2015-10-22

ACM Template

[文档副标题]

zhyfzy

UESTC

目录

基础

头文件

```
#include<cstdio>
#include<cstring>
#include<cmath>
#include<iostream>
#include<algorithm>
#include<vector>
#include<deque>
#include<queue>
#include<set>
#include<map>
#include<stack>
#include<string>
#include<sstream>
#include<ctime>
#define eps 1e-9
#define ALL(x) x.begin(),x.end()
```

```
#define INS(x) inserter(x,x.begin())
#define rep(i,j,k) for(int i=j;i<=k;i++)
#define MAXN 1005
#define MAXM 40005
#define INF 0x3fffffff
#define PB push_back
#define MP make_pair
#define X first
#define Y second
#define lc (k<<1)
#define rc ((k<<1)|1)
#define mii ((l+r)>>1)
#define fuck puts("fuck!")
#define clr(x,y) memset(x,y,sizeof(x))
using namespace std;
typedef long long LL;
int i,j,k,n,m,x,y,T,ans,mx,mi,cas,num,len;
bool flag;

int main()
{
    scanf("%d",&T);
    return 0;
}
```

扩栈代码

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
```

手写快排

```
void qsort(int l,int r)
{
    int i,j,t,mid;
    i=l;j=r;mid=a[(l+r)/2];
    do
    {
        while(a[i]<mid) i++;
        while(a[j]>mid) j--;
        if (i<=j)
        {
```

```
            t=a[i];a[i]=a[j];a[j]=t;
            i++;j--;
        }
    } while (i<=j);
    if (l<j) qsort(l,j);
    if (i<r) qsort(i,r);
}
```

输入挂

```
inline int in() {
    char ch=getchar();
    while(ch<'0'||ch>'9') ch=getchar();
    int res=0;
    while(ch>='0' && ch<='9') {
        res=res*10+ch-'0';
        ch=getchar();
    }
    return res;
}
```

随机数

建立随机化种子：(一般放在 main 的开头)

```
#include <ctime>
srand(time(0));
```

产生 0 到 32767 的随机数，用取模来获得任意更小区间的值

```
rand();
```

位运算

运算 k&-k
取出低位起，第一个 1 的位置
如 k=18=10010(二进制)，k&-k=2=10(二进制)
k=144=10010000(二进制)，k&-k=16=10000(二进制)
注意 k=0 时 0&-0=0，如果有循环的话会进入死循环 @树状数组的错误方式
==========
运算 k=k-(k&-k)

注意要带括号否则编译器会先运算 k-k
将 k 的二进制，削去低位起的第一个 1
如 k=1168=10010010000(二进制)，k-(k&-k)=10010000000(二进制)
==========
异或的性质
0^x=x
x^x=0
a^b=b^a
==========
枚举子集的方法
for (j=x;j;j=(j-1)&x)
其中 x 为原集合
==========
取反操作
(~i)&((1<<n)-1)
其中 n 为 i 的二进制位数，注意由于优先级的关系 1<<n 必须加括号
==========
取第 k 位
x&(1<<k)
==========
将第 i 位赋值为 1
s|1<<i
==========
将第 i 位赋值为 0
s&~(1<<i)

前向星

```
int edge,head[MAXN];
struct edgenode
{
    int to,next,w;
} G[MAXM];

void add_edge(int x,int y,int w)
{
    G[edge].to=y;
    G[edge].w=w;
    G[edge].next=head[x];
    head[x]=edge++;
}

int main()
```

```
{
    memset(head,-1,sizeof(head));
    edge=0;
}
```

图论

最短路

```
void dijkstra(int s,int t)
{
    for (i=0;i<=n;i++) dis[i]=INF;
    dis[s]=0;
    priority_queue<pir,vector<pir>,greater<pir> > q;
    q.push(MP(dis[s],s));
    while (!q.empty())
    {
        int d=q.top().X;
        int u=q.top().Y;
        q.pop();
        for (i=head[u];i!=-1;i=G[i].next)
        {
            int v=G[i].to;
            if (dis[v]>d+G[i].cap)
            {
                dis[v]=d+G[i].cap;
                q.push(MP(dis[v],v));
            }
        }
    }
}
```

```
bool SPFA(int s)
{
    int u,d,v;
    for (i=0;i<=n;i++) dis[i]=INF;□
    dis[s]=0;

    while (!q.empty()) q.pop();
```

```
    q.push(s);

    while (!q.empty())
    {
        int u=q.front();
        q.pop();
        vis[u]=0;

        for (int i=head[u];i!=-1;i=G[i].next)
        {
            int v=G[i].to;
            if (dis[v]>dis[u]+G[i].w)
            {
                dis[v]=dis[u]+G[i].w;
                if (!vis[v])
                {
                    q.push(v);
                    vis[v]=1;
                    if (cnt[v]>n) return false;
                }
            }
        }
    }
    return true;
}
```

最小生成树（**Prim**）

```
bool vis[N];
int dis[N];
int prim()
{
    int i,j,u,v,w,ans=0;
    for(i=1; i<=n; i++) dis[i]=e[1][i];
    memset(vis,0,sizeof(vis));
    vis[1]=1;
    for(i=2; i<=n; i++)
    {
        w=inf;
        u=-1;
        for(j=1; j<=n; j++) if(!vis[j]&&w>dis[j]) w=dis[j],u=j;
        if(u==-1) return -1;
        vis[u]=1;
```

```
        ans+=w;
        for(v=1; v<=n; v++)
            if(!vis[v])
                if(dis[v]>e[u][v]) dis[v]=e[u][v];
    }
    return ans;
}
```

最小树形图

有根的情况：直接求.
无根的情况：建立超级根节点,向每个点连有向边,边权大于所有边权和即可.

```
int edge,head[MAXN];
struct edgenode
{
    int from,to,next;
    double w;
} G[MAXM];

void add_edge(int x,int y,double w)
{
    G[edge].from=x;
    G[edge].to=y;
    G[edge].w=w;
    G[edge].next=head[x];
    head[x]=edge++;
}

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

int pre[MAXN],id[MAXN],vis[MAXN];
double in[MAXN];

//首先需要枚举根，这个模板并不包含根，然后 NV 是点数，NE 是边数，直接用 edge 即可
double Direct_MST(int root,int NV,int NE)
{
    double ret=0;
```

```
while (1)
{
    for (int i=0;i<NV;i++) in[i]=INF;
    memset(pre,-1,sizeof(pre));
    memset(vis,-1,sizeof(vis));
    memset(id,-1,sizeof(id));
    for (int i=0;i<NE;i++)
    {
        if (G[i].from==G[i].to) continue;
        int v=G[i].to;
        if (dcmp(G[i].w-in[v])<0)
        {
            pre[v]=G[i].from;
            in[v]=G[i].w;
        }
    }
    for (int i=0;i<NV;i++)
    {
        if (dcmp(in[i]-INF)==0 && i!=root)
        {
            return -1;
        }
    }
    int cnt=0;
    in[root]=0;
    for (int i=0;i<NV;i++)
    {
        ret+=in[i];
        int v=i;
        while (vis[v]!=i && id[v]==-1 && v!=root)
        {
            vis[v]=i;
            v=pre[v];
        }
        if (v!=root && id[v]==-1)
        {
            for (int u=pre[v];u!=v;u=pre[u])
            {
                id[u]=cnt;
            }
            id[v]=cnt++;
        }
    }
    if (cnt==0) break;
```

```
        for (int i=0;i<NV;i++)
        {
            if (id[i]==-1) id[i]=cnt++;
        }
        for (int i=0;i<NE;i++)
        {
            int v=G[i].to;
            G[i].from=id[G[i].from];
            G[i].to=id[G[i].to];
            G[i].w-=in[v];
        }
        NV=cnt;
        root=id[root];
    }
    return ret;
}
```

网络流

ISAP

```
struct edge{
    int from,to,w;
    int next;
}G[MAXM];
int head[MAXN];
int Edge;

void add_edge(int u, int v, int w)
{
    G[Edge].from = u;
    G[Edge].to = v;
    G[Edge].w = w;
    G[Edge].next = head[u];
    head[u] = Edge++;
    G[Edge].from = v;
    G[Edge].to = u;
    G[Edge].w = 0;
    G[Edge].next = head[v];
    head[v] = Edge++;
}
```

```
int curedge[MAXN],parent[MAXN],level[MAXN];
int cnts[MAXN],augment[MAXN];
int isap(int n,int s, int e)
{
    int i,u,v,max_flow,aug,min_lev;

    memset(level,0,sizeof(level));
    memset(cnts,0,sizeof(cnts));
    for (i=0;i<=n;i++) curedge[i] = head[i];
    max_flow=0;
    augment[s]=INF;
    parent[s]=-1;
    u=s;
    while (level[s]<n)
    {
        if (u==e)
        {
            max_flow+=augment[e];
            aug=augment[e];
            for (v=parent[e];v!=-1;v=parent[v])
            {
                i=curedge[v];
                G[i].w-=aug;
                G[i^1].w+=aug;
                augment[G[i].to]-=aug;
                if (G[i].w==0) u=v;
            }
        }
        for (i=curedge[u];i!=-1;i=G[i].next)
        {
            v=G[i].to;
            if (G[i].w>0 && level[u]==(level[v]+1))
            {
                augment[v]=min(augment[u],G[i].w);
                curedge[u]=i;
                parent[v]=u;
                u=v;
                break;
            }
        }
        if (i==-1)
        {
            if (--cnts[level[u]]==0)
            {
```

```
            break;
        }
        curedge[u]=head[u];
        min_lev=n;
        for (i=head[u];i!=-1;i=G[i].next)
        {
            if (G[i].w>0)
            {
                min_lev=min(level[G[i].to],min_lev);
            }
        }
        level[u]=min_lev+1;
        cnts[level[u]]++;
        if (u!=s) u=parent[u];
    }
  }
  return max_flow;
}
```

Dinic 非递归

```
const int inf = 0x3f3f3f3f;
struct edgenode
{
    int from,to,next;
    int cap;
}edge[MAXM];
int Edge,head[MAXN],ps[MAXN],dep[MAXN];

void addedge(int x,int y,int c)
{
    edge[Edge].from=x;
    edge[Edge].to=y;
    edge[Edge].cap=c;
    edge[Edge].next=head[x];
    head[x]=Edge++;

    edge[Edge].from=y;
    edge[Edge].to=x;
    edge[Edge].cap=0;
    edge[Edge].next=head[y];
    head[y]=Edge++;
}
```

```
int dinic(int n,int s,int t)
{
    int tr,flow=0;
    int i,j,k,l,r,top;
    while(1){
        memset(dep,-1,(n+1)*sizeof(int));
        for(l=dep[ps[0]=s]=0,r=1;l!=r;)//BFS 部分，将给定图分层
        {
            for(i=ps[l++],j=head[i];j!=-1;j=edge[j].next)
            {
                if (edge[j].cap&&-1==dep[k=edge[j].to])
                {
                    dep[k]=dep[i]+1;ps[r++]=k;
                    if(k==t)
                    {
                        l=r;
                        break;
                    }
                }
            }
        }
        if(dep[t]==-1)break;

        for(i=s,top=0;;)//DFS 部分
        {
            if(i==t)//当前点就是汇点时
            {
                for(k=0,tr=inf;k<top;++k)
                    if(edge[ps[k]].cap<tr)tr=edge[ps[l=k]].cap;

                for(k=0;k<top;++k)
                    edge[ps[k]].cap-=tr,edge[ps[k]^1].cap+=tr;

                flow+=tr;
                i=edge[ps[top=l]].from;
            }

            for(j=head[i];j!=-1;j=edge[j].next)//找当前点所指向的点
                if(edge[j].cap&&dep[i]+1==dep[edge[j].to]) break;

            if(j!=-1)
            {
                ps[top++]=j;//当前点有所指向的点，把这个点加入栈中
```

```
                i=edge[j].to;
            }
            else
            {
                if (!top) break;//当前点没有指向的点，回溯
                dep[i]=-1;
                i=edge[ps[--top]].from;
            }
        }
    }
    return flow;
}
```

最小费用最大流

```
/*最小费用流模板 -BEGIN-*/
/*size 表示网络中的结点数，编号从 0 开始，如果是从 1 开始则 size=n+1*/
/*初始化：head 全设为-1，Edge 为边数预先设为 0*/
int head[MAXN],vis[MAXN],dis[MAXN],pos[MAXN],Edge,size;
struct edgenode
{
    int to,next,w,cost;
} edge[MAXM];

void add_edge(int x,int y,int w,int cost)
{
    edge[Edge].to=y;
    edge[Edge].w=w;
    edge[Edge].cost=cost;
    edge[Edge].next=head[x];
    head[x]=Edge;
    Edge++;


    edge[Edge].to=x;
    edge[Edge].w=0;
    edge[Edge].cost=-cost;
    edge[Edge].next=head[y];
    head[y]=Edge;
    Edge++;
}

bool SPFA(int s, int t)
{
```

```
    int u,v,i;
    queue <int> q;
    memset(vis,0,sizeof(vis));
    for(i=0;i<size;i++) dis[i]=INF;
     dis[s]=0;
     vis[s]=1;
    q.push(s);
    while(!q.empty())
    {
        u=q.front(); q.pop(); vis[u]=0;
        for (i=head[u];i!=-1;i=edge[i].next)
          {
              v=edge[i].to;
              if(edge[i].w>0&&dis[u]+edge[i].cost<dis[v])
              {
               dis[v]=dis[u]+edge[i].cost;
               pos[v]=i;
               if(!vis[v])
               {
                   vis[v]=1;
                   q.push(v);
               }
              }
          }
    }
    return dis[t]!=INF;
}
int MinCostFlow(int s,int t)
{
    int i,cost=0,flow=0;
    while(SPFA(s,t))
    {
        int d=INF;
        for (i=t;i!=s;i=edge[pos[i]^1].to)
        {
            d=min(d,edge[pos[i]].w);
        }
        for(i=t;i!=s;i=edge[pos[i]^1].to)
        {
            edge[pos[i]].w-=d;
            edge[pos[i]^1].w+=d;
        }
        flow+=d;
        cost+=dis[t]*d;
```

```
    }
    return cost; // flow 是最大流值
}
/*最小费用流模板 -END-*/
```

无向图连通分量

```
void tarjan(int u,int fa,int path)//u 当前节点, fa 父亲节点, path 边 u->fa
在前向星中的编号
{
    vis[u]=1;
    dfn[u]=low[u]=++time;
    for (int i=head[u];i!=-1;i=G[i].next)
    {
        if (i==path) continue;

        int v=G[i].to;
        if (!vis[v])
        {
            tarjan(v,u,i^1);
            low[u]=min(low[u],low[v]);
            if (low[v]>dfn[u])
            {
                G[i].flag=1;
                G[i^1].flag=1;
            }
        }else low[u]=min(low[u],dfn[v]);
    }
    vis[u]=2;
}
```

例子 (CF231E)

```
int
edge,head[MAXN],bin[MAXN],headS[MAXN],edgeS,vis[MAXN],d[MAXN],id[
MAXN],num[MAXN];
struct edgenode
{
    int to,next,flag;
} G[MAXM],S[MAXM];

void add_edge(int x,int y)
{
```

```
    G[edge].to=y;
    G[edge].flag=0;
    G[edge].next=head[x];
    head[x]=edge++;
}

void add_edgeS(int x,int y)
{
    S[edgeS].to=y;
    S[edgeS].flag=0;
    S[edgeS].next=headS[x];
    headS[x]=edgeS++;
}

int fa[MAXN];
int findset(int x)
{
    return x==fa[x]?x:fa[x]=findset(fa[x]);
}
void unionset(int x,int y)
{
    fa[findset(x)]=findset(y);
}

int dfn[MAXN],low[MAXN],time;
void dfs(int u,int fa)
{
    vis[u]=1;
    dfn[u]=low[u]=++time;
    for (int i=head[u];i!=-1;i=G[i].next)
    {
        int v=G[i].to;
        if (v!=fa && vis[v]==1)
        {
            low[u]=min(low[u],dfn[v]);
        }

        if (!vis[v])
        {
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if (low[v]>dfn[u]) G[i].flag=1;
        }
    }
```

```
    vis[u]=2;
}

void dive(int u,int scc)
{
    id[u]=scc;
    vis[u]=1;
    num[scc]++;
    for (int i=head[u];i!=-1;i=G[i].next)
    {
        if (!G[i].flag && !vis[G[i].to])
            dive(G[i].to,scc);
    }
}

void dis(int u,int dep)//记录当前节点到根节点有多少个"环"
{
    vis[u]=1;
    d[u]=dep;
    for (int i=headS[u];i!=-1;i=S[i].next)
    {
        int v=S[i].to;
        if (!vis[v])
        {
            if (num[v]>2) dis(v,dep+1);
            else dis(v,dep);
        }
    }
}

vector<pair<int,int> > Q[MAXN];
int ans[MAXN];
void tarjan(int u)
{
    vis[u]=true;
    for (int i=0;i<Q[u].size();i++)
    {
        int v=Q[u][i].X,id=Q[u][i].Y;
        if (vis[v])
        {
            int com=findset(v);
            ans[id]=d[u]+d[v]-2*d[com];
            if (num[com]>2) ans[id]++;
        }
```

```
    }

    for (int i=headS[u];i!=-1;i=S[i].next)
    {
        int v=S[i].to;
        if (!vis[v])
        {
            tarjan(v);
            unionset(v,u);
        }
    }
}


int main()
{
    memset(head,-1,sizeof(head));
    edge=0;
    memset(headS,-1,sizeof(headS));
    edgeS=0;
    scanf("%d%d",&n,&m);
    while (m--)
    {
        scanf("%d%d",&x,&y);
        add_edge(x,y);
        add_edge(y,x);
    }

    dfs(1,-1);

    memset(vis,0,sizeof(vis));
    int scc=1;
    for (int i=1;i<=n;i++)
    {
        if (!vis[i]) dive(i,scc++);
    }

    for (i=1;i<=n;i++)
    {
        for (int j=head[i];j!=-1;j=G[j].next)
        {
            int v=G[j].to;
            if (id[i]!=id[v])
            {
```

```
                add_edgeS(id[i],id[v]);
                add_edgeS(id[v],id[i]);
            }
        }
    }

    int q;
    scanf("%d",&q);
    for (i=0;i<q;i++)
    {
        scanf("%d%d",&x,&y);
        x=id[x];y=id[y];
        Q[x].PB(MP(y,i));
        Q[y].PB(MP(x,i));
    }

    memset(vis,0,sizeof(vis));
    if (num[1]>2) dis(1,1); else dis(1,0);

    memset(vis,0,sizeof(vis));
    for (i=1;i<=n;i++) fa[i]=i;
    tarjan(1);

    bin[0]=1;
    for(int i=1;i<MAXN;i++)
        bin[i]=bin[i-1]*2%1000000007;
    for (int i=0;i<q;i++)
    {
        printf("%d\n",bin[ans[i]]);
    }
    return 0;
}
```

有向图连通分量

```
int
head[MAXN],ru[MAXN],chu[MAXN],index,dfn[MAXN],low[MAXN],cnt,scc[M
AXN],ans,r,c,Edge;
stack <int> tar;
struct edgenode
{
    int from,to,next;
} G[MAXM];
```

```
void add_edge(int x,int y)
{
    G[Edge].from=x;
    G[Edge].to=y;
    G[Edge].next=head[x];
    head[x]=Edge++;
}

void add_new_edge(int x,int y)
{
    chu[x]++;
    ru[y]++;
}

void tarjan(int u)
{
    int x;
    dfn[u]=low[u]=++index;
    tar.push(u);
    for (int i=head[u];i!=-1;i=G[i].next)
    {
        int v=G[i].to;
        if (!dfn[v])
        {
            tarjan(v);
            low[u]=min(low[u],low[v]);
        } else
        if (!scc[v])
        {
            low[u]=min(low[u],dfn[v]);
        }
    }
    if (low[u]==dfn[u])
    {
        cnt++;
        do
        {
            x=tar.top();
            tar.pop();
            scc[x]=cnt;
        } while (x!=u);
    }
}
```

```
void build_new_map()
{
    memset(ru,0,sizeof(ru));
    memset(chu,0,sizeof(chu));
    for (int i=1;i<=m;i++)
    {
        if (scc[G[i].to]==scc[G[i].from]) continue;
        add_new_edge(scc[G[i].from],scc[G[i].to]);
    }
}

void build_map()
{
    memset(dfn,0,sizeof(dfn));
    memset(low,0,sizeof(low));
    memset(scc,0,sizeof(scc));
    index=cnt=0;
    for (i=1;i<=n;i++)
        if (!dfn[i]) tarjan(i);
    build_new_map();
}
```

2-SAT

【解释】
给多个语句，每个语句为“ Xi 为真(假) 或者 Xj 为真(假)”
每个变量和拆成两个点 2*i 为假， 2*i+1 为真
“Xi 为真 或 Xj 为真” 等价于 “Xi 为假 -> Xj 为真”。
DFS 算法没有回溯过程。

【函数说明】
模板 bfs 函数在模板外一般用不到
void init(int n) :初始化
void add(int x,int xval,int y,int yval) :添加边，x,y 为节点编号，xval=1 表示真，xval=0 表示假，yval 同理
bool solve() :计算是否存在解。如果存在解返回 true，不存在返回 false

【变量说明】
vector<int>G[MAXN*2];//邻接表表示图。
bool mark[MAXN*2];//表示某个结点（不是变量）是否已经被访问
int s[MAXN*2],c;//s 存储某次 DFS 访问过那些点。用于重新标记时消去之前访问过的点的记录（mark[]值）

```
struct Twosat
{
    int n;
    vector<int>G[MAXN*2];//邻接表表示图。
    bool mark[MAXN*2];//表示某个点是否已经被访问
    int s[MAXN*2],c;//s 存储答案。

    bool dfs(int x)
    {
        if(mark[x^1])return  0;//如果 x^1 被标记过，说明 x 是不成立（x^1 成立），返回 0
        if(mark[x])return 1;//如果 x 被标记过，说明 x 是成立的，返回 1
        mark[x]=1;
        s[c++]=x;
        for(int i=0;i<G[x].size();i++)
        {
            if(!dfs(G[x][i]))return 0;
        }
        return 1;//与之相连的变量都满足。
    }

    void init(int n)//初始化，n 为变量个数（结点数 2*n，从 0 开始）
    {
        this->n=n;
        for(int i=0;i<n*2;i++)G[i].clear();
        memset(mark,0,sizeof(mark));
    }

    void add(int x,int xval,int y,int yval)//添加边，xval=1 表示真，xval=0 表示假，yval 同理
    {
        x=x*2+xval;
        y=y*2+yval;
        G[x^1].push_back(y);
        G[y^1].push_back(x);
    }

    bool solve()//计算。
    {
        for(int i=0;i<n*2;i+=2)
        {
            if(!mark[i]&&!mark[i+1])
            {
```

```
                c=0;
                if(!dfs(i))//如果"Xi 为假"这个假定不成立
                {
                    while(c>0)mark[s[--c]]=0;
                    if(!dfs(i+1))return 0;//改成"Xi 为真"，重新标记。
                }
            }
        }
        return 1;
    }
}sat;
```

欧拉路径

有向图的欧拉路径（无向图通过度数判断）
void dfs(int u,int fa)//c 表示当前 DFS 的位置，pa 表示这个字母是从第 pa 个边来的，最后循环完加入 ans 数组即可，显然 ans 是倒序加入的。

```
{
    for (int i=head[u];i!=-1;i=G[i].next)
    {
        int v=G[i].to;
        if (!b[v])
        {
            b[v]=1;
            dfs(v,i);
        }
    }
    if (pa>0) ans.PB(fa);
}
```

匈牙利

```
int linked[1005];
bool used[1005],flag[1005],map[1005][1005];

bool find(int u)
{
    for (int i=0;i<=m;i++)
    {
        if (map[u][i]&&!used[i])
        {
```

```
            used[i]=1;
            if (!linked[i]||find(linked[i]))
            {
                linked[i]=u;
                return 1;
            }
        }
    }
    return 0;
}

int max_match()
{
    int ans=0;
    memset(linked,0,sizeof(linked));
    for (int i=1;i<=n;i++)
    {
        memset(used,0,sizeof(used));
        if (find(i)) ans++;
    }
    return ans;
}

int main()
{
    while (scanf("%d%d%d",&k,&n,&m),k)
    {
        memset(map,0,sizeof(map));
        for (i=1;i<=k;i++)
        {
            scanf("%d%d",&x,&y);
            map[x][y]=1;
        }
        printf("%d\n",max_match());
    }
    return 0;
}
```

数论

扩展欧几里德定理

```
//拓展欧几里得定理，求 ax+by=gcd(a,b)的一组解(x,y),d=gcd(a,b)
void gcd(int a,int b,int &d,int &x,int &y)
{
    if(!b){d=a;x=1;y=0;}
    else{gcd(b,a%b,d,y,x);y-=x*(a/b);}
}
```

用法 1：
求 ax+by=c 的整数解。ax+by=gcd(a,b)=g 的一组解为(x0,y0),则 ax+by=c 的一组解为(x0*c/g,y0*c/g)。当 c 不是 g 的倍数时无整数解
若(x1,y1)是 ax+by=c 的一组解，则其任意整数解为(x1+k*bb,y1-k*aa)，其中 aa=a/gcd(a,b),bb=b/gcd(bb)，k 为任意整数

求模乘法的逆

```
//求得 a 在模 n 条件下的逆
int inv(int a,int n)
{
   int d,x,y;
   gcd(a,n,d,x,y);
   return d==1?(x+n)%n:-1;
}
```

或则：

```
v=pow_mod(a,n-m-1,n);//n 为素数，pow_mod(a,n-1,n)=1,费马小定理。所以 a^m*a^(n-m-1)=a^(n-1)=1(mod n).
```

快速幂求解 a^b

```
int pow_mod(int a,int b)
{
    int s=1;
    while(b)
    {
```

```
        if(b&1)
            s=(s*a)%mod;
        a=(a*a)%mod;
        b=b>>1;
    }
    return s;
}
```

求解模方程 a^x=b(mod n)

1.n 为素数。无解返回-1

```
//求解模方程 a^x=b(mod n)。n 为素数，无解返回-1
int log_mod (int a,int b,int n)
{
    int m,v,e=1,i;
    m=ceil(sqrt(n+0.5));
    v=inv(pow_mod(a,m),n);
    map<int,int>x;
    x[1]=0;
    for(i=1;i<m;i++)
    {
        e=(e*a)%n;
        if(!x.count(e))x[e]=i;
    }
    for(i=0;i<m;i++)
    {
        if(x.count(b))return i*m+x[b];
        b=(b*v)%n;
    }
    return -1;
}
```

2.n 不是素数。

```
//hdu 2815 Mod Tree
#include <cstdio>
#include <cstring>
#include <cmath>
#include <map>
#include <iostream>
#include <algorithm>
using namespace std;
```

```
#define LL __int64
LL gcd(LL a,LL b)
{
    return b==0?a:gcd(b,a%b);
}
//拓展欧几里得定理，求 ax+by=gcd(a,b)的一组解(x,y),d=gcd(a,b)
void gcd_mod(LL a,LL b,LL &d,LL &x,LL &y)
{
    if(!b){d=a;x=1;y=0;}
    else{gcd_mod(b,a%b,d,y,x);y-=x*(a/b);}
}
//求解模方程 d*a^(x-c)=b(mod n)。d,a 和 n 互质，无解返回-1
LL log_mod (LL a,LL b,LL n,LL c,LL d)
{
    LL m,v,e=1,i,x,y,dd;
    m=ceil(sqrt(n+0.5));     //x=i*m+j
    map<LL,LL>f;
    f[1]=m;
    for(i=1;i<m;i++)  //建哈希表，保存 a^0,a^1,...,a^m-1
    {
        e=(e*a)%n;
        if(!f[e])f[e]=i;
    }
    e=(e*a)%n;//e=a^m
    for(i=0;i<m;i++)//每次增加 m 次方，遍历所有 1<=f<=n
    {
        gcd_mod(d,n,dd,x,y);//d*x+n*y=1-->(d*x)%n=1-->d*(x*b)%n==b
        x=(x*b%n+n)%n;
        if(f[x])
        {
            LL num=f[x];
            f.clear();//需要清空，不然会爆内存
            return c+i*m+(num==m?0:num);
        }
        d=(d*e)%n;
    }

    return -1;
}
int main()
{
    LL a,b,n;
    while(scanf("%I64d%I64d%I64d",&a,&n,&b)!=EOF)
    {
```

```
        if(b>=n)
        {
            printf("Orz,I can't find D!\n");
            continue;
        }
        if(b==0)
        {
            printf("0\n");
            continue;
        }
        LL ans=0,c=0,d=1,t;
        while((t=gcd(a,n))!=1)
        {
            if(b%t){ans=-1;break;}
            c++;
            n=n/t;
            b=b/t;
            d=d*a/t%n;
            if(d==b){ans=c;break;}//特判下是否成立。
        }
        if(ans!=0)
        {
            if(ans==-1){printf("Orz,I can't find D!\n");}
            else printf("%I64d\n",ans);
        }
        else
        {
            ans=log_mod(a,b,n,c,d);
            if(ans==-1)printf("Orz,I can't find D!\n");
            else printf("%I64d\n",ans);
        }
    }
    return 0;
}
/*
    求解模方程 a^x=b(mod n)，n 不为素数。拓展 Baby Step Giant Step
    模板题。

    方法：
    初始 d=1,c=0,i=0;
    1.令 g=gcd(a,n),若 g==1 则执行下一步。否则由于 a^x=k*n+b;(k 为某一整数),
则(a/g)*a^k=k*(n/g)+b/g,(b/g 为整除，若不成立则无解)
令 n=n/g，d=d*a/g，b=b/g,c++则 d*a^(x-c)=b(mod n),接着重复 1 步骤。
    2.通过 1 步骤后，保证了 a 和 d 都与 n 互质，方程转换为 d*a^(x-c)=b(mod n)。
```

由于 a 和 n 互质，所以由欧拉定理 a^phi(n)==1(mod n),(a,n 互质)
可知，phi(n)<=n,a^0==1(mod n),所以构成循环，且循环节不大于 n。从而推出如果存在解，则必定 1<=x<n。(在此基础上我们就可以用
Baby Step Giant Step 方法了)

3.令 m=ceil(sqrt(n)),则 m*m>=n。用哈希表存储 a^0,a^1,..,a^(m-1)，接着判断 1~m*m-1 中是否存在解。

4.为了减少时间，所以用哈希表缩减复杂度。分成 m 次遍历，每次遍历 a^m 长度。由于 a 和 d 都与 n 互质，所以 gcd(d,n)=1,
所以用拓展的欧几里德定理求得 d*x+n*y=gcd(d,n)=1,的一组整数解(x,y)。则 d*x+n*y=1-->d*x%n=(d*x+n*y)%n=1-->d*(x*b)%n=((d*x)%n*b%n)%n=b。
所以若 x*b 在哈希表中存在，值为 k，则 a^k*d=b(mod n),答案就是 ans=k+c+i*m。
如果不存在,则令 d=d*a^m,i++后遍历下一个 a^m,直到遍历 a^0 到 a^(m-1)还未找到，则说明不解并退出。

*/

中国剩余定理

互质情况

```
//中国剩余定理，求得 M%A=a,M%B=b,...中的 M，其中 A,B,C...互质
int china(int a[])
{
    int i,j,k,d,ans=0,x,y,M=1;
    for(i=0;i<n;i++)
        M=M*x[i];
    for(i=0;i<n;i++)
    {
        m=M/x[i];
        gcd(x[i],m,d,x,y);
        ans=(ans+y*m*a[i])%M;
    }
    return (ans+M)%M;
}
```

不互质的情况

```
int China(int n)
{
    int m1,r1,m2,r2,flag=0,i,d,x,y,c,t;
    scanf("%d%d",&m1,&r1);
    flag=0;
    for(i=1;i<n;i++)
```

```
    {
        scanf("%d%d",&m2,&r2);
        if(flag)continue;
        gcd(m1,m2,d,x,y);//d=gcd(m1,m2);x*m1+y*m2=d;
        c=r2-r1;
        if(c%d)//对于方程 m1*x+m2*y=c，如果 c 不是 d 的倍数就无整数解
        {
            flag=1;
            continue;
        }
        t=m2/d;//对于方程 m1x+m2y=c=r2-r1,若(x0,y0)是一组整数解,那么(x0+k*m2/d,y0-k*m1/d)也是一组整数解(k 为任意整数)
                //其中 x0=x*c/d,y0=x*c/d;
        x=(c/d*x%t+t)%t;//保证 x0 是正数，因为 x+k*t 是解，(x%t+t)%t 也必定是正数解(必定存在某个 k 使得(x%t+t)%t=x+k*t)
        r1=m1*x+r1;//新求的 r1 就是前 i 组的解，Mi=m1*x+M(i-1)=r2-m2*y(m1 为前 i 个 m 的最小公倍数);对 m2 取余时，余数为 r2；
                    //对以前的 m 取余时，Mi%m=m1*x%m+M(i-1)%m=M(i-1)%m=r
        m1=m1*m2/d;
    }
    if(flag)return -1;
    else return r1;
}
```

素数

普通筛选法

```
const int maxn=10000001;
const int maxc=700000;
int vis[maxn];//vis[i]=0 时，i 为素数或者 1，否则为合数
int prime[maxc];
//筛素数
void sieve(int n)
{
    int m=(int)sqrt(n+0.5);//避免浮点误差
    memset(vis,0,sizeof(vis));
    for(int i=2;i<=m;i++)if(!vis[i])
        for(int j=i*i;j<=n;j+=i)vis[j]=1;
}
//生成素数表,存在 prime 数组中，返回素数个数
int primes(int n)
{
    sieve(n);
```

```
    int t=0;
    for(int i=2;i<=n;i++)if(!vis[i])
        prime[t++]=i;
    return t;
}
```

线性筛法：在 O（n）时间复杂度内找出所有的素数

```
void init()//预处理，找出所有 1e7 以内的素数，以减少查找 1e14 范围数的因子的时间
{          //现行筛素数的方法，时间复杂度为 O(n)
    memset(check,false,sizeof(check));
    int i,j;
    tot=0;
    for(i=2;i<=1e7;i++)
    {
        if(!check[i])prime[tot++]=i;
        for(j=0;j<tot;j++)
        {
            if(i*prime[j]>1e7)break;
            check[i*prime[j]]=true;
            if(i%prime[j]==0)break;
        }
    }
    //printf("%d\n",tot);
    //for(i=0;i<20;i++)
    //    printf("prime[%d]:%d\n",i,prime[i]);
}
```

欧拉 phi 函数

求不超过 n 且与 n 互质的正整数个数

```
int euler_phi(int n)//求单个欧拉函数
{
    int m=(int)sqrt(n+0.5);
    int i,ans=n;
    for(i=2;i<=m;i++)
        if(n%i==0)
        {
            ans=ans/i*(i-1);
            while(n%i==0)n/=i;
        }
```

```
    if(n>1)ans=ans/n*(n-1);
    return ans;
}

int phi[maxn];
void euler_phi()
{
    int i,j,k;
    //欧拉函数，phi[i]表示不超过 i 的与 i 互质的整数个数
    for(i=2;i<maxn;i++)phi[i]=0;
    phi[1]=1;
    for(i=2;i<maxn;i++)
        if(!phi[i])
        for(j=i;j<=maxn;j+=i){
            if(!phi[j])phi[j]=j;
            phi[j]=phi[j]/i*(i-1);
        }
}
```

三分法

```
for(i=0;i<100;i++)
{
    m1=l+(r-l)/3;
    m2=r-(r-l)/3;
    if(find(m1)<find(m2)) r=m2;
    else l=m1;
}
```

大素数判断和素因子分解(POJ 1811)

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#include<iostream>
#include<algorithm>
using namespace std;

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

```
// Miller_Rabin 算法进行素数测试
//速度快，而且可以判断 <2^63 的数
//************************************************************
*
const int S=20;//随机算法判定次数，S 越大，判错概率越小

//计算 (a*b)%c.   a,b 都是 long long 的数，直接相乘可能溢出的
//  a,b,c <2^63
long long mult_mod(long long a,long long b,long long c)
{
    a%=c;
    b%=c;
    long long ret=0;
    while(b)
    {
        if(b&1){ret+=a;ret%=c;}
        a<<=1;
        if(a>=c)a%=c;
        b>>=1;
    }
    return ret;
}

//计算  x^n %c
long long pow_mod(long long x,long long n,long long mod)//x^n%c
{
    if(n==1)return x%mod;
    x%=mod;
    long long tmp=x;
    long long ret=1;
    while(n)
    {
        if(n&1) ret=mult_mod(ret,tmp,mod);
        tmp=mult_mod(tmp,tmp,mod);
        n>>=1;
    }
    return ret;
}

//以 a 为基,n-1=x*2^t      a^(n-1)=1(mod n)  验证 n 是不是合数
//一定是合数返回 true,不一定返回 false
bool check(long long a,long long n,long long x,long long t)
{
    long long ret=pow_mod(a,x,n);
```

```
    long long last=ret;
    for(int i=1;i<=t;i++)
    {
        ret=mult_mod(ret,ret,n);
        if(ret==1&&last!=1&&last!=n-1) return true;//合数
        last=ret;
    }
    if(ret!=1) return true;
    return false;
}

// Miller_Rabin()算法素数判定
//是素数返回 true.(可能是伪素数，但概率极小)
//合数返回 false;

bool Miller_Rabin(long long n)
{
    if(n<2)return false;
    if(n==2)return true;
    if((n&1)==0) return false;//偶数
    long long x=n-1;
    long long t=0;
    while((x&1)==0){x>>=1;t++;}
    for(int i=0;i<S;i++)
    {
        long long a=rand()%(n-1)+1;//rand()需要 stdlib.h 头文件
        if(check(a,n,x,t))
            return false;//合数
    }
    return true;
}


//************************************************
//pollard_rho 算法进行质因数分解
//************************************************
long long factor[100];//质因数分解结果（刚返回时是无序的）
int tol;//质因数的个数。数组小标从 0 开始

long long gcd(long long a,long long b)
{
    if(a==0)return 1;//???????
    if(a<0) return gcd(-a,b);
    while(b)
```

```
    {
        long long t=a%b;
        a=b;
        b=t;
    }
    return a;
}

long long Pollard_rho(long long x,long long c)
{
    long long i=1,k=2;
    long long x0=rand()%x;
    long long y=x0;
    while(1)
    {
        i++;
        x0=(mult_mod(x0,x0,x)+c)%x;
        long long d=gcd(y-x0,x);
        if(d!=1&&d!=x) return d;
        if(y==x0) return x;
        if(i==k){y=x0;k+=k;}
    }
}
//对 n 进行素因子分解
void findfac(long long n)
{
    if(Miller_Rabin(n))//素数
    {
        factor[tol++]=n;
        return;
    }
    long long p=n;
    while(p>=n)p=Pollard_rho(p,rand()%(n-1)+1);
    findfac(p);
    findfac(n/p);
}

int main()
{
    //srand(time(NULL));//需要 time.h 头文件//POJ 上 G++不能加这句话
    long long n;
    while(scanf("%I64d",&n)!=EOF)
    {
        tol=0;
```

```
        findfac(n);
        for(int i=0;i<tol;i++)printf("%I64d ",factor[i]);
        printf("\n");
        if(Miller_Rabin(n))printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

数据结构

手写堆

```
int pl[MAXN];
void insert(int u)
{
    int t=u/2;
    while (t>0 && pl[t]>pl[u])
    {
        swap(pl[t],pl[u]);
        u=t;
        t=u/2;
    }
}

void down(int u)
{
    int t=1;
    while (t*2<=u)
    {
        int p=t*2;
        if (p<u && pl[p]>pl[p+1]) p++;
        if (pl[p]<pl[t])
        {
            swap(pl[p],pl[t]);
            t=p;
        }else break;
    }
}
```

```
int main()
{
    scanf("%d",&n);
    for (i=1;i<=n;i++)
    {
        scanf("%d",&k);
        if (k==1)
        {
            num++;scanf("%d",&pl[num]);
            insert(num);
        }else
        if (k==2)
        {
            printf("%d\n",pl[1]);
        }else
        if (k==3)
        {
            pl[1]=pl[num];
            pl[num--]=0;
            down(num);
        }
    }
    return 0;
}
```

LCA

离线 tarjan

```
int u,v,w,d;
int edge,head[MAXN];

int fa[MAXN],query[MAXN][3],dist[MAXN];
bool vis[MAXN];

struct node
{
    int v,id;
    node (int _v,int _id):v(_v),id(_id){}
};

int find(int x)
```

```
{
    if (x==fa[x]) return fa[x];
    return fa[x]=find(fa[x]);
}

vector <vector<node> > mp;

struct edgenode
{
    int to,next,w;
} G[MAXM];

void add_edge(int x,int y,int w)
{
    G[edge].to=y;
    G[edge].w=w;
    G[edge].next=head[x];
    head[x]=edge++;
}


void tarjan(int u)
{
    vis[u]=true;
    for (int i=0;i<mp[u].size();i++)
    {
        int v=mp[u][i].v,id=mp[u][i].id;
        if (vis[v]) query[id][2]=find(v);
    }

    for (int i=head[u];i!=-1;i=G[i].next)
    {
        int v=G[i].to,w=G[i].w;
        if (!vis[v])
        {
            dist[v]=dist[u]+w;
            tarjan(v);
            fa[v]=u;
        }
    }
}


int main()
```

```
{
    scanf("%d",&T);
    while (T--)
    {
        memset(head,-1,sizeof(head));
        edge=0;

        scanf("%d%d",&n,&m);
        for (i=0;i<n-1;i++)
        {
            scanf("%d%d%d",&u,&v,&w);
            add_edge(u,v,w);
            add_edge(v,u,w);
        }

        mp.clear();
        mp.resize(n+4);

        for (i=0;i<m;i++)
        {
            scanf("%d%d",&u,&v);
            query[i][0]=u;
            query[i][1]=v;
            mp[u].push_back(node(v,i));
            mp[v].push_back(node(u,i));
        }
        for (i=1;i<=n;i++) fa[i]=i;
        memset(vis,0,sizeof(vis));
        dist[1]=0;
        tarjan(1);

        for (i=0;i<m;i++)
        {
            u=query[i][0];
            v=query[i][1];
            d=query[i][2];
            printf("%d\n",dist[u]+dist[v]-2*dist[d]);
        }
    }
    return 0;
}
```

在线倍增法

```
#pragma comment(linker, "/STACK:1024000000,1024000000")
#include <cstdio>
#include <cstring>
#include <cmath>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;
const int maxn=40004;
struct node
{
    int to,w;
    node(int a=0,int b=0)
    {
        to=a;
        w=b;
    }
};
vector<node>e[maxn];
int f[maxn],dis[maxn],deep[maxn],p[maxn][20],n;
void dfs(int u,int pre,int t)
{
    int i,num;
    deep[u]=t;//深度
    f[u]=pre;//父节点
    num=e[u].size();
    for(i=0; i<num; i++)
    {
        int v=e[u][i].to;
        if(v!=pre)
        {
            dis[v]=dis[u]+e[u][i].w;//距离跟的距离
            dfs(v,u,t+1);
        }
    }
}
void init()
{
    //p[i][j]表示 i 结点的第 2^j 祖先
    int i,j;
    for(j=0; (1<<j)<=n; j++)
```

```
        for(i=1; i<=n; i++)
            p[i][j]=-1;
    for(i=1; i<=n; i++)p[i][0]=f[i];
    for(j=1; (1<<j)<=n; j++)
        for(i=1; i<=n; i++)
            if(p[i][j-1]!=-1)
                p[i][j]=p[p[i][j-1]][j-1];//i 的第 2^j 祖先就是 i 的第
2^(j-1)祖先的第 2^(j-1)祖先
}
int lca(int a,int b)//最近公共祖先
{
    int i,j;
    if(deep[a]<deep[b])swap(a,b);
    for(i=0; (1<<i)<=deep[a]; i++);
    i--;
    //使 a，b 两点的深度相同
    for(j=i; j>=0; j--)
        if(deep[a]-(1<<j)>=deep[b])
            a=p[a][j];
    if(a==b)return a;
    //倍增法，每次向上进深度 2^j，找到最近公共祖先的子结点
    for(j=i; j>=0; j--)
    {
        if(p[a][j]!=-1&&p[a][j]!=p[b][j])
        {
            a=p[a][j];
            b=p[b][j];
        }
    }
    return f[a];
}
int main()
{
    int T;
    scanf("%d",&T);
    while(T--)
    {
        int m,i,a,b,c,ans;
        scanf("%d%d",&n,&m);
        for(i=1; i<=n; i++)e[i].clear();
        for(i=1; i<n; i++)
        {
            scanf("%d%d%d",&a,&b,&c);
            e[a].push_back(node(b,c));
```

```
            e[b].push_back(node(a,c));
        }
        dis[1]=0;
        dfs(1,-1,0);//找到各点的深度和各点的父节点以及距离根的距离
        init();     //初始各个点的 2^j 祖先是谁
        for(i=0; i<m; i++)
        {
            scanf("%d%d",&a,&b);
            ans=dis[a]+dis[b]-2*dis[lca(a,b)];
            printf("%d\n",ans);
        }
    }
    return 0;
}
```

区间更新线段树

```
struct tree
{
    int l,r;
    int sum,lazy;
};

tree t[MAXN*4];
int a[MAXN];
int temp,l,r,s;

void pushdown(int k)
{
    int temp=t[k].lazy;
    t[lc].lazy+=temp;
    t[rc].lazy+=temp;
    t[lc].sum+=(t[lc].r-t[lc].l+1)*temp;
    t[rc].sum+=(t[rc].r-t[rc].l+1)*temp;
    t[k].lazy=0;
}

void buildtree(int k,int l,int r)
{
    if (l==r)
    {
        t[k].l=l;t[k].r=r;
        t[k].sum=a[l];
```

```
        t[k].lazy=0;
        return;
    }
    t[k].l=l;
    t[k].r=r;
    t[k].lazy=0;
    buildtree(lc,l,mii);
    buildtree(rc,mii+1,r);
    t[k].sum=t[lc].sum+t[rc].sum;
}

void add(int k,int l,int r,int s)
{
    if ((t[k].l==l)&&(t[k].r==r))
    {
        t[k].lazy+=s;
        t[k].sum+=(r-l+1)*s;
        return;
    }
    int mid=(t[k].l+t[k].r)/2;
    if (t[k].lazy!=0) pushdown(k);
    if (r<=mid) add(lc,l,r,s);else
    if (mid<l) add(rc,l,r,s);
    else
    {
        add(lc,l,mid,s);
        add(rc,mid+1,r,s);
    }
    t[k].sum=t[lc].sum+t[rc].sum;
}


int query(int k,int l,int r)
{
    if ((t[k].l==l)&&(t[k].r==r))
    {
        return t[k].sum;
    }
    int mid=(t[k].l+t[k].r)/2;
    if (t[k].lazy!=0) pushdown(k);
    if (mid<l) return query(rc,l,r);
    if (r<=mid) return query(lc,l,r);
    return query(lc,l,mid)+query(rc,mid+1,r);
}
```

```
int main()
{
    scanf("%d%d",&n,&m);
    for (i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
    }
    buildtree(1,1,n);
    while (m--)
    {
        scanf("%d",&temp);
        if (temp==0)
        {
            scanf("%d%d",&l,&r);
            printf("%d\n",query(1,l,r));
        } else
        {
            scanf("%d%d%d",&l,&r,&s);
            add(1,l,r,s);
        }
    }
}
```

树状数组

区间查询，单点修改

```
int Sum(int x)
{
    int sum=0;
    while(n>0)
    {
         sum+=c[x];
         x=x-(x&(-x));
    }
    return sum;
}
void change(int i,int x)
{
     while(i<=n)
     {
          c[i]=c[i]+x;
          i=i+lowbit(i);
```

```
    }
}
```

单点查询，区间修改

```
int C[MAXN];

void update(int x,int y)
{
    for (int i=x;i>0;i-=(-i)&i)
       C[i]+=y;
}
int query(int x)
{
    int s=0;
    for (int k=x;k<=n;k+=(-k)&k) s+=C[k];
    return s;
}
```

求逆序对
维护树状数组，每加入一个数，查询有多少个数比它大（总数减去小于等于它的个数）

字符串

字典树

```
struct Trie
{
    int ch[MAXN][26];
    int cnt;
    Trie()
    {
        cnt=1;
        memset(ch[0],0,sizeof(ch[0]));
    }

    void insert(char *s)
    {
        int u=0,len=strlen(s);
        for (int i=0; i<len; i++)
        {
```

```
            int c=s[i]-'a';
            if (!ch[u][c])
            {
                memset(ch[cnt],0,sizeof(ch[cnt]));
                ch[u][c]=cnt++;
            }
            u=ch[u][c];
        }
    }
} tree;
```

KMP 与扩展 KMP

普通 KMP：求出 A[i]往前和 B 的前缀匹配的最大匹配长度，记为 ex[i]
扩展 KMP：求出 A[i]往后和 B 的前缀匹配的最大匹配长度，记为 ex[i]

【KMP】

```
lenA = strlen(A);lenB = strlen(B);
next[0] = lenB;
int j = 0;
re2(i, 1, lenB)
{
    while (j && B[i] != B[j]) j = next[j - 1];
    if (B[i] == B[j]) j++;
    next[i] = j;
}
j = 0;
re(i, lenA)
{
    while (j && A[i] != B[j]) j = next[j - 1];
    if (A[i] == B[j]) j++;
    ex[i] = j;
}
```

【扩展 KMP】

```
lenA = strlen(A);
lenB = strlen(B);
next[0] = lenB;
next[1] = lenB - 1;
re(i, lenB-1) if (B[i] != B[i + 1])
{
```

```
    next[1] = i;
    break;
}
int j, k = 1, p, L;
re2(i, 2, lenB)
{
    p = k + next[k] - 1;
    L = next[i - k];
    if (i + L <= p) next[i] = L;
    else
    {
        j = p - i + 1;
        if (j < 0) j = 0;
        while (i + j < lenB && B[i + j] == B[j]) j++;
        next[i] = j;
        k = i;
    }
}
int minlen = lenA <= lenB ? lenA : lenB;
ex[0] = minlen;
re(i, minlen) if (A[i] != B[i])
{
    ex[0] = i;
    break;
}
k = 0;
re2(i, 1, lenA)
{
    p = k + ex[k] - 1;
    L = next[i - k];
    if (i + L <= p) ex[i] = L;
    else
    {
        j = p - i + 1;
        if (j < 0) j = 0;
        while (i + j < lenA && j < lenB && A[i + j] == B[j]) j++;
        ex[i] = j;
        k = i;
    }
}
```

AC 自动机

插入操作：ac.insert(p[i],i);
构造失配函数：ac.getFail();
计算模板 T 的 cnt 值：ac.find(T);

```
struct ACauto
{
    int  ch[MAXN][26];//  字典树，类似于前向星，ch[i][j]为当前编号为 i 的结点，下一个字符为 j 的所指向的编号。
    int size;
    int f[MAXN],last[MAXN],val[MAXN],cnt[MAXN];
    //val 用来在字典树中的模板串末尾处标记，标记为模板串的序号（从 1 开始）
    //last 后缀链接：结点 J 沿着失配指针往回走时，遇到的下一个单词尾结点。
    //cnt 用来统计配对数，每一个模板对应一个值，所以大小为模板数数量。只在 print 函数中使用

    void init()//初始化
    {
        size=1;//字典树中的节点数
        memset(ch[0],0,sizeof(ch[0]));//字典树
        memset(cnt,0,sizeof(cnt)); //用于统计配对数
    }

    int idx(char c)//用于返回编号
    {
        return c-'a';
    }

    void insert(char *s,int v)//将字符串 s 插入字典树中，其中 v 是字符串的编号，从 1 开始编号
    {
        int u=0,len=strlen(s);
        for (int i=0;i<len;i++)
        {
            int c=idx(s[i]);
            if (!ch[u][c])
            {
                memset(ch[size],0,sizeof(ch[size]));
                val[size]=0;
                ch[u][c]=size++;
            }
            u=ch[u][c];
        }
```

```
        val[u]=v;//在字符串末尾做出标记，标记为字符串的编号 i
    }

    void print(int j)//用于输出处理,
    {
        if (j)
        {
            cnt[val[j]]++;//成功配对数加 1
            print(last[j]);//继续沿后缀链接走检查是否和某个模板匹配。
        }
    }

    int getFail()//BFS 构造失配函数
    {
        queue <int> q;
        f[0]=0;
        for (int c=0;c<26;c++)//把各个模板的第一个字符压入队列中
        {
            int u=ch[0][c];
            if (u)
            {
                f[u]=0;
                q.push(u);
                last[u]=0;
            }
        }

        while (!q.empty())
        {
            int r=q.front(); q.pop();
            for (int c=0;c<26;c++)
            {
                int u=ch[r][c];
                if (!u)
                {
                    ch[r][c]=ch[f[r]][c];//如果节点不存在,直接链接到->失配
边所指向的节点，这样能够化简计算
                    continue;
                }
                q.push(u);
                f[u]=ch[f[r]][c];//构造当前节点的失配函数：如果失配，找到失
配点的父亲节点 r，父亲沿着失配边 f[r]走向下一个节点即可。
                last[u]=val[f[u]]?f[u]:last[f[u]];//构造后缀链接：如果沿
失配指针走的节点是尾节点，就标记为失配指针指向的节点,
```

```
                                                    //否则标记为其后缀链接的值
（类似于递归）。
            }
        }
    }


    void find(char *T)//AC 自动机主函数，在文本串 T 中寻找模板
    {
        int n=strlen(T);
        int j=0;
        for (int i=0;i<n;i++)
        {
            int c=idx(T[i]);//返回字符的编号
            while(j&&!ch[j][c]) j=f[j];//如果字符不存在，即失配，就顺着失
配边走，直到可以匹配
            j=ch[j][c];//如果可以匹配，就走向下一个结点
            if (val[j]) print(j);//如果 j 指向某个模板的尾部则输出
            else if (last[j]) print(last[j]);//即使不是某个模板的尾部也
要沿后缀链接走，检查是否为某个模板的尾部。
        }
    }
}ac;
```

Manacher 最长回文串

```
void kp()
{
    int i;
    int mx=0;
    int id;
    for (i=n; str[i]!=0; i++) //清除 n 后边多余的部分
        str[i]=0;//没有这一句有问题。。就过不了 ural1297，比如数据：ababaaba
    for (i=1; i<n; i++)
    {
        if (mx>i) p[i]=_min(p[2*id-i],p[id]+id-i);
            //因为是从左往右扫描的这里 i>id,2*id-i 是 i 关于 id 的对称点，该对
称点在 id 的左端
            //p[id]+id 是描述中的 mx，即 id 向右延伸的端点位置
            //显然向右延伸是可能超出 mx 的，所以要有下边的 for 循环
        else p[i]=1;
        for (;str[i+p[i]]==str[i-p[i]];p[i]++);
```

```
        if (p[i]+i>mx)//更新 mx 与 id，因为 mx 是向右延伸的最大长度，所以实时更新
        {
            mx=p[i]+i;
            id=i;
        }
    }
}

void init()//处理字符串
{
    int i,j,k;
    str[0]='$';
    str[1]='#';
    for(i=0;i<n;i++)
    {
        str[i*2+2]=s[i];
        str[i*2+3]='#';
    }
    n=n*2+2;
    s[n]=0;
}

int main()
{
    int i,ans;
    while(scanf("%s",s)!=EOF)
    {
        n=strlen(s);//n 为给定字符串 s 的长度
        init();
        kp();
        ans=0;
        for(i=0; i<n; i++)
            if(p[i]>ans)//寻找最大的长度 ans
                ans=p[i];
        printf("%d\n",ans-1);
    }
    return 0;
}
```

RMQ

```
int a[50005],mm[50005][30],mi[50005][30];
void init()
{
    for (int i=1;i<=n;i++) mm[i][0]=mi[i][0]=a[i];
    for (int j=1;(1<<j)<=n;j++)
        for (int i=1;i+(1<<j)-1<=n;i++)
        {
            mm[i][j]=max(mm[i][j-1],mm[i+(1<<(j-1))][j-1]);
            mi[i][j]=min(mi[i][j-1],mi[i+(1<<(j-1))][j-1]);
        }
}

int RMQ(int l,int r)
{
    int k=0;
    while ((1<<(k+1))<=r-l+1) k++;
    int ans1=max(mm[l][k],mm[r-(1<<k)+1][k]);//区间最大值
    int ans2=min(mi[l][k],mi[r-(1<<k)+1][k]);//区间最小值
    return ans1-ans2;
}

int main()
{
    while (~scanf("%d%d",&n,&m))
    {
        clr(mm,0);clr(mi,0);
        for (i=1;i<=n;i++) scanf("%d",&a[i]);
        init();
        for (i=1;i<=m;i++)
        {
            int l,r;
            scanf("%d%d",&l,&r);
            printf("%d\n",RMQ(l,r));
        }
    }
    return 0;
}
```

并查集

```
int fa[MAXN];
int findfa(int x)
{
    return x==fa[x]?x:fa[x]=findfa(fa[x]);
}
void join(int x,int y)
{
    fa[findfa(x)]=findfa(y);
}
```

计算几何

基础模板

```
const double pi=acos(-1.0);
int dcmp(double x) {if(fabs(x) < eps) return 0; else return x < 0 ?
-1 : 1; }
struct Vector
{
    double x, y;
    Vector (double x=0, double y=0) :x(x),y(y) {}
    Vector operator + (const Vector &B) const { return Vector
(x+B.x,y+B.y); }
    Vector operator - (const Vector &B) const { return Vector(x -
B.x, y - B.y); }
    Vector operator * (const double &p) const { return Vector(x*p,
y*p); }
    Vector operator / (const double &p) const { return Vector(x/p,
y/p); }
    double operator * (const Vector &B) const { return x*B.x +
y*B.y;}//点积
    double operator ^ (const Vector &B) const { return x*B.y -
y*B.x;}//叉积
    bool operator < (const Vector &b) const { return x < b.x || (x
== b.x && y < b.y); }
    bool operator ==(const Vector &b) const { return dcmp(x-b.x) ==
0 && dcmp(y-b.y) == 0; }
};
```

```
typedef Vector Point;
Point Read(){double x, y;scanf("%lf%lf", &x, &y);return Point(x, y);}
double Length(Vector A){ return sqrt(A*A); }//向量的模
double Angle(Vector A, Vector B){return acos(A*B / Length(A) / Length(B)); }//向量的夹角，返回值为弧度
double Area2(Point A, Point B, Point C){ return (B-A)^(C-A); }//向量 AB 叉乘 AC 的有向面积
Vector VRotate(Vector A, double rad){return Vector(A.x*cos(rad) - A.y*sin(rad), A.x*sin(rad) + A.y*cos(rad));}//向量 A 旋转 rad 弧度
Point  PRotate(Point A, Point B, double rad){return A + VRotate(B-A, rad);}//将 B 点绕 A 点旋转 rad 弧度
Vector Normal(Vector A){double l = Length(A);return Vector(-A.y/l, A.x/l);}//求向量 A 向左旋转 90°的单位法向量,调用前确保 A 不是零向量

Point  GetLineIntersection/*求直线交点，调用前要确保两条直线有唯一交点*/(Point P, Vector v, Point Q, Vector w){double t = (w^(P - Q)) / (v^w);return P + v*t;}//在精度要求极高的情况下，可以自定义分数类
double DistanceToLine/*P 点到直线 AB 的距离*/(Point P, Point A, Point B){Vector v1 = B - A, v2 = P - A;return fabs(v1^v2) / Length(v1);}//不加绝对值是有向距离
double DistanceToSegment/*点到线段的距离*/(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(v1*v2)<0) return Length(v2);else
    if (dcmp(v1*v3)>0) return Length(v3);else
    return fabs(v1^v2)/Length(v1);
}

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

bool OnSegment/*判断点是否在线段上(含端点)*/(Point P,Point a1,Point a2)
{
    Vector v1=a1-P,v2=a2-P;
    if     (dcmp(v1^v2)==0     &&     min(a1.x,a2.x)<=P.x          &&
P.x<=max(a1.x,a2.x)  && min(a1.y,a2.y)<=P.y && P.y<=max(a1.y,a2.y))
return true;
    return false;
}
```

```
bool SegmentInter/*线段相交判定*/(Point a1, Point a2, Point b1, Point
b2)
{
    //if    (OnSegment(a1,b1,b2)    ||    OnSegment(a2,b1,b2)    ||
OnSegment(b1,a1,a2) || OnSegment(b2,a1,a2)) return 1;
    //如果只判断线段规范相交（不算交点）,上面那句可以删掉
    double c1=(a2-a1)^(b1-a1),c2=(a2-a1)^(b2-a1);
    double c3=(b2-b1)^(a1-b1),c4=(b2-b1)^(a2-b1);
    return dcmp(c1)*dcmp(c2)<0 && dcmp(c3)*dcmp(c4)<0;
}

bool InTri/*判断点是否在三角形内*/(Point P, Point a,Point b,Point c)
{
    if    (dcmp(fabs((c-a)^(c-b))-fabs((P-a)^(P-b))-fabs((P-b)^(P-
c))-fabs((P-a)^(P-c)))==0) return true;
    return false;
}

double PolygonArea/*求多边形面积，注意凸包 P 序号从 0 开始*/(Point *P ,int
n)
{
    double ans = 0.0;
    for(int i=1;i<n-1;i++)
        ans+=(P[i]-P[0])^(P[i+1]-P[0]);
    return ans/2;
}
bool  CrossOfSegAndLine/*判断线段是否与直线相交*/(Point  a1,Point
a2,Point b1,Vector b2)
{
    if (OnSegment(b1,a1,a2) || OnSegment(b1+b2,a1,a2)) return true;
    return dcmp(b2^(a1-b1))*dcmp(b2^(a2-b1))<0;
}
double Cross/*B-A 和 C-A 的叉积*/(Point A, Point B,Point C)
{
    return (B-A)^(C-A);
}
double dis_pair_seg/*两条线段间的最短距离*/(Point p1, Point p2, Point
p3, Point p4)
{
    return       min(min(DistanceToSegment(p1,       p3,       p4),
DistanceToSegment(p2, p3, p4)),
     min(DistanceToSegment(p3, p1, p2), DistanceToSegment(p4, p1,
p2)));
```

```
}
```

凸包

```
int ConvexHull(Point* p,int n,Point* ch)
{
    sort(p,p+n);
    int m=0;
    for(int i=0;i<n;++i)
    {
        while(m>1&&((ch[m-1]-ch[m-2])^(p[i]-ch[m-2]))<=0) m--;
        ch[m++]=p[i];
    }
    int k=m;
    for(int i=n-2;i>=0;i--)
    {
        while(m>k&&((ch[m-1]-ch[m-2])^(p[i]-ch[m-2]))<=0) m--;
        ch[m++]=p[i];
    }
    if(n>1) m--;
    return m;
}
```

旋转卡壳

```
double rotating_calipers(Point *ch,int n)
{
    int q=1;
    double ans=0;
    ch[n]=ch[0];
    for(int p=0; p<n; p++)
    {
        while(((ch[q+1]-ch[p+1])^(ch[p]-ch[p+1]))>((ch[q]-
ch[p+1])^(ch[p]-ch[p+1]))) q=(q+1)%n;
        ans=max(ans,max(Length(ch[p]-ch[q]),Length(ch[p+1]-
ch[q+1])));
    }
    return ans;
}
//为什么是大于号而不是大于等于号？因为 ch 默认是凸多边形
//旋转卡壳只是一种思想，需要灵活运用
```

有向直线半平面交

```
struct Line//有向直线
{
    Point p;
    Vector v;
    double ang;
    Line()    { }
    Line(Point p, Vector v): p(p), v(v)    { ang = atan2(v.y, v.x); }
    Point point(double t)
    {
        return p + v*t;
    }
    bool operator < (const Line& L) const
    {
        if (fabs(ang-L.ang)<eps)
        {
            return ((v)^(L.p-p))<0;
        }
        return ang < L.ang;
    }
};

bool OnRight(Line L,Point p)//判断是否p点在有向直线右边
{
    return ((L.v^(p-L.p))<0);
}
Point GetIntersection (Line a,Line b)//求两个有向直线的交点
{
    Vector u=a.p-b.p;
    double t=(b.v^u)/(a.v^b.v);
    return a.p+a.v*t;
}


bool IsParallel(Line a,Line b)//判断两条有向直线是否平行
{
    if (dcmp(a.v ^ b.v)==0) return 1;
    return 0;
}

Line q[MAXN];
```

```
int HalfPlane(Line *L,int n,Point *poly)//半平面交，输入直线数组 L(需要保证逆时针顺序)，n 是直线的个数，poly 用于输出，返回值为 poly 中点的个数
{
    sort(L,L+n);
    int m,i;
    for (m=i=1;i<n;i++)
    {
        if (dcmp(L[i].ang-L[i-1].ang)!=0) L[m++]=L[i];
    }
    n=m;
    int l=0,r=1;
    q[0]=L[0];q[1]=L[1];
    for (int i=2;i<n;i++)
    {
        //if  (IsParallel(q[r],q[r-1])  ||  IsParallel(q[l],q[l+1]))
return 0;
        while (l < r && OnRight(L[i],GetIntersection(q[r-1],q[r])))
r--;
        while (l < r && OnRight(L[i],GetIntersection(q[l],q[l+1])))
l++;
        q[++r]=L[i];
    }
    while (l<r && OnRight(q[l],GetIntersection(q[r-1],q[r]))) r--;
    while (l<r && OnRight(q[r],GetIntersection(q[l],q[l+1]))) l++;
    if (r-l<=1 ) return 0;
    q[r+1]=q[l];
    m=0;
    for (int i=l;i<=r;i++) poly[m++]=GetIntersection(q[i],q[i+1]);
    return m;
}
```

转角法判定点 P 是否在多边形内部

（多边形不一定是凸多边形）

```
int isPointInPolygon(Point P, Point* Poly, int n)//转角法判定点 P 是否在多边形内部
{
    int wn=0;
    for(int i = 0; i < n; ++i)
    {
        if(OnSegment(P, Poly[i], Poly[(i+1)%n]))    return -1;    //
在边界上
        int k = dcmp((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
}
```

动态规划

区间类型

```
for (j=1;j<=n;j++)
    for (i=1;i<=n-j+1;i++)
        for (k=i;k<=i+j-1;k++)
            f[i][i+j-1]=f[i][k]+f[k][i+j-1];

for (i=n;i>=1;i--)
    for (j=i+1;j<=n;j++)
        for (k=i;k<=j-1;k++)
            f[i][j]=f[i][k]+f[k][j];
```

最长上升子序列

(1) 对序列 B 排序
(2) 计算 A 中每个元素在 B 中的序号，并构成新序列
(3) 使用 LIS 的方法计算最长严格递增子序列
(4) 获取最长公共子序列

```
struct node
{
    char c;
    int num;
} u[10005];
bool cmp(node a,node b)
{
    if (a.c==b.c) return a.num>b.num;
```

```
    return a.c<b.c;
}
vector <int> p;
char a[10005],b[10005],c[10005];
int lena,lenb,dp[10005];
int main()
{
    scanf("%s",a);//读入 a 串
    scanf("%s",b);//读入 b 串
    lena=strlen(a);
    lenb=strlen(b);
    for (i=0; i<lenb; i++)
    {
        u[i].c=b[i];
        u[i].num=i;
    }
    sort(u,u+lenb,cmp);//对 b 串排序
    for (i=0; i<lenb; i++) //排序后存入字符串 c 中，便于使用 lower_bound
    {
        c[i]=u[i].c;
    }
    c[lenb]='\0';
    for (i=0; i<lena; i++) //计算 A 中每个元素在 B 中的序号
    {
        k=lower_bound(c,c+lenb,a[i])-c;
        while (k<lenb && a[i]==c[k])
        {
            p.push_back(u[k].num);
            k++;
        }
    }
    n=p.size();
    memset(dp,0,sizeof(dp));//计算最长上升子序列
    num=0;
    for (i=0; i<n; i++)
    {
        if (p[i]>dp[num])
        {
            dp[++num]=p[i];
        }
        else
        {
            k=lower_bound(dp+1,dp+1+num,p[i])-dp;
            dp[k]=p[i];
```

```
        }
    }
    printf("%d\n",num);
    return 0;
}
```

最长公共子序列

```
int dp[100005],pre[100005],nam[100005];

void out(int u)
{
     if (pre[u]) out(pre[u]);

     printf("%d ",u);
}

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

    memset(dp,0,sizeof(dp));
    num=0;
    for (i=1;i<=n;i++)
    {
        if (p[i]>dp[num])//若求最长不下降子序列把这里改成大于等于
        {
            dp[++num]=p[i];

            nam[num]=i;//nam[i]用于记录 dp[i]在 p 中的编号，用于输出
            pre[i]=nam[num-1];//pre[i]用于记录 dp[i-1]在 p 中的编号，用于
输出
        }else
        {
            k=lower_bound(dp+1,dp+1+num,p[i])-dp;  //若求最长不下降子序
列把这里 upper_bound
            dp[k]=p[i];
```

```
            nam[k]=i;
            pre[i]=nam[k-1];
        }
    }

    printf("%d\n",num);
    out(nam[num]);
    printf("\n");
    return 0;
}
```

STL

Stack

```
s.top();      //取栈顶
s.push(x);    //入栈
s.pop();      //出栈
s.empty();    //是否为空
s.size();     //栈中元素个数
```

queue

队列 STL 基本用法

```
q.front();    //取队头
q.push(x);    //入队
q.pop();      //出队
q.empty();    //是否为空
q.size();     //队列中元素个数
```

priority_queue

```
q.top();      //取队首元素，默认是队列中的最大值，注意，没有元素时会 Runtime
error
q.empty() //判断为空，则返回 true
q.pop()       //删除第一个元素
q.push()      //加入一个元素
q.size()      //返回队列中元素的个数
```

这样默认是元素大的优先级高

如果要元素小的
```
priority_queue<int vector<int>,greater<int> >  q;//最右边两个>>中间应该有一个空格
```
也可自定义比较函数
```
struct cmp
{
    bool operator()(const int &a,const int &b)
    {
        return a<b;
    }
};
priority_queue<int,vector<int>,cmp> p;
```

set

```
p.insert(K);  //将 K 加入集合 p 中
p.size();     //返回元素个数
p.empty();    //判断是否为空
p.clear();        //清除所有元素
p.erase(K);   //将集合中元素 K 删去
p.count(K);   //返回某个值元素的个数 (只对 multiset 有效，set 的 count()值只有 0 和 1)
p.lower_bound()  //返回指向大于 (或等于) 某值的第一个元素的迭代器，注意这个比直接用 lower_bound 快
p.upper_bound()  //返回指向大于 (不等于) 某值的第一个元素的迭代器，注意这个比直接用 upper_bound 快

set<int>::iterator it;  //定义迭代器
set<int>::reverse_iterator it;    //定义反向迭代器
it=p.begin();     //指向头
it=p.end();       //指向末尾，末尾不包含元素
it=rbegin()       //返回指向集合中最后一个元素的反向迭代器
it=rend()         //返回指向集合中第一个元素的反向迭代器
it=p.find(K);     //返回一个指向被查找到元素的迭代器，如果没有找到返回 p.end();

for (set<int>::iterator it=p.begin();it!=p.end();it++)
{
    cout<<*it<<endl;
}//输出元素，默认按照升序排序

/*注意迭代器指针不能进行四则运算，只能自增自减*/
/*set 的插入，查找，删除操作复杂度均为 o(log(N))*/
```

其他不常用的函数：

```
equal_range()     返回集合中与给定值相等的上下限的两个迭代器
get_allocator()   返回集合的分配器
key_comp()        返回一个用于元素间值比较的函数
max_size()        返回集合能容纳的元素的最大限值
swap()            交换两个集合变量
value_comp()      返回一个用于比较元素间的值的函数
```

自定义比较函数

(1)元素不是结构体：

例：

```
//自定义比较函数 myComp,重载"()"操作符
struct myComp
{
    bool operator()(const your_type &a,const your_type &b)
    {
        return a.data-b.data>0;
    }
}
set<int,myComp>s;
......
set<int,myComp>::iterator it;
```

(2)如果元素是结构体，可以直接将比较函数写在结构体内。

例：

```
struct Info
{
    string name;
    float score;
    //重载"<"操作符，自定义排序规则
    bool operator < (const Info &a) const
    {
        //按 score 从大到小排列
        return a.score<score;
    }
}
set<Info> s;
......
set<Info>::iterator it;
```

关于 multiset

删除操作 st.erase(st.find(x));这样是删除一个数 X,而不是把值为 X 的数全删掉。

Bitset

std::bitset 是 STL 的一部分，
准确地说，std::bitset 是一个模板类，它的模板参数不是类型，而整形的数值（这一特性是 ISO C++2003 的新特性），
有了它我们可以像使用数组一样使用位。
下面看一个例子：

```
#include<bitset>
std::bitset<8> bs;  //它是一个模板，传递的参数告诉编译器 bs 有 8 个位。
```

我们接着看上面的代码，通过上面两行的代码我们得到一个 bitset 的对象 bs，bs 可以装入 8 个位，我们可以通过数组的下标运算符来存取：

```
bs[0]=1;  //把第 0 位设置为 1
bs[3]=true;  //把第 3 位设置为 1,因为 true 可以转换为 1
bs[7]=0;  //这个大家都明白了
```

bitset 被设计为开放的，也就是说一个 bitset 对象可以转换为其它类型的值，典型的，我们想把一个整数设置成具有特定的位模式，我们可以简单地把一个 bitset 转换为一个整数：

```
unsigned long value=bs.to_ulong();
std::bitset<32> bs32(value);
bs32[15]=1;
value=bs32.to_ulong();
```

此外 bitset 还可以也字符串互换，这样我们就可以更直观对 bitset 进行操作了，我只是简单地把我们想要的"01"字符串就可以了：

```
std::bitset<32> bs("011010101001");
std::string str=bs.to_string();
```

【初始化】

```
bitset<n> b;     //b 有 n 位，每位都为 0.参数 n 可以为一个表达式.
```

如 bitset<5> b0;则"b0"为"00000";

```
bitset<n> b(unsigned long u); //b 有 n 位,并用 u 赋值;如果 u 超过 n 位,则顶端被截除
```

如:bitset<5>b0(5);则"b0"为"00101";

```
bitset<n> b(string s); //b 是 string 对象 s 中含有的位串的副本
string bitval("10011");
bitset<5> b0(bitval4); //则"b0"为"10011";

bitset<n> b(s, pos);   //b 是 s 中从位置 pos 开始位的副本,前面的多余位自动填充 0;
string bitval("01011010");
bitset<10> b0(bitval5, 3); //则"b0" 为 "0000011010";
```

```
bitset<n> b(s, pos, num);  //b 是 s 中从位置 pos 开始的 num 个位的副本,如果 num<n,则前面的空位自动填充 0;
string bitval("11110011011");
bitset<6> b0(bitval5, 3, 6);  //则"b0" 为 "100110";
```

【流】

```
os << b    //把 b 中的位集输出到 os 流
os >>b     //输入到 b 中,如"cin>>b",如果输入的不是 0 或 1 的字符,只取该字符前面的二进制位.
```

【属性方法】

```
bool any()    //是否存在置为 1 的二进制位?和 none()相反
bool none()   //是否不存在置为 1 的二进制位,即全部为 0?和 any()相反.
size_t count()   //二进制位为 1 的个数.
size_t size() //二进制位的个数
flip() //把所有二进制位逐位取反
flip(size_t pos) //把在 pos 处的二进制位取反
bool operator[](size_type _Pos)   //获取在 pos 处的二进制位
set()  //把所有二进制位都置为 1
set(pos)  // 把在 pos 处的二进制位置为 1
reset()   // 把所有二进制位都置为 0
reset(pos)    // 把在 pos 处的二进制位置为 0
test(size_t pos) //在 pos 处的二进制位是否为 1?
unsigned long to_ulong()   //用同样的二进制位返回一个 unsigned long 值
string to_string()  //返回对应的字符串.
```