Fighting The Landlord

目录

[数论 3](#_Toc512210488)

[最大公约数 && 最小公倍数 4](#_Toc512210489)

[快速幂&&超级无敌防溢出乘法计算器 5](#_Toc512210490)

[逆元 6](#_Toc512210491)

[欧拉函数 7](#_Toc512210492)

[Euler函数线性筛＆＆线性素数筛 8](#_Toc512210493)

[Euler降幂应用实例1-瓜皮快速幂 10](#_Toc512210494)

[Euler降幂应用实例2 12](#_Toc512210495)

[Miller\_Rabbin素数判定 14](#_Toc512210496)

[杨辉三角打印组合数 16](#_Toc512210497)

[母函数 17](#_Toc512210498)

[数据结构 18](#_Toc512210499)

[zkw线段树 19](#_Toc512210500)

[01Tire求最大异或和实例 21](#_Toc512210501)

[RMQST 24](#_Toc512210502)

[Tire 25](#_Toc512210503)

[分治法 27](#_Toc512210504)

[二分搜索 28](#_Toc512210505)

[最小点对距离 29](#_Toc512210506)

[FFT 31](#_Toc512210507)

[动态规划 34](#_Toc512210508)

[KMP1 35](#_Toc512210509)

[KMP2 37](#_Toc512210510)

[输入输出挂 38](#_Toc512210511)

[accelerateIO.cpp 39](#_Toc512210512)

[accelerateIO.java 40](#_Toc512210513)

[图论 42](#_Toc512210514)

[并查集 43](#_Toc512210515)

[Kruskal 44](#_Toc512210516)

[Prim 46](#_Toc512210517)

[Dijkstra 48](#_Toc512210518)

[SPFA 51](#_Toc512210519)

[拓扑排序 53](#_Toc512210520)

# 数论

## 最大公约数 && 最小公倍数

ll gcd**(**ll a**,**ll b**)**

**{**

**return** b**==**0**?**a**:**gcd**(**b**,**a**%**b**);**

**}**

ll lcm**(**ll a**,**ll b**)**

**{**

**return** a**/**gcd**(**a**,**b**)\***b**;**

**}**

## 快速幂&&超级无敌防溢出乘法计算器

ll fpow**(**ll a**,** ll b**)**

**{**

ll ans **=** 1**;**

a **%=** MOD**;**

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

**{**

**if(**b**&**1**)**

ans **=** **(**ans **\*** a**)** **%** MOD**;**

a **=** **(**a **\*** a**)** **%** MOD**;**

b**>>=**1**;**

**}**

**return** ans**;**

**}**

ll mul**(**ll a**,** ll b**,** ll m**)**

**{**

**return** **(**a **\*** b **-** ll**(**a **/** **(**long double**)**m **\*** b **+** 1e-3**)** **\*** m **+** m**)** **%** m**;**

**}**

## 逆元

/\*

ex\_gcd() 参数: a, b(求a在模b下的逆元)

\*/

struct ex

**{**

ll \_\_\_inv**,**\_\_\_t**;**

void ex\_gcd\_fun**(**ll a**,** ll b**,**ll **&** \_\_\_inv**,**ll **&** \_\_\_t**)**

**{**

**if(!**b**)**

**{**

\_\_\_inv**=**1**;**

\_\_\_t**=**0**;**

**}**

**else**

**{**

ex\_gcd\_fun**(**b**,**a**%**b**,**\_\_\_t**,**\_\_\_inv**);**

\_\_\_t**-=**\_\_\_inv**\*(**a**/**b**);**

**}**

**}**

// 注意逆元可能为负数

ll ex\_gcd**(**ll a**,**ll b**)**

**{**

ex\_gcd\_fun**(**a**,**b**,**\_\_\_inv**,**\_\_\_t**);**

**return** \_\_\_inv**;**

**}**

**};**

## 欧拉函数

ll euler**(**ll x**)**

**{**

//if(x<=0)return std::runtime\_error("Euler error!");

ll ans **=** x**;**

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

**{**

**if(**x **%** i **==** 0**)**

**{**

ans **-=** ans **/** i**;**

**while(!(**x **%** i**)**

x**/=**i**;**

**}**

**}**

**if(**x **!=** 1**)**

ans **-=** ans **/** x**;**

**return** ans**;**

**}**

## Euler函数线性筛＆＆线性素数筛

/\*

\* 打表1~n内的欧拉函数和素数, 输出到Object.euler[], Object.prime[]内

\* 参数: n

\* Time Complexity: O(n)

\*/

struct GetEuler

**{**

int **\***prime**,** **\***euler**;**

bool **\***isprime**;**

int tot**;**

int len**;**

GetEuler**(**int \_len**)**

**{**

const int \_\_len **=** \_len **\*** 2**;**

len **=** \_len **\*** 2**;**

prime **=** **new** int **[**\_\_len**];**

euler **=** **new** int **[**\_\_len**];**

isprime **=** **new** bool **[**\_\_len**];**

solve**();**

**}**

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

**{**

memset**(**isprime**,** **true,** **sizeof(**bool**)** **\*** len**);**

memset**(**prime**,** 0**,** **sizeof(**int**)** **\*** len**);**

memset**(**euler**,** 0**,** **sizeof(**int**)** **\*** len**);**

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

**}**

void getprime**(**int n**)**

**{**

isprime**[**0**]** **=** isprime**[**1**]** **=** **false;**

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

**{**

**if(**isprime**[**i**])**

prime**[++**tot**]** **=** i**;**

**for(**int j **=** 1**;** j **<=** tot **&&** i **\*** prime**[**j**]** **<=** n**;** j**++)**

**{**

isprime**[**i **\*** prime**[**j**]]** **=** **false;**

**if(**i **%** prime**[**j**]** **==** 0**)**

**break;**

**}**

**}**

**}**

void solve**()**

**{**

init**();**

int n **=** len**;**

getprime**(**2 **\*** n**);**

euler**[**1**]** **=** 1**;**

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

**{**

**if(**isprime**[**i**])**

**{**

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

**continue;**

**}**

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

**{**

**if(**i **%** prime**[**j**])**

**continue;**

**if(**i **/** prime**[**j**]** **%** prime**[**j**])**

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

**else**

euler**[**i**]** **=** euler**[**i **/** prime**[**j**]]** **\*** prime**[**j**];**

**break;**

**}**

**}**

**}**

**};**

## Euler降幂应用实例1-瓜皮快速幂

/\*

Author: SemonChan

Problem: eoj-3394 瓜皮快速幂

Problem Source: https://acm.ecnu.edu.cn/problem/3394/

\*/

#include <bits/stdc++.h>

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

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

ll euler**(**ll n**)**

**{**

ll res **=** n**,** a **=** n**;**

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

**{**

**if** **(**a **%** i **==** 0**)**

**{**

res **=** res **/** i **\*** **(**i **-** 1**);**

**while** **(**a **%** i **==** 0**)**

a **/=** i**;**

**}**

**}**

**if** **(**a **>** 1**)**

res **=** res **/** a **\*** **(**a **-** 1**);**

**return** res**;**

**}**

ll mul**(**ll a**,** ll b**,** ll m**)**

**{**

**return** **(**a **\*** b **-** ll**(**a **/** **(**long double**)**m **\*** b **+** 1e-3**)** **\*** m **+** m**)** **%** m**;**

**}**

ll fpow**(**ll a**,** ll k**,** ll m**)**

**{**

ll base **=** 1**;**

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

**{**

**if** **(**k **&** 1**)**

base **=** mul**(**base**,** a**,** m**);**

a **=** mul**(**a**,** a**,** m**);**

k **>>=** 1**;**

**}**

**return** base**;**

**}**

ll solve**(**ll n**,** ll m**)**

**{**

**if** **(**n **==** 0 **||** m **==** 1**)**

**return** 2 **%** m**;**

**if** **(**n **==** 1**)**

**return** 4 **%** m**;**

**if** **(**n **==** 2**)**

**return** 16 **%** m**;**

**if** **(**n **==** 3**)**

**return** 65536 **%** m**;**

ll eu **=** euler**(**m**);**

**return** fpow**(**2**,** solve**(**n **-** 1**,** eu**)** **+** eu**,** m**);**

**}**

int main**()**

**{**

ll n**,** m**;**

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

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

**return** 0**;**

**}**

## Euler降幂应用实例2

/\*

Author: SemonChan

Problem: BZOJ - 3884

Problem Source: https://cn.vjudge.net/problem/HYSBZ-3884

https://www.lydsy.com/JudgeOnline/problem.php?id=3884

Solution Source: https://cn.vjudge.net/solution/13601290

\*/

#include <bits/stdc++.h>

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

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

ll euler**(**ll n**)**

**{**

ll res **=** n**,** a **=** n**;**

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

**{**

**if** **(**a **%** i **==** 0**)**

**{**

res **=** res **/** i **\*** **(**i **-** 1**);**

**while** **(**a **%** i **==** 0**)**

a **/=** i**;**

**}**

**}**

**if** **(**a **>** 1**)**

res **=** res **/** a **\*** **(**a **-** 1**);**

**return** res**;**

**}**

ll fpow**(**ll a**,** ll k**,** ll m**)**

**{**

ll base **=** 1**;**

**if** **(**a **>=** m**)**

a **=** a **%** m **+** m**;**

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

**{**

**if** **(**k **&** 1**)**

**{**

base **=** base **\*** a**;**

**if** **(**base **>=** m**)**

base **=** base **%** m **+** m**;**

**}**

a **=** a **\*** a**;**

**if** **(**a **>=** m**)**

a **=** a **%** m **+** m**;**

k **>>=** 1**;**

**}**

**return** base**;**

**}**

ll solve**(**ll m**)**

**{**

**if** **(**m **==** 1**)**

**return** 0**;**

ll eu **=** euler**(**m**);**

**return** fpow**(**2**,** solve**(**eu**)** **+** eu**,** m**);**

**}**

int main**()**

**{**

ll n**,** m**;**

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

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

**{**

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

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

**}**

**return** 0**;**

**}**

## Miller\_Rabbin素数判定

/\*

IsPrime(): 参数: n -- 需要判断的素数

Time Complexity: O(1)

错误概率: 2^(-k)

直接指定方式: n<1e6, k=2, a=2/a=3; n为32位无符号整数, k=3,a=2/a=7/a=61

\*/

struct Miller\_Rabbin

**{**

LL fpow**(**LL a**,** LL b**,** LL r**)**

**{**

LL ans**=**1**,**buff**=**a**;**

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

**{**

**if(**b**&**1**)**

ans **=** **(**ans **\*** buff**)** **%** r**;**

buff **=** **(**buff **\*** buff**)** **%** r**;**

b**>>=**1**;**

**}**

**return** ans**;**

**}**

bool Miller**(**LL n**,** LL a**)**

**{**

LL r **=** 0**,** s **=** n **-** 1**,** j**;**

**if(!(**n **%** a**))**

**return** **false;**

**while(!(**s**&**1**))**

s **>>=** 1**,** r**++;**

LL k **=** fpow**(**a**,** s**,** n**);**

**if(**k **==** 1**)**

**return** **true;**

**for(**j **=** 0**;** j **<** r**;** j**++,** k **=** k **\*** k **%** n**)**

**if(**k **==** n **-** 1**)**

**return** **true;**

**return** **false;**

**}**

bool IsPrime**(**int n**)**

**{**

**if(**n **<** 2**)**

**return** **false;**

int tab**[]** **=** **{**2**,** 3**,** 5**,** 7**};**

**for(**int i **=** 0**;** i **<** 4**;** i**++)**

**{**

**if(**n **==** tab**[**i**])**

**return** **true;**

**if(!**Miller**(**n**,** tab**[**i**]))**

**return** **false;**

**}**

**return** **true;**

**}**

**};**

## 杨辉三角打印组合数

#include <bits/stdc++.h>

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

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

const int MAX **=** 2e3 **+** 10**;**

const LL MOD **=** 1007**;**

int ans**[**MAX**][**MAX**];**

void init**()**

**{**

**for(**int i **=** 1**;** i **<=** 2005**;** i**++)**

**{**

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

rep**(**j**,** 1**,** 2005**)**

ans**[**i**][**j**]** **=** **(**ans**[**i **-** 1**][**j **-** 1**]** **+** ans**[**i **-** 1**][**j**])** **%** MOD**;**

**}**

**}**

int main**()**

**{**

init**();**

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

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

printf**(**"%d\n"**,** ans**[**n **+** 1**][**m**]);**

**return** 0**;**

**}**

## 母函数

/\*

\* init(): 参数: 所取最大值, 调用进行预处理打表

\* solve(): 返回n组合的最大值

\* Time Complexity: O(n^3)

\* 转换公式: (1+x^2+x^3+x^4+...+x^n)\*(1+x^2+x^4+x^6+x^8+...)\*(1+x^3+x^6+x^9+...) --- 共有n个表达式相乘

\* 应用实例: hdu1028 / hdu1398

\*/

struct GeneratingFunction

**{**

long long pre**[**1005**],** now**[**1005**];**

void init**(**int N**)**

**{**

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

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

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

pre**[**i**]** **=** 1**;**

**for(**int i **=** 2**;** i **<=** N**;** **++**i**)**

**{**

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

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

now**[**j **+** k**]** **+=** pre**[**j**];**

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

pre**[**j**]** **=** now**[**j**],** now**[**j**]** **=** 0**;**

**}**

**}**

long long solve**(**int n**)**

**{**

**return** pre**[**n**];**

**}**

**};**

# 数据结构

## zkw线段树

#define USE\_ZKWTREE

#ifdef USE\_ZKWTREE

**typedef** long long int64**;**

int64 how\_to\_update**(**int64 a**,**int64 b**)**

**{**

**return** a**+**b**;**

**}**

/\*\* such as max(a,b) min(a,b) ... \*\*/

class zkw\_tree

**{**

public**:**

zkw\_tree**(**int \_len**)**

**{**

int t**=**1**;**

**while(**\_len**>=**t**)**t**\*=**2**;**

len**=**t**\***2**;**

data**=new** int64 **[**len**\***2**+**5**];**

data**[**0**]=**0**;**

//if(data==NULL)throw std::runtime\_error("Memory Limit Exceeded or some other error.");

**}**

void update**(**int64 i**,**int64 x**)**

**{**

data**[**i**+=**len**]+=**x**;**

**for(**i**>>=**1**;**i**;**i**>>=**1**)**data**[**i**]=**how\_to\_update**(**data**[**i**<<**1**],**data**[**i**<<**1**|**1**]);**

**}**

int64 query**(**int s**,**int t**)**/// must make sure that s<t

**{**

int64 ans**=**0**;** /\*\* there is a point to recompose while you are not update a summation. \*\*/

**for(**s**+=**len**-**1**,**t**+=**len**+**1**;**s**^**t**^**1**;**s**>>=**1**,**t**>>=**1**)**

**{**

**if(~**s**&**1**)**ans**=**how\_to\_update**(**ans**,**data**[**s**^**1**]);**

**if(**t**&**1**)**ans**=**how\_to\_update**(**ans**,**data**[**t**^**1**]);**

**}**

**return** ans**;**

**}**

private**:**

int len**;**

int64 **\***data**;**

**};**

#endif // USE\_ZKWTREE

/\*\* This code is not to be used yet.

Welcome to hack it. \*\*/

/\*\* create a zkw\_tree: zkw\_tree treea(1000);

下标从1开始。the subscript is from 1.

更新树的下标为location为value: object.zkw\_tree::update(location,value);

查询区间[from,to]: object.zkw\_tree::query(from,to)

\*\*/

## 01Tire求最大异或和实例

/\*

Author: SemonChan

Problem：HDU-5536

Solution Source: https://cn.vjudge.net/solution/13181232

\*/

#include <stdio.h>

#include <string.h>

#include <algorithm>

#include <vector>

#include <queue>

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

struct node **{**

int num**;**

int next**[**3**];**

node**()** **{** num **=** 0**;** memset**(**next**,** 0**,** **sizeof** next**);** **}**

**};**

int n**,** cnt**;**

node tr**[**100003**];**

const int len **=** 30**;**

void init**()**

**{**

cnt **=** 0**;**

tr**[**0**]** **=** node**();**

**}**

void put**(**int temp**)**

**{**

int now **=** 0**;**

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

**{**

int k **=** bool**((**1 **<<** **(**len **-** i**))** **&** temp**);**

**if** **(**tr**[**now**].**next**[**k**]** **==** 0**)**

**{**

tr**[**now**].**next**[**k**]** **=** **++**cnt**;**

tr**[**cnt**]** **=** node**();**

**}**

now **=** tr**[**now**].**next**[**k**];**

tr**[**now**].**num**++;**

**}**

**}**

int search**(**int temp**)**

**{**

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

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

**{**

int k **=** bool**((**1 **<<** **(**len **-** i**))** **&** temp**);**

**if** **(**k**)**

**{**

**if** **(**tr**[**now**].**next**[**0**]** **&&** tr**[**tr**[**now**].**next**[**0**]].**num **>** 0**)**

now **=** tr**[**now**].**next**[**0**],** ans **<<=** 1**;**

**else**

now **=** tr**[**now**].**next**[**1**],** ans **=** ans **<<** 1 **|** 1**;**

**}**

**else**

**{**

**if** **(**tr**[**now**].**next**[**1**]** **&&** tr**[**tr**[**now**].**next**[**1**]].**num **>** 0**)**

now **=** tr**[**now**].**next**[**1**],** ans **=** ans **<<** 1 **|** 1**;**

**else**

now **=** tr**[**now**].**next**[**0**],** ans **<<=** 1**;**

**}**

**}**

**return** temp **^** ans**;**

**}**

void del**(**int temp**)**

**{**

int now **=** 0**;**

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

**{**

now **=** tr**[**now**].**next**[**bool**((**1 **<<** **(**len **-** i**))** **&** temp**)];**

tr**[**now**].**num**--;**

**}**

**}**

int s**[**1010**];**

int main**()**

**{**

int t**,** n**;**

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

**while** **(**t**--)**

**{**

init**();**

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

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

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

int ma **=** **-**1111**;**

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

**{**

del**(**s**[**i**]);**

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

**{**

del**(**s**[**j**]);**

ma **=** max**(**ma**,** search**(**s**[**i**]** **+** s**[**j**]));**

put**(**s**[**j**]);**

**}**

put**(**s**[**i**]);**

**}**

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

**}**

**return** 0**;**

**}**

## RMQST

#include <string.h>

#include <algorithm>

#include <math.h>

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

class RMQST**{**

public**:**

void init**(**int n\_**,** int v**[])**

**{**

n **=** n\_**;**

memcpy**(**Min**[**0**],** v**,** **sizeof** Min**[**0**]);**

solve**();**

**}**

int query**(**int l**,** int r**)** // O(1) 询问

**{**

int mid **=** **(**int**)((**log**(**r **-** l **+** 1**))** **/** **(**log**(**2.0**)));**

/\* 按需选择返回方式 \*/

**return** max**(**Max**[**l**][**mid**],** Max**[**r **-** **(**1 **<<** mid**)** **+** 1**][**mid**]);**

//return min(Min[l][mid], Min[r - (1 << mid) + 1][mid]);

**}**

RMQST**(){** n **=** 0**;** memset**(**Min**,** 0**,** **sizeof** Min**);** memset**(**Max**,** 0**,** **sizeof** Max**);** **}**

private**:**

static const int MAX **=** 1e5 **+** 5**;**

int n**;**

int Min**[**MAX**][**25**];**

int Max**[**MAX**][**25**];**

void solve**()** // O(NlogN) 建表

**{**

int l **=** **(**int**)((**log**(**n**))** **/** **(**log**(**2.0**)));**

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

**{**

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

**{**

Min**[**i**][**j**]** **=** min**(**Min**[**i**][**j **-** 1**],** Min**[**i **+** **(**1 **<<** **(**j **-** 1**))][**j **-** 1**]);**

Max**[**i**][**j**]** **=** max**(**Max**[**i**][**j **-** 1**],** Max**[**i **+** **(**1 **<<** **(**j **-** 1**))][**j **-** 1**]);**

**}**

**}**

**}**

**};**

## Tire

#include <algorithm>

#include <vector>

#include <string.h>

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

class Tire**{**

public**:**

void init**(**int k\_ **=** 1**)** // 初始化， k\_为预估需求空间

**{**

cnt **=** 0**;**

v**.**resize**(**k\_**);**

v**[**0**]** **=** node**();**

**}**

void put**(**char t**[],** int len**)** // 把t[]放入Tire， len为t[]长度

**{**

int now **=** 0**;**

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

**{**

**if** **(**v**[**now**].**next**[**t**[**i**]** **-** 'a'**]** **==** 0**)**

**{**

v**[**now**].**next**[**t**[**i**]** **-** 'a'**]** **=** **++**cnt**;**

**if** **(**cnt **==** v**.**size**())**

v**.**push\_back**(**node**());**

**else**

v**[**cnt**]** **=** node**();**

**}**

now **=** v**[**now**].**next**[**t**[**i**]** **-** 'a'**];**

**}**

v**[**now**].**ex **=** 1**;**

v**[**now**].**cnt**++;**

**}**

int check**(**char t**[],** int len**)** // 查找t[], 找到则返回以其为前缀的单词数量， 否则返回-1

**{**

int now **=** 0**;**

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

**{**

**if** **(**v**[**now**].**next**[**t**[**i**]** **-** 'a'**]** **==** 0**)**

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

now **=** v**[**now**].**next**[**t**[**i**]** **-** 'a'**];**

**}**

**return** v**[**now**].**ex **==** 1 **?** v**[**now**].**cnt **:** **-**1**;**

**}**

Tire**(){}**

private**:**

struct node**{**

int ex**,** cnt**;**

int next**[**30**];**

node**()** **{** ex **=** 0**;** cnt **=** 0**;** memset**(**next**,** 0**,** **sizeof** next**);** **}**

**};**

int cnt**;**

vector**<**node**>** v**;**

**};**

# 分治法

## 二分搜索

/\*

待优化

\*/

#include <bits/stdc++.h>

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

int search**(**int v**[],** int l**,** int r**,** int k**)**

**{**

**if** **(**l **>** r **||** l **<** 0 **||** r **>=** 100**)**

**return** 0**;**

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

**if** **(**v**[**mid**]** **<** k**)**

**return** search**(**v**,** mid **+** 1**,** r**,** k**);**

**if** **(**v**[**mid**]** **>** k**)**

**return** search**(**v**,** l**,** mid **-** 1**,** k**);**

**return** v**[**mid**]** **==** k**;**

**}**

int main**()**

**{**

int n**;**

int v**[**100**];**

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

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

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

sort**(**v**,** v **+** n**);**

int q**;**

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

**while** **(**q**--)**

**{**

int k**;**

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

int tmp **=** search**(**v**,** 0**,** n **-** 1**,** k**);**

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

/\*

int tmp = lower\_bound(v, v + n, k) - v;

\*/

**}**

**return** 0**;**

**}**

## 最小点对距离

#include <bits/stdc++.h>

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

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

struct node**{**

double x**,** y**;**

node**(**double x\_ **=** 0**,** double y\_ **=** 0**):**x**(**x\_**),** y**(**y\_**){}**

bool **operator** **<** **(**const node b**)**const

**{**

**return** y **<** b**.**y**;**

**}**

**};**

node v**[**100005**];**

node nd**[**100005**];**

/\* 求欧氏距离 \*/

inline double getd**(**const node a**,** const node b**)**

**{**

**return** sqrt**((**a**.**x **-** b**.**x**)** **\*** **(**a**.**x **-** b**.**x**)** **+** **(**a**.**y **-** b**.**y**)** **\*** **(**a**.**y **-** b**.**y**));**

**}**

double solve**(**int a**,** int b**)**

**{**

/\* 若区间只有一个点 \*/

**if** **(**a **==** b**)**

**return** 1e30**;**

int mid **=** **(**a **+** b**)** **/** 2**,** cnt **=** 0**;**

double d **=** min**(**solve**(**a**,** mid**),** solve**(**mid **+** 1**,** b**));** // 递归求两个子区间的最小点对距离

/\* 遍历区间，搜索与v[mid]横坐标差值小于d的点 \*/

**for** **(**int i **=** mid**;** i **>=** a**;** i**--)**

**if** **(**fabs**(**v**[**i**].**x **-** v**[**mid**].**x**)** **<** d**)**

nd**[**cnt**++]** **=** v**[**i**];**

**else**

**break;**

**for** **(**int i **=** mid **+** 1**;** i **<=** b**;** i**++)**

**if** **(**fabs**(**v**[**i**].**x **-** v**[**mid**].**x**)** **<** d**)**

nd**[**cnt**++]** **=** v**[**i**];**

**else**

**break;**

/\* 按纵坐标从小到大排序 \*/

sort**(**nd**,** nd **+** cnt**);**

/\* 枚举求区间内最小点对距离 \*/

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

**for** **(**int j **=** i **+** 1**;** j **<** cnt**;** j**++)**

**{**

**if** **(**fabs**(**nd**[**i**].**y **-** nd**[**j**].**y**)** **>** d**)**

**break;**

d **=** min**(**d**,** getd**(**nd**[**i**],** nd**[**j**]));**

**}**

**return** d**;**

**}**

bool cmp**(**const node a**,** const node b**)**

**{**

**return** a**.**x **<** b**.**x**;**

**}**

int main**()**

**{**

int n**;**

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

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

scanf**(**"%lf%lf"**,** **&**v**[**i**].**x**,** **&**v**[**i**].**y**);**

/\* 按横坐标从小到大排序 \*/

sort**(**v **+** 1**,** v **+** 1 **+** n**,** cmp**);**

printf**(**"%.2f\n"**,** solve**(**1**,** n**));**

**return** 0**;**

**}**

## FFT

#include <bits/stdc++.h>

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

const double PI **=** acos**(-**1.0**);**

//复数结构体

struct complex

**{**

double r**,**i**;**

complex**(**double \_r **=** 0.0**,**double \_i **=** 0.0**)**

**{**

r **=** \_r**;** i **=** \_i**;**

**}**

complex **operator** **+(**const complex **&**b**)**

**{**

**return** complex**(**r**+**b**.**r**,**i**+**b**.**i**);**

**}**

complex **operator** **-(**const complex **&**b**)**

**{**

**return** complex**(**r**-**b**.**r**,**i**-**b**.**i**);**

**}**

complex **operator** **\*(**const complex **&**b**)**

**{**

**return** complex**(**r**\***b**.**r**-**i**\***b**.**i**,**r**\***b**.**i**+**i**\***b**.**r**);**

**}**

**};**

/\*

\* 进行FFT和IFFT前的反转变换。

\* 位置i和 （i二进制反转后位置）互换

\* len必须去2的幂

\*/

void change**(**complex y**[],**int len**)**

**{**

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

**for(**i **=** 1**,** j **=** len**/**2**;**i **<** len**-**1**;** i**++)**

**{**

**if(**i **<** j**)**swap**(**y**[**i**],**y**[**j**]);**

//交换互为小标反转的元素，i<j保证交换一次

//i做正常的+1，j左反转类型的+1,始终保持i和j是反转的

k **=** len**/**2**;**

**while(** j **>=** k**)**

**{**

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

k **/=** 2**;**

**}**

**if(**j **<** k**)** j **+=** k**;**

**}**

**}**

/\*

\* 做FFT

\* len必须为2^k形式，

\* on==1时是DFT，on==-1时是IDFT

\*/

void fft**(**complex y**[],**int len**,**int on**)**

**{**

change**(**y**,**len**);**

**for(**int h **=** 2**;** h **<=** len**;** h **<<=** 1**)**

**{**

complex wn**(**cos**(-**on**\***2**\***PI**/**h**),**sin**(-**on**\***2**\***PI**/**h**));**

**for(**int j **=** 0**;**j **<** len**;**j**+=**h**)**

**{**

complex w**(**1**,**0**);**

**for(**int k **=** j**;**k **<** j**+**h**/**2**;**k**++)**

**{**

complex u **=** y**[**k**];**

complex t **=** w**\***y**[**k**+**h**/**2**];**

y**[**k**]** **=** u**+**t**;**

y**[**k**+**h**/**2**]** **=** u**-**t**;**

w **=** w**\***wn**;**

**}**

**}**

**}**

**if(**on **==** **-**1**)**

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

y**[**i**].**r **/=** len**;**

**}**

const int MAXN **=** 200010**;**

complex x1**[**MAXN**],**x2**[**MAXN**];**

char str1**[**MAXN**/**2**],**str2**[**MAXN**/**2**];**

int sum**[**MAXN**];**

int main**()**

**{**

**while(**scanf**(**"%s%s"**,**str1**,**str2**)==**2**)**

**{**

int len1 **=** strlen**(**str1**);**

int len2 **=** strlen**(**str2**);**

int len **=** 1**;**

**while(**len **<** len1**\***2 **||** len **<** len2**\***2**)**len**<<=**1**;**

**for(**int i **=** 0**;**i **<** len1**;**i**++)**

x1**[**i**]** **=** complex**(**str1**[**len1**-**1**-**i**]-**'0'**,**0**);**

**for(**int i **=** len1**;**i **<** len**;**i**++)**

x1**[**i**]** **=** complex**(**0**,**0**);**

**for(**int i **=** 0**;**i **<** len2**;**i**++)**

x2**[**i**]** **=** complex**(**str2**[**len2**-**1**-**i**]-**'0'**,**0**);**

**for(**int i **=** len2**;**i **<** len**;**i**++)**

x2**[**i**]** **=** complex**(**0**,**0**);**

//求DFT

fft**(**x1**,**len**,**1**);**

fft**(**x2**,**len**,**1**);**

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

x1**[**i**]** **=** x1**[**i**]\***x2**[**i**];**

fft**(**x1**,**len**,-**1**);**

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

sum**[**i**]** **=** **(**int**)(**x1**[**i**].**r**+**0.5**);**

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

**{**

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

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

**}**

len **=** len1**+**len2**-**1**;**

**while(**sum**[**len**]** **<=** 0 **&&** len **>** 0**)**len**--;**

**for(**int i **=** len**;**i **>=** 0**;**i**--)**

printf**(**"%c"**,**sum**[**i**]+**'0'**);**

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

**}**

**return** 0**;**

**}**

# 动态规划

## KMP1

/\*

\*

\* FUNCTION: 求一个串在另一个串的所有匹配位置

\*

\* TIME COMPLEXITY: O(N + M)

\*

\* setvalue() -- 参数：模式串+文本串 返回值: void

\* find\_substring() -- 参数: void 返回值: 所有匹配点的首字符下标

\*

\* P.S. 直接使用cin cout时, 流加速防止tle

\*/

class Kmp

**{**

private**:**

string pattern**,** text**;**

public**:**

void setvalue**(**string a**,** string b**)**

**{**

pattern **=** a**,** text **=** b**;**

**}**

vector**<**int**>** find\_substring**()**

**{**

int n **=** pattern**.**size**();**

vector**<**int**>** nxt**(**n **+** 1**,**0**);**

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

**{**

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

**while(**j **>** 0**)**

**{**

j **=** nxt**[**j**];**

**if(**pattern**[**j**]** **==** pattern**[**i**])**

**{**

nxt**[**i **+** 1**]** **=** j **+** 1**;**

**break;**

**}**

**}**

**}**

vector**<**int**>** pos**;**

int m **=** text**.**size**();**

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

**{**

**if(**j **<** n **&&** text**[**i**]** **==** pattern**[**j**])**

j**++;**

**else**

**{**

**while(**j **>** 0**)**

**{**

j **=** nxt**[**j**];**

**if(**text**[**i**]** **==** pattern**[**j**])**

**{**

j**++;**

**break;**

**}**

**}**

**}**

**if(**j **==** n**)**

pos**.**push\_back**(**i **-** n **+** 1**);**

**}**

**return** pos**;**

**}**

**};**

## KMP2

class Kmp

**{**

int pre**[**MAX**];**

char s**[**MAX**],** p**[**MAN**];**

void init**()**

**{**

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

**for(**int i **=** 1**,** j **=** **-**1**;** p**[**i**];** **++**i**)**

**{**

**while(**j **>=** 0 **&&** p**[**i**]** **!=** p**[**j **+** 1**])**

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

**if(**p**[**i**]** **==** p**[**j **+** 1**])**

**++**j**;**

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

**}**

**}**

void solve**(**vector**<**int**>** **&**match**)**

**{**

match**.**clear**();** prepare**();**

**for** **(**int i **=** 0**,** j **=** **-**1**;** s**[**i**];** **++**i**)**

**{**

**while(**j **>=** 0 **&&** s**[**i**]** **!=** p**[**j **+** 1**])**

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

**if(**s**[**i**]** **==** p**[**j **+** 1**])**

**++**j**;**

**if(!**p**[**j **+** 1**])**

match**.**push\_back**(**i **-** j**);**

**}**

**}**

**};**

# 输入输出挂

## accelerateIO.cpp

#include<cstdio>

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

inline ll in**()**

**{**

char c**=**getchar**();**

**while(!**isdigit**(**c**))**c**=**getchar**();**

int x**=**0**;**

**while(**isdigit**(**c**))**

**{**

x**=**x**\***10**+**c**-**'0'**;**

c**=**getchar**();**

**}**

**return** x**;**

**}**

void out**(**ll x**)**

**{**

**if(**x**>**9**)**out**(**x**/**10**);**

putchar**(**x**%**10**+**'0'**);**

**}**

#define in in()

## accelerateIO.java

import java**.**io**.**BufferedReader**;**

import java**.**io**.**IOException**;**

import java**.**io**.**InputStream**;**

import java**.**io**.**InputStreamReader**;**

import java**.**io**.**OutputStream**;**

import java**.**io**.**PrintWriter**;**

import java**.**math**.**BigInteger**;**

import java**.**util**.**StringTokenizer**;**

public class Main **{**

public static void main**(**String**[]** args**)** **{**

InputStream inputStream **=** System**.**in**;**

OutputStream outputStream **=** System**.**out**;**

InputReader in **=** **new** InputReader**(**inputStream**);**

PrintWriter out **=** **new** PrintWriter**(**outputStream**);**

Task solver **=** **new** Task**();**

solver**.**solve**(**1**,** in**,** out**);**

out**.**close**();**

**}**

static class Task **{**

public void solve**(**int testNumber**,** InputReader in**,** PrintWriter out**)** **{**

**try** **{**

**}**

**catch(**RuntimeException e**)** **{**

**return;**

**}**

**}**

**}**

static class InputReader **{**

public BufferedReader reader**;**

public StringTokenizer tokenizer**;**

public InputReader**(**InputStream stream**)** **{**

reader **=** **new** BufferedReader**(new** InputStreamReader**(**stream**),** 32768**);**

tokenizer **=** null**;**

**}**

public String next**()** **{**

**while** **(**tokenizer **==** null **||** **!**tokenizer**.**hasMoreTokens**())** **{**

**try** **{**

tokenizer **=** **new** StringTokenizer**(**reader**.**readLine**());**

**}** **catch** **(**IOException e**)** **{**

**throw** **new** RuntimeException**(**e**);**

**}**

**}**

**return** tokenizer**.**nextToken**();**

**}**

public int nextInt**()** **{**

**return** Integer**.**parseInt**(**next**());**

**}**

public long nextLong**()** **{**

**return** Long**.**parseLong**(**next**());**

**}**

public BigInteger nextBigInteger**()** **{**

**return** **new** BigInteger**(**next**());**

**}**

**}**

**}**

# 图论

## 并查集

class union\_find

**{**

public**:**

union\_find**(**int \_len**)**

**{**

opn**=**0**;**

len**=**\_len**+**5**;**

data**=NULL;**

data**=new** int**[**len**];**

//if(data==NULL)throw std::runtime\_error("ERROR!");

**for(**int i**=**1**;**i**<**len**;**i**++)**data**[**i**]=**i**;**

**}**

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

**{**

//if(x>len)throw std::runtime\_error("ERROR!");

**return** x**==**data**[**x**]?**x**:**data**[**x**]=**find**(**data**[**x**]);**

**}**

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

**{**

x**=**find**(**x**);**

y**=**find**(**y**);**

**if(**x**!=**y**)**

**{**

data**[**x**]=**y**;**

opn**++;**

**}**

**}**

int opn**;** //操作数

private**:**

int **\***data**;**

int len**;**

**};**

/\* 建一个长度为len的并查集： union\_find(len);

\* 搜索n在哪个集合内： object.find(n);

\* 合并n和m所在的集合： object.find(n,m);

\* opn存储了合并了多少个集合（并非运行了多少次merge(int,int)函数）

\*/

## Kruskal

#include <algorithm>

#include <string.h>

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

class union\_find**{** // 并查集引用

public**:**

union\_find**(**int \_len **=** 0**)**

**{**

opn**=**0**;**

len**=**\_len**+**5**;**

data**=NULL;**

data**=new** int**[**len**];**

//if(data==NULL)throw std::runtime\_error("ERROR!");

**for(**int i**=**1**;**i**<**len**;**i**++)**data**[**i**]=**i**;**

**}**

void init**()**

**{**

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

data**[**i**]** **=** i**;**

**}**

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

**{**

//if(x>len)throw std::runtime\_error("ERROR!");

**return** x**==**data**[**x**]?**x**:**data**[**x**]=**find**(**data**[**x**]);**

**}**

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

**{**

x**=**find**(**x**);**

y**=**find**(**y**);**

**if(**x**!=**y**)**

**{**

data**[**x**]=**y**;**

opn**++;**

**}**

**}**

int opn**;**

private**:**

int **\***data**;**

int len**;**

**};**

class Kruskal**{**

public**:**

void init**(**int n\_ **=** 0**)** // 初始化, n\_ 为结点数

**{**

n **=** n\_**;** cnt **=** 0**;** ans **=** 0**;**

p**.**init**();**

**}**

void addedge**(**int u**,** int v**,** int w**)** // 加边u-v

**{**

e**[++**cnt**]** **=** edge**(**u**,** v**,** w**);**

**}**

int solve**()** // 求最小生成树权值总和

**{**

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

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

**{**

int a **=** p**.**find**(**e**[**i**].**u**),** b **=** p**.**find**(**e**[**i**].**v**);**

**if** **(**a **!=** b**)**

**{**

ans **+=** e**[**i**].**w**;**

p**.**merge**(**a**,** b**);**

**}**

**}**

**return** ans**;**

**}**

Kruskal**(){** p **=** union\_find**(**MAX**);** e **=** **new** edge**[**MAX**];}**

private**:**

static const int MAX **=** 1e6 **+** 5**;**

struct edge**{**

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

edge**(**int u\_ **=** 0**,** int v\_ **=** 0**,** int w\_ **=** 0**):**u**(**u\_**),** v**(**v\_**),** w**(**w\_**){}**

bool **operator<(**const edge b**)**const

**{**

**return** w **<** b**.**w**;**

**}**

**};**

edge **\***e**;**

union\_find p**;**

int ans**,** cnt**,** n**;**

**};**

## Prim

#include <string.h>

#include <queue>

#include <algorithm>

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

class Prim**{**

public**:**

void init**(**int n\_ **=** 0**)** // 初始化, n为结点数

**{**

cnt **=** 0**;**

n **=** n\_**;**

memset**(**head**,** **-**1**,** **(**n **+** 2**)** **\*** **sizeof(**int**));**

**}**

void addedge**(**int u **=** 0**,** int v **=** 0**,** int w **=** 0**)** // 加边 u-v 双向边， 权值为w

**{**

e**[++**cnt**]** **=** edge**(**v**,** w**,** head**[**u**]);** head**[**u**]** **=** cnt**;**

e**[++**cnt**]** **=** edge**(**u**,** w**,** head**[**v**]);** head**[**v**]** **=** cnt**;**

**}**

int solve**()** // 求最小生成树权值总和

**{**

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

memset**(**dis**,** 0x3f3f3f3f**,** **(**n **+** 2**)** **\*** **sizeof(**int**));**

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

dis**[**1**]** **=** 0**;** ans **=** 0**;**

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

**{**

int u **=** q**.**top**().**u**,** tmp **=** q**.**top**().**w**;**

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

**if** **(**tmp **>** dis**[**u**])**

**continue;**

ans **+=** dis**[**u**];**

dis**[**u**]** **=** 0**;**

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

**{**

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

**if** **(**dis**[**v**]** **>** w**)**

**{**

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

q**.**push**(**node**(**v**,** w**));**

**}**

**}**

**}**

**return** ans**;**

**}**

Prim**(){** e **=** **new** edge**[**MAX **<<** 1**];** head **=** **new** int**[**MAX**];** dis **=** **new** int**[**MAX**];** **}**

private**:**

static const int MAX **=** 1e6 **+** 5**;**

int n**,** cnt**,** ans**;**

struct node**{**

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

node**(**int u\_ **=** 0**,** int w\_ **=** 0**):**u**(**u\_**),** w**(**w\_**){}**

bool **operator<(**const node b**)**const

**{**

**return** w **>** b**.**w**;**

**}**

**};**

struct edge**{**

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

int next**;**

edge**(**int v\_ **=** 0**,** int w\_ **=** 0**,** int n\_ **=** **-**1**):**v**(**v\_**),** w**(**w\_**),** next**(**n\_**){}**

**};**

edge **\***e**;**

int **\***head**,** **\***dis**;**

**};**

## Dijkstra

#include <algorithm>

#include <string.h>

#include <vector>

#include <queue>

#include <stack>

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

class Dijkstra**{**

public**:**

void init**(**int n\_ **=** 0**)** // 初始化, n\_为结点数

**{**

cnt **=** 0**;**

n **=** n\_**;**

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

**}**

void addedge**(**int u**,** int v**,** int w**)** // 单向加边u->v, w为权值

**{**

e**[**cnt**]** **=** edge**(**v**,** w**,** head**[**u**]);** head**[**u**]** **=** cnt**++;**

**}**

int solve**(**int s**,** int t**)** // 求s->t最短路程

**{**

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

memset**(**dis**,** 0x3f3f3f3f**,** **sizeof** dis**);**

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

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

q**.**push**(**node**(**s**,** 0**));**

dis**[**s**]** **=** 0**;**

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

**{**

int u **=** q**.**top**().**u**;**

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

**if** **(**u **==** t**)**

**return** dis**[**t**];**

**if** **(**vis**[**u**]** **!=** 0**)**

**continue;**

vis**[**u**]** **=** 1**;**

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

**{**

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

**if** **(!**vis**[**v**]** **&&** dis**[**v**]** **>** dis**[**u**]** **+** w**)**

**{**

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

pre**[**v**]** **=** u**;**

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

**}**

**}**

**}**

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

**}**

vector**<**int**>** getpath**(**int s**,** int t**)** // 求s->t最短路径

**{**

solve**(**s**,** t**);**

stack**<**int**>** st**;**

vector**<**int**>** ans**;**

int i **=** t**;**

**while** **(**i **!=** **-**1**)**

**{**

st**.**push**(**i**);**

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

**}**

**while** **(!**st**.**empty**())**

**{**

ans**.**push\_back**(**st**.**top**());**

st**.**pop**();**

**}**

**return** ans**;**

**}**

Dijkstra**(){}**

private**:**

static const int MAX **=** 1e5 **+** 5**;**

int cnt**,** n**;**

struct node**{**

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

node**(**int u\_ **=** 0**,** int w\_ **=** 0**):**u**(**u\_**),** w**(**w\_**){}**

bool **operator<(**const node b**)**const

**{**

**return** w **>** b**.**w**;**

**}**

**};**

struct edge**{**

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

int next**;**

edge**(**int v\_ **=** 0**,** int w\_ **=** 0**,** int n\_ **=** **-**1**):**v**(**v\_**),** w**(**w\_**),** next**(**n\_**){}**

**}**e**[**MAX**];**

int dis**[**MAX**];**

int head**[**MAX**];**

int pre**[**MAX**];**

bool vis**[**MAX**];**

**};**

## SPFA

#include <string.h>

#include <vector>

#include <queue>

#include <stack>

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

class spfa**{**

public**:**

void init**(**int n\_ **=** 0**)** // 初始化， n为结点数

**{**

cnt **=** 0**;**

n **=** n\_**;**

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

**}**

void addedge**(**int u**,** int v**,** int w**)** // 单向加边,u -> v， 权值为w

**{**

e**[**cnt**]** **=** edge**(**v**,** w**,** head**[**u**]);** head**[**u**]** **=** cnt**++;**

**}**

int solve**(**int s**,** int t**)** // 求s -> t的最短路程

**{**

queue**<**int**>** q**;**

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

memset**(**dis**,** 0x3f3f3f3f**,** **sizeof** dis**);**

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

dis**[**s**]** **=** 0**;** inq**[**s**]** **=** 1**;**

q**.**push**(**s**);**

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

**{**

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

q**.**pop**();** inq**[**u**]** **=** 0**;**

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

**{**

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

**if** **(**dis**[**v**]** **>** dis**[**u**]** **+** w**)**

**{**

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

pre**[**v**]** **=** u**;**

**if** **(!**inq**[**v**])**

q**.**push**(**v**),** inq**[**v**]** **=** 1**;**

**}**

**}**

**}**

**return** dis**[**t**]** **==** 0x3f3f3f3f **?** **-**1 **:** dis**[**t**];**

**}**

vector**<**int**>** getpath**(**int s**,** int t**)** // 求s -> t的最短路径

**{**

vector**<**int**>** p**;**

solve**(**s**,** t**);**

stack**<**int**>** ss**;**

int i **=** t**;**

**while** **(**i **!=** **-**1**)**

**{**

ss**.**push**(**i**);**

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

**}**

**while** **(!**ss**.**empty**())**

**{**

p**.**push\_back**(**ss**.**top**());**

ss**.**pop**();**

**}**

**return** p**;**

**}**

spfa**(){}**

private**:**

static const int MAX **=** 1e5 **+** 5**;**

int cnt**,** n**;**

struct edge**{**

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

int next**;**

edge**(**int v\_ **=** 0**,** int w\_ **=** 0**,** int n\_ **=** **-**1**):**v**(**v\_**),** w**(**w\_**),** next**(**n\_**){}**

**}**e**[**MAX**];**

int dis**[**MAX**];**

int head**[**MAX**];**

int pre**[**MAX**];**

bool inq**[**MAX**];**

**};**

## 拓扑排序

#include <bits/stdc++.h>

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

const int MAX **=** 505**;**

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

vector**<**int**>** e**[**MAX**];**

int deg**[**MAX**];**

vector**<**int**>** ans**;**

/\* dfs实现， 判环用， 需对所有为vis点调用一次

int vis[MAX];

bool tps(const int& u)

{

vis[u] = 1;

for (auto v : e[u])

if (vis[v] == 1 || (!vis[v] && !tps(v)))

return 0;

ans.push\_back(u);

vis[u] = 2;

return 1;

}

\*/

bool tps**()**

**{**

/\* 优先队列不是必须的 \*/

priority\_queue**<**int**,** vector**<**int**>,** greater**<**int**>** **>** q**;**

/\* 所有入度为0的点进队 \*/

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

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

q**.**push**(**i**);**

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

**{**

int u **=** q**.**top**();**

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

ans**.**push\_back**(**u**);**

**for** **(**auto v **:** e**[**u**])**

**{**

deg**[**v**]--;**

**if** **(!**deg**[**v**])** // 入度为0的点入队

q**.**push**(**v**);**

**}**

**}**

/\* 若有入度不为0的点， 则图中存在环 \*/

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

**if** **(**deg**[**i**])**

**return** 0**;**

**return** 1**;**

**}**

int main**()**

**{**

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

**while** **(**m**--)**

**{**

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

scanf**(**"%d%d"**,** **&**u**,** **&**v**);**

e**[**u**].**push\_back**(**v**);** // 建边

deg**[**v**]++;** // v入度加1

**}**

**if** **(!**tps**())**

printf**(**"No Answer\n"**);**

**else**

**{**

**for** **(**auto i **:** ans**)**

printf**(**"%d "**,** i**);**

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

**}**

**return** 0**;**

**}**