# 字符串

## KMP算法

inline void getNext**()** **{**

int i**=**0**;**

int j**=-**1**;**

next**[**0**]=-**1**;**

**while** **(**i**<**m**)** **{**

**if** **(**j**==-**1 **||** p**[**i**]==**p**[**j**])** **{**

i**++;**

j**++;**

next**[**i**]=**j**;**

**}** **else** **{**

j**=**next**[**j**];**

**}**

**}**

**}**

inline void getAdvancedNext**()** **{**

int i**=**0**;**

int j**=-**1**;**

next**[**0**]=-**1**;**

**while** **(**i**<**m**)** **{**

**if** **(**j**==-**1 **||** p**[**i**]==**p**[**j**])** **{**

i**++;**

j**++;**

**if** **(**p**[**i**]==**p**[**j**])** **{**

next**[**i**]=**next**[**j**];**

**}** **else** **{**

next**[**i**]=**j**;**

**}**

**}** **else** **{**

j**=**next**[**j**];**

**}**

**}**

**}**

# 区间操作

## Splay树

#include<cstdio>

#include<algorithm>

#include<queue>

#include<cstring>

**using** **namespace** std**;**

const int maxn**=**1000005**;**

long long stack**[**maxn**];**

int tot**;**

int ch**[**maxn**][**2**],** f**[**maxn**],** size**[**maxn**],** cnt**[**maxn**];**

long long key**[**maxn**];**

int sz**,** root**;**

inline void pushdown**(**int p**)** **{**

**while** **(**p**<<**1**<=**tot**)** **{**

**if** **((**p**<<**1**|**1**)>**tot**)** **{**

**if** **(**stack**[**p**<<**1**]<**stack**[**p**])** **{**

swap**(**stack**[**p**<<**1**],**stack**[**p**]);**

p**=**p**<<**1**;**

**}** **else** **{**

**break;**

**}**

**}** **else** **{**

**if** **(**stack**[**p**]>**stack**[**p**<<**1**]** **||** stack**[**p**]>**stack**[**p**<<**1**|**1**])** **{**

**if** **(**stack**[**p**<<**1**|**1**]<**stack**[**p**<<**1**])** **{**

swap**(**stack**[**p**<<**1**|**1**],**stack**[**p**]);**

p**=**p**<<**1**|**1**;**

**}** **else** **{**

swap**(**stack**[**p**<<**1**],**stack**[**p**]);**

p**=**p**<<**1**;**

**}**

**}** **else** **{**

**break;**

**}**

**}**

**}**

**return** **;**

**}**

inline void push**(**int x**)** **{**

tot**++;**

stack**[**tot**]=**x**;**

int p**=**tot**;**

**while** **(**p**!=**1**)** **{**

**if** **(**stack**[**p**>>**1**]>**stack**[**p**])** **{**

swap**(**stack**[**p**>>**1**],**stack**[**p**]);**

p**=**p**>>**1**;**

**}** **else** **{**

**break;**

**}**

**}**

**}**

inline void pop**()** **{**

stack**[**1**]=**stack**[**tot**];**

tot**--;**

pushdown**(**1**);**

**}**

inline void clear**(**int x**)** **{**

ch**[**x**][**0**]=**0**;**

ch**[**x**][**1**]=**0**;**

f**[**x**]=**0**;**

size**[**x**]=**0**;**

cnt**[**x**]=**0**;**

key**[**x**]=**0**;**

**}**

inline bool get**(**int x**)** **{**

**return** ch**[**f**[**x**]][**1**]** **==** x**;**

**}**

inline void update**(**int x**)** **{**

**if** **(**x**)** **{**

size**[**x**]** **=** cnt**[**x**];**

**if** **(**ch**[**x**][**0**])** size**[**x**]+=**size**[**ch**[**x**][**0**]];**

**if** **(**ch**[**x**][**1**])** size**[**x**]+=**size**[**ch**[**x**][**1**]];**

**}**

**}**

inline void rotate**(**int x**)** **{**

int old **=** f**[**x**],** oldf **=** f**[**old**],** whichx **=** get**(**x**);**

ch**[**old**][**whichx**]** **=** ch**[**x**][**whichx **^** 1**];**

f**[**ch**[**old**][**whichx**]]** **=** old**;**

ch**[**x**][**whichx **^** 1**]** **=** old**;**

f**[**old**]** **=** x**;**

f**[**x**]** **=** oldf**;**

**if** **(**oldf**)** **{**

ch**[**oldf**][**ch**[**oldf**][**1**]** **==** old**]** **=** x**;**

**}**

update**(**old**);**

update**(**x**);**

**}**

inline void splay**(**int x**)** **{**

**for** **(**int fa**;** fa **=** f**[**x**];** rotate**(**x**))**

**if** **(**f**[**fa**])** **{**

rotate**((**get**(**x**)** **==** get**(**fa**))** **?** fa **:** x**);**

**}**

root **=** x**;**

**}**

inline void insert**(**long long x**)** **{**

**if** **(**root **==** 0**)** **{**

sz**++;**

ch**[**sz**][**0**]** **=** ch**[**sz**][**1**]** **=** f**[**sz**]** **=** 0**;**

root **=** sz**;**

size**[**sz**]** **=** cnt**[**sz**]** **=** 1**;**

key**[**sz**]** **=** x**;**

**return;**

**}**

int now **=** root**,** fa **=** 0**;**

**while** **(**1**)** **{**

**if** **(**x **==** key**[**now**])** **{**

cnt**[**now**]++;**

update**(**now**);**

update**(**fa**);**

splay**(**now**);**

**break;**

**}**

fa **=** now**;**

now **=** ch**[**now**][**key**[**now**]** **<** x**];**

**if** **(**now **==** 0**)** **{**

sz**++;**

ch**[**sz**][**0**]** **=** ch**[**sz**][**1**]** **=** 0**;**

f**[**sz**]** **=** fa**;**

size**[**sz**]** **=** cnt**[**sz**]** **=** 1**;**

ch**[**fa**][**key**[**fa**]** **<** x**]** **=** sz**;**

key**[**sz**]** **=** x**;**

update**(**fa**);**

splay**(**sz**);**

**break;**

**}**

**}**

**}**

inline int find**(**long long x**)** **{**

int now **=** root**,** ans **=** 0**;**

**while** **(**1**)** **{**

**if** **(**x **<** key**[**now**])**

now **=** ch**[**now**][**0**];**

**else** **{**

ans **+=** **(**ch**[**now**][**0**]** **?** size**[**ch**[**now**][**0**]]** **:** 0**);**

**if** **(**x **==** key**[**now**])** **{**

splay**(**now**);**

**return** ans **+** 1**;**

**}**

ans **+=** cnt**[**now**];**

now **=** ch**[**now**][**1**];**

**}**

**}**

**}**

inline long long findx**(**int x**)** **{**

int now **=** root**;**

**while** **(**1**)** **{**

**if** **(**ch**[**now**][**0**]** **&&** x **<=** size**[**ch**[**now**][**0**]])** **{**

now **=** ch**[**now**][**0**];**

**}**

**else** **{**

int temp **=** **(**ch**[**now**][**0**]** **?** size**[**ch**[**now**][**0**]]** **:** 0**)** **+** cnt**[**now**];**

**if** **(**x **<=** temp**)** **return** key**[**now**];**

x **-=** temp**;**

now **=** ch**[**now**][**1**];**

**}**

**}**

**}**

inline int pre**()** **{**

int now **=** ch**[**root**][**0**];**

**while** **(**ch**[**now**][**1**])** now **=** ch**[**now**][**1**];**

**return** now**;**

**}**

inline int next**()** **{**

int now **=** ch**[**root**][**1**];**

**while** **(**ch**[**now**][**0**])** now **=** ch**[**now**][**0**];**

**return** now**;**

**}**

inline void del**(**long long x**)** **{**

int whatever **=** find**(**x**);**

**if** **(**cnt**[**root**]** **>** 1**)** **{**

cnt**[**root**]--;**

update**(**root**);**

**return;**

**}**

**if** **(!**ch**[**root**][**0**]** **&&** **!**ch**[**root**][**1**])** **{**

clear**(**root**);**

root **=** 0**;**

**return;**

**}**

**if** **(!**ch**[**root**][**0**])** **{**

int oldroot **=** root**;**

root **=** ch**[**root**][**1**];**

f**[**root**]** **=** 0**;**

clear**(**oldroot**);**

**return;**

**}** **else** **if** **(!**ch**[**root**][**1**])** **{**

int oldroot **=** root**;**

root **=** ch**[**root**][**0**];**

f**[**root**]** **=** 0**;**

clear**(**oldroot**);**

**return;**

**}**

int leftbig **=** pre**();**

int oldroot **=** root**;**

splay**(**leftbig**);**

ch**[**root**][**1**]** **=** ch**[**oldroot**][**1**];**

f**[**ch**[**oldroot**][**1**]]** **=** root**;**

clear**(**oldroot**);**

update**(**root**);**

**}**

inline void init**()** **{**

sz**=**0**;**

root**=**0**;**

tot**=**0**;**

**}**

int main**()** **{**

int T**;**

scanf**(**"%d"**,** **&**T**);**

**while** **(**T**--)** **{**

int n**;**

long long m**;**

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

int count**=**0**;**

long long delta**=**0**;**

init**();**

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

int x**;**

char tmp**[**10**];**

scanf**(**"%s%d"**,** tmp**,** **&**x**);**

**if** **(**tmp**[**0**]==**'I'**)** **{**

**if** **(**x**<**m**)** **{**

//count++;

**}** **else** **{**

push**(**x**-**delta**);**

insert**(**x**-**delta**);**

**}**

**}**

**if** **(**tmp**[**0**]==**'A'**)** **{**

delta**+=(**long long**)**x**;**

**}**

**if** **(**tmp**[**0**]==**'S'**)** **{**

delta**-=(**long long**)**x**;**

**while** **(**tot**>**0 **&&** **(**stack**[**1**]+**delta**)<**m**)** **{**

del**(**stack**[**1**]);**

count**++;**

pop**();**

**}**

**}**

**if** **(**tmp**[**0**]==**'Q'**)** **{**

**if** **(**tot**-**x**+**1**>**0**)** **{**

printf**(**"%lld\n"**,** findx**(**tot**-**x**+**1**)+**delta**);**

**}** **else** **{**

printf**(**"-1\n"**);**

**}**

**}**

**}**

printf**(**"%d\n"**,** count**);**

**}**

**}**

## 树状数组

void add**(**int k**,**int num**)**

**{**

**while(**k**<=**n**)**

**{**

tree**[**k**]+=**num**;**

k**+=**k**&-**k**;**

**}**

**}**

int read**(**int k**)**//1~k的区间和

**{**

int sum**=**0**;**

**while(**k**)**

**{**

sum**+=**tree**[**k**];**

k**-=**k**&-**k**;**

**}**

**return** sum**;**

**}**

## 线段树

#include<bits/stdc++.h>

**using** **namespace** std**;**

const int maxn**=**1e5**+**10**;**

int a**[**maxn**];**

int sum**[**maxn**<<**2**],**exc**[**maxn**<<**2**];**

void maintain**(**int k**)**

**{**

sum**[**k**]=**sum**[**k**<<**1**]+**sum**[**k**<<**1**|**1**];**

**}**

void pushdown**(**int lenl**,**int lenr**,**int k**)**//标记下放，并更细节点信息

**{**

**if(**exc**[**k**]){**

exc**[**k**<<**1**]=**exc**[**k**];**

exc**[**k**<<**1**|**1**]=**exc**[**k**];**

sum**[**k**<<**1**]=**exc**[**k**]\***lenl**;**

sum**[**k**<<**1**|**1**]=**exc**[**k**]\***lenr**;**

exc**[**k**]=**0**;**

**}**

**}**

void build**(**int l**,**int r**,**int k**)**

**{**

**if(**l**>**r**)**

**return** **;**

**if(**l**==**r**){**

sum**[**k**]=**a**[**l**];**

exc**[**k**]=**0**;**

**return** **;**

**}**

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

build**(**l**,**mid**,**k**<<**1**);**

build**(**mid**+**1**,**r**,**k**<<**1**|**1**);**

maintain**(**k**);**

**}**

void change**(**int l**,**int r**,**int cl**,**int cr**,**int k**,**int newp**)**

**{**

**if(**l**>**r**||**cl**>**r**||**cr**<**l**)**

**return** **;**

**if(**l**>=**cl**&&**r**<=**cr**){**

sum**[**k**]=**newp**\*(**r**-**l**+**1**);**//在发现现在区域小于需要更新区域时

exc**[**k**]=**newp**;**//更新节点的结果，并增加延迟标记exc，用于之后的标记下放

**return** **;**

**}**

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

pushdown**(**mid**-**l**+**1**,**r**-**mid**,**k**);**

change**(**l**,**mid**,**cl**,**cr**,**k**<<**1**,**newp**);**

change**(**mid**+**1**,**r**,**cl**,**cr**,**k**<<**1**|**1**,**newp**);**

maintain**(**k**);**

**}**

int query**(**int l**,**int r**,**int ql**,**int qr**,**int k**)**

**{**

**if(**l**>**r**||**ql**>**r**||**qr**<**l**)**

**return** 0**;**

**if(**l**>=**ql**&&**r**<=**qr**)**

**return** sum**[**k**];**

int mid**=(**l**+**r**)>>**1**,**ans**=**0**;**

pushdown**(**mid**-**l**+**1**,**r**-**mid**,**k**);**//每一层询问执行到这一步，为了下一次递归更新叶节点信息

**if(**mid**>=**l**)**

ans**+=**query**(**l**,**mid**,**ql**,**qr**,**k**<<**1**);**

**if(**mid**<**r**)**

ans**+=**query**(**mid**+**1**,**r**,**ql**,**qr**,**k**<<**1**|**1**);**

**return** ans**;**

**}**

int main**()**

**{**

ios**::**sync\_with\_stdio**(false);**

//freopen("in.txt","r",stdin);

int n**,**m**,**cmd**,**l**,**r**,**newp**;**

cin**>>**n**;**

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

cin**>>**a**[**i**];**

build**(**1**,**n**,**1**);**

cin**>>**m**;**

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

cin**>>**cmd**>>**l**>>**r**;**

**if(**cmd**){**

cin**>>**newp**;**

change**(**1**,**n**,**l**,**r**,**1**,**newp**);**

**}else**

cout**<<**query**(**1**,**n**,**l**,**r**,**1**)<<**endl**;**

**}**

**return** 0**;**

**}**

## 左偏树

#include <cstdio>

**using** **namespace** std**;**

const int N**=**1000000**;**

void swap**(**int **&**x**,**int **&**y**){**int t**=**x**;**x**=**y**,**y**=**t**;}**

int ch**[**N**][**2**],**val**[**N**],**dis**[**N**],**f**[**N**];**

int n**,**m**;**

int merge**(**int x**,**int y**)**

**{**

**if** **(**x**==**0 **||** y**==**0**)**

**return** x**+**y**;**

**if** **(**val**[**x**]<**val**[**y**]** **||** **(**val**[**x**]==**val**[**y**]** **&&** x**>**y**))**

swap**(**x**,**y**);**

ch**[**x**][**1**]=**merge**(**ch**[**x**][**1**],**y**);**

f**[**ch**[**x**][**1**]]=**x**;**

**if** **(**dis**[**ch**[**x**][**0**]]<**dis**[**ch**[**x**][**1**]])**

swap**(**ch**[**x**][**0**],**ch**[**x**][**1**]);**

dis**[**x**]=**dis**[**ch**[**x**][**1**]]+**1**;**

**return** x**;**

**}**

int getf**(**int x**)** **{**

**while(**f**[**x**])** x**=**f**[**x**];**

**return** x**;**

**}**

int pop**(**int x**)** **{**

val**[**x**]=-**1**;**

f**[**ch**[**x**][**0**]]=**0**;**

f**[**ch**[**x**][**1**]]=**0**;**

int lc **=** ch**[**x**][**0**];**

int rc **=** ch**[**x**][**1**];**

ch**[**x**][**0**]=**0**;**

ch**[**x**][**1**]=**0**;**

dis**[**x**]=**0**;**

**return** merge**(**lc**,**rc**);**

**}**

int change**(**int x**)**

**{**

val**[**x**]=**val**[**x**]/**2**;**

f**[**ch**[**x**][**0**]]=**0**;**

f**[**ch**[**x**][**1**]]=**0**;**

int lc **=** ch**[**x**][**0**];**

int rc **=** ch**[**x**][**1**];**

ch**[**x**][**0**]=**0**;**

ch**[**x**][**1**]=**0**;**

dis**[**x**]=**0**;**

**return** merge**(**x**,**merge**(**lc**,**rc**));**

**}**

int solve**(**int x**,**int y**)** **{**

int xx **=** getf**(**x**);**

int yy **=** getf**(**y**);**

**if** **(**xx**==**yy**)** **{**

**return** **-**1**;**

**}**

**return** val**[**merge**(**change**(**xx**),** change**(**yy**))];**

**}**

main**()** **{**

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

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

ch**[**i**][**0**]=**ch**[**i**][**1**]=**0**;**

f**[**i**]=**0**;**

dis**[**i**]=**0**;**

**}**

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

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

**}**

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

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

int x**,**y**;**

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

printf**(**"%d\n"**,** solve**(**x**,**y**));**

**}**

**}**

**return** 0**;**

**}**

# 图

## 最小树形图

#include <iostream>

#include <cmath>

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

#include <string>

**typedef** long long LL**;**

**using** **namespace** std**;**

const int MAXV **=** 100**;**

const int MAXE **=** 10000**;**

const int INF **=** 0x3f3f3f3f**;**

//求具有V个点,以root为根节点的图map的最小树形图

int zhuliu**(**int root**,** int V**,** int map**[**MAXV **+** 7**][**MAXV **+** 7**]){**

bool visited**[**MAXV **+** 7**];**

bool flag**[**MAXV **+** 7**];**//缩点标记为true,否则仍然存在

int pre**[**MAXV **+** 7**];**//点i的父节点为pre[i]

int sum **=** 0**;**//最小树形图的权值

int i**,** j**,** k**;**

**for(**i **=** 0**;** i **<=** V**;** i**++)** flag**[**i**]** **=** **false,** map**[**i**][**i**]** **=** INF**;**

pre**[**root**]** **=** root**;**

**while(true){**

**for(**i **=** 1**;** i **<=** V**;** i**++){**//求最短弧集合E0

**if(**flag**[**i**]** **||** i **==** root**)** **continue;**

pre**[**i**]** **=** i**;**

**for(**j **=** 1**;** j **<=** V**;** j**++)**

**if(!**flag**[**j**]** **&&** map**[**j**][**i**]** **<** map**[**pre**[**i**]][**i**])**

pre**[**i**]** **=** j**;**

**if(**pre**[**i**]** **==** i**)** **return** **-**1**;**

**}**

**for(**i **=** 1**;** i **<=** V**;** i**++){**//检查E0

**if(**flag**[**i**]** **||** i **==** root**)** **continue;**

**for(**j **=** 1**;** j **<=** V**;** j**++)** visited**[**j**]** **=** **false;**

visited**[**root**]** **=** **true;**

j **=** i**;**//从当前点开始找环

**do{**

visited**[**j**]** **=** **true;**

j **=** pre**[**j**];**

**}while(!**visited**[**j**]);**

**if(**j **==** root**)continue;**//没找到环

i **=** j**;**//收缩G中的有向环

**do{**//将整个环的取值保存,累计计入原图的最小树形图

sum **+=** map**[**pre**[**j**]][**j**];**

j **=** pre**[**j**];**

**}while(**j **!=** i**);**

j **=** i**;**

**do{**//对于环上的点有关的边,修改其权值

**for(**k **=** 1**;** k **<=** V**;** k**++)**

**if(!**flag**[**k**]** **&&** map**[**k**][**j**]** **<** INF **&&** k **!=** pre**[**j**])**

map**[**k**][**j**]** **-=** map**[**pre**[**j**]][**j**];**

j **=** pre**[**j**];**

**}while(**j **!=** i**);**

**for(**j **=** 1**;** j **<=** V**;** j**++){**//缩点,将整个环缩成i号点,所有与环上的点有关的边转移到点i

**if(**j **==** i**)** **continue;**

**for(**k **=** pre**[**i**];** k **!=** i**;** k **=** pre**[**k**]){**

**if(**map**[**k**][**j**]** **<** map**[**i**][**j**])** map**[**i**][**j**]** **=** map**[**k**][**j**];**

**if(**map**[**j**][**k**]** **<** map**[**j**][**i**])** map**[**j**][**i**]** **=** map**[**j**][**k**];**

**}**

**}**

**for(**j **=** pre**[**i**];** j **!=** i**;** j **=** pre**[**j**])** flag**[**j**]** **=** **true;**//标记环上其他点为被缩掉

**break;**//当前环缩点结束,形成新的图G',跳出继续求G'的最小树形图

**}**

**if(**i **>** V**){**//如果所有的点都被检查且没有环存在,现在的最短弧集合E0就是最小树形图.累计计入sum,算法结束

**for(**i **=** 1**;** i **<=** V**;** i**++)**

**if(!**flag**[**i**]** **&&** i **!=** root**)** sum **+=** map**[**pre**[**i**]][**i**];**

**break;**

**}**

**}**

**return** sum**;**

**}**

## 强连通tarjan

#include<cstdio>

#include<algorithm>

**using** **namespace** std**;**

const int maxn**=**1000000**;**

const int maxm**=**1000000**;**

struct Edge**{**

int u**;**

int v**;**

int next**;**

**}** edge**[**maxm**];**

int tot**;**

int head**[**maxn**];**

int dfn**[**maxn**];**

int low**[**maxn**];**

int vis**[**maxn**];**

int col**[**maxn**];** //强连通分量染色块

int stack**[**maxn**];**

int top**;**

int clk**;**

int coln**;** //强连通分量个数

inline void init**(**int n**)** **{**

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

low**[**i**]=**0**;**

dfn**[**i**]=**0**;**

vis**[**i**]=false;**

head**[**i**]=**0**;**

**}**

top**=**0**;**

tot**=**0**;**

clk**=**0**;**

coln**=**0**;**

**}**

inline void addedge**(**int x**,**int y**)** **{**

edge**[++**tot**].**next**=**head**[**x**];**

edge**[**tot**].**u**=**x**;**

edge**[**tot**].**v**=**y**;**

head**[**x**]=**tot**;**

**}**

void tarjan**(**int x**)** **{**

clk**++;**

dfn**[**x**]=**clk**;**

low**[**x**]=**clk**;**

top**++;**

stack**[**top**]=**x**;**

vis**[**x**]=true;**

**for** **(**int p**=**head**[**x**];** p**!=**0**;** p**=**edge**[**p**].**next**)** **{**

int u**=**x**;**

int v**=**edge**[**p**].**v**;**

**if** **(!**dfn**[**v**])** **{**

tarjan**(**v**);**

low**[**u**]=**min**(**low**[**u**],**low**[**v**]);**

**}** **else** **if** **(**vis**[**v**])** **{**

low**[**u**]=**min**(**low**[**u**],**dfn**[**v**]);**

**}**

**}**

**if** **(**dfn**[**x**]==**low**[**x**])** **{**

coln**++;**

**do** **{**

vis**[**stack**[**top**]]=false;**

col**[**stack**[**top**]]=**coln**;**

top**--;**

**}while** **(**stack**[**top**+**1**]!=**x**);**

**}**

**}**

## 最短路dijkstra

//Til the Cows Come Home

#include<cstdio>

#include<queue>

**using** **namespace** std**;**

const int maxn**=**1000000**;**

const int INF**=**1E9**;**

struct Edge**{**

int next**;**

int to**;**

int w**;**

**}** edge**[**maxn**];**

int head**[**maxn**];**

bool vis**[**maxn**];**

int tot**;**

struct Node**{**

int dis**;**

int index**;**

bool **operator** **<** **(**const Node **&**x**)** const **{**

**return** dis**>**x**.**dis**;**

**}**

**}** node**[**maxn**];**

priority\_queue **<**Node**>** q**;**

int f**[**maxn**];**

int n**,**m**;**

void init**()** **{**

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

head**[**i**]=**0**;**

node**[**i**].**index**=**i**;**

**}**

tot**=**0**;**

**}**

inline void addedge**(**int x**,** int y**,** int w**)** **{**

edge**[++**tot**].**next**=**head**[**x**];**

edge**[**tot**].**to**=**y**;**

edge**[**tot**].**w**=**w**;**

head**[**x**]=**tot**;**

**}**

void dijkstra**(**int x**)** **{**

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

vis**[**i**]=false;**

node**[**i**].**dis**=**INF**;**

**}**

**while** **(!**q**.**empty**())** **{**

q**.**pop**();**

**}**

node**[**x**].**dis**=**0**;**

q**.**push**(**node**[**x**]);**

**while** **(!**q**.**empty**())** **{**

Node top\_node**=**q**.**top**();**

int u**=**top\_node**.**index**;**

q**.**pop**();**

**if** **(**vis**[**u**])** **{**

**continue;**

**}**

vis**[**u**]=true;**

**for** **(**int p**=**head**[**u**];**p**!=**0**;**p**=**edge**[**p**].**next**)** **{**

int v**=**edge**[**p**].**to**;**

**if** **(**node**[**v**].**dis**>**node**[**u**].**dis**+**edge**[**p**].**w**)** **{**

node**[**v**].**dis**=**node**[**u**].**dis**+**edge**[**p**].**w**;**

q**.**push**(**node**[**v**]);**

**}**

**}**

**}**

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

f**[**i**]=**node**[**i**].**dis**;**

**}**

**}**

int main**()** **{**

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

init**();**

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

int x**,**y**,**z**;**

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

addedge**(**x**,**y**,**z**);**

addedge**(**y**,**x**,**z**);**

**}**

dijkstra**(**1**);**

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

**return** 0**;**

**}**

# 网络流

## 最大流最小割(最大流等于最小割权值和)

#include<bits/stdc++.h>

**using** **namespace** std**;**

const int maxn**=**60005**;**

const int maxm**=**400000**;**

const int INF**=**0x3f3f3f3f**;**

int n**,**m**;**

int st**,**en**;**

struct Edge**{**

int next**;**

int to**;**

int val**;**

**}** e**[**maxm**\***2**];**

int head**[**maxn**];**

int cur**[**maxn**];**

int dis**[**maxn**];**

int tot**;**

int dfs**(**int u**,**int flow**)** **{**

**if** **(**u**==**en**)** **{return** flow**;}**

**for** **(**int **&**i**=**cur**[**u**];**i**!=-**1**;**i**=**e**[**i**].**next**){**

**if** **((**dis**[**e**[**i**].**to**]** **==** dis**[**u**]+**1**)** **&&** **(**e**[**i**].**val**!=**0**))** **{**

int di**=**dfs**(**e**[**i**].**to**,** min**(**flow**,** e**[**i**].**val**));**

**if** **(**di**>**0**)** **{**

e**[**i**].**val**-=**di**;**

e**[**i**^**1**].**val**+=**di**;**

**return** di**;**

**}**

**}**

**}**

**return** 0**;**

**}**

int bfs**()** **{**

queue**<**int**>** Q**;**

**while** **(!**Q**.**empty**())** **{**Q**.**pop**();}**

memset**(**dis**,**0**,sizeof(**dis**));**

dis**[**st**]=**1**;**

Q**.**push**(**st**);**

**do** **{**

int u**=**Q**.**front**();**

Q**.**pop**();**

**for** **(**int i**=**head**[**u**];**i**!=-**1**;**i**=**e**[**i**].**next**)** **{**

**if** **((**dis**[**e**[**i**].**to**]==**0**)** **&&** **(**e**[**i**].**val**>**0**))** **{**

dis**[**e**[**i**].**to**]** **=** dis**[**u**]** **+**1**;**

Q**.**push**(**e**[**i**].**to**);**

**}**

**}**

**}** **while** **(!**Q**.**empty**());**

**if** **(**dis**[**en**]>**0**)** **{**

**return** 1**;**

**}**

**return** 0**;**

**}**

int Dinic**(){**

int ans**=**0**;**

**while** **(**bfs**())** **{**

**for** **(**int i**=**0**;**i**<=**n**+**m**+**2**;**i**++)** **{**

cur**[**i**]=**head**[**i**];**

**}**

**while** **(**int d**=**dfs**(**st**,** INF**))** **{**

ans**+=**d**;**

**}**

**}**

**return** ans**;**

**}**

inline void addedge**(**int x**,**int y**,**int c**)** **{**

e**[**tot**].**next**=**head**[**x**];**

head**[**x**]=**tot**;**

e**[**tot**].**val**=**c**;**

e**[**tot**].**to**=**y**;**

tot**++;**

e**[**tot**].**next**=**head**[**y**];**

head**[**y**]=**tot**;**

e**[**tot**].**to**=**x**;**

e**[**tot**].**val**=**0**;**

tot**++;**

**return** **;**

**}**

inline void init**()** **{**

tot**=**0**;**

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

head**[**i**]=-**1**;**

**}**

**}**

# 数学

## 阶乘逆元打表(费马小定理)

const long long mod**=**1000000007**;**

const int maxn**=**100000**;**

long long fac**[**maxn**+**9**],**inv\_fac**[**maxn**+**9**];**

long long quickpow**(**long long x**,**long long n**)** **{**

long long res**=**1**;**

x**=**x**%**mod**;**

**while(**n**)**

**{**

**if(**n**%**2**==**1**)**res**=(**res**\***x**)%**mod**;**

n**=**n**/**2**;**

x**=(**x**\***x**)%**mod**;**

**}**

**return** res**;**

**}**

void print**()** **{**

fac**[**0**]=**1**;**

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

fac**[**i**]=(**fac**[**i**-**1**]\***i**)%**mod**;**//阶乘取余打表

**}**

//切记,求阶乘逆元时maxn最大值为mod-1，因为用这个公式时要保证待求逆元的数（此处为n!）要和mod互质。

inv\_fac**[**maxn**]=**quickpow**(**fac**[**maxn**],**mod**-**2**);**//最大阶乘逆元

**for(**int i**=**maxn**-**1**;**i**>=**0**;**i**--)** **{**

inv\_fac**[**i**]=(**inv\_fac**[**i**+**1**]\*(**i**+**1**))%**mod**;**//递推阶乘逆元

**}**

**}**

/\*

费马小定理

假如p是质数，且gcd(a,p)=1，那么 a(p-1)≡1（mod p）

如果求解 a / b mod m 求 b 的模m乘法逆元，若b ， m互质

则 k\* b mod m = 1 且 b ^ (m-1) mod m = 1 所以 k\* b = b ^ (m-1)

可直接得出b的模m乘法逆元为 b ^ (m-2)

在算法中常模1e9+7为质数，可用费马小定理转换。

\*/

# 动态规划

## 数位DP

#include<iostream>

#include<cstdio>

#include<cstring>

#include<string>

**using** **namespace** std**;**

**typedef** long long ll**;**

int a**[**20**];**

ll dp**[**20**][**4**];**

ll dfs**(**int pos**,**int pre**,**int sta**,**bool limit**)**//底下这个是求没有38和49子序列的个数的

**{**

**if(**pos**==-**1**)** **return** 1**;**

**if(!**limit **&&** dp**[**pos**][**sta**]!=-**1**)** **return** dp**[**pos**][**sta**];**

int up**=**limit **?** a**[**pos**]** **:** 9**;**

ll tmp**=**0**;**

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

**{**

**if(**sta**==**1 **&&** i**==**9**)continue;**

**if(**sta**==**2 **&&** i**==**8**)continue;**

**if(**sta**==**3 **&&(**i**==**8**||**i**==**9**))** **continue;**

**if(**i**==**4**&&**sta**==**0**){**

tmp**+=**dfs**(**pos**-**1**,**i**,**1**,**limit **&&** i**==**a**[**pos**]);**

**}else** **if(**i**==**3**&&**sta**==**0**){**

tmp**+=**dfs**(**pos**-**1**,**i**,**2**,**limit **&&** i**==**a**[**pos**]);**

**}else** **if((**i**==**3**&&**sta**==**1**)||(**i**==**4**&&**sta**==**2**)){**

tmp**+=**dfs**(**pos**-**1**,**i**,**3**,**limit **&&** i**==**a**[**pos**]);**

**}**

**else{**

tmp**+=**dfs**(**pos**-**1**,**i**,**sta**,**limit **&&** i**==**a**[**pos**]);**

**}**

**}**

**if(!**limit**)** dp**[**pos**][**sta**]=**tmp**;**

**return** tmp**;**

**}**

ll solve**(**ll x**)**

**{**

ll pos**=**0**;**

**while(**x**)**

**{**

a**[**pos**++]=**x**%**10**;**

x**/=**10**;**

**}**

**return** dfs**(**pos**-**1**,-**1**,**0**,true);**

**}**

int main**()**

**{**

int T**;**long long ri**;**

scanf**(**"%d"**,&**T**);**

//memset(dp,-1,sizeof dp);可优化

**while(**T**--)**

**{**

scanf**(**"%lld"**,&**ri**);**

memset**(**dp**,-**1**,sizeof** dp**);**

printf**(**"%lld\n"**,(**ri**-**solve**(**ri**)+**1**));**

**}**

**return** 0**;**

**}**