Talancode

fstqwq模板

1. **LCT**
2. **FFT**
3. **Splay Hash**
4. **Suffix Array**
5. **Pollard Rho**
6. **点分治**
7. **K短路**
8. **SLFSPFA**
9. **支配树**
10. **KM**
11. **带花树**
12. **Simplex**
13. **最小表示法**
14. **树哈希**
15. **ZKW费用流**
16. **无向图最小割**
17. **KDT**
18. **二次剩余**
19. **原根**
20. **Burnside**

#pragma GCC optimize("-O3")

#define \_\_ \_\_attribute\_\_((optimize("-O3")))

#define inline \_\_ \_\_inline \_\_attribute\_\_((\_\_always\_inline\_\_, \_\_artificial\_\_))

source $VIMRUNTIME/vimrc\_example.vim

source $VIMRUNTIME/mswin.vim

behave mswin

set hlsearch ci ai si nu ts=4 sw=4

color slate

map <F9> : ! g++ % -o %< -g -Wall -Wextra -Wconversion -Wshadow -std=c++11 <CR>

map <F5> : ! time ./%< <CR>

map <F8> : ! time ./%< < %<.in <CR>

map <F12> :r std.cpp <CR> ggdd : 15 <CR>

# 林克卡特树

struct Node **{**

Node **\***fa**,** **\***top**,** **\***c**[**2**],** **\***mi**;**

int val**;** bool rev**,** isrt**;**

Node **()** **:** mi**(this),** val**(**INF**)** **{}**

void setc**(**Node **\***ch**,** int p**)** **{**c**[**p**]** **=** ch**;** ch**->**fa **=** **this;}**

bool d**()** **{return** fa**->**c**[**1**]** **==** **this;}**

void upd**()** **{**

mi **=** **this;**

**if** **(**c**[**0**]->**mi**->**val **<** mi**->**val**)** mi **=** c**[**0**]->**mi**;**

**if** **(**c**[**1**]->**mi**->**val **<** mi**->**val**)** mi **=** c**[**1**]->**mi**;**

**}**

void revIt**()** **{**

rev **^=** 1**;** swap**(**c**[**0**],** c**[**1**]);**

**}**

void relax**()** **{**

**if** **(**rev**)** **{**

c**[**0**]->**revIt**();** c**[**1**]->**revIt**();** rev **=** 0**;**

**}**

**}**

void setrt**(**Node **\***f**);**

**}** Tnull**,** **\***null **=** **&**Tnull**,** mem**[**MAXN **+** MAXM**],** **\***ptr **=** mem**;**

void Node**::**setrt**(**Node **\***f**)** **{**top **=** f**;** fa **=** null**;** isrt **=** 1**;}**

Node **\***newNode**(**int v **=** INF**)** **{**

ptr**->**fa **=** ptr**->**top **=** ptr**->**c**[**0**]** **=** ptr**->**c**[**1**]** **=** null**;**

ptr**->**mi **=** ptr**;** ptr**->**rev **=** 0**;** ptr**->**isrt **=** 1**;** ptr**->**val **=** v**;**

**return** ptr**++;**

**}**

void rot**(**Node **\***x**)** **{**

Node **\***fa **=** x**->**fa**;** bool d **=** x**->**d**();**

fa**->**fa**->**setc**(**x**,** fa**->**d**());**

fa**->**setc**(**x**->**c**[!**d**],** d**);**

x**->**setc**(**fa**,** **!**d**);**

fa**->**upd**();**

**if** **(**fa**->**isrt**)** **{**

fa**->**isrt **=** 0**;** x**->**isrt **=** 1**;** x**->**top **=** fa**->**top**;**

**}**

**}**

void pushto**(**Node **\***x**)** **{**

static Node **\***st**[**MAXN**];** int top **=** 0**;**

**while** **(**x **!=** null**)** st**[**top**++]** **=** x**,** x **=** x**->**fa**;**

**while** **(**top**--)** st**[**top**]->**relax**();**

**}**

void splay**(**Node **\***x**)** **{**

pushto**(**x**);**

**while** **(**x**->**fa **!=** null**)** **{**

**if** **(**x**->**fa**->**fa **==** null**)** rot**(**x**);**

**else** x**->**d**()** **==** x**->**fa**->**d**()** **?** **(**rot**(**x**->**fa**),** rot**(**x**))** **:** **(**rot**(**x**),** rot**(**x**));**

**}**

x**->**upd**();**

**}**

void expose**(**Node **\***x**)** **{**

static Node **\***y**;**

**for** **(**y **=** null**;** x **!=** null**;** y **=** x**,** x **=** x**->**top**)** **{**

splay**(**x**);**

x**->**c**[**1**]->**setrt**(**x**);** x**->**setc**(**y**,** 1**);**

**}**

//return y;

**}**

void mkrt**(**Node **\***x**)** **{**

expose**(**x**);** splay**(**x**);** x**->**revIt**();**

**}**

void Link**(**Node **\***u**,** Node **\***v**)** **{**

mkrt**(**u**);** u**->**top **=** v**;**

**}**

void Cut**(**Node **\***u**,** Node **\***v**)** **{**

mkrt**(**u**);** expose**(**v**);** splay**(**u**);**

u**->**setc**(**null**,** 1**);** u**->**upd**();**

v**->**setrt**(**null**);**

**}**

Node **\***query**(**Node **\***u**,** Node **\***v**)** **{**

mkrt**(**u**);** expose**(**v**);** splay**(**v**);**

**return** v**->**mi**;**

**}**

Node **\***getrt**(**Node **\***x**)** **{**

expose**(**x**);** splay**(**x**);**

**while** **(**x**->**c**[**0**]** **!=** null**)** x **=** x**->**c**[**0**];**

splay**(**x**);**

**return** x**;**

**}**

struct Edge **{**

int u**,** v**,** w**;**

bool **operator** **<** **(**const Edge **&**a**)** const **{**

**return** w **<** a**.**w**;**

**}**

**}** e**[**MAXM**];**

int n**,** m**,** ans **=** INF**;**

Node **\***a**[**MAXN**];**

bool vis**[**MAXM**];**

int E**[**MAXN **+** MAXM**];**

void newE**(**int i**)** **{**

Node **\***mid **=** newNode**(**e**[**i**].**w**);**

Link**(**mid**,** a**[**e**[**i**].**u**]);** Link**(**mid**,** a**[**e**[**i**].**v**]);**

E**[**mid **-** mem**]** **=** i**;**

vis**[**i**]** **=** 1**;**

**}**

void delE**(**Node **\***mid**)** **{**

int i **=** E**[**mid **-** mem**];**

vis**[**i**]** **=** 0**;**

int u **=** e**[**i**].**u**,** v **=** e**[**i**].**v**;**

Cut**(**mid**,** a**[**u**]);** Cut**(**mid**,** a**[**v**]);**

**}**

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

read**(**n**);** read**(**m**);**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** a**[**i**]** **=** newNode**();**

**for** **(**int i **=** 1**;** i **<=** m**;** i**++)** read**(**e**[**i**].**u**),** read**(**e**[**i**].**v**),** read**(**e**[**i**].**w**);**

sort**(**e **+** 1**,** e **+** m **+** 1**);**

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

**if** **(**getrt**(**a**[**e**[**i**].**u**])** **!=** getrt**(**a**[**e**[**i**].**v**]))** **{**

cnt**++;**

newE**(**i**);**

**}**

**else** **if** **(**e**[**i**].**u **!=** e**[**i**].**v**){**

Node **\***mi **=** query**(**a**[**e**[**i**].**u**],** a**[**e**[**i**].**v**]);**

delE**(**mi**);**

newE**(**i**);**

**}**

**while** **(!**vis**[**j**])** j**++;**

**if** **(**cnt **==** n **-** 1**)** ans **=** min**(**ans**,** e**[**i**].**w **-** e**[**j**].**w**);**

**}**

printf**(**"%d\n"**,** ans **==** INF **?** **-**1 **:** ans**);**

**return** 0**;**

**}**

# 法法塔

#define MAX 262144

**typedef** double D**;**

struct Comp **{**

D r**,** i**;**

Comp**(**D x **=** 0**,** D y **=** 0**)** **:** r**(**x**),** i**(**y**)** **{}**

Comp **operator** **+** **(**Comp b**)** **{return** Comp**(**r **+** b**.**r**,** i **+** b**.**i**);}**

Comp **operator** **-** **(**Comp b**)** **{return** Comp**(**r **-** b**.**r**,** i **-** b**.**i**);}**

Comp **operator** **\*** **(**Comp b**)** **{return** Comp**(**r **\*** b**.**r **-** i **\*** b**.**i**,** r **\*** b**.**i **+** i **\*** b**.**r**);}**

**};**

int R**[**MAX **\*** 2**];**

void getR**(**int n**)** **{**

int b **=** 0**;**

**for** **(**int i **=** 2**;** i **<** n**;** i **<<=** 1**,** b**++);**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** R**[**i**]** **=** **(**R**[**i **>>** 1**]** **>>** 1**)** **|** **((**i **&** 1**)** **<<** b**);**

**}**

void FFT**(**Comp a**[],** int n**,** int rev**)** **{**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** **if** **(**R**[**i**]** **>** i**)** swap**(**a**[**R**[**i**]],** a**[**i**]);**

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

Comp step**(**cos**(**M\_PI **/** i**),** rev **\*** sin**(**M\_PI **/** i**));**

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

Comp cur**(**1**,** 0**);**

**for** **(**int k **=** 0**;** k **<** i**;** k**++,** cur **=** cur **\*** step**)** **{**

Comp x **=** a**[**j **+** k**],** y **=** cur **\*** a**[**j **+** k **+** i**];**

a**[**j **+** k**]** **=** x **+** y**,** a**[**j **+** k **+** i**]** **=** x **-** y**;**

**}**

**}**

**}**

**if** **(**rev **==** **-**1**)** **for** **(**int i **=** 0**;** i **<** n**;** i**++)** a**[**i**].**r **/=** n**;**

**}**

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

char sa**[**MAX**],** sb**[**MAX**];**

Comp a**[**MAX **\*** 2**],** b**[**MAX **\*** 2**];**

int ans**[**MAX **\*** 2**];**

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

scanf**(**"%s%s"**,** sa**,** sb**);**

n **=** **(**int**)**strlen**(**sa**),** m **=** **(**int**)**strlen**(**sb**);**

**for** **(**int i **=** 0**;** i **<** n**;** i**++)** a**[**i**]** **=** Comp**(**sa**[**n **-** i **-** 1**]** **-** '0'**);**

**for** **(**int i **=** 0**;** i **<** m**;** i**++)** b**[**i**]** **=** Comp**(**sb**[**m **-** i **-** 1**]** **-** '0'**);**

n **=** max**(**n**,** m**);**

**for** **(**int i **=** 1**;** **;** i **<<=** 1**)** **if** **(**i **>=** n**)** **{**n **=** i **\*** 2**;** **break;}**

getR**(**n**);**

FFT**(**a**,** n**,** 1**);** FFT**(**b**,** n**,** 1**);**

**for** **(**int i **=** n**;** i**--;** a**[**i**]** **=** a**[**i**]** **\*** b**[**i**]);**

FFT**(**a**,** n**,** **-**1**);**

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

ans**[**i**]** **+=** int**(**a**[**i**].**r **+** 0.5**);**

ans**[**i **+** 1**]** **+=** ans**[**i**]** **/** 10**;**

ans**[**i**]** **%=** 10**;**

**}**

bool flag **=** 0**;**

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

**if** **(**ans**[**i**]** **>** 0 **||** flag**)** flag **=** 1**,** putchar**(**'0' **+** ans**[**i**]);**

**}**

**if** **(!**flag**)** putchar**(**'0'**);**

**return** 0**;**

**}**

# Spaly 蛤习

1、询问。语法：Qxy，x,y均为正整数。功能：计算LCP(x,y)限制：1<=x,y<=当前字符串长度。  
2、修改。语法：Rxd，x是正整数，d是字符。功能：将字符串中第x个数修改为字符d。限制：x不超过当前字符串长度。  
3、插入。语法：Ixd，x是非负整数，d是字符。功能：在字符串第x个字符之后插入字符d，如果x=0，则在字符串开头插入。

const int C **=** /\*2\*/1**;**

const LL P**[**C**]** **=** **{**811/\*, 911\*/**},** MOD**[**C**]** **=** **{**2123456767/\*, 2147483647\*/**};**

LL X**[**C**][**MAXN**];**

struct Ha **{**

LL v**[**C**];**

Ha **(**LL val **=** 0**)** **{**

**for** **(**int i **=** 0**;** i **<** C**;** i**++)** v**[**i**]** **=** val **%** P**[**i**];**

**}**

Ha append**(**LL y**)** **{**

Ha ret**;**

**for** **(**int i **=** 0**;** i **<** C**;** i**++)** ret**.**v**[**i**]** **=** **(**v**[**i**]** **\*** P**[**i**]** **+** y**)** **%** MOD**[**i**];**

**return** ret**;**

**}**

Ha merge**(**Ha **&**x**,** LL len**)** **{**

Ha ret**;**

**for** **(**int i **=** 0**;** i **<** C**;** i**++)** ret**.**v**[**i**]** **=** **(**v**[**i**]** **\*** X**[**i**][**len**]** **+** x**.**v**[**i**])** **%** MOD**[**i**];**

**return** ret**;**

**}**

bool **operator** **==** **(**Ha a**)** const **{**

**for** **(**int i **=** 0**;** i **<** C**;** i**++)** **if** **(**v**[**i**]** **!=** a**.**v**[**i**])** **return** 0**;**

**return** 1**;**

**}**

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

printf**(**"{%lld"**,** v**[**0**]);**

**for** **(**int i **=** 1**;** i **<** C**;** i**++)** printf**(**", %lld"**,** v**[**i**]);**

printf**(**"}\n"**);**

**}**

**};**

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

**for** **(**int j **=** 0**;** j **<** C**;** j**++)** X**[**j**][**0**]** **=** 1**;**

**for** **(**int j **=** 0**;** j **<** C**;** j**++)** **for** **(**int i **=** 1**;** i **<** MAXN**;** i**++)** X**[**j**][**i**]** **=** X**[**j**][**i **-** 1**]** **\*** P**[**j**]** **%** MOD**[**j**];**

**}**

struct Node **{**

Node **\***fa**,** **\***c**[**2**];**

int siz**;** LL val**;** Ha ha**;**

Node **()** **:** siz**(**0**),** val**(**0**)** **{}**

bool dir**()** **{**

**return** fa**->**c**[**1**]** **==** **this;**

**}**

void setc**(**Node **\***ch**,** int p**)** **{**

c**[**p**]** **=** ch**;**

ch**->**fa **=** **this;**

**}**

void upd**()** **{**

siz **=** c**[**0**]->**siz **+** c**[**1**]->**siz **+** 1**;**

ha **=** **((**c**[**0**]->**ha**).**append**(**val**)).**merge**(**c**[**1**]->**ha**,** c**[**1**]->**siz**);**

**}**

**}** mem**[**MAXN **\*** 2**],** **\***ptr **=** mem**,** Tnull**,** **\***null **=** **&**Tnull**,** **\***root **=** null**;**

Node **\***newNode**(**LL v**)** **{**

ptr**->**c**[**0**]** **=** ptr**->**c**[**1**]** **=** ptr**->**fa **=** null**;**

ptr**->**val **=** v**;** ptr**->**siz **=** 1**;** ptr**->**ha **=** Ha**(**v**);**

**return** ptr**++;**

**}**

char s**[**MAXN**];**

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

Node **\***build**(**int l**,** int r**)** **{**

**if** **(**l **>** r**)** **return** null**;**

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

Node **\***x **=** newNode**(**s**[**mid**]);**

x**->**setc**(**build**(**l**,** mid **-** 1**),** 0**);**

x**->**setc**(**build**(**mid **+** 1**,** r**),** 1**);**

x**->**upd**();**

**return** x**;**

**}**

void rot**(**Node **\***x**)** **{**

Node **\***fa **=** x**->**fa**;**

int dir **=** x**->**dir**();**

fa**->**fa**->**setc**(**x**,** fa**->**dir**());**

fa**->**setc**(**x**->**c**[!**dir**],** dir**);**

x**->**setc**(**fa**,** **!**dir**);**

fa**->**upd**();**

**if** **(**fa **==** root**)** root **=** x**;**

**}**

void splay**(**Node **\***x**,** Node **\***fa **=** null**)** **{**

**while** **(**x**->**fa **!=** fa**)** **{**

**if** **(**x**->**fa**->**fa **==** fa**)** rot**(**x**);**

**else** x**->**dir**()** **==** x**->**fa**->**dir**()** **?** **(**rot**(**x**->**fa**),** rot**(**x**))** **:** **(**rot**(**x**),** rot**(**x**));**

**}**

x**->**upd**();**

**}**

Node**\*** select**(**int k**)** **{**

**for** **(**Node **\***x **=** root**;** **;** **)** **{**

int ls **=** x**->**c**[**0**]->**siz**;**

**if** **(**k **<** ls**)** x **=** x**->**c**[**0**];**

**else** **if** **(**k **>** ls**)** k **-=** ls **+** 1**,** x **=** x**->**c**[**1**];**

**else** **return** x**;**

**}**

**}**

Node**\*** get**(**int l**,** int r**)** **{** // [L, R)

Node **\***L **=** select**(**l **-** 1**),** **\***R **=** select**(**r**);**

splay**(**L**);**

splay**(**R**,** L**);**

**return** R**;**

**}**

void modify**(**int p**,** char c**)** **{**

Node **\***x **=** get**(**p**,** p **+** 1**);**

x**->**c**[**0**]->**val **=** c**;**

x**->**c**[**0**]->**upd**(),** x**->**upd**(),** root**->**upd**();**

**}**

void insert**(**int x**)** **{**

int len **=** **(**int**)**strlen**(**s **+** 1**);** n **+=** len**;**

Node **\***r **=** get**(**x **+** 1**,** x **+** 1**);**

r**->**setc**(**build**(**1**,** len**),** 0**),** r**->**upd**(),** root**->**upd**();**

**}**

bool check**(**int l1**,** int r1**,** int l2**,** int r2**)** **{**

Ha v1 **=** get**(**l1**,** r1**)->**c**[**0**]->**ha**,** v2 **=** get**(**l2**,** r2**)->**c**[**0**]->**ha**;**

**return** v1 **==** v2**;**

**}**

void work**(**int x**,** int y**)** **{**

int l **=** 0**,** r **=** n **-** max**(**x**,** y**)** **+** 1**;**

**while** **(**l **<** r**)** **{**

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

**if** **(**check**(**x**,** x **+** mid**,** y**,** y **+** mid**))** l **=** mid**;**

**else** r **=** mid **-** 1**;**

**}**

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

**}**

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

init**();**

scanf**(**"%s%d"**,** s **+** 1**,** **&**m**);**

n **=** **(**int**)**strlen**(**s **+** 1**);**

root **=** build**(**0**,** n **+** 1**);**

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

scanf**(**"%s"**,** s**);**

**switch(**s**[**0**])** **{**

**case** 'Q'**:**

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

work**(**x**,** y**);**

**break;** **case** 'R'**:**

scanf**(**"%d%s"**,** **&**x**,** s **+** 1**);**

modify**(**x**,** s**[**1**]);**

**break;** **case** 'I'**:**

scanf**(**"%d%s"**,** **&**x**,** s **+** 1**);**

insert**(**x**);**

**}**

**}**

**return** 0**;**

**}**

# 萨菲克斯阿瑞

求出在两个字符串中各取出一个子串使得这两个子串相同的方案数。两个方案不同当且仅当这两个子串中有一个位置不同。

void Sort**(**int in**[],** int out**[],** int p**[],** int n**,** int m**)** **{**

static int P**[**MAXN**];**

**for** **(**int i **=** 1**;** i **<=** m**;** i**++)** P**[**i**]** **=** 0**;**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** P**[**in**[**i**]]++;**

**for** **(**int i **=** 2**;** i **<=** m**;** i**++)** P**[**i**]** **+=** P**[**i **-** 1**];**

**for** **(**int i **=** n**;** i**;** i**--)** out**[**P**[**in**[**p**[**i**]]]--]** **=** p**[**i**];**

**}**

char s**[**MAXN**];**

int sa**[**MAXN**],** rk**[**MAXN**],** h**[**MAXN**];**

int n**,** l1**;**

void getsa**()** **{**

static int t1**[**MAXN**],** t2**[**MAXN**],** **\***x **=** t1**,** **\***y **=** t2**;**

int m **=** 127**;**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** x**[**i**]** **=** s**[**i**],** y**[**i**]** **=** i**;**

Sort**(**x**,** sa**,** y**,** n**,** m**);**

**for** **(**int j **=** 1**,** i**,** k **=** 0**;** k **<** n**;** m **=** k**,** j **<<=** 1**)** **{**

**for** **(**i **=** n **-** j **+** 1**,** k **=** 0**;** i **<=** n**;** i**++)** y**[++**k**]** **=** i**;**

**for** **(**i **=** 1**;** i **<=** n**;** i**++)** **if** **(**sa**[**i**]** **>** j**)** y**[++**k**]** **=** sa**[**i**]** **-** j**;**

Sort**(**x**,** sa**,** y**,** n**,** m**);**

**for** **(**swap**(**x**,** y**),** i **=** 2**,** x**[**sa**[**1**]]** **=** k **=** 1**;** i **<=** n**;** i**++)** **{**

x**[**sa**[**i**]]** **=** **(**y**[**sa**[**i **-** 1**]]** **==** y**[**sa**[**i**]]** **&&** y**[**sa**[**i **-** 1**]** **+** j**]** **==** y**[**sa**[**i**]** **+** j**])** **?** k **:** **++**k**;**

**}**

**}**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** rk**[**sa**[**i**]]** **=** i**;**

**for** **(**int i **=** 1**,** k **=** 0**;** i **<=** n**;** h**[**rk**[**i**++]]** **=** k**)** **{**

k **-=** **!!**k**;**

**for** **(**int j **=** sa**[**rk**[**i**]** **-** 1**];** s**[**i **+** k**]** **==** s**[**j **+** k**];** k**++);**

**}**

**}**

int sta**[**MAXN**],** p**[**MAXN**],** L**[**MAXN**],** R**[**MAXN**],** s1**[**MAXN**],** s2**[**MAXN**];**

void work**()** **{**

int top **=** 0**;** sta**[**0**]** **=** **-**1**;**

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

**while** **(**sta**[**top**]** **>=** h**[**i**])** **{**

R**[**p**[**top**]]** **=** i **-** 1**;**

top**--;**

**}**

L**[**i**]** **=** p**[**top**]** **+** 1**;**

sta**[++**top**]** **=** h**[**i**],** p**[**top**]** **=** i**;**

**}**

**while** **(**top**)** R**[**p**[**top**--]]** **=** n**;**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** s1**[**i**]** **=** s1**[**i **-** 1**]** **+** **(**sa**[**i**]** **<** l1**);**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** s2**[**i**]** **=** s2**[**i **-** 1**]** **+** **(**sa**[**i**]** **>** l1**);**

LL ans **=** 0**;**

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

ans **+=** **(**LL**)**h**[**i**]** **\*** **((**s2**[**i **-** 1**]** **-** s2**[**L**[**i**]** **-** 2**])** **\*** **(**s1**[**R**[**i**]]** **-** s1**[**i **-** 1**])** **+** **(**s1**[**i **-** 1**]** **-** s1**[**L**[**i**]** **-** 2**])** **\*** **(**s2**[**R**[**i**]]** **-** s2**[**i **-** 1**]));**

**}**

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

**}**

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

scanf**(**"%s"**,** s **+** 1**);**

n **=** **(**int**)**strlen**(**s **+** 1**);**

s**[**l1 **=** **++**n**]** **=** '|'**;**

scanf**(**"%s"**,** s **+** n **+** 1**);**

n **+=** **(**int**)**strlen**(**s **+** n **+** 1**);**

getsa**();**

work**();**

**return** 0**;**

**}**

# 破腊肉

**namespace** Pollard\_Rho **{**

void mul**(**LL **&**x**,** LL y**,** LL m**)** **{**

x **%=** m**,** y **%=** m**;**

ULL t **=** **(**LL**)((**LD**)**x **\*** y **/** m**);**

x **=** **(**ULL**)** x **\*** y **-** t **\*** m**;**

x **=** **(**x **%** m **+** m**)** **%** m**;**

**}**

LL power**(**LL x**,** LL y**,** LL m**)** **{**

LL ret **=** 1**;**

**for** **(**x **%=** m**;** y**;** y **>>=** 1**)** **{**

**if** **(**y **&** 1**)** mul**(**ret**,** x**,** m**);**

mul**(**x**,** x**,** m**);**

**}**

**return** ret**;**

**}**

LL Rand**()** **{**

static LL a **=** 2333333**;**

a **^=** a **>>** 12**;** a **^=** a **<<** 25**;** a **^=** a **>>** 27**;**

**return** a **\*** 0x2545F4914F6CDD1D**;**

**}**

LL gcd**(**LL a**,** LL b**)** **{**

**while** **(**b**)** **{**

LL c **=** a**;**

a **=** b**;** b **=** c **%** b**;**

**}**

**return** a**;**

**}**

const int P**[]** **=** **{**2**,** 3**,** 5**,** 7**,** 11**,** 13**,** 17**,** 19**,** 23**,** 29**,** 31**,** 37**,** 41**,** 43**,** **-**1**};**

vector**<**LL**>** D**;**

bool witness**(**int p**,** LL x**,** int t**,** LL n**)** **{**

LL y **=** power**(**p**,** x**,** n**);**

**if** **(**y **==** 1**)** **return** 0**;**

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

**if** **(**y **==** n **-** 1**)** **return** 0**;**

**if** **(**y **==** 1**)** **return** 0**;**

mul**(**y**,** y**,** n**);**

**}**

**return** 1**;**

**}**

bool MR**(**LL n**)** **{**

**if** **(**n **<** 4**)** **return** 1**;**

**for** **(**int i **=** 0**;** **~**P**[**i**];** i**++)** **{**

**if** **(**n **==** P**[**i**])** **return** 1**;**

**if** **(**n **%** P**[**i**]** **==** 0**)** **return** 0**;**

**}**

LL x **=** n **-** 1**;** int t **=** 0**;**

**while** **(**x **&** 1 **^** 1**)** **{**

**++**t**;**

x **>>=** 1**;**

**}**

**for** **(**int i **=** 0**;** **~**P**[**i**];** i**++)** **{**

**if** **(**witness**(**P**[**i**],** x**,** t**,** n**))** **return** 0**;**

**}**

**return** 1**;**

**}**

void PR**(**LL n**)** **{**

**if** **(**MR**(**n**))** **{**

D**.**push\_back**(**n**);**

**return;**

**}**

LL a**,** b**,** c**,** d**;**

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

c **=** Rand**()** **%** n**;**

a **=** b **=** Rand**()** **%** n**;**

mul**(**b**,** b**,** n**);** **(**b **+=** c**)** **%=** n**;**

**while** **(**a **!=** b**)** **{**

d **=** a **>** b **?** a **-** b **:** b **-** a**;**

d **=** gcd**(**d**,** n**);**

**if** **(**d **>** 1 **&&** d **<** n**)** **{**

PR**(**d**);** PR**(**n **/** d**);**

**return;**

**}**

mul**(**a**,** a**,** n**);** **(**a **+=** c**)** **%=** n**;**

mul**(**b**,** b**,** n**);** **(**b **+=** c**)** **%=** n**;**

mul**(**b**,** b**,** n**);** **(**b **+=** c**)** **%=** n**;**

**}**

**}**

**}**

void solve**(**LL n**)** **{**

D**.**clear**();**

**if** **(**n **<** 2**)** **return;**

PR**(**n**);**

sort**(**D**.**begin**(),** D**.**end**());**

**}**

**}**

# 淀粉质

设(*i*,*j*)之间的简单路径边数为*dis*(*i*,*j*)，求所有满足*dis*(*i*,*j*)≤*K*且*i*<*j*的数对(*i*,*j*)的数量。

LL ans **=** 0**;**

struct edge **{**

int next**,** to**;**

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

int ecnt **=** 0**,** head**[**MAXN**];**

void add**(**int u**,** int v**)** **{**

e**[++**ecnt**].**to **=** v**;** e**[**ecnt**].**next **=** head**[**u**];** head**[**u**]** **=** ecnt**;**

e**[++**ecnt**].**to **=** u**;** e**[**ecnt**].**next **=** head**[**v**];** head**[**v**]** **=** ecnt**;**

**}**

int n**,** k**;**

int siz**[**MAXN**];**

bool vis**[**MAXN**];**

int ds**(**int x**)** **{**

siz**[**x**]** **=** 1**;** vis**[**x**]** **=** 1**;**

**for** **(**int now **=** head**[**x**];** now**;** now **=** e**[**now**].**next**)** **{**

**if** **(!**vis**[**e**[**now**].**to**])** siz**[**x**]** **+=** ds**(**e**[**now**].**to**);**

**}**

vis**[**x**]** **=** 0**;**

**return** siz**[**x**];**

**}**

int find**(**int x**)** **{**

ds**(**x**);**

**for** **(**int rt **=** x**;** **;** **)** **{**

int maxs **=** 0**,** to **=** **-**1**;**

**for** **(**int now **=** head**[**rt**];** now**;** now **=** e**[**now**].**next**)** **{**

**if** **(!**vis**[**e**[**now**].**to**]** **&&** siz**[**e**[**now**].**to**]** **<** siz**[**rt**]** **&&** siz**[**e**[**now**].**to**]** **>** maxs**)**

maxs **=** siz**[**e**[**now**].**to**],** to **=** e**[**now**].**to**;**

**}**

**if** **(**maxs **>** siz**[**x**]** **/** 2**)** rt **=** to**;**

**else** **return** rt**;**

**}**

**}**

int l**[**MAXN**],** lcnt **=** 0**,** dep**[**MAXN**];**

void dd**(**int x**)** **{**

l**[++**lcnt**]** **=** dep**[**x**];** vis**[**x**]** **=** 1**;**

**for** **(**int now **=** head**[**x**];** now**;** now **=** e**[**now**].**next**)** **{**

**if** **(!**vis**[**e**[**now**].**to**])** **{**

dep**[**e**[**now**].**to**]** **=** dep**[**x**]** **+** 1**;**

dd**(**e**[**now**].**to**);**

**}**

**}**

vis**[**x**]** **=** 0**;**

**}**

LL work**(**int rt**,** int base**)** **{**

dep**[**rt**]** **=** base**;** lcnt **=** 0**;** dd**(**rt**);**

sort**(**l **+** 1**,** l **+** lcnt **+** 1**);**

LL ret **=** 0**;** int i **=** 1**,** j **=** lcnt**;**

**while** **(**i **<** j**)** **{**

**if** **(**l**[**i**]** **+** l**[**j**]** **<=** k**)** ret **+=** j **-** i**,** i**++;**

**else** j**--;**

**}**

**return** ret**;**

**}**

void solve**(**int x**)** **{**

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

ans **+=** work**(**rt**,** 0**);** vis**[**rt**]** **=** 1**;**

**for** **(**int now **=** head**[**rt**];** now**;** now **=** e**[**now**].**next**)** **{**

**if** **(!**vis**[**e**[**now**].**to**])** **{**

ans **-=** work**(**e**[**now**].**to**,** 1**);**

solve**(**e**[**now**].**to**);**

**}**

**}**

**}**

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

read**(**n**);** read**(**k**);**

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

read**(**u**);** read**(**v**);** add**(**u**,** v**);**

**}**

solve**(**1**);**

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

**return** 0**;**

**}**

# K短路

#define N 200005

#define INF 0x3f3f3f3f3f3f3f3f

int n**,** m**,** S**,** T**,** K**;**

LL dis**[**N**];**

vector **<** pair**<**int**,** LL**>** **>** E**[**N**];**

void dij**()** **{**

static priority\_queue **<** pair**<**LL**,** int**>** **>** q**;**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** dis**[**i**]** **=** INF**;**

dis**[**T**]** **=** 0**;**

q**.**push**({**0ll**,** T**});**

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

pair**<**LL**,** int**>** y **=** q**.**top**();** q**.**pop**();**

LL d **=** **-**y**.**x**;** int x **=** y**.**y**;**

**if** **(**d **!=** dis**[**x**])** **continue;**

**for** **(**auto o **:** E**[**x**])** **{**

int v **=** o**.**x**;** LL w **=** o**.**y**;**

**if** **(**dis**[**v**]** **>** dis**[**x**]** **+** w**)** **{**

dis**[**v**]** **=** dis**[**x**]** **+** w**;**

q**.**push**({-**dis**[**v**],** v**});**

**}**

**}**

**}**

**}**

struct Edge **{**

int u**,** v**;** LL w**;**

bool **operator** **<** **(**const Edge **&**a**)** const **{**

**return** w **<** a**.**w**;**

**}**

**};**

struct Node **{**

int l**,** r**,** v**;**

**}** t**[**N **\*** 50**];**

int rt**[**N**],** tcnt **=** 0**;**

int upd**(**int pre**,** int l**,** int r**,** int p**)** **{**

int x **=** **++**tcnt**;**

t**[**x**]** **=** t**[**pre**];**

t**[**x**].**v**++;**

**if** **(**l **<** r**)** **{**

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

**if** **(**p **<=** mid**)** t**[**x**].**l **=** upd**(**t**[**x**].**l**,** l**,** mid**,** p**);**

**else** t**[**x**].**r **=** upd**(**t**[**x**].**r**,** mid **+** 1**,** r**,** p**);**

**}**

**return** x**;**

**}**

int qry**(**int x**,** int l**,** int r**,** int k**)** **{**

**if** **(**l **==** r**)** **return** l**;**

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

**if** **(**t**[**t**[**x**].**l**].**v **>=** k**)** **return** qry**(**t**[**x**].**l**,** l**,** mid**,** k**);**

**else** **return** qry**(**t**[**x**].**r**,** mid **+** 1**,** r**,** k **-** t**[**t**[**x**].**l**].**v**);**

**}**

int fa**[**N**];**

vector **<**Edge**>** e**;**

vector **<**pii**>** c**[**N**];**

int q**[**N**],** he **=** 0**,** ta **=** 1**;**

void bfs**()** **{**

q**[**he**]** **=** T**;** fa**[**T**]** **=** T**;**

**while** **(**he **<** ta**)** **{**

int x **=** q**[**he**++];**

**for** **(**auto o **:** E**[**x**])** **if** **(**dis**[**o**.**x**]** **==** dis**[**x**]** **+** o**.**y**){**

**if** **(**fa**[**o**.**x**])** **{**

e**.**push\_back**((**Edge**){**o**.**x**,** x**,** 0ll**});**

**}**

**else** **{**

fa**[**o**.**x**]** **=** x**;**

q**[**ta**++]** **=** o**.**x**;**

**}**

**}**

**}**

**}**

priority\_queue **<**pair**<**LL**,** pii**>>** Q**;**

vector **<**int**>** res**[**N**];**

int M**;**

int Kth**(**int x**,** int k**)** **{**

**while** **(**res**[**x**].**size**()** **<** k**)** res**[**x**].**push\_back**(**qry**(**rt**[**x**],** 0**,** M**,** **(**int**)**res**[**x**].**size**()** **+** 1**));**

**return** res**[**x**][**k **-** 1**];**

**}**

void work**()** **{**

bfs**();**

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

**for** **(**auto o **:** E**[**i**])** **if** **(**dis**[**o**.**x**]** **<** dis**[**i**]** **+** o**.**y**)** **{**

e**.**push\_back**((**Edge**)** **{**o**.**x**,** i**,** dis**[**i**]** **+** o**.**y **-** dis**[**o**.**x**]});**

**}**

**}**

sort**(**e**.**begin**(),** e**.**end**());**

M **=** **(**int**)**e**.**size**();**

**for** **(**int i **=** 0**;** i **<** M**;** i**++)** c**[**e**[**i**].**u**].**push\_back**({**e**[**i**].**v**,** i**});**

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

int x **=** q**[**i**];**

rt**[**x**]** **=** rt**[**fa**[**x**]];**

**for** **(**auto o **:** c**[**x**])** **{**

rt**[**x**]** **=** upd**(**rt**[**x**],** 0**,** M**,** o**.**y**);**

**}**

**}**

int k **=** S **==** T **?** 0 **:** 1**;**

LL ans **=** 0**;**

**if** **(**t**[**rt**[**S**]].**v**)** Q**.**push**({(**LL**)-**e**[**Kth**(**S**,** 1**)].**w**,** **{**S**,** 1**}});**

**while** **(**k **<** K **&&** **!**Q**.**empty**())** **{**

pair **<**LL**,** pii**>** o **=** Q**.**top**();** Q**.**pop**();**

ans **=** **-**o**.**x**;** k**++;**

int x **=** o**.**y**.**x**,** kk **=** o**.**y**.**y**;**

int id **=** Kth**(**x**,** kk**),** v **=** e**[**id**].**v**;**

**if** **(**t**[**rt**[**v**]].**v**)** Q**.**push**({-(**ans **+** e**[**Kth**(**v**,** 1**)].**w**),** **{**v**,** 1**}});**

**if** **(++**kk **>** t**[**rt**[**x**]].**v**)** **continue;**

int id2 **=** Kth**(**x**,** kk**);**

Q**.**push**({-(**ans **-** e**[**id**].**w **+** e**[**id2**].**w**),** **{**x**,** kk**}});**

**}**

**if** **(**k **<** K**)** puts**(**"-1"**);**

printf**(**"%lld\n"**,** ans **+** dis**[**S**]);**

**}**

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

read**(**n**);** read**(**m**);** read**(**S**);** read**(**T**);** read**(**K**);**

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

int u**,** v**,** w**;**

read**(**u**);** read**(**v**);** read**(**w**);**

E**[**v**].**push\_back**({**u**,** w**});**

**}**

dij**();**

work**();**

**return** 0**;**

**}**

# SLFSPFA

struct Edge **{**int v**,** w**;};**

vector **<**Edge**>** E**[**N**];**

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

int dis**[**N**];**

void spfa**()** **{**

const int M **=** 262143**;**

static int q**[**M **+** 1**],** h**,** t**;**

static bool vis**[**N**];**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** dis**[**i**]** **=** INF**,** vis**[**i**]** **=** 0**;**

dis**[**S**]** **=** 0**,** h **=** 0**,** t **=** 0**,** q**[**0**]** **=** S**;**

**while** **(**h **<=** t**)** **{**

int x **=** q**[**h**++** **&** M**];**

vis**[**x**]** **=** 0**;**

**if** **(**h **<** t **&&** dis**[**q**[**t **&** M**]]** **<** dis**[**q**[**h **&** M**]])** swap**(**q**[**h **&** M**],** q**[**t **&** M**]);**

**for** **(**auto o **:** E**[**x**])** **{**

**if** **(**dis**[**o**.**v**]** **>** **(**LL**)**dis**[**x**]** **+** o**.**w**)** **{**

dis**[**o**.**v**]** **=** dis**[**x**]** **+** o**.**w**;**

**if** **(!**vis**[**o**.**v**])** **{**

q**[++**t **&** M**]** **=** o**.**v**;**

vis**[**o**.**v**]** **=** 1**;**

**if** **(**h **<** t **&&** dis**[**q**[**t **&** M**]]** **<** dis**[**q**[**h **&** M**]])** swap**(**q**[**h **&** M**],** q**[**t **&** M**]);**

**}**

**}**

**}**

**}**

**}**

# 支配树

#define N 500005

struct DT **{**

int n**,** s**,** cnt**;** int dfn**[**N**],** id**[**N**],** pa**[**N**],** semi**[**N**],** idom**[**N**],** p**[**N**],** mn**[**N**],** f**[**N**];**

vector **<**int**>** E**[**N**],** dom**[**N**],** be**[**N**];**

void ins**(**int x**,** int y**)** **{**E**[**x**].**push\_back**(**y**);}**

void dfs**(**int x**)** **{**

dfn**[**x**]** **=** **++**cnt**;** id**[**cnt**]** **=** x**;**

**for** **(**auto i **:** E**[**x**])** **{**

**if** **(!**dfn**[**i**])** dfs**(**i**),** pa**[**dfn**[**i**]]** **=** dfn**[**x**];**

be**[**dfn**[**i**]].**push\_back**(**dfn**[**x**]);**

**}**

**}**

int get**(**int x**)** **{**

**if** **(**p**[**x**]** **!=** p**[**p**[**x**]])** **{**

**if** **(**semi**[**mn**[**x**]]** **>** semi**[**get**(**p**[**x**])])** mn**[**x**]** **=** get**(**p**[**x**]);**

p**[**x**]** **=** p**[**p**[**x**]];**

**}**

**return** mn**[**x**];**

**}**

void LT**()** **{**

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

**for** **(**auto j **:** be**[**i**])** semi**[**i**]** **=** min**(**semi**[**i**],** semi**[**get**(**j**)]);**

dom**[**semi**[**i**]].**push\_back**(**i**);** int x **=** p**[**i**]** **=** pa**[**i**];**

**for** **(**auto j **:** dom**[**x**])** idom**[**j**]** **=** **(**semi**[**get**(**j**)]** **<** x **?** get**(**j**)** **:** x**);**

dom**[**x**].**clear**();**

**}**

**for** **(**int i **=** 2**;** i **<=** cnt**;** i**++)** **{**

**if** **(**idom**[**i**]** **!=** semi**[**i**])** idom**[**i**]** **=** idom**[**idom**[**i**]];**

dom**[**id**[**idom**[**i**]]].**push\_back**(**id**[**i**]);**

**}**

**}**

void build**()** **{**

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

dfn**[**i**]** **=** 0**,** dom**[**i**].**clear**(),** be**[**i**].**clear**(),** p**[**i**]** **=** mn**[**i**]** **=** semi**[**i**]** **=** i**;**

cnt **=** 0**;** dfs**(**s**);** LT**();**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** f**[**id**[**i**]]** **=** id**[**idom**[**i**]];**

**}**

**}** sblbn**;**

int m**;**

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

read**(**sblbn**.**n**);** read**(**m**);** read**(**sblbn**.**s**);**

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

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

read**(**x**);** read**(**y**);**

sblbn**.**ins**(**x**,** y**);**

**}**

sblbn**.**build**();**

**for** **(**int i **=** 1**;** i **<=** sblbn**.**n**;** i**++)** printf**(**"%d "**,** sblbn**.**f**[**i**]);**

**return** 0**;**

**}**

# KM

struct KM **{** // source from ymkzpx

int n**,** nl**,** nr**;**

LL a**[**N**][**N**];**

LL hl**[**N**],** hr**[**N**],** s**[**N**];**

int fl**[**N**],** fr**[**N**],** vl**[**N**],** vr**[**N**],** pre**[**N**],** q**[**N**],** ql**,** qr**;**

int check**(**int i**)** **{**

**if** **(**vl**[**i**]** **=** 1**,** fl**[**i**]** **!=** **-**1**)** **return** vr**[**q**[**qr**++]** **=** fl**[**i**]]** **=** 1**;**

**while** **(**i **!=** **-**1**)** swap**(**i**,** fr**[**fl**[**i**]** **=** pre**[**i**]]);**

**return** 0**;**

**}**

void bfs**(**int st**)** **{**

fill**(**s**,** s **+** n**,** INF**);**

fill**(**vl**,** vl **+** n**,** 0**);**

fill**(**vr**,** vr **+** n**,** 0**);**

q**[**ql **=** 0**]** **=** st**;**

vr**[**st**]** **=** qr **=** 1**;**

**for** **(**LL d**;;)** **{**

**for** **(;** ql **<** qr**;** **++**ql**)**

**for** **(**int i **=** 0**,** j **=** q**[**ql**];** i **<** n**;** **++**i**)**

**if** **(**d **=** hl**[**i**]** **+** hr**[**j**]** **-** a**[**i**][**j**],** **!**vl**[**i**]** **&&** s**[**i**]** **>=** d**)** **{**

**if** **(**pre**[**i**]** **=** j**,** d**)** s**[**i**]** **=** d**;**

**else** **if** **(!**check**(**i**))** **return;**

**}**

d **=** INF**;**

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

**if** **(!**vl**[**i**]** **&&** d **>** s**[**i**])**d **=** s**[**i**];**

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

**if** **(**vl**[**i**])**hl**[**i**]** **+=** d**;**

**else** s**[**i**]** **-=** d**;**

**if** **(**vr**[**i**])**hr**[**i**]** **-=** d**;**

**}**

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

**if** **(!**vl**[**i**]** **&&** **!**s**[**i**]** **&&** **!**check**(**i**))** **return;**

**}**

**}**

void solve**()** **{**

n **=** max**(**nl**,** nr**);**

fill**(**pre**,** pre **+** n**,** **-**1**);**

fill**(**fl**,** fl **+** n**,** **-**1**);**

fill**(**fr**,** fr **+** n**,** **-**1**);**

fill**(**hr**,** hr **+** n**,** 0**);**

**for** **(**int i **=** 0**;** i **<** n**;** **++**i**)** hl**[**i**]** **=** **\***max\_element**(**a**[**i**],** a**[**i**]** **+** n**);**

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

**}**

LL calc**()** **{**

LL ans **=** 0**;**

**for** **(**int i **=** 0**;** i **<** nl**;** **++**i**)** **if** **(~**fl**[**i**])** ans **+=** a**[**i**][**fl**[**i**]];**

**return** ans**;**

**}**

void output**()** **{**

**for** **(**int i **=** 0**;** i **<** nl**;** **++**i**)** printf**(**"%d%c"**,** **(~**fl**[**i**]** **&&** a**[**i**][**fl**[**i**]]** **?** fl**[**i**]** **+** 1 **:** 0**),** i **==** nl **-** 1 **?** '\n' **:** ' '**);**

**}**

**}** km**;**

int m**;**

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

read**(**km**.**nl**);** read**(**km**.**nr**);** read**(**m**);**

**for** **(**int i **=** 1**,** u**,** v**,** x**;** i **<=** m**;** i**++)** **{**

read**(**u**);** read**(**v**);** read**(**x**);**

km**.**a**[**u **-** 1**][**v **-** 1**]** **=** x**;**

**}**

km**.**solve**();**

printf**(**"%lld\n"**,** km**.**calc**());**

km**.**output**();**

**return** 0**;**

**}**

# 带花树

struct Blossom **{** // source from lbn187

int n**,** m**,** h**,** t**,** W**,** tot**,** fir**[**N**],** la**[**M**],** ne**[**M**],** F**[**N**],** mat**[**N**],** pre**[**N**],** tp**[**N**],** q**[**N**],** vs**[**N**];**

void ins**(**int x**,** int y**)** **{**

la**[++**tot**]** **=** y**;**

ne**[**tot**]** **=** fir**[**x**];**

fir**[**x**]** **=** tot**;**

**}**

void in**(**int x**,** int y**)** **{**

n **=** x**,** m **=** y**;**

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

fill**(**fir**,** fir **+** n **+** 1**,** 0**);**

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

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

**}**

**}**

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

**return** F**[**x**]** **?** F**[**x**]** **=** getf**(**F**[**x**])** **:** x**;**

**}**

int lca**(**int u**,** int v**)** **{**

**for** **(++**W**;** **;** u **=** pre**[**mat**[**u**]],** swap**(**u**,** v**))**

**if** **(**vs**[**u **=** getf**(**u**)]** **==** W**)** **return** u**;**

**else** vs**[**u**]** **=** u **?** W **:** 0**;**

**}**

void aug**(**int u**,** int v**)** **{**

**for** **(**int w**;** u**;** v **=** pre**[**u **=** w**])** w **=** mat**[**v**],** mat**[**mat**[**u**]** **=** v**]** **=** u**;**

**}**

void blo**(**int u**,** int v**,** int f**)** **{**

**for** **(**int w**;** getf**(**u**)** **^** f**;** u **=** pre**[**v **=** w**])** pre**[**u**]** **=** v**,** F**[**u**]** **?** 0 **:** F**[**u**]** **=** f**,** F**[**w **=** mat**[**u**]]** **?** 0 **:** F**[**w**]** **=** f**,** tp**[**w**]** **^** 1 **?** 0 **:** tp**[**q**[++**t**]** **=** w**]** **=** **-**1**;**

**}**

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

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** tp**[**i**]** **=** F**[**i**]** **=** 0**;**

**for** **(--**tp**[**q**[**h **=** t **=** 1**]** **=** u**];** h **<=** t**;** u **=** q**[++**h**])**

**for** **(**int i **=** fir**[**u**],** x**,** E**;** i**;** i **=** ne**[**i**])** **if** **(!**tp**[**E **=** la**[**i**]])** **{**

**if** **(!**mat**[**E**])** **return** aug**(**E**,** u**),** 1**;**

pre**[**E**]** **=** u**,** **++**tp**[**E**],** **--**tp**[**q**[++**t**]** **=** mat**[**E**]];**

**}**

**else** **if** **(**tp**[**E**]** **^** 1 **&&** getf**(**u**)^**getf**(**E**))** blo**(**u**,** E**,** x **=** lca**(**u**,** E**)),** blo**(**E**,** u**,** x**);**

**return** 0**;**

**}**

int solve**()** **{**

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

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** mat**[**i**]** **?** 0 **:** ans **+=** bfs**(**i**);**

**return** ans**;**

**}**

void out**()** **{**

**for** **(**int i **=** 1**;** i **<=** n**;** i**++)** printf**(**"%d%c"**,** mat**[**i**],** i **==** n **?** '\n' **:** ' '**);**

**}**

**}** G**;**

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

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

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

G**.**in**(**n**,** m**);**

printf**(**"%d\n"**,** G**.**solve**());**

G**.**out**();**

**}**

# Simplex

#define N 105

const LD eps **=** 1e-8**,** INF **=** 1e9**;**

int K**;**

**namespace** Simplex **{** // source from bright\_sun

int n**,** m**,** id**[**N**],** tp**[**N**];** LD a**[**N**][**N**];**

void pivot**(**int r**,** int c**)** **{**

swap**(**id**[**r **+** n**],** id**[**c**]);**

LD t **=** **-**a**[**r**][**c**];** a**[**r**][**c**]** **=** **-**1**;**

**for** **(**int i **=** 0**;** i **<=** n**;** i**++)** a**[**r**][**i**]** **/=** t**;**

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

**if** **(**a**[**i**][**c**]** **&&** r **!=** i**)** **{**

t **=** a**[**i**][**c**];** a**[**i**][**c**]** **=** 0**;**

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

a**[**i**][**j**]** **+=** t **\*** a**[**r**][**j**];**

**}**

**}**

bool solve**()** **{**

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

**for** **(** **;** **;** **)** **{**

int i **=** 0**,** j **=** 0**;** LD w **=** **-**eps**;**

**for** **(**int k **=** 1**;** k **<=** m**;** k**++)**

**if** **(**a**[**k**][**0**]** **<** w **||** **(**a**[**k**][**0**]** **<** **-**eps **&&** rand**()** **%** 8 **==** 1**))**

w **=** a**[**i **=** k**][**0**];**

**if** **(!**i**)** **break;**

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

**if** **(**a**[**i**][**k**]** **>** eps**)** **{**

j **=** k**;**

**break;**

**}**

**if** **(!**j**)** **{**

printf**(**"Infeasible"**);**

**return** 0**;**

**}**

pivot**(**i**,** j**);**

**}**

**for** **(** **;** **;** **)** **{**

int i **=** 0**,** j **=** 0**;** LD w **=** eps**,** t**;**

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

**if** **(**a**[**0**][**k**]** **>** w**)** w **=** a**[**0**][**j **=** k**];**

**if** **(!**j**)** **break;**

w **=** INF**;**

**for** **(**int k **=** 1**;** k **<=** m**;** k**++)**

**if** **(**a**[**k**][**j**]** **<** **-**eps **&&** **(**t **=** **-**a**[**k**][**0**]** **/** a**[**k**][**j**])** **<** w**)**

w **=** t**,** i **=** k**;**

**if** **(!**i**)** **{**

printf**(**"Unbounded"**);**

**return** 0**;**

**}**

pivot**(**i**,** j**);**

**}**

**return** 1**;**

**}**

LD ans**()** **{return** a**[**0**][**0**];}**

void output**()** **{**

**for** **(**int i **=** n **+** 1**;** i **<=** n **+** m**;** i**++)** tp**[**id**[**i**]]** **=** i **-** n**;**

**for** **(**int i **=** 1**;** i **<=**n**;** i**++)** printf**(**"%.9lf "**,** tp**[**i**]** **?** **(**double**)**a**[**tp**[**i**]][**0**]** **:** 0**);**

**}**

**}**

**using** **namespace** Simplex**;**

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

read**(**n**);** read**(**m**);** read**(**K**);**

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

double x**;**

scanf**(**"%lf"**,** **&**x**);** a**[**0**][**i**]** **=** x**;**

**}**

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

double x**;**

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

scanf**(**"%lf"**,** **&**x**),** a**[**i**][**j**]** **=** **-**x**;**

scanf**(**"%lf"**,** **&**x**);** a**[**i**][**0**]** **=** x**;**

**}**

**if** **(**solve**())** **{**

printf**(**"%.9lf\n"**,** **(**double**)**ans**());**

**if** **(**K**)** output**();**

**}**

**return** 0**;**

**}**

# 最小表示

int solve**(**char **\***text**,** int length**)** **{**

int i **=** 0**,** j **=** 1**,** delta **=** 0**;**

**while** **(**i **<** length **&&** j **<** length **&&** delta **<** length**)** **{**

char tokeni **=** text**[(**i **+** delta**)** **%** length**];**

char tokenj **=** text**[(**j **+** delta**)** **%** length**];**

**if** **(**tokeni **==** tokenj**)** **{**

delta**++;**

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

**if** **(**tokeni **>** tokenj**)** **{**

i **+=** delta **+** 1**;**

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

j **+=** delta **+** 1**;**

**}**

**if** **(**i **==** j**)** **{**

j**++;**

**}**

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

**}**

**}**

**return** std**::**min**(**i**,** j**);**

**}**

# 树哈希

const unsigned long long MAGIC **=** 4423**;**

unsigned long long magic**[**N**];**

std**::**pair**<**unsigned long long**,** int**>** hash**[**N**];**

void solve**(**int root**)** **{**

magic**[**0**]** **=** 1**;**

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

magic**[**i**]** **=** magic**[**i **-** 1**]** **\*** MAGIC**;**

**}**

std**::**vector**<**int**>** queue**;**

queue**.**push\_back**(**root**);**

**for** **(**int head **=** 0**;** head **<** **(**int**)**queue**.**size**();** **++**head**)** **{**

int x **=** queue**[**head**];**

**for** **(**int i **=** 0**;** i **<** **(**int**)**son**[**x**].**size**();** **++**i**)** **{**

int y **=** son**[**x**][**i**];**

queue**.**push\_back**(**y**);**

**}**

**}**

**for** **(**int index **=** n **-** 1**;** index **>=** 0**;** **--**index**)** **{**

int x **=** queue**[**index**];**

hash**[**x**]** **=** std**::**make\_pair**(**0**,** 0**);**

std**::**vector**<**std**::**pair**<**unsigned long long**,** int**>** **>** value**;**

**for** **(**int i **=** 0**;** i **<** **(**int**)**son**[**x**].**size**();** **++**i**)** **{**

int y **=** son**[**x**][**i**];**

value**.**push\_back**(**hash**[**y**]);**

**}**

std**::**sort**(**value**.**begin**(),** value**.**end**());**

hash**[**x**].**first **=** hash**[**x**].**first **\*** magic**[**1**]** **+** 37**;**

hash**[**x**].**second**++;**

**for** **(**int i **=** 0**;** i **<** **(**int**)**value**.**size**();** **++**i**)** **{**

hash**[**x**].**first **=** hash**[**x**].**first **\*** magic**[**value**[**i**].**second**]** **+** value**[**i**].**first**;**

hash**[**x**].**second **+=** value**[**i**].**second**;**

**}**

hash**[**x**].**first **=** hash**[**x**].**first **\*** magic**[**1**]** **+** 41**;**

hash**[**x**].**second**++;**

**}**

**}**

# ZKW费用流

int S**,** T**,** totFlow**,** totCost**;**

int dis**[**N**],** slack**[**N**],** visit**[**N**];**

int modlable **()** **{**

int delta **=** INF**;**

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

**if** **(!**visit**[**i**]** **&&** slack**[**i**]** **<** delta**)** delta **=** slack**[**i**];**

slack**[**i**]** **=** INF**;**

**}**

**if** **(**delta **==** INF**)** **return** 1**;**

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

**if** **(**visit**[**i**])** dis**[**i**]** **+=** delta**;**

**return** 0**;**

**}**

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

**if** **(**x **==** T**)** **{**

totFlow **+=** flow**;**

totCost **+=** flow **\*** **(**dis**[**S**]** **-** dis**[**T**]);**

**return** flow**;**

**}**

visit**[**x**]** **=** 1**;**

int left **=** flow**;**

**for** **(**int i **=** e**.**last**[**x**];** **~**i**;** i **=** e**.**succ**[**i**])**

**if** **(**e**.**cap**[**i**]** **>** 0 **&&** **!**visit**[**e**.**other**[**i**]])** **{**

int y **=** e**.**other**[**i**];**

**if** **(**dis**[**y**]** **+** e**.**cost**[**i**]** **==** dis**[**x**])** **{**

int delta **=** dfs **(**y**,** min **(**left**,** e**.**cap**[**i**]));**

e**.**cap**[**i**]** **-=** delta**;**

e**.**cap**[**i **^** 1**]** **+=** delta**;**

left **-=** delta**;**

**if** **(!**left**)** **{** visit**[**x**]** **=** 0**;** **return** flow**;** **}**

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

slack**[**y**]** **=** min **(**slack**[**y**],** dis**[**y**]** **+** e**.**cost**[**i**]** **-** dis**[**x**]);**

**}**

**}**

**return** flow **-** left**;**

**}**

pair **<**int**,** int**>** minCost **()** **{**

totFlow **=** 0**;** totCost **=** 0**;**

fill **(**dis **+** 1**,** dis **+** T **+** 1**,** 0**);**

**do** **{**

**do** **{**

fill **(**visit **+** 1**,** visit **+** T **+** 1**,** 0**);**

**}** **while** **(**dfs **(**S**,** INF**));**

**}** **while** **(!**modlable **());**

**return** make\_pair **(**totFlow**,** totCost**);**

**}**

# 无向图最小割Stoer-Wagner

int node**[**N**],** dist**[**N**];**

bool visit**[**N**];**

int solve**(**int n**)** **{**

int answer **=** INT\_MAX**;**

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

node**[**i**]** **=** i**;**

**}**

**while** **(**n **>** 1**)** **{**

int max **=** 1**;**

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

dist**[**node**[**i**]]** **=** graph**[**node**[**0**]][**node**[**i**]];**

**if** **(**dist**[**node**[**i**]]** **>** dist**[**node**[**max**]])** **{**

max **=** i**;**

**}**

**}**

int prev **=** 0**;**

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

visit**[**node**[**0**]]** **=** **true;**

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

**if** **(**i **==** n **-** 1**)** **{**

answer **=** std**::**min**(**answer**,** dist**[**node**[**max**]]);**

**for** **(**int k **=** 0**;** k **<** n**;** **++**k**)** **{**

graph**[**node**[**k**]][**node**[**prev**]]** **=**

**(**graph**[**node**[**prev**]][**node**[**k**]]** **+=** graph**[**node**[**k**]][**node**[**max**]]);**

**}**

node**[**max**]** **=** node**[--**n**];**

**}**

visit**[**node**[**max**]]** **=** **true;**

prev **=** max**;**

max **=** **-**1**;**

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

**if** **(!**visit**[**node**[**j**]])** **{**

dist**[**node**[**j**]]** **+=** graph**[**node**[**prev**]][**node**[**j**]];**

**if** **(**max **==** **-**1 **||** dist**[**node**[**max**]]** **<** dist**[**node**[**j**]])** **{**

max **=** j**;**

**}**

**}**

**}**

**}**

**}**

**return** answer**;**

**}**

# KDTree

long long norm**(**const long long **&**x**)** **{**

**return** std**::**abs**(**x**);**

**return** x **\*** x**;**

**}**

struct Point **{**

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

const int**&** **operator** **[]** **(**int index**)** const **{**

**if** **(**index **==** 0**)** **{**

**return** x**;**

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

**return** y**;**

**}**

**}**

friend long long dist**(**const Point **&**a**,** const Point **&**b**)** **{**

long long result **=** 0**;**

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

result **+=** norm**(**a**[**i**]** **-** b**[**i**]);**

**}**

**return** result**;**

**}**

**}** point**[**N**];**

struct Rectangle **{**

int min**[**2**],** max**[**2**];**

Rectangle**()** **{**

min**[**0**]** **=** min**[**1**]** **=** INT\_MAX**;** // sometimes int is not enough

max**[**0**]** **=** max**[**1**]** **=** INT\_MIN**;**

**}**

void add**(**const Point **&**p**)** **{**

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

min**[**i**]** **=** std**::**min**(**min**[**i**],** p**[**i**]);**

max**[**i**]** **=** std**::**max**(**max**[**i**],** p**[**i**]);**

**}**

**}**

long long dist**(**const Point **&**p**)** **{**

long long result **=** 0**;**

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

result **+=** norm**(**std**::**min**(**std**::**max**(**p**[**i**],** min**[**i**]),** max**[**i**])** **-** p**[**i**]);**

result **+=** std**::**max**(**norm**(**max**[**i**]** **-** p**[**i**]),** norm**(**min**[**i**]** **-** p**[**i**]));**

**}**

**return** result**;**

**}**

**};**

struct Node **{**

Point seperator**;**

Rectangle rectangle**;**

int child**[**2**];**

void reset**(**const Point **&**p**)** **{**

seperator **=** p**;**

rectangle **=** Rectangle**();**

rectangle**.**add**(**p**);**

child**[**0**]** **=** child**[**1**]** **=** 0**;**

**}**

**}** tree**[**N **<<** 1**];**

int size**,** pivot**;**

bool compare**(**const Point **&**a**,** const Point **&**b**)** **{**

**if** **(**a**[**pivot**]** **!=** b**[**pivot**])** **{**

**return** a**[**pivot**]** **<** b**[**pivot**];**

**}**

**return** a**.**id **<** b**.**id**;**

**}**

// 左閉右開: build(1, n + 1)

int build**(**int l**,** int r**,** int type **=** 1**)** **{**

pivot **=** type**;**

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

**return** 0**;**

**}**

int x **=** **++**size**;**

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

std**::**nth\_element**(**point **+** l**,** point **+** mid**,** point **+** r**,** compare**);**

tree**[**x**].**reset**(**point**[**mid**]);**

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

tree**[**x**].**rectangle**.**add**(**point**[**i**]);**

**}**

tree**[**x**].**child**[**0**]** **=** build**(**l**,** mid**,** type **^** 1**);**

tree**[**x**].**child**[**1**]** **=** build**(**mid **+** 1**,** r**,** type **^** 1**);**

**return** x**;**

**}**

int insert**(**int x**,** const Point **&**p**,** int type **=** 1**)** **{**

pivot **=** type**;**

**if** **(**x **==** 0**)** **{**

tree**[++**size**].**reset**(**p**);**

**return** size**;**

**}**

tree**[**x**].**rectangle**.**add**(**p**);**

**if** **(**compare**(**p**,** tree**[**x**].**seperator**))** **{**

tree**[**x**].**child**[**0**]** **=** insert**(**tree**[**x**].**child**[**0**],** p**,** type **^** 1**);**

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

tree**[**x**].**child**[**1**]** **=** insert**(**tree**[**x**].**child**[**1**],** p**,** type **^** 1**);**

**}**

**return** x**;**

**}**

// For minimum distance

// For maximum:下面递归query时0, 1 换顺序;< and >;min and max

void query**(**int x**,** const Point **&**p**,** std**::**pair**<**long long**,** int**>** **&**answer**,** int type **=** 1**)** **{**

pivot **=** type**;**

**if** **(**x **==** 0 **||** tree**[**x**].**rectangle**.**dist**(**p**)** **>** answer**.**first**)** **{**

**return;**

**}**

answer **=** std**::**min**(**answer**,**

std**::**make\_pair**(**dist**(**tree**[**x**].**seperator**,** p**),** tree**[**x**].**seperator**.**id**));**

**if** **(**compare**(**p**,** tree**[**x**].**seperator**))** **{**

query**(**tree**[**x**].**child**[**0**],** p**,** answer**,** type **^** 1**);**

query**(**tree**[**x**].**child**[**1**],** p**,** answer**,** type **^** 1**);**

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

query**(**tree**[**x**].**child**[**1**],** p**,** answer**,** type **^** 1**);**

query**(**tree**[**x**].**child**[**0**],** p**,** answer**,** type **^** 1**);**

**}**

**}**

std**::**priority\_queue**<**std**::**pair**<**long long**,** int**>** **>** answer**;**

void query**(**int x**,** const Point **&**p**,** int k**,** int type **=** 1**)** **{**

pivot **=** type**;**

**if** **(**x **==** 0 **||** **(**int**)**answer**.**size**()** **==** k **&&** tree**[**x**].**rectangle**.**dist**(**p**)** **>** answer**.**top**().**first**)** **{**

**return;**

**}**

answer**.**push**(**std**::**make\_pair**(**dist**(**tree**[**x**].**seperator**,** p**),** tree**[**x**].**seperator**.**id**));**

**if** **((**int**)**answer**.**size**()** **>** k**)** **{**

answer**.**pop**();**

**}**

**if** **(**compare**(**p**,** tree**[**x**].**seperator**))** **{**

query**(**tree**[**x**].**child**[**0**],** p**,** k**,** type **^** 1**);**

query**(**tree**[**x**].**child**[**1**],** p**,** k**,** type **^** 1**);**

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

query**(**tree**[**x**].**child**[**1**],** p**,** k**,** type **^** 1**);**

query**(**tree**[**x**].**child**[**0**],** p**,** k**,** type **^** 1**);**

**}**

**}**

# 二次剩余

LL quadraticResidue**(**LL a**,** LL p**)**

**{**

**if** **(**pow**(**a**,** **(**p **-** 1**)** **/** 2**,** p**)** **==** p **-** 1**)** **return** **-**1**;**

LL s **=** p **-** 1**;**

int t **=** 0**;**

**for** **(;** **~**s **&** 1**;** s **>>=** 1**)** t**++;**

LL b **=** 1**;**

**for** **(;** pow**(**b**,** **(**p **-** 1**)** **/** 2**,** p**)** **%** p **==** 1**;** b**++);**

LL inva **=** inv**(**a**,** p**);**

LL x **=** pow**(**a**,** **(**s **+** 1**)** **/** 2**,** p**);**

LL e **=** pow**(**a**,** s**,** p**);**

**for** **(**int k **=** 1**;** k **<** t**;** k**++)**

**{**

**if** **(**pow**(**e**,** pow**(**2**,** t **-** **(**k **+** 1**),** p **-** 1**),** p**)** **%** p **!=** 1**)**

x **=** x **\*** pow**(**b**,** **(**pow**(**2**,** k **-** 1**,** p **-** 1**)** **\*** s**)** **%** **(**p **-** 1**),** p**)** **%** p**;**

e **=** inva **\*** x **%** p **\*** x **%** p**;**

**}**

**return** x**;**

**}**

# 求原根

