树

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int MAXM = 1E6 + 9;
4  const int MAXR = 20 + 9;
5  const int INF = INT_MAX / 10;
6  struct Node
7  {
8      int setv, addv, sumv, minv, maxv;
9      Node()
10     {
11         setv = -1, addv = 0, sumv = 0, minv = 0, maxv =
12         0;
13     }
14 };
15 struct Tree
16 {
17     Node tree[4 * MAXM];
18     int mid(int l, int r)
19     {
20         return l + (r-l) / 2;
21     }
22     void pushdown(int o)
23     {
24         int lc(o*2), rc(o*2+1);
25         if (tree[o].setv >= 0) {
26             tree[lc].setv = tree[rc].setv = tree[o].setv;
27             tree[lc].addv = tree[rc].addv = 0;
28             tree[o].setv = -1;
29         }
30         if (tree[o].addv) {
31             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].
41             minv);
42         tree[o].maxv = max(tree[lc].maxv, tree[rc].
43             maxv);
44     }
45     if (tree[o].setv >= 0) {
46         tree[o].minv = tree[o].maxv = tree[o].setv;
47         tree[o].sumv = (r-l+1)*tree[o].setv;
48     }
49     if (tree[o].addv) {
50         tree[o].minv += tree[o].addv;
51         tree[o].maxv += tree[o].addv;
52         tree[o].sumv += (r-l+1) * tree[o].addv;
53     }
54 }
55 void update(int o, int l, int r, int op, int ql, int
56     qr, int v)
57 {
58     int lc(o*2), rc(o*2+1);
59     if (ql <= l && qr >= r) {
60         if (op == 1) tree[o].addv += v;
61         else {
62             tree[o].setv = v;
63             tree[o].addv = 0;
64         }
65     } else {
66         pushdown(o);
67         int m = mid(l, r);
68         if (ql <= m) update(lc, l, m, op, ql, qr, v);
69         else maintain(lc, l, m);
70         if (qr > m) update(rc, m+1, r, op, ql, qr, v)
71             ; else maintain(rc, m+1, r);
72     }
73     maintain(o, l, r);
74 }
75 Node query(int o, int l, int r, int ql, int qr)
76 {

```

```

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);
85         if (qr > m) rres = query(rc, m+1, r, ql, qr);
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", &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
104                (1, 1, c, op, y1, y2, v);
105        } else {
106            Node ans;
107            ans.sumv = 0, ans.minv = INF, ans.maxv = -INF;
108            for (int i(x1); i <= x2; i++) {
109                Node res(tree[i].query(1, 1, c, y1, y2));
110                ans.sumv += res.sumv;
111                ans.minv = min(ans.minv, res.minv);
112                ans.maxv = max(ans.maxv, res.maxv);
113            }
114        }
115    }
116 }

```

```

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 {
61     if(yl <= l && r <= yr) {
62         minv = min(minv, mn[xo][o]);
63         maxv = max(maxv, mx[xo][o]);
64         return;
65     }
66     int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
67     if(mid >= yl) queryy(xo, lch, l, mid, yl, yr);
68     if(mid < yr) queryy(xo, rch, mid + 1, r, yl, yr);
69 }
70 void queryx(int o, int l, int r, int xl, int xr, int yl,
71 int yr) {
72     if(xl <= l && r <= xr) {
73         queryy(o, 1, 1, n, yl, yr);
74         return;
75     }

```

```

73     }
74     int mid = (l + r) / 2, lch = o * 2, rch = lch + 1;
75     if(mid >= xl) queryx(lch, l, mid, xl, xr, yl, yr);
76     if(mid < xr) queryx(rch, mid + 1, r, xl, xr, yl, yr);
77 }
78 int main() {
79     int T; scanf("%d", &T);
80     int cas = 1;
81     while(T--) {
82         scanf("%d", &n);
83         for(int i = 1; i <= n; i++)
84             for(int j = 1; j <= n; j++)
85                 scanf("%d", &g[i][j]);
86         buildx(1, 1, n);
87         int q; scanf("%d", &q);
88         printf("Case_#%d:\n", cas++);
89         while(q--) {
90             int x, y, len; scanf("%d%d%d", &x, &y, &len);
91             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);
92             minv = 1 << 30, maxv = -minv;
93             queryx(1, 1, n, xl, xr, yl, yr);
94             int ans = (maxv + minv) / 2;
95             updatex(1, 1, n, x, y, ans);
96             printf("%d\n", ans);
97         }
98     }
99 }

```

## 6.6 Treap

```

1 struct node {
2     node* ch[2];
3     int w, sum, v, k;
4     void maintain() {sum = w + ch[0]->sum + ch[1]->
        sum;}
5     int cmp(int vv) {return vv == v ? -1 : vv > v;}
6     bool operator < (const node& rhs) {return k < rhs
        .k;}
7 }nd[MAXN * 10];
8 node* null, *rt;
9 int ncnt;
10 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             {
41                 int dd = o->ch[0] > o->ch[1];
42                 rot(o, dd);
43                 del(o->ch[dd], x);
44             }
45             else o = o->ch[0] == null ? o->ch[1] : o
46                 ->ch[0];
47         }
48     }
49     else del(o->ch[d], x);
50     o->maintain();
51 }
52 int getrk(node* o, int x) {
53     if(o == null) return 0;
54     int d = o->cmp(x);
55     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         ];}
26     if(r) ++cnt[r];
27     else {
28         r = ++ncnt;
29         sz[r] = cnt[r] = 1;
30         val[r] = x; fa[r] = y;
31         ch[r][0] = ch[r][1] = 0;
32         if(y) ch[y][val[y] < x] = r;
33     }
34     splay(r, 0);
35 }
36 void get(int v) {
37     int x = rt; if(!x) return;
38     while(ch[x][val[x] < v] && val[x] != v) x = ch[x][val
39         [x] < v];
40     splay(x, 0);
41 }
42 int getrk(int v) {
43     get(v);
44     return sz[ch[rt][0]];
45 }
46 int getkth(int x) {
47     int y = rt, p;
48     if(x > sz[rt]) return 0;
49     while(1) {
50         p = ch[y][0];
51         if(sz[p] + cnt[y] < x) {
52             x -= sz[p] + cnt[y];
53             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))
60         return rt;
61     int p = ch[rt][op];
62     while(ch[p][op^1]) p = ch[p][op^1];
63     return p;
64 }
65 void del(int v) {
66     int p = nxt(v, 0), s = nxt(v, 1);
67     splay(p, 0);
68     splay(s, p);
69     p = ch[s][0];
70     if(cnt[p] > 1) -- cnt[p], splay(p, 0);
71     else ch[s][0] = 0;
72 }

```

## 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
4             要有这个if
5             p[i][j] = p[p[i][j - 1]][j - 1];
6 }
7 int LCA(int u, int v) {
8     if(d[u] < d[v]) swap(u, v);
9     int lim;
10    //确定最大的 $2^{lim}$ 不超过 $d[u]$ 
11    for(lim = 0; (1 << lim) <= d[u]; lim++); lim--;
12    int ret = 0;
13    //把 $u$ 上升到 $v$ 相同的高度
14    for(int i = lim; i >= 0; i--) if(d[u] - (1 << i) >= d
15        [v]) u = p[u][i];
16    if(u == v) return u; //一定要有这个判断
17    for(int i = lim; i >= 0; i--) if(p[u][i] != p[v][i])
18        {u = p[u][i]; v = p[v][i];}
19    return p[u][0];
20 }

```

## 6.9 主席树

```
1 void build(node* &now, node* &pre, int l, int r, int x) {
2     now = &t[n[ncnt++]];
3     *now = *null;
4     int mid = (l + r) / 2;
5     if(l == r) {
6         *now = *pre;
7         now->val++;
8         return;
9     }
10    if(x <= sq[mid]) {
11        build(now->ch[0], pre->ch[0], l, mid, x);
12        now->ch[1] = pre->ch[1];
13        now->maintain();
14    }
15    else {
16        build(now->ch[1], pre->ch[1], mid + 1, r, x);
17        now->ch[0] = pre->ch[0];
18        now->maintain();
19    }
20 }
```

## 6.10 树剖

```
1 void dfs1(int u, int fa, int dep) {
2     sz[u] = 1; d[u] = dep; ch[u] = 0; p[u] = fa;
3     for(int i = head[u]; i != -1; i = ed[i].next) {
4         int v = ed[i].v;
5         if(v == fa) continue;
6         dfs1(v, u, dep + 1);
7         sz[u] += sz[v];
8         if(sz[v] > sz[ch[u]]) ch[u] = v;
9     }
10 }
11 void dfs2(int u, int rt) {
12     idx[u] = id++;
13     top[u] = rt;
14     if(ch[u]) dfs2(ch[u], rt);
15     for(int i = head[u]; i != -1; i = ed[i].next) {
16         int v = ed[i].v;
17         if(v == p[u] || v == ch[u]) continue;
18         dfs2(v, v);
19     }
20 }
```

```

20 }
21 int ask(int u, int v) {
22     int ret = 0;
23     while(top[u] != top[v]) {
24         //一定是  $top[u]$  的深度大于等于  $top[v]$ 
25         if(d[top[u]] < d[top[v]]) swap(u, v);
26         ret = max(ret, query(1, 1, n, idx[top[u]], idx[u]
27                               ));
28         u = p[top[u]];
29     }
30     if(d[u] < d[v]) swap(u, v);
31     if(u != v) ret = max(ret, query(1, 1, n, idx[ch[v]],
32                                     idx[u]));
33     return ret;
34 }

```

## 6.11 点分治

```

1 void getsz(int u, int fa) {
2     sz[u] = 1; f[u] = 0;
3     for(int i = head[u]; i != -1; i = ed[i].next) {
4         int v = ed[i].v;
5         if(v == fa || vis[v]) continue;
6         getsz(v, u);
7         sz[u] += sz[v];
8         f[u] = max(f[u], sz[v]);
9     }
10 }
11 //找到最大子树最小的点作为分治的中心
12 void getrt(int r, int u, int fa) {
13     //用父边所连的子树更新  $f$ 
14     f[u] = max(f[u], sz[r] - sz[u]);
15     if(f[u] < minf) {minf = f[u]; rt = u;}
16     for(int i = head[u]; i != -1; i = ed[i].next) {
17         int v = ed[i].v;
18         if(v == fa || vis[v]) continue;
19         getrt(r, v, u);
20     }
21 }
22 int solve(int u) {
23     minf = n;
24     getsz(u, 0);
25     getrt(u, u, 0);
26     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!

```

```

21         Q[i].CURRENT = Q[i].CURRENT + ANS[i]
22         向数组Q2末尾添加Q[i]
23     end if
24 end for
25 //分治,递归处理
26 Divide_Conquer(Q1, AL, AM)
27 Divide_Conquer(Q2, AM+1, AR)
28 以上别人的伪代码 非常清楚
29
30 带修改的整体二分
31 void cal(int ql, int qr, int l, int mid) {
32     for(int i = ql; i <= qr; i++) {
33         if(nd[i].k > nd[i].cnt = query(nd[i].r) - query(nd
34             [i].l - 1);
35         else if(nd[i].r <= mid) update(nd[i].l, nd[i].cnt
36             );
37     }
38     for(int i = ql; i <= qr; i++) if(nd[i].r <= mid && (!
39         nd[i].k)) update(nd[i].l, -nd[i].cnt);
40 }
41 void divide(int ql, int qr, int l, int r) {
42     if(l == r) {
43         for(int i = ql; i <= qr; i++) if(nd[i].k) ans[nd[
44             i].id] = l;
45         return;
46     }
47     int mid = (l + r) / 2;
48     cal(ql, qr, l, mid);
49     int p1 = 0, p2 = 0;
50     for(int i = ql; i <= qr; i++) {
51         if(nd[i].k) {
52             if(nd[i].cnt >= nd[i].k) t1[++p1] = nd[i];
53             else {
54                 nd[i].k -= nd[i].cnt;
55                 t2[++p2] = nd[i];
56             }
57         }
58         else {
59             if(nd[i].r <= mid) t1[++p1] = nd[i];
60             else t2[++p2] = nd[i];
61         }
62     }
63     for(int i = 1; i <= p1; i++) nd[ql - 1 + i] = t1[i];
64     for(int i = 1; i <= p2; i++) nd[ql + p1 - 1 + i] = t2
65         [i];
66     if(p1) divide(ql, ql + p1 - 1, l, mid);

```

```

62         if(p2) divide(ql + p1, qr, mid + 1, r);
63     }

```

## 6.14 莫队

```

1  struct qnode {
2      int l, r, id;
3      ll a, b;
4      bool operator < (const qnode& rhs) const {
5          return dv[l] == dv[rhs.l] ? dv[r] < dv[rhs.r] :
              dv[l] < dv[rhs.l];
6      }
7  }qn[MAXN];
8  int bd = (int)sqrt(n);
9  for(int i = 1; i <=n; i++) {
10     dv[i] = (i - 1) / bd + 1;
11 }

```

## 6.15 KDtree

给  $N$  个  $K$  维点, 找这  $K$  维点里面最近的  $M$  个点

```

1  #include <bits/stdc++.h>
2  using namespace std;
3  const int N = 50080;
4  const int K = 5;
5  typedef pair<int, int> Pair;
6  struct Node
7  {
8      int x[K], d;
9      bool f;
10     static int k, cd;
11     void read() {
12         for (int i = 0; i < k; i++) {
13             cin >> x[i];
14         }
15         f = 0;
16     }
17     bool operator < (Node node) const {
18         return x[cd] < node.x[cd];
19     }
20 };
21 int Node::k = 0, Node::cd = 0;
22 int mid(int l, int r)

```

```

23 {
24     return l + (r - 1) / 2;
25 }
26 int dist(Node a, Node b) //算k维的距离
27 {
28     int res = 0;
29     for(int i = 0; i < Node::k; i++)
30         res += (a.x[i] - b.x[i])*(a.x[i] - b.x[i]);
31     return res;
32 }
33 Node a[N];
34 priority_queue<Pair> pq; // 距离 下标
35 void build(int l, int r, int d)
36 {
37     if (l > r) return;
38     int m = mid(l, r);
39     Node::cd = d;
40     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
41     //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
42     a[m].d = d; //这个点的维度是d
43     if(l == r) //到了叶节点了
44     {
45         a[m].f = true; //应是不是叶节点的意思
46         return;
47     }
48     build(l, m-1, (d+1)%Node::k); //递归建树
49     build(m+1, r, (d+1)%Node::k);
50 }
51 int num;
52 void fd(int l, int r, Node tar)
53 {
54     if (l > r) return;
55     int m = mid(l, r);
56     int d = dist(a[m], tar);
57     if(a[m].f) { //如果是叶子
58         if(pq.size() < num) { // 还每找满m个
59             pq.push(make_pair(d, m));
60         } 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
61             pq.pop();
62             pq.push(make_pair(d, m));
63         }
64         return;
65     }

```



```

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 }

```

## Chapter 7

# 字符串

### 7.1 最小表示法

```
1 #include <cstdio>
2 #include <algorithm>
3 #include <iostream>
4 using namespace std;
5 const int maxn = 100010;
6 int main()
7 {
8     int n;
9     while (scanf("%d", &n) != EOF)
10    {
11        string s[maxn];
12        for (int ii = 0; ii < n; ii++)
13        {
14            string ss;
15            cin >> ss;
16            ss = ss + ss;
17            bool flag = false;
18            int i = 0, j = 1, k = 0, l = ss.size() / 2, p = 0;
19            while (i < l && j < l)
20            {
21                k = 0;
22                while (ss[i + k] == ss[j + k] && k < l) k++;
23                if (k == l)
24                {
25                    p = i;
26                    flag = true;
27                    break;
28                }
```

```

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, 1);
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);

```

```

2 int n = 1, pre = 0, ans = 0;
3 ss[0] = '$';
4 for(int i = 0; i < len; i++) {
5     ss[n++] = '#';
6     ss[n++] = s[i];
7 }
8 ss[n] = '#';
9 for(int i = 1; i < n; i++) {
10 //pre是在i之前半径延伸地最远的点
11 //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
12 延伸的较小值
13 if(i < pre + p[pre]) p[i] = min(p[pre - (i - pre)], p[
    pre] + pre - i);
14 else p[i] = 1;
15 //继续扩展i的回文区间
16 while(ss[i - p[i]] == ss[i + p[i]]) ++p[i];
17 //用i更新pre
18 if(pre + p[pre] < i + p[i]) pre = i;
19 }

```

## 7.4 AC 自动机

```

1 void getfail() {
2     queue<int> q;
3     for(int i = 0; i < 26; i++) if(ch[0][i]) q.push(ch
4         [0][i]);
5     while(!q.empty()) {
6         int rt = q.front(); q.pop();
7         for(int i = 0; i < 26; i++) {
8             int u = ch[rt][i];
9             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
10            q.push(u);
11            int v = fail[rt];
12            while(v && !ch[v][i]) v = fail[v];
13            fail[u] = ch[v][i];
14            //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
15            last[u] = val[fail[u]] ? fail[u] : last[fail[
                u]];
16        }

```

```

17     }
18 }
19 void print(int j) {
20     if(j) {
21         printf("%d:_%d", j, val[j]);
22         print(last[j]);
23     }
24 }
25 void findp(char * T) {
26     int len = strlen(T);
27     int j = 0;
28     for(int i = 0; i < n; i++) {
29         int x = T[i] - 'a';
30         while(j && !ch[j][x]) j = fail[j];
31         j = ch[j][x];
32         if(val[j]) print(j);
33         else if(last[j]) print(last[j]);
34     }
35 }

```

## 7.5 后缀数组

```

1  int sa[MAXN], rk[MAXN], ht[MAXN], cnt[MAXN], t1[MAXN], t2
    [MAXN];
2  void getsa(int m) {
3      int *x = t1, *y = t2;
4      for(int i = 0; i < m; i++) cnt[i] = 0;
5      for(int i = 0; i < n; i++) cnt[x[i]] = s[i]++;
6      for(int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
7      for(int i = n - 1; i >= 0; i--) sa[--cnt[x[i]]] = i;
8      //sa数组从0到n-1
9      for(int k = 1; ; k *= 2) {
10         int p = 0;
11         //2nd
12         for(int i = n - k; i < n; i++) y[p++] = i;
13         for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++]
            = sa[i] - k;
14         //1st
15         for(int i = 0; i < m; i++) cnt[i] = 0;
16         for(int i = 0; i < n; i++) cnt[x[y[i]]]++;
17         for(int i = 1; i < m; i++) cnt[i] += cnt[i - 1];
18         for(int i = n - 1; i >= 0; i--) sa[--cnt[x[y[i]
            ]]] = y[i];
19         swap(x, y);

```

```

20     p = 1; x[sa[0]] = 0;
21     for(int i = 1; i < n; i++)
22         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++;
23     if(p >= n) break;
24     m = p;
25 }
26 }
27 void getheight() {
28     int k = 0;
29     for(int i = 0; i < n; i++) rk[sa[i]] = i;
30     for(int i = 0; i < n; i++) {
31         if(rk[i] == 0) continue;
32         if(k) k--;
33         int j = sa[rk[i] - 1];
34         while(s[i + k] == s[j + k] && i + k < n && j + k
                < n) k++;
35         ht[rk[i]] = k;
36     }
37 }

```

## 7.6 后缀自动机

```

1 //rt = 1
2 int ncnt, last, ch[MAXN][26], val[MAXN], par[MAXN];
3 int c[MAXN], rk[MAXN];
4 void init(int x, int v) {
5     memset(ch[x], 0, sizeof(ch[x]));
6     par[x] = 0; val[x] = v;
7 }
8 void add(int x) {
9     int p = last, np = ++ncnt;
10    memset(ch[np], 0, sizeof(ch[np]));
11    init(np, val[p] + 1);
12    while(p && !ch[p][x]) {
13        ch[p][x] = np;
14        p = par[p];
15    }
16    if(p == 0) par[np] = 1;
17    else {
18        int q = ch[p][x];
19        if(val[p] + 1 == val[q]) par[np] = q;
20        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 }

```



## Chapter 8

# 其他

### 8.1 蔡勒公式

```
1 int calc(int y,int m,int d)
2 {
3     if (m < 3) m += 12, y--;
4     int w = (d + 2 * m + 3 * (m + 1) / 5 + y + y / 4
5         - y / 100 + y / 400) % 7;
6     return w;
7 }
8 bool isRun(int YYYY)
9 {
10     return (YYYY % 4 == 0 && YYYY % 100 != 0) || (YYYY %
11         400 == 0);
12 }
```

### 8.2 斜率 DP

### 8.3 最长子序列

```
1 bool solve()
2 {
3     int n, k;
4     scanf("%d%d", &n, &k);
5     int a[N];
6     int dp[N];
7     int g[N];
```

```

8     rep(i, 1, n) scanf("%d", &a[i]), g[i] = inf;
9     int ans1 = 0;
10    rep(i, 1, n) {
11        int k = upper_bound(g+1, g+1+n, a[i]) - g;
12        dp[i] = k;
13        g[k] = a[i];
14        ans1 = max(ans1, k);
15    }

```

## 8.4 四边形不等式

```

1  for(int i=2;i<=m+1;++i)
2  {
3      s[i][n+1]=n;
4      for(int j=n;j>i;--j)
5      {
6          for(int k=s[i-1][j];k<=s[i][j+1];++k)
7          {
8              LL tmp=dp[i-1][k]+w[k+1][j];
9              // cout<<i<<' '<<j<<' '<<k<<' '<<tmp<<endl;
10             if(tmp<dp[i][j])
11             {
12                 dp[i][j]=tmp;
13                 s[i][j]=k;
14             }
15         }
16     }
17 }

```

## 8.5 数位 DP

```

1  //基本上都是这个模板，但是有点慢
2  int dp(int pos, int sum, int lz, int lim) {
3      if(!lim && !lz && d[pos][sum] != -1) return d[pos][sum];
4      if(pos == 0) return sum >= 33;
5      int bd = lim ? s[pos] : 1;
6      int res = 0;
7      for(int i = 0; i <= bd; i++) {
8          if(i == 0) {
9              if(lz) res += dp(pos - 1, sum, lz, lim && i == bd);

```

```

10         else res += dp(pos - 1, sum + 1, 0, lim && i
11             == bd);
12     }
13     else res += dp(pos - 1, sum - 1, 0, lim && i ==
14         bd);
15 }
16 if(!lim && !lz) d[pos][sum] = res;
17 return res;
18 }
19 int solve(int x) {
20     int cnt = 0;
21     while(x) {
22         if(x & 1) s[++cnt] = 1;
23         else s[++cnt] = 0;
24         x /= 2;
25     }
26     return dp(cnt, 33, 1, 1);
27 }

```

## 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
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;

```

```

56 }
57 BigNum::BigNum(const int b)          //将一个int类型的变量转化
    为大数
58 {
59     int c,d = b;
60     len = 0;
61     memset(a,0,sizeof(a));
62     while(d > MAXN)
63     {
64         c = d - (d / (MAXN + 1)) * (MAXN + 1);
65         d = d / (MAXN + 1);
66         a[len++] = c;
67     }
68     a[len++] = d;
69 }
70 BigNum::BigNum(const char*s)        //将一个字符串类型的变量
    转化为大数
71 {
72     int t,k,index,l,i;
73     memset(a,0,sizeof(a));
74     l=strlen(s);
75     len=l/DLEN;
76     if(l%DLEN)
77         len++;
78     index=0;
79     for(i=l-1;i>=0;i-=DLEN)
80     {
81         t=0;
82         k=i-DLEN+1;
83         if(k<0)
84             k=0;
85         for(int j=k;j<=i;j++)
86             t=t*10+s[j]-'0';
87         a[index++]=t;
88     }
89 }
90 BigNum::BigNum(const BigNum & T) : len(T.len) //拷贝构造
    函数
91 {
92     int i;
93     memset(a,0,sizeof(a));
94     for(i = 0 ; i < len ; i++)
95         a[i] = T.a[i];
96 }

```

```

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 BigInt BigInt::operator+(const BigInt & T) const //两个
    大数之间的相加运算
142 {
143     BigInt 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 BigInt BigInt::operator-(const BigInt & T) const //两个
    大数之间的相减运算
162 {
163     int i, j, big;
164     bool flag;
165     BigInt 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 }
```

# Bibliography

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