WWB’s模板

目录

[WWB’s模板 1](#_Toc526631727)

[1. 数据结构 4](#_Toc526631728)

[1.1线段树 4](#_Toc526631729)

[1.1.1描述： 4](#_Toc526631730)

[1.1.2代码： 4](#_Toc526631731)

[1.1.3注意： 7](#_Toc526631732)

[1.1.4线段树+离散化（poj2528） 7](#_Toc526631733)

[1.1.5线段树求在给定区间中第一个大于某个数的位置 11](#_Toc526631734)

[STLbitset 13](#_Toc526631735)

[1.2.1使用 13](#_Toc526631736)

[1.3莫队算法 14](#_Toc526631737)

[1.3.1Code： 14](#_Toc526631738)

[1.3.2 原理思想 16](#_Toc526631739)

[1.4划分树 17](#_Toc526631740)

[1.4.1code： 17](#_Toc526631741)

[1.4.2原理思想 19](#_Toc526631742)

[1.5主席树 20](#_Toc526631743)

[1.5.1code 20](#_Toc526631744)

[1.5.2思想： 22](#_Toc526631745)

[1.5.3主席树求区间多少种数 23](#_Toc526631746)

[1.6树状数组（维护前缀和） 25](#_Toc526631747)

[1.6.1code 25](#_Toc526631748)

[1.6.2思想 27](#_Toc526631749)

[1.7树上最长路（O(V)） 27](#_Toc526631750)

[1.7.1code 27](#_Toc526631751)

[1.7.2思路： 28](#_Toc526631752)

[2数论 29](#_Toc526631753)

[2.1最大公约数 29](#_Toc526631754)

[2.2最小公倍数 29](#_Toc526631755)

[2.3快速幂 29](#_Toc526631756)

[2.4求一个正整数的所有约数和 30](#_Toc526631757)

[2.5求a在mod下的逆元（1/a） 32](#_Toc526631758)

[2.5.1code 32](#_Toc526631759)

[2.5.2推导 32](#_Toc526631760)

[2.6费马大定理 32](#_Toc526631761)

[2.7知道其中一条直角边的直角三角形构造另外两条边 33](#_Toc526631762)

[2.8BM模板 33](#_Toc526631763)

[2.9欧拉函数（欧拉表） 35](#_Toc526631764)

[2.9.1code： 35](#_Toc526631765)

[2.9.2例题（UVA11426） 36](#_Toc526631766)

[图论 37](#_Toc526631767)

[3.1拓扑排序 37](#_Toc526631768)

[3.1.1kahn算法 37](#_Toc526631769)

[3.1.2原理思想 39](#_Toc526631770)

[3.2最小生成树 39](#_Toc526631771)

[3.2.1code： 39](#_Toc526631772)

[3.2.2思想，原理 41](#_Toc526631773)

[3.2.3次小生成树 41](#_Toc526631774)

[3.3最短路算法 44](#_Toc526631775)

[3.3.1BellMan\_Ford 44](#_Toc526631776)

[3.3.2Floyed 45](#_Toc526631777)

[3.3.3Dijstra 47](#_Toc526631778)

[3.3.4 SPFA 49](#_Toc526631779)

[3.3.5分层图最短路 51](#_Toc526631780)

[3.3.6第K短路（迪杰斯特拉+A\*算法） 53](#_Toc526631781)

[3.4最大流 56](#_Toc526631782)

[3.4.1code 56](#_Toc526631783)

[3.4.2原理思想 60](#_Toc526631784)

[3.5最小费用最大流 62](#_Toc526631785)

[3.5.1code： 62](#_Toc526631786)

[3.5.2原理思想（拆点思想） 65](#_Toc526631787)

[3.5.3poj2195 65](#_Toc526631788)

[3.6二分图最大匹配 69](#_Toc526631789)

[3.6.1code 69](#_Toc526631790)

[3.6.2原理思想 71](#_Toc526631791)

[3.7二分图完美匹配 71](#_Toc526631792)

[3.7.1code 71](#_Toc526631793)

[3.7.2原理思想 71](#_Toc526631794)

[3.8强连通分量（缩点） 71](#_Toc526631795)

[3.8.1code（java） 71](#_Toc526631796)

[3.8.2原理思想 74](#_Toc526631797)

[3.8.3缩点（Tarjan）+SPFA 75](#_Toc526631798)

[4字符串 78](#_Toc526631799)

[4.1回文 78](#_Toc526631800)

[4.2trie 81](#_Toc526631801)

[4.2.1code（poj2001） 81](#_Toc526631802)

[4.2.2思想 82](#_Toc526631803)

[4.3kmp 83](#_Toc526631804)

[4.3.1code 83](#_Toc526631805)

[4.3.2思想： 84](#_Toc526631806)

[4.3.3kmp变形（hdu1686） 84](#_Toc526631807)

[4.3.4求一个字符串的最小循环节 85](#_Toc526631808)

[算法技巧 86](#_Toc526631809)

[5.1求逆序对 86](#_Toc526631810)

[5.1.1code： 86](#_Toc526631811)

[5.1.2思想 87](#_Toc526631812)

[6.思想 87](#_Toc526631813)

[6.1状态压缩 87](#_Toc526631814)

[6.1.1题目 87](#_Toc526631815)

[6.1.2code 89](#_Toc526631816)

[6.1.3思想 91](#_Toc526631817)

[7.其他 91](#_Toc526631818)

[7.1hdu6301 91](#_Toc526631819)

[7.1.1描述 91](#_Toc526631820)

[7.1.2code 93](#_Toc526631821)

[7.1.3问题及思路 94](#_Toc526631822)

[7.2银行家算法优化 94](#_Toc526631823)

[7.2.1code：（hdu6396） 94](#_Toc526631824)

[7.2.2思路： 97](#_Toc526631825)

[8.leetcode 98](#_Toc526631826)

[8.1leetcode128 98](#_Toc526631827)

[8.1.1描述 98](#_Toc526631828)

[8.1.2code： 98](#_Toc526631829)

[8.1.3思想： 99](#_Toc526631830)

[9.博弈论 99](#_Toc526631831)

[9.1.1description 99](#_Toc526631832)

[9.1.2解题方法： 99](#_Toc526631833)

[9.1.3Code： 99](#_Toc526631834)

[10. 输入输出 100](#_Toc526631835)

[10.1输入（如果时间为1s，且输入数据>1e6必须加快输入） 100](#_Toc526631836)

[10.1.1Code：（不适合调试） 100](#_Toc526631837)

[10.2刷新缓冲区 101](#_Toc526631838)

[10.3输出（感觉没有printf快） 101](#_Toc526631839)

[DP 102](#_Toc526631840)

[11.1最长上升子序列 102](#_Toc526631841)

[11.1.1（从1~i）（O（n）） 102](#_Toc526631842)

[11.1.2（i~n）（O（nlogn）） 103](#_Toc526631843)

[11.2(poj3616) 105](#_Toc526631844)

[11.2.0 描述 105](#_Toc526631845)

[11.2.1code: 105](#_Toc526631846)

[11.2.2思路： 107](#_Toc526631847)

# 数据结构

## 1.1线段树

### 1.1.1描述：

区间修改和区间查询

### 1.1.2代码：

#include<stdio.h>

#include<string.h>

#include<vector>

using namespace std;

typedef int ll;

const int maxn=1e4\*5+2;

ll tree[maxn<<2];

ll lazy[maxn<<2];

ll array[maxn];

void up(ll index)

{

tree[index]=tree[index<<1]+tree[index<<1|1];

}

void build(ll l,ll r,ll index)

{

if(l==r)

{

tree[index]=array[l];

lazy[index]=0;

return ;

}

ll mid=(l+r)>>1;

build(l,mid,index<<1);

build(mid+1,r,index<<1|1);

up(index);

}

void down(ll index,ll l,ll r)

{

if(lazy[index]!=0)

{

ll mid=(l+r)>>1;

tree[index<<1]+=(mid-l+1)\*lazy[index];

tree[index<<1|1]+=(r-mid)\*lazy[index];

lazy[index<<1]+= lazy[index];

lazy[index<<1|1]+=lazy[index];

lazy[index]=0;

}

}

void updatenode(ll l,ll r,ll L,ll index,ll c)

{

if(l==r && L==l)

{

tree[index]+=c;

return;

}

ll mid=(l+r)>>1;

if(L<=mid)

updatenode(l,mid,L,index<<1,c);

else

updatenode(mid+1,r,L,index<<1|1,c);

up(index);

}

void update(ll l,ll r,ll L,ll R,ll index,ll c)

{

if(L<=l && R>=r)

{

tree[index]+=c\*(r-l+1);

lazy[index]+=c;

return ;

}

ll mid=(l+r)>>1;

down(index,l,r);

if(L<=mid)

update(l,mid,L,R,index<<1,c);

if(R>mid)

update(mid+1,r,L,R,index<<1|1,c);

up(index);

}

ll query(ll l,ll r,ll L,ll R,ll index)

{

if(L<=l && R>=r)

return tree[index];

ll mid=(l+r)>>1;

down(index,l,r);

ll ans=0;

if(L<=mid)

ans+=query(l,mid,L,R,index<<1);

if(R>mid)

ans+=query(mid+1,r,L,R,index<<1|1);

return ans;

}

char buf[10];

int main()

{

ll t=0,i=1;

//vector<int>ans;

scanf("%d",&t);

while(i<=t)

{

//ans.clear();

ll n=0;

scanf("%d",&n);

for(int j=1;j<=n;j++)

scanf("%d",&array[j]);

build(1,n,1);

printf("Case %d:\n",i++);

while(scanf("%s",buf))

{

if(buf[0]=='Q')

{

ll x=0,y=0;

scanf("%d %d",&x,&y);

printf("%d\n",query(1,n,x,y,1));

//ans.push\_back(query(1,n,x,y,1));

}

else if(buf[0]=='A')

{

ll x=0,y=0;

scanf("%d %d",&x,&y);

updatenode(1,n,x,1,y);

}

else if(buf[0]=='S')

{

ll x=0,y=0;

scanf("%d %d",&x,&y);

updatenode(1,n,x,1,0-y);

}

else if(buf[0]=='E')

{

break;

}

}

}

return 0;

}

### 1.1.3注意：

这里面还包含字符串的输入

### 1.1.4线段树+离散化（poj2528）

#### 1.1.4.1code

#include<stdio.h>

#include<cstring>

#include<vector>

#include<algorithm>

#include<map>

using namespace std;

const int maxn=2e4+10;

const int maxn1=1e4+10;

int tree[maxn<<2];

int lazy[maxn<<2];

int ans=0;

bool vis[maxn1];

void up(int index)

{

if(tree[index<<1]!=tree[index<<1|1])

tree[index]=0;

else

tree[index]=tree[index<<1];

}

void build(int l,int r,int index)

{

if(l==r)

{

tree[index]=0;

lazy[index]=0;

return ;

}

lazy[index]=0;

int mid=(l+r)>>1;

build(l,mid,index<<1);

build(mid+1,r,index<<1|1);

up(index);

}

void pushdown(int index,int l,int r)

{

if(lazy[index])

{

tree[index<<1]=lazy[index];

tree[index<<1|1]=lazy[index];

lazy[index<<1]=lazy[index];

lazy[index<<1|1]=lazy[index];

lazy[index]=0;

}

}

void update(int l,int r,int L,int R,int index,int v)

{

if(L<=l && R>=r)

{

tree[index]=v;

lazy[index]=v;

return ;

}

int mid=(l+r)>>1;

pushdown(index,l,r);

if(L<=mid)

update(l,mid,L,R,index<<1,v);

if(R>mid)

update(mid+1,r,L,R,index<<1|1,v);

up(index);

}

void query(int l,int r,int index)

{

if(tree[index]!=0)

{

if(vis[tree[index]]==0)return;

ans++;

vis[tree[index]]=0;

return;

}

if(l==r)return;

pushdown(index,l,r);

int mid=(l+r)>>1;

query(l,mid,index<<1);

query(mid+1,r,index<<1|1);

}

struct node

{

int x,y;

node(int x1=0,int y1=0)

{

x=x1;y=y1;

}

};

int main()

{

int c=0;

scanf("%d",&c);

while(c--)

{

int n=0;

scanf("%d",&n);

memset(vis,1,sizeof(vis));

vector<node>v;

vector<int>s;

int x,y;

for(int i=0;i<n;i++)

{

scanf("%d %d",&x,&y);

v.push\_back(node(x,y));

s.push\_back(x);

s.push\_back(y);

}

sort(s.begin(),s.end());

int rm=0;

map<int,int>mp;

for(int i=0;i<s.size();i++)

{

if(mp[s[i]]==0)

{

mp[s[i]]=++rm;

}

}

build(1,rm,1);

node tmp;

for(int i=0;i<v.size();i++)

{

tmp=v[i];

update(1,rm,mp[tmp.x],mp[tmp.y],1,i+1);

}

ans=0;

query(1,rm,1);

printf("%d\n",ans);

}

return 0;

}

#### 1.1.4.2思想

如果这道题不用离散化的话会MLE，将给定的线段根据端点进行排序，用map来记录每个端点的离散后的值，再套线段树即可。

### 1.1.5线段树求在给定区间中第一个大于某个数的位置

#### 1.1.5.1code

typedef long long ll;

const int maxn=1e5+20;

struct node

{

ll mmax;

int cur;

node(ll mx=0,int c=0)

{

mmax=mx;cur=c;

}

}tree[maxn<<2];

ll a[maxn];

void up(int index)

{

if(tree[index<<1].mmax>tree[index<<1|1].mmax)

{

tree[index].mmax=tree[index<<1].mmax;

tree[index].cur=tree[index<<1].cur;

}

else

{

tree[index].mmax=tree[index<<1|1].mmax;

tree[index].cur=tree[index<<1|1].cur;

}

}

void build(int l,int r,int index)

{

if(l==r)

{

tree[index].mmax=a[l];

tree[index].cur=l;

return ;

}

int mid=(l+r)>>1;

build(l,mid,index<<1);

build(mid+1,r,index<<1|1);

up(index);

}

node query(int l,int r,int L,int R,int index)

{

if(L<=l && R>=r)

{

return tree[index];

}

int mid=(l+r)>>1;

node ans1;

node ans2;

if(L<=mid)

ans1=query(l,mid,L,R,index<<1);

if(R>mid)

ans2=query(mid+1,r,L,R,index<<1|1);

if(ans1.mmax>ans2.mmax)

return ans1;

else

return ans2;

}

void FLS(int l,int r,int L,int R,ll c,int index,int &cur)

{

if(l==r)

{

if(tree[index].mmax>c)

cur=min(cur,tree[index].cur);

return ;

}

int mid=(l+r)>>1;

if(L<=l && R>=r)

{

if(tree[index<<1].mmax>c)

FLS(l,mid,L,R,c,index<<1,cur);

else if(tree[index<<1|1].mmax>c)

FLS(mid+1,r,L,R,c,index<<1|1,cur);

return ;

}

if(L<=mid)FLS(l,mid,L,R,c,index<<1,cur);

if(R>mid)FLS(mid+1,r,L,R,c,index<<1|1,cur);

return ;

}

int FirstLarge(int L,int R,ll c,int n)

{

int cur=INF;

FLS(1,n,L,R,c,1,cur);

return cur;

}

#### 1.1.5.2思想：

利用线段树可以实现不需要排序可以找到指定的数在给定区间内第一个大于他的数所在的位置，实现的函数是FLS（），接口是FirstLarge（），返回值是位置，如果没找到则返回INF。

时间复杂度是O（nlogn）；

该算法可应用于从当前位置i开始到n的最长上升子序列，时间复杂度为O（nlogn）。具体实现代码见11章。

## STLbitset

### 1.2.1使用

#include<bitset>

bitset<1000> bs; //1000位

function：

Bitset常用方法：

| **成员函数** | **函数功能** |
| --- | --- |
| bs.any() | 是否存在值为1的二进制位 |
| bs.none() | 是否不存在值为1的二进制位 或者说是否全部位为0 |
| bs.size() | 位长，也即是非模板参数值 |
| bs.count() | 值为1的个数 |
| bs.test(pos) | 测试pos处的二进制位是否为1 与0做或运算 |
| bs.set() | 全部位置1 |
| bs.set(pos) | pos位处的二进制位置1 与1做或运算 |
| bs.reset() | 全部位置0 |
| bs.reset(pos) | pos位处的二进制位置0 与0做或运算 |
| bs.flip() | 全部位逐位取反 |
| bs.flip(pos) | pos处的二进制位取反 |
| bs.to\_ulong() | 将二进制转换为unsigned long输出 |
| bs.to\_string() | 将二进制转换为字符串输出 |
| ~bs | 按位取反 效果等效为bs.flip() |
| os << b | 将二进制位输出到os流 小值在右，大值在左 |

## 1.3莫队算法

### 1.3.1Code：

*#include<iostream>*

*#include<algorithm>*

*#include<math.h>*

*using namespace std;*

*#define N 500002*

*int cnt[N]={0};*

*int \*a;*

*int n;*

*int k;*

*int ans=0;*

*int sq;*

*struct node*

*{*

*int l;*

*int r;*

*int id;*

*node(int l1=0,int r1=0,int i=0)*

*{*

*l=l1;*

*r=r1;*

*id=i;*

*}*

*};*

*bool cmp(node a,node b)*

*{*

*if(a.l/sq<b.l/sq)*

*return true;*

*else if(a.l/sq==b.l/sq)*

*{*

*if(a.r<=b.r)*

*return true;*

*else*

*return false;*

*}*

*else*

*return false;*

*}*

*void add(int x)*

*{*

*cnt[a[x]]++;*

*if(cnt[a[x]]==k)*

*ans++;*

*else if(cnt[a[x]]==k+1)*

*ans--;*

*}*

*void del(int x)*

*{*

*cnt[a[x]]--;*

*if(cnt[a[x]]==k)*

*ans++;*

*else if(cnt[a[x]]==k-1)*

*ans--;*

*}*

*int main()*

*{*

*int m=0;*

*k=1;*

*while(cin>>n>>m)*

*{*

*sq=sqrt((double)n);*

*ans=0;*

*a=new int[n];*

*node \*qury=new node[m];*

*int \*answer=new int[m];*

*for(int i=0;i<N;i++)*

*cnt[i]=0;*

*for(int i=0;i<n;i++)*

*cin>>a[i];*

*int l=0,r=0;*

*for(int i=0;i<m;i++)*

*{*

*cin>>l>>r;*

*//qury[i]=new node(l-1,r-1,i);*

*qury[i].id=i;qury[i].l=l-1;qury[i].r=r-1;*

*}*

*sort(qury,qury+m,cmp);*

*int left=0,right=0;*

*add(0);*

*for(int i=0;i<m;i++)*

*{*

*node tmp=qury[i];*

*while(left<tmp.l){del(left);left++;}*

*while(left>tmp.l){left--;add(left);}*

*while(right<tmp.r){right++;add(right);}*

*while(right>tmp.r){del(right);right--;}*

*answer[tmp.id]=ans;*

*}*

*for(int i=0;i<m;i++)*

*cout<<answer[i]<<endl;*

*}*

*return 0;*

*}*

### 1.3.2 原理思想

*莫队算法的时间复杂度O(sqrt(序列长度)\*（序列长度+询问的规模）)*

思想：莫队算法将整个序列分成sqrt（n）块，然后莫队algorithm也是一个离线算法，需要事先知道所有的输入数据。然后根据前面分好的块进行分组，在同一块中又按照r进行排序。这样做的目的是让前面的答案对后面都有贡献。然后求解的方式是暴力的方式，用一个数组cnt来维护出现的值出现的个数。设置两个指针left，和right。然后每次询问都去移动left和right指针。

莫队算法分析：

时间复杂度：由于将序列分为sqrt（n），right每次在块中都会移动O（n），所以一共有sqrt（n）块，所以right的时间复杂度为O（sqrt（n）\*n）。

Left每次跨越块是sqrt（n），由于有m次询问，所以left的时间复杂度是O（sqrt（n）\*m）

加法原理：莫队算法：O（sqrt（n）\*（m+n））

## 1.4划分树

### 1.4.1code：

#include<stdio.h>

#include<iostream>

#include<algorithm>

using namespace std;

#define N 100002

int tree[20][N];

int num[20][N];

int sorted[N];

int a[N];

void build(int l,int r,int c)

{

if(l==r)

{

return;

}

int mid=(l+r)>>1;

int pro=sorted[mid];

int issame=mid-l+1;

for(int i=l;i<=r;i++)

if(tree[c][i]<pro)

issame--;

int ls=l,rs=mid+1;

for(int i=l;i<=r;i++)

{

if(i==l)

num[c][i]=0;

else

num[c][i]=num[c][i-1];

if(tree[c][i]<pro || (tree[c][i]==pro && issame!=0))

{

tree[c+1][ls++]=tree[c][i];

num[c][i]++;

if(tree[c][i]==pro)

issame--;

}

else

{

tree[c+1][rs++]=tree[c][i];

}

}

build(l,mid,c+1);

build(mid+1,r,c+1);

}

int query(int s,int e,int l,int r,int c,int k)

{

if(s==e)

return tree[c][s];

int ll=0;

if(l>s)

ll=num[c][l-1];

int rl=num[c][r];

int mid=(s+e)>>1;

if(k<=rl-ll)

return query(s,mid,s+ll,s+rl-1,c+1,k);

else

return query(mid+1,e,mid+1+(l-s)-ll,mid+1+(l-s)-ll+(r-l+1)-(rl-ll)-1,c+1,k-(rl-ll));

}

int main()

{

int x=0;

scanf("%d",&x);

while(x--)

{

int n=0,m=0;

scanf("%d %d",&n,&m);

for(int i=1;i<=n;i++)

{

scanf("%d",&tree[0][i]);

sorted[i]=tree[0][i];

}

sort(sorted+1,sorted+n+1);

build(1,n,0);

int ans=0;

int l,r,k;

for(int i=0;i<m;i++)

{

scanf("%d %d %d",&l,&r,&k);

//ans=query(1,n,l,r,0,r-l+1-k);

ans=query(1,n,l,r,0,k);

printf("%d\n",ans);

}

}

return 0;

}

### 1.4.2原理思想

分两步：

1. 建树

建树的过程比较简单，对于区间[l,r]，首先通过对原数组的排序找到这个区间的中位数a[mid]，小于a[mid]的数划入他的左子树[l,mid-1]，大于它的划入右子树[mid,r]。同时，对于第i个数，记录在[l,i]区间内有多少数被划入左子树。最后，对它的左子树区间[l,mid-1]和右子树区间[mid,r]递归的继续建树就可以了。

建树的时候要注意对于被分到同一子树的元素，元素间的相对位置不能改变。

查找

查找的过程中主要问题就是确定将要查找的区间。这个问题有些麻烦。

先看一下查找过程tree\_find.他的定义如下：

查找深度为h，在大区间[st,ed]中找小区间[s,e]中的第k元素。

再看看他是如何工作的。我们的想法是，先判断[s,e]中第k元素在[st,ed]的哪个子树中，然后找出对应的小区间和k，递归的进行查找，直到小区间的s=e为止。

那如何解决这个问题呢？这时候前面记录的进入左子树的元素个数就派上用场了。通过之前的记录可以知道，在区间[st,s-1]中有el[h,s-1]进入左子树，记它为l同理区间[st,e]中有el[h,e]个数进去左子树，记它为r。所以，我们知道区间小区间[s,e]中有(r-l)个数进入左子树。那么如果(r-l)>=k，那么就在左子树中继续查找，否则就在右子树中继续查找。

接着解决查找的小区间的问题。如果接下来要查找的是左子树，那么小区间应该是[st+([st,s-1]区间进入左子树的个数),st+([st,e]区间内进入左子树的个数)-1]，即区间[st+l,st+r-1]。显然，这里k不用变。

如果接下来要查找的是右子树，那么小区间应该是[mid+([st,s-1]区间中进入右子树的个数),mid+([st,e]区间进入右子树的个数)-1]。即区间[mid+(s-st-l),mid+(e-st-r)]。显然，这里k要减去区间里已经进入左子树的个数，即k变为k-(r-l)。于是递归继续查找直到s=e即可。

划分树我觉得算法可行是因为在每一层中每个节点中的数相对位置不变。

## 1.5主席树

### 1.5.1code

#include<stdio.h>

#include<algorithm>

#include<iostream>

using namespace std;

typedef int ll;

const int maxn =1e5+10;

struct node

{

ll x;

ll L;

ll R;

node(ll x1=0,ll l=0,ll r=0)

{

x=x1;L=l;R=r;

}

}tree[maxn\*20];//error :\*20,not <<2

ll cur;

void update(ll num,ll l,ll r,ll &rt)

{

tree[cur++]=tree[rt];

rt=cur-1;

tree[rt].x++;

if(l==r)return ;

ll mid=(l+r)>>1;

if(num<=mid)

update(num,l,mid,tree[rt].L);

else

update(num,mid+1,r,tree[rt].R);

}

ll query(ll k,ll l,ll r,ll m\_l, ll m\_r)

{

if(l==r)return l;

ll d=tree[tree[m\_r].L].x-tree[tree[m\_l].L].x;

ll mid=(l+r)>>1;

if(k<=d)

return query(k,l,mid,tree[m\_l].L,tree[m\_r].L);

else

return query(k-d,mid+1,r,tree[m\_l].R,tree[m\_r].R);

}

ll a[maxn];

struct li

{

ll sum;

ll id;

li(ll s=0,ll id1=0)

{

sum=s;

id=id1;

}

}value[maxn];

bool cmp(li a,li b)

{

return a.sum<b.sum;

}

ll rank[maxn];

ll root[maxn];

int main()

{

ll n,m;

cin>>n>>m;

for(int i=1;i<=n;i++)

{

cin>>value[i].sum;

value[i].id=i;

}

sort(value+1,value+1+n,cmp);

for(int i=1;i<=n;i++)

rank[value[i].id]=i;

cur=1;

root[0]=0;

for(int i=1;i<=n;i++)

{

root[i]=root[i-1];

update(rank[i],1,n,root[i]);

}

ll i,j,k;

for(int t=0;t<m;t++)

{

scanf("%d %d %d",&i,&j,&k);

printf("%d\n",value[query(k,1,n,root[i-1],root[j])].sum);

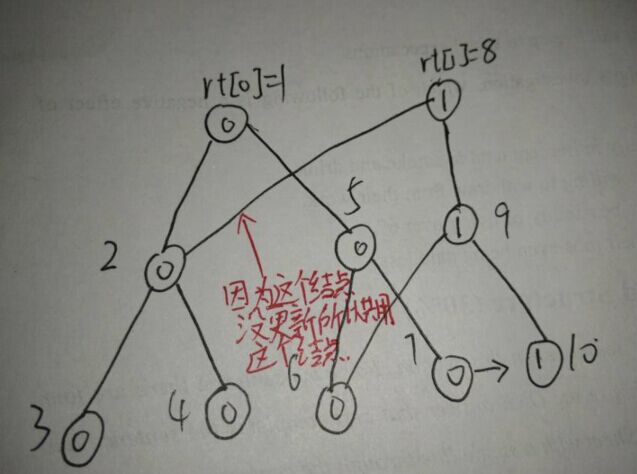
}

return 0;

}

### 1.5.2思想：

主席树其实是由多颗线段树组成，但是每一次更新都是在上一次的基础上进行，所以每一次更新只修改其中一条链。



主席树需要离散化，相当于将从【1，i】的排序存放在线段树中。

查询的思路：查找在[l,r]中第k小的数，那么需要知道在[l,r]的左孩子中有多少个数，如果左孩子的个数小于k，则在右孩子中，否则在左孩子中。计算[l,r]用[1,r]-[1,l-1]。

### 1.5.3主席树求区间多少种数

#### 1.5.3.1code

#include<stdio.h>

#include<map>

#include<cstring>

using namespace std;

const int maxn=3e4+10;

struct node

{

int l,r,sum;

node(int l1=0,int r1=0,int s1=0)

{

l=l1;r=r1;sum=s1;

}

}tree[maxn\*40];

int cnt=1;

int root[maxn];

int a[maxn];

void update(int pos,int l,int r,int v,int &rt)

{

tree[cnt++]=tree[rt];

rt=cnt-1;

tree[rt].sum+=v;

if(l==r)return ;

int mid=(l+r)>>1;

if(pos<=mid)

update(pos,l,mid,v,tree[rt].l);

else

update(pos,mid+1,r,v,tree[rt].r);

}

int query(int pos,int l,int r,int rt)// acroding left pos

{

if(l==r)return tree[rt].sum;

int mid=(l+r)>>1;

if(pos<=mid)

return tree[tree[rt].r].sum+query(pos,l,mid,tree[rt].l);

else

return query(pos,mid+1,r,tree[rt].r);

}

int main()

{

int n,m;

scanf("%d",&n);

map<int,int>mp;

for(int i=1;i<=n;i++){

scanf("%d",&a[i]);

}

root[0]=0;

for(int i=1;i<=n;i++)

{

root[i]=root[i-1];

if(mp[a[i]]==0)

update(i,1,n,1,root[i]);

else

{

update(mp[a[i]],1,n,-1,root[i]);

update(i,1,n,1,root[i]);

}

mp[a[i]]=i;

}

scanf("%d",&m);

int x,y;

for(int i=1;i<=m;i++)

{

scanf("%d %d",&x,&y);

printf("%d\n",query(x,1,n,root[y]));

}

return 0;

}

#### 1.5.3.2思想

先讲怎么查询。由于在建树的时候每一次记录都是该数字最后一次出现的位置。所以给定[l,r]那么就从r的结点开始查找。以l为依据，如果l<=mid，那么就加上右孩子再递归到左孩子，如果l>mid，直接递归到右孩子。

如何建树?用一个map来记录该数字出现的上一次位置在哪?如果出现过，则将上一次位置-1，然后再在当前位置+1，一定要保证记录的位置一定是最后出现的位置。

## 1.6树状数组（维护前缀和）

### 1.6.1code

例题hdu1166

#include<stdio.h>

#include<cstring>

using namespace std;

const int maxn=5e4+10;

int c[maxn];

int a[maxn];

char buf[10];

int lowbit(int x)

{

return x&(-x);

}

void add(int index,int v,int n)

{

while(index<=n)

{

c[index]+=v;

index+=lowbit(index);

}

}

int sum(int x)

{

int ans=0;

while(x)

{

ans+=c[x];

x-=lowbit(x);

}

return ans;

}

void init(int n)

{

memset(c,0,sizeof(c));

for(int i=1;i<=n;i++)

{

int x=lowbit(i);

for(int j=i-x+1;j<=i;j++)

c[i]+=a[j];

}

}

int main()

{

int t,n;

scanf("%d",&t);

int ca=1;

while(t--)

{

scanf("%d",&n);

for(int i=1;i<=n;i++)

scanf("%d",&a[i]);

init(n);

printf("Case %d:\n",ca++);

while(true)

{

scanf("%s",buf);

int x,y;

if(buf[0]=='Q')

{

scanf("%d %d",&x,&y);

printf("%d\n",sum(y)-sum(x-1));

}

else if(buf[0]=='A')

{

scanf("%d %d",&x,&y);

add(x,y,n);

}

else if(buf[0]=='S')

{

scanf("%d %d",&x,&y);

add(x,-y,n);

}

else

break;

}

}

return 0;

}

### 1.6.2思想

在求前缀和问题时，我们开了大小为n的数组，但是如果需要修改，我们不得不从l修改到r，在极端情况下从1修改到n，这样效率极低。树状数组利用二进制的性质，同样只开一个大小为n的数组来记录前缀和，但是每一个元素都不是普通的记录前缀和。

C1=a1 C2=a1+a2 C3=a3 规律是找二进制的最靠右边的1，假设一个数x，起最靠右边的1的十进制是y，那么他所记录的位置是从x-y+1到x。

在求和时求的是从1~x。在求区间和时将边界s[r]-s[l-1].求和时是从孩子结点开始往左找。修改时从孩子结点开始往右找。x&（-x）可以找到一个二进制数中最靠右边的1的十进制。

## 1.7树上最长路（O(V)）

### 1.7.1code

#include<stdio.h>

#include<queue>

#include<vector>

#include<cstring>

using namespace std;

struct node

{

int v;

int mx;

node(int v1=0,int mx1=0)

{

v=v1;mx=mx1;

}

};

const int maxn=1e6+10;

vector<int>v[maxn];

bool vis[maxn];

void dfs(int x,node &ans,int mx)

{

if(vis[x])return;

vis[x]=1;

if(ans.mx<mx)

{

ans.mx=mx;

ans.v=x;

}

for(int i=0;i<v[x].size();i++)

{

dfs(v[x][i],ans,mx+1);

}

return;

}

int main()

{

int n;

scanf("%d",&n);

int x,y;

for(int i=0;i<n-1;i++)

{

scanf("%d %d",&x,&y);

v[x].push\_back(y);

v[y].push\_back(x);

}

node ans;

memset(vis,0,sizeof(vis));

dfs(1,ans,1);

int v\_a=ans.v;

ans.v=0;ans.mx=0;

memset(vis,0,sizeof(vis));

dfs(v\_a,ans,1);

printf("%d\n",ans.mx);

return 0;

}

### 1.7.2思路：

树上存在最远的两个点(b,e)，那么距离任意点x最远的点一定是b或者e。   
因此可以任选一个点做dfs找到最远的点b，再从b做一次dfs找到e，最远距离便算出来了。两遍dfs

# 2数论

## 2.1最大公约数

//gcd最大公约数 a>b

int gcd(int a,int b)

{

int c=0;

while(b!=0)

{

c=a%b;

a=b;b=c;

}

return a;

}

## 2.2最小公倍数

//最小公倍数 a>b 最小公倍数=两数的乘积/最大公约数

int MutilMin(int a,int b)

{

int ans=gcd(a,b);

a=a/ans;

return a\*b;

}

## 2.3快速幂

#include<iostream>

using namespace std;

typedef long long ll;

#define MD 1000000007

ll m\_pow(ll n,ll k)

{

ll base=n;

ll ans=1;

while(k!=0)

{

if(k&1!=0)

ans=ans\*base;

base\*=base;

k>>=1;

}

return ans;

}

int main()

{

ll n=0,k=0;

int x=1;

while(cin>>n>>k)

{

ll ans=m\_pow(n,k);

cout<<"Case #"<<x++<<": "<<ans<<endl;

}

return 0;

}

## 2.4求一个正整数的所有约数和

Code：

#include<iostream>

#include<map>

#include<vector>

#include<math.h>

using namespace std;

typedef long long ll;

vector<ll>yue;

map<ll,ll>p;

void add(ll x)

{

if(p.find(x)==p.end())

p[x]=1;

else

p[x]++;

}

void YueShu(ll a)

{

for(ll i=2;i<=a;i++)

{

if(a%i==0)

{

add(i);

a=a/i;

i=1;

}

}

}

ll m\_pow(ll a,ll x)

{

ll base=a;

ll ans=1;

while(x)

{

if(x&1!=0)

ans\*=base;

base\*=base;

x>>=1;

}

return ans;

}

int main()

{

ll a=1e10;

//cin>>a;

ll ans=1;

YueShu(a);

map<ll,ll>::iterator iter;

for(iter=p.begin();iter!=p.end();iter++)

{

pair<ll,ll>ax=\*iter;

//ll tmp=m\_pow(ax.first,ax.second+1);

ll tmp=pow(ax.first,ax.second+1);//system function is faster than myself write

ans\*=(1-tmp)/(1-ax.first);

}

cout<<ans<<endl;

return 0;

}

## 2.5求a在mod下的逆元（1/a）

### 2.5.1code

const int mod=1e9+7;

ll m\_pow(ll n,ll k)

{

ll base=n;

ll ans=1;

while(k!=0)

{

if(k&1!=0)

{

ans=(ans\*base)%mod;

}

base=(base\*base)%mod;

k>>=1;

}

return ans;

}

m\_pow(z,mod-2);

### 2.5.2推导

**（1）.费马小定理**

　　　　当p为质数时，对于任意整数a，满足ap-a是p的整数倍

　　　　在mod p意义下可以表示为

**ap-a≡0（mod p）**

**即为ap≡a（mod p）**

**所以ap-1≡1（mod p）**

**即为a\*ap-2≡1（mod p）**

　　　　所以ap-2即为a在 mod p 意义下的逆元

## 2.6费马大定理

它断言当整数n >2时，关于x, y, z的方程 x^n + y^n = z^n 没有正整数解。

## 2.7知道其中一条直角边的直角三角形构造另外两条边

分奇偶性：

奇：2×n+1 2\*n^2+2\*n 2\*n^2+2\*n+1

偶：2×n+1 2\*n^2+2\*n 2\*n^2+2\*n+1

## 2.8BM模板

#include <cstdio>

#include <cstring>

#include <cmath>

#include <algorithm>

#include <vector>

#include <string>

#include <set>

#include <cassert>

using namespace std;

#define rep(i,a,n) for (ll i=a;i<n;i++)

#define per(i,a,n) for (ll i=n-1;i>=a;i--)

#define pb push\_back

#define mp make\_pair

#define all(x) (x).begin(),(x).end()

#define fi first

#define se second

#define SZ(x) ((ll)(x).size())

typedef long long ll;

typedef vector<ll> VI;

typedef pair<ll, ll> PII;

const ll mod = 1000000007;

ll powmod(ll a, ll b) { ll res = 1; a %= mod; assert(b >= 0); for (; b; b >>= 1) { if (b & 1)res = res\*a%mod; a = a\*a%mod; }return res; }

ll \_, n;

namespace linear\_seq {

const ll N = 10010;

ll res[N], base[N], \_c[N], \_md[N];

vector<ll> Md;

void mul(ll \*a, ll \*b, ll k) {

rep(i, 0, k + k) \_c[i] = 0;

rep(i, 0, k) if (a[i]) rep(j, 0, k) \_c[i + j] = (\_c[i + j] + a[i] \* b[j]) % mod;

for (ll i = k + k - 1; i >= k; i--) if (\_c[i])

rep(j, 0, SZ(Md)) \_c[i - k + Md[j]] = (\_c[i - k + Md[j]] - \_c[i] \* \_md[Md[j]]) % mod;

rep(i, 0, k) a[i] = \_c[i];

}

ll solve(ll n, VI a, VI b) {

ll ans = 0, pnt = 0;

ll k = SZ(a);

assert(SZ(a) == SZ(b));

rep(i, 0, k) \_md[k - 1 - i] = -a[i]; \_md[k] = 1;

Md.clear();

rep(i, 0, k) if (\_md[i] != 0) Md.push\_back(i);

rep(i, 0, k) res[i] = base[i] = 0;

res[0] = 1;

while ((1ll << pnt) <= n) pnt++;

for (ll p = pnt; p >= 0; p--) {

mul(res, res, k);

if ((n >> p) & 1) {

for (ll i = k - 1; i >= 0; i--) res[i + 1] = res[i]; res[0] = 0;

rep(j, 0, SZ(Md)) res[Md[j]] = (res[Md[j]] - res[k] \* \_md[Md[j]]) % mod;

}

}

rep(i, 0, k) ans = (ans + res[i] \* b[i]) % mod;

if (ans < 0) ans += mod;

return ans;

}

VI BM(VI s) {

VI C(1, 1), B(1, 1);

ll L = 0, m = 1, b = 1;

rep(n, 0, SZ(s)) {

ll d = 0;

rep(i, 0, L + 1) d = (d + (ll)C[i] \* s[n - i]) % mod;

if (d == 0) ++m;

else if (2 \* L <= n) {

VI T = C;

ll c = mod - d\*powmod(b, mod - 2) % mod;

while (SZ(C) < SZ(B) + m) C.pb(0);

rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c\*B[i]) % mod;

L = n + 1 - L; B = T; b = d; m = 1;

}

else {

ll c = mod - d\*powmod(b, mod - 2) % mod;

while (SZ(C) < SZ(B) + m) C.pb(0);

rep(i, 0, SZ(B)) C[i + m] = (C[i + m] + c\*B[i]) % mod;

++m;

}

}

return C;

}

ll gao(VI a, ll n) {

VI c = BM(a);

c.erase(c.begin());

rep(i, 0, SZ(c)) c[i] = (mod - c[i]) % mod;

return solve(n, c, VI(a.begin(), a.begin() + SZ(c)));

}

};

int main() {

for (scanf("%I64d", &\_); \_; \_--)

{

scanf("%I64d", &n);

VI vec;

vec.push\_back(31);

vec.push\_back(197);

vec.push\_back(1255);

vec.push\_back(7997);

vec.push\_back(50959);

vec.push\_back(324725);

printf("%I64d\n", linear\_seq::gao(vec, n - 1));

}

}

## 2.9欧拉函数（欧拉表）

求给定n与n互质且比n小的数的个数。

### 2.9.1code：

int phi[maxn];

void phi\_table(int n)

{

for(int i=2;i<=n;i++)phi[i]=0;

phi[1]=1;

for(int i=2;i<=n;i++)if(!phi[i])

for(int j=i;j<=n;j+=i)

{

if(!phi[j])phi[j]=j;

phi[j]=phi[j]/i\*(i-1);

}

}

### 2.9.2例题（UVA11426）

求所有gcd(i,j)的和。（1<=i<j<=n）

代码：

#include<iostream>

using namespace std;

const int maxn=4e6+2;

typedef long long ll;

int phi[maxn];

void phi\_table(int n)

{

for(int i=2;i<=n;i++)phi[i]=0;

phi[1]=1;

for(int i=2;i<=n;i++)if(!phi[i])

for(int j=i;j<=n;j+=i)

{

if(!phi[j])phi[j]=j;

phi[j]=phi[j]/i\*(i-1);

}

}

ll f[maxn];

ll s[maxn];

int main()

{

phi\_table(maxn);

for(int i=1;i<=maxn;i++)

{

for(int n=i\*2;n<=maxn;n+=i)

f[n]+=i\*phi[n/i];

}

int n=0;

while(cin>>n)

{

if(n==0)break;

s[1]=0;

for(int i=2;i<=n;i++)

{

s[i]=s[i-1]+f[i];

}

cout<<s[n]<<endl;

}

return 0;

}

思路：

假设f[n]=gcd(1,n)+gcd(2,n)+...+gcd(n-1,n)

那么所求答案为f[1]+f[2]+f[3]+...f[n];

现在变为求解f[n].

Gcd(x,n)=i,gcd(x/i,n/i)=1

f[n]中的答案为i\*g[n,i]，i为n的约数，g[n,i]为i为n的约数的个数。

g[n,i]可用欧拉函数求得。

# 图论

## 3.1拓扑排序

### 3.1.1kahn算法

#include<stdio.h>

#include<vector>

#include<stack>

#include<cstring>

using namespace std;

const int maxn=101;

vector<int>v[maxn];

vector<int>ans;

int d[maxn];

bool kahn(int n)

{

int cur=-1;

stack<int>s;

for(int i=1;i<=n;i++)if(d[i]==0)s.push(i),ans.push\_back(i);

while(!s.empty())

{

cur=s.top();s.pop();

int sz=v[cur].size();

for(int i=0;i<sz;i++)

{

int tmp=v[cur][i];

d[tmp]--;

if(d[tmp]==0)

{

s.push(tmp);

ans.push\_back(tmp);

}

}

}

return ans.size()==n;

}

void init(int n)

{

memset(d,0,sizeof(d));

ans.clear();

int tmp=-1;

for(int i=1;i<=n;i++)

{

while(true)

{

scanf("%d",&tmp);

if(tmp==0)break;

v[i].push\_back(tmp);

d[tmp]++;

}

}

}

int main()

{

int n=0;

while(~scanf("%d",&n))

{

init(n);

kahn(n);

for(int i=0;i<ans.size();i++)

printf("%d ",ans[i]);

printf("\n");

}

}

### 3.1.2原理思想

拓扑排序是解决在一个图中查看节点排列顺序的方法。Kahn算法的思想是优先选择入度为0的节点，将其加入集合中，然后每次从集合中选出尚未选中的节点，将该节点删去，然后与他相连的节点入度都会减一，再看这些节点会不会有出现入度为0的节点，如果有将其加入解集中。

## 3.2最小生成树

### 3.2.1code：

#include<iostream>

#include<algorithm>

#include<vector>

using namespace std;

int boss[100006];

int find(int x)

{

if(boss[x]==x)

return x;

else

return boss[x]=find(boss[x]);//避免超时

}

void Union(int x,int y)

{

int tmp\_x=find(x);

int tmp\_y=find(y);

boss[tmp\_x]=tmp\_y;

}

struct Node

{

int x;

int y;

int w;

Node(int x1=0,int y1=0,int w1=0)

{

x=x1;

y=y1;

w=w1;

}

};

Node edge[1000001];

bool cmp(Node &a,Node &b)

{

if(a.w<b.w)

{

return true;

}

else

{

return false;

}

}

int main()

{

int n=0;

int m=0;

cin>>n>>m;

int ans=0;

for(int i=0;i<m;i++)

{

cin>>edge[i].x>>edge[i].y>>edge[i].w;

}

sort(edge,edge+m,cmp);

for(int i=1;i<=n;i++)

boss[i]=i;

int num=0;

for(int i=0;i<m && num<n-1;i++)

{

int boss\_x=find(edge[i].x);

int boss\_y=find(edge[i].y);

if(boss\_x!=boss\_y)

{

Union(boss\_x,boss\_y);

ans+=edge[i].w;

num++;

}

}

cout<<ans<<endl;//输出最小花费

return 0;

}

### 3.2.2思想，原理

最小生成树算法解决在一个复杂图中，构造一个简单图，n个节点，只有n-1条边。先将边排序，每次从边集中选择权值较小的边。然后判断两个端点是否属于同一个集合（最开始有n个集合），然后使用并查集将两个点所属的集合加入到另一个中。

总的思想就是找到n-1条权值最小的边并且可以将所有点都变成在同一个集合中。

### 3.2.3次小生成树

#### 3.2.3.1code：

#include<stdio.h>

#include<cstring>

#include<vector>

#include<algorithm>

using namespace std;

typedef long long ll;

const int maxn=105;

const int maxm=205;

struct edge

{

ll c;

int s;

int e;

edge(ll c1=0,int s1=0,int e1=0){c=c1;s=s1;e=e1;}

};

edge ee[maxm];

bool cmp(edge &a,edge &b)

{

return a.c<b.c;

}

int boss[maxn];

int find(int x){return x==boss[x]? x:x=find(boss[x]);}

void Union(int dx,int dy){boss[dx]=dy;}

int record[maxn];

#define INF 0Xffffffff

void init(int n)

{

for(int i=1;i<=n;i++)boss[i]=i;

}

void solve(int n,int m)

{

sort(ee,ee+m,cmp);

int k=0;ll ans1=0;

for(int i=0;i<m;i++)

{

edge &tmp=ee[i];

int x=find(tmp.s);

int y=find(tmp.e);

if(x!=y)

{

Union(x,y);

ans1+=tmp.c;

record[k]=i;

k++;

}

if(k==n-1)break;

}

if(k!=n-1)

{

printf("No way\n");

return ;

}

ll ans2=INF;

for(int i=0;i<k;i++)

{

init(n);ll ans\_tmp=0;

int kk=0;

for(int j=0;j<m;j++)

{

if(record[i]==j)continue;

edge &tmp=ee[j];

int x=find(tmp.s);

int y=find(tmp.e);

if(x!=y)

{

Union(x,y);

ans\_tmp+=tmp.c;

kk++;

}

if(kk==n-1)break;

}

if(kk==n-1)

{

ans2=min(ans2,ans\_tmp);

}

}

if(ans2!=INF){printf("%lld\n",ans2);}

else printf("No second way\n");

}

int main()

{

int t=0;

scanf("%d",&t);

int tt=1;

while(t--)

{

int s,t;ll c;

int n,m;

scanf("%d %d",&n,&m);

init(n);

for(int i=0;i<m;i++)

{

scanf("%d %d %lld",&s,&t,&c);

ee[i]=edge(c,s,t);

}

printf("Case #%d : ",tt);

tt++;

solve(n,m);

}

return 0;

}

#### 3.2.3.2 思想：

该题是一道带有重边的求次小生成树的题，解题思路：先求一遍最小生成树，然后枚举最小生成树中的每一条边删除，然后重新构建一颗最小生成树集合，该集合中权重最小的即为次小生成树的权重。

## 3.3最短路算法

### 3.3.1BellMan\_Ford

#### 3.3.1.1code:

#include<vector>

#include<stdio.h>

using namespace std;

#define INF 0xfffff

typedef long long ll;

const int maxn=1000;

const int maxm=1000;

ll dis[maxn];

struct edge

{

ll x;

ll y;

ll w;

edge(ll x1=0,ll y1=0,ll w1=0)

{

x=x1;y=y1;w=w1;

}

}e[maxm];

bool BELLMANFORD(int n,int m)

{

for(int i=1;i<=n-1;i++)

for(int i=0;i<m;i++)

if(dis[e[i].y]>dis[e[i].x]+e[i].w)

dis[e[i].y]=dis[e[i].x]+e[i].w;

for(int i=0;i<m;i++)

if(dis[e[i].y]>dis[e[i].x]+e[i].w)

return false;//you fu huan

return true;

}

int main()

{

int n=0,m=0;

while(~scanf("%d %d",&n,&m))

{

int x,y,c;

for(int i=0;i<m;i++)

{

scanf("%d %d %d",&x,&y,&c);

e[i]=edge(x,y,c);

}

for(int i=1;i<=n;i++)

dis[i]=INF;

dis[1]=0;

BELLMANFORD(n,m);

for(int i=2;i<=n;i++)

printf("%d ",dis[i]);

printf("\n");

}

return 0;

}

#### 3.3.1.2思想，原理

时间复杂度O(VE)。贝尔曼福德算法可以检查是否有带有负权值的环路出现。贝算法的思想是对每一条边进行松弛，直至收敛，或者超过n-1次。在一个没有负权值的环路中，最短路最长的是由n-1条边所组成的最短路。所以每一次循环都是在找寻在最短路径中的位置。

### 3.3.2Floyed

#### 3.3.2.1code

#include<iostream>

#define INF 0xfffffff

using namespace std;

int main()

{

int n=0,m=0;

while(cin>>n>>m)

{

if(n==0 && m==0)return 0;

int \*\*map=new int\*[n+1];

int ans=0;

for(int i=1;i<=n;i++)

{

map[i]=new int [n+1];

for(int j=1;j<=n;j++)

{

map[i][j]=INF;

}

map[i][i]=0;

}

for(int i=0;i<m;i++)

{

int a=0,b=0,cost=0;

cin>>a>>b>>cost;

map[a][b]=map[b][a]=cost;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*floyed algorithm

for(int k=1;k<=n;k++)

{

for(int i=1;i<=n;i++)

{

for(int j=1;j<=n;j++)

{

if(map[i][j]>map[i][k]+map[k][j])

{

map[i][j]=map[i][k]+map[k][j];

}

}

}

}

ans=map[1][n];

cout<<ans<<endl;

}

return 0;

}

#### 3.3.2.2思想原理

时间复杂度O(N3 ),可以计算多源最短路，以k为中间节点，进行松弛操作

### 3.3.3Dijstra

#### 3.3.3.1code：

#include<queue>

#include<iostream>

#include<vector>

using namespace std;

#define INF 0xffffff

struct node

{

int x;

int cost;

node(int x1=0,int cost1=0)

{

x=x1;

cost=cost1;

}

node& operator=(node& a)

{

x=a.x;

cost=a.cost;

return \*this;

}

};

bool operator<(const node &a,const node& b)

{

if(a.cost>b.cost)

return true;

else

return false;

}

int main()

{

int n=0,m=0;

while(cin>>n>>m)

{

if(n==0 && m==0)return 0;

int \*\*map=new int\*[n+1];

bool \*vis=new bool[n+1];

int \*pre=new int [n+1];

int \*dis=new int [n+1];

int ans=0;

for(int i=1;i<=n;i++)

{

map[i]=new int [n+1];

vis[i]=1;

pre[i]=-1;

dis[i]=INF;

for(int j=1;j<=n;j++)

{

map[i][j]=INF;

}

}

for(int i=0;i<m;i++)

{

int a=0,b=0,cost=0;

cin>>a>>b>>cost;

map[a][b]=map[b][a]=cost;

}

dis[1]=0;

priority\_queue<node>que;

que.push(node(1,0));

int count=0;

while(!que.empty()&& count<n)

{

node tmp=que.top();

que.pop();

if(!vis[tmp.x])continue;

vis[tmp.x]=0;

count++;

for(int i=1;i<=n;i++)

{

if(map[tmp.x][i]!=INF)

{

if(vis[i] && dis[i]>dis[tmp.x]+map[tmp.x][i])

{

dis[i]=dis[tmp.x]+map[tmp.x][i];

que.push(node(i,dis[i]));

pre[i]=tmp.x;

}

}

}

}

cout<<dis[n]<<endl;

}

return 0;

}

#### 3.3.3.2思想原理

不能够有负边，时间复杂度O(（n+m）logn)。每次都从集合中选出距离最小的顶点加入到解集中，并且进行松弛操作。

### 3.3.4 SPFA

#### 3.3.4.1code

#include<stdio.h>

#include<vector>

#include<queue>

#include<cstring>

#include<algorithm>

using namespace std;

#define INF 0xffffff

const int maxn = 110;

const int maxm=1e4+10;

int map[maxn][maxn];

vector<int>v[maxn];

int dis[maxn];

bool vis[maxn];

struct node

{

int dis;

int cur;

node(int d=0,int c=0)

{

dis =d;

cur =c;

}

};

int main()

{

int n,m;

while(~scanf("%d %d",&n,&m))

{

if(n==0 && m==0)break;

int a,b,c;

memset(map,-1,sizeof(map));

memset(vis,0,sizeof(vis));

for(int i=0;i<=n;i++)dis[i]=INF,v[i].clear();

for(int i=0;i<m;i++)

{

scanf("%d %d %d",&a,&b,&c);

if(map[a][b]!=-1)

{

map[a][b]=map[b][a]=min(map[a][b],c);

}

else

{

map[a][b]=map[b][a]=c;

v[a].push\_back(b);

v[b].push\_back(a);

}

}

queue<node>q;

dis[1]=0;

q.push(node(0,1));

vis[1]=1;

node tmp;

while(!q.empty())

{

tmp=q.front();q.pop();

vis[tmp.cur]=0;

for(int i=0;i<v[tmp.cur].size();i++)

{

int u=v[tmp.cur][i];

if(dis[u]>map[tmp.cur][u]+dis[tmp.cur])

{

dis[u]=map[tmp.cur][u]+dis[tmp.cur];

if(!vis[u]){q.push(node(dis[u],u));vis[u]=1;}

}

}

}

printf("%d\n",dis[n]);

}

return 0;

}

#### 3.3.4.2思想

设立一个先进先出的队列用来保存待优化的结点，优化时每次取出队首结点u，并且用u点当前的最短路径估计值对离开u点所指向的结点v进行松弛操作，如果v点的最短路径估计值有所调整，则对v的最短路径进行调整。然后再看v点在不在当前的队列中，若不在就将v点放入队尾。这样不断从队列中取出结点来进行松弛操作，直至队列空为止。（不管v在不在队列中都要对v进行松弛）时间复杂度O（km）k<=2，　如果某个点进入队列的次数超过N次则存在负环（SPFA无法处理带负环的图）

### 3.3.5分层图最短路

#### 3.3.5.1code

#include<stdio.h>

#include<queue>

#include<vector>

#include<cstring>

#define INF 0xfffffff

using namespace std;

typedef long long ll;

const int maxn=1e5+10;

const int maxm=2e5+10;

const int maxk=15;

struct edge

{

int u;

int v;

ll d;

edge(int u1=0,int v1=0,ll d1=0)

{

u=u1;

v=v1;

d=d1;

}

};

struct node

{

int cur;

int k;

node(int c=0,int k1=0)

{

cur=c;

k=k1;

}

};

ll dis[maxn][maxk];

vector<int>v[maxn];

vector<edge>e;

bool vis[maxn][maxk];

void init(int n,int m,int k)

{

for(int i=0;i<=n;i++)

{

v[i].clear();

for(int j=0;j<=k;j++)

dis[i][j]=INF;

}

memset(vis,0,sizeof(vis));

e.clear();

}

void read(int n,int m,int k)

{

int u,vv;

ll c;

for(int i=0;i<m;i++)

{

scanf("%d %d %lld",&u,&vv,&c);

e.push\_back(edge(u,vv,c));

v[u].push\_back(e.size()-1);

}

}

void spfa(int n,int m,int k)

{

queue<node>q;

node tmp;

q.push(node(1,0));

vis[1][0]=1;

dis[1][0]=0;

while(!q.empty())

{

tmp=q.front();q.pop();

vis[tmp.cur][tmp.k]=0;

for(int i=0;i<v[tmp.cur].size();i++)

{

edge &vv=e[v[tmp.cur][i]];

if(dis[vv.v][tmp.k]>dis[tmp.cur][tmp.k]+vv.d)

{

dis[vv.v][tmp.k]=dis[tmp.cur][tmp.k]+vv.d;

if(!vis[vv.v][tmp.k])

{

vis[vv.v][tmp.k]=1;

q.push(node(vv.v,tmp.k));

}

}

if(tmp.k+1<=k)

{

if(dis[vv.v][tmp.k+1]>dis[tmp.cur][tmp.k])

{

dis[vv.v][tmp.k+1]=dis[tmp.cur][tmp.k];

if(!vis[vv.v][tmp.k+1])

{

vis[vv.v][tmp.k+1]=1;

q.push(node(vv.v,tmp.k+1));

}

}

}

}

}

}

int main()

{

int t=0;

scanf("%d",&t);

while(t--)

{

int n,m,k;

scanf("%d %d %d",&n,&m,&k);

init(n,m,k);

read(n,m,k);

spfa(n,m,k);

printf("%lld\n",dis[n][k]);

}

return 0;

}

#### 3.3.5.2思想

分层图是在给定一个图的情况下，你最多可以选择k条边使得经过他们的花费变为0,这时只需要在dis数组上多加一维，dis[i][k]表示经过i节点，选择k条边的最短路径，其他的套最短路算法，这里用的是spfa

### 3.3.6第K短路（迪杰斯特拉+A\*算法）

#### 3.3.6.1code：

#include <iostream>

#include<unordered\_map>

#include<stdio.h>

#include<queue>

#include<vector>

#include<cstring>

using namespace std;

typedef long long ll;

const int maxn=1e3+10;

const int maxm=1e4+10;

struct node

{

int v;

ll h;

ll g;

node(int v1=0,ll h1=0,ll g1=0)

{

v=v1;h=h1;g=g1;

}

friend bool operator<(node a,node b)

{

return a.g+a.h>b.g+b.h;

}

};

struct edge

{

int v;

ll c;

edge(int v1=0,ll c1=0)

{

v=v1;c=c1;

}

};

struct node\_d

{

int v;

ll dis;

node\_d(int v1=0,ll dis1=0)

{

v=v1;dis=dis1;

}

friend bool operator<(node\_d a,node\_d b)

{

return a.dis>b.dis;

}

};

vector<edge>ver[maxn];

vector<edge>re\_v[maxn];

#define INF 0xffffff

ll dis[maxn];

bool vis[maxn];

void addEdge(int v,int u,ll c)

{

ver[v].push\_back(edge(u,c));

re\_v[u].push\_back(edge(v,c));

}

void Dijstra(int e,int n)

{

memset(vis,0,sizeof(vis));

for(int i=0;i<=n;i++)dis[i]=INF;

priority\_queue<node\_d>q;

dis[e]=0;

q.push(node\_d(e,0));

node\_d tmp;

while(!q.empty())

{

tmp=q.top();q.pop();

if(vis[tmp.v])continue;

vis[tmp.v]=1;

for(int i=0;i<re\_v[tmp.v].size();i++)

{

edge ee=re\_v[tmp.v][i];

if(!vis[ee.v] && dis[ee.v]>dis[tmp.v]+ee.c)

{

dis[ee.v]=dis[tmp.v]+ee.c;

q.push(node\_d(ee.v,dis[ee.v]));

}

}

}

}

ll a\_star(int n,int s,int e,int k)

{

k--;

priority\_queue<node>q;

q.push(node(s,0,dis[s]));

node tmp;

while(!q.empty())

{

tmp=q.top();q.pop();

if(tmp.v==e)

{

if(k)k--;

else

return tmp.h;

}

for(int i=0;i<ver[tmp.v].size();i++)

{

edge ee=ver[tmp.v][i];

q.push(node(ee.v,tmp.h+ee.c,dis[ee.v]));

}

}

return -1;

}

int main()

{

int n,m;

while(~scanf("%d %d",&n,&m))

{

for(int i=0;i<=n;i++){ver[i].clear();re\_v[i].clear();}//yilou

int s,e,k;

ll T;

scanf("%d %d %d %lld",&s,&e,&k,&T);

int u,v;ll w;

for(int i=0;i<m;i++)

{

scanf("%d %d %lld",&u,&v,&w);

addEdge(u,v,w);

}

Dijstra(e,n);

if(s==e)k++;

if(dis[s]==INF){printf("Whitesnake!\n");continue;}

ll ans=a\_star(n,s,e,k);

//printf("%lld\n",ans);

if(ans !=-1 && ans<=T)

printf("yareyaredawa\n");

else

printf("Whitesnake!\n");

}

return 0;

}

#### 3.3.6.2思想

要求第k短路，那么就先需要知道第1短，第2短，。。。第k-1短路，则如果在A\*算法中到达e点到达了k次，则说明此次算得到的就是第k短路。在A\*算法中，有一个启发式的值f=g+h，g代表预测还需要多少，而h则是已经走了多少。在求第k短路中，h容易知道，而g就需要先反向求从e到其他点的最短路，所以先跑了一遍迪杰斯特拉。

## 3.4最大流

### 3.4.1code

#include<stdio.h>

#include<queue>

#include<vector>

#include<algorithm>

#include<cstring>

using namespace std;

typedef long long ll;

#define MAX 0xfffffff

const int maxn=200+10;

const int maxm=200+10;

ll mp[maxn][maxn];

ll a[maxn];

ll p[maxn];

vector<int>v[maxn];

ll bfs(int s,int e,int n)

{

memset(a,0,sizeof(a));

memset(p,0,sizeof(p));

queue<int>q;

a[s]=MAX;

q.push(s);

while(!q.empty())

{

int tmp=q.front();q.pop();

for(int i=0;i<v[tmp].size();i++)

{

int to=v[tmp][i];

if(!a[to] && mp[tmp][to]>0)

{

a[to]=min(a[tmp],mp[tmp][to]);

p[to]=tmp;

q.push(to);

}

}

if(a[e]!=0)break;

}

return a[e];

}

ll solve(int s,int e,int n)

{

ll ans=0;

while(true)

{

ll tmp=bfs(s,e,n);

if(tmp==0)break;

ans+=tmp;

int cur=e;

while(true)

{

mp[p[cur]][cur]-=tmp;

mp[cur][p[cur]]+=tmp;

cur=p[cur];

if(cur==s)break;

}

}

return ans;

}

int main()

{

int n,m;

int u,vv;ll c;

scanf("%d %d",&n,&m);

for(int i=0;i<n;i++)

{

scanf("%d %d %lld",&u,&vv,&c);

v[u].push\_back(vv);

v[vv].push\_back(u);

mp[u][vv]+=c;

mp[vv][u]+=0;

}

printf("%lld\n",solve(1,m,n));

return 0;

}

Code2（建模）

#include<stdio.h>

#include<vector>

#include<queue>

#include<cstring>

using namespace std;

#define MAX 0xfffffff

const int maxn=410;

const int maxm=210;

int a[maxn];

int p[maxn];

vector<int>v[maxn];

int mp[maxn][maxn];

int bfs(int s,int e,int n)

{

memset(a,0,sizeof(a));

memset(p,-1,sizeof(p));

queue<int>q;

a[0]=MAX;

q.push(0);

int cur=0;

while(!q.empty())

{

cur=q.front();q.pop();

for(int i=0;i<v[cur].size();i++)

{

int to=v[cur][i];

if(!a[to] && mp[cur][to])

{

a[to]=min(a[cur],mp[cur][to]);

p[to]=cur;

q.push(to);

}

}

if(a[e]!=0)break;

}

return a[e];

}

int solve(int s,int e,int n)

{

int ans=0;

while(true)

{

int tmp=bfs(s,e,n);

if(tmp==0)break;

ans+=tmp;

int cur=e;

while(true)

{

int from=p[cur];

mp[from][cur]-=tmp;

mp[cur][from]+=tmp;

cur=from;

if(cur==s)break;

}

}

return ans;

}

int main()

{

int n,m;

scanf("%d %d",&n,&m);

for(int i=1;i<=n;i++)

{

int s=0,sch=0;

scanf("%d",&s);

for(int j=0;j<s;j++)

{

scanf("%d",&sch);

v[i].push\_back(sch+n);

v[sch+n].push\_back(i);

mp[i][sch+n]=1;

mp[sch+n][i]=0;

}

}

for(int i=1;i<=n;i++)

{

v[0].push\_back(i);

mp[0][i]=1;

mp[i][0]=0;

v[i].push\_back(0);

}

for(int i=1;i<=m;i++)

{

v[i+n].push\_back(n+m+1);

v[n+m+1].push\_back(i+n);

mp[i+n][n+m+1]=1;

mp[n+m+1][i+n]=0;

}

printf("%d\n",solve(0,n+m+1,n+m+2));

return 0;

}

### 3.4.2原理思想

在求最大流问题中不能出现平行边。通过bfs查找增广路，每查到一条增广路，当前的最大流就会增加这条增广路中最小的权值。

求最大流的思路：

1. 在残缺网络中找出增广路

如果没有找到增广路，则说明已经找到最大流

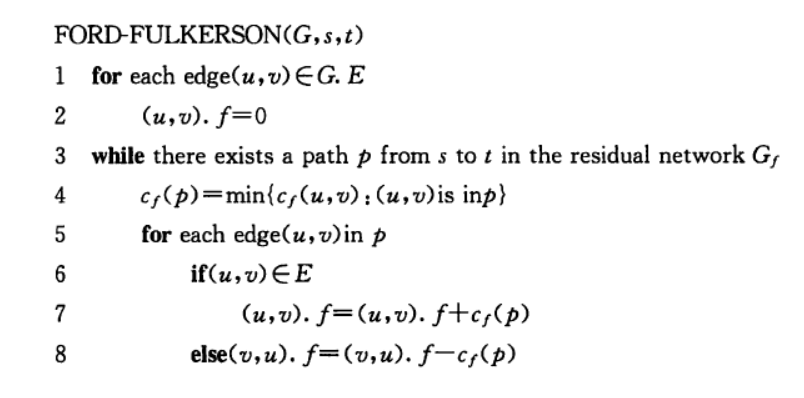
如果找到增广路，则在增广路中找到所能增加的最小的流，然后在增广路的所有边中加上最小的流

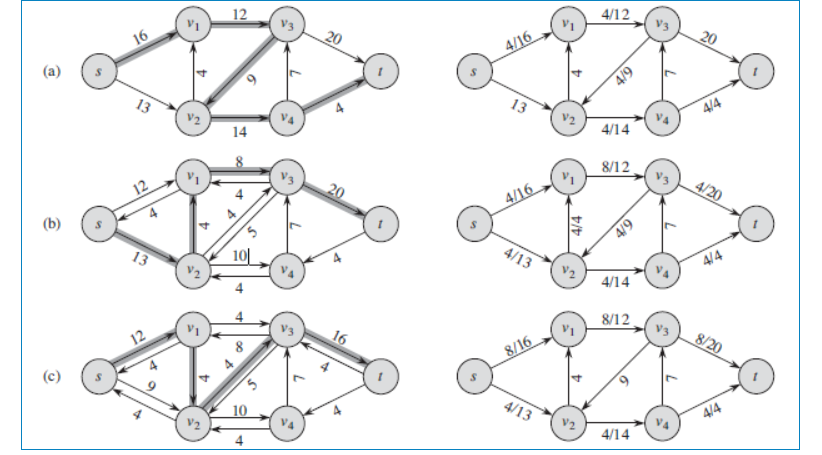
重复上述操作，直到没有增广路为止。

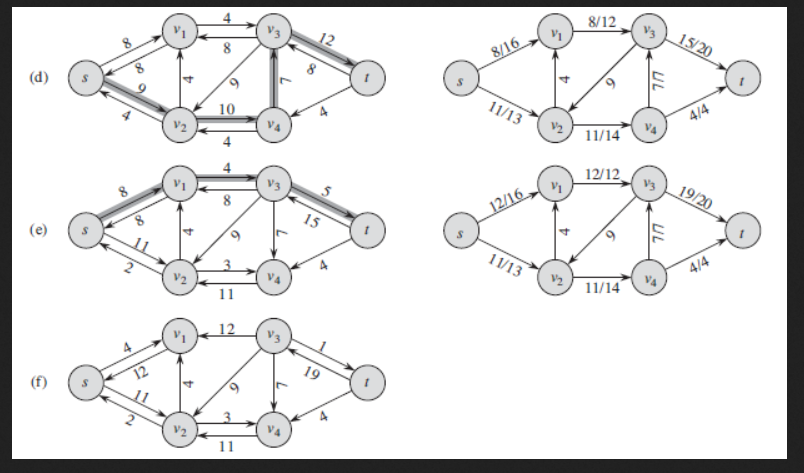
有增广路说明从源点到汇点还可以增加，并且每次增加将更接近最大流。

找增广路用bfs。记录增广路的路径

这道题对图的构造建立了多重索引，所需内存增多，但是方便，易于读懂。







## 3.5最小费用最大流

### 3.5.1code：

#include<vector>

#include<algorithm>

#include<iostream>

#define MAX 0xffffff

using namespace std;

struct edge

{

int from;

int to;

int cap;

int cost;

edge(int f,int t,int ca,int c)

{

from=f;

to=t;

cap=ca;

cost=c;

}

};

vector<int>\*g;

vector<edge>e;

int \*p;

long long \*d;

long long bellman(int n)

{

d=new long long[(n<<1)+1];

p=new int [(n<<1)+1];

for(int i=0;i<(n<<1)+1;i++)

{

d[i]=MAX;

p[i]=-1;

}

d[1]=0;

for(int i=0;i<(n<<1)+1;i++)

{

for(int j=0;j<e.size();j++)

{

edge &ee=e[j];

if(ee.cap>0 && d[ee.to]>d[ee.from]+ee.cost)

{

d[ee.to]=d[ee.from]+ee.cost;

p[ee.to]=j;

}

}

}

return d[n<<1];

}

long long solve(int n)

{

long long cost=0;

while(true)

{

long long tmp=bellman(n);

if(p[n<<1]==-1)break;

cost+=tmp;

int cur=n<<1;

while(true)

{

edge &ee=e[p[cur]];

ee.cap-=1;

if(ee.from+n!=ee.to)

{

edge &eb=e[p[cur]^1];

eb.cap+=1;

}

cur=ee.from;

if(cur==1)break;

}

}

return cost;

}

int main()

{

int n,m;

while(cin>>n>>m)

{

int f,t,c;

e.clear();

g=new vector<int>[n<<1+1];

for(int i=0;i<m;i++)

{

cin>>f>>t>>c;

e.push\_back(edge(f+n,t,1,c));

g[f+n].push\_back(e.size()-1);

e.push\_back(edge(t,f+n,0,0-c));

g[t].push\_back(e.size()-1);

}

for(int i=2;i<=n-1;i++)

{

e.push\_back(edge(i,i+n,1,0));

g[i].push\_back(e.size()-1);

}

e.push\_back(edge(1,1+n,2,0));

g[1].push\_back(e.size()-1);

e.push\_back(edge(n,n<<1,2,0));

g[n].push\_back(e.size()-1);

long long ans=solve(n);

cout<<ans<<endl;

}

return 0;

}

### 3.5.2原理思想（拆点思想）

这道题是一道最小费用流的问题。

这道题需要做一些变化，才能用最小费用流的算法。该题有一个限制就是两条路径中的点不能有相交，所以每个点只可能出现在其中一条路径中（除了其实点和终点），但是最大流问题只是对边的容量进行了限制，并没有对点进行限制。所以这道题还用到了一个技巧，就是拆点，将2~v-1的每个点拆成两个点，一个点负责入度，一个负责出度，两个点之间用一条费用为0流量为1的边进行相连。

这道题还在上一题构造图的技巧中用了一个小技巧，所有除了1和v的点的边和其反向边可以通过二进制运算^1求得0，1为一组，2，3，为一组以此类推。

最小费用流我的理解是最小费用最大流，只不过在最大流算法的寻找增广路时从bfs换成了bellman算法。为什么不用迪杰斯特拉？因为有负数。

### 3.5.3poj2195

#### 3.5.3.1题意：

给定一个矩阵，m代表人，h代表房子，问最少花费多少才能将人装进屋子。多个人可以在同一格，人和房子也可以在同一格，人每走一步消耗1点。例如下面的图，答案为10

5 5

HH..m

.....

.....

.....

mm..H

#### 3.5.3.2：思路

计算出每个人到每一个房子的距离，然后跑最小费用最大流即可。注意这里跑最短路的算法不能用O（n^2）会超时，这里用spfa可以过。

#### 3.5.3.3code

#include<iostream>

#include<stdio.h>

#include<vector>

#include<queue>

#include<algorithm>

#include<math.h>

using namespace std;

const int maxn=5e4+10;

struct point

{

int x,y;

point(int x1=0,int y1=0)

{

x=x1;y=y1;

}

};

struct edge

{

int from;

int to;

int cap;

int cost;

edge(int f,int t,int ca,int c)

{

from=f;

to=t;

cap=ca;

cost=c;

}

};

vector<int>\*g;

vector<edge>e;

int p[maxn];

long long d[maxn];

bool vis[maxn];

int me,h;

point men[maxn];

point house[maxn];

#define MAX 0xffffff

long long bellman(int n)

{

for(int i=0;i<n;i++){d[i]=MAX;p[i]=-1;}

d[0]=0;

queue<int>q;

q.push(0);

while(!q.empty())

{

int tmp=q.front();q.pop();

vis[tmp]=0;

for(int i=0;i<g[tmp].size();i++)

{

edge &ee=e[g[tmp][i]];

if(ee.cap>0 && d[ee.to]>d[ee.from]+ee.cost)

{

d[ee.to]=d[ee.from]+ee.cost;

p[ee.to]=g[tmp][i];

if(!vis[ee.to])

{

q.push(ee.to);

vis[ee.to]=1;

}

}

}

}

return d[n-1];

}

long long solve(int n)

{

long long cost=0;

while(1)

{

long long tmp=bellman(n);

if(p[n-1]==-1)break;

cost+=tmp;

int cur=n-1;

while(true)

{

edge &ee=e[p[cur]];

ee.cap=-1;

if(ee.to!=n-1)

{

edge &eb=e[p[cur]^1];

eb.cap+=1;

}

cur=ee.from;

if(cur==0)break;

}

}

return cost;

}

void create(int n,int m)

{

for(int i=1;i<me;i++)

{

for(int j=1;j<h;j++)

{

int len=abs(men[i].x-house[j].x)+abs(men[i].y-house[j].y);

e.push\_back(edge(i,j+me-1,1,len));

g[i].push\_back(e.size()-1);

e.push\_back(edge(j+me-1,i,0,-len));

g[j+me-1].push\_back(e.size()-1);

}

}

for(int i=1;i<me;i++)

{

e.push\_back(edge(0,i,1,0));

g[0].push\_back(e.size()-1);

e.push\_back(edge(i,0,0,0));

g[i].push\_back(e.size()-1);

}

for(int i=1;i<h;i++)

{

e.push\_back(edge(i+me-1,h+me-1,1,0));

g[i+me-1].push\_back(e.size()-1);

e.push\_back(edge(h+me-1,i+me-1,0,0));

g[h+me-1].push\_back(e.size()-1);

}

}

int main()

{

int n,m;

while(1)

{

scanf("%d %d",&n,&m);

if(n==0 && m==0)break;

char tmp;

me=1,h=1;

e.clear();

for(int i=1;i<=n;i++)

{

for(int j=1;j<=m;j++)

{

cin>>tmp;

if(tmp=='H')

{

house[h++]=point(i,j);

}

else if(tmp=='m')

{

men[me++]=point(i,j);

}

}

}

g=new vector<int> [me+h];

create(n,m);

int ans=solve(me+h);

printf("%d\n",ans);

}

return 0;

}

## 3.6二分图最大匹配

### 3.6.1code

匈牙利算法

#include<iostream>

#include<vector>

#include<string.h>

using namespace std;

vector<int> \*m;

int \*ri;

int \*le;

bool \*used;

int XiongYaLi(int index)

{

for(int i=0;i<m[index].size();i++)

{

int tmp=m[index][i];

if(!used[tmp])

{

used[tmp]=1;

if(ri[tmp]==-1 || XiongYaLi(ri[tmp]))

{

ri[tmp]=index;

le[index]=tmp;

return 1;

}

}

}

return 0;

}

int solve(int n)

{

le=new int[n];

ri=new int [n];

for(int i=0;i<n;i++)

{

le[i]=-1;

ri[i]=-1;

}

int ans=0;

used=new bool[n];

for(int i=0;i<n;i++)

{

if(le[i]==-1)

{

for(int j=0;j<n;j++)

used[j]=0;

ans+=XiongYaLi(i);

}

}

return ans;

}

int main()

{

int n,k;

cin>>n>>k;

m=new vector<int>[n];

int x=0,y=0;

for(int i=0;i<k;i++)

{

cin>>x>>y;

m[x-1].push\_back(y-1);

}

cout<<solve(n)<<endl;

return 0;

}

### 3.6.2原理思想

最小覆盖：用最少的点包含二分图中所有的边。

这道题边就是怪，x轴做为x集点，y轴做y集点。最小覆盖节点数=最大匹配数。

求二分图的最大匹配：

1. 找增广路（先从未盖点开始查找，未匹配边，匹配边，未匹配边，匹配边。。。未匹配边）最后一定是未匹配边，结尾是未盖点。将增广路进行反转

每找到一条增广路就多一条边

## 3.7二分图完美匹配

### 3.7.1code

### 3.7.2原理思想

## 3.8强连通分量（缩点）

### 3.8.1code（java）

Java

**class** solve

{

Vector<Integer>[]zhen;

Vector<Integer>[]ni;

Vector<Integer>tmp;

Vector<Integer>[]suo;

Vector<Integer>num;

**boolean** []vis;

**int** []cnt;

**int** n;

**void** f\_dfs(**int** cur)

{

**if**(vis[cur])

{

**return** ;

}

vis[cur]=**true**;

**for**(**int** i=0;i<zhen[cur].size();i++)

{

f\_dfs(zhen[cur].elementAt(i));

}

tmp.addElement(cur);

}

**void** s\_dfs(**int** cur,**int** k)

{

**if**(cnt[cur]!=-1)

{

**if**(cnt[cur]!=k)

{

suo[k].addElement(cnt[cur]);

}

**return** ;

}

cnt[cur]=k;

num.setElementAt(1+num.elementAt(k), k);

**for**(**int** i=0;i<ni[cur].size();i++)

{

s\_dfs(ni[cur].elementAt(i),k);

}

}

**void** t\_dfs(**int** cur)

{

**if**(vis[cur])

{

**return** ;

}

vis[cur]=**true**;

**for**(**int** i=0;i<suo[cur].size();i++)

{

t\_dfs(suo[cur].elementAt(i));

}

}

**public** solve()

{

Scanner sc=**new** Scanner(System.***in***);

**int** m=0;

n=sc.nextInt();

m=sc.nextInt();

zhen=**new** Vector[n+1];

ni=**new** Vector[n+1];

tmp=**new** Vector();

num=**new** Vector();

suo=**new** Vector[n+1];

cnt=**new** **int**[n+1];

vis=**new** **boolean**[n+1];

**for**(**int** i=0;i<vis.length;i++)

{

cnt[i]=-1;

vis[i]=**false**;

zhen[i]=**new** Vector();

ni[i]=**new** Vector();

suo[i]=**new** Vector();

}

**int** f,t;

**for**(**int** i=0;i<m;i++)

{

f=sc.nextInt();

t=sc.nextInt();

zhen[f].addElement(t);

ni[t].addElement(f);

}

**for**(**int** i=1;i<=n;i++)

{

**if**(!vis[i])

{

f\_dfs(i);

}

}

**int** k=0;

**for**(**int** i=tmp.size()-1;i>=0;i--)

{

**if**(cnt[tmp.elementAt(i)]==-1)

{

num.addElement(0);

s\_dfs(tmp.elementAt(i),k);

k++;

}

}

**for**(**int** i=0;i<vis.length;i++)

{

vis[i]=**false**;

}

t\_dfs(k-1);

**int** ans=0;

**boolean** flag=**false**;

**for**(**int** i=0;i<k;i++)

{

**if**(!vis[i])

{

flag=**true**;

**break**;

}

}

**if**(flag)

{

System.***out***.println(ans);

}

**else**

{

ans=num.elementAt(k-1);

System.***out***.println(ans);

}

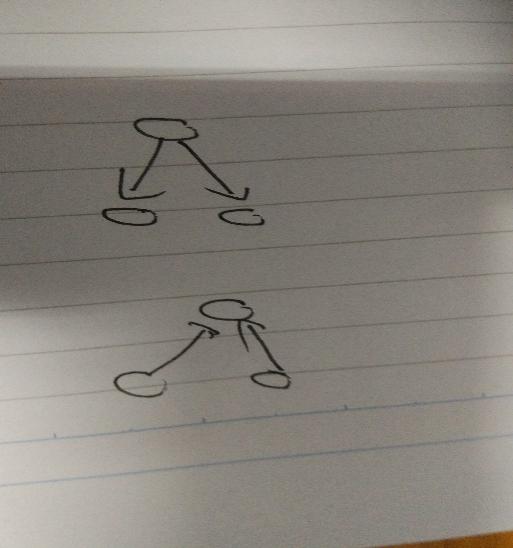
}

}

### 3.8.2原理思想

如何求出强连通分量，并进行缩点？

求出强连通分量需要两次dfs，时间复杂度是O（n+m），思路是：进行缩点后那一定是一个DAG图，那么如果采用按照拓扑排序的逆序开始进行标记，则可以准确无误的将所有的强连通分量进行标记



在进行第二次dfs时就可以确定一个缩点后的DAG图了。

DAG的性质：给定一个DAG（有向无环图），至少需要max（入度为0得个数，出度为0的个数）条边才能使DAG变成强连通分量。

### 3.8.3缩点（Tarjan）+SPFA

Code：

#include<iostream>

#include<vector>

#include<algorithm>

#include<cstring>

#include<queue>

using namespace std;

typedef long long ll;

const int maxn=5e4+10;

struct node

{

ll dis;

int v;

node(int v1=0,ll d1=0){v=v1;dis=d1;}

};

vector<node>v[maxn];

vector<node>suo[maxn];

vector<int>tmp[maxn];

int low[maxn];

int dfn[maxn];

int stack[maxn];

int tt;

int scc;

int cnt;

int vis[maxn];

int color[maxn];

void Targan(int cur)

{

vis[cur]=2;

low[cur]=dfn[cur]=++cnt;

stack[++tt]=cur;

for(int i=0;i<tmp[cur].size();i++)

{

int u=tmp[cur][i];

if(!dfn[u]){Targan(u);low[cur]=min(low[cur],low[u]);}

else if(vis[u]==2)

low[cur]=min(low[cur],dfn[u]);

}

if(low[cur]==dfn[cur])

{

scc++;

do

{

color[stack[tt]]=scc;

vis[stack[tt]]=1;

}while(stack[tt--]!=cur);

}

}

ll dis[maxn];

const ll INF=0x3fffffff;

ll spfa(int s,int e)

{

memset(vis,0,sizeof(vis));

for(int i=0;i<=maxn;i++)dis[i]=INF;

dis[s]=0;

queue<int>q;

q.push(s);

int tmp=0;

while(!q.empty())

{

tmp=q.front();q.pop();

vis[tmp]=0;

for(int i=0;i<suo[tmp].size();i++)

{

node &cur=suo[tmp][i];

if(dis[cur.v]>dis[tmp]+cur.dis)

{

dis[cur.v]=dis[tmp]+cur.dis;

if(!vis[cur.v])

{

q.push(cur.v);

vis[cur.v]=1;

}

}

}

}

return dis[e];

}

void init()

{

memset(vis,0,sizeof(vis));

tt=0;

cnt=0;

scc=0;//bixuwei1

}

int main()

{

int n,m,k;

cin>>n>>m>>k;

int ui,vi;ll ti;

for(int i=0;i<m;i++)

{

cin>>ui>>vi>>ti;

v[ui].push\_back(node(vi,ti));

tmp[ui].push\_back(vi);

}

init();

for(int i=1;i<=n;i++)

{

if(!dfn[i])

{

Targan(i);

}

}

for(int i=1;i<=n;i++)

{

for(int j=0;j<tmp[i].size();j++)

{

if(color[i]!=color[tmp[i][j]])

{

suo[color[i]].push\_back(node(color[v[i][j].v],v[i][j].dis));

}

}

}

cout<<spfa(color[1],color[k])<<endl;

return 0;

}

# 4字符串

### 4.1回文

#### 4.1.1描述

给定一个字符串，判断是否能够进行一次交换，使得字符串变成回文字符串

#### 4.1.2code

#include<stdio.h>

#include<string.h>

#include<map>

#include<vector>

#include<set>

using namespace std;

const int maxn=5e5+10;

char buf[maxn];

int arr[28];

int main()

{

scanf("%s",buf);

int i=0,j=strlen(buf)-1;

int count=0;

int r[2];

int l[2];

set<char>s;

map<char,int>m;

while(i<j)

{

if(buf[i]!=buf[j])

{

s.insert(buf[i]);

s.insert(buf[j]);

m[buf[i]]++;

m[buf[j]]++;

count++;

if(count>2)

break;

}

i++;j--;

}

if(count==0)

{

printf("YES\n");

}

else if(count==2)

{

if(s.size()==2)

{

set<char>::iterator it=s.begin();

bool flag=true;

while(it!=s.end())

{

if(m[\*it]!=2)

{

flag=false;

break;

}

it++;

}

if(flag)

printf("YES\n");

else

printf("NO\n");

}

else

{

printf("NO\n");

}

}

else if(count==1)

{

if(strlen(buf)%2==0)

printf("NO\n");

else

{

int size=strlen(buf);

char tmp=buf[size/2];

bool flag=false;

set<char>::iterator iter=s.begin();

while(iter!=s.end())

{

if(\*iter==tmp)

{

flag=true;

break;

}

iter++;

}

if(flag)

printf("YES\n");

else

printf("NO\n");

}

}

else

printf("NO\n");

return 0;

}

#### 4.1.3思路

首先判断是否是回文字符串，如果是则可以在对称的地方进行交换。

如果有1处地方不同，如果字符串长度为偶数，则不可能进行交换完后变成回文字符串

如果为奇数，判断不对的两个位置和中间是否相同，如果相同可以交换，否则不行。

如果有两处不同，那么不相同的字符只能出现两种，并且两种字符的次数要一样。

其他的情况不行

## 4.2trie

### 4.2.1code（poj2001）

#include<stdio.h>

#include<cstring>

using namespace std;

const int maxn=2e4+10;

const int sigmod\_size=30;

char str[1001][25];

struct trie

{

int sz;

int tree[maxn][sigmod\_size];

int val[maxn];

char ans[sigmod\_size];

trie()

{

sz=1;

memset(tree[0],0,sizeof(tree[0]));

memset(val,0,sizeof(val));

}

int idx(char c){return c-'a';}

char getx(int c){return c+'a';}

void insert\_word(char \*s)

{

int u=0,size=strlen(s);

for(int i=0;i<size;i++)

{

int tmp=idx(s[i]);

val[u]++;

if(tree[u][tmp]==0)

{

memset(tree[sz],0,sizeof(tree[sz]));

tree[u][tmp]=sz++;

}

u=tree[u][tmp];

}

val[u]++;

}

void find\_word(char \*s)

{

int u=0,size=strlen(s),k=0;

for(int i=0;i<size;i++)

{

ans[k++]=s[i];

int tmp=idx(s[i]);

u=tree[u][tmp];

if(val[u]==1)

break;

}

ans[k++]='\0';

return ;

}

}t;

int main()

{

int i=0;

while(scanf("%s",str[i])!=EOF)

{

t.insert\_word(str[i]);

i++;

}

for(int j=0;j<i;j++)

{

t.find\_word(str[j]);

printf("%s %s\n",str[j],t.ans);

}

return 0;

}

### 4.2.2思想

Trie的思想是利用各字符串的前缀相同的特点，进行构造一颗树。

## 4.3kmp

### 4.3.1code

#include<stdio.h>

#include<cstring>

using namespace std;

const int maxn=1e6+10;

const int maxm=1e4+10;

int a[maxn];

int b[maxm];

int tmp[maxm];

void makenext(int m)//最长前后缀相同次数

{

for(int i=1,j=0;i<m;i++)

{

while(j>0 && b[j]!=b[i])j=tmp[j-1];

if(b[j]==b[i])j++;

tmp[i]=j;

}

}

int kmp(int n,int m)

{

int ans=-1;

int sz=n;

int szb=m;

for(int i=0,j=0;i<sz;i++)

{

while(j>0 && b[j]!=a[i])j=tmp[j-1];

if(a[i]==b[j])j++;

if(j==szb){ans=i-szb+1;return ans+1;}

}

return ans;

}

int main()

{

int t=0;

scanf("%d",&t);

while(t--)

{

int n,m;

scanf("%d %d",&n,&m);

for(int i=0;i<n;i++)

scanf("%d",&a[i]);

for(int i=0;i<m;i++)scanf("%d",&b[i]);

makenext(m);

printf("%d\n",kmp(n,m));

}

return 0;

}

### 4.3.2思想：

Kmp算法的思想是每次在字符串进行匹配的时候如果有位置与模板字符串不相同则不会简单的往后移一位。而是有一个next数组（模板中为tmp），该数组记录的是在当前位置的前缀子串中，最长的前后缀长度，每次不匹配的时候都是将cur移动到next[cur-1]所指向的位置。时间复杂度：O(n)。

### 4.3.3kmp变形（hdu1686）

#### 4.3.3.1code：

#include<stdio.h>

#include<cstring>

using namespace std;

const int maxw=1e4+10;

const int maxt=1e6+10;

char w[maxw];

char t[maxt];

int tmp[maxw];

void makenext(int szw)

{

int sz=szw;

for(int i=1,j=0;i<sz;i++)

{

while(j>0 && w[i]!=w[j])j=tmp[j-1];

if(w[i]==w[j])j++;

tmp[i]=j;

}

}

int kmp(int szw,int szt)

{

int ans=0;

for(int i=0,j=0;i<szt;i++)

{

while(j>0 && w[j]!=t[i])j=tmp[j-1];

if(w[j]==t[i])j++;

if(j==szw){ans++;j=tmp[j-1];}

}

return ans;

}

int main()

{

int ca=0;

scanf("%d",&ca);

while(ca--)

{

scanf("%s %s",w,t);

int szw=strlen(w);

int szt=strlen(t);

makenext(szw);

printf("%d\n",kmp(szw,szt));

}

return 0;

}

#### 4.3.3.2思想：

这道题yongkmp求在给定的一个文本串中，找出模板串在文本串中出现几次，但是有可能出现这样的数据：文本串：AZAZAZA，模板串：AZA，ans=3；如果每次都匹配成功后又将模板串的cur重新置0太慢。需要利用next数组的记录前后缀的信息，从next[j-1]的位置开始，并且此时i不能--，因为比较的是i的下一个元素开始比较。

### 4.3.4求一个字符串的最小循环节

#### 4.3.4.1code：

void makenext(int sz)

{

for(int i=1,j=0;i<sz;i++)

{

while(j>0 && a[j]!=a[i])j=tmp[j-1];

if(a[j]==a[i])j++;

tmp[i]=j;

}

}

Ans=strlen(str)-tmp[strlen(str)-1]//字符串长度-kmp算法next数组的最后一个元素的值。

#### 4.3.4.2思想：

,由于next数组记录的是该字符串中最长的前后缀长度，所以如果最后一个元素不为0，那字符串长度-她所代表的值（重复出现的次数），就是该字符串的最小循环节长度。

性质：如果len%（len-next[len-1]）==0，则字符串中必存在最小循环节，且循环次数即为len/（len-next[len-1]）;

证明：在前len个字符组成的字符串，存在最小循环节k，那么next[len-1]=len-k;（为什么呐？因为next数组的定义就是最大前后缀相同的子串的长度，len的总长度减去最小循环节，比如有3个循环节，减去一个剩下两个，就是最大循环节）那么循环次数就是len/（len-next[len-1]）;因为len-next[len-1]=k;所以得出公式；

# 算法技巧

## 5.1求逆序对

### 5.1.1code：

#include<iostream>

#include<cstdio>

using namespace std;

int n,a[2000001],i,c[2000001];

long long ans;

typedef

void x(int l,int r)

{

int mid=(l+r)/2,i,j,tmp;

if(r>l)

{

x(l,mid);

x(mid+1,r);

tmp=l;

for(i=l,j=mid+1;i<=mid&&j<=r;)

{

if(a[i]>a[j])

{

c[tmp++]=a[j++];

ans+=mid-i+1;

}

else c[tmp++]=a[i++];

}

if(i<=mid) for(;i<=mid;) c[tmp++]=a[i++];

if(j<=r) for(;j<=r;) c[tmp++]=a[j++];

for(i=l;i<=r;i++) a[i]=c[i];

}

}

int main()

{

cin>>n;

for(i=0;i<n;i++) scanf("%d",&a[i]);

x(0,n-1);

cout<<ans;

}

### 5.1.2思想

利用归并排序的思想求解逆序对

# 6.思想

## 6.1状态压缩

### 6.1.1题目

度度熊为了完成毕业论文，需要收集一些数据来支撑他的论据，于是设计了一份包含 m*m* 个问题的调查问卷，每个问题只有 'A' 和 'B' 两种选项。

将问卷散发出去之后，度度熊收到了 n*n* 份互不相同的问卷，在整理结果的时候，他发现可以只保留其中的一部分问题，使得这 n*n* 份问卷仍然是互不相同的。这里认为两张问卷是不同的，当且仅当存在至少一个被保留的问题在这两份问卷中的回答不同。

现在度度熊想知道，存在多少个问题集合，使得这 n*n* 份问卷在只保留这个集合的问题之后至少有 k*k* 对问卷是不同的。

**Input**

第一行包含一个整数 T*T*，表示有 T*T* 组测试数据。

接下来依次描述 T*T* 组测试数据。对于每组测试数据：

第一行包含三个整数 n*n*，m*m* 和 k*k*，含义同题目描述。

接下来 n*n* 行，每行包含一个长度为 m*m* 的只包含 'A' 和 'B' 的字符串，表示这份问卷对每个问题的回答。

保证 1 \leq T \leq 1001≤*T*≤100，1 \leq n \leq 10^31≤*n*≤10​3​​，1 \leq m \leq 101≤*m*≤10，1 \leq k \leq 10^61≤*k*≤10​6​​，给定的 n*n* 份问卷互不相同。

**Output**

对于每组测试数据，输出一行信息 "Case #x: y"（不含引号），其中 x 表示这是第 x*x* 组测试数据，y 表示满足条件的问题集合的个数，行末不要有多余空格。

**Sample Input**

2

2 2 1

AA

BB

2 2 2

AA

BB

**Sample Output**

Copy

Case #1: 3

Case #2: 0

### 6.1.2code

#include<stdio.h>

#include<map>

#include<cstring>

using namespace std;

typedef long long ll;

const int maxn=1e3+10;

ll num[maxn];

char buf[12];

bool check(int n,map<ll,ll>& m,int k)

{

ll tmp=0;

for(map<ll,ll>::iterator it=m.begin();it!=m.end();it++)

{

tmp+=(it->second)\*(n-it->second);

n-=it->second;

}

if(tmp>=k)

return true;

else

return false;

}

int main()

{

int t=0;

scanf("%d",&t);

int x=1;

while(x<=t)

{

int n,m,k;

scanf("%d%d%d",&n,&m,&k);

for(int i=1;i<=n;i++)

num[i]=0;

for(int i=1;i<=n;i++)

{

scanf("%s",buf);

for(int j=0;j<m;j++)

{

if(buf[j]=='A')

num[i]=num[i]<<1|1;

else

num[i]<<=1;

}

}

map<ll,ll>mp;

ll ans=0;

for(int i=1;i<(1<<m);i++)

{

mp.clear();

for(int j=1;j<=n;j++)

{

if(mp[num[j]&i]==-1)

mp[num[j]&i]=1;

else

mp[num[j]&i]++;

}

if(check(n,mp,k))ans++;

}

printf("Case #%d: %I64d\n",x++,ans);

}

return 0;

}

### 6.1.3思想

这道题是将m个问题进行压缩，对于m个问题，一共有2^m种组合，然后通过所选择的问题与每位成员的答题情况进行&运算，用map记录出现的情况。然后计算这些组合的个数会不会超过k。A,b,c,d,e ,n为答题人数。A\*(n-a)为与a有关的所有组合数。

# 7.其他

## 7.1hdu6301

### 7.1.1描述

Chiaki has an array of nn positive integers. You are told some facts about the array: for every two elements aiai and ajaj in the subarray al..ral..r (l≤i<j≤rl≤i<j≤r), ai≠ajai≠aj holds.   
Chiaki would like to find a lexicographically minimal array which meets the facts.

**Input**

There are multiple test cases. The first line of input contains an integer TT, indicating the number of test cases. For each test case:   
  
The first line contains two integers nn and mm (1≤n,m≤1051≤n,m≤105) -- the length of the array and the number of facts. Each of the next mm lines contains two integers lili and riri (1≤li≤ri≤n1≤li≤ri≤n).   
  
It is guaranteed that neither the sum of all nnnor the sum of all mm exceeds 106106.

**Output**

For each test case, output nn integers denoting the lexicographically minimal array. Integers should be separated by a single space, and no extra spaces are allowed at the end of lines.

**Sample Input**

3

2 1

1 2

4 2

1 2

3 4

5 2

1 3

2 4

**Sample Output**

1 2

1 2 1 2

1 2 3 1 1

### 7.1.2code

#include<stdio.h>

#include<algorithm>

#include<cstring>

using namespace std;

const int maxn=1e5+10;

struct node

{

int l;

int r;

}p[maxn];

int pre[maxn];

int ans[maxn];

bool cmp(node a,node b)

{

if(a.l<b.l)

return true;

else if(a.l==b.l)

return a.r>b.r;

else

return false;

}

int main()

{

int t=0,n,m;

scanf("%d",&t);

while(t--)

{

scanf("%d %d",&n,&m);

memset(pre,0,sizeof(pre));

memset(ans,0,sizeof(ans));

for(int i=0;i<m;i++)

scanf("%d %d",&p[i].l,&p[i].r);

sort(p,p+m,cmp);

int cur=1;

for(int i=0;i<m;i++)

{

int cnt=1;

for(cur=max(cur,p[i].l);cur<=p[i].r;cur++)

{

while(pre[cnt]>=p[i].l)

cnt++;

ans[cur]=cnt;

pre[cnt]=cur;

cnt++;

}

}

for(int i=1;i<n;i++)

{

if(ans[i]==0)

printf("1 ");

else

printf("%d ",ans[i]);

}

if(ans[n]==0)

printf("1\n");

else

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

}

return 0;

}

### 7.1.3问题及思路

解题思路：先将每一次的问题先排序，越靠近左边的优先，然后如果左边的端点一样，则比较右边的端点，越靠右的优先。然后设置一个前驱数组，记录cnt的上一次位置在哪，如果小于当前询问的左端点，则允许分配，否则cnt自加。

## 7.2银行家算法优化

### 7.2.1code：（hdu6396）

#include<stdio.h>

#include<queue>

using namespace std;

namespace Input

{

const int BUF = 65536;

char buf[BUF + 1];

char \*head = buf, \*tail = buf;

}

inline char inputchar()

{

using namespace Input;

if (head == tail)

\*(tail = (head = buf) + fread(buf, 1, BUF, stdin)) = 0;

return \*head++;

}

inline void input(int &ret)

{

char ch = inputchar();

while (ch < '0' || ch > '9')

ch = inputchar();

ret = ch - '0';

ch = inputchar();

while (ch >= '0' && ch <= '9')

{

ret = ret \* 10 + ch - '0';

ch = inputchar();

}

}

inline void input(long long &ret)

{

char ch = inputchar();

while (ch < '0' || ch > '9')

ch = inputchar();

ret = ch - '0';

ch = inputchar();

while (ch >= '0' && ch <= '9')

{

ret = ret \* 10 + ch - '0';

ch = inputchar();

}

}

struct node

{

int kill;

int id;

node(int k=0,int i=0)

{

kill=k;id=i;

}

friend bool operator< (node a,node b)

{

return a.kill>b.kill;

}

};

const int maxn=1e5+10;

int a[maxn][5+1];

int b[maxn][6];

int v[6];

int ans=0;

priority\_queue<node> q[6];

inline void add(int x,int k)

{

ans++;

for(int i=1;i<=k;i++)

v[i]+=b[x][i];

}

int main()

{

int t=0;

//read(t);

input(t);

while(t--)

{

int n,k;

ans=0;

input(n);input(k);//read(n);read(k);

for(int i=1;i<=k;i++)

input(v[i]);//read(v[i]);

for(int i=1;i<=n;i++)

{

for(int j=1;j<=k;j++)

input(a[i][j]);//read(a[i][j]);

for(int j=1;j<=k;j++)

input(b[i][j]);//read(b[i][j]);

}

for(int i=1;i<=k;i++)q[i]=priority\_queue<node>();

for(int i=1;i<=n;i++)

q[1].push(node(a[i][1],i));

bool flag=true;

while(flag)

{

flag=false;

for(int i=1;i<=k;i++)

{

while(!q[i].empty())

{

node tmp;tmp=q[i].top();

if(tmp.kill>v[i])break;

//flag=true;

q[i].pop();

if(i==k){add(tmp.id,k);flag=true;}

else

{

q[i+1].push(node(a[tmp.id][i+1],tmp.id));

}

}

}

}

printf("%d\n",ans);

for(int i=1;i<k;i++)

printf("%d ",v[i]);

printf("%d\n",v[k]);

}

return 0;

}

### 7.2.2思路：

假定现在有n个程序，然后共有k个资源，每个资源都是有限的，现在问最大能够满足的程序有多少个，并且输出最后剩下的资源。

思路是将k个资源进行单独排序，一开始先将n个程序加入第一个堆中，该堆按照第一个资源进行排序，每次都从堆中取出一个，如果当前的程序所需的第一个资源允许分配，则将其从堆中弹出将其加入到第二个堆中（第二个堆维护的是第二个资源），一直到最后的第k个堆，如果从第k个堆中弹出，说明可以分配，如果当前堆的堆首不能分配，则跳至下一个堆，直到不能分配为止。

# 8.leetcode

## 8.1leetcode128

### 8.1.1描述

给定一个未排序的整数数组，找出最长连续序列的长度。

要求算法的时间复杂度为 O(n)。

示例:

输入: [100, 4, 200, 1, 3, 2]

输出: 4

解释: 最长连续序列是 [1, 2, 3, 4]。它的长度为 4。

### 8.1.2code：

class Solution {

public:

int longestConsecutive(vector<int>& nums) {

unordered\_map<int,bool>mp;

unordered\_map<int,int>index;

for(int i=0;i<nums.size();i++)

{

mp[nums[i]]=true;

index[nums[i]]=i;

}

int ans=0;

bool \*array=new bool[nums.size()+1];

for(int i=0;i<nums.size();i++)array[i]=false;

for(int i=0;i<nums.size();i++)

{

if(array[i])continue;

array[i]=true;

int ret=1;

int l=nums[i]-1;

while(l>=INT\_MIN && mp[l] )

{

array[index[l]]=true;

ret++;

l--;

}

int r=nums[i]+1;

while(r<=INT\_MAX && mp[r])

{

array[index[r]]=true;

ret++;

r++;

}

ans=max(ans,ret);

}

return ans;

}

};

### 8.1.3思想：

该题最重要的是使用STL库里的unordered\_map，时间复杂度为O（1），但是在较新的c++编译器才会有。然后该题的思想是从一个数的两边扩展，查看是否可以扩展。如果在之前扩展的过程中出现过，则跳过。

# 9.博弈论

## 9.1.1description

给定一棵*n* 个点的树，每个点有权值。两个人玩游戏，先

手需要占领若干不相邻的点，然后后手占领剩下所有点。

每个人的得分为占领的点权异或和，分高的获胜。问最优策

略下游戏的结果。

1 *≤ n ≤* 105。

Shortest judge solution: 235 bytes

## 9.1.2解题方法：

设*sum* 为所有点权的异或和，*A* 为先手得分，*B* 为后手得

分。

若*sum* = 0，则*A* = *B*，故无论如何都是平局。

否则考虑*sum* 二进制下最高的1 所在那位，一定有奇数个

点那一位为1。

若先手拿走任意一个那一位为1 的点，则*B* 该位为0，故

先手必胜。

时间复杂度*O*(*n*)。

## 9.1.3Code：

#include<stdio.h>

using namespace std;

typedef long long ll;

int main()

{

int t=0;

scanf("%d",&t);

while(t--)

{

int n=0;

scanf("%d",&n);

ll ans=0;

ll x=0;

for(int i=0;i<n;i++)

{

scanf("%lld",&x);

ans=ans^x;

}

int z,y;

for(int i=0;i<n-1;i++)

scanf("%d %d",&z,&y);

if(ans==0)printf("D\n");

else printf("Q\n");

}

return 0;

}

# 输入输出

## 10.1输入（如果时间为1s，且输入数据>1e6必须加快输入）

### 10.1.1Code：（不适合调试）

namespace Input

{

const int BUF = 65536;

char buf[BUF + 1];

char \*head = buf, \*tail = buf;

}

inline char inputchar()

{

using namespace Input;

if (head == tail)

\*(tail = (head = buf) + fread(buf, 1, BUF, stdin)) = 0;

return \*head++;

}

inline void input(int &ret)

{

char ch = inputchar();

while (ch < '0' || ch > '9')

ch = inputchar();

ret = ch - '0';

ch = inputchar();

while (ch >= '0' && ch <= '9')

{

ret = ret \* 10 + ch - '0';

ch = inputchar();

}

}

inline void input(long long &ret)

{

char ch = inputchar();

while (ch < '0' || ch > '9')

ch = inputchar();

ret = ch - '0';

ch = inputchar();

while (ch >= '0' && ch <= '9')

{

ret = ret \* 10 + ch - '0';

ch = inputchar();

}

}

## 10.2刷新缓冲区

fflush(stdin);

不过最好还是用字符串来存比较好。

## 10.3输出（感觉没有printf快）

inline void out(int x)

{

if(x>9)out(x/10);

putchar(x%10+'0');

}

# DP

## 11.1最长上升子序列

### 11.1.1（从1~i）（O（n））

#### 11.1.1.1code：

const int maxn=1e5+20;

int dp\_pre[maxn];

int dp\_next[maxn];

dp\_pre[1]=1;

ll maxx=a[1];

for(int i=2;i<=n;i++)

{

if(maxx<a[i])

{

dp\_pre[i]=dp\_pre[i-1]+1;

maxx=a[i];

}

else

dp\_pre[i]=dp\_pre[i-1];

}

#### 11.1.1.2思想

Dp[i-1]+1, a[i]>max;

状态转移方程：dp[i]=

Dp[i-1], a[i]<=max

其中max为前缀中记录的最大值。

### 11.1.2（i~n）（O（nlogn））

该方法需要用到线段树

#### 11.1.2.1code

typedef long long ll;

const int maxn=1e5+20;

int dp\_next[maxn];

#define INF 0xffffff

struct node

{

ll mmax;

int cur;

node(ll mx=0,int c=0)

{

mmax=mx;cur=c;

}

}tree[maxn<<2];

ll a[maxn];

void up(int index)

{

if(tree[index<<1].mmax>tree[index<<1|1].mmax)

{

tree[index].mmax=tree[index<<1].mmax;

tree[index].cur=tree[index<<1].cur;

}

else

{

tree[index].mmax=tree[index<<1|1].mmax;

tree[index].cur=tree[index<<1|1].cur;

}

}

void build(int l,int r,int index)

{

if(l==r)

{

tree[index].mmax=a[l];

tree[index].cur=l;

return ;

}

int mid=(l+r)>>1;

build(l,mid,index<<1);

build(mid+1,r,index<<1|1);

up(index);

}

void FLS(int l,int r,int L,int R,ll c,int index,int &cur)

{

if(l==r)

{

if(tree[index].mmax>c)

cur=min(cur,tree[index].cur);

return ;

}

int mid=(l+r)>>1;

if(L<=l && R>=r)

{

if(tree[index<<1].mmax>c)

FLS(l,mid,L,R,c,index<<1,cur);

else if(tree[index<<1|1].mmax>c)

FLS(mid+1,r,L,R,c,index<<1|1,cur);

return ;

}

if(L<=mid)FLS(l,mid,L,R,c,index<<1,cur);

if(R>mid)FLS(mid+1,r,L,R,c,index<<1|1,cur);

return ;

}

int FirstLarge(int L,int R,ll c,int n)

{

int cur=INF;

FLS(1,n,L,R,c,1,cur);

return cur;

}

void init(int n)

{

build(1,n,1);//必须先建树，否则没法查询

dp\_next[n]=1;

int cur;

for(int i=n-1;i>=1;i--)

{

cur=FirstLarge(i+1,n,a[i],n);

if(cur>i && cur<=n)

{

dp\_next[i]=dp\_next[cur]+1;

}

else

dp\_next[i]=1;//说明从i~n的最长上升子序列的长度为1。

}

}

#### 11.1.2.2思想

该算法从n开始往前移动，在计算dp\_next[i]时会先从线段树中找在[i+1,n]这个区间中第一个大于他的数的位置cur，如果找到dp\_next[i]=dp\_next[cur]+1，否则，dp\_next[i]=1;

## 11.2(poj3616)

### 11.2.0 描述

给定时间间隔，问工作哪几个间隔可以收获最多。

### 11.2.1code:

#include<stdio.h>

#include<algorithm>

#include<cstring>

using namespace std;

const int maxn=1e6+10;

const int maxm=1e3+10;

typedef long long ll;

ll dp[maxm];

struct node

{

int st;

int ed;

ll de;

node(int s1=-1,int ed1=-1,ll de1=0){st=s1;ed=ed1;de=de1;}

};

bool cmp(node &a,node &b)

{

return a.st<b.st;

}

node a[maxm];

ll ans=0;

ll solve(int cur,int m,int n,int r)

{

if(dp[cur]!=-1)return dp[cur];

ll tmp=0;

for(int i=cur+1;i<=m;i++)

{

if(a[cur].ed<=a[i].st)

{

tmp=max(tmp,solve(i,m,n,r)+a[i].de);

}

}

dp[cur]=tmp;

return tmp;

}

int main()

{

int n,m,r;

scanf("%d %d %d",&n,&m,&r);

for(int i=1;i<=m;i++)

{

scanf("%d %d %lld",&a[i].st,&a[i].ed,&a[i].de);

a[i].ed=a[i].st+a[i].ed-a[i].st+r;

}

sort(a+1,a+m+1,cmp);

memset(dp,-1,sizeof(dp));

printf("%lld\n",solve(0,m,n,r));

return 0;

}

//xie dao de na ge xiao shi

### 11.2.2思路：

简单dp，根据给定的时间间隔进行排序，然后dp即可。注意休息够的那个时间点可以工作，且一旦工作就工作结束才能停止。