ACM-template

Weiers&IronHead

October 25, 2018

Contents

1 头文件

```
#include<bits/stdc++.h>
#define pb(x) push_back(x)
#define fir first
#define sec second
#define mem(a,x) memset(a,x,sizeof(a))
#define mkr make_pair
typedef long long ll;
using namespace std;
const int inf=0x3f3f3f3f;
const ll INF= 0x3f3f3f3f3f3f3f3f;
const double pi = acos(-1.0);
ll gcd(ll a,ll b) { return b?gcd(b,a%b):a;}
/*
 #ifndef ONLINE_JUDGE
    freopen("data.in","r",stdin);
     freopen("data.out","w",stdout);
    #endif
*/
int main(){
 //ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);

 return 0;
  }

/*
#include<cstdio>
#include<iostream>
#include<queue>
#include<stack>
#include<cmath>
#include<cstring>
#include<string>
#include<set>
#include<map>
#include<sstream>
#include<algorithm>
*/
```

2 图论

2.1 最短路

```
/* Dijkstra POJ - 2387*/
/* 复杂度 O (E log E) */
const int maxn = " ";
typedef pair<int,int> P;
int d[maxn];
int vis[maxn];
int n;
struct node{
        int to;int cost;
};
vector<node>g[maxn];
void dij(int s){
        memset(d,0x3f,sizeof(d));
        memset(vis,0,sizeof(vis));
    d[s]=0;
        priority_queue< P,vector<P>,greater<P> >que;
        que.push(P(0,1));
        while(!que.empty()){
       P a=que.top();que.pop();
       int u=a.second;
       if(vis[u])  continue;
        vis[u]=1; //or if(d[u]<a.first) continue; optimize in cf 938d
                   for(int j=0;j<g[u].size();j++){
                   node e=g[u][j];
                  if(!vis[e.to]&&d[e.to]>e.cost+d[u]) {
                     d[e.to]=e.cost+d[u];
             que.push(P(d[e.to],e.to));
              }
        }}
}

/****SPFA****/
/*若存在负环回路则返回1*/

const int maxn = " ";
struct node{
  int to,cost;
  node(int To,int Cost):to(To),cost(Cost){};
};
vector<node>edge[maxn];
int d[maxn];
int vis[maxn];//标记每个点是否在队列里
int cnt[maxn];//判断是否存在负环回路;若有点更新超过n次，则存在负环
int spfa(int x)
{
  memset(d,0x3f,sizeof(d));
  memset(vis,0,sizeof(vis));
  memset(cnt,0,sizeof(cnt));
  d[x]=0;
  queue<int>que;
  que.push(x);
  vis[x]=1;cnt[x]=1;
```

```
  while(!que.empty())
  {
    int t=que.front();
    que.pop();
    vis[t]=0;
    for(int i=0;i<edge[t].size;i++){
        node e=edge[t][i];
        if(d[e.to]>d[t]+e.cost){
            d[e.to]=d[t]+e.cost;
          if(!vis[e.to]){
            que.push(e.to);
            vis[e.to]=1;
            if(++cnt[e.to]>n) return 1;
         }}
      }
  }
 return 0;
 }

/****floyd****/
for(int k=1;k<=n;k++)
    for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
        d[i][j]=min(d[i][j],d[i][k]+d[k][j]);

/*差分约束系统*/
/*
根据最短路的性质
对于任何一条边u -> v，都有
d(v) <= d(u) + w(u, v)
*/
```

2.2 最小生成树

```
/****最小生成树Kruskal poj2421****/
/*时间复杂度取决于边数 o(E logE)*/
const int maxn=" ";
struct node{
  int u,v;ll w;
  bool operator<(const node &b)const{
      return w<b.w;
  }
};
vector<node>edge[maxn];
int p[maxn];
void init(int n)
{
    for(int i=0; i<=n; i++)
        p[i]=i;
}
int Find(int x)
{
    if(x==p[x])
        return p[x];
    int y=Find(p[x]);
    return p[x]=y;
```

```
}

int Union(int x,int y)
{
    int x1=Find(x);
    int y1=Find(y);
    if(x1==y1)
        return 0;
    p[x1]=y1;
      return 1;
}

void kruskal(int n)
{
  ll sum=0;
  int num=0;//已选用的边的数目
  init(n);
  sort(edge.begin(),edge.end());
  for(int i=0;i<edge.size();i++)
  {
    int u=edge[i].u;
    int v=edge[i].v;
    ll w=edge[i].w;
    if(Union(u,v))
      {
        num++;sum+=w;
      }
     if(num==n-1) break;
   }
}

/*****最小生成树Prim *****/
/*时间复杂度取决于顶点数 O(V^2)*/
const int maxn = " ";
int cost[maxn][maxn];
int lowc[maxn];
int prim(int n){
  for(int i=2;i<=n;i++)
    lowc[i]=cost[0][i];
  lowc[1]=-1;//tag visited
  int sum=0;
  for(int i=1;i<n;i++){
      int min=inf;
      int u;
     for(int j=1;j<=n;j++){
        if(lowc[j]!=-1&&lowc[j]<min){
            min=lowc[j];u=j;
        }
       }
    if(min==inf) return -1;
    lowc[u]=-1;
    sum+=min;

  for(int j=1;j<=n;j++){
    if(lowc[j]!=-1&&lowc[j]>cost[u][j])
        lowc[j]=cost[u][j];
```

```
   }
  }
  return sum;
}
```

2.3 二分图

```
/****二分图最大匹配 匈牙利算法****/
/*o(VE)*/
int line[maxn][maxn];
int used[maxn];//标记这条边有没有用过
int match[maxn];//标记右侧的点是否被匹配，以及匹配的是左侧哪个点
int nl;
int nr;
bool find(int x){
  for(int i=1;i<=nr;i++){
    if(line[x][i]&&!used[i]){
        used[i]=1;
        if(match[i]==0||find(match[i])){
            match[i]=x;
            return true;
        }
    }
  }
  return false;
}
int hungarian()
{
    int ans = 0;
    memset(match,0,sizeof(match));
    for (int i=1;i<=nl;i++) {
       memset(used,0,sizeof(used));
       if(find(i)) ans++;
    }
    return ans;
}


/* * 二分图匹配（Hopcroft-Carp算法）
  复杂度O(sqrt(n)*E)  邻接表存图 vector实现
  vector先初始化，然后假如边uN 为左端的顶点数，使用前赋值(点编号0开始)
  */
const int MAXN = 3000;
vector<int>G[MAXN];
int uN;
int Mx[MAXN],My[MAXN];
int dx[MAXN],dy[MAXN];
int dis;
bool used[MAXN];
bool SearchP() {
 queue<int>Q;
  dis = inf;
  memset(dx,-1,sizeof(dx));
  memset(dy,-1,sizeof(dy));
  for(int i = 0 ; i < uN; i++)
     if(Mx[i] == -1)  {
```

```
          Q.push(i);
         dx[i] = 0;  }
       while(!Q.empty()) {
           int u = Q.front();
           Q.pop();
       if(dx[u] > dis)
         break;
       int sz = G[u].size();
       for(int i = 0;i < sz;i++) {
             int v = G[u][i];
        if(dy[v] == -1)   {
             dy[v] = dx[u] + 1;
     if(My[v] == -1)dis = dy[v];
     else    {
        dx[My[v]] = dy[v] + 1;
        Q.push(My[v]);
         } } } }

return dis != inf;
}

 bool DFS(int u) {
     int sz = G[u].size();
     for(int i = 0;i < sz;i++) {
             int v = G[u][i];
       if(!used[v] && dy[v] == dx[u] + 1)  {
             used[v] = true;
       if(My[v] != -1 && dy[v] == dis)
          continue;
        if(My[v] == -1 || DFS(My[v]))  {
                 My[v] = u;
                Mx[u] = v;
      return true;
       }   } }
return false; }


int MaxMatch() {
    int res = 0;
    memset(Mx,-1,sizeof(Mx));
     memset(My,-1,sizeof(My));
     while(SearchP()) {
        memset(used,false,sizeof(used));
        for(int i = 0;i < uN;i++)
        if(Mx[i] == -1 && DFS(i))
          res++;
          }
return res; }


/*KM算法*/
/**二分图最佳完美匹配**/
//求权值和最大的完美匹配
//完美匹配：所有点都是匹配点
/*steal from csl*/
const int maxn = " ";
```

```
int n;
int cost[maxn][maxn];
int lx[maxn], ly[maxn], match[maxn], slack[maxn];
int prev[maxn];
bool vy[maxn];

void augment(int root)
{
    fill(vy + 1, vy + n + 1, false);
    fill(slack + 1, slack + n + 1, inf);
    int py;
    match[py = 0] = root;
    do
    {
        vy[py] = true;
        int x = match[py], yy;
        int delta = inf;
        for (int y = 1; y <= n; y++)
        {
            if (!vy[y])
            {
                if (lx[x] + ly[y] - cost[x][y] < slack[y])
                    slack[y] = lx[x] + ly[y] - cost[x][y], prev[y] = py;
                if (slack[y] < delta) delta = slack[y], yy = y;
            }
        }
        for (int y = 0; y <= n; y++)
        {
            if (vy[y])
                lx[match[y]] -= delta, ly[y] += delta;
            else
                slack[y] -= delta;
        }
        py = yy;
    } while (match[py] != -1);
    do
    {
        int pre = prev[py];
        match[py] = match[pre], py = pre;
    } while (py);
}
int KM()
{
    for (int i = 1; i <= n; i++)
    {
        lx[i] = ly[i] = 0;
        match[i] = -1;
        for (int j = 1; j <= n; j++) lx[i] = max(lx[i], cost[i][j]);
    }
    int answer = 0;
    for (int root = 1; root <= n; root++) augment(root);
    for (int i = 1; i <= n; i++) answer += lx[i], answer += ly[i];
    return answer;
}
```

2.4 2-SAT

```
/**2-SAT**/
vector<int> g[maxn*2];
bool mark[maxn*2];
int s[maxn*2],c;
bool dfs(int x){
    if(mark[x^1]) return false;
    if(mark[x]) return true;
    mark[x]=true;
    s[c++]=x;
    for(int i=0;i<g[x].size();i++)
        if(!dfs(g[x][i])) return false;
    return true;
}
void init(int n){
    for(int i=0;i<n*2;i++) g[i].clear();
    memset(mark,0,sizeof(mark));
}
void add_clause(int x,int xval,int y,int yval){
//x的xval状态与y的yval状态冲突
    x=x*2+xval;
    y=y*2+yval;
    g[x^1].push_back(y); //选了x^1就必须选y，连边表示 推导出
    g[y^1].push_back(x);
}
bool solve(int n){
    for(int i=0;i<n*2;i+=2)
        if(!mark[i]&&!mark[i+1]){
            c=0;
            if(!dfs(i)){
                while(c>0) mark[s[--c]]=false;
                if(!dfs(i+1)) return false;
            }
        }
    return true;
}
```

2.5 割点与强连通

```
/**求割点*/
/*根节点的儿子数量>=2即为割点*/
vector<int>edge[maxn];
int low[maxn],dfn[maxn],tot;
bool iscut[maxn];//判断是不是割点
void init(){
    for(int i = 1; i <= n; i++)
        edge[i].clear();
        mem(low);
        mem(dfn);
        mem(iscut);
        tot = 0;
}
void dfs(int u,int fa){
    low[u] = dfn[u] = ++tot;
    int child=0;
```

```
    for(int i = 0; i < edge[u].size(); i++){
        int v = edge[u][i];
        if(!dfn[v]){
            dfs(v,u);
            child++;
            low[u] = min(low[u],low[v]);
            if(low[v] >= dfn[u])
                iscut[u] = true;
        }
        else if(v != fa){
            low[u] = min(low[u],dfn[v]);
        }
    }
    if(fa<0&&child == 1) iscut[u] = 0;//根节点
}



/*强连通分量*/
/* Tarjan算法  * 复杂度O(N+M)*/
/*边双连通分量加一个!=pre即可*/
vector<int>edge[maxn];
stack<int>st;
int low[maxn],dfn[maxn];
int instack[maxn];
int tot;
int scc;//强连通分量个数
int belong[maxn];//记录每个点属于哪个连通分量
void init(){
 mem(dfn,0);
 mem(low,0);
 mem(instack,0);
 for(int i=0;i<maxn;i++)
    edge[i].clear();
 while(!st.empty())
    st.pop();
 tot=scc=0;
}

void tar(int u){
  dfn[u]=low[u]=++tot;
  st.push(u);
  instack[u]=1;
  for(int i=0;i<edge[u].size();i++){
     int v=edge[u][i];
     if(!dfn[v]){
        tar(v);
        if(low[u]>low[v])
            low[u]=low[v];
        }
     else if(instack[v]&&low[u]>dfn[v])
        low[u]=dfn[v];
        }
     if(low[u]==dfn[u]) {
        int v;
         scc++;
```

```
        do{
           v=st.top();
           st.pop();
           belong[v]=scc;
           instack[v]=0;
          }while(v!=u);
      }
     }
```

2.6 最大流

```
/***Dinic算法 HDU - 1532****/
/*复杂度o(n^2*m)*/
struct E{
 int to,cap;
 int rev;//反向边的序号
};
vector<E>edge[maxn];
int level[maxn];

void addedge(int u,int v,int cap){
 edge[u].push_back((E){v,cap,edge[v].size()});
 edge[v].push_back((E){u,0,edge[u].size()-1});
}
void bfs(int s){
  memset(level,-1,sizeof(level));
  level[s]=0;
  queue<int>que;
  que.push(s);
  while(!que.empty()){
    int u=que.front();que.pop();
  for(int i=0;i<edge[u].size();i++){
    E e=edge[u][i];
    if(e.cap>0&&level[e.to]==-1){
      level[e.to]=level[u]+1;
      que.push(e.to);}
  }
}
}
int dfs(int u,int t,int f){
  if(u==t) return f;
  for(int i=0;i<edge[u].size();i++){
    E e=edge[u][i];
    if(level[e.to]==level[u]+1&&e.cap>0){
        int d=dfs(e.to,t,min(f,e.cap));
        if(d>0){
            edge[u][i].cap-=d;
            edge[e.to][e.rev].cap+=d;
            return d;
        }
    }
  }
  return 0;
}

int max_flow(int s,int t){
```

```
   int flow=0;
 while(1){
    bfs(s);
    if(level[t]<0) return flow;
    int f;
    while((f=dfs(s,t,inf))>0)
        flow+=f;
    }
    return flow;
 }


/*ISAP*/
const int maxn= " ";
struct Edge{
        int from,to,cap,flow;
        Edge(int u,int v,int c,int f):from(u),to(v),cap(c),flow(f){}
};
struct ISAP{
        int n,m,s,t;
        vector<Edge> edges;
        vector<int > G[maxn];
        bool vis[maxn];
        int d[maxn],cur[maxn];
        int p[maxn],num[maxn];
        void init(int n)
        {
                this ->n=n;
                for(int i=0;i<=n;i++)
                         G[i].clear();
                edges.clear();
                memset(d,0x3f3f3f3f,sizeof(d));
        }
        void addedge(int from,int to,int cap)
        {
                edges.push_back(Edge(from,to,cap,0));
                edges.push_back(Edge(to,from,0,0));
                m=edges.size();
                G[from].push_back(m-2);
                G[to].push_back(m-1);
        }
        bool BFS()
        {
                memset(vis,false,sizeof(vis));
                queue<int >q;
                q.push(t);
                d[t]=0;
                vis[t]=true;
                while(!q.empty())
                {
                        int u=q.front();
                        q.pop();
                        for(int i=0;i<G[u].size();i++)
                        {
                                Edge &e=edges[G[u][i]^1];
                                if(!vis[e.from]&&e.cap>e.flow)
```

```
                        {
                                vis[e.from]=true;
                                d[e.from]=d[u]+1;
                                q.push(e.from);
                        }
                }
        }
        return vis[s];
}
int Augment()
{
        int flow=inf;
        for(int u=t;u!=s;u=edges[p[u]].from)
        {
                Edge &e=edges[p[u]];
                flow=min(flow,e.cap-e.flow);
        }
        for(int u=t;u!=s;u=edges[p[u]].from)
        {
                edges[p[u]].flow+=flow;
                edges[p[u]^1].flow-=flow;
        }
        return flow;
}
int Maxflow(int s,int t)//cal
{
        this->s=s;
        this ->t=t;
        int flow=0;
        BFS();
        if(d[s]>=n)
                return 0;
        memset(num,0,sizeof(num));
        for(int i=0;i<n;i++)
                if(d[i]<INF)
                        num[d[i]]++;
        int u=s;
        memset(cur,0,sizeof(cur));
        while(d[s]<n)
        {
                if(u==t)
                {
                        flow+=Augment();
                        u=s;
                }
                int ok=0;
                for(int i=cur[u];i<G[u].size();i++)
                {
                        Edge &e=edges[G[u][i]];
                        if(e.cap>e.flow && d[u]==d[e.to]+1)
                        {
                                ok=1;
                                p[e.to]=G[u][i];
                                cur[u]=i;
                                u=e.to;
                                break;
```

```
                        }
                    }
                    if(!ok)
                    {
                        int m=n-1;
                        for(int i=0;i<G[u].size();i++)
                        {
                            Edge &e=edges[G[u][i]];
                            if(e.cap>e.flow)
                                m=min(m,d[e.to]);
                        }
                        if(--num[d[u]]==0)
                            break;
                        ++num[d[u]=m+1];
                        cur[u]=0;
                        if(u!=s)
                            u=edges[p[u]].from;
                    }
                }
                return flow;
        }
};
```

2.7 最小费用最大流

```
/*费用流*/

const int maxn = " ";
struct Edge
{
    int from, to, cap, flow, cost;
    Edge(int u, int v, int c, int f, int w)
        : from(u), to(v), cap(c), flow(f), cost(w) {}
};
struct MCMF
{
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    int inq[maxn]; //是否在队列中
    int d[maxn];   //bellmanford
    int p[maxn];   //上一条弧
    int a[maxn];   //可改进量
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++) G[i].clear();
        edges.clear();
    }
    void AddEdge(int from, int to, int cap, int cost)
    {
        edges.pb(Edge(from, to, cap, 0, cost));
        edges.pb(Edge(to, from, 0, 0, -cost));
        m = edges.size();
        G[from].pb(m - 2);
        G[to].pb(m - 1);
```

```
    }
    bool BellmanFord(int s, int t, int& flow, ll& cost)
    {
        for (int i = 0; i < n; i++) d[i] = inf;
        mem(inq, 0);
        d[s] = 0;
        inq[s] = 1;
        p[s] = 0;
        a[s] = inf;
        queue<int> q;
        q.push(s);
        while (!q.empty())
        {
            int u = q.front();
            q.pop();
            inq[u] = 0;
            for (int i = 0; i < G[u].size(); i++)
            {
                Edge& e = edges[G[u][i]];
                if (e.cap > e.flow && d[e.to] > d[u] + e.cost)
                {
                    d[e.to] = d[u] + e.cost;
                    p[e.to] = G[u][i];
                    a[e.to] = min(a[u], e.cap - e.flow);
                    if (!inq[e.to])
                    {
                        q.push(e.to);
                        inq[e.to] = 1;
                    }
                }
            }
        }
        if (d[t] == inf) return false; // 当没有可增广的路时退出
        flow += a[t];
        cost += (ll)d[t] * (ll)a[t];
        for (int u = t; u != s; u = edges[p[u]].from)
        {
            edges[p[u]].flow += a[t];
            edges[p[u] ^ 1].flow -= a[t];
        }
        return true;
    }
    int MincostMaxflow(int s, int t, ll& cost)
    {
        int flow = 0;
        cost = 0;
        while (BellmanFord(s, t, flow, cost));
        return flow;
    }
};
```

2.8 第k短路

```
/*第k短路*/
const int MAXM= "" ;
const int MAXN= "" ;
```

```
struct node
{
    int v, w, next;
}edge[MAXM], revedge[MAXM];
struct A
{
    ll f, g, v;
    bool operator <(const A a)const {
        if(a.f == f) return a.g < g;
        return a.f < f;
    }
};
int e, vis[MAXN], q[MAXM * 5];
ll d[MAXN];
int head[MAXN], revhead[MAXN];
int n, m, s, t, k;
void init()
{
    e = 0;
    memset(head, -1, sizeof(head));
    memset(revhead, -1, sizeof(revhead));
}
void insert(int x, int y, int w)//插入边
{
    edge[e].v = y;
    edge[e].w = w;
    edge[e].next = head[x];
    head[x] = e;
    revedge[e].v = x;
    revedge[e].w = w;
    revedge[e].next =revhead[y];
    revhead[y] = e++;
}
void spfa(int src)
{
    for(int i = 1; i <= n; i++) d[i] = INF;
    memset(vis, 0, sizeof(vis));
    vis[src] = 0;
    int h = 0, t = 1;
    q[0] = src;
    d[src] = 0;
    while(h < t)
    {
        int u = q[h++];
        vis[u] = 0;
        for(int i = revhead[u] ; i != -1; i = revedge[i].next)
        {
            int v = revedge[i].v;
            int w = revedge[i].w;
            if(d[v] > d[u] + w)
            {
                d[v] = d[u] + w;
                if(!vis[v])
                {
                    q[t++] = v;
                    vis[v] = 1;
```

```
                }
            }
        }
    }
}
ll Astar(int src, int des)
{
    int cnt = 0;
    priority_queue<A>Q;
    if(src == des) k++;
    if(d[src] == INF) return -1;
    A t, tt;
    t.v = src, t.g = 0, t.f = t.g + d[src];
    Q.push(t);
    while(!Q.empty())
    {
        tt = Q.top();
        Q.pop();
        if(tt.v == des)
        {
            cnt++;
            if(cnt == k)
                return tt.g;
        }
        for(int i = head[tt.v]; i != -1; i = edge[i].next)
        {
            t.v = edge[i].v;
            t.g = tt.g + edge[i].w;
            t.f = t.g + d[t.v];
            Q.push(t);
        }
    }
    return -1;
}
int main()
{
        init();
        scanf("%d%d%d", &s, &t, &k);
        for(int i = 1; i <= m; i++)
        {
            int x,y,w;
            scanf("%d%d%d", &x, &y, &w);
            insert(x, y, w);
        }
        spfa(t);
        ll ans=Astar(s,t);
    return 0;
}
```

2.9 欧拉路径

```
/*找欧拉路径*/

#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5+10;
```

```
struct edge
{
    int to;
    int id;
    edge(){}
    edge(int to,int id):to(to),id(id){}
};
int top;
vector<edge> G[maxn];
vector<int> J[maxn],ans[maxn];
int N,M;
int tot;
int cnt;
int s[maxn*5];
int vis[maxn*5];
int deg[maxn];
int f[maxn];
int Find(int x)
{
    if(f[x]==x)
        return f[x];
    return f[x]=Find(f[x]);
}
void dfs(int u)
{
    for(int i=0;i<G[u].size();i++)
    {
        edge e=G[u][i];
        if(vis[e.id>>1])
            continue;
        vis[e.id>>1]=1;
        dfs(e.to);
        if(e.id%2==1)
            s[++top]=(-(e.id>>1));
        else
            s[++top]=(e.id>>1);
    }
}
void init()
{
    for(int i=0;i<maxn;i++)
    {
        f[i]=i;
        G[i].clear();
        J[i].clear();
        ans[i].clear();
    }
    memset(vis,0,sizeof(vis));
    memset(deg,0,sizeof(deg));
}
int main()
{
    //freopen("1003.in","r",stdin);
    while(scanf("%d%d",&N,&M)!=EOF)
    {
        init();
```

```
tot=1;
for(int i=1;i<=M;i++)
{
    int u,v;
    scanf("%d%d",&u,&v);
    G[u].push_back(edge(v,++tot));
    G[v].push_back(edge(u,++tot));
    deg[u]++;
    deg[v]++;
    f[Find(u)]=Find(v);
}
for(int i=1;i<=N;i++)
{
    if(deg[i]%2==1)
        J[Find(i)].push_back(i);
}
cnt=0;
for(int i=1;i<=N;i++)
{
    if(f[i]==i)
    {
        if(!J[i].size())
        {
            top=0;
            dfs(i);
            cnt++;
            while(top)
            {
                ans[cnt].push_back(s[top--]);
            }
        }
        else
        {
            top=0;
            for(int j=0;j<J[i].size();j+=2)
            {
                int u=J[i][j],v=J[i][j+1];
                G[u].push_back(edge(v,++tot));
                G[v].push_back(edge(u,++tot));
            }
            dfs(i);
            vector<int> pos;
            for(int i=top;i;i--)
            {
                if(s[i]>M||s[i]<-M)
                    pos.push_back(i);
            }
            for(int i=0;i<pos.size()-1;i++)
            {
                cnt++;
                for(int j=pos[i]-1;j>pos[i+1];j--)
                    ans[cnt].push_back(s[j]);
            }
            cnt++;
            for(int j=pos[pos.size()-1]-1;j;j--)
                ans[cnt].push_back(s[j]);
```

```
                    for(int j=top;j>pos[0];j--)
                        ans[cnt].push_back(s[j]);
                }
            }
        }
        int k=cnt;
        for(int i=1;i<=cnt;i++)
            if(ans[i].size()==0)
                k--;
        printf("%d\n",k);
        for(int i=1;i<=cnt;i++)
        {
            if(ans[i].size()!=0)
            {
                printf("%d",ans[i].size());
                for(int j=0;j<ans[i].size();j++)
                    printf(" %d",ans[i][j]);
                printf("\n");
            }
        }
    }
    return 0;
}
```

2.10 次小生成树

```
/*次小生成数*/

#include <cstdio>
#include <iostream>
#include <cstring>
#include <algorithm>
#include <vector>
#include <cmath>
using namespace std;
const int maxn=1000+5;
int N,M;
int f[maxn];
int judge[maxn];
int depth[maxn];
int gra[maxn][18];
int maxd[maxn][18];
struct Edge
{
    int u,v,d;
    Edge(int from,int to,int cost):u(from),v(to),d(cost){}
    bool operator <(const Edge &a) const
    {
        return d<a.d;
    }
};
vector<Edge> edges;
vector<Edge> G[maxn];
void init()
{
    for(int i=0;i<maxn;i++)
```

```
        f[i]=i;
    memset(judge,0,sizeof(judge));
    memset(depth,0,sizeof(depth));
    depth[1]=1;
    edges.clear();
    for(int i=0;i<maxn;i++)
        G[i].clear();
}
int Find(int x)
{
    if(f[x]==x)
        return f[x];
    return f[x]=Find(f[x]);
}
void unit(int x,int y)
{
    x=Find(x);
    y=Find(y);
    if(x==y)
        return ;
    f[x]=y;
}
bool same(int x,int y)
{
    return Find(x)==Find(y);
}
int Kruskal()
{
    sort(edges.begin(),edges.end());
    int res=0;
    for(int i=0;i<edges.size();i++)
    {
        Edge e=edges[i];
        if(!same(e.u,e.v))
        {
            res+=e.d;
            G[e.u].push_back(Edge(e.u,e.v,e.d));
            G[e.v].push_back(Edge(e.v,e.u,e.d));
            judge[i]=1;
            unit(e.u,e.v);
        }
    }
    return res;
}
void dfs(int x,int fa)
{
    for(int i=0;i<G[x].size();i++)
    {
        Edge e=G[x][i];
        if(e.v==fa)
            continue;
        depth[e.v]=depth[x]+1;
        gra[e.v][0]=x;
        maxd[e.v][0]=e.d;
        dfs(e.v,x);
    }
```

```
}
void solve()
{
    for(int i=1;(1<<i)<=N;i++)
    {
        for(int u=1;u<=N;u++)
        {
            gra[u][i]=gra[gra[u][i-1]][i-1];
            maxd[u][i]=max(maxd[u][i-1],maxd[gra[u][i-1]][i-1]);
        }
    }
}
int lca(int u,int v)
{
    if(depth[u]<depth[v])
        swap(u,v);
    int d=depth[u]-depth[v];
    for(int i=0;(1<<i)<=d;i++)
    {
        if((1<<i)&d)
            u=gra[u][i];
    }
    if(u==v)
        return u;
    for(int i=(int)log(N);i>=0;i--)
    {
        if(gra[u][i]!=gra[v][i])
            u=gra[u][i],v=gra[v][i];
    }
    return gra[u][0];
}
int qmax(int u,int v)
{
    int tmp=-0x3f3f3f3f;
    for(int i=0;(1<<i)<=N;i++)
    {
        if(depth[gra[u][i]]>=depth[v])
            tmp=max(tmp,maxd[u][i]);
    }
    //cout<<u<<" "<<v<<" "<<tmp<<endl;
    return tmp;
}
int main()
{
    ios::sync_with_stdio(0);
    int T;
    cin>>T;
    while(T--)
    {
        cin>>N>>M;
        int u,v,c;
        init();
        while(M--)
        {
            cin>>u>>v>>c;
            edges.push_back(Edge(u,v,c));
```

```
            //edges.push_back(Edge(v,u,c));
        }
        int MST=Kruskal();
        dfs(1,-1);
        solve();
        int ans=0x3f3f3f3f;
        for(int i=0;i<edges.size();i++)
        {
            if(judge[i]==1)
                continue;
            Edge e=edges[i];
            u=e.u,v=e.v,c=e.d;
            int LCA=lca(u,v);
            int maxu=qmax(u,LCA);
            int maxv=qmax(v,LCA);
            ans=min(ans,c-max(maxu,maxv));
        }
        cout<<MST<<" "<<ans+MST<<endl;
    }
    return 0;
}
```

2.11 最小树型图zhuliu

```
const int maxn = "Edit";
// 固定根的最小树型图，邻接矩阵写法
struct MDST
{
    int n;
    int w[maxn][maxn]; // 边权
    int vis[maxn];     // 访问标记，仅用来判断无解
    int ans;           // 计算答案
    int removed[maxn]; // 每个点是否被删除
    int cid[maxn];     // 所在圈编号
    int pre[maxn];     // 最小入边的起点
    int iw[maxn];      // 最小入边的权值
    int max_cid;       // 最大圈编号
    void init(int n)
    {
        this->n = n;
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) w[i][j] = INF;
    }
    void AddEdge(int u, int v, int cost)
    {
        w[u][v] = min(w[u][v], cost); // 重边取权最小的
    }
    // 从s出发能到达多少个结点
    int dfs(int s)
    {
        vis[s] = 1;
        int ans = 1;
        for (int i = 0; i < n; i++)
            if (!vis[i] && w[s][i] < INF) ans += dfs(i);
        return ans;
    }
```

```
// 从u出发沿着pre指针找圈
bool cycle(int u)
{
    max_cid++;
    int v = u;
    while (cid[v] != max_cid)
    {
        cid[v] = max_cid;
        v = pre[v];
    }
    return v == u;
}
// 计算u的最小入弧，入弧起点不得在圈c中
void update(int u)
{
    iw[u] = INF;
    for (int i = 0; i < n; i++)
        if (!removed[i] && w[i][u] < iw[u])
        {
            iw[u] = w[i][u];
            pre[u] = i;
        }
}
// 根结点为s，如果失败则返回false
bool solve(int s)
{
    memset(vis, 0, sizeof(vis));
    if (dfs(s) != n) return false;
    memset(removed, 0, sizeof(removed));
    memset(cid, 0, sizeof(cid));
    for (int u = 0; u < n; u++) update(u);
    pre[s] = s;
    iw[s] = 0; // 根结点特殊处理
    ans = max_cid = 0;
    for (;;)
    {
        bool have_cycle = false;
        for (int u = 0; u < n; u++)
            if (u != s && !removed[u] && cycle(u))
            {
                have_cycle = true;
                // 以下代码缩圈，圈上除了u之外的结点均删除
                int v = u;
                do
                {
                    if (v != u) removed[v] = 1;
                    ans += iw[v];
                    // 对于圈外点i，把边i->v改成i->u（并调整权值）；v->i改为u->i
                    // 注意圈上可能还有一个v'使得i->v'或者v'->i存在，
                    // 因此只保留权值最小的i->u和u->i
                    for (int i = 0; i < n; i++)
                        if (cid[i] != cid[u] && !removed[i])
                        {
                            if (w[i][v] < INF)
                                w[i][u] = min(w[i][u], w[i][v] - iw[v]);
                            w[u][i] = min(w[u][i], w[v][i]);
```

```
                            if (pre[i] == v) pre[i] = u;
                        }
                    v = pre[v];
                } while (v != u);
                update(u);
                break;
            }
        if (!have_cycle) break;
    }
    for (int i = 0; i < n; i++)
        if (!removed[i]) ans += iw[i];
    return true;
  }
};
```

3 数据结构

3.1 并查集

```
/*并查集（带路径压缩）*/
const int maxn= " ";
int p[maxn];
void init(int n)
{
    for(int i=0; i<=n; i++)
        p[i]=i;
}
int Find(int x)
{
    if(x==p[x])
        return p[x];
    int y=Find(p[x]);
    return p[x]=y;
}

int Union(int x,int y)
{
    int x1=Find(x);
    int y1=Find(y);
    if(x1==y1)
        return 0;
    p[x1]=y1;
      return 1;
}

/* 带权并查集 */
const int maxn=" ";
int p[maxn],ran[maxn];
void init(int n)
{
    for(int i=0; i<=n; i++)
    {
        p[i]=i;
        ran[i]=0;//
    }
    return;
}
int Find(int x)
{
    if(x==p[x])
        return p[x];
    int y=Find(p[x]);
        ran[x]=(ran[x]+ran[p[x]])%3;//
    return p[x]=y;
}
int Union(int x,int y,int typ)
{
    int x1=Find(x);
    int y1=Find(y);
    if(x1==y1)
    {
```

```
        if((ran[x]-ran[y]+3)%3==typ-1)//
            return 0;
        return 1;
    }
    p[x1]=y1;
    ran[x1]=(-ran[x]+typ-1+ran[y]+3)%3;//
    return 0;
}
```

3.2 LCA

```
/**倍增lca*/
const int maxn= " ";
const int N= "30";
int n;
int fa[maxn][N+5];
int deep[maxn];
vector<int>edge[maxn];

void dfs(int u,int pre){
  for(int i=0;i<edge[u].size();i++){
    int v=edge[u][i];
    if(v==pre) continue;
    fa[v][0]=u;//should give fa[v][0] value
    deep[v]=deep[u]+1; //also can preprocessing distance here
    dfs(v,u);
  }
}

void bz(){
 for(int j=1;j<=N;j++)
    for(int i=1;i<=n;i++)
      fa[i][j]=fa[fa[i][j-1]][j-1];
}

int lca(int u,int v){
  if(deep[u]<deep[v]) swap(u,v);
  int dc=deep[u]-deep[v];
  for(int i=0;i<N;i++){
    if((1<<i)&dc)//move u to dc+u
        u=fa[u][i];
  }
  if(u==v) return u;
  for(int i=N-1;i>=0;i--){
    if(fa[u][i]!=fa[v][i]){
        u=fa[u][i];v=fa[v][i];
    }
  }
  u=fa[u][0];//on the next level of lca,just move up one
  return u;
}

/*ST表预处理lca  o（nlogn+q） */
vector<int> edge[maxn], sp;
int dep[maxn], dfn[maxn];
pair<int,int> dp[21][maxn << 1];
```

```
void init(int n)
{
    for (int i = 0; i < n; i++) edge[i].clear();
    sp.clear();
}
void dfs(int u, int fa)
{
    dep[u] = dep[fa] + 1;
    dfn[u] = sp.size();//欧拉序列
    sp.push_back(u);
    for (auto& v : edge[u])
    {
        if (v == fa) continue;
        dfs(v, u);
        sp.push_back(u);
    }
}
/*i,j的lca为i,j进栈之间进出栈的点中进栈时间最早的*/
void initrmq()
{
    int n = sp.size();
    for (int i = 0; i < n; i++) dp[0][i] = {dfn[sp[i]], sp[i]};
    for (int i = 1; (1 << i) <= n; i++)
        for (int j = 0; j + (1 << i) - 1 < n; j++)
            dp[i][j] = min(dp[i - 1][j], dp[i - 1][j + (1 << (i - 1))]);
}
int lca(int u, int v)
{
    int l = dfn[u], r = dfn[v];
    if (l > r) swap(l, r);
    int k = 31 - __builtin_clz(r - l + 1);
    return min(dp[k][l], dp[k][r - (1 << k) + 1]).sec;
}
```

3.3 RMQ

```
/*RMQ HDU-4123*/
void ST(int n) {
    for (int j = 1; (1 << j) <= n; j++) {
        for (int i = 1; i + (1 << j) - 1 <= n; i++) {
                dp[i][j] = max(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
        }
    }
}

int RMQ(int l, int r) {
    // int k = 0;
    int k = 31 - __builtin_clz(r - l + 1);
    // while ((1 << (k + 1)) <= r - l + 1) k++;
    return max(dp[l][k], dp[r - (1 << k) + 1][k]);
}
```

3.4 线段树

```
/****单点更新 HDU - 1166****/
T tree[maxn<<2];
```

```
void pushup(int rt){
  tree[rt]=tree[rt*2]+tree[rt*2+1];
}

void build(int l,int r,int rt){
  if(l==r) {
      //scanf("%d",&tree[rt]);
      return;
      }
  int mid=(l+r)/2;
  build(l,mid,rt*2);
  build(mid+1,r,rt*2+1);
  pushup(rt);
}

T query(int l,int r,int L,int R,int rt){
 if(l>=L&&r<=R)
       return tree[rt];
 T ans=0;
 int mid=(l+r)/2;
 if(L<=mid){
    ans+=query(l,mid,L,R,rt*2);
 }
 if(R>mid){
    ans+=query(mid+1,r,L,R,rt*2+1);
 }
 return ans;
}

void update(int l,int r,int index,T add,int rt){
  if(l==r) {
    tree[rt]+=add;
    return;
  }
  int mid=(l+r)/2;
  if(index<=mid)
    update(l,mid,index,add,rt*2);

  else update(mid+1,r,index,add,rt*2+1);
  pushup(rt);
}


/****区间更新 poj-3468****/
T tree[maxn<<2];
T seg[maxn<<2];
void pushup(int rt){
 tree[rt]=tree[rt*2]+tree[rt*2+1];
}

void pushdown(int len,int rt){
  if(seg[rt]){
    seg[rt*2]+=seg[rt];
    seg[rt*2+1]+=seg[rt];
    tree[rt*2]+=(len-len/2)*seg[rt];
```

```
    tree[rt*2+1]+=len/2*seg[rt];
    seg[rt]=0;
  }
}
void build(int l,int r,int rt){
  seg[rt]=0;
  if(l==r) {
    //scanf("%d",&tree[rt]);
    return;}
  int mid=(l+r)/2;
  build(l,mid,rt*2);
  build(mid+1,r,rt*2+1);
  pushup(rt);
}
void update(int l,int r,int L,int R,T add,int rt){
  if(l>=L&&r<=R) {
    seg[rt]+=add;
    tree[rt]+=(r-l+1)*add;
    return;
  }
  pushdown(r-l+1,rt);
  int mid=(l+r)/2;
  if(L<=mid)
    update(l,mid,L,R,add,rt*2);

  if(R>mid)
    update(mid+1,r,L,R,add,rt*2+1);

  pushup(rt);
}

T query(int l,int r,int L,int R,int rt){
  if(l>=L&&r<=R){
     return tree[rt];
  }
  T ans=0;
  pushdown(r-l+1,rt);
  int mid=(l+r)/2;
  if(L<=mid) ans+=query(l,mid,L,R,rt*2);
  if(R>mid) ans+=query(mid+1,r,L,R,rt*2+1);
  return ans;
}
```

3.5 树状数组

```
/* 树状数组单点更新 */

int lowbit(int x){
return x&(-x);
}
T sum(int x){
  T ret=0;
  while(x>0){
    ret+=bit[x];
    x-=lowbit(x);
  }
```

```
return ret;
}

void add(int x,T d){
    if(x<0) return;
   while(x<=n){
    bit[x]+=d;
    x+=lowbit(x);
   }
}

/**区间更新区间查询 **/
int lowbit(int x){
return x&(-x);
}

void add(int x,int y){
   for(int i=x;i<=n;i+=lowbit(i))
    for(int j=y;j<=n;j+=lowbit(j))
       bit[i][j]++;
}

T sum(int x,int y){
   T ret=0;
   for(int i=x;i>0;i-=lowbit(i))
     for(int j=y;j>0;j-=lowbit(j))
       ret+=bit[i][j];
   return ret;
}
```

3.6 主席树

```
/**主席树 区间第k小 POJ 2104*/
int n,m,cnt;
int root[maxn],a[maxn];
int x,y,k;
struct node{
  int l,r,sum;
}T[maxn*40];
vector<int>v;

int getid(int x){
 return lower_bound(v.begin(),v.end(),x)-v.begin()+1;

}
void init()
{
    cnt=0;
    root[0]=0;
    T[0].l = T[0].r = T[0].sum = 0;
    v.clear();
}

void update(int l,int r,int &x,int y,int pos){
  T[++cnt]=T[y];
  T[cnt].sum++;
```

```
  x=cnt;
  if(l==r) return ;
  int mid=(l+r)/2;
  if(mid>=pos) update(l,mid,T[x].l,T[y].l,pos);
  else update(mid+1,r,T[x].r,T[y].r,pos);
}
int query(int l,int r,int x,int y,int k){
  if(l==r) return l;
  int mid=(l+r)/2;
  int sum=T[T[y].l].sum-T[T[x].l].sum;
  if(sum>=k) return query(l,mid,T[x].l,T[y].l,k);
  else return query(mid+1,r,T[x].r,T[y].r,k-sum);
}
int main(){
  while(scanf("%d%d",&n,&m)==2){
    init();
    //cnt=0;
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]),v.push_back(a[i]);
    sort(v.begin(),v.end()),v.erase(unique(v.begin(),v.end()),v.end());

    for(int i=1;i<=n;i++)
         update(1,n,root[i],root[i-1],getid(a[i]));
    while(m--){
        int l,r,k;
        scanf("%d%d%d",&x,&y,&k);
        printf("%d\n",v[query(1,n,root[x-1],root[y],k)-1]);
    }
  }
return 0;
}
```

```
/**主席树区间更新 HDU 4348*/
const int maxn=1e5+100;
struct node{
 int l;int r;
 ll lazy;
 ll sum;

}T[maxn*40];
int cnt;
int root[maxn];
void pushup(int x,int len){
   T[x].sum=T[T[x].l].sum+T[T[x].r].sum+T[x].lazy*len;
}
void build(int l,int r,int &x){
   x=++cnt;
   if(l==r) {
    T[x].lazy=0;
      scanf("%lld",&T[x].sum);
      return ;
   }
   int mid=(l+r)/2;
   build(l,mid,T[x].l);
```

```
   build(mid+1,r,T[x].r);
   pushup(x,r-l+1);
}

void update(int l,int r,int L,int R,int &x,int y,int val){
   T[++cnt]=T[y];
   x=cnt;
   if(l>=L&&r<=R){
      T[x].lazy+=val;
      T[x].sum+=(r-l+1)*val;
      return;
   }
   int mid=(l+r)/2;
   if(mid>=L)   update(l,mid,L,R,T[x].l,T[y].l,val);
   if(mid<R) update(mid+1,r,L,R,T[x].r,T[y].r,val);
  pushup(x,r-l+1);

}
ll query(int l,int r,int L,int R,ll adv,int x){
 if(l>=L&&r<=R){
    return T[x].sum+adv*(r-l+1);

 }
  adv+=T[x].lazy;
  int mid=(l+r)/2;
  ll sum=0;
  if(L<=mid) sum+=query(l,mid,L,R,adv,T[x].l);
   if(R>mid) sum+=query(mid+1,r,L,R,adv,T[x].r);
 return sum;
}
```

3.7 树链剖分

```
#include<bits/stdc++.h>
using namespace std;
#define lson rt<<1
#define rson rt<<1|1
#define Lson L,mid,lson
#define Rson mid+1,R,rson
const int maxn=1e5+10;
typedef unsigned long long ull;
ull INF=0xffffffffffffffff;
int top[maxn],son[maxn],dep[maxn],f[maxn];
int sz[maxn],key[maxn];
int id[maxn];
vector<int> G[maxn];
int N;
int tot;
ull sum[maxn*4];
ull add[maxn*4];
ull mul[maxn*4];
void pushup(int rt)
{
    sum[rt]=sum[lson]+sum[rson];
}
void pushdown(int rt,int len)
```

```
{
    if(add[rt]!=0||mul[rt]!=1)
    {
        add[rt<<1]=(add[rt<<1]*mul[rt]+add[rt]);
        add[rt<<1|1]=(add[rt<<1|1]*mul[rt]+add[rt]);
        mul[rt<<1]=(mul[rt<<1]*mul[rt]);
        mul[rt<<1|1]=(mul[rt<<1|1]*mul[rt]);
        sum[rt<<1]=(add[rt]*(len-(len>>1))+sum[rt<<1]*mul[rt]);
        sum[rt<<1|1]=((add[rt]*(len>>1))+sum[rt<<1|1]*mul[rt]);
        add[rt]=0;
        mul[rt]=1;
    }
}
void init()
{
    memset(son,0,sizeof(son));
    memset(sz,0,sizeof(sz));
    for(int i=0;i<maxn;i++)
        G[i].clear();
    tot=0;
    dep[1]=0;
}
void build(int L,int R,int rt)
{
    add[rt]=0;
    mul[rt]=1;
    if(L==R)
    {
        sum[rt]=0;
        return ;
    }
    int mid=(L+R)>>1;
    build(Lson);
    build(Rson);
    pushup(rt);
}
void dfs1(int u,int fa)
{
    sz[u]=1;
    f[u]=fa;
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i];
        if(v==fa)
            continue;
        dep[v]=dep[u]+1;
        dfs1(v,u);
        sz[u]+=sz[v];
        if(son[u]==0||sz[v]>sz[son[u]])
        {
            son[u]=v;
        }
    }
}
void dfs2(int u,int fa)
{
```

```
    top[u]=fa;
    id[u]=++tot;
    if(son[u])
        dfs2(son[u],fa);
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i];
        if(v==f[u])
            continue;
        if(v!=son[u])
            dfs2(v,v);
    }
}
void updateplus(int l,int r,ull val,int L,int R,int rt)
{
    if(l<=L&&r>=R)
    {
        add[rt]+=val;
        sum[rt]+=val*(R-L+1);
        return ;
    }
    pushdown(rt,R-L+1);
    int mid=(L+R)>>1;
    if(l<=mid)
        updateplus(l,r,val,Lson);
    if(r>mid)
        updateplus(l,r,val,Rson);
    pushup(rt);
}
void updatemul(int l,int r,ull val,int L,int R,int rt)
{
    if(l<=L&&r>=R)
    {
        add[rt]*=val;
        mul[rt]*=val;
        sum[rt]*=val;
        return ;
    }
    pushdown(rt,R-L+1);
    int mid=(L+R)>>1;
    if(l<=mid)
        updatemul(l,r,val,Lson);
    if(r>mid)
        updatemul(l,r,val,Rson);
    pushup(rt);
}
void changeadd(int x,int y,ull val)
{
    while(top[x]!=top[y])
    {
        if(dep[top[x]]<dep[top[y]])
            swap(x,y);
        updateplus(id[top[x]],id[x],val,1,N,1);
        x=f[top[x]];
    }
    if(dep[x]>dep[y])
```

```
        swap(x,y);
    updateplus(id[x],id[y],val,1,N,1);
}
void changemul(int x,int y,ull val)
{
    //cout<<x<<" "<<y<<" "<<val<<endl;
    while(top[x]!=top[y])
    {
        if(dep[top[x]]<dep[top[y]])
            swap(x,y);
        updatemul(id[top[x]],id[x],val,1,N,1);
        x=f[top[x]];
    }
    if(dep[x]>dep[y])
        swap(x,y);
    updatemul(id[x],id[y],val,1,N,1);
}
ull query(int l,int r,int L,int R,int rt)
{
    if(l<=L&&r>=R)
    {
        return sum[rt];
    }
    pushdown(rt,R-L+1);
    int mid=(L+R)>>1;
    ull res=0;
    if(l<=mid)
        res+=query(l,r,Lson);
    if(r>mid)
        res+=query(l,r,Rson);
    return res;
}
ull get(int x, int y)
{
    ull res=0;
    while(top[x] != top[y])
    {
        if(dep[top[x]] < dep[top[y]])
            swap(x, y);
        res+=query(id[top[x]],id[x],1,N,1);
        x = f[top[x]];
    }
    if(dep[x] > dep[y])
         swap(x, y);
    res+=query(id[x],id[y],1,N,1);
    return res;
}
```

3.8 LCT

```
//维护点权
struct LCT
{
    int val[maxn],sum[maxn];
    int rev[maxn],ch[maxn][2],fa[maxn];
    int nxt[maxn];
```

```
int stk[maxn];
void init(int n)
{
    for(int i=1;i<=n;i++)
        val[i]=1,fa[i]=0,rev[i]=0,ch[i][0]=ch[i][1]=0;
}
bool isroot(int x)
{
    return ch[fa[x]][0]!=x&&ch[fa[x]][1]!=x;
}
bool get(int x)
{
    return ch[fa[x]][1]==x;
}
void pushdown(int x)
{
    if(!rev[x])
        return ;
    swap(ch[x][0],ch[x][1]);
    if(ch[x][0])
        rev[ch[x][0]]^=1;
    if(ch[x][1])
        rev[ch[x][1]]^=1;
    rev[x]^=1;
}
void pushup(int x)
{
    sum[x]=val[x]+sum[ch[x][0]]+sum[ch[x][1]];
}
void rotate(int x)
{
    int y=fa[x],z=fa[fa[x]],d=get(x);
    if(!isroot(y))
        ch[z][get(y)]=x;
    fa[x]=z;
    ch[y][d]=ch[x][d^1],fa[ch[y][d]]=y;
    ch[x][d^1]=y,fa[y]=x;
    pushup(y),pushup(x);
}
void splay(int x)
{
    int top=0;
    stk[++top]=x;
    for(int i=x;!isroot(i);i=fa[i]) stk[++top]=fa[i];
    for(int i=top;i;i--) pushdown(stk[i]);
    for(int f;!isroot(x);rotate(x))
        if(!isroot(f=fa[x]))
            rotate(get(x)==get(f)?f:x);
}
void access(int x)
{
    for(int y=0;x;y=x,x=fa[x])
    {
        splay(x);
        ch[x][1]=y;
        pushup(x);
```

```
        }
    }
    int find(int x)
    {
        access(x),splay(x);
        while(ch[x][0])
            x=ch[x][0];
        return x;
    }
    void makeroot(int x) {access(x),splay(x),rev[x]^=1;}
    void link(int x,int y){makeroot(x),fa[x]=y,splay(x);}
    void cut(int x,int y){makeroot(x),access(y),splay(y),fa[x]=ch[y][0]=0;}
    void update(int x,int v){val[x]=v,access(x),splay(x);}
    int query(int x,int y)
    {
        makeroot(y),access(x),splay(x);
        return sum[ch[x][0]];
    }
}lct;

//维护子树
#include <cstdio>
#include <algorithm>
#define N 100010
using namespace std;
int fa[N] , c[2][N] , si[N] , sum[N] , rev[N];
char str[5];
void pushup(int x)
{
    sum[x] = sum[c[0][x]] + sum[c[1][x]] + si[x] + 1;
}
void pushdown(int x)
{
    if(rev[x])
    {
        int l = c[0][x] , r = c[1][x];
        swap(c[0][l] , c[1][l]) , swap(c[0][r] , c[1][r]);
        rev[l] ^= 1 , rev[r] ^= 1 , rev[x] = 0;
    }
}
bool isroot(int x)
{
    return c[0][fa[x]] != x && c[1][fa[x]] != x;
}
void update(int x)
{
    if(!isroot(x)) update(fa[x]);
    pushdown(x);
}
void rotate(int x)
{
    int y = fa[x] , z = fa[y] , l = (c[1][y] == x) , r = l ^ 1;
    if(!isroot(y)) c[c[1][z] == y][z] = x;
    fa[x] = z , fa[y] = x , fa[c[r][x]] = y , c[l][y] = c[r][x] , c[r][x] = y;
    pushup(y) , pushup(x);
}
```

```
void splay(int x)
{
    update(x);
    while(!isroot(x))
    {
        int y = fa[x] , z = fa[y];
        if(!isroot(y))
        {
            if((c[0][y] == x) ^ (c[0][z] == y)) rotate(x);
            else rotate(y);
        }
        rotate(x);
    }
}
void access(int x)
{
    int t = 0;
    while(x)
    {
        splay(x) ;
        si[x] += sum[c[1][x]] - sum[t] , c[1][x] = t , pushup(x) , t = x , x = fa[x];
    }
}
void makeroot(int x)
{
    access(x) , splay(x) , swap(c[0][x] , c[1][x]) , rev[x] = 1;
}
void split(int x , int y)
{
    makeroot(x) , makeroot(y);
}
void link(int x , int y)
{
    split(x , y) , fa[x] = y , si[y] += sum[x] , pushup(y);
}
```

3.9 Splay

```
#define key_value ch[ch[root][1]][0]
const int maxn = 1 << 19;

struct Splay
{
    int a[maxn];
    int sz[maxn], ch[maxn][2], fa[maxn];
    int key[maxn], rev[maxn];
    int root, tot;
    int stk[maxn], top;

#ifndef ONLINE_JUDGE
    void Treavel(int x)
    {
        if (x)
        {
            Treavel(ch[x][0]);
            printf("结点:%2d: 左儿子 %2d 右儿子 %2d 父结点 %2d size= %2d key= %2d\n",
```

```
            x, ch[x][0], ch[x][1], fa[x], sz[x], key[x]);
        Treavel(ch[x][1]);
    }
}

void debug()
{
    printf("root:%d\n", root);
    Treavel(root);
}
#endif

void init(int n)
{
    tot = 0, top = 0;
    root = newnode(0, -1);
    ch[root][1] = newnode(root, -1);
    for (int i = 0; i < n; i++) a[i] = i + 1;
    key_value = build(0, n - 1, ch[root][1]);
    pushup(ch[root][1]);
    pushup(root);
}

int newnode(int p = 0, int k = 0)
{
    int x = top ? stk[top--] : ++tot;
    fa[x] = p;
    sz[x] = 1;
    ch[x][0] = ch[x][1] = 0;
    key[x] = k;
    rev[x] = 0;
    return x;
}

void pushdown(int x)
{
    if (rev[x])
    {
        swap(ch[x][0], ch[x][1]);
        if (ch[x][0]) rev[ch[x][0]] ^= 1;
        if (ch[x][1]) rev[ch[x][1]] ^= 1;
        rev[x] = 0;
    }
}

void pushup(int x)
{
    sz[x] = sz[ch[x][0]] + sz[ch[x][1]] + 1;
}

void rotate(int x, int d)
{
    int y = fa[x];
    pushdown(y), pushdown(x);
    ch[y][d ^ 1] = ch[x][d];
    fa[ch[x][d]] = y;
```

```
    if (fa[y]) ch[fa[y]][ch[fa[y]][1] == y] = x;
    fa[x] = fa[y];
    ch[x][d] = y;
    fa[y] = x;
    pushup(y);
}

void splay(int x, int goal = 0)
{
    pushdown(x);
    while (fa[x] != goal)
    {
        if (fa[fa[x]] == goal)
            rotate(x, ch[fa[x]][0] == x);
        else
        {
            int y = fa[x];
            int d = ch[fa[y]][0] == y;
            ch[y][d] == x ? rotate(x, d ^ 1) : rotate(y, d);
            rotate(x, d);
        }
    }
    pushup(x);
    if (goal == 0) root = x;
}

int kth(int r, int k)
{
    pushdown(r);
    int t = sz[ch[r][0]] + 1;
    if (t == k) return r;
    return t > k ? kth(ch[r][0], k) : kth(ch[r][1], k - t);
}

int build(int l, int r, int p)
{
    if (l > r) return 0;
    int mid = l + r >> 1;
    int x = newnode(p, a[mid]);
    ch[x][0] = build(l, mid - 1, x);
    ch[x][1] = build(mid + 1, r, x);
    pushup(x);
    return x;
}

void select(int l, int r)
{
    splay(kth(root, l), 0);
    splay(kth(ch[root][1], r - l + 2), root);
}

void filp(int l, int r)
{
    select(l, r);
    rev[key_value] ^= 1;
}
```

```
    void cut(int l, int r, int c)
    {
        select(l, r);
        int tmp = key_value;
        key_value = 0;
        pushup(ch[root][1]), pushup(root);
        select(c + 1, c);
        key_value = tmp, fa[key_value] = ch[root][1];
        pushup(ch[root][1]), pushup(root);
        splay(tmp);
    }

    int ans[maxn], pos;

    void dfs(int x)
    {
        if (x)
        {
            pushdown(x);
            dfs(ch[x][0]);
            if (~key[x]) ans[pos++] = key[x];
            dfs(ch[x][1]);
        }
    }

    void print()
    {
        pos = 0;
        dfs(root);
        for (int i = 0; i < pos; i++) printf("%d%c", ans[i], " \n"[i == pos - 1]);
    }
} gao;
```

3.10 kdtree

```
//求最近点
//hdu5992
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const ll INF=0x3f3f3f3f3f3f3f3f;
const int maxn=2e5+100;
struct Point
{
    int xy[2];
    int l,r,id;
    int c;//题目额外要求的
    void read(int i)
    {
        id=i;
        scanf("%d%d%d",&xy[0],&xy[1],&c);
    }
}p[maxn];
Point result;
int cmpw;//标记是哪一维的比较
```

```
ll ans;
int cost;
bool cmp(const Point &a,const Point &b)
{
    return a.xy[cmpw]<b.xy[cmpw];
}
int build(int l,int r,int w)//w是维度标记
{
    int m=(l+r)/2;cmpw=w;
    nth_element(p+l,p+m,p+1+r,cmp);
    if(l!=m)
      p[m].l=build(l,m-1,!w);
    else p[m].l=0;
   if(r!=m)
     p[m].r=build(m+1,r,!w);
     else p[m].r=0;
     return m;
}
ll dis(ll x,ll y=0)
{
    return x*x+y*y;
}
void  query(int rt,int w,ll x,ll y)
{
    ll tmp=dis(x-p[rt].xy[0],y-p[rt].xy[1]);
    if(cost<p[rt].c)
     tmp=INF;
    if(tmp<ans||(tmp!=INF&&tmp==ans&&p[rt].id<result.id))//attention 按题目要求来
     result=p[rt];
    ans=min(ans,tmp);
    if(p[rt].l&&p[rt].r)
    {
        bool flag;ll d;
        if(!w)
        {
            flag=(x<=p[rt].xy[0]);
            d=dis(x-p[rt].xy[0]);
        }
        else
        {
            flag=(y<=p[rt].xy[1]);
            d=dis(y-p[rt].xy[1]);
        }

       query(flag?p[rt].l:p[rt].r,!w,x,y);
       if(d<ans)
         query(flag?p[rt].r:p[rt].l,!w,x,y);
    }
    else if(p[rt].l) query(p[rt].l,!w,x,y);
    else if(p[rt].r) query(p[rt].r,!w,x,y);

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

```
    while(t--)
    {
        scanf("%d%d",&n,&q);
        for(int i=1;i<=n;i++)
          p[i].read(i);
          int rt=build(1,n,0);

        for(int i=1;i<=q;i++)
        {
            ans=INF;
            int x,y;
            scanf("%d%d%d",&x,&y,&cost);
            query(rt,0,x,y);
            printf("%d %d %d\n",result.xy[0],result.xy[1],result.c);
        }
    }
}
```

3.11 区间不同数

```
/* 区间不同数 */
/*树状数组*/
const int maxn=" ";
int bit[maxn];
int a[maxn];
int ans[maxn];
map<int,int>mp;
struct node{
    int l,r,id;
    bool operator<(const node &t)const{
        return r<t.r;
    }
}q[maxn];
int sum(int x);
void add(int x,int val);
int main()
{
       for(int i=1;i<=n;i++)
         scanf("%d",&a[i]);//输入数组
      sort(q+1,q+1+Q);//将询问离散
      int pre=1;
     for(int i=1;i<=Q;i++)
     {
         for(int j=pre;j<=q[i].r;j++)
         {
             if(mp[a[j]])
             {
                 add(mp[a[j]],-1);
             }
             add(j,1);
             mp[a[j]]=j;
         }
         pre=q[i].r+1;
        ans[q[i].id]=sum(q[i].r)-sum(q[i].l-1);
     }
     for(int i=1;i<=Q;i++)
```

```
        printf("%d\n",ans[i]);
}

/*主席树*/
const int maxn="";
int n,cnt;
int root[maxn],a[maxn];
map<int,int>mp;
struct node{
  int l,r,sum;
}T[maxn*40];
void init()
{
    cnt=0;
    mp.clear();
    root[0]=0;
    T[0].l = T[0].r = T[0].sum = 0;
}

void update(int l,int r,int &x,int y,int pos,int val){
  T[++cnt]=T[y];
  T[cnt].sum+=val;
  x=cnt;
  if(l==r) return ;
  int mid=(l+r)/2;
  if(mid>=pos) update(l,mid,T[x].l,T[y].l,pos,val);
  else update(mid+1,r,T[x].r,T[y].r,pos,val);

}

int query(int l,int r,int pos,int y){
  if(l==r) return T[y].sum;
  int mid=(l+r)/2;
  if(pos <= mid)
        return query(l,mid,pos,T[y].l) + T[T[y].r].sum;
  else
      return query(mid+1,r,pos, T[y].r);
}

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

    int tmp;
    for(int i=1;i<=n;i++){
        if(mp[a[i]]==0){
           update(1,n,root[i],root[i-1],i,1);
          mp[a[i]]=i;
        }
        else{
            update(1,n,tmp,root[i-1],mp[a[i]],-1);
            update(1,n,root[i],tmp,i,1);
            mp[a[i]]=i;
         }
    }
```

```
    for(int i=1;i<=q;i++)
    {
        int l,r;
        scanf("%d%d",&l,&r);
        printf("%d\n",query(1,n,l,root[r]));
    }
  }
```

3.12 矩形面积并

```
/*矩形面积并*/
//hdu1542
#include<bits/stdc++.h>
using namespace std;
const int maxn=2010;
struct seg{
   double l,r,h;
   int s;
}s[maxn];
int res;
int col[maxn<<2];
double sum[maxn<<2];
vector<double>v;
int cmp(seg a,seg b){
return a.h<b.h;
}
/*对点离散化之后，原来的区间[l,r]变为[l,mid],[mid+1,r]时，会缺少一段*/
void pushup(int rt,int l,int r){
   if(col[rt]) sum[rt]=v[r+1]-v[l];//[)
   else if(l==r) sum[rt]=0;
   else sum[rt]=sum[rt<<1]+sum[rt<<1|1];
}

void update(int l,int r,int c,int rt,int ll,int rr){//l,r is fresh area
  if(ll>=l&&rr<=r){
    col[rt]+=c;
    pushup(rt,ll,rr);
    return;
  }

  int mid=(ll+rr)/2;
  if(l<=mid) update(l,r,c,rt*2,ll,mid);
  if(r>mid) update(l,r,c,rt*2+1,mid+1,rr);
  pushup(rt,ll,rr);
}
int getid(double x)
{
    return lower_bound(v.begin(),v.end(),x)-v.begin();
}
int main(){
    int n;int k=0;
    while(cin>>n&&n){
            k++;
            int cnt=0;
            v.clear();
        for(int i=0;i<n;i++){
```

```
            double a,b,c,d;
            cin>>a>>b>>c>>d;
            s[cnt++]=seg{a,c,b,1};//bottom line
            s[cnt++]=seg{a,c,d,-1};//top line
           v.push_back(a);v.push_back(c);
        }
        sort(v.begin(),v.end());v.erase(unique(v.begin(),v.end()),v.end());
        sort(s,s+cnt,cmp);
        res=v.size();
        memset(col,0,sizeof(col));
        memset(sum,0,sizeof(sum));
        double ans=0;
        for(int i=0;i<cnt-1;i++){
            int l=getid(s[i].l);
            int r=getid(s[i].r)-1;//attention
            update(l,r,s[i].s,1,0,res);
            ans+=sum[1]*(s[i+1].h-s[i].h);
        }
        printf("Test case #%d\nTotal explored area: %.2lf\n\n",k,ans);

    }

}
```

3.13 矩形面积交

```
/*矩形面积交*/

#include<cstdio>
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
#define lson rt<<1
#define rson rt<<1|1
#define Lson L,mid,lson
#define Rson mid+1,R,rson
const int maxn=2005;
struct segment
{
    int l;
    int r;
    int h;
    int type;
    segment(){}
    segment(int l,int r,int h,int type):l(l),r(r),h(h),type(type){}
    bool operator <(const segment & ryh) const
    {
        return h<ryh.h;
    }
}a[maxn];
struct point
{
    int x1,y1,x2,y2,z1,z2;
    point(){}
    point(int x1,int y1,int z1,int x2,int y2,int z2):x1(x1),y1(y1),z1(z1),x2(x2),y2(y2),z2(z2){}
```

```
}cube[maxn];
int cnt[maxn<<2];
int allx[maxn];
int allz[maxn];
int one[maxn<<2],two[maxn<<2],three[maxn<<2];
void pushup(int L,int R,int rt)
{
    if(cnt[rt]>=3)
    {
        one[rt]=two[rt]=three[rt]=allx[R+1]-allx[L];
    }
    else if(cnt[rt]==2)
    {
        one[rt]=two[rt]=allx[R+1]-allx[L];
        if(L==R)
            three[rt]=0;
        else
            three[rt]=one[lson]+one[rson];
    }
    else if(cnt[rt]==1)
    {
        one[rt]=allx[R+1]-allx[L];
        if(L==R)
        {
            two[rt]=three[rt]=0;
        }
        else
        {
            three[rt]=two[lson]+two[rson];
            two[rt]=one[lson]+one[rson];
        }
    }
    else
    {
        if(L==R)
        {
            one[rt]=two[rt]=three[rt]=0;
        }
        else
        {
            one[rt]=one[lson]+one[rson];
            two[rt]=two[lson]+two[rson];
            three[rt]=three[lson]+three[rson];
        }
    }
}
void update(int l,int r,int val,int L,int R,int rt)
{
    if(l<=L&&r>=R)
    {
        cnt[rt]+=val;
        pushup(L,R,rt);
        return ;
    }
    int mid=(L+R)>>1;
    if(l<=mid)
```

```
        update(l,r,val,Lson);
    if(r>mid)
        update(l,r,val,Rson);
    pushup(L,R,rt);
}
int main()
{
    int T;
    scanf("%d",&T);
    int kase=0;
    while(T--)
    {
        int N;
        scanf("%d",&N);
        for(int i=1;i<=N;i++)
        {
            int x1,x2,y1,y2,z1,z2;
            scanf("%d%d%d%d%d%d",&x1,&y1,&z1,&x2,&y2,&z2);
            cube[i]=point(x1,y1,z1,x2,y2,z2);
            allz[i]=z1,allz[i+N]=z2;
        }
        sort(allz+1,allz+1+N*2);
        int cntz=unique(allz+1,allz+1+N*2)-allz-1;
        long long ans=0;
        for(int i=1;i<cntz;i++)
        {
            int tot=0;
            memset(cnt,0,sizeof(cnt));
            memset(one,0,sizeof(one));
            memset(two,0,sizeof(two));
            memset(three,0,sizeof(three));
            for(int j=1;j<=N;j++)
            {
                if(cube[j].z1<=allz[i]&&cube[j].z2>=allz[i+1])
                {
                    a[++tot]=segment(cube[j].x1,cube[j].x2,cube[j].y1,1);
                    allx[tot]=cube[j].x1;
                    a[++tot]=segment(cube[j].x1,cube[j].x2,cube[j].y2,-1);
                    allx[tot]=cube[j].x2;
                }
            }
            sort(allx+1,allx+1+tot);
            int m=unique(allx+1,allx+1+tot)-allx-1;
            sort(a+1,a+tot+1);
            for(int j=1;j<tot;j++)
            {
                int l=lower_bound(allx+1,allx+1+m,a[j].l)-allx;
                int r=lower_bound(allx+1,allx+1+m,a[j].r)-allx;
                if(l<r)
                    update(l,r-1,a[j].type,1,m,1);
                ans+=(long long)(three[1])*(a[j+1].h-a[j].h)*(allz[i+1]-allz[i]);
            }
        }
        printf("Case %d: %lld\n",++kase,ans);
    }
    return 0;
```

```
}
```

3.14 矩形周长并

```
#include<cstdio>
#include<iostream>
#include<queue>
#include<cmath>
#include<cstring>
#include<algorithm>
//#define ll long long
#define pb(x) push_back(x)
#define fir first
#define sec second
using namespace std;

//freopen("data.in","r",stdin);
//freopen("data.out","w",stdout);
//ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);


const int INF=0x3f3f3f3f;
const int maxn=5010;
struct seg{
   int l,r,h;
   int s;
}s[maxn<<1];
int cn[maxn*3];
unsigned char rnum[maxn*3],lnum[maxn*3];
int col[maxn*3];
int sum[maxn*3];

int cmp(seg a,seg b){
return a.h<b.h;
}
void pushup(int rt,int l,int r){
   if(col[rt]) {sum[rt]=r-l+1;//[ )
               cn[rt]=1;rnum[rt]=lnum[rt]='1';}
   else if(l==r) {sum[rt]=0;cn[rt]=0;rnum[rt]=lnum[rt]='0';}
   else {sum[rt]=sum[rt<<1]+sum[rt<<1|1];
          lnum[rt]=lnum[rt<<1];rnum[rt]=rnum[rt<<1|1];
          cn[rt]=cn[rt<<1]+cn[rt<<1|1]-(rnum[rt<<1]-'0')&&(lnum[rt<<1|1]-'0');
          }
}

void update(int l,int r,int c,int rt,int ll,int rr){//l,r is fresh area
  if(ll>=l&&rr<=r){
    col[rt]+=c;
    pushup(rt,ll,rr);
    return;
  }
  int mid=(ll+rr)/2;
  if(l<=mid) update(l,r,c,rt*2,ll,mid);
  if(r>mid) update(l,r,c,rt*2+1,mid+1,rr);
  pushup(rt,ll,rr);
}
```

```
int main(){
    int n;int k=0;
    while(scanf("%d",&n)==1&&n){
            k++;
            int cnt=0;int ll=INF;int rr=-INF;
        for(int i=0;i<n;i++){
            int a,b,c,d;
            scanf("%d%d%d%d",&a,&b,&c,&d);
            //cin>>a>>b>>c>>d;
            s[cnt++]=seg{a,c,b,1};//bottom line

            s[cnt++]=seg{a,c,d,-1};//top line
            ll=min(ll,a);
            rr=max(rr,c);
          //  x[cnt++]=c;
        }
       // sort(x,x+cnt);
        sort(s,s+cnt,cmp);

        memset(col,0,sizeof(col));
        memset(sum,0,sizeof(sum));
        //memset(rnum,'0',sizeof(rnum));
        //memset(lnum,'0',sizeof(lnum));
        memset(cn,0,sizeof(cn));
        for(int i=0;i<maxn*3;i++){
            rnum[i]=lnum[i]='0';
        }
        int ans=0;int pre=0;
        for(int i=0;i<cnt-1;i++){
            update(s[i].l,s[i].r-1,s[i].s,1,ll,rr);
            ans+=abs(sum[1]-pre)+cn[1]*2*(s[i+1].h-s[i].h);
            pre=sum[1];
        }
        ans+=sum[1];
        printf("%d\n",ans);
        //cout<<ans<<endl;
    }
return 0;
}
```

3.15 二维线段树

```
/*
单点更新，区间查询
HDU 4819 Mosaic
给定一个n*n的矩阵,每次给定一个子矩阵区域(x,y,l),
求出该区域内的最大值(A)和最小值(B),输出(A+B)/2,并用这个值更新矩阵[x,y]的值
*/
# include<cstdio>
# include<cstring>
# include<algorithm>
using namespace std;

# define lson l,m,rt<<1
```

```
# define rson m+1,r,rt<<1|1
# define MAXN 805
int xL,xR,yL,yR,val;
int maxv,minv;
int Max[MAXN<<2][MAXN<<2],Min[MAXN<<2][MAXN<<2];
int N,mat[MAXN][MAXN];

void PushUp(int xrt,int rt)
{
    Max[xrt][rt]=max(Max[xrt][rt<<1],Max[xrt][rt<<1|1]);
    Min[xrt][rt]=min(Min[xrt][rt<<1],Min[xrt][rt<<1|1]);
}

void BuildY(int xrt,int x,int l,int r,int rt)
{
    int m;
    if(l==r)
    {
        if(x!=-1) Max[xrt][rt]=Min[xrt][rt]=mat[x][l];
        else
        {
            Max[xrt][rt]=max(Max[xrt<<1][rt],Max[xrt<<1|1][rt]);
            Min[xrt][rt]=min(Min[xrt<<1][rt],Min[xrt<<1|1][rt]);
        }
        return;
    }
    m=(l+r)>>1;
    BuildY(xrt,x,lson);
    BuildY(xrt,x,rson);
    PushUp(xrt,rt);
}

void BuildX(int l,int r,int rt)
{
    int m;
    if(l==r)
    {
        BuildY(rt,l,1,N,1);
        return;
    }
    m=(l+r)>>1;
    BuildX(lson);
    BuildX(rson);
    BuildY(rt,-1,1,N,1);
}

void UpdateY(int xrt,int x,int l,int r,int rt)
{
    int m;
    if(l==r)
    {
        if(x!=-1) Max[xrt][rt]=Min[xrt][rt]=val;
        else
        {
            Max[xrt][rt]=max(Max[xrt<<1][rt],Max[xrt<<1|1][rt]);
            Min[xrt][rt]=min(Min[xrt<<1][rt],Min[xrt<<1|1][rt]);
```

```
        }
        return;
    }
    m=(l+r)>>1;
    if(yL<=m) UpdateY(xrt,x,lson);
    else UpdateY(xrt,x,rson);
    PushUp(xrt,rt);
}

void UpdateX(int l,int r,int rt)
{
    int m;
    if(l==r)
    {
        UpdateY(rt,l,1,N,1);
        return;
    }
    m=(l+r)>>1;
    if(xL<=m) UpdateX(lson);
    else UpdateX(rson);
    UpdateY(rt,-1,1,N,1);
}


void QueryY(int xrt,int l,int r,int rt)
{
    int m;
    if(yL<=l&&yR>=r)
    {
        minv=min(minv,Min[xrt][rt]);
        maxv=max(maxv,Max[xrt][rt]);
        return;
    }
    m=(l+r)>>1;
    if(yL<=m) QueryY(xrt,lson);
    if(yR>m) QueryY(xrt, rson);
}

void QueryX(int l,int r,int rt)
{
    int m;
    if(xL<=l&&xR>=r)
    {
        QueryY(rt,1,N,1);
        return;
    }
    m=(l+r)>>1;
    if(xL<=m) QueryX(lson);
    if(xR>m) QueryX(rson);
}

int main()
{
    //freopen("in.txt","r",stdin);
    int i,j,q,cas,T,x,y,l;
    char op[5];
```

```
    scanf("%d",&T);
    for(cas=1;cas<=T;cas++)
    {
        scanf("%d",&N);
        for(i=1;i<=N;i++)
            for(j=1;j<=N;j++)
                scanf("%d",&mat[i][j]);
        BuildX(1,N,1);
        scanf("%d",&q);
        printf("Case #%d:\n",cas);
        while(q--)
        {
            scanf("%d%d%d",&x,&y,&l);
            l=(l+1)/2;
            xL=max(1,x-l+1),xR=min(N,x+l-1);
            yL=max(1,y-l+1),yR=min(N,y+l-1);
            minv=1<<30,maxv=-(1<<30);
            QueryX(1,N,1);
            val=(maxv+minv)/2;
            xL=x,yL=y;
            printf("%d\n",val);
            UpdateX(1,N,1);
        }
    }
    return 0;
}
```

4 字符串

4.1 哈希

```
/* hash */
/**备选素数 1572869, 3145739, 6291469, 12582917, 25165843, 50331653*/
const int maxn = "1e5";
const int seed=31;
ull h[maxn];
ull base[maxn];
typedef pair<int,int> P;
void init()
{
  base[0]=1;
  for(int i=1;i<maxn;i++)
    base[i]=base[i-1]*seed;
}
ull str_hash(int l,int r){
  return h[r]-h[l-1]*base[r-l+1];
}
void Hash()
 {
   for(int i=0;i<len;i++)
      h[i+1]=h[i]*seed+s[i]-'a'+1;
  }

  /* 随机数双哈希（csl）Gym101808B */
  map<pair<int,int>,pair<ull,ull> dic;

  inline pair<ull,ull>gethash(int x,int y)
  {
    if(x>y) swap(x,y);
    if(dic.find({x,y})!=dic.end())
        return dic[{x,y}];
    ull h1=1;
    ull h2=1;
    for(int i=0;i<5;i++) h1*=rand();
    for(int i=0;i<5;i++) h2*=rand();//用随机数hash
    return dic[{x,y}]={h1,h2};
  }

  map<pair<ull,ulll>,int> cnt[maxn];

   dic.clear();
  for(int i=0;i<n;i++)
    a[i]=gethash(x,y);

for(int i=0;i<n;i++)
{
  ull hash1=0,hash2=0;
  for(int j=i;j<n;j++)
  {
    hash1+=a[j].first;hash2+=a[j].second;
   ans+=cnt[j-i][{hash1,hash2}];
    cnt[j-i][{hash1,hash2}]++;//长度相同的放在一个map
  }
```

```
}

/* 双哈希（csl）*/

const int seed1= "19260817" ;
const int mod1= "1e9+7";
const int seed2= "23333333";
const int mod2= "1e9+9" ;
int base1[maxn],base2[maxn];

map<pair<int,int>,int> dic;
inline int getid(int x,int y)
{
  if(x>y) swap(x,y);
  if(dic[{x,y}])  return dic[{x,y}];
  return dic[{x,y}]=dic.size();
}

void init(int n)
{
  base1[0]=1,base2[0]=1;
  for(int i=1;i<=n;i++)
    base1[i]=(1LL*base1[i-1]*seed1);
    base2[i]=(1LL*base2[i-1]*seed2);
}

dic.clear();
for(int i=0;i<n;i++)
    a[i]=getid(x,y);

map<pair<int,int>,int> cnt[maxn];
for(int i=0;i<n;i++)
{
  int sum1=0,sum2=0;
  for(int j=i;j<n;j++)
  {
    sum1=(sum1+base1[a[j]])%mod1;
    sum2=(sum2+base2[a[j]])%mod2;
     ans+=cnt[j-i][{sum1,sum2}];
    cnt[j-i][{sum1,sum2}]++;//长度相同的放在一个map
  }
}

/*PROVIDE BY CSL**/
typedef unsigned long long ull;
const ull Seed_Pool[] = {146527, 19260817};
const ull Mod_Pool[] = {1000000009, 998244353};
struct Hash
{
    ull SEED, MOD;
    vector<ull> p, h;
    Hash() {}
    Hash(const string& s, const int& seed_index, const int& mod_index)
    {
        SEED = Seed_Pool[seed_index];
        MOD = Mod_Pool[mod_index];
```

```
        int n = s.length();
        p.resize(n + 1), h.resize(n + 1);
        p[0] = 1;
        for (int i = 1; i <= n; i++) p[i] = p[i - 1] * SEED % MOD;
        for (int i = 1; i <= n; i++) h[i] = (h[i - 1] * SEED % MOD + s[i - 1]) % MOD;
    }
    ull get(int l, int r) { return (h[r] - h[l] * p[r - l] % MOD + MOD) % MOD; }
    ull substr(int l, int m) { return get(l, l + m); }
};
```

4.2 KMP

```
const int maxn = " ";
int fail[maxn];
void getfail(char *x)
{
    int m=strlen(x);
    int i = 0, j = fail[0] = -1;
    while (i < m)
    {
        while (j != -1 && x[i] != x[j]) j = fail[j];
        fail[++i] = ++j;
    }
}
 //x是模式串，y是主串
// 返回y中x的个数
int kmp(char *x,char *y)
{
    int i, j, ans;
    i = j = ans = 0;
    getfail(x);
    int m=strlen(x);
    int n=strlen(y);
    while (i < n)
    {
        while (j != -1 && y[i] != x[j]) j = fail[j];
        i++, j++;
        if (j >= m) ans++, j = fail[j];
    }
    return ans;
}
```

4.3 扩展KMP

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <iostream>
#include <map>
using namespace std;
const int maxn=1e5+5;
struct exKMP
{
    char t[maxn];
    char p[maxn];
    int f[maxn];
```

```
    int extend[maxn];
    void getfail(char *p,int *f)
    {
        int m=strlen(p);
        f[0]=m;
        int i=0;
        while(i<m-1&&p[i]==p[i+1])
            i++;
        f[1]=i;
        int po=1;
        for(i=2;i<m;i++)
        {
            if(f[i-po]+i<po+f[po])
                f[i]=f[i-po];
            else
            {
                int j=po+f[po]-i;
                if(j<0)
                    j=0;
                while((i+j<m)&&p[i+j]==p[j])
                    j++;
                f[i]=j;
                po=i;
            }
        }
    }
    void getextend(char *t,char *p,int *f,int *extend)
    {
        int n=strlen(t);
        int m=strlen(p);
        getfail(p,f);
        int i=0;
        while(t[i]==p[i]&&i<n&&i<m)
            i++;
        extend[0]=i;
        int po=0;
        for(int i=1;i<n;i++)
        {
            if(f[i-po]+i<extend[po]+po)
                extend[i]=f[i-po];
            else
            {
                int j=extend[po]+po-i;
                if(j<0)
                    j=0;
                while(i+j<n&&j<m&&t[i+j]==p[j])
                    j++;
                extend[i]=j;
                po=i;
            }
        }
    }
}ans;
map<char,char> m;
int main()
{
```

```
    int T;
    scanf("%d",&T);
    char code[30];
    while(T--)
    {
        scanf("%s",code);
        scanf("%s",ans.t);
        int n=strlen(ans.t);
        for(int i=0;i<26;i++)
            m[code[i]]=i+'a';
        for(int i=0;i<n;i++)
        {
            ans.p[i]=m[ans.t[i]];
        }
        ans.p[n]='\0';
        ans.getfail(ans.p,ans.f);
        ans.getextend(ans.t,ans.p,ans.f,ans.extend);
        int i;
        for( i=0;i<n;i++)
            if(i+ans.extend[i]>=n&&i>=ans.extend[i])
            {
                break;
            }
        for(int j=0;j<i;j++)
            printf("%c",ans.t[j]);
        for(int j=0;j<i;j++)
            printf("%c",ans.p[j]);
        printf("\n");
    }
    return 0;
}
```

4.4 Manacher

```
/*
复杂度线性
加完特殊字符后，最长子串的长度是半径减1，起始位置是中间位置减去半径再除以2。
*/
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e6+5;
struct Manacher
{
    char p[maxn];
    char temp[maxn<<1];
    int f[maxn<<1];
    void init(char *p,char *temp)
    {
        int n=strlen(p);
        temp[0]='*';
        for(int i=0;i<=n;i++)
        {
            temp[i*2+1]='#';
            temp[i*2+2]=p[i];
        }
        temp[2*n+2]='\0';
```

```
    }
    void getlen(char *p,int *f)
    {
        int mx=0,po=0;
        int n=strlen(p);
        f[0]=0;
        for(int i=2;i<n;i++)
        {
            if(mx>i)
                f[i]=min(mx-i,f[2*po-i]);
            else
                f[i]=1;
            while(p[i-f[i]]==p[i+f[i]])
                f[i]++;
            if(f[i]+i>mx)
            {
                po=i;
                mx=f[i]+i;
            }
        }
    }
}ans;
int main()
{
    int kase=0;
    while(scanf("%s",ans.p)==1&&ans.p[0]!='E')
    {
        ans.init(ans.p,ans.temp);
        ans.getlen(ans.temp,ans.f);
        int n=strlen(ans.temp);
        int res=1;
        for(int i=2;i<n;i++)
        {
            res=max(res,ans.f[i]-1);
        }
        printf("Case %d: %d\n",++kase,res);
    }
    return 0;
}
```

4.5 01字典树

```
//HDU4835
struct Trie{
    int next[maxnode][2];
    ll val[maxnode];
    int root,cnt;
    int newnode()
    {
        next[cnt][0]=next[cnt][1]=-1;
        val[cnt++]=0;
        return cnt-1;
    }
    void init()
    {
        cnt=0;
```

```
        root=newnode();
    }
    void insert(ll x)
    {
        int now=root;
        for(int i=32;i>=0;i--)//attention
        {
           int id=((x>>i)&1);
           if(next[now][id]==-1)
                next[now][id]=newnode();

            now=next[now][id];
        }
        val[now]=x;
    }
    ll query(ll x)
    {
        int now=root;
        for(int i=32;i>=0;i--)
        {
            int id=((x>>i)&1);
            if(next[now][id^1]!=-1)
            {
                now=next[now][id^1];
            }
            else
            {
                now=next[now][id];
            }
        }
        return val[now];
     }

};
```

4.6 ac自动机

```
/****AC自动机 HUD-2222****/
/*以now节点结尾的后缀 与 root-fail[now]所表示的字符串 相同*/
const int NUMA=26;
struct trie{

   int next[maxnode][NUMA],fail[maxnode],ed[maxnode];//attention
   int root,cnt;
   int newnode(){
      for(int i=0;i<NUMA;i++)
        next[cnt][i]=-1;
      ed[cnt++]=0;
      return cnt-1;
   }

   void init(){
      cnt=0;
      root=newnode();
   }
```

```
    void inser(char* buf){
      int len=strlen(buf);
      int now=root;
      for(int i=0;i<len;i++){
          if(next[now][buf[i]-'a']==-1)
              next[now][buf[i]-'a']=newnode();
          now=next[now][buf[i]-'a'];
      }
      ed[now]++;
    }

    void build(){
      queue<int>que;
      fail[root]=root;
      for(int i=0;i<NUMA;i++){
         if(next[root][i]==-1)
              next[root][i]=root;
         else{
             fail[next[root][i]]=root;
             que.push(next[root][i]);
         }}
         while(!que.empty()){
             int now=que.front();
             que.pop();
             for(int i=0;i<NUMA;i++){
                 if(next[now][i]==-1)
                     next[now][i]=next[fail[now]][i];
                 else{
                     fail[next[now][i]]=next[fail[now]][i];
                     que.push(next[now][i]);
                 }
             }
         }
      }

    int query(char* buf){
      int len=strlen(buf);
      int now=root;
      int res=0;
      for(int i=0;i<len;i++){
          now=next[now][buf[i]-'a'];
          int temp=now;
          while(temp!=root){
              res+=ed[temp];
              ed[temp]=0;
              temp=fail[temp];
          }
      }
    return res;
    }
};
char buf[maxn];
trie ac;
int main(){
    int n;
    cin>>n;
```

```
    ac.init();
    for(int i=0;i<n;i++)
        cin>>buf,ac.inser(buf);
    ac.build();
    cin>>buf;
    cout<<ac.query(buf)<<endl;
return 0;
}
```

4.7 后缀数组

```
/*后缀数组 DA倍增算法 o(n*log(n))*/
const int maxn = "Edit";
char s[maxn];
int sa[maxn], t[maxn], t2[maxn], c[maxn], ran[maxn], height[maxn];
/*
sa为后缀数组，保存sa[第i个名次]=是i开头后缀
rank为名次数组rank[i开头的后缀]=的名次
height为相邻两个后缀的最长公共前缀
*/
//n为字符串的长度,字符集的值为0~m-1
/*
build(128,s.size())
height[ 2-s.size() ] 有效
height[i]为以sa[i-1]和sa[i]开头的后缀 的最长公共前缀
*/
void build_sa(int m, int n)
{
    n++;
    int *x = t, *y = t2;
    //基数排序
    for (int i = 0; i < m; i++) c[i] = 0;
    for (int i = 0; i < n; i++) c[x[i] = s[i]]++;
    for (int i = 1; i < m; i++) c[i] += c[i - 1];
    for (int i = n - 1; ~i; i--) sa[--c[x[i]]] = i;
    for (int k = 1; k <= n; k <<= 1)
    {
        //直接利用sa数组排序第二关键字
        int p = 0;
        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;
        //基数排序第一关键字
        for (int i = 0; i < m; i++) c[i] = 0;
        for (int i = 0; i < n; i++) c[x[y[i]]]++;
        for (int i = 0; i < m; i++) c[i] += c[i - 1];
        for (int i = n - 1; ~i; i--) sa[--c[x[y[i]]]] = y[i];
        //根据sa和y数组计算新的x数组
        swap(x, y);
        p = 1;
        x[sa[0]] = 0;
        for (int i = 1; i < n; i++)
            x[sa[i]] = y[sa[i - 1]] == y[sa[i]] &&
            y[sa[i - 1] + k] == y[sa[i] + k] ? p - 1 : p++;
        if (p >= n) break; //以后即使继续倍增，sa也不会改变，推出
        m = p;             //下次基数排序的最大值
```

```
    }
    n--;
    int k = 0;
    for (int i = 0; i <= n; i++) ran[sa[i]] = i;
    for (int i = 0; i < n; i++)
    {
        if (k) k--;
        int j = sa[ran[i] - 1];
        while (s[i + k] == s[j + k]) k++;
        height[ran[i]] = k;
    }
}

int dp[maxn][30];
void initrmq(int n)
{
    for (int i = 1; i <= n; i++)
        dp[i][0] = height[i];
    for (int j = 1; (1 << j) <= n; j++)
        for (int i = 1; i + (1 << j) - 1 <= n; i++)
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
}
int rmq(int l, int r)
{
    int k = 31 - __builtin_clz(r - l + 1);//__builtin_clz二进制中前导零的个数
    return min(dp[l][k], dp[r - (1 << k) + 1][k]);
}
int lcp(int a, int b)
{ // 求两个后缀的最长公共前缀
    a = ran[a], b = ran[b];
    if (a > b) swap(a, b);
    return rmq(a + 1, b);
}
```

4.8 后缀自动机

```
const int maxn = "";
struct SAM
{
    int len[maxn << 1], link[maxn << 1], ch[maxn << 1][26];
    int sz, rt, last;
    int newnode(int x = 0)
    {
        len[sz] = x;
        link[sz] = -1;
        mem(ch[sz], -1);
        return sz++;
    }
    void init() { sz = last = 0, rt = newnode(); }
    void extend(int c)
    {
        int np = newnode(len[last] + 1);
        int p;
        for (p = last; ~p && ch[p][c] == -1; p = link[p]) ch[p][c] = np;
        if (p == -1)
            link[np] = rt;
```

```
        else
        {
            int q = ch[p][c];
            if (len[p] + 1 == len[q])
                link[np] = q;
            else
            {
                int nq = newnode(len[p] + 1);
                memcpy(ch[nq], ch[q], sizeof(ch[q]));
                link[nq] = link[q], link[q] = link[np] = nq;
                for (; ~p && ch[p][c] == q; p = link[p]) ch[p][c] = nq;
            }
        }
        last = np;
      }

    int topcnt[maxn], topsam[maxn << 1];
    void sort()
    { // 加入串后拓扑排序
        mem(topcnt, 0);
        for (int i = 0; i < sz; i++) topcnt[len[i]]++;
        for (int i = 0; i < maxn - 1; i++) topcnt[i + 1] += topcnt[i];
        for (int i = 0; i < sz; i++) topsam[--topcnt[len[i]]] = i;
    }

};
```

4.9 回文自动机

```
/*回文自动机*/
const int N = 100005;
struct Palindromic_Tree
{
    int ch[N][26], f[N], cnt[N], len[N], s[N];
    int last, sz, n;
    int newnode(int x)
    {
        clr(ch[sz], 0);
        cnt[sz] =  0, len[sz] = x;
        return sz++;
    }
    void init()
    {
        sz = 0;
        newnode(0), newnode(-1);
        last = 0, n = 0, s[0] = -1, f[0] = 1;
    }
    int get_fail(int u)
    {
        while (s[n - len[u] - 1] != s[n]) u = f[u];
        return u;
    }
    void add(int c)
    {
        s[++n] = c;
        int u = get_fail(last);
```

```
        if (!ch[u][c])
        {
            int np = newnode(len[u] + 2);
            f[np] = ch[get_fail(f[u])][c];
            ch[u][c] = np;
        }
        last = ch[u][c];
        cnt[last]++;
    }
    void count()
    {
        for (int i = sz - 1; ~i; i--) cnt[f[i]] += cnt[i];
    }
};
```

5 优化算法

5.1 二分

```
/**upper_bound **/

 while(l<=r)
 {
   int mid=(l+r)/2;
   if(ok)  l=mid+1;
   else  r=mid-1;
 }
  return l;


/**lower_bound **/
 int lb=-1,ub=res;
  while(ub-lb>1)
  {
    int mid=(lb+ub)/2;
    if(ok) ub=mid;
    else lb=mid;
  }
  return ub;

/*另一种好用的lower_bound*/
int ans=0;
while(l<=r)
{
  int mid=(l+r)/2;
  if(ok) l=mid+1,ans=mid;
  else r=mid-1;
}
return ans;

/**浮点数二分*/
 for(int i = 0; i < 100; i++)
 {
      double mid = (l + r) / 2.0;
      if(check(mid)) r = mid;
      else l = mid;
  }
```

5.2 数位DP

```
/* 数位DP HDU-2089 不要62*/
int bit[30];
ll dp[30][2];
ll dfs(int pos,int st,int flag)
{
   if(pos==0) return 1;
   if(flag&&dp[pos][st]!=-1)
      return dp[pos][st];
   int u=flag?9:bit[pos];
   ll ans=0;
   for(int i=0;i<=u;i++){
```

```
      if(i==4) continue;
      else if(st==0&&i==2) continue;
      else if(i!=6) ans+=dfs(pos-1,1,flag||i<u);
      else if(i==6) ans+=dfs(pos-1,0,flag||i<u);
   }
   if(flag) dp[pos][st]=ans;
return ans;
}

ll solve(int n){
  int len=0;
  while(n){
     bit[++len]=n%10;//len=1为最低位
     n/=10;
  }
  return dfs(len,1,0);
}
```

5.3 树上启发式合并

```
//CF 600E
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+10;
vector<int> G[maxn];
int v[maxn];
int sz[maxn];
int f[maxn];
int son[maxn];
int vis[maxn];
int cnt[maxn];
typedef long long ll;
ll ans[maxn];
ll sum;
ll mx;
void lh(int u,int fa)
{
    sz[u]=1;
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i];
        if(v==fa)
            continue;
        lh(v,u);
        sz[u]+=sz[v];
        if(son[u]==0||sz[v]>sz[son[u]])
        {
            son[u]=v;
        }
    }
}
void add(int u,int fa,int val)
{
    cnt[v[u]]+=val;
    if(cnt[v[u]]>mx)
    {
```

```
        sum=v[u],mx=cnt[v[u]];
    }
    else if(cnt[v[u]]==mx)
    {
        sum+=v[u];
    }
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i];
        if(v!=fa&&vis[v]==0)
            add(v,u,val);
    }
}
void dfs(int u,int fa,int flag)
{
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i];
        if(v!=fa&&v!=son[u])
            dfs(v,u,0);
    }
    if(son[u])
        dfs(son[u],u,1),vis[son[u]]=1;
    add(u,fa,1);
    ans[u]=sum;
    if(son[u])
        vis[son[u]]=0;
    if(flag==0)
        add(u,fa,-1),sum=0,mx=-1;
}
int main()
{
    int N;
    scanf("%d",&N);
    for(int i=1;i<=N;i++)
        scanf("%d",&v[i]);
    for(int i=1;i<=N-1;i++)
    {
        int u,v;
        scanf("%d%d",&u,&v);
        G[u].push_back(v);
        G[v].push_back(u);
    }
    lh(1,0);
    dfs(1,0,1);
    for(int i=1;i<=N;i++)
        printf("%lld ",ans[i]);
    printf("\n");
    //scanf("%d",&N);
    return 0;
}
```

5.4 树上点分治

```
/*树上点分治*/
#include<bits/stdc++.h>
```

```
using namespace std;
typedef long long ll;
const int maxn=1e4+5;
struct Edge
{
    int to,cost;
};
vector<Edge> G[maxn];
int N,K,root;
int vis[maxn];
int sz[maxn],maxson[maxn];
int dep[maxn];
int total;
ll ans;
vector<int> deep;
void getroot(int u,int fa)
{
    sz[u]=1,maxson[u]=0;
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i].to;
        if(v==fa||vis[v])
            continue;
        getroot(v,u);
        sz[u]+=sz[v];
        maxson[u]=max(maxson[u],sz[v]);
    }
    maxson[u]=max(maxson[u],total-sz[u]);
    if(maxson[u]<maxson[root])
        root=u;
}
void getdeep(int u,int fa)
{
    deep.push_back(dep[u]);
    sz[u]=1;
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i].to;
        if(v==fa||vis[v])
            continue;
        dep[v]=dep[u]+G[u][i].cost;
        getdeep(v,u);
        sz[u]+=sz[v];
    }
}
ll cal(int u,ll init)
{
    deep.clear();
    dep[u]=init;
    getdeep(u,0);
    sort(deep.begin(),deep.end());
    ll res=0;
    for(int l=0,r=deep.size()-1;l<r;)
    {
        if(deep[l]+deep[r]<=K)
        {
```

```
            res+=r-l;
            l++;
        }
        else
            r--;
    }
    return res;
}
void solve(int u)
{
    ans+=cal(u,0);
    vis[u]=1;
    for(int i=0;i<G[u].size();i++)
    {
        int v=G[u][i].to;
        if(!vis[v])
        {
            ans-=cal(v,G[u][i].cost);
            maxson[0]=total=sz[v];
            getroot(v,root=0);
            solve(root);
        }
    }
}
int main()
{
    while(scanf("%d%d",&N,&K)!=EOF,N+K)
    {
        memset(vis,0,sizeof(vis));
        for(int i=1;i<=N;i++)
            G[i].clear();
        for(int i=1;i<=N-1;i++)
        {
            int u,v,c;
            scanf("%d%d%d",&u,&v,&c);
            G[u].push_back({v,c});
            G[v].push_back({u,c});
        }
        root=0;
        maxson[root]=N;
        getroot(1,root);
        ans=0;
        solve(root);
        printf("%lld\n",ans);
    }
    return 0;
}
```

5.5 莫队算法

```
/*
莫队算法复杂度 O(n*sqrt(n))
NBUT-1457
*/
ll lastans;
int block;
```

```
struct node{
 int l,r,id;
 int pos;//分块
 void init(){
  pos=l/block;
 }
 bool operator<(const node &a)const{
   if(pos==a.pos) return r<a.r;
   return pos<a.pos;
 }
}q[maxn];

void addl(){

}
void dell(){

}
void addr(){

}
void delr(){

}

  block=sqrt(n+0.5);//
  for(int i=1;i<=m;i++)
  {
    scanf("%d%d",&q[i].l,&q[i].r);
    q[i].id=i;
    q[i].init();
   }
   sort(q+1,q+m+1);
  int lastl=2,lastr=1;
   lastans=0;
    for(int i=1;i<=m;i++)
     {
        while(lastl>q[i].l) dell(--lastl);
        while(lastr<q[i].r) addr(++lastr);
        while(lastl<q[i].l) addl(lastl++);
        while(lastr>q[i].r) delr(lastr--);
       ans[q[i].id]=lastans;
     }
  for(int i=1;i<=m;i++)
    printf("%lld\n",ans[i]);
```

5.6 单调栈单调队列笛卡尔树

```
/*单调栈单调队列和笛卡尔树*/
/** 单调栈 cf602D **/
/*找出左侧第一个大于它的数*/
 int top=-1;
 for(int i=1;i<=n;i++)
  {
```

```
     while(top>=0&&h[i]>=h[st[top]]) //delete the elem in stack no more larger than i
        top--;
     if(top==-1)
            //all the elem in the left no more larger than i
      else
            //the nearest left elem larger than i

    st[++top]=i;//add i to the stack
  }

/*单调队列 POJ - 2823 */
/*求长度为k的区间最小（大）值，或长度不超过k的区间最小（大）值*/
/*求长度为k的区间最小值*/
    int top=-1;
    for(int i=1;i<=k;i++)
    {
        while(top>=0&&a[i]<=a[que[top]])
            top--;

        que[++top]=i;
        b[0]=a[que[0]];
    }

    int s=0;
    for(int i=k+1;i<=n;i++)
    {
        if(que[s]==i-k) s++;
        while(top>=s&&a[i]<=a[que[top]])
            top--;

        que[++top]=i;
        b[i-k]=a[que[s]];
    }

/**笛卡尔树**/
/**
 * 中序遍历得到的序列为原数组序列
 * 节点的key值要大于其左右子节点的key值
 * 利用单调栈建树
 **/
void build() {
        int top=0;
        for(int i=1;i<=n;i++)
      l[i]=0,r[i]=0,vis[i]=0;
        for(int i=1;i<=n;i++)
     {
                int k=top;
                while (k>0&&a[stk[k-1]]<a[i]) --k;
                if (k) r[stk[k-1]]=i;//找出i左边第一个比它大的数，把i连到它的右子树
                if (k<top) l[i]=stk[k];//将该数字原来的右子树连到i的左子树
                stk[k++]=i;
                top=k;
        }
    for(int i=1;i<=n;i++)
           vis[l[i]]=vis[r[i]]=1;
        int rt=0;
```

```
    for(int i=1;i<=n;i++)
       if (vis[i]==0)  rt=i;//find the root
}
```

5.7 最长上升子序列

```
/*最长上升子序列 o(nlogn) */
const int maxn= " ";
int a[maxn],b[maxn];

int lis(int n) {
    b[1]=a[1];
    int len=1;
    for(int i=2;i<=n;i++)
    {
        if(a[i]>=b[len])
                 {
             len=len+1;
             b[len]=a[i];
         }
        else
        {
            int pos=upper_bound(b+1,b+1+len,a[i])-b;
            b[pos]=a[i];
                }
    }
  return len;
  }
```

5.8 斜率优化DP

学习一:

四边形不等式优化DP:

形如 dp(i,j)=min{dp(i,k)+dp(k,j)+w(i,j)}

只需证明

w为满足四边形不等式,即w(i,j)+w(i+1,j+1)<=w(i+1,j)+w(i,j+1)

即证明 f(j)=w(i+1,j)-w(i,j) 单调递减

即证明 $w(i_2,j)$ <= $w(i,j_2)$ $i <= i_2 < j <= j_2$

可以证明若w满足四边形不等式，则dp也满足四边形不等式，则

定义s(i,j)=maxdp(i,j),s(i,j)具有单调性

即 s(i,j)<=s(i,j+1)<=s(i+1,j+1)

也写为 s(i,j-1)<=s(i,j)<=s(i+1,j)

四边形不等式优化可以将复杂度从$o(n^3)$降为$o(n^2)$

学习二:

- $f[x] = min_{i=1}^{x-1}\{f[i] + w[i,x]\}$ 证明w(i,j)+w(i+1,j+1)<=w(i+1,j)+w(i,j+1)，即满足决策单调性 可以用栈维护，然后二分$o(nlogn)$
- 若w为一个前缀和，即w[i,j]+w[j,k]=w[i,k],则可以使用单调队列优化$o(n)$ 即为$f[x] = min_{k=b[x]}^{x-1}\{g[k]\}+w[x]$,b[x]不降 单调队列求固定长度区间的最大最小值也可以转换为这个模型
- 相当于简单的斜率优化 $dp[i] = a[j] + b(i,j) + c$

  a[j]:只与j有关 b(i,j):与i,j同时有关

  证明j>k时,且$(f[j] - f[k])/(s[j] - s[k]) < c * s[i]$ 成立时,j比k优

  f[j],f[k]为前面算出的答案，s[x]只与x有关 去掉没有用的点，即维护一个凸包。

- 当s[i]具有单调性时，可以用单调队列维护。
  $(f[j]-f[k])/(s[j]-s[k])<c*s[i]$，删去点k(删头)
  k<j<i,若$k_{ij}<k_{kj}$，则删去点j（删尾)

```
//HDU3507
#include<bits/stdc++.h>
#define pb(x) push_back(x)
#define fir first
#define sec second
#define mem(a,x) memset(a,x,sizeof(a))
#define mpr make_pair
typedef long long ll;
using namespace std;
const int inf=0x3f3f3f3f;
const ll INF= 0x3f3f3f3f3f3f3f3f;
const double pi = acos(-1.0);
const int maxn=500100;
int que[maxn];
ll dp[maxn];
ll sum[maxn];
int n,m;
ll c[maxn];
ll getup(int j,int k)
{
    return dp[j]+sum[j]*sum[j]-(dp[k]+sum[k]*sum[k]);
}
ll getdown(int j,int k)
{
  return 2*(sum[j]-sum[k]);
}
ll getdp(int i,int j)
{
     return dp[j]+m+(sum[i]-sum[j])*(sum[i]-sum[j]);
}
int main(){
    while(scanf("%d%d",&n,&m)==2)
    {
        for(int i=1;i<=n;i++)
          scanf("%lld",&c[i]),sum[i]=sum[i-1]+c[i];
       int s=0;int t=-1;
       sum[0]=dp[0]=0;
       que[++t]=0;
       for(int i=1;i<=n;i++)
       {
           while(s<t&&getup(que[s+1],que[s])<=sum[i]*getdown(que[s+1],que[s]))
             s++;
           dp[i]=getdp(i,que[s]);
           while(s<t&&getup(i,que[t])*getdown(que[t],que[t-1])<=
           getup(que[t],que[t-1])*getdown(i,que[t]))
              t--;
           que[++t]=i;
       }
       printf("%d\n",dp[n]);
    }
 return 0;
  }
```

6 不会数学

6.1 快速幂与矩阵快速幂

```
/* 快速幂 */
ll quickmod(ll a,ll b,ll c){
        ll res=1;
        while(b){
                if(b&1)
                  res=res*a%c;
                a=a*a%c;
                b>>=1;
        }
        return res;
}

/****矩阵快速幂***/
typedef vector<ll> vec;
typedef vector<vec> mat;
const ll mod=" ";
//A*B
mat mul(mat &A,mat &B){
        mat C(A.size(),vec(B[0].size()));
        for(int i=0;i<A.size();i++)
          for(int k=0;k<B.size();k++)
            for(int j=0;j<B[0].size();j++)
              C[i][j]=(C[i][j]+A[i][k]*B[k][j])%mod;
        return C;
}
//A^n
mat pow(mat A,ll n){
        mat B(A.size(),vec(A.size()));
        for(int i=0;i<A.size();i++)
          B[i][i]=1;
        while(n>0){
                if(n&1) B=mul(B,A);
                A=mul(A,A);
                n>>=1;
        }
        return B;
}
ll n;
void solve(){
        mat A(2,vec(2));//行数，列数
        A[0][0]=1;A[0][1]=1;
        A[1][0]=1;A[1][1]=0;
        A=pow(A,n);
        cout<<A[1][0]<<endl;
}
```

6.2 欧几里得与逆元

```
ll gcd(ll a,ll b){
    if(a < b)
         swap(a,b);
    while(a%b){
```

```
        ll r = a % b;
        a = b;
        b = r;
     }
   return b;
}

/*求ax+by=c的xy 其中c%gcd(a,b)==0时才有解
解出的为特解x0,y0 */
ll ex_gcd(ll a,ll b,ll &x,ll &y)
{
    if(a==0&&b==0) return -1;
    if(b==0){x=1;y=0;return a;}
    ll d=ex_gcd(b,a%b,y,x);
    y-=a/b*x;
    return d;
}

/*求最小正整数解x;若无解返回false*/
/*已求出特解 x0,y0 设t=c/d,则通解为 x=t*x0+b/d*k; y=t*y0-a/d*k; k为任意整数
最小正整数解 x=x0*t; x=(x%s+s)%s;*/
  bool min_x(int a,int b,int &x,int &y,int c)
{
    int d=ex_gcd(a,b,x,y);
    if(c%d) return false;
    x=x*c/d;
    int s=b/d;
    s=s>0?s:-1*s;
    x=(x%s+s)%s;
    y=(c-a*x)/b;
    return true;
}

/****费马小定理求逆元****/
/*mod为素数,而且a和m互质 a^(p-1)=1(mod p) */
ll inv(ll a,ll mod) {
    return quickmod(a,mod-2,mod);
}

/*线性递推求逆元*/
void init()
{
    inv[1]=1;
   for(int i=2;i<=1e6;i++)
   inv[i]=inv[mod%i]*(mod-mod/i)%mod;

}
```

6.3 欧拉函数

```
/***筛法欧拉函数***/
/*
1. i mod p==0 phi(i * p) == p * phi(i)
2. i mod p!=0 phi(i * p) == phi(i) * (p-1)
算法就是将 X 中的素数因子pi减1，并且相同的素数因子只有1个减1
*/
```

```
const int maxn= "3e5+10";
int euler[maxn];
void getEuler() {
    memset(euler,0,sizeof(euler));
    euler[1] = 1;
    for(int i = 2;i < maxn;i++)
        if(!euler[i])
          for(int j = i;j < maxn; j += i){
            if(!euler[j])
              euler[j] = j;
            euler[j] = euler[j]/i*(i-1);
           }
 }

/*求单个数的欧拉函数*/
/*
phi(n)=n*((u(d)/d)之和d|n)
*/
ll eular(ll n) {
    ll ans = n;
    for(int i = 2;i*i <= n;i++)
     {
        if(n % i == 0)
        {
          ans -= ans/i;
          while(n % i == 0)
            n /= i;
         }
    }
  if(n > 1)  ans -= ans/n;
   return ans;
   }

/*
其他结论:
 phi(mn)=d*phi(m)*phi(n)/pi(d)    d=(m,n)
*/
```

6.4 素数与质因子

```
/* 素数判断 */
ull mul(ull x,ull y,ull Z)
{
    ull tmp=x/(long double)Z*y+1e-3;
    return (x*y+Z-tmp*Z)%Z;
}
ull MUL(ull x,ull p,ull Z)
{
    ull y=1;
    while(p)
    {
        if(p&1)y=mul(y,x,Z);
        x=mul(x,x,Z);
        p>>=1;
    }
    return y;
```

```
}
bool miuller_rabin(ull n)
{
    if(n<=1)return 0;
    if(n==2)return 1;
    if(n%2==0)return 0;
    ull p=n-1;
    srand(time(NULL));
    int TIMES=8;
    for(int i=1;i<=TIMES;i++)
    {
        ull x=rand()%(n-1)+1;
        if(MUL(x,p,n)!=1)return 0;
    }
    return 1;
}

/* 埃式筛 */
/*值为false表示素数，值为true表示非素数 */
/* 复杂度O(n*log(logn)) */

const int MAXN = " 1e6 ";
bool notprime[MAXN];
void init()
 {
    memset(notprime,false,sizeof(notprime));
    notprime[0]=notprime[1]=true;
    for(int i=2;i<MAXN;i++)
        if(!notprime[i])
        {
            if(i>MAXN/i)  continue;//防爆 long long
          for(int j=i*i;j<MAXN;j+=i)
             notprime[j]=true;
      }
}

/* 区间内素数筛选 埃式筛 */
/*prime[0]存素数个数，prime[i]为1-MAXN间第i个素数*/
/* hash[i]=flase 表示i为素数 */
/* 每个合数都只会被其最小质因子筛到 复杂度线性 */
const int MAXN = "1e7";
bool has[MAXN+1];
int prime[MAXN/10+1];
void getPrime()
{
    memset(prime,0,sizeof(prime));
    memset(has,false,sizeof(has));
    for(int i=2;i<=MAXN;i++)
    {
        if(!has[i]) prime[++prime[0]]=i;
        for(int j=1;j<=prime[0]&&prime[j]<=MAXN/i;j++)
        {
            has[prime[j]*i]=true;
            if(i%prime[j]==0)
                break;
        }
```

```
    }
}

/*合数分解*/
/* fat[i][0]表示第i个质因子，fat[i][1]表示该质因子的个数 */
ll fat[100][2];
int fcnt;
int getfats(ll x)
{
    fcnt=0;
    ll tmp=x;
    for(int i=1;prime[i]<=tmp/prime[i];i++)
    // if x is larger than 1e7,may need add condition i<prime[0]
    {
        fat[fcnt][1]=0;
        if(tmp%prime[i]==0)
        {
            fat[fcnt][0]=prime[i];
            while(tmp%prime[i]==0)
             {
            fat[fcnt][1]++;
            tmp/=prime[i];
             }
            fcnt++;
        }
    }
       if(tmp!=1)
       {
         fat[fcnt][0]=tmp;
         fat[fcnt++][1]=1;
        }
   return fcnt;
}

/* 预处理1-n所有数的质因子 接近o(n) */
const int maxn = " ";
vector<int>d[maxn];
int vis[maxn];
void init(){
    memset(vis,0,sizeof(vis));
   for(int i=2;i<maxn;i++){
      if(!vis[i]){
        for(int j=i;j<maxn;j+=i)
        {
            d[j].push_back(i);
            vis[j]=1;
        }
      }}
}
```

6.5 组合数学与容斥原理

```
/* 容斥原理  UVA-10325 */
for(int i=1;i<(1<<m);i++){
    ll ans=1;int ant=0;
    for(int j=0;j<m;j++){
```

```
        if(i&(1<<j)){
           ans=lcm(ans,a[j]);
            ant++;
            }
         }
            if((ant-1)%2) num-=(n/ans);
           else num+=(n/ans);
}

/* 组合数预处理 */
/* c(n,m) n>m */
const int N=50;
for(int j=0;j<N;j++)
    c[j][0]=1;
for(int i=0;i<N;i++)
   for(int j=1;j<N;j++)
    {
     if(i==j) c[i][j]=1;
      else if(i<j) c[i][j]=0;
      else c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
    }

/*Lucas 组合数取模*/
/* 模数p<=1e5 */
const int maxn="1e5";
ll fac[maxn];
void getf(ll p)
{
    fac[0]=1;
    for(int i=1;i<=p;i++)
        fac[i]=fac[i-1]*i%p;
}

ll quickmod(ll a,ll b,ll c){
        ll res=1;
        while(b){
                if(b&1)
                  res=res*a%c;
                a=a*a%c;
                b>>=1;
        }
        return res;
}

ll lucas(ll n,ll m,ll p)
{
    ll ans=1;
    while(n&&m)
    {
        ll a=n%p;
        ll b=m%p;
        if(a<b) return 0;
        ans=(ans*fac[a]*quickmod(fac[b]*fac[a-b]%p,p-2,p))%p;
        n/=p;
        m/=p;
    }
```

```
    return ans;
}
```

6.6 中国剩余定理

```
/*中国剩余定理*/
/*x=bi(mod m0i)*/
/*x0+=biMiyi*/
ll m0[maxn];
ll b[maxn];
ll ChinaRemainder(int n){
        ll m=1,a=0;
        for (int i=1; i<=n; i++)  m=m*m0[i];
        for (int i=1; i<=n; i++) {
                ll MM=m/m0[i];
                ll x=inv(MM,m0[i]);
                a=(a+MM*x*b[i]) % m;
        }
        return a;
}
/*不互质的中国剩余定理*/
ll m0[maxn];
ll b[maxn];

ll china(ll n)
{
    ll a,bb,d,x,y,dm;
    ll c,c1,c2;
    ll dg;//lcm
    a=m0[1]; c1=b[1];
    for (int i=2; i<=n; i++)
    {
        bb=m0[i]; c2=b[i];
        d=ex_gcd(a, bb,x, y);
        dm=bb/d;
        c=c2-c1;
        if (c%d) return -1;//无解
        x=((x*c/d)%dm+dm)%dm;//x可能为负
        c1=a*x+c1;
        a=a*bb/d;
    }
    dg=a;//dg是最大公约数
    if (!c1)//考虑c1为0的情况
    {
        c1=1;
        for (int i=1; i<=n; i++)
        {
            c1=c1*m0[i]/__gcd(c1, m0[i]);
        }
        dg=c1;//此时dg为最小公倍数
    }
    return c1;//c1为最小的X
}
```

6.7 FFT

```
typedef complex<double> cd;
void fft(cd *a, int n,int f) {
   for(int i = 0;i < n;i ++)
         if(i < rev[i])
             swap(a[i], a[rev[i]]);
     for(int i = 1;i < n;i <<= 1) {
         cd wn(cos(pi / i), sin(f * pi / i)), x, y;
         for(int j = 0;j < n;j += (i << 1)) {
             cd w(1, 0);
             for(int k = 0;k < i;k ++, w *= wn) {
                 x = a[j + k], y = w * a[i + j + k];
                 a[j + k] = x + y;
                 a[i + j + k] = x - y;
             }
         }
     }
 }

  for(N = 1;N <= (M + 1) * 2;N <<= 1) bit++; //bit为数组的大小

  for(int i = 0;i < N;i ++)
      rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (bit - 1));

  for(int i=0;i<N;i++)
      d[i]=(ll)(c[i].real()/N+0.5);//四舍五入
```

7 计算几何

Thanks for csl.

7.1 点的定义

```
#define zero(x) ((fabs(x) < eps ? 1 : 0))
#define sgn(x) (fabs(x) < eps ? 0 : ((x) < 0 ? -1 : 1))

struct point
{
    double x, y;
    point(double a = 0, double b = 0) { x = a, y = b; }
    point operator-(const point& b) const { return point(x - b.x, y - b.y); }
    point operator+(const point& b) const { return point(x + b.x, y + b.y); }
    // 两点是否重合
    bool operator==(point& b) { return zero(x - b.x) && zero(y - b.y); }
    // 点积(以原点为基准)
    double operator*(const point& b) const { return x * b.x + y * b.y; }
    // 叉积(以原点为基准)
    double operator^(const point& b) const { return x * b.y - y * b.x; }
    // 绕P点逆时针旋转a弧度后的点
    point rotate(point b, double a)
    {
        double dx, dy;
        (*this - b).split(dx, dy);
        double tx = dx * cos(a) - dy * sin(a);
        double ty = dx * sin(a) + dy * cos(a);
        return point(tx, ty) + b;
    }
    // 点坐标分别赋值到a和b
    void split(double& a, double& b) { a = x, b = y; }
};
struct line
{
    point s, e;
    line() {}
    line(point ss, point ee) { s = ss, e = ee; }
};
```

7.2 点直线与线段之间的位置关系

```
/**点直线与线段之间的位置关系*/

/*两点间距离*/
double dist(point a, point b) { return sqrt((a - b) * (a - b)); }


/*两条直线之间的关系*/
// <0, *> 表示重合; <1, *> 表示平行; <2, P> 表示交点是P;
pair<int, point> spoint(line l1, line l2)
{
    point res = l1.s;
    if (sgn((l1.s - l1.e) ^ (l2.s - l2.e)) == 0)
        return {sgn((l1.s - l2.e) ^ (l2.s - l2.e)) != 0, res};
    double t = ((l1.s - l2.s) ^ (l2.s - l2.e)) / ((l1.s - l1.e) ^ (l2.s - l2.e));
```

```
    res.x += (l1.e.x - l1.s.x) * t;
    res.y += (l1.e.y - l1.s.y) * t;
    return {2, res};
}


/**判断线段相交*/
bool segxseg(line l1, line l2)
{
    return
        max(l1.s.x, l1.e.x) >= min(l2.s.x, l2.e.x) &&
        max(l2.s.x, l2.e.x) >= min(l1.s.x, l1.e.x) &&
        max(l1.s.y, l1.e.y) >= min(l2.s.y, l2.e.y) &&
        max(l2.s.y, l2.e.y) >= min(l1.s.y, l1.e.y) &&
        sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e-l1.e) ^ (l1.s - l1.e)) <= 0 &&
        sgn((l1.s - l2.e) ^ (l2.s - l2.e)) * sgn((l1.e-l2.e) ^ (l2.s - l2.e)) <= 0;
}

/*直线与线段相交*/
//l1是直线,l2是线段
bool segxline(line l1, line l2)
{
    return sgn((l2.s - l1.e) ^ (l1.s - l1.e)) * sgn((l2.e - l1.e) ^ (l1.s - l1.e)) <= 0;
}

/**点与直线相交*/
double pointtoline(point p, line l)
{
    point res;
    double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
    res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
    return dist(p, res);
}

/**点与线段相交*/
double pointtosegment(point p, line l)
{
    point res;
    double t = ((p - l.s) * (l.e - l.s)) / ((l.e - l.s) * (l.e - l.s));
    if (t >= 0 && t <= 1)
        res.x = l.s.x + (l.e.x - l.s.x) * t, res.y = l.s.y + (l.e.y - l.s.y) * t;
    else
        res = dist(p, l.s) < dist(p, l.e) ? l.s : l.e;
    return dist(p, res);
}

/**点是否在线段上*/
bool PointOnSeg(point p, line l)
{
    return
        sgn((l.s - p) ^ (l.e-p)) == 0 &&
        sgn((p.x - l.s.x) * (p.x - l.e.x)) <= 0 &&
        sgn((p.y - l.s.y) * (p.y - l.e.y)) <= 0;
}
```

7.3 多边形

```
/*多边形面积计算*/
double area(point p[], int n)
{
    double res = 0;
    for (int i = 0; i < n; i++) res += (p[i] ^ p[(i + 1) % n]) / 2;
    return fabs(res);
}

/*判断点是否在凸多边形内*/
// 点形成一个凸包，而且按逆时针排序(如果是顺时针把里面的<0改为>0)
// 点的编号 : [0,n)
// -1 : 点在凸多边形外
// 0  : 点在凸多边形边界上
// 1  : 点在凸多边形内
int PointInConvex(point a, point p[], int n)
{
    for (int i = 0; i < n; i++)
        if (sgn((p[i] - a) ^ (p[(i + 1) % n] - a)) < 0)
            return -1;
        else if (PointOnSeg(a, line(p[i], p[(i + 1) % n])))
            return 0;
    return 1;
}

/*判断凸多边形*/
//点可以是顺时针给出也可以是逆时针给出
//点的编号1~n-1
bool isconvex(point poly[], int n)
{
    bool s[3];
    memset(s, 0, sizeof(s));
    for (int i = 0; i < n; i++)
    {
        s[sgn((poly[(i + 1) % n] - poly[i]) ^ (poly[(i + 2) % n] - poly[i])) + 1] = 1;
        if (s[0] && s[2]) return 0;
    }
    return 1;
}
```

7.4 求圆的外心

```
/*求圆的外心*/

point waixin(point a, point b, point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    return point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 - a2 * c1) / d);
}
```

7.5 求整点数

```
/*求整点数*/
```

```
/*线段上整点数*/
int OnSegment(line l) { return __gcd(fabs(l.s.x - l.e.x), fabs(l.s.y - l.e.y)) + 1; }


/*多边形边上整点数*/
int OnEdge(point p[], int n)
{
    int i, ret = 0;
    for (i = 0; i < n; i++)
        ret += __gcd(fabs(p[i].x - p[(i + 1) % n].x), fabs(p[i].y - p[(i + 1) % n].y));
    return ret;
}

/*多边形内部整点数*/
int InSide(point p[], int n)
{
    int i, area = 0;
    for (i = 0; i < n; i++)
        area += p[(i + 1) % n].y * (p[i].x - p[(i + 2) % n].x);
    return (fabs(area) - OnEdge(p, n)) / 2 + 1;
}
```

刘汝佳版

7.6 刘汝佳版点的定义

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

typedef Point Vector;

//向量+向量=向量，点+向量=点
Vector operator+(Vector A, Vector B) { return Vector(A.x + B.x, A.y + B.y); }
//点-点=向量
Vector operator-(Point A, Point B) { return Vector(A.x - B.x, A.y - B.y); }
//向量*数=向量
Vector operator*(Vector A, double p) { return Vector(A.x * p, A.y * p); }
//向量/数=向量
Vector operator/(Vector A, double p) { return Vector(A.x / p, A.y / 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;
double 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) == 0;
}

/*
 * 基本运算:
 * 点积
 * 叉积
 * 向量旋转
 */
double Dot(Vector A, Vector B) { return A.x * B.x + A.y * B.y; }
double Length(Vector A) { return sqrt(Dot(A, A)); }
double Angle(Vector A, Vector B) { return acos(Dot(A, B) / Length(A) / Length(B)); }

double Cross(Vector A, Vector B) { return A.x * B.y - A.y * B.x; }
double Area2(Point A, Point B, Point C) { return Cross(B - A, C - A); }

//rad是弧度
Vector Rotate(Vector A, double rad)
{
    return Vector(A.x * cos(rad) - A.y * sin(rad),
                  A.x * sin(rad) + A.y * cos(rad));
}

//调用前请确保A不是零向量
Vector Normal(Vector A)
{
    double L = Length(A);
    return Vector(-A.y / L, A.x / L);
}

/*
 * 点和直线:
 * 两直线交点
 * 点到直线的距离
 * 点到线段的距离
 * 点在直线上的投影
 * 线段相交判定
 * 点在线段上判定
 */

//调用前保证两条直线P+tv和Q+tw有唯一交点。当且仅当Cross(v, w)非0
Point GetLineIntersection(Point P, Vector v, Point Q, Vector w)
{
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}

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

```
double DistanceToSegment(Point P, Point A, Point B)
{
    if (A == B) return Length(P - A);
    Vector v1 = B - A, v2 = P - A, v3 = P - B;
    if (dcmp(Dot(v1, v2)) < 0) return Length(v2);
    if (dcmp(Dot(v1, v3)) > 0) return Length(v3);
    return fabs(Cross(v1, v2)) / Length(v1);
}

Point GetLineProjection(Point P, Point A, Point B)
{
    Vector v = B - A;
    return A + v * (Dot(v, P - A) / Dot(v, v));
}

bool SegmentProperIntersection(Point a1, Point a2, Point b1, Point b2)
{
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - b1),
           c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}

bool OnSegment(Point p, Point a1, Point a2)
{
    return dcmp(Cross(a1 - p, a2 - p)) == 0 && dcmp(Dot(a1 - p, a2 - p)) < 0;
}
```

7.7 刘汝佳版多边形

多边形面积，点在多边形内，凸包，半平面交

```
typedef vector<Point> Polygon;
//多边形的有向面积
double PolygonArea(Polygon po)
{
    int n = po.size();
    double area = 0.0;
    for (int i = 1; i < n - 1; i++)
        area += Cross(po[i] - po[0], po[i + 1] - po[0]);
    return area / 2;
}

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

```
    }
    if (wn != 0) return 1; //内部
    return 0;              //外部
}

//凸包(Andrew算法)
//如果不希望在凸包的边上有输入点，把两个 <= 改成 <
//如果不介意点集被修改，可以改成传递引用
Polygon ConvexHull(vector<Point> p)
{
    sort(p.begin(), p.end());
    p.erase(unique(p.begin(), p.end()), p.end());
    int n = p.size(), m = 0;
    Polygon res(n + 1);
    for (int i = 0; i < n; i++)
    {
        while (m > 1 && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
        res[m++] = p[i];
    }
    int k = m;
    for (int i = n - 2; i >= 0; i--)
    {
        while (m > k && Cross(res[m - 1] - res[m - 2], p[i] - res[m - 2]) <= 0) m--;
        res[m++] = p[i];
    }
    m -= n > 1;
    res.resize(m);
    return res;
}

//半平面交
vector<Point> HalfplaneIntersection(vector<Line>& L)
{
    int n = L.size();
    sort(L.begin(), L.end()); // 按极角排序

    int first, last;    // 双端队列的第一个元素和最后一个元素的下标
    vector<Point> p(n); // p[i]为q[i]和q[i+1]的交点
    vector<Line> q(n);  // 双端队列
    vector<Point> ans;  // 结果

    q[first = last = 0] = L[0]; // 双端队列初始化为只有一个半平面L[0]
    for (int i = 1; i < n; i++)
    {
        while (first < last && !OnLeft(L[i], p[last - 1])) last--;
        while (first < last && !OnLeft(L[i], p[first])) first++;
        q[++last] = L[i];
        if (fabs(Cross(q[last].v, q[last - 1].v)) < eps)
        { // 两向量平行且同向，取内侧的一个
            last--;
            if (OnLeft(q[last], L[i].p)) q[last] = L[i];
        }
        if (first < last) p[last - 1] = GetLineIntersection(q[last - 1], q[last]);
    }
    while (first < last && !OnLeft(q[first], p[last - 1])) last--; // 删除无用平面
    if (last - first <= 1) return vector<Point>();                 // 空集
```

```
    p[last] = GetLineIntersection(q[last], q[first]);              // 计算首尾两个半平面的交点

    return vector<Point>(q.begin() + first, q.begin() + last + 1);
}
```

7.8 刘汝佳版直线和圆

```
struct Line
{
    Point p;    //直线上任意一点
    Vector v;   //方向向量。它的左边就是对应的半平面
    double ang; //极角。即从x正半轴旋转到向量v所需要的角（弧度）
    Line() {}
    Line(Point p, Vector v) : p(p), v(v) { ang = atan2(v.y, v.x); }
    bool operator<(const Line& L) const // 排序用的比较运算符
    {
        return ang < L.ang;
    }
    Point point(double t) { return p + v * t; }
};

struct Circle
{
    Point c;
    double r;
    Circle(Point c, double r) : c(c), r(r) {}
    Point point(double a) { return c.x + cos(a) * r, c.y + sin(a) * r; }
};

int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2, vector<Point>& sol)
{
    double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
    double e = a * a + c * c, f = 2 * (a * b + c * d), g = b * b + d * d - C.r * C.r;
    double delta = f * f - 4 * e * g; //判别式
    if (dcmp(delta) < 0) return 0;    //相离
    if (dcmp(delta) == 0)             //相切
    {
        t1 = t2 = -f / (2 * e);
        sol.push_back(L.point(t1));
        return 1;
    }
    //相交
    t1 = (-f - sqrt(delta)) / (2 * e);
    t2 = (-f + sqrt(delta)) / (2 * e);
    sol.push_back(t1);
    sol.push_back(t2);
    return 2;
}

double angle(Vector v) { return atan2(v.y, v.x); }

int getCircleCircleIntersection(Circle C1, Circle C2, vector<Point>& sol)
{
    double d = Length(C1.c - C2.c);
    if (dcmp(d) == 0)
    {
```

```
        if (dcmp(C1.r - C2.r) == 0) return -1; //两圆重合
        return 0;
    }
    if (dcmp(C1.r + C2.r - d) < 0) return 0;        //内含
    if (dcmp(fabs(C1.r - C2.r) - d) > 0) return 0; //外离

    double a = angle(C2.c - C1.c); //向量C1C2的极角
    double da = acos((C1.r * C1.r + d * d - C2.r * C2.r) / (2 * C1.r * d));
    //C1C2到C1P1的角
    Point p1 = C1.point(a - da), p2 = C1.point(a + da);

    sol.push_back(p1);
    if (p1 == p2) return 1;
    sol.push_back(p2);
    return 2;
}

//过点p到圆C的切线，v[i]是第i条切线的向量，返回切线条数
int getTangents(Point p, Circle C, Vector* v)
{
    Vector u = C.c - p;
    double dist = Length(u);
    if (dist < C.r)
        return 0;
    else if (dcmp(dist - C.r) == 0)
    { //p在圆上，只有一条切线
        v[0] = Rotate(u, M_PI / 2);
        return 1;
    }
    else
    {
        double ang = asin(C.r / dist);
        v[0] = Rotate(u, -ang);
        v[1] = Rotate(u, +ang);
        return 2;
    }
}

//两圆的公切线
//返回切线的条数。-1表示无穷条切线。
//a[i]和b[i]分别是第i条切线在圆A和圆B上的切点
int getTangents(Circle A, Circle B, Point* a, Point* b)
{
    int cnt = 0;
    if (A.r < B.r)
    {
        swap(A, B);
        swap(a, b);
    }
    int d2 = (A.c.x - B.c.x) * (A.c.x - B.c.x) + (A.c.y - B.c.y) * (A.c.y - B.c.y);
    int rdiff = A.r - B.r;
    int rsum = A.r + B.r;
    if (d2 < rdiff * rdiff) return 0; //内含
    double base = atan2(B.c.y - A.c.y, B.c.x - A.c.x);
    if (d2 == 0 && A.r == B.r) return -1; //无限多条切线
    if (d2 == rdiff * rdiff)
```

```
    { //内切，一条切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(base);
        cnt++;
        return 1;
    }
    //有外共切线
    double ang = acos(A.r - B.r) / sqrt(d2);
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base + ang);
    cnt++;
    a[cnt] = A.point(base + ang);
    b[cnt] = B.point(base - ang);
    cnt++;
    if (d2 == rsum * rsum)
    {
        a[cnt] = A.point(base);
        b[cnt] = B.point(M_PI + base);
        cnt++;
    }
    else if (d2 > rsum * rsum)
    {
        double ang = acos((A.r + B.r) / sqrt(d2));
        a[cnt] = A.point(base + ang);
        b[cnt] = B.point(M_PI + base + ang);
        cnt++;
        a[cnt] = A.point(base - ang);
        b[cnt] = B.point(M_PI + base - ang);
        cnt++;
    }
    return cnt;
}

//三角形外接圆（三点保证不共线）
Circle CircumscribedCircle(Point p1, Point p2, Point p3)
{
    double Bx = p2.x - p1.x, By = p2.y - p1.y;
    double Cx = p3.x - p1.x, Cy = p3.y - p1.y;
    double D = 2 * (Bx * Cy - By * Cx);
    double cx = (Cy * (Bx * Bx + By * By) - By * (Cx * Cx + Cy * Cy)) / D + p1.x;
    double cy = (Bx * (Cx * Cx + Cy * Cy) - Cx * (Bx * Bx + By * By)) / D + p1.y;
    Point p = Point(cx, cy);
    return Circle(p, Length(p1 - p));
}

//三角形内切圆
Circle InscribedCircle(Point p1, Point p2, Point p3)
{
    double a = Length(p2 - p3);
    double b = Length(p3 - p1);
    double c = Length(p1 - p2);
    Point p = (p1 * a + p2 * b + p3 * c) / (a + b + c);
    return Circle(p, DistanceToLine(p, p1, p2));
}
```

8 各种操作

8.1 二进制位操作

- __builtin_ffs(x) 返回x中最后一个为1的位是从后向前的第几位，从1开始计数
- __builtin_popcount(x) x中1的个数。
- __builtin_ctz(x) x末尾0的个数。x=0时结果未定义。
- __builtin_clz(x) x前导0的个数。x=0时结果未定义。
  上面的x都是unsigned int型的，如果传入signed或者是char型，会被强制转换成unsigned int
- __builtin_parity(x) x中1的个数的奇偶性,返回1为奇。

8.2 bitset

- bitset<100000>bt;
- bt«1; //整体移位
- bt|=10;
- bt.count(); //b中置为1的二进制位的个数
- bt.size(); //b中二进制位的个数
- bt[pos]; //访问b中在pos处的二进制位
- bt.test(pos); //b中在pos处的二进制位是否为1
- bt.set(); //把b中所有二进制位都置为1
- bt.set(pos); //把b中在pos处的二进制位置为1
- bt.reset(); //把b中所有二进制位都置为0
- bt.reset(pos); //把b中在pos处的二进制位置为0

8.3 nth_element

- nth_element(first,nth,last)
- first，last 第一个和最后一个元素位置
- nth:要定位的第n个元素
- nth_element会将第n_th 元素放到它该放的位置上，左边元素都小于它，右边元素都大于它
- 期望复杂度o(n)
- nth_element(a,a+6,a+10)

8.4 Rope

```
/*
专用于块状链表计算的rope容器
平衡树实现，各种操作的复杂度都是O(log n)
*/
//头文件
#include <ext/rope>
using namespace __gnu_cxx;

rope<int> T;

T.push_back(x);//在末尾添加x

T.insert(pos,x);//在pos插入x

T.erase(pos,x);//从pos开始删除x个

T.copy(pos,len,x);//从pos开始到pos+len为止用x代替

T.replace(pos,x);//从pos开始换成x

T.substr(pos,x);//提取pos开始x个

T.at(x)/[x];//访问第x个元素

printf("%d\n",T[i]) //输出T[i]
cout<<T<<endl;//输出T

/* 2018nowcoder多校3 https://www.nowcoder.com/acm/contest/141/C */
#include <bits/stdc++.h>
#include <ext/rope>//函数头文件
using namespace __gnu_cxx;
using namespace std;
const int maxn=1e5+10;
rope<int> T;
int n,m;
int main()
{
    scanf("%d%d",&n,&m);
    for(int i=1;u<=n;i++)
     T.push_back(i);
     while(m--)
     {
         int p,s;
         scanf("%d%d",&p,&s);
         p--;
         T=T.substr(p,s)+T.substr(0,p)+T.substr(p+s,n-p-s);
     }
     for(int i=0;i<n;i++)
      printf("%d ",T[i]);
      return 0;
}
```

8.5 pb_ds

参考文献:C++的pb_ds库在OI中的应用

__gnu_pbds::priority_queue 可合并堆

- 头文件 ext/pb_ds/priority_queue.hpp
  __gnu_pbds::priority_queue<T,greater,TAG>
- 函数：size(),empty(),push(T),top(),pop(),clear()
- 新增功能
  - begin(),end()获取iterator遍历
  - increase_key,decrease_key
  - 删除单个元素 erase(point_iterator)
  - point_iterator push(T)
  - 修改元素 modify(point_iterator,T)
  - 合并堆：q1.join(q2) 将q2合并到q1，q2被清空
- TAG:
  五种操作:push,pop,modify,erase,join
  - pairing_heap_tag(配对堆)：push,join$o(1)$，其余均摊$o(logn)$ (默认)
  - binary_heap_tag(二叉堆): 只支持push,pop 均摊$o(logn)$
  - binomial_heap_tag(二项堆): push均摊$o(1)$,其余$o(logn)$
  - rc_binomial_heap_tag: push$o(1)$，其余$o(logn)$
  - thin_heap_tag(斐波那契堆): push$o(1)$,不支持join,其余$o(logn)$,只有increase_key的话modity$o(1)$
- 合并,dij均使用: pairing_heap_tag
- 只有push,pop,join: binary_heap_tag

__gnu_pbds::tree

- 头文件 ext/pb_ds/assoc_container.hpp
  ext/pb_ds/tree_policy.hpp
  __gnu_gbds::tree<key,T,TAG,Node_Update>
- 函数类似于map：begin(),end(),size(),empty(),clear(),
  find(key),lower_bound(key),upper_bound(key),
  erase(iterator),erase(key),insert(<key,T>),operator
- 第二个参数改为null_type(null_mapped_type)即为set
- TAG:
  - rb_tree_tag
  - splay_tree_tag
- 寻找第order+1小的元素,order过大返回end()：iterator find_by_order(order)
- 询问有多少个比key小的元素：order_of_key(key)
- t1.join(t2)将t2所有元素移动到t1,t1、t2值域不能相交
- t1.split(key,t2)清空t2，把所有大于key的元素移动到other
- 自带的Node_Update:tree_order_statistics_node_update统计子树大小

- 自定义Node_Update

```
template<class Node_CItr,class Node_Itr,class Cmp_Fn,class _Alloc>
struct my_node_update{
    virtual Node_CItr node_begin() const =0;
    virtual Node_CItr node_end() const =0;
    typedef int metadata_typde;//节点记录的额外信息的类型
}
/*
将系欸但it的信息更新为其左右孩子的信息
传入end_it表示空节点
*/
inline void operator()(Node_Itr it,Node_CItr end_it)
{
    Node_Itr l=it.get_l_child(),r=it.get_r_child();
    int left=0,right=0;
    if(l!=end_it) left=l.get_metadata();
    if(r!=end_it) right=r.get_metadata();
    const_cast<metadata_type &>(it.get_metadata())
     =left+right+(*it)->second;
}
inline int prefix_sum(int x)
{
    int ans=0;
    Node_CItr it=node_begin();
    while(it!=node_end())
    {
        Node_CItr l= it.get_l_child(),r=it.get_r_child();
        if(Cmp_Fn()(x,(*it)->first)) it=l;
        else{
            ans+=(*it)->second;
            if(l!=node_end())
               ans+=l.get_metadata();
               it=r;
        }
    }
    return ans;
}
inline int interval_sum(int l,int r)
{
    return prefix_sum(r)-prefix_sum(l-1);
}
```

get_l_child,get_r_child获取左右孩子,(**it)获取节点信息，get_metadata获取节点额外信息

hash_table

- 头文件 ext/pb_ds_assoc_container.hpp
  ext/pb_ds/hash_policy.hpp
  __gnu_pbds::cc_hash_table<key,mapped>(拉链法)
  __gun_pbds::gp_hash_table<key,mpped>（查探法较快）

8.6 String And Char

```
/*大小写转换函数*/
a=tolower(a);
b=toupper(b);

/**sstream**/
string  sstream
string str; getline(cin,str);
stringstream ss(str);//对string对象进行读写
while(ss>>x)
   s.compare(b);

ss.clear();//多次使用stringstream，要先清空下
/**stringstream可以用来把string类型的字符串转换成int **/
string s("12345");
int x;
stringstream ss(s);
ss>>x;
/**法二**/
string s("12345");
int x;
stringstream ss;
ss<<s;
ss>>x;

/**将多种数值转换成字符串**/
typename x=5.222;
cin>>x;
stringstream ss;
ss<<x;
string s;
s=ss.str(); //s="5.222"

/**字符串处理*/
    char s[] = "a,b*c,d";
    const char *sep = ",*"; //可按多个字符来分割
    char *p;
    p = strtok(s, sep);
    //在第一次被调用的时间str是传入需要被切割字符串的首地址；在后面调用的时间传入NULL
    while(p){
        printf("%s ", p);
        p = strtok(NULL, sep);
    }

/*取子串*/
string sub1 = s.substr(5); //从下标为5开始一直到结尾
string sub2 = s.substr(5, 3); //从下标为5开始截取长度为3位

/*char 数组操作*/
strcpy
/*https://blog.csdn.net/ncabhd/article/details/72903123*/
strcat(charr5, " juice");//添加到末尾
```

8.7 String To Int

int/float to string/array

- itoa()：将整型值转换为字符串。
- ltoa()：将长整型值转换为字符串。
- ultoa()：将无符号长整型值转换为字符串。
- gcvt()：将浮点型数转换为字符串，取四舍五入。
- ecvt()：将双精度浮点型值转换为字符串，转换结果中不包含十进制小数点。
- fcvt()：指定位数为转换精度，其余同ecvt()。

string/array to int/float

- atof()：将字符串转换为双精度浮点型值。
- atoi()：将字符串转换为整型值。
- atol()：将字符串转换为长整型值。
- strtod()：将字符串转换为双精度浮点型值，并报告不能被转换的所有剩余数字。
- strtol()：将字符串转换为长整值，并报告不能被转换的所有剩余数字。
- strtoul()：将字符串转换为无符号长整型值，并报告不能被转换的所有剩余数字。

8.8 IO

```
template <class T> inline bool scan_d(T &ret) {
    char c; int sgn;
    if(c=getchar(),c==EOF) return 0; //EOF
    while(c!='-'&&(c<'0'||c>'9'))
        c=getchar();
    sgn=(c=='-')?-1:1;
    ret=(c=='-')?0:(c-'0');
    while(c=getchar(),c>='0'&&c<='9')
        ret=ret*10+(c-'0');
    ret*=sgn;
    return 1; }

inline void out(ll x) {
   if(x>9) out(x/10);
   putchar(x%10+'0');
   }

/*steal from zerol*/
1inline char next_char() {
    static char buf[100000], *p1 = buf, *p2 = buf;
    return p1 == p2 &&
    (p2 = (p1 = buf) + fread(buf, 1, 100000, stdin), p1 == p2) ? EOF : *p1++;
}
inline bool maybe_digit(char c) {
    return c >= '0' && c <= '9';
}
template <typename T>
void rn(T& _v) {
    static char ch;
```

```
    static bool negative = false;
    _v = 0;
    while (!maybe_digit(ch)) {
        negative = ch == '-';
        ch = next_char();
    }
    do _v = (_v << 1) + (_v << 3) + ch - '0';
    while (maybe_digit(ch = next_char()));
    if (negative) _v = -_v;
}

template <typename T>
void o(T p) {
    static int stk[70], tp;
    if (p == 0) {
        putchar('0');
        return;
    }
    if (p < 0) { p = -p; putchar('-'); }
    while (p) stk[++tp] = p % 10, p /= 10;
    while (tp) putchar(stk[tp--] + '0');
}
```

8.9 BigInt

```
/*steal from csl*/
#define N 10000
class bint
{
private:
    int a[N]; // 用 N 控制最大位数
    int len;  // 数字长度
public:
    // 构造函数
    bint() { len = 1, clr(a, 0); }
    // int -> bint
    bint(int n)
    {
        len = 0;
        clr(a, 0);
        int d = n;
        while (n)
            d = n / 10 * 10, a[len++] = n - d, n = d / 10;
    }
    // char[] -> int
    bint(const char s[])
    {
        clr(a, 0);
        len = 0;
        int l = strlen(s);
        for (int i = l - 1; ~i; i--) a[len++] = s[i];
    }
    // 拷贝构造函数
    bint(const bint& b)
    {
        clr(a, 0);
```

```
    len = b.len;
    for (int i = 0; i < len; i++) a[i] = b.a[i];
}
// 重载运算符 bint = bint
bint& operator=(const bint& n)
{
    len = n.len;
    for (int i = 0; i < len; i++) a[i] = n.a[i];
    return *this;
}
// 重载运算符 bint + bint
bint operator+(const bint& b) const
{
    bint t(*this);
    int res = b.len > len ? b.len : len;
    for (int i = 0; i < res; i++)
    {
        t.a[i] += b.a[i];
        if (t.a[i] >= 10) t.a[i + 1]++, t.a[i] -= 10;
    }
    t.len = res + a[res] == 0;
    return t;
}
// 重载运算符 bint - bint
bint operator-(const bint& b) const
{
    bool f = *this > b;
    bint t1 = f ? *this : b;
    bint t2 = f ? b : *this;
    int res = t1.len, j;
    for (int i = 0; i < res; i++)
        if (t1.a[i] < t2.a[i])
        {
            j = i + 1;
            while (t1.a[j] == 0) j++;
            t1.a[j--]--;
            while (j > i) t1.a[j--] += 9;
            t1.a[i] += 10 - t1.a[i];
        }
        else
            t1.a[i] -= t2.a[i];
    t1.len = res;
    while (t1.a[len - 1] == 0 && t1.len > 1) t1.len--, res--;
    if (f) t1.a[res - 1] = 0 - t1.a[res - 1];
    return t1;
}
// 重载运算符 bint * bint
bint operator*(const bint& b) const
{
    bint t;
    int i, j, up, tmp, tmp1;
    for (i = 0; i < len; i++)
    {
        up = 0;
        for (j = 0; j < b.len; j++)
        {
```

```
                tmp = a[i] * b.a[j] + t.a[i + j] + up;
                if (tmp > 9)
                    tmp1 = tmp - tmp / 10 * 10, up = tmp / 10, t.a[i + j] = tmp1;
                else
                    up = 0, t.a[i + j] = tmp;
            }
            if (up) t.a[i + j] = up;
        }
        t.len = i + j;
        while (t.a[t.len - 1] == 0 && t.len > 1) t.len--;
        return t;
    }
    // 重载运算符 bint / int
    bint operator/(const int& b) const
    {
        bint t;
        int down = 0;
        for (int i = len - 1; ~i; i--)
            t.a[i] = (a[i] + down * 10) / b, down = a[i] + down * 10 - t.a[i] * b;
        t.len = len;
        while (t.a[t.len - 1] == 0 && t.len > 1) t.len--;
        return t;
    }
    // 重载运算符 bint ^ n (n次方快速幂, 需保证n非负)
    bint operator^(const int n) const
    {
        bint t(*this), rt(1);
        if (n == 0) return 1;
        if (n == 1) return *this;
        int m = n;
        for (; m; m >>= 1, t = t * t)
            if (m & 1) rt = rt * t;
        return rt;
    }
    // 重载运算符 bint > bint 比较大小
    bool operator>(const bint& b) const
    {
        int p;
        if (len > b.len) return 1;
        if (len == b.len)
        {
            p = len - 1;
            while (a[p] == b.a[p] && p >= 0) p--;
            return p >= 0 && a[p] > b.a[p];
        }
        return 0;
    }
    // 重载运算符 bint > int 比较大小
    bool operator>(const int& n) const { return *this > bint(n); }
    // 输出
    void out()
    {
        for (int i = len - 1; ~i; i--) printf("%d", a[i]);
        puts("");
    }
};
```

8.10 日期计算DateMagic

```
/*日期计算*/
string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)
int DateToInt(int m, int d, int y)
{
    return 1461 * (y + 4800 + (m - 14) / 12) / 4
           + 367 * (m - 2 - (m - 14) / 12 * 12) / 12
           - 3 * ((y + 4900 + (m - 14) / 12) / 100) / 4
           + d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year
void IntToDate(int jd, int& m, int& d, int& y)
{
    int x, n, i, j;
    x = jd + 68569;
    n = 4 * x / 146097;
    x -= (146097 * n + 3) / 4;
    i = (4000 * (x + 1)) / 1461001;
    x -= 1461 * i / 4 - 31;
    j = 80 * x / 2447;
    d = x - 2447 * j / 80;
    x = j / 11;
    m = j + 2 - 12 * x;
    y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week
string IntToDay(int jd) { return dayOfWeek[jd % 7]; }
```

8.11 Other Tips

```
printf("%04d\n",x);//输出4位,不足则前面填充0

/*c++格式化 long double的输出*/
 long double t1 = (l - t2 * v2) / v1;
cout << fixed << setprecision(10) << t1 << endl;//保留10位小数

/*枚举真子集*/
for(int i=x;i;i=(i-1)&x)
    cout<<i<<endl;

/*枚举大小为 k 的子集*/
void subset(int k, int n)
{
   int t = (1 << k) - 1;
   while (t < (1 << n))
   {
   // do something
   int x = t & -t, y = t + x;
 t = ((t & ~y) / x >> 1) | y;
  }
}
```

```
/**mt19937随机数*/
 unsigned seed=chrono::system_clock::now().time_since_epoch().count();
    mt19937 g1(seed);
    cout<<g1()<<endl;
```

9 Java

9.1 高精度

```
/*
四则运算:add（加） subtract（减） multiply（乘） divide（除）
remainder（取余）
*/

import java.util.Scanner;
import java.math.*;

public class Main {
        public static class point {
                BigDecimal a, b;
        }

        public static point tempx = new point();

        public static point find(point x, point y, point z) {
                BigDecimal a, b, a1, a2, b1, b2, c1, c2, t, d;
                t = BigDecimal.valueOf(2);
                a1 = y.a.subtract(x.a);
                b1 = y.b.subtract(x.b);
                tempx.a = a1;
                tempx.b = b1;
                c1 = ((a1.multiply(a1)).add(b1.multiply(b1))).divide(t);
                a2 = z.a.subtract(x.a);
                b2 = z.b.subtract(x.a);
                c2 = ((a2.multiply(a2)).add(b2.multiply(b2))).divide(t);
                d = (a1.multiply(b2)).subtract(a2.multiply(b1));
                tempx.a = x.a.add((c1.multiply(b2).subtract(c2.multiply(b1))).divide(d, 20, 0));
                tempx.b = x.b.add((a1.multiply(c2).subtract(a2.multiply(c1))).divide(d, 20, 0));
                return tempx;
        }

        public static BigDecimal distance(point x, point y)// 没有开根
        {
                BigDecimal temp;
                temp = ((x.a.subtract(y.a)).multiply(x.a.subtract(y.a))).add((x.b.subtract(y.b)).
multiply(x.b.subtract(y.b)));
                return temp;
        }

        static point p[] = new point[5], temp = null;

        public static void main(String[] args) {
                Scanner s = new Scanner(System.in);
                int T, i;
                T = s.nextInt();
                for (i = 0; i < 4; i++)
                        p[i] = new point();
                while (T-- > 0) {

                        for (i = 0; i < 4; i++) {
                                p[i].a = s.nextBigDecimal();
```

```
                        p[i].b = s.nextBigDecimal();
                }
                temp = find(p[0], p[1], p[2]);
                if (distance(p[0], temp).compareTo(distance(p[3], temp)) < 0)
                        System.out.println("Accepted");
                else
                        System.out.println("Rejected");
        }
    }
}
```

9.2 Java输入输出

```
/*Java大数高精度亲情讲义

java输入输出架构（注意主类为Main）
*/
import java.util.Scanner;//输入架构
public class Main {
        public static void main(String[] args) {
                Scanner s=new Scanner(System.in);
                int a,b;
                a=s.nextInt();
                b=s.nextInt();
                System.out.println((a+b));
        }
}

/*java函数调用*/
import java.util.Scanner;
public class Main {
        public static int gcd(int a,int b)
        {
                if(b==0)
                        return a;
                else
                        return gcd(b,a%b);
        }
        public static void main(String[] args) {
                Scanner s=new Scanner(System.in);
                int a,b,sum,i;
                while(s.hasNext())
                {
                a=s.nextInt();
                b=s.nextInt();
                sum=1;
                i=b;
                while(i!=0)
                {
                        int temp=gcd(a,b);
                        if(temp==1)
                                break;
                        else
                        {
                                sum*=temp;
                                a/=temp;
```

```
                        }
                        i--;
                }
                System.out.print(sum+"\n");
                }
        }

}
```

9.3 Java快速读入

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.math.*;
import java.nio.Buffer;
import java.util.*;
public class Main {

        public static void main(String[] args) {

                int n;
                FastScanner sc=new FastScanner();
                PrintWriter pw=new PrintWriter(System.out);

                n = sc.nextInt();
                int ans;
                pw.println(ans);

                pw.flush();
        }

}
class FastScanner{
                BufferedReader br;
                StringTokenizer st;

                public FastScanner()
                {
                        try{
                                br=new BufferedReader(new InputStreamReader(System.in),32768);
                                st=new StringTokenizer("");
                        }
                        catch (Exception e) {
                                // TODO: handle exception
                           e.printStackTrace();
                        }
                }
         public boolean hasNext() {
                while(!st.hasMoreTokens()){
                        String line=nextLine();
                        if(line==null)
                                return false;
                   st=new StringTokenizer(line);
                }
                return true;
```

```
}
 public String next()
 {
         while(!st.hasMoreTokens()){
                 st=new StringTokenizer(nextLine());
         }
         return st.nextToken();
 }
 public int nextInt(){
         return Integer.parseInt(next());
 }
 public long nextLong(){
         return Long.parseLong(next());
 }
 public double nextDouble(){
         return Double.parseDouble(next());
 }
 public String nextLine(){
         String line="";
         try{
                 line=br.readLine();
         }catch (Exception e) {
                 e.printStackTrace();
                // TODO: handle exception
        }
         return line;
 }

}
```

嚶嚶嚶