华南师范大学
ACM-ICPC 集训队
DeDeRong

2019 年 10 月 21 日

目录

1 图论

1.1 最短路径

1.1.1 Dijkstra

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f;

//d数组用来记录源点s到顶点i的最短距离
//v表示该顶点是否在顶点集S中
//g邻接矩阵存图，g[i][j]表示i到j的边的权值，无边时为inf
//n为顶点数量
int d[maxn], v[maxn];
int g[maxn][maxn];
int n;
void dij(int s)
{
  memset(v, 0, sizeof(v));
  for(int i=1;i<=n;i++)
  d[i] = g[s][i];
  v[s] = 1;
  for(int i=1;i<=n;i++)
  {
    int u = 0;
    for(int j=1;j<=n;j++)
    {
      if(!v[j] && (u==0 || d[j] < d[u]))
        u = j;
    }
    if(u==0)return ;
    v[u] = 1;
    for(int j=1;j<=n;j++)
    {
      d[j] = min(d[j], d[u]+g[u][j]);
    }
  }
}
```

1.1.2 Dijkstra 优化

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f;
typedef pair<int, int> P; //first表示最短距离，second表示顶点编号
//边：to表示这条边指向的顶点，权值为w
struct Edge
{
  int to, w;
};
//用vector实现邻接表
vector<Edge> g[maxn];
int d[maxn]; //记录源点到顶点i的最短距离
int n;

void dij(int s)
{
  priority_queue<P, vector<P>, greater<P> > q;
  memset(d, inf, sizeof(d));
  d[s] = 0;
  q.push(P(0, s));
  while(!q.empty())
  {
    P p = q.top();
    q.pop();
    int u = p.second;
    if(d[u] < p.first) continue;
    for(int i=0; i<g[u].size(); i++)
    {
      Edge e = g[u][i];
      if(d[e.to] > d[u] + e.w)
      {
        d[e.to] = d[u] + e.w;
        q.push(P(d[e.to], e.to));
      }
    }
  }
}
```

1.1.3 Floyd

```
int g[maxn][maxn];
int n;
void floyd()
{
```

```
  for(int k=1;k<=n;k++)
    for(int i=1;i<=n;i++)
      for(int j=1;j<=n;j++)
        g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
}
```

1.1.4 Bellman-Ford

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f; //常用于表示无穷大

//边结构体，记录u—>v的边，权值为w
struct Edge
{
  int u, v, w;
  Edge(int uu, int vv, int ww) { u=uu; v=vv; w=ww; }
  Edge(){}
}e[maxn];
int edgecnt; // 边的数量
//加边操作
void addEdge(int u, int v, int w)
{
  e[edgecnt++] = Edge(u, v, w);
}

int n; //顶点总数
int d[maxn]; //记录最短距离的数组

//存在负权回路则返回true，否则返回false
bool bellman_ford(int s)
{
  memset(d, inf, sizeof(d));
  d[s] = 0;
  //进行n—1次松弛操作,第n次检查是否含有负权回路
  for(int i=1;i<=n;i++)
  {
    int flag = 0;
    for(int j=0; j<edgecnt; j++)
    {
      Edge t = e[j];
      int u, v, w;
      u = t.u; v = t.v; w = t.w;
      if(d[v] > d[u] + w)
      {
        d[v] = d[u] + w;
        flag = 1;
      }
    }
    if(!flag) return false;
    if(i==n && flag) return true;
  }
  return false;
}
```

1.1.5 SPFA

```
const int maxn = 1e4;
const int inf = 0x3f3f3f3f; //常用于表示无穷大

//边结构体，to表示边指向的顶点编号，权值为w
struct Edge
{
  int to, w;
  Edge(int tt, int ww) { to = tt; w = ww; }
  Edge(){}
};
//vector实现的邻接表
vector<Edge> g[maxn];
int n;//顶点数
//d表示最短距离， inq[i]表示结点是否在队列中，为1则在，cnt[i]记录i入队的次数
int d[maxn], inq[maxn], cnt[maxn];
//初始化
void init()
{
  memset(d, inf, sizeof(d));
  memset(inq, 0, sizeof(inq));
  memset(cnt, 0, sizeof(cnt));
}
//返回true表示存在负权回路
bool spfa(int s)
{
  init();
  d[s] = 0;
  inq[s] = 1;
```

```
  cnt[s] = 1;
  queue<int> q;
  q.push(s);
  while(!q.empty())
  {
    int u = q.front();
    inq[u] = 0;
    q.pop();
    for(int i=0;i < g[u].size(); i++)
    {
      Edge e = g[u][i];
      if(d[e.to] > d[u] + e.w)
      {
        d[e.to] = d[u] + e.w;
        if(inq[e.to] == 0)
        {
          inq[e.to] = 1;
          q.push(e.to);
          cnt[e.to]++;
          if(cnt[e.to] > n) return true;
        }
      }
    }
  }
  return true;
}
```

1.2 次短路

```
#include <bits/stdc++.h>
#define INF 1e16+100
#define ms(x,y) memset(x,y,sizeof(x))
using namespace std;

typedef long long ll;
typedef pair<ll,ll> P;

const double pi = acos(-1.0);
const int mod = 1e9 + 7;
const int maxn = 1e5 + 5;

struct Edge{
  ll to,cost;
};

ll n,m;
vector<Edge> a[maxn];
ll dist[maxn],dist2[maxn];

void addedge(ll u,ll v,ll w)
{
  a[u].push_back(Edge{v,w});
  a[v].push_back(Edge{u,w});
}

void solve()
{
  priority_queue<P, vector<P>, greater<P> >que;
  //ms(dist,INF);
  //ms(dist2,INF);
  fill(dist,dist+n,INF);
  fill(dist2,dist2+n,INF);
  dist[0]=0;
  que.push(P(0,0));
  while(que.size())
  {
    P u=que.top();que.pop();
    int v=u.second;
    ll d=u.first;
    if(dist2[v]<d) continue;  //不是次短距离则抛弃
    for(int i=0;i<a[v].size();i++)
    {
      Edge e=a[v][i];
      ll d2=d+e.cost;
      if(dist[e.to]>d2) //更新最短
      {
        swap(dist[e.to],d2);
        que.push(P(dist[e.to],e.to));
      }
      if(dist2[e.to]>d2&&dist[e.to]<d2) //更新次短
      {
        dist2[e.to]=d2;
        que.push(P(dist2[e.to],e.to));
      }
    }
  }
```

```
  printf("%lld\n",dist2[n-1]);
}

int main()
{
  //freopen("in.txt","r",stdin);
  //freopen("out.txt","w",stdout);
  int t;
  scanf("%d",&t);
  while(t--)
  {
    scanf("%lld%lld",&n,&m);
    for(int i=0;i<n;i++) a[i].clear();
    for(int i=0;i<m;i++)
    {
      ll p,q,w;
      scanf("%lld%lld%lld",&p,&q,&w);
      addedge(p-1,q-1,w);
    }
    solve();
  }
  return 0;
}
```

1.3 LCA

1.3.1 倍增

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 5e5+5;
const ll mod = 1e9+7;

vector<int> son[maxn]; // 存储儿子顶点
// dep[i]表示顶点i的深度，n个顶点，m个询问，rt为树根，fa数组用来预处理顶点i向上跳2^j步之后的顶点
int dep[maxn], n, m, rt, fa[maxn][20];
int v[maxn]={0}; // 是否访问标记

// pre是父顶点，rt是当前顶点
void dfs(int pre, int rt)
{
  dep[rt] = dep[pre]+1; // 当前顶点的深度为父顶点加一
  fa[rt][0] = pre; // 当前顶点向上跳一步为父顶点
  v[rt] = 1; // 访问
  // dp预处理
  for(int i=1;i<=19;i++)
  fa[rt][i] = fa[fa[rt][i-1]][i-1];
  // 继续dfs
  for(int i=0;i<son[rt].size();i++)
  if(v[son[rt][i]]==0)
  dfs(rt, son[rt][i]);
}

// 求解LCA(a, b)
int lca(int a, int b)
{
  if(dep[a] < dep[b])
    swap(a, b);
  for(int i=19;i>=0;i--)
  {
    if(dep[a]-dep[b] >= (1<<i))
    {
      a = fa[a][i];
    }
  }
  if(a==b)return a;
  for(int i=19;i>=0;i--)
  {
    if(fa[a][i] != fa[b][i])
    {
      a = fa[a][i];
      b = fa[b][i];
    }
  }
  return fa[a][0];
}

int main()
{
  scanf("%d%d%d", &n, &m, &rt);
  for(int i=1;i<n;i++)
  {
    int a, b;
```

```
    scanf("%d%d", &a, &b);
    son[a].push_back(b);
    son[b].push_back(a);
  }
  memset(fa, 0, sizeof(fa));
  memset(dep, inf, sizeof(dep));
  v[0]=1;
  dep[0] = 0;
  dfs(0, rt);
  for(int i=1;i<=m;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d\n", lca(a, b));
  }
  return 0;
}
```

1.3.2 RMQ

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 5e5+5;
const ll mod = 1e9+7;

vector<int> g[maxn]; // 存图
// dep记录DFS序中每一个顶点的深度， vis记录DFS序， id记录顶点i第一次在DFS序中的位置， st表
int dep[maxn<<1]={0}, vis[maxn<<1]={0}, id[maxn]={0}, st[maxn<<1][25];
// dfs序计数用，看代码能理解
int dfs_c=1;

// 父顶点为pre， 当前顶点为now， 当前深度为d
void dfs(int pre, int now, int d)
{
  id[now] =  dfs_c; // now顶点在DFS序中第一次出现的位置是dfs_c
  dep[dfs_c] = d; // 记录now的深度
  vis[dfs_c++] = now; // DFS序中第dfs_c个顶点是now，同时将dfs_c加一
  for(int i=0;i<g[now].size();i++)
  {
    if(g[now][i]!=pre)
    {
      dfs(now, g[now][i], d+1);
      vis[dfs_c] = now;
      dep[dfs_c++] = d;
    }
  }
}

// 预处理st表
void getSt(int n)
{
  for(int i=1;i<=n;i++)
  st[i][0] = i;
  for(int j=1; (1<<j)<=n; j++)
  {
    for(int i=1;i+(1<<j)<=n; i++)
    {
      int a = st[i][j-1], b = st[i+(1<<(j-1))][j-1];
      if(dep[a] < dep[b])
        st[i][j] = a;
      else st[i][j] = b;
    }
  }
}

// 查询DFS序中区间[l, r]深度最小的顶点在DFS序中的位置
int query(int l, int r)
{
  int k = log2(r-l+1);
  int a = st[l][k];
  int b = st[r-(1<<k)+1][k];
  // 返回深度较小的那一个顶点在DFS序中的位置
  if(dep[a]<dep[b])return a;
  else return b;
}

// 求LCA(a, b)
int lca(int a, int b)
{
  int x, y;
  x = id[a], y = id[b];
  if(x>y)return vis[query(y, x)];
```

```
  else return vis[query(x, y)];
}

// 检查用的
void check(int n)
{
  for(int i=1;i<=dfs_c;i++)cout<<dep[i]<<" ";cout<<"\n\n";
  for(int i=1;i<=dfs_c;i++)cout<<vis[i]<<" ";cout<<"\n\n";
  for(int i=1;i<=n;i++)cout<<id[i]<<" ";cout<<"\n\n";
}

int main()
{
  int n, m, rt;
  scanf("%d%d%d", &n, &m, &rt);
  for(int i=1;i<n;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    g[a].push_back(b);
    g[b].push_back(a);
  }
  dfs(0, rt, 1);
  getSt(dfs_c);
  //check(n);
  for(int i=1;i<=m;i++)
  {
    int a, b;
    scanf("%d%d", &a, &b);
    printf("%d\n", lca(a, b));
  }
  return 0;
}
```

1.4 强连通分量

```
vector<int> g[maxn];
int low[maxn], dfn[maxn], sta[maxn], ins[maxn], belong[maxn];
int cnt, ind, tot; //cnt：强连通分量的数量， ind：时间戳， tot：sta的top

void init()
{
  memset(ins, 0, sizeof(ins));
  memset(belong, 0, sizeof(belong));
  memset(dfn, 0, sizeof(dfn));
  cnt = ind = tot = 0;
}

void Tarjan(int u)
{
  low[u] = dfn[u] = ++ind;
  ins[u] = 1;
  sta[++tot] = u;
  for(int i=0;i<g[u].size();i++)
  {
    int v = g[u][i];
    if(!dfn[v])
    {
      Tarjan(v);
      low[u] = min(low[u], low[v]);
    }
    else if(ins[v])
    low[u] = min(low[u], dfn[v]);
  }
  int p;
  if(low[u] == dfn[u])
  {
    ++cnt;
    do
    {
      p = sta[tot--];
      belong[p] = cnt;
      ins[p] = 0;
    }while(p != u);
  }
}
```

1.5 割点

```
vector<int> g[maxn];
// iscut[i]：若顶点i是割点，则为1，反之为0
int low[maxn], dfn[maxn], iscut[maxn];
int ind;

```

```
void init()
{
  memset(dfn, 0, sizeof(dfn));
  memset(iscut, 0, sizeof(iscut));
  ind = 0;
}

// pa为u的父节点，初始时Tarjan(i, i)
void Tarjan(int u, int pa)
{
  int cnt = 0; //用来记录子树的数量
  low[u] = dfn[u] = ++ind;
  for(int i=0;i<g[u].size();i++)
  {
    int v = g[u][i];
    if(!dfn[v])
    {
      Tarjan(v, u);
      low[u] = min(low[u], low[v]);
      // 若low[v]>=dfn[u]，并且u不是根节点，则u是割点
      if(low[v] >= dfn[u] && pa!=u)
      iscut[u] = 1;
      // 若u是根节点，则cnt++
      if(u == pa)
        cnt++;
    }
    else if(v != pa) //若v不等于父节点
    low[u] = min(low[u], dfn[v]);
  }
  if(cnt>=2 && u==pa) //根节点子树数量大于等于2，则为割点
    iscut[u] = 1;
}
```

1.6 桥

```
// 用链式前向星来存储边
struct Edge
{
  // iscut表示是否为桥
  int to, next, iscut;
}e[maxn*maxn*2];

int head[maxn], low[maxn], dfn[maxn];
int ind, tot; // tot是边的数量

void init()
{
  memset(head, -1, sizeof(head));
  memset(dfn, 0, sizeof(dfn));
  ind = tot = 0;
}

void addedge(int u, int v)
{
  e[tot].to = v;
  e[tot].next = head[u];
  e[tot].iscut = 0;
  head[u] = tot++;
}

void Tarjan(int u, int pa)
{
  low[u] = dfn[u] = ++ind;
  for(int i=head[u]; ~i; i = e[i].next)
  {
    int v = e[i].to;
    if(v == pa) continue;
    if(!dfn[v])
    {
      Tarjan(v, u);
      low[u] = min(low[u], low[v]);
      // 是桥
      if(low[v] > dfn[u])
      {
        e[i].iscut = e[i^1].iscut = 1;
      }
    }
    else
    {
      low[u] = min(low[u], dfn[v]);
    }
  }
}
```

1.7 最大流

1.7.1 Dinic

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
const int maxn = 1e6+5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9+7;

// 用链式前向星来存储图
struct ed
{
  int to, val, ne;
}edge[maxn<<1];
int head[maxn], dep[maxn];
// 顶点数n，边数m，源点s，汇点e，加边时的指针tot
int n, m, s, e, tot;

void init()
{
  tot = -1;
  memset(head, -1, sizeof(head));
}

void addEdge(int u, int v, int val)
{
  edge[++tot].to = v;
  edge[tot].val = val;
  edge[tot].ne = head[u];
  head[u] = tot;
}

// 就是最普通的bfs
int bfs()
{
  memset(dep, -1, sizeof(dep));
  dep[s] = 0;
  queue<int> q;
  q.push(s);
  while(!q.empty())
  {
    int u = q.front();
    q.pop();
    for(int i=head[u]; ~i; i=edge[i].ne)
    {
      int v = edge[i].to;
      if(dep[v]==-1 && edge[i].val>0)
      {
        dep[v] = dep[u]+1;
        q.push(v);
      }
    }
  }
  return (dep[e] != -1); //若dep[e]==-1则表示没有可以到达e的增广路了，算法结束。
}

// 当前顶点u，当前流量flow
// 初始时dfs(s, inf)
int dfs(int u, int flow)
{
  if(u == e)return flow;
  for(int i=head[u]; ~i; i=edge[i].ne)
  {
    int v = edge[i].to;
    if(dep[v]==dep[u]+1 && edge[i].val)
    {
      int a = dfs(v, min(flow, edge[i].val));
      if(a>0) //若找到增广路
      {
        edge[i].val -= a;
        edge[i^1].val += a;
        return a;
      }
    }
  }
  return 0;
}

ll dinic()
{
  ll ans = 0;
  while(bfs())
  {
    int a = dfs(s, (1<<30));
```

```
    ans += a;
  }
  return ans;
}

int main()
{
  scanf("%d%d%d%d", &n, &m, &s, &e);
  init();
  for(int i=1;i<=m;i++)
  {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    addEdge(u, v, w);
    addEdge(v, u, 0); //反边
  }
  printf("%lld\n", dinic());
  return 0;
}
```

1.7.2 Dinic 优化

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
const int maxn = 1e6+5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9+7;

struct ed
{
  int to, val, ne;
}edge[maxn<<1];
int head[maxn],dep[maxn], cur[maxn];
int n, m, s, e, tot;

void init()
{
  tot = -1;
  memset(head, -1, sizeof(head));
}

void addEdge(int u, int v, int val)
{
  edge[++tot].to = v;
  edge[tot].val = val;
  edge[tot].ne = head[u];
  head[u] = tot;
}

int bfs()
{
  memset(dep, -1, sizeof(dep));
  dep[s] = 0;
  queue<int> q;
  q.push(s);
  while(!q.empty())
  {
    int u = q.front();
    q.pop();
    for(int i=head[u]; ~i; i=edge[i].ne)
    {
      int v = edge[i].to;
      if(dep[v]==-1 && edge[i].val>0)
      {
        dep[v] = dep[u]+1;
        q.push(v);
      }
    }
  }
  return (dep[e] != -1);
}

int dfs(int u, int flow)
{
  if(u == e)return flow;
  // rflow用于多路增广，表示流入到顶点u的剩余未流出的流量
  int rflow = flow;
  // 当前弧优化，通过引用，可以改变cur[i]的值，使得下次遍历到顶点u时，会直接从上次增广的边开始遍历
  for(int& i=cur[u]; ~i; i=edge[i].ne)
  {
    int v = edge[i].to;
    if(dep[v]==dep[u]+1 && edge[i].val)
    {
```

```
      int a = dfs(v, min(rflow, edge[i].val));
      edge[i].val -= a;
      edge[i^1].val += a;
      rflow -= a; // 剩余流量要减少
      if(rflow<=0)break; // 若没有剩余流量了，就break
    }
  }
  // 若没有一丝流量流出，则表示通过顶点u已经无法增广了，于是炸点，dep可以设置为任何无意义值
  if(rflow == flow)
  dep[u] = -2;
  return flow - rflow; // 返回流出的流量
}

ll dinic()
{
  ll ans = 0;
  while(bfs())
  {
    // 新一轮dfs之前要对cur进行初始化
    for(int i=1;i<=n;i++)cur[i] = head[i];
    int a = dfs(s, (1<<30));
    ans += a;
  }
  return ans;
}

int main()
{
  scanf("%d%d%d%d", &n, &m, &s, &e);
  init();
  for(int i=1;i<=m;i++)
  {
    int u, v, w;
    scanf("%d%d%d", &u, &v, &w);
    addEdge(u, v, w);
    addEdge(v, u, 0);
  }
  printf("%lld\n", dinic());
  return 0;
}
```

1.8 二分图匹配

1.8.1 匈牙利算法

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> P;
const int maxn = 1e3+5;
const int inf = 0x3f3f3f3f;
const int mod = 1e9+7;


int mp[maxn][maxn];
int use[maxn], link[maxn];
int n, m, e, ans;

int found(int u)
{
  for(int i=1;i<=m;i++)
  {
    if(!use[i] && mp[u][i])
    {
      use[i] = 1;
      if(!link[i] || found(link[i]))
      {
        link[i] = u;
        return 1;
      }
    }
  }
  return 0;
}

int main()
{
  scanf("%d%d%d", &n, &m, &e);
  for(int i=1;i<=e;i++)
  {
    int u, v;
    scanf("%d%d", &u, &v);
    if(u<=n && v<=m)
    {
      mp[u][v] = 1;
```

```
    }
  }
  for(int i=1;i<=n;i++)
  {
    memset(use, 0, sizeof(use));
    if(found(i))ans++;
  }
  printf("%d\n", ans);
  return 0;
}
```

1.9 最小生成树

1.9.1 Prim

```
const int MAX=10000007;
int dis[5002],map[5002][5002],mark[5002];
int prim(int n)
{
  for(int i=1;i<=n;i++)  //初始化每个点到生成树中点的距离
  {
    dis[i]=map[1][i];
    mark[i]=0;
  }
  dis[1]=0;
  mark[1]=1; //1这个点加入生成树中。
  int sum=0;
  for(int i=1;i<n;i++) //枚举n-1条边
  {
    int sta=-1,Min=MAX;
    for(int j=1;j<=n;j++)  //找不在生成树中的点中距离生成树中的点长度最小的
    {
      if(!mark[j]&&dis[j]<Min)
      {
        Min=dis[j];
        sta=j;
      }
    }
    if(sta==-1) return -1; //没找到可以可以联通的路
    mark[sta]=1;   //新找到的点加入生成树
    sum+=Min;
    for(int j=1;j<=n;j++)  //更新树外的点到树中的点的距离
    {
      if(!mark[j]&&dis[j]>map[sta][j])
      dis[j]=map[sta][j];
    }
  }
  return sum;
}

int main()
{
  int n,m;
  cin>>n>>m;
  for(int i=1;i<=n;i++)
  {
    for(int j=1;j<=n;j++)
    {
      map[i][j]=MAX;
    }
  }
  for(int i=1;i<=m;i++)
  {
    int a,b,c;
    cin>>a>>b>>c;
    if(c<map[a][b])
    {
      map[a][b]=c;
      map[b][a]=c;
    }
  }
  int ans = prim(n);
  if(ans==-1)
    cout<<"orz"<<endl;
  else
    cout<<ans<<endl;
  return 0;
}
```

1.9.2 kruskal

```
#include<bits/stdc++.h>
using namespace std;
#define N 5005
int father[N];
```

```

int find(int x)
{
  int k = x;
  while(father[k]!=k)
  {
    k = father[k];
  }
  while(father[x]!=x)
  {
    int temp = x;
    x = father[x];
    father[temp] = k;
  }
  return k;
}

void join(int a, int b)
{
  int f1, f2;
  f1 = find(a);
  f2 = find(b);
  father[f1] = f2;
}

struct edge
{
  int node1, node2;
  int cost;
};

vector<edge> edges;

bool cmp(edge a, edge b)
{
  return a.cost > b.cost;
}

int kruskal(int n)
{
  sort(edges.begin(), edges.end(), cmp);
  for(int i=1;i<=n;i++)
    father[i] = i;
  int sum=0;
  while(n!=1 && !edges.empty())
  {
    edge temp = edges[edges.size()-1];
    edges.pop_back();
    if(find(temp.node1)!=find(temp.node2))
    {
      sum += temp.cost;
      n--;
      join(temp.node1, temp.node2);
    }
  }
  if(n!=1 && edges.empty())
    sum = -1;
  return sum;
}

int main()
{
  int n,m;
  int result;
  cin>>n>>m;
  for(int i=1;i<=m;i++)
  {
    int a,b,c;
    cin>>a>>b>>c;
    edge t;
    t.node1=a;t.node2=b;t.cost=c;
    edges.push_back(t);
  }
  result = kruskal(n);
  if(result == -1)
    cout<<"orz"<<endl;
  else
    cout<<result<<endl;
  return 0;
}
```

1.10 次小生成树-POJ1679

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
```

```
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 110;
const ll mod = 1e9+7;

int n, m;
int g[maxn][maxn];
int d[maxn], v[maxn], maxd[maxn][maxn], pre[maxn], mst[maxn][maxn];
int ans = 0;

void prim()
{
  for(int i=1;i<=n;i++)
  {
    v[i] = 0; d[i] = inf; pre[i] = 1;
  }
  memset(maxd, 0, sizeof(maxd));
  memset(mst, 0, sizeof(mst));
  ans = 0;
  priority_queue<P, vector<P>, greater<P> > q;
  d[1] = 0; q.push(P(0, 1));
  while(!q.empty())
  {
    P p = q.top(); q.pop();
    int u = p.second;
    if(v[u]) continue;
    v[u] = 1; ans += d[u];
    mst[pre[u]][u] = mst[u][pre[u]] = 1;
    for(int i=1; i<=n;i++)
    {
      if(v[i] && g[u][i] < inf)
        maxd[u][i] = maxd[i][u] = max(maxd[pre[u]][u], d[u]);
      if(d[i] > g[u][i])
      {
        d[i] = g[u][i];
        pre[i] = u;
        q.push(P(d[i], i));
      }
    }
  }
}

int main()
{
  int t;
  cin>>t;
  while(t--)
  {
    memset(g, inf, sizeof(g));
    cin>>n>>m;
    while(m--)
    {
      int a, b, c;
      cin>>a>>b>>c;
      g[a][b] = g[b][a] = c;
    }
    prim();
    int flag = 0;
    for(int i=1;i<=n&&!flag;i++)
    {
      for(int j=1;j<=n;j++)
      {
        if(mst[i][j] || g[i][j]==inf)continue;
        if(g[i][j] == maxd[i][j])
        {
          flag = 1;
          break;
        }
      }
    }
    if(flag) cout<<"Not Unique!"<<endl;
    else cout<<ans<<endl;
  }
  return 0;
}
```

1.11 拓扑排序

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=30;

int head[maxn],ip,indegree[maxn];
int n,m,seq[maxn];

struct note
```

```
{
  int v,next;
}edge[maxn*maxn];

void init()
{
  memset(head,-1,sizeof(head));
  ip=0;
}

void addedge(int u,int v)
{
  edge[ip].v=v,edge[ip].next=head[u],head[u]=ip++;
}

int topo()
{
  queue<int>q;
  int indeg[maxn];
  for(int i=0; i<n; i++)
  {
    indeg[i]=indegree[i];
    if(indeg[i]==0)
    q.push(i);
  }
  int k=0;
  bool res=false;
  while(!q.empty())
  {
    if(q.size()!=1)res=true;
    int u=q.front();
    q.pop();
    seq[k++]=u;
    for(int i=head[u]; i!=-1; i=edge[i].next)
    {
      int v=edge[i].v;
      indeg[v]--;
      if(indeg[v]==0)
        q.push(v);
    }
  }
  if(k<n)return -1;// no
  if(res)return 0;// more
  return 1; // only
}
```

1.12 Floyd 找最小环

```
const int INF = 0x3f3f3f3f;
const int MAXN = 110;

int n, m;               //  n:节点个数, m:边的个数
int g[MAXN][MAXN];      //  无向图
int dist[MAXN][MAXN];   //  最短路径
int r[MAXN][MAXN];      //  r[i][j]: i到j的最短路径的第一步
int out[MAXN], ct;      //  记录最小环

int solve(int i, int j, int k)
{   //  记录最小环
  ct = 0;
  while (j != i)
  {
    out[ct++] = j;
    j = r[i][j];
  }
  out[ct++] = i;
  out[ct++] = k;
  return 0;
}

int main()
{
  while (scanf("%d%d", &n, &m) != EOF)
  {
    int i, j, k;
    for (i = 0; i < n; i++)
    {
      for (j = 0; j < n; j++)
      {
        g[i][j] = INF;
        r[i][j] = i;
      }
    }
    for (i = 0; i < m; i++)
    {
      int x, y, l;
```

```
      scanf("%d%d%d", &x, &y, &l);
      --x;
      --y;
      if (l < g[x][y])
      {
        g[x][y] = g[y][x] = l;
      }
    }
    memmove(dist, g, sizeof(dist));
    int Min = INF;              //  最小环
    for (k = 0; k < n; k++)
    {                           //  Floyd
      for (i = 0; i < k; i++) //  一个环中的最大结点为k(编号最大)
      {
        if (g[k][i] < INF)
        {
          for (j = i + 1; j < k; j++)
          {
            if (dist[i][j] < INF && g[k][j] < INF && Min > dist[i][j] + g[k
              ][i] + g[k][j])
            {
              Min = dist[i][j] + g[k][i] + g[k][j];
              solve(i, j, k);     //  记录最小环
            }
          }
        }
      }
      for (i = 0; i < n; i++)
      {
        if (dist[i][k] < INF)
        {
          for (j = 0; j < n; j++)
          {
            if (dist[k][j] < INF && dist[i][j] > dist[i][k]+dist[k][j])
            {
              dist[i][j] = dist[i][k] + dist[k][j];
              r[i][j] = r[k][j];
            }
          }
        }
      }
    }
    if (Min < INF)
    {
      for (ct--; ct >= 0; ct--)
      {
        printf("%d", out[ct] + 1);
        if (ct)
        {
          printf(" ");
        }
      }
    }
    else
    {
      printf("No solution.");
    }
    printf("\n");
  }
  return 0;
}
```

2 数据结构

2.1 并查集

```
int pre[maxn];

int Find(int x)
{
  int p,tmp;
  p=x;
  while(x!=pre[x])
    x=pre[x];
  while(p!=x)
  {
    tmp=pre[x];
    pre[x]=x;
    p=tmp;
  }
  return x;
}

void join(int x,int y)
{
  int fx=Find(x);
```

```
  int fy=Find(y);
  if(fx!=fy)
    pre[fx]=fy;
}
```

2.2 ST 表

```
int st[maxn][20];
void st_init()
{
  for(int i=1;i<=n;i++) st[i][0]=a[i]; // 长度为1的区间最小值党委就为自身
  // 预处理从i开始，长度为2^j的区间
  for(int j=1;(1<<j)<=n;j++)
    for(int i=1;i+(1<<j)-1<=n;i++)
      st[i][j]=max(st[i][j-1],st[i+(1<<(j-1))][j-1]);
}

int query(int l,int r)
{
  int k=log2(r-l+1);
  return min(st[l][k],st[r-(1<<k)+1][k]);
}
```

2.3 树状数组

```
/*
*  INIT: ar[]置为0;
*  CALL: add(i, v): 将i点的值加v; sum(i): 求[1, i]的和;
*/
#define typev int   //  type of res
const int N = 1010;
typev ar[N];        //  index: 1 ~ N
int lowb(int t)
{
  return t & (-t);
}

void add(int i, typev v)
{
  for (; i < N; ar[i] += v, i += lowb(i));
  return ;
}

typev sum(int i)
{
  typev s = 0;
  for (; i > 0; s += ar[i], i -= lowb(i));
  return s;
}
```

2.4 线段树

2.4.1 单点修改 + 区间求和 HDU1166

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

int num[50005];
ll tree[4 * 50000 + 5];

void build(int p, int l, int r)
{
  if(l == r)
  {
    tree[p] = num[l];
    return ;
  }
  else
  {
    int mid = (r+l) >> 1;
    build(p<<1, l, mid);
    build(p<<1|1, mid+1, r);
    tree[p] = tree[p<<1] + tree[p<<1|1];
  }
}

void add(int p, int l, int r, int ind, int v)
{
  if(l == r)
  {
    tree[p] += v;
    return ;
  }
```

```
  else
  {
    int mid = (r+l) >> 1;
    if(ind <= mid) add(p<<1, l, mid, ind, v);
    else add(p<<1|1, mid+1, r, ind, v);
    tree[p] = tree[p<<1] + tree[p<<1|1];
  }
}

ll query(int p, int l, int r, int x, int y)
{
  if(x <= l && r <= y)
  {
    return tree[p];
  }
  else
  {
    int mid = (l+r) >> 1;
    ll ans = 0;
    if(x <= mid)
      ans += query(p<<1, l, mid, x, y);
    if(mid < y)
      ans += query(p<<1|1, mid+1, r, x, y);
    return ans;
  }
}

int main()
{
  int t;
  scanf("%d", &t);
  for(int i=1;i<=t;i++)
  {
    int n;
    scanf("%d", &n);
    for(int j=1;j<=n;j++)
      scanf("%d", &num[j]);
    build(1, 1, n);
    string s;
    printf("Case %d:\n", i);
    while(cin>>s && s[0] != 'E')
    {
      if(s[0] == 'Q')
      {
        int x, y;
        scanf("%d%d", &x, &y);
        printf("%lld\n", query(1, 1, n, x, y));
      }
      else if(s[0] == 'A')
      {
        int x, y;
        scanf("%d%d", &x, &y);
        add(1, 1, n, x, y);
      }
      else
      {
        int x, y;
        scanf("%d%d", &x, &y);
        add(1, 1, n, x, -y);
      }
    }
  }
  return 0;
}
```

2.4.2 区间修改 + 区间求和 POJ3465

```
#include<iostream>
using namespace std;
typedef long long ll;
ll num[100005];
ll tree[4 * 100000 + 5];
ll lazy[4 * 100000 + 5]={0};
int n,m;
void build(int p, int l, int r)
{
  if(l == r)
  tree[p] = num[l];
  else
  {
    int mid = (l+r) >> 1;
    build(p<<1, l, mid);
    build(p<<1|1, mid+1, r);
    tree[p]  = tree[p<<1] + tree[p<<1|1];
  }
}
```

```

void pushdown(int p, int l, int r)
{
  if(lazy[p])
  {
    lazy[p<<1] += lazy[p];
    lazy[p<<1|1] += lazy[p];
    tree[p<<1] += lazy[p] * (((l+r)>>1) - l + 1);
    tree[p<<1|1] += lazy[p] * (r - ((l+r)>>1));
    lazy[p] = 0;
  }
}

void add(int p, int l, int r, int x, int y, ll v)
{
  if(x <= l && r <= y)
  {
    lazy[p] += v;
    tree[p] += v * (r-l+1);
    return ;
  }
  else
  {
    int mid = (l+r) >> 1;
    pushdown(p, l, r);
    if(x <= mid)
      add(p<<1, l, mid, x, y, v);
    if(y > mid)
      add(p<<1|1, mid+1, r, x, y, v);
    tree[p] = tree[p<<1] + tree[p<<1|1];
  }
}

ll query(int p, int l, int r, int x, int y)
{
  if(x <= l && r <= y)
  {
    return tree[p];
  }
  else
  {
    int mid = (l+r) >> 1;
    ll ans = 0;
    pushdown(p, l, r);
    if(x <= mid)
      ans += query(p<<1, l, mid, x, y);
    if(y > mid)
      ans += query(p<<1|1, mid+1, r, x, y);
    return ans;
  }
}

int main()
{
  cin>>n>>m;
  for(int i=1;i<=n;i++)
  cin>>num[i];
  build(1, 1, n);
  for(int i=1;i<=m;i++)
  {
    char a;
    cin>>a;
    if(a == 'Q')
    {
      int x, y;
      cin>>x>>y;
      cout<<query(1, 1, n, x, y)<<endl;
    }
    else
    {
      int x, y;
      ll c;
      cin>>x>>y>>c;
      add(1, 1, n, x, y, c);
    }
  }
  return 0;
}
```

2.4.3 单点修改 + 区间最值 HDU1754

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
int n,m;

```

```
int num[200005];
int tree[200000 * 4 + 5];

void build(int p, int l, int r)
{
  if(l == r)
  {
    tree[p] = num[l];
    return ;
  }
  else
  {
    int mid = (l+r) >> 1;
    build(p<<1, l, mid);
    build(p<<1|1, mid+1, r);
    tree[p] = max(tree[p<<1], tree[p<<1|1]);
    return ;
  }
}

void update(int p, int l, int r, int ind, int v)
{
  if(l == r)
  {
    tree[p] = v;
    return ;
  }
  else
  {
    int mid = (l+r) >> 1;
    if(ind <= mid)
      update(p<<1, l, mid, ind, v);
    else
      update(p<<1|1, mid+1, r, ind, v);
    tree[p] = max(tree[p<<1], tree[p<<1|1]);
  }
}

ll query(int p, int l, int r, int x, int y)
{
  if(x <= l && r <= y)
  {
    return tree[p];
  }
  else
  {
    int mid = (l+r) >> 1;
    ll ans = -1;
    if(x <= mid)
      ans = max(ans, query(p<<1, l, mid, x, y));
    if(y > mid)
      ans = max(ans, query(p<<1|1, mid+1, r, x, y));
    return ans;
  }
}

int main()
{
  while(scanf("%d %d", &n, &m) != EOF)
  {
    for(int i=1;i<=n;i++)
    {
      scanf("%d", &num[i]);
    }
    build(1, 1, n);
    for(int i=1;i<=m;i++)
    {
      char a;
      int x, y;
      scanf(" %c%d%d", &a, &x, &y);
      if(a == 'Q')
      {
        printf("%lld\n", query(1, 1, n, x, y));
      }
      else
      {
        update(1, 1, n, x, y);
      }
    }
  }
  return 0;
}
```

2.4.4 区间染色 + 统计 + 离散化 POJ2528

```
#include<iostream>
```

```
#include<algorithm>
#include<string.h>
using namespace std;
typedef long long ll;

const int maxn = 20000 + 100;
int tree[maxn<<4];
int li[maxn],ri[maxn];
bool vis[maxn];
int lisan[maxn*3];
int ans = 0;

void init()
{
  memset(tree, -1, sizeof(tree));
  memset(vis, 0, sizeof(vis));
  ans = 0;
}

void pushdown(int p)
{
  tree[p<<1] = tree[p<<1|1] = tree[p];
  tree[p] = -1;
}

void update(int p, int l, int r, int x, int y, int v)
{
  if(x <= l && r <= y)
  {
    tree[p] = v;
    return ;
  }
  if(tree[p]!=-1)
    pushdown(p);
  int mid = (l+r)>>1;
  if(x <= mid)
    update(p<<1, l, mid, x, y, v);
  if(y > mid)
    update(p<<1|1, mid+1, r, x, y, v);
  tree[p] = -1;
}

void query(int p, int l, int r)
{
  if(tree[p]!=-1)
  {
    if(vis[tree[p]]==0)
    {
      vis[tree[p]] = 1;
      ans++;
    }
    return;
  }
  if(l==r)return;
  int mid = (l+r)>>1;
  query(p<<1, l, mid);
  query(p<<1|1, mid+1, r);
}

int main()
{
  int t;
  cin>>t;
  while(t--)
  {
    init();
    int n;
    cin>>n;
    int tot = 0;
    for(int i=0;i<n;i++)
    {
      cin>>li[i]>>ri[i];
      lisan[tot++] = li[i];
      lisan[tot++] = ri[i];
    }
    sort(lisan, lisan+tot);
    int m = unique(lisan, lisan+tot) - lisan;
    int t = m;
    for(int i=1;i<t;i++)
    {
      if(lisan[i]-lisan[i-1]>1)
      {
        lisan[m++] = lisan[i-1]+1;
      }
    }
    sort(lisan, lisan+m);
    for(int i=0;i<n;i++)
    {
```

```
      int x,y;
      x = lower_bound(lisan, lisan+m, li[i]) - lisan;
      y = lower_bound(lisan, lisan+m, ri[i]) - lisan;
      update(1, 0, m-1, x, y, i);
    }
    query(1, 0, m-1);
    cout<<ans<<endl;
  }
  return 0;
}
```

2.4.5 线段树 + 扫描线求矩阵覆盖周长 POJ1177

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<cmath>
using namespace std;
typedef long long ll;

const int maxn = 10005;
vector<int> x;
int getID(int v)
{
  return lower_bound(x.begin(), x.end(), v) - x.begin();
}

struct Segment
{
  int l, r;
  int h;
  int flag;
}segment[maxn];
bool cmp(Segment a, Segment b)
{
  return a.h < b.h;
}

struct Node
{
  int l,r;
  int lr, rr;
  int len;
  int line;
  int s;
}node[maxn<<2];

void build(int p, int l, int r)
{
  node[p].l = l; node[p].r = r;
  node[p].line = node[p].len = node[p].s = 0;
  node[p].lr = node[p].rr = 0;
  if(l==r)return;
  int mid = (l+r)>>1;
  build(p<<1, l, mid);
  build(p<<1|1, mid+1, r);
}

void pushup(int p)
{
  if(node[p].s)
  {
    node[p].line = 1;
    node[p].rr = node[p].lr = 1;
    node[p].len = x[node[p].r+1] - x[node[p].l];
    return;
  }
  else if(node[p].l == node[p].r)
  {
    node[p].lr = node[p].rr = node[p].line = node[p].len = 0;
  }
  else
  {
    node[p].lr = node[p<<1].lr;
    node[p].rr = node[p<<1|1].rr;
    node[p].len = node[p<<1].len + node[p<<1|1].len;
    node[p].line = node[p<<1].line + node[p<<1|1].line - (node[p<<1].rr&&
      node[p<<1|1].lr);
  }
}

void update(int p, int l, int r, int v)
{
  if(node[p].r < l || node[p].l > r)return;
  if(l <= node[p].l && node[p].r <= r)
  {
```

```
    node[p].s += v;
    pushup(p);
    return;
  }
  update(p<<1, l, r, v);
  update(p<<1|1, l, r, v);
  pushup(p);
}

int main()
{
  int n;
  cin>>n;
  for(int i=1;i<=n;i++)
  {
    int x1,x2,y1,y2;
    cin>>x1>>y1>>x2>>y2;
    Segment &s1 = segment[2*i-1];
    Segment &s2 = segment[i<<1];
    s1.l = s2.l = x1;
    s1.r = s2.r = x2;
    s1.h = y1; s2.h = y2;
    s1.flag = 1; s2.flag = -1;
    x.push_back(x1);
    x.push_back(x2);
  }
  sort(segment+1, segment+2*n+1, cmp);

  sort(x.begin(), x.end());
  x.erase(unique(x.begin(), x.end()), x.end());

  build(1, 0, x.size()-1);

  ll ans = 0;
  int last = 0;
  for(int i=1;i<=2*n;i++)
  {
    int l = getID(segment[i].l);
    int r = getID(segment[i].r);
    update(1, l, r-1, segment[i].flag);
    ans += abs(node[1].len - last);
    if(i!=2*n)
      ans += node[1].line * 2 * (segment[i+1].h - segment[i].h);
    last = node[1].len;
  }
  cout<<ans<<endl;
  return 0;
}
```

2.4.6 线段树 + 扫描线求矩阵面积并 HDU1542

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;

const int maxn = 210;
int n;
vector<double> x;
inline int getID(double v)
{
  return lower_bound(x.begin(), x.end(), v) - x.begin();
}

struct Segment
{
  double l, r;
  double h;
  int flag;
}segment[maxn];
bool cmp(Segment a, Segment b)
{
  return a.h < b.h;
}

struct Node
{
  int l, r;
  int s;
  double len;
}node[maxn<<2];

void pushup(int p)
{
  if(node[p].s)
    node[p].len = x[node[p].r+1] - x[node[p].l];
  else if(node[p].l == node[p].r)
```

```
    node[p].len = 0;
  else
    node[p].len = node[p<<1].len + node[p<<1|1].len;
}

void build(int p, int l, int r)
{
  if(l>r)return;
  node[p].l = l; node[p].r = r;
  node[p].s = 0; node[p].len = 0;
  if(l==r) return;
  int mid = (l+r)>>1;
  build(p<<1, l, mid);
  build(p<<1|1, mid+1, r);
  pushup(p);
}

void update(int p, int l, int r, int v)
{
  if(l>node[p].r || r<node[p].l) return;
  if(l <= node[p].l && node[p].r <= r)
  {
    node[p].s += v;
    pushup(p);
    return;
  }
  update(p<<1, l, r, v);
  update(p<<1|1, l, r, v);
  pushup(p);
}

int main()
{
  int cas = 0;
  while(scanf("%d", &n) && n)
  {
    x.clear();
    for(int i=1;i<=n;i++)
    {
      double x1,x2,y1,y2;
      scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
      Segment &s1 = segment[2*i-1];
      Segment &s2 = segment[i<<1];
      s1.l=s2.l=x1;
      s1.r=s2.r=x2;
      s1.h=y1;
      s2.h=y2;
      s1.flag=1;
      s2.flag=-1;
      x.push_back(x1);
      x.push_back(x2);
    }
    sort(segment+1, segment+2*n+1, cmp);

    sort(x.begin(), x.end());
    x.erase(unique(x.begin(), x.end()), x.end());

    build(1, 0, x.size()-1);
    double ans = 0;
    for(int i=1;i<=2*n;i++)
    {
      int l=getID(segment[i].l);
      int r=getID(segment[i].r);
      update(1, l, r-1, segment[i].flag);
      ans+=node[1].len*(segment[i+1].h - segment[i].h);
    }
    printf("Test case #%d\n", ++cas);
    printf("Total explored area: %.2f\n\n", ans);
  }
  return 0;
}
```

2.5 主席树

2.5.1 区间第 K 大

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 1e6+5;
const ll mod = 1e9+7;

// 顶点，代表区间[l, r]中有v个数字
struct node
{
```

```
  int l, r, v;
}tree[maxn*20]; // 空间开大一点，因为要动态开点

// edit[i]存的是第i颗权值线段树的根节点在tree数组中的位置
// a是存放原数据的数组， b是离散化后的数组， tot表示顶点的个数
int edit[maxn], a[maxn], b[maxn], tot=0;

// 建树
int build(int l, int r)
{
  // 这里就是动态开辟新的结点，就是将tot加一
  int pos = ++tot;
  tree[pos].v = 0; //初始化为0
  if(l==r)return pos; //到根节点了，返回
  // 二分建树没什么好说的
  int mid = (l+r)>>1;
  tree[pos].l = build(l, mid);
  tree[pos].r = build(mid+1, r);
  //要返回pos的位置，因为edit数组要存新根的位置
  return pos;
}

// 就是插入操作，插入新的元素
// ed是前一版本的结点在tree的位置， 更新区间[l, r]， 位置为v
int update(int ed, int l, int r, int v)
{
  // 动态开点
  int pos = ++tot;
  //先将新一版本的当前结点复制为上一个版本的对应结点
  tree[pos] = tree[ed]; tree[pos].v++; //新一版本的结点的v要加一，因为对应区间插入了一个数
  if(l==r) return pos; //到叶子节点了，返回
  // 二分
  int mid = (l+r)>>1;
  // 如果更新位置v在左子树中，递归更新即可，在右子树中同理，最后要返回pos
  if(v<=mid) tree[pos].l = update(tree[ed].l, l, mid, v);
  else tree[pos].r = update(tree[ed].r, mid+1, r, v);
  return pos;
}

// 查询区间[l, r]第k大的数字，pre对应l-1版本的权值线段树的节点位置， ed代表r版本的权值线段树的节点位置
int query(int pre, int ed, int l, int r, int k)
{
  // 到叶子节点就返回
  if(l==r) return l;
  // 二分
  int mid = (l+r)>>1;
  // 先计算左子树的数字个数
  int x = tree[tree[ed].l].v - tree[tree[pre].l].v;
  // 若左子树的数字个数大于等于k，说明我们要找的数字在左子树中，递归走到左子树继续寻找
  if(x>=k) return query(tree[pre].l, tree[ed].l, l, mid, k);
  // 否则就在右子树中，我们要在右子树中寻找k-x大的数字，递归寻找就好
  else return query(tree[pre].r, tree[ed].r, mid+1, r, k-x);
}

int main()
{
  int n, q; // n个数字， q次询问
  scanf("%d%d", &n, &q); // 这一题cin/cout会被卡
  // 输入数据，并copy到b数组中
  for(int i=1;i<=n;i++)
  {
    scanf("%d", &a[i]);
    b[i] = a[i];
  }
  // 离散化
  sort(b+1, b+1+n);
  int m = unique(b+1, b+n+1) - b-1;
  // 构建一颗空的权值线段树，edit[0]存放的就是这颗空树的根节点的位置
  edit[0] = build(1, m);
  // 插入n个数据
  for(int i=1;i<=m;i++)
  {
    // 找到a[i]离散化后对应的位置
    a[i] = lower_bound(b+1, b+m+1, a[i]) - b;
    // edit[i]存放第i版本的权值线段树的根节点位置
    edit[i] = update(edit[i-1], 1, m, a[i]);
  }
  // 处理q次询问
  while(q--)
  {
    int x, y, k;
    scanf("%d%d%d", &x, &y, &k);
    // pos对应的是离散化后的位置，所以最后输出b[pos]即可
    int pos = query(edit[x-1], edit[y], 1, m, k);
    printf("%d\n", b[pos]);
```

```
  }
  return 0;
}
```

2.5.2 动态区间第 K 大 (主席树套树状数组) ZOJ2112

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 60010;
const ll mod = 1e9+7;
const int M = 2500010;

int n, m, q, tot;
struct node
{
  int l, r, v;
}tree[M];

// T是主席树，与上面代码的edit作用一样， S数组就是树状数组， use数组是树状数组求和时用的，记录的是树状数组中哪一些权值线段树要被用来求和
int T[maxn], S[maxn], use[maxn], a[maxn], b[maxn];

// 记录询问，因为要将修改后的值一起构建主席树，所以将在线转为离线
struct Q
{
  // 对于查询区间[l, r]第k大的询问来说，flag为1
  //若是修改操作，l记录修改的位置，r记录新值，flag为0
  int l, r, k, flag;
}query[10010];

// 快速找出x在离散化后的位置
int HASH(int x)
{
  return lower_bound(b+1, b+m+1, x) - b;
}

// 建静态主席树，和之前的一样
int build(int l, int r)
{
  int pos = ++tot;
  tree[pos].v = 0;
  if(l==r) return pos;
  int mid = (l+r)>>1;
  tree[pos].l = build(l, mid);
  tree[pos].r = build(mid+1, r);
  return pos;
}

// 和之前的静态主席树update差不多，只不过不是直接将tree[pos].v+1,而是加参数v
//消除影响v就为-1， 添加影响v就为1， 其他没什么不同
int update(int ed, int l, int r, int p, int v)
{
  int pos = ++tot;
  tree[pos] = tree[ed];
  tree[pos].v += v;
  if(l==r) return pos;
  int mid = (l+r)>>1;
  if(p<=mid) tree[pos].l = update(tree[ed].l, l, mid, p, v);
  else tree[pos].r = update(tree[ed].r, mid+1, r, p, v);
  return pos;
}

// 树状数组的lowbit
int lowbit(int x) { return x&(-x); }

// 修改操作，修改位置x的影响
int add(int x, int v)
{
  // 找出a[x]在离散化后的位置p
  int p = HASH(a[x]);
  while(x<=n)
  {
    // 修改操作，对树状数组中相应的权值线段树进行修改，消除影响：v=-1， 添加影响：v=1
    // 因为树状数组中的权值线段树不需要可持久化，所以直接在原版本上修改就可以了
    S[x] = update(S[x], 1, m, p, v);
    x+=lowbit(x);
  }
}

// 树状数组求和， 求左子树包含的数字个数，和静态主席树一样的思想，都是先求左子树
```

```
int sum(int x)
{
  int ret = 0;
  while(x)
  {
    // use[i]记录的就是树状数组中相应的权值线段树的结点位置
    // 似乎这一句有一点点难以理解，结合整体代码多看几遍吧
    ret += tree[tree[use[x]].l].v;
    x-=lowbit(x);
  }
  return ret;
}

// 询问操作。树状数组求[pre, ed]的和， tpre和ted是静态主席树的区间左右顶点的位置， 区间[l, r]第k大
int Query(int pre, int ed, int tpre, int ted, int l, int r, int k)
{
  if(l==r)return l;
  int mid = (l+r)>>1;
  // sum就是树状数组的求和，相对于静态主席树，多了sum求修改操作的影响
  // tmp为当前左子树的数字个数
  int tmp = sum(ed) - sum(pre) + tree[tree[ted].l].v - tree[tree[tpre].l].v;
  // 若左子树的数字个数大于等于k就往左子树继续走，这里和主席树没什么区别
  if(tmp >= k)
  {
    // 两个for循环是更新左子树需要用到的树状数组中的权值线段树的位置，有那么一丢丢难以理解，多看几遍？
    for(int i=ed; i; i-=lowbit(i)) use[i] = tree[use[i]].l;
    for(int i=pre; i; i-=lowbit(i)) use[i] = tree[use[i]].l;
    return Query(pre, ed, tree[tpre].l, tree[ted].l, l, mid, k);
  }
  else
  {
    // 走右子树同理
    for(int i=ed; i; i-=lowbit(i)) use[i] = tree[use[i]].r;
    for(int i=pre; i; i-=lowbit(i)) use[i] = tree[use[i]].r;
    return Query(pre, ed, tree[tpre].r, tree[ted].r, mid+1, r, k-tmp);
  }
}

int main()
{
  int t; // t个case
  scanf("%d", &t);
  while(t--)
  {
    scanf("%d%d", &n, &q);
    m = tot = 0; // 记得初始化
    for(int i=1;i<=n;i++)
    {
      scanf("%d", &a[i]);
      b[++m] = a[i];
    }
    char op[5];
    for(int i=1;i<=q;i++)
    {
      scanf("%s", op);
      // 查询操作
      if(op[0]=='Q')
      {
        scanf("%d%d%d", &query[i].l, &query[i].r, &query[i].k);
        query[i].flag = 1;
      }
      else
      {
        scanf("%d%d", &query[i].l, &query[i].r);
        b[++m] = query[i].r; // 注意要将修改后的新值加入到待离散化的b数组
        query[i].flag = 0;
      }
    }
    // 离散化
    sort(b+1, b+m+1);
    m = unique(b+1, b+m+1) - b-1;
    // 构建主席树
    T[0] = build(1, m);
    for(int i=1;i<=n;i++)
      T[i] = update(T[i-1], 1, m, HASH(a[i]), 1);
    // 构建树状数组，每一个节点都是一颗空的权值线段树
    for(int i=1;i<=n;i++)
      S[i] = T[0];
    // 离线处理q个询问
    for(int i=1;i<=q;i++)
    {
      if(query[i].flag) // 查询
      {
        // 两个for循环标记区间[l, r]要使用的树状数组中的权值线段树的位置
        for(int j=query[i].r; j; j-=lowbit(j)) use[j] = S[j];
        for(int j=query[i].l-1; j; j-=lowbit(j)) use[j] = S[j];
```

```
        printf("%d\n", b[Query(query[i].l-1, query[i].r, T[query[i].l-1], T[
          query[i].r], 1, m, query[i].k)]);
      }
      else
      {
        // 先消除影响
        add(query[i].l, -1);
        // 在原数组中更新值
        a[query[i].l] = query[i].r;
        //添加新值的影响
        add(query[i].l, 1);
      }
    }
  }
  return 0;
}
```

3 DP

3.1 背包

```
const int MAXN = 10000;
const int SIZE = 100000;

int dp[SIZE];
int volume[MAXN], value[MAXN], c[MAXN];
int n, v;           //  总物品数，背包容量

//  01背包
void ZeroOnepark(int val, int vol)
{
  for (int j = v ; j >= vol; j--)
  {
    dp[j] = max(dp[j], dp[j - vol] + val);
  }
}

//  完全背包
void Completepark(int val, int vol)
{
  for (int j = vol; j <= v; j++)
  {
    dp[j] = max(dp[j], dp[j - vol] + val);
  }
}

//  多重背包
void Multiplepark(int val, int vol, int amount)
{
  if (vol * amount >= v)
  {
    Completepark(val, vol);
  }
  else
  {
    int k = 1;
    while (k < amount)
    {
      ZeroOnepark(k * val, k * vol);
      amount -= k;
      k <<= 1;
    }
    if (amount > 0)
    {
      ZeroOnepark(amount * val, amount * vol);
    }
  }
}

int main()
{
  while (cin >> n >> v)
  {
    for (int i = 1 ; i <= n ; i++)
    {
      cin >> volume[i] >> value[i] >> c[i];      //   费用，价值，数量
    }
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= n; i++)
    {
      Multiplepark(value[i], volume[i], c[i]);
    }
    cout << dp[v] << endl;
  }
  return 0;
}
```

3.2 最长公共递增子序列

```
/*
*  最长公共递增子序列 O(n^2)
*  f记录路径,DP记录长度, 用a对b扫描,逐步最优化。
*/
const int N = 1010;

int f[N][N], dp[N];

int gcis(int a[], int la, int b[], int lb, int ans[])
{   //  a[1...la], b[1...lb]
  int i, j, k, mx;
  memset(f, 0, sizeof(f));
  memset(dp, 0, sizeof(dp));
  for (i = 1; i <= la; i++)
  {
    memcpy(f[i], f[i-1], sizeof(f[0]));
    for (k = 0, j = 1; j <= lb; j++)
    {
      if (b[j - 1] < a[i - 1] && dp[j] > dp[k])
      {
        k = j;
      }
      if (b[j - 1] == a[i - 1] && dp[k] + 1 > dp[j])
      {
        dp[j] = dp[k] + 1,
        f[i][j] = i * (lb + 1) + k;
      }
    }
  }
  for (mx = 0, i = 1; i <= lb; i++)
  {
    if (dp[i] > dp[mx])
    {
      mx = i;
    }
  }
  for (i = la * lb + la + mx, j = dp[mx]; j; i = f[i / (lb + 1)][i % (lb +
    1)], j--)
  {
    ans[j - 1] = b[i % (lb + 1) - 1];
  }
  return dp[mx];
}
```

3.3 最长公共子序列

```
const int N = 1010;

int a[N][N];

int LCS(const char *s1, const char *s2)
{   //  s1:0...m, s2:0...n
  int m = (int)strlen(s1), n = (int)strlen(s2);
  int i, j;
  a[0][0] = 0;
  for (i = 1; i <= m; ++i)
  {
    a[i][0] = 0;
  }
  for (i = 1; i <= n; ++i)
  {
    a[0][i] = 0;
  }
  for (i = 1; i <= m; ++i)
  {
    for (j = 1; j <= n; ++j)
    {
      if (s1[i - 1] == s2[j - 1])
      {
        a[i][j] = a[i - 1][j - 1] + 1;
      }
      else if (a[i - 1][j] > a[i][j - 1])
      {
        a[i][j]= a[i - 1][j];
      }
      else
      {
        a[i][j] = a[i][j - 1];
      }
    }
  }
```

```
  return a[m][n];
}
```

3.4 最长有序子序列

```
int n;
int a[maxn]; // 原数组
int s[maxn]; // 记录LIS
int LIS()
{
  int top = 0;
  for(int i=0;i<n;i++)
  {
    int pos = upper_bound(s, s+top, a[i])-s;
    s[pos] = a[i];
    top = max(top, pos+1);
  }
  return top;
}
```

```
/*
*  递增（默认）
*  递减
*  非递增
*  非递减 (1)>= && <  (2)<  (3)>=
*/
const int MAXN = 1001;

int a[MAXN], f[MAXN], d[MAXN];   //  d[i] 用于记录 a[0...i] 以 a[i] 结尾的最大长度

int bsearch(const int *f, int size, const int &a)
{
  int l = 0, r = size - 1;
  while (l <= r)
  {
    int mid = (l + r) / 2;
    if (a > f[mid - 1] && a <= f[mid])  //  (1)
    {
      return mid;
    }
    else if (a < f[mid])
    {
      r = mid - 1;
    }
    else
    {
      l = mid + 1;
    }
  }
  return -1;
}

int LIS(const int *a, const int &n)
{
  int i, j, size = 1;
  f[0] = a[0];
  d[0] = 1;
  for (i = 1; i < n; ++i)
  {
    if (a[i] <= f[0])               //  (2)
    {
      j = 0;
    }
    else if (a[i] > f[size - 1])    //  (3)
    {
      j = size++;
    }
    else
    {
      j = bsearch(f, size, a[i]);
    }
    f[j] = a[i];
    d[i] = j + 1;
  }
  return size;
}

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

```
    }
    printf("%d\n", LIS(a, n));      // 求最大递增 / 上升子序列(如果为最大非降子序列,只需把上面的注释部分给与替换)
  }
  return 0;
}
```

4 字符串

5 数学

5.1 快速幂

```
typedef long long ll;
ll pow(ll x,ll n,ll mod)
{
  ll res=1;
  while(n>0)
  {
    if(n&1)
    {
      res=res*x;
      res=res%mod;
    }
    x=x*x;
    x=x%mod;
    n>>=1;
  }
  return res;
}
```

5.2 逆序数

```
/*
*  也可以用树状数组做
*  a[0...n-1] cnt=0; call: MergeSort(0, n)
*/
const int N = 1010;
int a[N];
int c[N];
int cnt = 0;

void MergeSort(int l, int r)
{
  int mid, i, j, tmp;
  if (r > l + 1)
  {
    mid = (l + r) / 2;
    MergeSort(l, mid);
    MergeSort(mid, r);
    tmp = l;
    for (i = l, j = mid; i < mid && j < r;)
    {
      if (a[i] > a[j])
      {
        c[tmp++] = a[j++];
        cnt += mid - i;
      }
      else
      {
        c[tmp++] = a[i++];
      }
    }
    if (j < r)
    {
      for (; j < r; ++j)
      {
        c[tmp++] = a[j];
      }
    }
    else
    {
      for (; i < mid; ++i)
      {
        c[tmp++]=a[i];
      }
    }
    for (i = l; i < r; ++i)
    {
      a[i] = c[i];
    }
  }
  return ;
```

```
}
```

5.3 无序序列变有序的最少交换次数

5.3.1 相邻元素交换

```
\\ 等于逆序数个数
```

5.3.2 任意元素交换

```
*
*  交换任意两数的本质是改变了元素位置，
*  故建立元素与其目标状态应放置位置的映射关系
*/
int getMinSwaps(vector<int> &A)
{
  //  排序
  vector<int> B(A);
  sort(B.begin(), B.end());
  map<int, int> m;
  int len = (int)A.size();
  for (int i = 0; i < len; i++)
  {
    m[B[i]] = i;    //  建立每个元素与其应放位置的映射关系
  }

  int loops = 0;      //  循环节个数
  vector<bool> flag(len, false);
  //  找出循环节的个数
  for (int i = 0; i < len; i++)
  {
    if (!flag[i])
    {
      int j = i;
      while (!flag[j])
      {
        flag[j] = true;
        j = m[A[j]];    //  原序列中j位置的元素在有序序列中的位置
      }
      loops++;
    }
  }
  return len - loops;
}
```

5.4 GCD

5.4.1 非递归

```
int gcd(int x, int y)
{
  if (!x || !y)
  {
    return x > y ? x : y;
  }

  for (int t; t = x % y, t; x = y, y = t) ;

  return y;
}
```

5.4.2 递归

```
int gcd(int a, int b)
{
  return b==0 ? a : gcd(b, a%b);
}
```

5.5 EXGCD

```
/*
*  求x，y使得gcd(a, b) = a * x + b * y;
*/
int extgcd(int a, int b, int &x, int &y)
{
  if (b == 0)
  {
    x = 1;
    y = 0;
    return a;
  }
```

```
12
13   int d = extgcd(b, a % b, x, y);
14   int t = x;
15   x = y;
16   y = t - a / b * y;
17
18   return d;
19 }
```

5.6 素数

5.6.1 判断小于 MAXN 的数是否为素数

```
1  /*
2  *  素数筛选，判断小于MAXN的数是不是素数
3  *  notprime是一张表，false表示是素数，true表示不是
4  */
5  const int MAXN = 1000010;
6  bool notprime[MAXN];
7
8  void init()
9  {
10   memset(notprime, false, sizeof(notprime));
11   notprime[0] = notprime[1] = true;
12   for (int i = 2; i < MAXN; i++)
13   {
14     if (!notprime[i])
15     {
16       if (i > MAXN / i)   //  阻止后边i * i溢出（或者i,j用long long)
17       {
18         continue;
19       }
20       //  直接从i * i开始就可以，小于i倍的已经筛选过了
21       for (int j = i * i; j < MAXN; j += i)
22       {
23         notprime[j] = true;
24       }
25     }
26   }
27 }
```

5.6.2 生成小于等于 MAXN 的全部素数

```
1  /*
2  *  素数筛选，查找出小于等于MAXN的素数
3  *  prime[0]存素数的个数
4  */
5
6  const int MAXN = 100000;
7  int prime[MAXN + 1];
8
9  void getPrime()
10 {
11   memset(prime, 0, sizeof(prime));
12   for (int i = 2; i <= MAXN; i++)
13   {
14     if (!prime[i])
15     {
16       prime[++prime[0]] = i;
17     }
18     for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++)
19     {
20       prime[prime[j] * i] = 1;
21       if (i % prime[j] == 0)
22       {
23         break;
24       }
25     }
26   }
27 }
```

5.6.3 Miller Rabin

```
1  /*
2  *  随机素数测试（伪素数原理）
3  *  CALL: bool res = miller(n);
4  *  快速测试n是否满足素数的“必要”条件，出错概率极低
5  *  对于任意奇数n > 2和正整数s，算法出错概率 2^(-s)
6  */
7
8  int witness(int a, int n)
9  {
10   int x, d = 1;
```

```
  int i = ceil(log(n - 1.0) / log(2.0)) - 1;
  for (; i >= 0; i--)
  {
    x = d;
    d = (d * d) % n;
    if (d == 1 && x != 1 && x != n - 1)
    {
      return 1;
    }
    if (((n - 1) & (1 << i)) > 0)
    {
      d = (d * a) % n;
    }
  }
  return (d == 1 ? 0 : 1);
}

int miller(int n, int s = 50)
{
  if (n == 2)     //  质数返回1
    return 1;
  if (n % 2 == 0) //  偶数返回0
    return 0;
  int j, a;
  for (j = 0; j < a; j++)
  {
    a = rand() * (n - 2) / RAND_MAX + 1;
    //  rand()只能随机产生[0, RAND_MAX)内的整数
    //  而且这个RAND_MAX只有32768直接%n的话是永远
    //  也产生不了[RAND_MAX, n)之间的数
    if (witness(a, n))
    {
      return 0;
    }
  }
  return 1;
}
```

5.7 组合数

```
int com(int n, int r)   //  return C(n, r)
{
  if (n - r > r)
  {
    r = n - r;      //  C(n, r) = C(n, n - r)
  }
  int i, j, s = 1;
  for (i = 0, j = 1; i < r; ++i)
  {
    s *= (n - i);
    for (; j <= r && s % j == 0; ++j)
    {
      s /= j;
    }
  }
  return s;
}
```

5.8 阶乘长度

```
#define PI 3.1415926

int main()
{
  int n, a;
  while (~scanf( “%d”, &n))
  {
    a = (int)((0.5 * log(2 * PI * n) + n * log(n) - n) / log(10));
    printf(”%d\n”, a + 1);
  }
  return 0;
}
```

5.9 全排列

```
#define MAX_N 10
int n;                      //  共n个数
int rcd[MAX_N];             //  记录每个位置填的数
int used[MAX_N];            //  标记数是否用过
int num[MAX_N];             //  存放输入的n个数

void full_permutation(int l)
{
```

```
  int i;
  if (l == n)
  {
    for (i = 0; i < n; i++)
    {
      printf("%d", rcd[i]);
      if (i < n-1)
      {
        printf(" ");
      }
    }
    printf("\n");
    return ;
  }
  for (i = 0; i < n; i++)         //  枚举所有的数(n个),循环从开始
  if (!used[i])
  {                             //  若num[i]没有使用过，则标记为已使用
    used[i] = 1;
    rcd[l] = num[i];          //  在l位置放上该数
    full_permutation(l+1);  //  填下一个位置
    used[i] = 0;            //  清标记
  }
}
```

5.10 求斐波那契第 N 项

```
/*
*  求斐波那契数列第N项，模MOD
*/
#define mod(a, m) ((a) % (m) + (m)) % (m)
const int MOD = 1e9 + 9;
struct MATRIX
{
  long long a[2][2];
};

MATRIX a;
long long f[2];

void ANS_Cf(MATRIX a)
{
  f[0] = mod(a.a[0][0] + a.a[1][0], MOD);
  f[1] = mod(a.a[0][1] + a.a[1][1], MOD);
  return ;
}

MATRIX MATRIX_Cf(MATRIX a, MATRIX b)
{
  MATRIX ans;
  int k;
  for (int i = 0; i < 2; i++)
  {
    for (int j = 0; j < 2; j++)
    {
      ans.a[i][j] = 0;
      k = 0;
      while (k < 2)
      {
        ans.a[i][j] += a.a[k][i] * b.a[j][k];
        ans.a[i][j] = mod(ans.a[i][j], MOD);
        ++k;
      }
    }
  }
  return ans;
}

MATRIX MATRIX_Pow(MATRIX a, long long n)
{
  MATRIX ans;
  ans.a[0][0] = 1;
  ans.a[1][1] = 1;
  ans.a[0][1] = 0;
  ans.a[1][0] = 0;
  while (n)
  {
    if (n & 1)
    {
      ans = MATRIX_Cf(ans, a);
    }
    n = n >> 1;
    a = MATRIX_Cf(a, a);
  }
  return ans;
}

```

```
int main()
{
  long long n;
  while (cin >> n)
  {
    if (n == 1)
    {
      cout << '1' << '\n';
      continue;
    }
    a.a[0][0] = a.a[0][1] = a.a[1][0] = 1;
    a.a[1][1] = 0;
    a = MATRIX_Pow(a, n - 2);
    ANS_Cf(a);
    cout << f[0] << '\n';
  }
  return 0;
}
```

6 STL

6.1 vector

```
vector<int> s;
//  定义一个空的vector对象，存储的是int类型的元素
vector<int> s(n);
//  定义一个含有n个int元素的vector对象
vector<int> s(first, last);
//  定义一个vector对象，并从由迭代器first和last定义的序列[first, last)中复制初值

s[i]                //  直接以下标方式访问容器中的元素
s.front()           //  返回首元素
s.back()            //  返回尾元素
s.push_back(x)      //  向表尾插入元素x
s.size()            //  返回表长
s.empty()           //  表为空时，返回真，否则返回假
s.pop_back()        //  删除表尾元素
s.begin()           //  返回指向首元素的随机存取迭代器
s.end()             //  返回指向尾元素的下一个位置的随机存取迭代器
s.insert(it, val)   //  向迭代器it指向的元素前插入新元素val
s.insert(it, n, val)//  向迭代器it指向的元素前插入n个新元素val
s.insert(it, first, last)
//  将由迭代器first和last所指定的序列[first, last)插入到迭代器it指向的元素前面
s.erase(it)         //  删除由迭代器it所指向的元素
s.erase(first, last)//  删除由迭代器first和last所指定的序列[first, last)
s.reserve(n)        //  预分配缓冲空间，使存储空间至少可容纳n个元素
s.resize(n)         //  改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），元素默认值将填满扩展出的空间
s.resize(n, val)    //  改变序列长度，超出的元素将会全部被删除，如果序列需要扩展（原空间小于n），val将填满扩展出的空间
s.clear()           //  删除容器中的所有元素
s.swap(v)           //  将s与另一个vector对象进行交换
s.assign(first, last) //  将序列替换成由迭代器first和last所指定的序列[first, last)，[first, last)不能是原序列中的一部分

//  要注意的是，resize操作和clear操作都是对表的有效元素进行的操作，但并不一定会改变缓冲空间的大小
//  vector上还定义了序列之间的比较操作运算符（>、<、>=、<=、==、!=），可以按照字典序比较两个序列。
```

6.2 set

```
// 有序
set<int> s;
s.begin()       //  返回指向第一个元素的迭代器
s.clear()       //  清除所有元素
s.count()       //  返回某个值元素的个数
s.empty()       //  如果集合为空，返回true(真)
s.end()         //  返回指向最后一个元素之后的迭代器，不是最后一个元素
s.equal_range() //  返回集合中与给定值相等的上下限的两个迭代器
s.erase()       //  删除集合中的元素
s.find()        //  返回一个指向被查找到元素的迭代器
s.get_allocator()   //  返回集合的分配器
s.insert()      //  在集合中插入元素
s.lower_bound() //  返回指向大于（或等于）某值的第一个元素的迭代器
s.key_comp()    //  返回一个用于元素间值比较的函数
s.max_size()    //  返回集合能容纳的元素的最大限值
s.rbegin()      //  返回指向集合中最后一个元素的反向迭代器
s.rend()        //  返回指向集合中第一个元素的反向迭代器
s.size()        //  集合中元素的数目
s.swap()        //  交换两个集合变量
s.upper_bound() //  返回大于某个值元素的迭代器
```

```
s.value_comp()  //  返回一个用于比较元素间的值的函数

// 多重集合
multiset<int> s;
// 操作类似
```

6.3 pair

```
pair<T1, T2> p1;
pair<T1, T2> p1(v1, v2);
p1.first;
p1.second;
p1 = make_pair(v1, v2);

vector<pair<int, int> > v;
sort(v.begin(), v.end()); //根据pair的first排序，从小到大
```

6.4 stack

```
stack<int> s;
s.push(x);  //  入栈
s.pop();    //  出栈
s.top();    //  访问栈顶
s.empty();  //  当栈空时，返回true
s.size();   //  访问栈中元素个数
```

6.5 queue

```
queue<int> q;
q.push(x);  //  入队列
q.pop();    //  出队列
q.front();  //  访问队首元素
q.back();   //  访问队尾元素
q.empty();  //  判断队列是否为空
q.size();   //  访问队列中的元素个数

//priority_queue（优先队列）。优先队列与队列的差别在于优先队列不是按照入队的顺序出队，而是按照队列中元素的优先权出队列（默认为大者优先，也可以通过指定算子来指定自己的优先顺序）。

priority_queue模版类有三个模版参数，第一个是元素类型，第二个是容器类型，第三个是比较算子。其中后两者都可以忽略，默认容器为vector，默认算子为less，即小的往前排，大的往后排（出队列时列尾元素先出队）。

priority_queue<int> q;
priority_queue<pair<int, int> > qq; //  注意在两个尖括号之间一定要留空格，防止误判
priority_queue<int, vector<int>, greater<int> > qqq;//  定义小的先出队列

q.empty()     //  如果队列为空，则返回true，否则返回false
q.size()      //  返回队列中元素的个数
q.pop()       //  删除队首元素，但不返回其值
q.top()       //  返回具有最高优先级的元素值，但不删除该元素
q.push(item)  //  在基于优先级的适当位置插入新元素

//deque双端队列
#include<deque>
deque<int> dep;

deq.push_front(const T& x); //头插
deq.push_back(const T& x);  //尾插
deq.insert(iterator it, const T& x);  //任意位置

deq.pop_front();  //删除头部
deq.pop_back();   //删除尾部
deq.erase(iterator it);   //删除任意

deq[1]; // 并不会检查是否越界
deq.at(1); // 以上两者的区别就是 at 会检查是否越界，是则抛出 out of range 异常
deq.front();  //访问头部
deq.back();   //访问尾部
```

6.6 map

```
map<T1, T2> mp;
mp[key] = value;
T2 value = mp[key];
mp.size();       //  返回元素个数
mp.empty();      //  判断是否为空
mp.clear();      //  清空所有元素
```

6.7 bitset

```
const int MAXN = 32;

bitset<MAXN> bt;            //  bt 包括 MAXN 位，下标 0 ~ MAXN − 1，默认初始化为 0
bitset<MAXN> bt1(0xf);      //  0xf 表示十六进制数 f，对应二进制 1111，将 bt1 低 4 位初始化为 1
bitset<MAXN> bt2(012);      //  012 表示八进制数 12，对应二进制 1010，即将 bt2 低 4 位初始化为 1010
bitset<MAXN> bt3("1010");   //  将 bt3 低 4 位初始化为 1010

bt.any()        //  bt 中是否存在置为 1 的二进制位?
bt.none()       //  bt 中不存在置为 1 的二进制位吗?
bt.count()      //  bt 中置为 1 的二进制位的个数
bt.size()       //  bt 中二进制位的个数
bt[pos]         //  访问 bt 中在 pos 处的二进制位
bt.test(pos)    //  bt 中在 pos 处的二进制位是否为 1
bt.set()        //  把 bt 中所有二进制位都置为 1
bt.set(pos)     //  把 bt 中在 pos 处的二进制位置为 1
bt.reset()      //  把 bt 中所有二进制位都置为 0
bt.reset(pos)   //  把 bt 中在pos处的二进制位置为0
bt.flip()       //  把 bt 中所有二进制位逐位取反
bt.flip(pos)    //  把 bt 中在 pos 处的二进制位取反
bt[pos].flip()  //  同上
bt.to_ulong()   //  用 bt 中同样的二进制位返回一个 unsigned long 值
```

6.8 algorithm

```
reverse(begin, end) // 反转

unique(begin, end)  // 需排序，去除重复的相邻元素，常用于求不同元素个数
int n=unique(a, a+10)-a;

lower_bound(begin, end, value)  //返回指向第一个不小于给定值的元素的迭代器

upper_bound(begin, end, value)  //返回指向第一个大于给定值的元素的迭代器

next_permutation(array) //一种排列的下一种排列
```

7 计算几何

7.1 三角形面积

```
\\ 海伦公式
int p = (a+b+c)/2;
int s = sqrt(p * (p − a) * (p − b) * (p − c));

\\ 两边和夹角
\\ a, b为边, x为a和b的夹角
s = 0.5 * a * b * sin(x/90.0*acos(0));
```

7.2 两圆面积交

```
const double PI = acos(−1);

struct circle
{
  double x, y, r;
};

// 计算圆心距
double dist(circle a, circle b)
{
  return sqrt((a.x−b.x)*(a.x−b.x) + (a.y−b.y)*(a.y−b.y));
}

double area(circle a, circle b)
{
  if((dist(a, b) + min(a.r, b.r)) <= max(a.r, b.r)) // 内含或重合
  {
    if(a.r < b.r)
      return PI*a.r*a.r;
    else
      return PI*b.r*b.r;
  }
  else if(dist(a, b) >= (a.r + b.r)) // 相切
  {
    return 0.0;
  }
  else
  {
```

```
    double length = dist(a, b);
    // 利用三角形余弦定理求圆心角
    double d1 = 2*acos((a.r*a.r+length*length-b.r*b.r)/(2*a.r*length));
    double d2 = 2*acos((b.r*b.r+length*length-a.r*a.r)/(2*b.r*length));
    // 利用圆心角求得扇形面积再减去三角形面积后两部分相加就是相交面积
    double area1 = a.r*a.r*d1/2 - a.r*a.r*sin(d1)/2;
    double area2 = b.r*b.r*d2/2 - b.r*b.r*sin(d2)/2;
    return area1 + area2;
  }
}
```

8 其它

8.1 数据类型范围

数据类型	范围
char	-128 - 127
int	2147483648 - 2147483647(十位)
long long	-9223372036854775808 - 9223372036854775807(大约十九位)
double	1.7 * 10308

8.2 头文件

```
//#include<stdio.h>
//#include<iostream>
//#include<queue>
//#include<algorithm>
//#include<cstring>
//#include<vector>
//#include<cmath>
//#include<string>
//#include<map>
//#include<set>
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define inf 0x3f3f3f3f
typedef pair<int, int> P;
const int maxn = 5e4+5;
const ll mod = 1e9+7;
```

8.3 Vim 配置

```
// open ~/.vimrc
syntax on
set nu ts=4 sw=4 mouse=a cin
colo desert

map <C-A> ggVG"+y
map <F5> :call CR()<CR>
func! CR()
exec "w"
exec "!g++ -O2 -g -std=c++11 -Wall % -o %<"
exec "! ./%<"
endfunc
```

8.4 输入挂

8.4.1 关闭同步

```
#define endl '\n'

ios::sync_whit_stdio(0);
cin.tie(0);
```

8.4.2 IO

```
#include<cstdio>

inline void read(int &x)  //看情况可去掉负数部分
{
  int t = 1;
  char ch = getchar();
  while(ch < '0' || ch > '9') { if(ch == '-') t = -1; ch = getchar();}
  x = 0;
  while(ch >= '0' && ch <= '9'){ x=x*10+ch-'0'; ch = getchar(); }
  x*=t;
}

void print(int i){
  if(i<10){
    putchar('0'+i);
```

```
16      return ;
17    }
18    print(i/10);
19    putchar('0'+i%10);
20  }
```

8.5 C++ 大数

8.5.1 大数加法

```
1  string add(string a,string b)
2  { //两数相加
3    string res="";
4    int i=1;
5    string first="0";
6    while(true) {
7      int tai=a.size()-i;
8      int tbi=b.size()-i;
9      if(tai<0 && tbi<0)
10       break;       //从两数最右边开始模拟加法运算直到两数都遍历完
11     int ta,tb;
12     if(tai<0)
13       ta=0; //如果没数则至为0
14     else
15       ta=a[tai]-'0';
16     if(tbi<0)
17       tb=0; //如果没数则至为0
18     else
19       tb=b[tbi]-'0';
20     int temp=ta+tb+first[0]-'0';  //相加 first保存上一个的进位信息
21     first[0]=temp%10+'0'; //当前位是对10取余
22     res=first+res;
23     first[0]=temp/10+'0'; //进位是除10
24     i++;
25   }
26   if(first!="0")
27     res=first+res;  //如果进位还有则添加
28   if(res[0]=='0' && res.size()>1) //去除前导0
29     res.erase(res.begin());
30   return res;
31 }
```

8.5.2 大数乘法

```
1  // 还需要配合大数加法
2  string mul(string a,string b) {
3    if(b.size()==1) { //如果b只有一位则使其分别相乘
4      string res="";
5      string first="0";
6      int mb=b[0]-'0';
7      for(int i=a.size()-1; i>=0; i--) {  //从最后开始依次计算
8        int temp=(a[i]-'0') * mb + (first[0]-'0');
9        first[0]=temp%10+'0'; //当前位
10       res=first+res;
11       first[0]=temp/10+'0'; //进位
12     }
13     if(first!="0") {
14       res=first+res;  //处理进位
15     }
16     if(res[0]=='0' && res.size()>1)
17       res.erase(res.begin()); //除去前导0
18     return res;
19     }
20   //否则则把b拆分为一位
21   string res="0";
22   string zero="";
23   for(int i=b.size()-1; i>=0; i--) {  //从b的最后一位开始
24     string temp=mul(a,b.substr(i,1)); //计算当前为与a相乘
25     res=add(res,temp+zero); //在其后添加适当的0再与结果相加
26     zero=zero+"0";
27   }
28   return res;
29 }
```

8.5.3 整数转 string

```
1  string inttostring(int m) { //将整数转为string
2    string res="";
3    string temp="0";
4    while(m) {
5      temp[0]=m%10+'0';
6      res=temp+res;
7      m/=10;
8    }
```

```
  if(res=="")
    res="0";
  return res;
}
```

8.6 Java

```
valueOf(parament); 将参数转换为制定的类型

比如 int a=3;

BigInteger b=BigInteger.valueOf(a);

则b=3;

String s=”12345”;

BigInteger c=BigInteger.valueOf(s);

则c=12345;

// 常用函数
1.赋值:
BigInteger a=new BigInteger("1");
BigInteger b=BigInteger.valueOf(1);

2.运算:
  add(); 大整数相加
BigInteger a=new BigInteger(“23”);
BigInteger b=new BigInteger(“34”);
a.add(b);

 subtract(); 相减
 multiply(); 相乘
 divide(); 相除取整
 remainder(); 取余
 pow(); a.pow(b)=a^b
 gcd(); 最大公约数
 abs(); 绝对值
 negate(); 取反数
 mod(); a.mod(b)=a%b=a.remainder(b);

3.BigInteger构造函数:
一般用到以下两种:
BigInteger(String val);
将指定字符串转换为十进制表示形式;
BigInteger(String val,int radix);
将指定基数的 BigInteger 的字符串表示形式转换为 BigInteger

4.基本常量:
A=BigInteger.ONE 1
B=BigInteger.TEN 10
C=BigInteger.ZERO 0

5.n.compareTo(BigInteger.ZERO)==0  //相当于n==0

6.if(a[i].compareTo(n)>=0 && a[i].compareTo(m)<=0)   // a[i]>=n && a[i]<=m

// 模板
import java.math.BigInteger;
import java.math.BigDecimal;
import java.util.Scanner;
import java.util.*;
import java.io.*;
public class Main {
  public static void main(String [] args){
    Scanner cin = new Scanner(System.in);
    BigInteger a, b;
    while(cin.hasNext())//相当于c语言中的scanf("%d", &n) != EOF
    {
      a = cin.nextBigInteger();
      b = cin.nextBigInteger();
      System.out.println(a.add(b));//大整数加法
      System.out.println(a.subtract(b));//大整数减法
      System.out.println(a.multiply(b));//大整数乘法
      System.out.println(a.divide(b));//大整数除法，取整
      System.out.println(a.remainder(b));//大整数取模
      System.out.println(a.abs());//对大整数a取绝对值
      int x = 0;
      System.out.println(a.pow(x));//大整数a的x次幂
      int y = 8;
      System.out.println(a.toString(y));//返回大整数a的p进制用字符串表现的形
        式
      System.out.println(a.toString());//返回大整数a的十进制用字符串表现的形
        式
      //大整数之间的比较
```

```
78      if( a.compareTo(b) == 0 ) System.out.println("a == b"); //大整数a==b
79      else if( a.compareTo(b) > 0 ) System.out.println("a > b"); //大整数a>b
80      else if( a.compareTo(b) < 0 ) System.out.println("a < b"); //大整数a<b
          )
81
82      BigDecimal c, d;
83      c = cin.nextBigDecimal();
84      d = cin.nextBigDecimal();
85      System.out.println(c.add(d));//浮点数相加
86      System.out.println(c.subtract(d));//浮点数相减
87      System.out.println(c.multiply(d));//浮点数相乘
88    }
89  }
90 }
91
92 // 输入方式
93 int a = cin.nextInt();
94 String s = cin.nextLine();
95 double b = cin.nextDouble();
```