1. 全排列函数
2. 下一个排列函数：

next\_permutation(start, end, (cmp));

1. 上一个排列函数：

prev\_permutation(start, end, (cmp));

1. 链接：

[next\_permutation(a,a+n)\_如梦山河的博客-CSDN博客\_next\_permutation](https://blog.csdn.net/qq_43488547/article/details/100032724?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522163905938916780357223799%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=163905938916780357223799&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-100032724.first_rank_v2_pc_rank_v29&utm_term=next_permutation&spm=1018.2226.3001.4187)

1. 给定区间（a~a[n – 1]）内第K小的元素
2. 函数：

nth\_element(a, a + (k - 1), a + n);

1. 链接：

[STL 之 nth\_element详解\_sugarbliss-CSDN博客\_nth\_element](https://blog.csdn.net/sugarbliss/article/details/88050145?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164085343916780264051590%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164085343916780264051590&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-88050145.first_rank_v2_pc_rank_v29&utm_term=nth_element&spm=1018.2226.3001.4187)

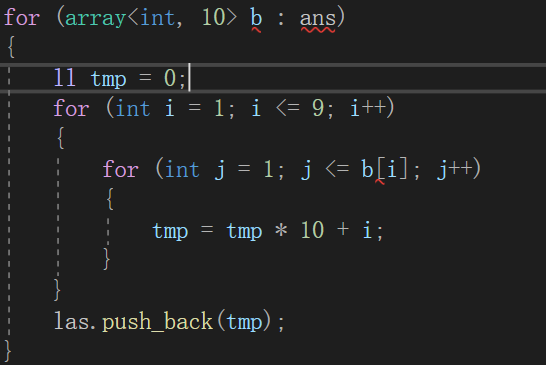
1. 用 vector 开二维数组：

vector<vector<int>> a(n, vector<int> (m));

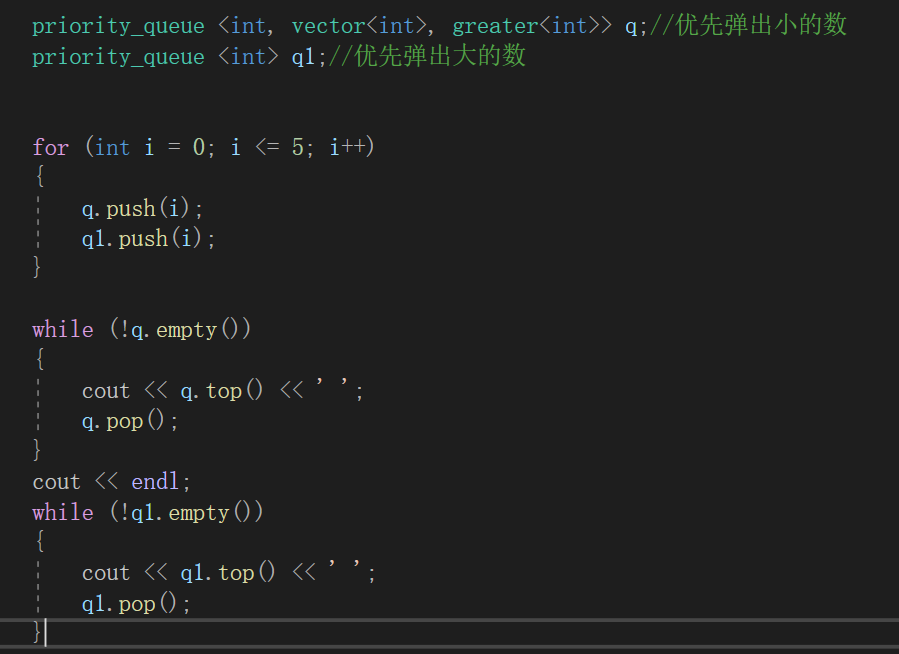
1. 用 vector 开固定大小的数组：

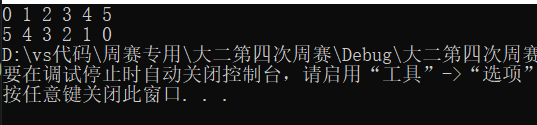
vector<arrary<int, 10>> ans;

遍历方法：

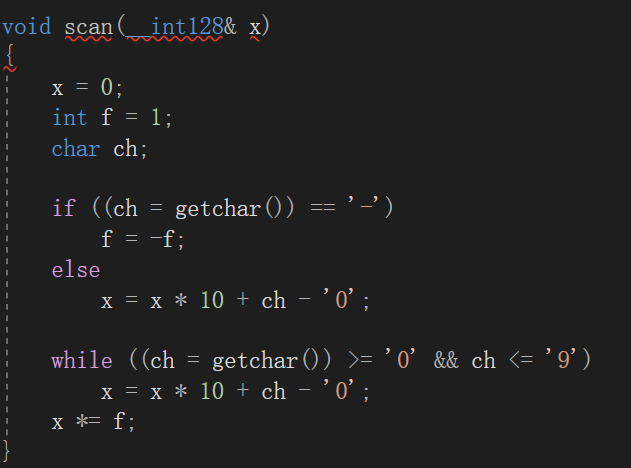


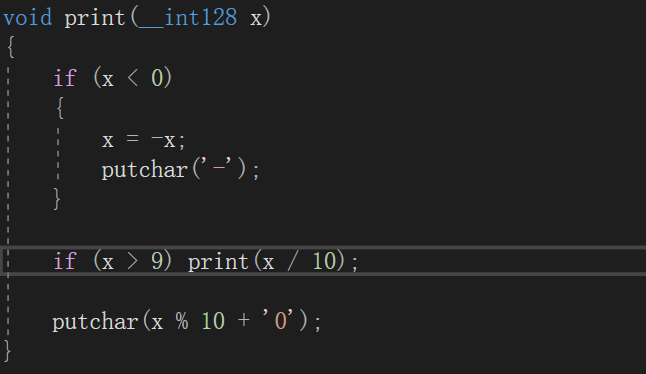
1. 优先队列
2. 小的数据先出队：priority\_queue <int, vector<int>, greater<int>> q;
3. 大的数据先出队：priority\_queue <int> q1;
4. 实例：





1. 链接：[c++优先队列(priority\_queue)用法详解\_吕白\_的博客-CSDN博客\_c++优先队列](https://blog.csdn.net/weixin_36888577/article/details/79937886?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164118657116780274123918%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164118657116780274123918&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-79937886.first_rank_v2_pc_rank_v29&utm_term=%E4%BC%98%E5%85%88%E9%98%9F%E5%88%97&spm=1018.2226.3001.4187)
2. \_\_int128的输入输出





链接：[对于大数\_\_int128的使用\_Round moon的博客-CSDN博客\_int128最大有多大](https://blog.csdn.net/qq_35339563/article/details/103940172?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164118870916780357245038%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164118870916780357245038&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~baidu_landing_v2~default-4-103940172.first_rank_v2_pc_rank_v29&utm_term=int128%E6%80%8E%E4%B9%88%E7%94%A8&spm=1018.2226.3001.4187)

1. reverse(s.begin(), s.end()); 反转所有数组

链接：[reverse()函数反转字符串以及任意类型数组\_jym蒟蒻-CSDN博客\_reverse函数反转字符串](https://blog.csdn.net/qq_40828914/article/details/81138117?ops_request_misc=&request_id=&biz_id=102&utm_term=string%E7%B1%BB%E7%9A%84%E5%AD%97%E7%AC%A6%E4%B8%B2%E5%8F%8D%E8%BD%AC%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-6-81138117.first_rank_v2_pc_rank_v29&spm=1018.2226.3001.4187)

reverse(开始地址， 结束地址 + 1);

1. unique(begin(), end())

返回去重后的尾地址。即将重复元素移动到末尾，然后返回无重复的最后一个元素的下一个元素地址。

链接：[unique()\_Dosth\_Magic-CSDN博客](https://blog.csdn.net/a197p/article/details/44055503?ops_request_misc=&request_id=&biz_id=102&utm_term=unique&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-5-44055503.first_rank_v2_pc_rank_v29&spm=1018.2226.3001.4187)

1. 删除单个元素或者连续元素

函数：erase()函数

用法：erase(地址)——删除单个元素；erase(起始地址， 结束地址)——删除连续元素

链接：[erase()函数\_如梦山河的博客-CSDN博客\_erase函数](https://blog.csdn.net/qq_43488547/article/details/100113385?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164224303516780274151003%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164224303516780274151003&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-100113385.first_rank_v2_pc_rank_v29&utm_term=erase&spm=1018.2226.3001.4187)

1. 求点到线段的最短距离

long double pointToSegDist(long double x, long double y, long double x1, long double y1, long double x2, long double y2)//点(x, y)到线段(x1, y1)~(x2, y2)的最短距离

{

long double cross = (x2 - x1) \* (x - x1) + (y2 - y1) \* (y - y1);

if (cross <= 0)

return sqrt((x - x1) \* (x - x1) + (y - y1) \* (y - y1));

long double d2 = (x2 - x1) \* (x2 - x1) + (y2 - y1) \* (y2 - y1);

if (cross >= d2)

return sqrt((x - x2) \* (x - x2) + (y - y2) \* (y - y2));

long double r = cross / d2;

long double px = x1 + (x2 - x1) \* r;

long double py = y1 + (y2 - y1) \* r;

return sqrt((x - px) \* (x - px) + (py - y) \* (py - y));

}

链接：[代码实现 求点到线段的最短距离\_t\_z\_l的博客-CSDN博客](https://blog.csdn.net/weixin_41316824/article/details/90699499?ops_request_misc=&request_id=&biz_id=102&utm_term=%E6%B1%82%E7%82%B9%E5%88%B0%E7%BA%BF%E6%AE%B5%E7%9A%84%E8%B7%9D%E7%A6%BB&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-90699499.first_rank_v2_pc_rank_v29&spm=1018.2226.3001.4187)

11．排列组合公式





12．lower\_bound() 和 upper\_bound()

在从小到大的排序数组中，

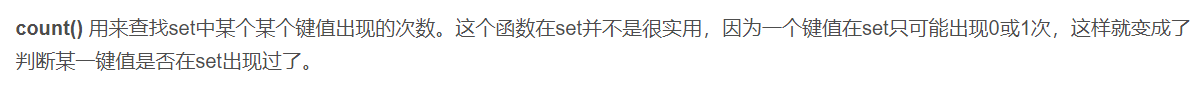
lower\_bound( begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于或等于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

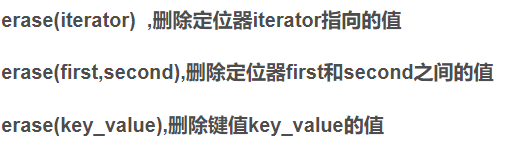
upper\_bound( begin,end,num)：从数组的begin位置到end-1位置二分查找第一个大于num的数字，找到返回该数字的地址，不存在则返回end。通过返回的地址减去起始地址begin,得到找到数字在数组中的下标。

链接：[关于lower\_bound( )和upper\_bound( )的常见用法\_brandong-CSDN博客\_lower\_bound](https://blog.csdn.net/qq_40160605/article/details/80150252?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164459342116781685391333%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164459342116781685391333&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-3-80150252.first_rank_v2_pc_rank_v29&utm_term=lower_bound%E5%92%8Cupper_bound&spm=1018.2226.3001.4187)

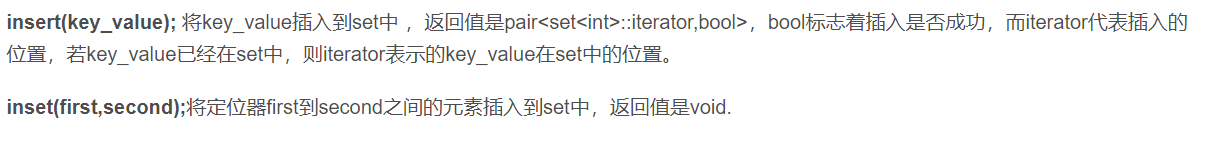
13．set 集合

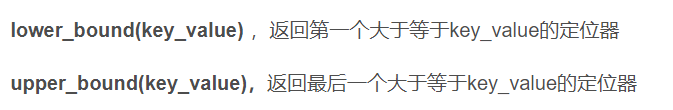






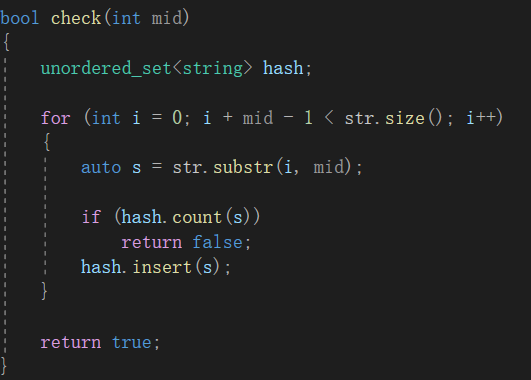






链接：[C++中set用法详解\_byn12345的博客-CSDN博客\_c++的set](https://blog.csdn.net/byn12345/article/details/79523516?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164459125516780261929129%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164459125516780261929129&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-79523516.first_rank_v2_pc_rank_v29&utm_term=set&spm=1018.2226.3001.4187)

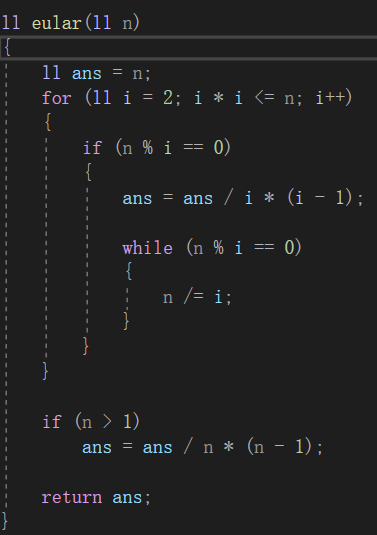
14.简单哈希表（STL版本）



链接：[unordered\_set和unordered\_map用法详解\_谢白羽-CSDN博客\_unordered\_set](https://blog.csdn.net/weixin_43679037/article/details/118833261?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164506351716780261954141%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164506351716780261954141&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_positive~default-1-118833261.first_rank_v2_pc_rank_v29&utm_term=unordered_set&spm=1018.2226.3001.4187)

15.欧拉函数（求小于n且与n互质的数据个数）

链接：[数论基础——欧拉函数\_浦柳人的博客-CSDN博客\_欧拉函数](https://blog.csdn.net/weixin_43237242/article/details/97388834?ops_request_misc=&request_id=&biz_id=102&utm_term=%E6%AC%A7%E6%8B%89%E5%87%BD%E6%95%B0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-0-97388834.first_rank_v2_pc_rank_v29&spm=1018.2226.3001.4187)



16.整数分块

用途：在2倍根号N的时间复杂度下求出∟N / x」的所有可能值

链接：[整除分块（数论分块） - <blank> - 博客园 (cnblogs.com)](https://www.cnblogs.com/0xfffe/p/9648943.html)

17.线性求组合数

[**线性求逆元 组合数**](https://www.cnblogs.com/gshdyjz/p/7642740.html)

线性求逆元：

A[1]=1;

A[i]=(p-p/i)\*A[p%i]   (mod p);

逆元的作用就是在求（a/b）%p时转变成求a×b的逆元%p。

还有一个技巧就是a×b的逆元等于a和b的逆元之积（可用于求阶乘的逆元）

求组合数：

(前面的)A[i]=A[i]\*A[i-1];（逆元累乘相当与除以阶乘）

B[0]=1;

B[i]=B[i-1]\*i;

所以组合数公式套进去就是

C(n,m)=B[n]\*A[m]\*A[n-m];

借鉴代码（有所差别，非同源，卢卡斯定理）：

#include<iostream>

#include<cstdio>

#include<cstdlib>

#include<cstring>

#include<algorithm>

#define ll long long

#define il inline

#define db double

using namespace std;

int k, n, m, p;

ll a[100045], b[100045];

ll luca(int x, int y)

{

if (x < y)

return 0;

else

if (x < p)

return b[x] \* a[y] \* a[x - y] % p;

else

return luca(x / p, y / p) \* luca(x % p, y % p) % p;

}

int main()

{

scanf("%d", &k);

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

{

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

a[0] = a[1] = b[0] = b[1] = 1;

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

a[i] = (p - p / i) \* a[p % i] % p, b[i] = b[i - 1] \* i % p;

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

a[i] = a[i - 1] \* a[i] % p;

printf("%lld\n", luca(n + m, m));

}

return 0;

}

链接：[线性求逆元 组合数 - GSHDYJZ - 博客园 (cnblogs.com)](https://www.cnblogs.com/gshdyjz/p/7642740.html)

18.矩阵快速幂

和快速幂一个道理

代码：

struct matrix

{

vector <vector<ll>> a;

int n;

matrix(int n) : a(n + 1, vector<ll>(n + 1)), n(n) {}

inline matrix operator \*(const matrix& b)const//inline用来定义内联函数，即在类中用的函数，可以加快速度。

{

matrix tmp(n + 1);

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

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

{

tmp.a[i][j] = 0;

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

{

tmp.a[i][j] += a[i][k] \* b.a[k][j];

tmp.a[i][j] %= mod;

}

}

return tmp;

}

};

matrix pow(matrix a, int k)

{

matrix tmp = a;

k--;

while (k)

{

if (k & 1)

tmp = tmp \* a;

a = a \* a;

k >>= 1;

}

return tmp;

}

链接：[P3390 【模板】矩阵快速幂 - GSHDYJZ - 博客园 (cnblogs.com)](https://www.cnblogs.com/gshdyjz/p/7134261.html)

19.hash类接口型

#include <algorithm>

#include <bitset>

#include <cassert>

#include <cmath>

#include <cstring>

#include <functional>

#include <iomanip>

#include <iostream>

#include <map>

#include <queue>

#include <set>

#include <vector>

using namespace std;

typedef unsigned long long hash\_t;

const int MX = 1000001;

const int base = 133;

// a template for string hash based on unsigned long long

vector<hash\_t> pows;

class Hash {

public:

vector<hash\_t> prefix\_hash;

int N;

Hash(const string& S) {

build(S);

}

void build(const string& S) {

N = (int)S.size();

assert(N > 0);

prefix\_hash.assign(N + 1, 0);

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

prefix\_hash[i + 1] = prefix\_hash[i] \* base + S[i];

}

if (pows.size() == 0) {

pows.push\_back(1);

}

pows.reserve(N + N / 32 + 1);

while ((int)pows.size() <= N + N / 32) {

hash\_t last = pows.back();

pows.push\_back(last \* base);

}

}

hash\_t sub\_hash(int from, int to) {

assert(0 <= from and to < N);

to += 1;

return prefix\_hash[to] - prefix\_hash[from] \* pows[to - from];

}

};

int main() {

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

cin.tie(0);

string S;

while (cin >> S and S != ".") {

int N = (int)S.size();

int ans = 1;

Hash hasher(S);

for (int k = 1; k <= N; ++k) if (N % k == 0) {

bool failed = false;

const hash\_t val = hasher.sub\_hash(0, k - 1);

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

if (hasher.sub\_hash(i, i + k - 1) != val) {

failed = true;

break;

}

}

if (!failed) {

ans = N / k;

break;

}

}

cout << ans << '\n';

}

return 0;

}

双hash类：

代码：

// copied from neal's submission and modified by me

typedef unsigned long long hash\_t;

const int MAX\_TEXT = 1e5 + 5;

const int HASH\_COUNT = 2;

const hash\_t BASE = 1280770307;

const hash\_t HASH\_MOD[] = {4255233847U, 4212947987U};

vector<hash\_t> hash\_pow[HASH\_COUNT];

class Hash {

public:

int N;

vector<hash\_t> prefix\_hash[HASH\_COUNT];

Hash(const string& str, int hash\_count = 2) {

assert(hash\_count <= HASH\_COUNT);

N = (int) str.size();

for (int h = 0; h < hash\_count; ++h) {

prefix\_hash[h].resize(N+1);

compute\_hash(str);

}

}

hash\_t compute\_hash(const string& str, int h = 0) {

hash\_t value = 0;

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

value = BASE \* value % HASH\_MOD[h] + str[i];

if (value >= HASH\_MOD[h])

value -= HASH\_MOD[h];

prefix\_hash[h][i + 1] = value;

}

return value;

}

hash\_t segment\_hash(int start, int end, int h = 0) {

assert(0 <= start and end < N);

const hash\_t mod = HASH\_MOD[h];

end += 1;

int length = end - start;

hash\_t value = prefix\_hash[h][end] + mod - hash\_pow[h][length] \* prefix\_hash[h][start] % mod;

return value >= mod ? value - mod : value;

}

hash\_t combined\_hash(int start, int end) {

return segment\_hash(start, end, 0) + (segment\_hash(start, end, 1) << 32);

}

};

void prepare\_hashes() {

for (int h = 0; h < HASH\_COUNT; h++) {

const hash\_t mod = HASH\_MOD[h];

hash\_pow[h].resize(MAX\_TEXT);

hash\_pow[h][0] = 1;

for (int i = 1; i < MAX\_TEXT; i++)

hash\_pow[h][i] = BASE \* hash\_pow[h][i - 1] % mod;

}

}

20.树形dp

代码：

//#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include <iostream>

#include <cstring>

#include <string>

#include <algorithm>

#include <queue>

#include <stack>

#include <vector>

#include <map>

#include <set>

#include <list>

#include <cstdio>

#include <utility>

#include <bitset>

#include <ctime>

#include <random>

#include <iomanip>

#include <cmath>

using namespace std;

#define IOS ios::sync\_with\_stdio(false);cin.tie(nullptr);cout.tie(nullptr);

#define endl '\n'

#define mem(f,z) memset(f,z,sizeof f)

#define INF 0x7fffffff

#define INFF 0x3f3f3f3f

#define ll long long

#define ull unsigned long long

#define PP pair<int ,int>

const int N = 1e5 + 100;

ll mod = 1000000007;

struct node

{

int t;

int apple;

int next;

};

node e[2 \* 101];

int dp[101][101];

int head[101], n, q, tot = 0;

void add(int x, int y, int z)

{

e[++tot].t = y;

e[tot].apple = z;

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

head[x] = tot;

}

void dfs(int f, int fa, int apple)

{

int son[101] = { 0 }, cnt = 0;

bool flag = 0;

for (int xun = head[f]; xun; xun = e[xun].next)

{

if (e[xun].t != fa)

{

flag = true;

son[++cnt] = xun;

dfs(e[xun].t, f, e[xun].apple);

}

}

if (!flag)

{

return;

}

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

{

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

{

int t1 = 0;

if (j - 1 >= 0)

t1 += e[son[1]].apple;

if (i - j - 1 >= 0)

t1 += e[son[2]].apple;

if (j != 0 && j != i)

{

dp[f][i] = max(dp[f][i], dp[e[son[1]].t][j - 1] + t1 + dp[e[son[2]].t][i - j - 1]);

}

else if (j == 0)

dp[f][i] = max(dp[f][i], dp[e[son[2]].t][i - j - 1] + t1);

else

dp[f][i] = max(dp[f][i], dp[e[son[1]].t][j - 1] + t1);

}

}

}

int main()

{

IOS;

cin >> n >> q;

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

{

int x, y, z;

cin >> x >> y >> z;

add(x, y, z);

add(y, x, z);

}

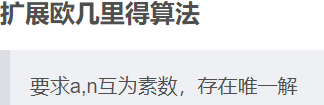
dfs(1, 0, 0);

cout << dp[1][q] << endl;

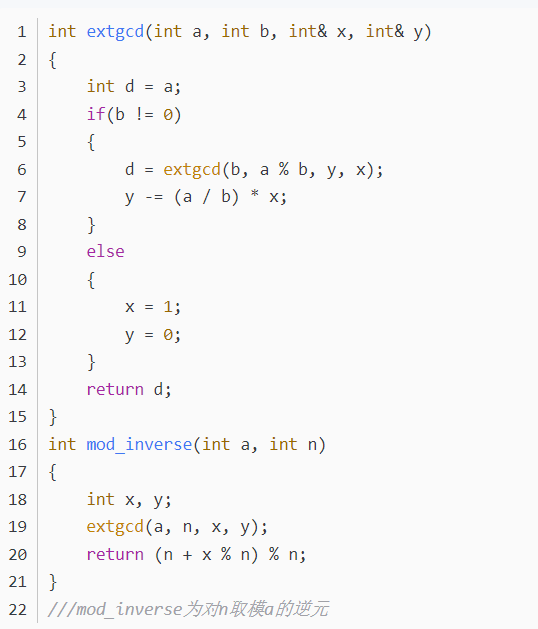
return 0;

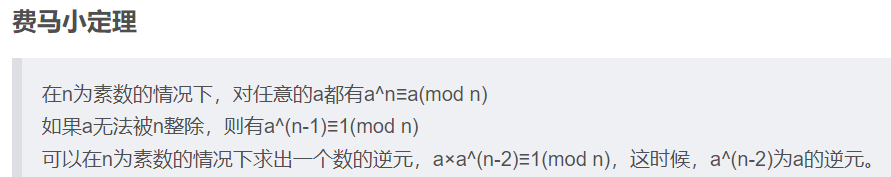
}

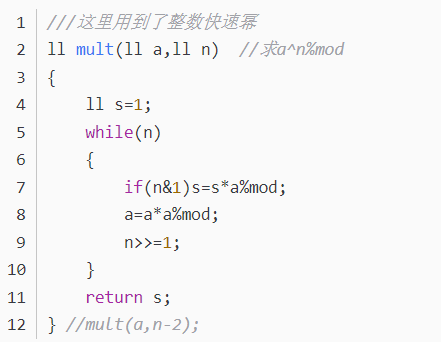
21．除法逆元：



注意：要求模数和取逆元数互质



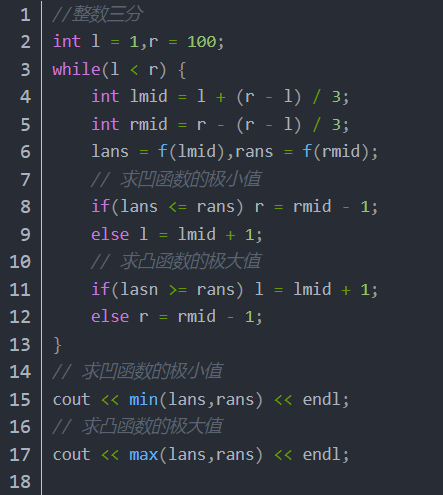


注意费马小定理要求模数必须是质数

补：x的y次方在模意义下的逆元可以表示为：ksm（x， mod – 1 – y）;

22.三分

原本我以为自己已经完全懂了三分，但是被人问起的时候才发现自己没办法快速的打出三分，所以还是贴一份例子在这里提醒自己吧。（注意：代码不一定正确，仅供参考）



23.线性求组合数（升级版）

ll Finv[N], fac[N], inv[N];//Finv是阶乘的逆元，fac是阶乘，inv是逆元

void init(int n)//n<N

{

inv[1] = 1;

for(int i = 2; i <= n; ++i) inv[i] = ((mod - mod / i) \* inv[mod % i]) % mod;

fac[0] = Finv[0] = 1;

for(int i = 1; i <= n; ++i) fac[i] = fac[i-1] \* i % mod, Finv[i] = Finv[i-1] \* inv[i] % mod;

}

24.树状数组（类版）

struct Fenwick{

using node=int;

vector<node>dat;

int n;

Fenwick(int n):dat(n+1),n(n){}

void add(int x,node v){

for(;x<=n;x+=x&-x)

dat[x]+=v;

}

node get(int x){

node ans=node(0);

for(;x>=1;x-=x&-x)

ans+=dat[x];

return ans;

}

};

25.大数分解为质因数

ull modmul(ull a, ull b, ull M) {

ll ret = a \* b - M \* ull(1.L / M \* a \* b);

return ret + M \* (ret < 0) - M \* (ret >= (ll)M);

}

ull modpow(ull b, ull e, ull mod) {

ull ans = 1;

for (; e; b = modmul(b, b, mod), e /= 2)

if (e & 1) ans = modmul(ans, b, mod);

return ans;

}

bool isPrime(ull n) {

if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;

ull A[] = { 2, 325, 9375, 28178, 450775, 9780504, 1795265022 },

s = \_\_builtin\_ctzll(n - 1), d = n >> s;

for (ull a : A) { // ^ count trailing zeroes

ull p = modpow(a % n, d, n), i = s;

while (p != 1 && p != n - 1 && a % n && i--)

p = modmul(p, p, n);

if (p != n - 1 && i != s) return 0;

}

return 1;

}

ull pollard(ull n) {

auto f = [n](ull x) { return modmul(x, x, n) + 1; };

ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;

while (t++ % 40 || \_\_gcd(prd, n) == 1) {

if (x == y) x = ++i, y = f(x);

if ((q = modmul(prd, max(x, y) - min(x, y), n))) prd = q;

x = f(x), y = f(f(y));

}

return \_\_gcd(prd, n);

}

vector<ull> factor(ull n) {//factor(n)的作用是将n分解为质因数，

if (n == 1) return {};//并将所有质因数放在vector中返回

if (isPrime(n)) return { n };

ull x = pollard(n);

auto l = factor(x), r = factor(n / x);

l.insert(l.end(), r.begin(), r.end());

return l;

}

26.迪杰斯特拉（求最短路算法）

typedef pair<int, int> PII;

const int N = 2e5 + 5;

int u, v, c, dist[N], h[N], idx, e[N], ne[N], w[N];

bool st[N];

void add(int a, int b, int c)

{

e[idx] = b, w[idx] = c, ne[idx] = h[a], h[a] = idx++;

}

void dijkstra(int startx)

{

memset(dist, 0x3f, sizeof dist);

memset(st, 0, sizeof st);

dist[startx] = 0;

priority\_queue<PII, vector<PII>, greater<PII>> heap;

heap.push({ 0, startx });

while (heap.size())

{

auto t = heap.top();

heap.pop();

int ver = t.second, distance = t.first;

if (st[ver]) continue;

st[ver] = true;

for (int i = h[ver]; i != -1; i = ne[i])

{

int j = e[i];

if (dist[j] > dist[ver] + w[i])

{

dist[j] = dist[ver] + w[i];

heap.push({ dist[j], j });

}

}

}

return;

}

27.叉乘

两个向量a，b，现在定义一个新的值：K=a.x \* b.y – a.y \* b.x ，则|K|可以表示a与b向量构成的平行四边形的面积；如果K>0，那么a旋转到b的角度为<180度，如果K<0, 那么a旋转到b的角度>180度，如果K=0，那么a与b平行。

链接：[数学基础 —— 向量运算（叉乘）\_keng\_s的博客-CSDN博客\_向量叉乘](https://blog.csdn.net/keng_s/article/details/52131034?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522164915960616780271588150%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=164915960616780271588150&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-3-52131034.142%5ev5%5epc_search_insert_es_download,157%5ev4%5econtrol&utm_term=%E5%8F%89%E4%B9%98&spm=1018.2226.3001.4187)

28.极角排序

const int N = 2e6 + 10;

const long double pi = acosl(-1);

long double a[N];

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

{

int x, y;

cin >> x >> y;

a[i] = atan2l(1.0 \* y, 1.0 \* x);//a[i]大小从负圆周率到正圆周率

}

sort(a + 1, a + n + 1);//极角排序完成，从第三象限开始逆时针排序

//下附上寻找圆上最少点组成的半环方法

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

{

a[i + n] = a[i] + 2 \* pi;

}

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

{

while (j <= 2 \* n && a[j] - a[i] < pi)

j++;

re = min(re, j - i - 1);

}

cout << re << endl;

极角排序方法二

struct node

{

ll x, y;

};

int xx(node a)//判断象限

{

if (a.x > 0 && a.y >= 0)return 1;

if (a.x <= 0 && a.y > 0)return 2;

if (a.x < 0 && a.y <= 0)return 3;

if (a.x >= 0 && a.y < 0)return 4;

}

ll cj(node a, node b)//叉积计算

{

return a.x \* b.y - a.y \* b.x;

}

bool cmp(node a, node b)

{

if (xx(a) == xx(b))return cj(a, b) > 0;//逆时针排序

else return xx(a) < xx(b);

}

//使用

vector<node> a;

sort(a.begin(), a.end(), cmp);

29.判断两条线段是否相交

struct Line

{

double x1, y1, x2, y2;

};

bool check(Line l1, Line l2)

{

//快速排斥实验

if (max(l1.x1, l1.x2) < min(l2.x1, l2.x2) || max(l1.y1, l1.y2) < min(l2.y1, l2.y2) || min(l1.x1, l1.x2) > max(l2.x1, l2.x2) || min(l1.y1, l1.y2) > max(l2.y1, l2.y2))

return 0;

//跨立实验

if ((((l1.x1 - l2.x1) \* (l2.y2 - l2.y1) - (l1.y1 - l2.y1) \* (l2.x2 - l2.x1)) \* ((l1.x2 - l2.x1) \* (l2.y2 - l2.y1) - (l1.y2 - l2.y1) \* (l2.x2 - l2.x1))) > 0 || (((l2.x1 - l1.x1) \* (l1.y2 - l1.y1) - (l2.y1 - l1.y1) \* (l1.x2 - l1.x1)) \* ((l2.x2 - l1.x1) \* (l1.y2 - l1.y1) - (l2.y2 - l1.y1) \* (l1.x2 - l1.x1))) > 0)

return 0;

return 1;

}

30.求两直线的交点

struct Line

{

double x1, y1, x2, y2;

Line() :x1(0), y1(0), x2(0), y2(0) {}

Line(int x1, int y1, int x2, int y2) :x1(x1), y1(y1), x2(x2), y2(y2) {}

};

Line jiaodian(Line l1, Line l2)//求两条直线的交点

{

Line ans;

double d = (l2.y1 - l2.y2) \* (l1.x1 - l1.x2) - (l1.y1 - l1.y2) \* (l2.x1 - l2.x2);

if (d == 0)//如果两条直线平行则返回两个坐标不同的点

{

ans.x1 = 0;

ans.y1 = 0;

ans.x2 = 1;

ans.y2 = 1;

return ans;

}

ans.y1 = ans.y2 = ((l2.y1 - l2.y2) \* (l1.y2 \* l1.x1 - l1.y1 \* l1.x2) - (l1.y1 - l1.y2) \* (l2.y2 \* l2.x1 - l2.y1 \* l2.x2)) / d;

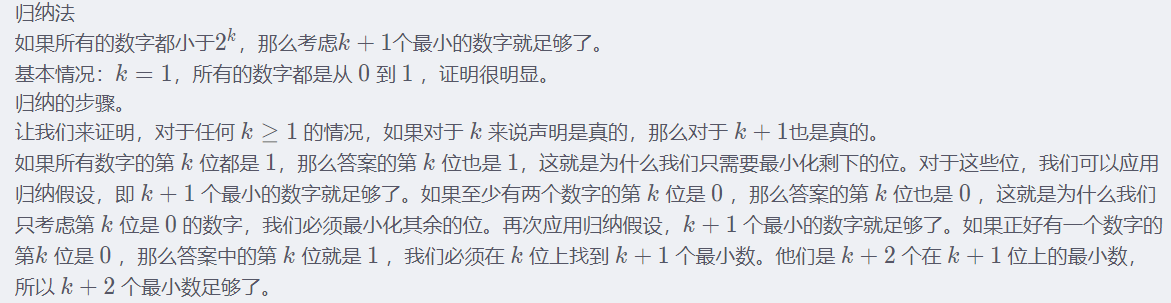
ans.x1 = ans.x2 = ((l2.x1 - l2.x2) \* (l1.x2 \* l1.y1 - l1.x1 \* l1.y2) - (l1.x1 - l1.x2) \* (l2.x2 \* l2.y1 - l2.x1 \* l2.y2)) / (-d);

return ans;

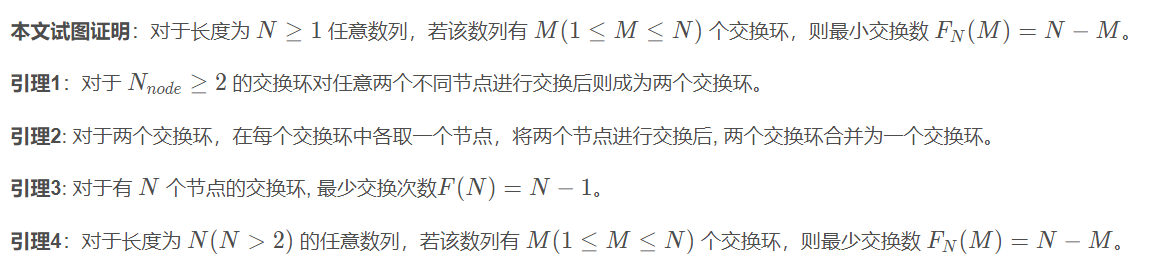
}

31.求区间最小或值





32.交换环的相关性质



链接：[排序算法-最少交换次数证明\_玉曦的博客-CSDN博客\_交换次数最少的排序算法](https://blog.csdn.net/yunxiaoqinghe/article/details/113153795)

33.数组离散化

int n;

cin >> n;

vector<int> a(n + 1), b(n + 1);

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

{

cin >> a[i];

b[i] = a[i];

}

sort(b.begin() + 1, b.begin() + n + 1);

int len = unique(b.begin() + 1, b.begin() + n + 1) - b.begin();

b.erase(b.begin() + len, b.end());

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

{

a[i] = lower\_bound(b.begin() + 1, b.end(), a[i]) - b.begin();

}

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

{

cout << a[i] << ' ';

}

cout << endl;

34.FFT（多项式相乘计算各项系数）

const int maxn = 2100000; //由于最后要取2的幂次，maxn要适当放大一些

const double PI = acos(-1.0);

struct Complex {

double x, y;

Complex(double xx = 0, double yy = 0) {

x = xx; y = yy;

}

}a[maxn + 5], b[maxn + 5];

Complex operator + (Complex a, Complex b) {

return Complex(a.x + b.x, a.y + b.y);

}

Complex operator - (Complex a, Complex b) {

return Complex(a.x - b.x, a.y - b.y);

}

Complex operator \* (Complex a, Complex b) {

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

}

int r[maxn + 5];

int sum = 1, l = 0;

void FFT(Complex\* a, int tmp) {

for (int i = 0; i < sum; ++i) {

if (i < r[i]) {

swap(a[i], a[r[i]]);

}

}

for (int i = 1; i < sum; i <<= 1) { //区间长度

Complex d(cos(PI / i), tmp \* sin(PI / i));

int len = i << 1;

for (int j = 0; j < sum; j += len) { //区间起始位置

Complex p(1, 0);

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

Complex tmp = p \* a[j + i + k];

a[j + i + k] = a[j + k] - tmp;

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

p = p \* d;

}

}

}

}

void useFFT(int n, Complex\* a, int m, Complex\* b, vector<int>& ans)//n次多项式\*m次多项式, 结果放入ans中

{

memset(r, 0, sizeof r);

while (sum <= n + m)

{

sum <<= 1;

l++;

}

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

{

r[i] = (r[i >> 1] >> 1) | ((i & 1) << (l - 1));

}

FFT(a, 1);

FFT(b, 1);

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

{

a[i] = a[i] \* b[i];

}

FFT(a, -1);

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

{

ans[i] = (int)(a[i].x / sum + 0.5);

}

}

35.st表

struct rmq\_st

{

vector<vector<int>> st;

vector<int> lg;

int n;

rmq\_st(int n, int m, vector<int>& a) :st(n + 1, vector<int>(m + 1)), lg(n + 2), n(n)

{

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

{

if (i != 1)

lg[i] = lg[i / 2] + 1;

st[i][0] = a[i];

}

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

{

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

{

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

}

}

}

int get(int l, int r)

{

int len = lg[r - l + 1];

return min(st[l][len], st[r - (1 << len) + 1][len]);

}

};

36.线段树

struct node

{

int l, r;

long long sum;

long long lazy;

};

struct segtree

{

vector<node> tree;

int n;

segtree(int n, vector<int>& a) :tree(4 \* n + 10), n(n) {

build(1, 1, n, a);

}

void push\_up(int i){

tree[i].sum = tree[i \* 2].sum + tree[i \* 2 + 1].sum;

}

void push\_down(int i) {

if (tree[i].lazy == 0)

return;

tree[i \* 2].sum += (tree[i \* 2].r - tree[i \* 2].l + 1) \* tree[i].lazy;

tree[i \* 2].lazy += tree[i].lazy;

tree[i \* 2 + 1].sum += (tree[i \* 2 + 1].r - tree[i \* 2 + 1].l + 1) \* tree[i].lazy;

tree[i \* 2 + 1].lazy += tree[i].lazy;

tree[i].lazy = 0;

}

void build(int i, int l, int r, vector<int>& a) {

tree[i].l = l;

tree[i].r = r;

tree[i].lazy = 0;

if (l == r){

tree[i].sum = a[l];

return;

}

int mid = l + r >> 1;

build(i \* 2, l, mid, a);

build(i \* 2 + 1, mid + 1, r, a);

push\_up(i);

}

void update(int i, int l, int r, ll val){

if (tree[i].l > r || tree[i].r < l)

return;

if (tree[i].l >= l && tree[i].r <= r) {

tree[i].sum += (tree[i].r - tree[i].l + 1) \* val;

tree[i].lazy += val;

return;

}

push\_down(i);

update(i \* 2, l, r, val);

update(i \* 2 + 1, l, r, val);

push\_up(i);

}

ll query(int i, int l, int r)

{

if (tree[i].l > r || tree[i].r < l)

return 0;

if (tree[i].l >= l && tree[i].r <= r)

return tree[i].sum;

push\_down(i);

return query(i \* 2, l, r) + query(i \* 2 + 1, l, r);

}

};

37.树链剖分（结合上面线段树使用）

vector<vector<int>> g(100020);//存图

vector<int> depth(100020);//深度

vector<int> siz(100020);//子树大小

vector<int> son(100020);//重儿子

vector<int> nid(100020);//节点新编号

vector<int> top(100020);//链顶点

vector<int> f(100020);//父亲节点

int cnt = 1;//节点新编号从1开始

void dfs1(int u, int fa)

{

f[u] = fa;

siz[u] = 1;

son[u] = 0;

depth[u] = depth[fa] + 1;

for (auto v : g[u]) {

if (depth[v])

continue;

dfs1(v, u);

siz[u] += siz[v];

if (siz[v] > siz[son[u]])

son[u] = v;

}

}

void dfs2(int u, int topx)

{

top[u] = topx;

nid[u] = cnt++;

if (son[u])

dfs2(son[u], topx);

else

return;

for (auto v : g[u])

{

if (v == son[u] || v == f[u])

continue;

dfs2(v, v);

}

}

int LCA(int x, int y) {//求LCA

while (top[x] != top[y]) {

if (depth[top[x]] < depth[top[y]])

swap(x, y);

x = f[top[x]];

}

if (depth[x] > depth[y])

swap(x, y);

return x;

}

void up\_road(int x, int y, int val, segtree& st)//路径加减

{

while (top[x] != top[y]) {

if (depth[top[x]] < depth[top[y]])

swap(x, y);

st.update(1, nid[top[x]], nid[x], val);

x = f[top[x]];

}

if (depth[x] > depth[y])

swap(x, y);

st.update(1, nid[x], nid[y], val);

}

ll q\_road(int x, int y, segtree& st) {// 路径查询

ll ans = 0;

while (top[x] != top[y]) {

if (depth[top[x]] < depth[top[y]])

swap(x, y);

ans += st.query(1, nid[top[x]], nid[x]);

x = f[top[x]];

}

if (depth[x] > depth[y])

swap(x, y);

ans += st.query(1, nid[x], nid[y]);

return ans;

}

ll q\_d(int u, segtree& st)//单点查询

{

return st.query(1, nid[u], nid[u]);

}

void up\_subtree(int u, int val, segtree& st) {//子树修改

st.update(1, nid[u], nid[u] + siz[u] - 1, val);

}

ll q\_subtree(int u, segtree& st)//子树查询

{

return st.query(1, nid[u], nid[u] + siz[u] - 1);

}

//使用

vector<int> a(n + 1);

for (int i = 1; i <= n; i++) {//节点初值

cin >> a[i];

}

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

int u, v;

cin >> u >> v;

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

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

}

dfs1(root, root);

dfs2(root, root);

vector<int> b(n + 1);

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

b[nid[i]] = a[i];

}

segtree st(n, b);

链接：[树链剖分（轻重链）入门\_追随远方的某R的博客-CSDN博客](https://blog.csdn.net/qq_35866893/article/details/123957047?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522165897791516782395370835%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=165897791516782395370835&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~top_click~default-2-123957047-null-null.142%5ev35%5eexperiment_28w_v1,185%5ev2%5econtrol&utm_term=%E6%A0%91%E9%93%BE%E5%89%96%E5%88%86&spm=1018.2226.3001.4187)

38.虚树（采用树链剖分的LCA做法，过程中顺便得出dfs序）

vector<vector<pair<int,ll>>> g(N);

vector<vector<int>> g1(N);

vector<int> depth(N), siz(N), son(N), f(N), top(N), dfsn(N);

vector<ll> c(N, 1e17);

vector<int> mark(N);

int cnt = 1;

void dfs1(int u, int fa) {

depth[u] = depth[fa] + 1;

siz[u] = 1;

son[u] = 0;

dfsn[u] = cnt++;

f[u] = fa;

for (auto [v, len] : g[u]) {

if (v == fa)

continue;

c[v] = min(len, c[u]);

dfs1(v, u);

siz[u] += siz[v];

if (siz[v] > siz[son[u]])

son[u] = v;

}

}

void dfs2(int u, int topx) {

top[u] = topx;

if (son[u])

dfs2(son[u], topx);

else

return;

for (auto [v, len] : g[u]) {

if (v == f[u] || v == son[u])

continue;

dfs2(v, v);

}

}

int LCA(int x, int y) {

while (top[x] != top[y]) {

if (depth[top[x]] < depth[top[y]])

swap(x, y);

x = f[top[x]];

}

if (depth[x] > depth[y])

swap(x, y);

return x;

}

bool cmp(int a, int b) {

return dfsn[a] < dfsn[b];

}

void insert(int x, stack<int>& st) {

if (st.size() == 1) {

st.push(x);

return;

}

int lca = LCA(x, st.top());

if (lca == st.top()) {

st.push(x);

return;

}

int d = st.top();

st.pop();

while (!st.empty() && dfsn[lca] <= dfsn[st.top()]) {

g1[st.top()].push\_back(d);

d = st.top();

st.pop();

}

if (d != lca) {

g1[lca].push\_back(d);

}

st.push(lca);

st.push(x);

}

ll dfs(int u) {

ll cost = 0;

for (auto v : g1[u]) {

cost += dfs(v);

}

if (mark[u])

cost = c[u];

else if (u != 1)

cost = min(cost, c[u]);

g1[u].clear();

return cost;

}

int main()

{

int n;

cin >> n;

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

int u, v, w;

cin >> u >> v >> w;

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

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

}

dfs1(1, 1);

dfs2(1, 1);

int q;

cin >> q;

while (q--) {

int k;

cin >> k;

vector<int> a(k + 1);

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

cin >> a[i];

sort(a.begin() + 1, a.begin() + k + 1, cmp);

stack<int> st;

st.push(1);

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

insert(a[i], st);

mark[a[i]] = 1;

}

while (!st.empty()) {

int d = st.top();

st.pop();

if (!st.empty())

g1[st.top()].push\_back(d);

}

cout << dfs(1) << endl;

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

mark[a[i]] = 0;

}

}

return 0;

}

链接：[虚树-树上动态规划的利器\_phython96的博客-CSDN博客](https://blog.csdn.net/weixin_37517391/article/details/82744605?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522165890474616780357237110%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=165890474616780357237110&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~baidu_landing_v2~default-5-82744605-null-null.142%5ev35%5eexperiment_28w_v1,185%5ev2%5econtrol&utm_term=%E8%99%9A%E6%A0%91&spm=1018.2226.3001.4187)

39.博弈打表

运用bitset<>的一种sg打表方式（后附例题链接）

const int N = 9;

bitset<N> s[100];

vector<vector<int>> ans(100 + 1, vector<int> (100 + 1));

int mex(bitset<N> s){//得出状态sg值

int i = 0;

while (s[i])

i++;

return i;

}

void getSG(){

for (int i = 2; i <= 10; i++)

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

s[i].set(ans[j][i - j] = mex(s[j] | s[i - j]));//枚举合并

for (int i = 1; i <= 9; i++){//输出矩阵

for (int j = 1; j <= 9; j++){

cout << ans[i][j] << " \n"[j == 9];

}

}

cout << endl;

for (int i = 1; i <= 10; i++){//输出对于每一个a，所有c + d = a的(c,d)的SG值集合

cout << i << ": sg " << mex(s[i]) << ' ' << s[i] << endl;

}

}

链接：[P2148 [SDOI2009] E&D - 洛谷 | 计算机科学教育新生态 (luogu.com.cn)](https://www.luogu.com.cn/problem/P2148)

两种博客sg打表方法（后附博客链接）

//f[]：可以取走的石子个数

//sg[]:0~n的SG函数值

//hash[]:mex{}

int f[N],sg[N],hash[N];

void getSG(int n)

{

int i,j;

memset(sg,0,sizeof(sg));

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

{

memset(hash,0,sizeof(hash));

for(j=1;f[j]<=i;j++)

hash[sg[i-f[j]]]=1;

for(j=0;j<=n;j++) //求mes{}中未出现的最小的非负整数

{

if(hash[j]==0)

{

sg[i]=j;

break;

}

}

}

}

Dfs打表法

//注意 S数组要按从小到大排序 SG函数要初始化为-1 对于每个集合只需初始化1遍

//n是集合s的大小 S[i]是定义的特殊取法规则的数组

int s[110],sg[10010],n;

int SG\_dfs(int x)

{

int i;

if(sg[x]!=-1)

return sg[x];

bool vis[110];

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

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

{

if(x>=s[i])

{

SG\_dfs(x-s[i]);

vis[sg[x-s[i]]]=1;

}

}

int e;

for(i=0;;i++)

if(!vis[i])

{

e=i;

break;

}

return sg[x]=e;

}

链接： [博弈论 SG函数\_Astronomical的博客-CSDN博客\_sg函数](https://blog.csdn.net/strangedbly/article/details/51137432)

40.使用sort（）函数进行排序时，自定义cmp函数，当前一个和后一个元素相等时一定要返回false，否则会RE或者TLE。

41. Convex hull trick

是一种算法或者说数据结构，用于在一组线性函数（形如y=mi\*x+bi）中，每次查询给以具体的x，可以快速求出最大/最小的y。这里是用来求最大值的，外部使用的时候直线要按斜率升序添加。要改成求最小值也很容易：二分改成求最小值，另外外部使用的时候直线要按降序依次添加。

struct Line

{

LL m,b;

LL get(LL x)

{

return m\*x+b;

}

};

struct ConvexHull

{

int size;

vector<Line> hull;

ConvexHull(int maxSize)

{

hull=vector<Line>(maxSize+1);

size=0;

}

bool isBad(int l1,int l2,int l3)

{

double left=1.0\*(hull[l3].b-hull[l1].b)/(hull[l1].m-hull[l3].m);

double right=1.0\*(hull[l1].b-hull[l2].b)/(hull[l2].m-hull[l1].m);

return left<right;

}

void addLine(LL m,LL b)

{

hull[size++]=Line{m,b};

while(size>2&&isBad(size-3,size-2,size-1))

{

hull[size-2]=hull[size-1];

--size;

}

}

LL query(LL x)

{

int l=0,r=size-1;

while(l<r)

{

int m=(l+r)/2;

if(hull[m].get(x)<=hull[m+1].get(x))

l=m+1;

else

r=m;

}

return hull[l].get(x);

}

};

42.后缀数组

sa[]数组表示排名第i的串是下标从sa[i]开始的；

rank[]数组表示下标从i开始的串排名第rank[i];

height[]数组表示排名第i的串与排名第i – 1的串的公共前缀长为height[i].

void suffix\_array(string &s, vector<int>& sa, vector<int>& rank, vector<int>& height){

int n = s.size(), m = max(n + 1, 129);

vector<int> x(n + 10), y(n + 10), c(m + 10);

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

x[i] = s[i];

c[x[i]]++;

}

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

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

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

sa[--c[x[i]]] = i;

}

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

int p = 0;

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

y[p++] = i;

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

if (sa[i] >= j)

y[p++] = sa[i] - j;

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

c[i] = 0;

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

c[x[i]]++;

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

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

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

sa[--c[x[y[i]]]] = y[i];

swap(x, y);

p = 1;

x[sa[0]] = 0;

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

if (check(y, sa[i - 1], sa[i], j, n))

x[sa[i]] = p - 1;

else

x[sa[i]] = p++;

}

m = p;

if (p >= n)

break;

}

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

rank[sa[i]] = i;

}

int k = 0;

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

if (k)

k--;

if (rank[i] == 0){

continue;

}

int j = sa[rank[i] - 1];

while (i + k < n && j + k < n && s[i + k] == s[j + k])k++;

height[rank[i]] = k;

}

}

43. Manacher（马拉车）

求最长回文子串，Mp[]数组中存放对应位置的最长回文串长度，结果要减一，原因细品

void Manacher(string& s, vector<int>& Mp){

string s1;

s1 += "$#";

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

s1 += s[i];

s1 += '#';

}

s1 += '\0';

int mx = 0, id = 0;

for (int i = 0; i < s1.size(); i++){

Mp[i] = mx > i ? min(Mp[2 \* id - i], mx - i) : 1;

while (s1[i + Mp[i]] == s1[i - Mp[i]])

Mp[i]++;

if (i + Mp[i] > mx){

mx = i + Mp[i];

id = i;

}

}

}

//使用

string s;

cin>>s;

vector<int> Mp(s.size() \* 2 + 10);

Manacher(s, Mp);

int ans = 0;

for (int i = 0; i < s.size() \* 2 + 2; i++){

ans = max(ans, Mp[i] - 1);

}

cout << ans << endl;

44.tarjan求割点

void tarjan(int u, int fa, vector<vector<int>>& g, vector<int>& low, vector<int>& dfn, vector<int>& cut, int& t){

low[u] = dfn[u] = t++;

int child = 0;

for (auto v : g[u]){

if (v == fa)

continue;

if (dfn[v]){

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

continue;

}

child++;

tarjan(v, u, g, low, dfn, cut, t);

if (low[v] >= dfn[u] && fa != 0)

cut[u] = 1;

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

}

if (child > 1 && fa == 0)

cut[u] = 1;

}

// dfn表示dfs序，low表示可到达点的最小dfs序，cut表示是否为割点

//使用：

vector<vector<int>> g(n + 10);

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

int u, v;

cin>>u>>v;

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

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

}

int t = 1;

vector<int> dfn(n + 10), low(n + 10), cut(n + 10), a;

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

if (!dfn[i]){

tarjan(i, 0, g, low, dfn, cut, t);

}

}

45.构建圆方树（g1为新建树）

void tarjan(int u, int fa, vector<vector<int>>& g, vector<vector<int>>& g1, vector<int>& low, vector<int>& dfn, vector<int>& cut, stack<int>& st, int& t, int& d){

low[u] = dfn[u] = t++;

int child = 0;

st.push(u);

for (auto v : g[u]){

if (v == fa)

continue;

if (dfn[v]){

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

continue;

}

child++;

tarjan(v, u, g, g1, low, dfn, cut, st, t, d);

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

if (low[v] >= dfn[u] && fa){

cut[u] = 1;

}

if (low[v] >= dfn[u]){

d++;

while (1){

g1[d].push\_back(st.top());

g1[st.top()].push\_back(d);

if (st.top() == v){

st.pop();

break;

}

st.pop();

}

g1[d].push\_back(u);

g1[u].push\_back(d);

}

}

if (child > 1 && fa == 0)

cut[u] = 1;

}

46.求强连通分量（有向图）

void tarjan(int u, int fa, vector<vector<int>>& g, vector<int>& low, vector<int>& dfn, vector<int>& co, stack<int>& st, int& t, int& d){

low[u] = dfn[u] = t++;

st.push(u);

for (auto v : g[u]){

if (!dfn[v]){

tarjan(v, u, g, low, dfn, co, st, t, d);

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

}

else if (!co[v])

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

}

if (low[u] == dfn[u]){

while (st.top() != u){

co[st.top()] = d;

st.pop();

}

co[st.top()] = d;

st.pop();

d++;

}

}

//使用,co[i]存储i结点属于第co[i]这个强连通分量

vector<int> low(n + 10), dfn(n + 10), co(n + 10);

stack<int> st;

int t = 1, d = 1;

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

if (!dfn[i])

tarjan(i, 0, g, low, dfn, co, st, t, d);

}

47.缩点（强连通分量缩点）

void tarjan(int u, int fa, vector<vector<int>>& g, vector<int>& low, vector<int>& dfn, vector<int>& co, stack<int>& st, int& t, int& d){

low[u] = dfn[u] = t++;

st.push(u);

for (auto v : g[u]){

if (!dfn[v]){

tarjan(v, u, g, low, dfn, co, st, t, d);

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

}

else if (!co[v]){

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

}

}

if (low[u] == dfn[u]){

while (st.top() != u){

co[st.top()] = d;

st.pop();

}

co[st.top()] = d;

st.pop();

d++;

}

}

//使用

int n, m;

cin>>n>>m;

vector<pair<int,int>> a(m + 10);//存m条边

vector<vector<int>> g(n + 10), g1(n + 10);

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

int u, v;

cin>>u>>v;

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

a[i].first = u;

a[i].second = v;

}

vector<int> low(n + 10), dfn(n + 10), co(n + 10);

int t = 1, d = 1;

stack<int> st;

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

if (!dfn[i])

tarjan(i, 0, g, low, dfn, co, st, t, d);

vector<int> indu(d + 10);

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

int u = a[i].first, v = a[i].second;

u = co[u], v = co[v];

if (u == v)

continue;

g1[u].push\_back(v);//建立新图

indu[v]++;

}

48.缩点（边双连通分量缩点）

void tarjan(int u, int fa, vector<vector<int>>& g, vector<int>& low, vector<int>& dfn, vector<int>& co, stack<int>& st, int& t, int& d){

low[u] = dfn[u] = t++;

st.push(u);

for (auto v : g[u]){

if (v == fa)

continue;

if (!dfn[v]){

tarjan(v, u, g, low, dfn, co, st, t, d);

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

}

else if (!co[v])

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

}

if (low[u] == dfn[u]){

while (st.top() != u){

co[st.top()] = d;

st.pop();

}

co[st.top()] = d;

st.pop();

d++;

}

}

//使用

vector<pair<int,int>> edge(m + 1);

vector<vector<int>> g(n + 10);

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

int u, v;

cin>>u>>v;

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

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

edge[i].first = u;

edge[i].second = v;

}

vector<int> low(n + 10), dfn(n + 10), co(n + 10);

stack<int> st;

int t = 1, d = 1;

tarjan(1, 0, g, low, dfn, co, st, t, d);//不保证连通时要遍历

vector<vector<pair<int,int>>> g1(d + 10);

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

int u = edge[i].first, v = edge[i].second;

if (co[u] == co[v])

continue;

u = co[u];

v = co[v];

g1[u].push\_back({v, i});

g1[v].push\_back({u, i});

}

49.2-sat

#include <bits/stdc++.h>

using namespace std;

#define ll long long

#define ull unsigned long long

//#define endl '\n'

#define IOS ios::sync\_with\_stdio(false);cin.tie(0);cout.tie(0);

void tarjan(int u, int fa, vector<vector<int>>& g, vector<int>& low, vector<int>& dfn, vector<int>& co, stack<int>& st, int& t, int& d){

low[u] = dfn[u] = t++;

st.push(u);

for (auto v : g[u]){

if (!dfn[v]){

tarjan(v, u, g, low, dfn, co, st, t, d);

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

}

else if (!co[v])

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

}

if (low[u] == dfn[u]){

while (st.top() != u){

co[st.top()] = d;

st.pop();

}

co[st.top()] = d;

st.pop();

d++;

}

}

int main(){

IOS;

int n, m;

cin>>n>>m;

vector<vector<int>> g(2 \* n + 10);

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

int x, fx, y, fy;

cin>>x>>fx>>y>>fy;//矛盾关系

int u = 2 \* x - fx;

int v = 2 \* y - (fy == 0);

g[u].push\_back(v);//建立边

u = 2 \* y - fy;

v = 2 \* x - (fx == 0);

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

}

vector<int> low(2 \* n + 10), dfn(2 \* n + 10), co(2 \* n + 10);

stack<int> st;

int t = 1, d = 1;

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

if (!dfn[i])

tarjan(i, 0, g, low, dfn, co, st, t, d);

int flag = 0;

for (int i = 1; i <= 2 \* n; i += 2){

if (co[i] == co[i + 1])//判断可行否

flag = 1;

}

if (flag){

cout << "IMPOSSIBLE" << endl;

return 0;

}

cout << "POSSIBLE" << endl;

vector<int> ans(n + 10);

vector<int> vis(d + 10);

for (int i = 1; i <= 2 \* n; i += 2){//输出方案

int fi = co[i], fj = co[i + 1];

//下面注释部分是多余的

/\*if (vis[fi]){

ans[i / 2 + 1] = 0;

continue;

}

if (vis[fj]){

ans[i / 2 + 1] = 1;

continue;

}\*/

if (fi < fj){

ans[i / 2 + 1] = 0;

vis[fi] = 1;

}

else{

ans[i / 2 + 1] = 1;

vis[fj] = 1;

}

}

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

cout << ans[i] << " \n"[i == n];

return 0;

}

链接：[2-SAT - OI Wiki (oi-wiki.org)](https://oi-wiki.org/graph/2-sat/)

50.KM算法（求二分图完备匹配下的最大权匹配，如果不存在完备匹配，那么结果是最大匹配下的最大权匹配，cntx <= cnty，Map[i][j]为xi匹配yj的权值）

#define INF 1e17

bool dfs(int u, vector<vector<ll>>& Map, vector<ll>& wx, vector<ll>& wy, vector<int>& cx, vector<int>& cy, vector<int>& visx, vector<int>& visy, int& cntx, int& cnty, ll& minz){

visx[u] = 1;

for (int v = 1; v <= cnty; v++)

if (!visy[v] && Map[u][v] != INF){

ll t = wx[u] + wy[v] - Map[u][v];

if (t == 0){

visy[v] = 1;

if (cy[v] == -1 || dfs(cy[v], Map, wx, wy, cx, cy, visx, visy, cntx, cnty, minz)){

cx[u] = v;

cy[v] = u;

return true;

}

}

else if (t > 0)

minz = min(minz, t);

}

return false;

}

ll KM(vector<vector<ll>>& Map, int cntx, int cnty){

vector<int> cx(cntx + 10, -1), cy(cnty + 10, -1);

vector<ll> wx(cntx + 10), wy(cnty + 10);

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

for (int j = 1; j <= cnty; j++){

if (Map[i][j] == INF)

continue;

wx[i] = max(wx[i], Map[i][j]);

}

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

while(1){

ll minz = INF;

vector<int> visx(cntx + 10), visy(cnty + 10);

if (dfs(i, Map, wx, wy, cx, cy, visx, visy, cntx, cnty, minz))

break;

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

if (visx[j])

wx[j] -= minz;

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

if (visy[j])

wy[j] += minz;

}

ll ans = 0;

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

if (cx[i] != -1)

ans += Map[i][cx[i]];

return ans;

}

链接：[KM算法原理+证明\_梦之泪殇的博客-CSDN博客\_km算法](https://blog.csdn.net/qq_25379821/article/details/83750678)

51.匈牙利算法（求不带权的二分图最大匹配）

bool dfs(int u, vector<vector<int>>& Map, vector<int>& cx, vector<int>& cy, vector<int>& visy, int& cntx, int& cnty){

for (int v = 1; v <= cnty; v++)

if (Map[u][v] && !visy[v]){

visy[v] = 1;

if (cy[v] == -1 || dfs(cy[v], Map, cx, cy, visy, cntx, cnty)){

cx[u] = v;

cy[v] = u;

return 1;

}

}

return 0;

}

int maxmatch(vector<vector<int>>& Map, int& cntx, int& cnty){

int ans = 0;

vector<int> cx(cntx + 10, -1), cy(cnty + 10, -1);

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

if (cx[i] == -1){

vector<int> visy(cnty + 10);

ans += dfs(i, Map, cx, cy, visy, cntx, cnty);

}

return ans;

}

52.匈牙利算法优化版

bool dfs(int u, vector<vector<int>>& g, vector<int>& cx, vector<int>& cy, vector<int>& visy, int& cntx, int& cnty, int& now){

for (auto v : g[u])

if (visy[v] != now){

visy[v] = now;

if (cy[v] == -1 || dfs(cy[v], g, cx, cy, visy, cntx, cnty, now)){

cx[u] = v;

cy[v] = u;

return 1;

}

}

return 0;

}

//返回匹配对数

int maxmatch(vector<vector<int>>& g, int& cntx, int& cnty){

int ans = 0;

vector<int> cx(cntx + 10, -1), cy(cnty + 10, -1);

vector<int> visy(cnty + 10);

int now = 0;

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

if (cx[i] == -1){

now++;

ans += dfs(i, g, cx, cy, visy, cntx, cnty, now);

}

return ans;

}

53.二分图最小点覆盖

第一步先做最大匹配（匈牙利算法），之后我们每次从左边不在匹配边中的一个点开始去按照：未匹配边->匹配边->未匹配边->匹配边……匹配边与未匹配边交替选择的顺序，标记途中经过的顶点，则最后一条经过的边必定为匹配边。这里我开了两个数组covx和covy分别记录每个左侧点是否被标记和每个右侧点是否被标记，但是务必牢记最后最小点覆盖的点集是左侧未被标记的点和右侧被标记的点组成的集合。

bool dfs(int u, vector<vector<int>>& g, vector<int>& cx, vector<int>& cy, vector<int>& visy, int& cntx, int& cnty, int& now){

for (auto v : g[u]){

if (visy[v] != now){

visy[v] = now;

if (!cy[v] || dfs(cy[v], g, cx, cy, visy, cntx, cnty, now)){

cx[u] = v;

cy[v] = u;

return 1;

}

}

}

return 0;

}

void mark(int u, vector<vector<int>>& g, vector<int>& cx, vector<int>& cy, vector<int>& covx, vector<int>& covy){

if (covx[u])

return;

covx[u] = 1;

for (auto v : g[u]){

if (cy[v] && !covy[v]){

covy[v] = 1;

mark(cy[v], g, cx, cy, covx, covy);

}

}

}

int maxmatch(vector<vector<int>>& g, int cntx, int cnty){

vector<int> cx(cntx + 10), cy(cnty + 10), visy(cnty + 10);

int ans = 0;

int now = 0;

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

if (!cx[i]){

now++;

ans += dfs(i, g, cx, cy, visy, cntx, cnty, now);

}

}

vector<int> covx(cntx + 10), covy(cnty + 10);

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

if (!cx[i])

mark(i, g, cx, cy, covx, covy);

cout << ans << endl;

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

if (!covx[i])

cout << i << ' ';

cout << endl;

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

if (covy[i])

cout << i << ' ';

cout << endl;

return ans;

}

链接：[算法：二分图最小点覆盖——Konig定理 - 000紫外线000 - 博客园 (cnblogs.com)](https://www.cnblogs.com/000zwx000/p/12511066.html)

54.Hopcroft-Karp算法（求二分图最大匹配，时间复杂度为n^0.5\*m）得自（koderkushy）

edge中存二分图有效边

struct HopcroftKarp {

const int N, M;

std::vector<std::vector<int>> adj\_left;

std::vector<int> matchL, matchR;

HopcroftKarp (int N, int M, const std::vector<std::pair<int, int>>& edge)

: N(N), M(M), matchL(N, -1), matchR(M, -1), adj\_left(N) {

for (auto [l, r]: edge)

adj\_left[l].push\_back(r);

}

int maxmatching () {

int sz = 0;

for (bool updated = true; updated; ) {

updated = false;

static std::vector<int> root(N), prev(N), qq(N);

static int qi, qj;

qi = qj = 0;

std::fill(root.begin(), root.end(), -1),

std::fill(prev.begin(), prev.end(), -1);

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

if (matchL[i] == -1)

qq[qj++] = i, root[i] = i, prev[i] = i;

while (qi < qj) {

int u = qq[qi++];

if (matchL[root[u]] != -1) continue;

for (int v: adj\_left[u]) {

if (matchR[v] == -1) {

while (v != -1)

matchR[v] = u, std::swap(matchL[u], v), u = prev[u];

updated = true, sz++;

break;

}

if (prev[matchR[v]] == -1)

v = matchR[v], prev[v] = u, root[v] = root[u], qq[qj++] = v;

}

}

}

return sz;

}

};

55. Hopcroft-Karp算法

dx[i]表示左集合i顶点的距离编号，dy[i]表示右集合i顶点的距离编号\*\*/

mx[i]表示左集合顶点所匹配的右集合顶点序号，my[i]表示右集合i顶点匹配到的左集合顶点序号。

#define CLR(a, b) memset(a, b, sizeof(a));

#define INF 1e9

const int Mm = 2e5 + 10;

const int Mn = 1e5 + 10;

struct edge {

int v,next;

}e[Mm];

int tot,head[Mn];

void addedge(int u,int v) {

e[tot].v=v;

e[tot].next=head[u];

head[u]=tot++;

}

int mx[Mn],my[Mn],vis[Mn];

int dis;

int dx[Mn],dy[Mn];

int n,m;

bool searchp() {

queue<int>q;

dis=INF;

CLR(dx,-1);

CLR(dy,-1);

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

if(mx[i]==-1) {

q.push(i);

dx[i]=0;

}

}

while(!q.empty()) {

int u=q.front();

q.pop();

if(dx[u]>dis) break;

for(int i=head[u];~i;i=e[i].next) {

int v=e[i].v;

if(dy[v]==-1) {

dy[v]=dx[u]+1;

if(my[v]==-1) dis=dy[v];

else {

dx[my[v]]=dy[v]+1;

q.push(my[v]);

}

}

}

}

return dis!=INF;

}

bool dfs(int u) {

for(int i=head[u];~i;i=e[i].next) {

int v=e[i].v;

if(vis[v]||(dy[v]!=dx[u]+1)) continue;

vis[v]=1;

if(my[v]!=-1&&dy[v]==dis) continue;

if(my[v]==-1||dfs(my[v])) {

my[v]=u;

mx[u]=v;

return true;

}

}

return false;

}

int maxMatch() {

int res = 0;

CLR(mx,-1);

CLR(my,-1);

while(searchp()) {

CLR(vis,0);

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

if(mx[i] == -1 && dfs(i))

res++;

}

return res;

}

void init() {

tot=0;

CLR(head,-1);

}

int main(){

IOS;

init();

int s;

cin>>n>>m>>s;

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

int u, v;

cin>>u>>v;

u++;

v++;

addedge(u, v);

}

//cout << "sf" << endl;

int ans = maxMatch();

cout << ans << endl;

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

if (mx[i] != -1)

cout << i - 1 << ' ' << mx[i] - 1 << endl;

return 0;

}

链接：[【二分图】匈牙利算法，Hopcroft-Karp算法 - som\_nico - 博客园 (cnblogs.com)](https://www.cnblogs.com/Kohinur/p/9010751.html)

56.dsu on tree（树上启发式合并）

const int N = 1e5;

vector<int> col(N + 10);

vector<vector<int>> g(N + 10);//图

vector<int> siz(N + 10);//子树大小

vector<int> dfn(N + 10), fdfn(N + 10), zdfn(N + 10);//dfs序，反dfs序，子树最大dfs序

vector<int> bigson(N + 10);//重儿子

vector<int> cnt(N + 10);//子树统计量

vector<ll> ans(N + 10);//答案

ll mx = 0, sum = 0;//答案过程统计量

void dfs0(int u, int fa, int& t){

fdfn[t] = u;

dfn[u] = t++;

siz[u] = 1;

int mx = 0;

for (auto v : g[u]){

if (v == fa)

continue;

dfs0(v, u, t);

siz[u] += siz[v];

if (mx < siz[v]){

bigson[u] = v;

mx = siz[v];

}

}

zdfn[u] = t - 1;

}

void dfs1(int u, int fa, bool keep){

for (auto v : g[u]){

if (v == fa || v == bigson[u])

continue;

dfs1(v, u, false);

}

if (bigson[u])

dfs1(bigson[u], u, true);

for (auto v : g[u]){

if (v == fa || v == bigson[u])

continue;

for (int j = dfn[v]; j <= zdfn[v]; j++){

int p = fdfn[j];

cnt[col[p]]++;

if (cnt[col[p]] > mx){

mx = cnt[col[p]];

sum = col[p];

}

else if (mx == cnt[col[p]])

sum += col[p];

}

}

cnt[col[u]]++;

if (cnt[col[u]] > mx){

mx = cnt[col[u]];

sum = col[u];

}

else if (mx == cnt[col[u]])

sum += col[u];

ans[u] = sum;

if (keep)

return;

for (int j = dfn[u]; j <= zdfn[u]; j++){

int p = fdfn[j];

cnt[col[p]]--;

}

mx = 0;

sum = 0;

}

链接：[树上启发式合并 - OI Wiki (oi-wiki.org)](https://oi-wiki.org/graph/dsu-on-tree/)

#include <bits/stdc++.h>

using namespace std;

const int N = 2e5 + 5;

int n;

// g[u]: 存储与 u 相邻的结点

vector<int> g[N];

// sz: 子树大小

// big: 重儿子

// col: 结点颜色

// L[u]: 结点 u 的 DFS 序

// R[u]: 结点 u 子树中结点的 DFS 序的最大值

// Node[i]: DFS 序为 i 的结点

// ans: 存答案

// cnt[i]: 颜色为 i 的结点个数

// totColor: 目前出现过的颜色个数

int sz[N], big[N], col[N], L[N], R[N], Node[N], totdfn;

int ans[N], cnt[N], totColor;

void add(int u) {

if (cnt[col[u]] == 0) ++totColor;

cnt[col[u]]++;

}

void del(int u) {

cnt[col[u]]--;

if (cnt[col[u]] == 0) --totColor;

}

int getAns() { return totColor; }

void dfs0(int u, int p) {

L[u] = ++totdfn;

Node[totdfn] = u;

sz[u] = 1;

for (int v : g[u])

if (v != p) {

dfs0(v, u);

sz[u] += sz[v];

if (!big[u] || sz[big[u]] < sz[v]) big[u] = v;

}

R[u] = totdfn;

}

void dfs1(int u, int p, bool keep) {

// 计算轻儿子的答案

for (int v : g[u])

if (v != p && v != big[u]) {

dfs1(v, u, false);

}

// 计算重儿子答案并保留计算过程中的数据（用于继承）

if (big[u]) {

dfs1(big[u], u, true);

}

for (int v : g[u])

if (v != p && v != big[u]) {

// 子树结点的 DFS 序构成一段连续区间，可以直接遍历

for (int i = L[v]; i <= R[v]; i++) {

add(Node[i]);

}

}

add(u);

ans[u] = getAns();

if (keep == false) {

for (int i = L[u]; i <= R[u]; i++) {

del(Node[i]);

}

}

}

int main() {

scanf("%d", &n);

for (int i = 1; i <= n; i++) scanf("%d", &col[i]);

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

int u, v;

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

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

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

}

dfs0(1, 0);

dfs1(1, 0, false);

for (int i = 1; i <= n; i++) printf("%d%c", ans[i], " \n"[i == n]);

return 0;

}

57.素数筛

const int maxn = 1e2;

int prime[maxn + 10];

void getprime(){

for (int i = 2; i <= maxn; i++){

if (!prime[i])

prime[++prime[0]] = i;

for (int j = 1; j <= prime[0] && prime[j] <= maxn / i; j++){

prime[i \* prime[j]] = 1;

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

break;

}

}

}

58.最小树形图（使得根节点可以到其他所有结点的最小边权和，有向图）

const ll INF = 1e17;

const int maxn = 2e5 + 1;

struct UnionFind {

int fa[maxn << 1];

UnionFind() { memset(fa, 0, sizeof(fa)); }

void clear(int n) { memset(fa + 1, 0, sizeof(int) \* n); }

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

int operator[](int x) { return find(x); }

};

struct Edge{

int u, v;

ll w, w0;

};

struct Heap{

Edge\* e;

int rk, constant;

Heap\* lch, \*rch;

Heap(Edge\* e):e(e), rk(1), constant(0), lch(NULL), rch(NULL){}

void push(){

if (lch) lch->constant += constant;

if (rch) rch->constant += constant;

e->w += constant;

constant = 0;

}

};

Heap\* merge(Heap\* x, Heap\* y){

if (!x)

return y;

if (!y)

return x;

if (x->e->w + x->constant > y->e->w + y->constant)

swap(x, y);

x->push();

x->rch = merge(x->rch, y);

if (!x->lch || x->lch->rk < x->rch->rk)

swap(x->lch, x->rch);

if (x->rch)

x->rk = x->rch->rk + 1;

else

x->rk = 1;

return x;

}

Edge\* extract(Heap\* &x){

Edge\* r = x->e;

x->push();

x = merge(x->lch, x->rch);

return r;

}

vector<Edge> in[maxn];

int n, m, fa[maxn << 1], nxt[maxn << 1];

Edge\* ed[maxn << 1];

Heap\* Q[maxn << 1];

UnionFind id;

void contract(){

bool mark[maxn << 1];

memset(mark, 0, sizeof(mark));

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

queue<Heap\*> q;

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

q.push(new Heap(&in[i][j]));

while (q.size() > 1){

Heap \*u = q.front();

q.pop();

Heap \*v = q.front();

q.pop();

q.push(merge(u, v));

}

Q[i] = q.front();

}

mark[1] = true;

for (int a = 1, b = 1, p; Q[a]; b = a, mark[b] = true){

do{

ed[a] = extract(Q[a]);

a = id[ed[a]->u];

}while (a == b && Q[a]);

if (a == b)

break;

if (!mark[a])

continue;

for (a = b, n++; a != n; a = p){

id.fa[a] = fa[a] = n;

if (Q[a])

Q[a]->constant -= ed[a]->w;

Q[n] = merge(Q[n], Q[a]);

p = id[ed[a]->u];

nxt[p == n ? b : p] = a;

}

}

}

ll expand(int x, int r);

ll expand\_iter(int x){

ll r = 0;

for (int u = nxt[x]; u != x; u = nxt[u]){

if (ed[u]->w0 >= INF)

return INF;

else

r += expand(ed[u]->v, u) + ed[u]->w0;

}

return r;

}

ll expand(int x, int t){

ll r = 0;

for (; x != t; x = fa[x]){

r += expand\_iter(x);

if (r >= INF)

return INF;

}

return r;

}

void link(int u, int v, ll w){

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

}

int main(){

IOS;

int rt;

cin>>n>>m>>rt;

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

int u, v, w;

cin>>u>>v>>w;

link(u, v, w);

}

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

link(i > 1 ? i - 1 : n, i, INF);

contract();

ll ans = expand(rt, n);

if (ans >= INF)

cout << -1 << endl;

else

cout << ans << endl;

return 0;

}

59.对拍

g++ std.cpp -o std -g

g++ my.cpp -o my -g

g++ mkd.cpp -o mkd -g

:loop

mkd.exe > 1.in

my.exe < 1.in > 1.out

std.exe < 1.in > 2.out

fc 1.out 2.out

if not errorlevel 1 goto loop

pause

goto loop

60.求最小直径生成树的直径

#define INF 1e15

const int N = 1000;

struct edge{

int u, v, w;

};

ll val[N];

ll d[N][N];

int rk[N][N];

int n, m;

edge a[N \* N];

bool cmp(int a, int b) { return val[a] < val[b]; }

void Floyd() {

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

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

for (int j = 1; j <= n; j++) d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

}

void solve() {

Floyd();

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

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

rk[i][j] = j;

val[j] = d[i][j];

}

sort(rk[i] + 1, rk[i] + 1 + n, cmp);

}

ll ans = INF;

// 图的绝对中心可能在结点上

for (int i = 1; i <= n; i++) ans = min(ans, d[i][rk[i][n]] \* 2);

// 图的绝对中心可能在边上

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

int u = a[i].u, v = a[i].v, w = a[i].w;

for (int p = n, i = n - 1; i >= 1; i--) {

if (d[v][rk[u][i]] > d[v][rk[u][p]]) {

ans = min(ans, d[u][rk[u][i]] + d[v][rk[u][p]] + w);

p = i;

}

}

}

double t = ans;

cout << fixed << setprecision(9) << ans / 2.0 << endl;

}

int main(){

IOS;

cin>>n>>m;

memset(d, 0x3f, sizeof(d));

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

int u, v, w;

cin>>u>>v>>w;

d[u][v] = w;

d[v][u] = w;

a[i].u = u;

a[i].v = v;

a[i].w = w;

}

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

d[i][i] = 0;

solve();

return 0;

}

61.求直径最小生成树

#include <bits/stdc++.h>

using namespace std;

const int MAXN = 502;

typedef long long ll;

typedef pair<int, int> pii;

ll d[MAXN][MAXN], dd[MAXN][MAXN], rk[MAXN][MAXN], val[MAXN];

const ll INF = 1e17;

int n, m;

bool cmp(int a, int b) { return val[a] < val[b]; }

void floyd() {

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

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

for (int j = 1; j <= n; j++) d[i][j] = min(d[i][j], d[i][k] + d[k][j]);

}

struct node {

ll u, v, w;

} a[MAXN \* (MAXN - 1) / 2];

void solve() {

// 求图的绝对中心

floyd();

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

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

rk[i][j] = j;

val[j] = d[i][j];

}

sort(rk[i] + 1, rk[i] + 1 + n, cmp);

}

ll P = 0, ansP = INF;

// 在点上

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

if (d[i][rk[i][n]] \* 2 < ansP) {

ansP = d[i][rk[i][n]] \* 2;

P = i;

}

}

// 在边上

int f1 = 0, f2 = 0;

ll disu = INT\_MIN, disv = INT\_MIN, ansL = INF;

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

ll u = a[i].u, v = a[i].v, w = a[i].w;

for (int p = n, i = n - 1; i >= 1; i--) {

if (d[v][rk[u][i]] > d[v][rk[u][p]]) {

if (d[u][rk[u][i]] + d[v][rk[u][p]] + w < ansL) {

ansL = d[u][rk[u][i]] + d[v][rk[u][p]] + w;

f1 = u, f2 = v;

disu = (d[u][rk[u][i]] + d[v][rk[u][p]] + w) / 2 - d[u][rk[u][i]];

disv = w - disu;

}

p = i;

}

}

}

cout << min(ansP, ansL) / 2 << '\n';

// 最小路径生成树

vector<pii> pp;

for (int i = 1; i <= 501; ++i)

for (int j = 1; j <= 501; ++j) dd[i][j] = INF;

for (int i = 1; i <= 501; ++i) dd[i][i] = 0;

if (ansP <= ansL) {

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

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

ll u = a[i].u, v = a[i].v, w = a[i].w;

if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {

dd[P][v] = dd[P][u] + w;

pp.push\_back({u, v});

}

u = a[i].v, v = a[i].u, w = a[i].w;

if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {

dd[P][v] = dd[P][u] + w;

pp.push\_back({u, v});

}

}

}

for (auto [x, y] : pp) cout << x << ' ' << y << '\n';

} else {

d[n + 1][f1] = disu;

d[f1][n + 1] = disu;

d[n + 1][f2] = disv;

d[f2][n + 1] = disv;

a[m + 1].u = n + 1, a[m + 1].v = f1, a[m + 1].w = disu;

a[m + 2].u = n + 1, a[m + 2].v = f2, a[m + 2].w = disv;

n += 1;

m += 2;

floyd();

P = n;

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

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

ll u = a[i].u, v = a[i].v, w = a[i].w;

if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {

dd[P][v] = dd[P][u] + w;

pp.push\_back({u, v});

}

u = a[i].v, v = a[i].u, w = a[i].w;

if (dd[P][u] + w == d[P][v] && dd[P][u] + w < dd[P][v]) {

dd[P][v] = dd[P][u] + w;

pp.push\_back({u, v});

}

}

}

cout << f1 << ' ' << f2 << '\n';

for (auto [x, y] : pp)

if (x != n && y != n) cout << x << ' ' << y << '\n';

}

}

void init() {

for (int i = 1; i <= 501; ++i)

for (int j = 1; j <= 501; ++j) d[i][j] = INF;

for (int i = 1; i <= 501; ++i) d[i][i] = 0;

}

int main() {

init();

cin >> n >> m;

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

ll u, v, w;

cin >> u >> v >> w;

w \*= 2;

d[u][v] = w, d[v][u] = w;

a[i].u = u, a[i].v = v, a[i].w = w;

}

solve();

return 0;

}

62.欧拉降幂（求a的b次幂在模mod意义下的值，当b大到用字符串来存储时使用）

ll ksm(ll a, ll b, ll mod){

ll res = 1;

while (b){

if (b & 1){

res \*= a;

res %= mod;

}

a \*= a;

a %= mod;

b /= 2;

}

return res;

}

ll eular(ll x){//欧拉函数

ll ans = x;

for (ll i = 2; i <= x / i; i++){

if (x % i == 0){

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

while (x % i == 0)

x /= i;

}

}

if (x > 1)

ans = ans / x \* (x - 1);

return ans;

}

ll eularksm(ll a, string& b, ll mod){

a %= mod;

ll eul = eular(mod);

ll m = 0;

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

m = m \* 10 + b[i] - '0';

m %= eul;

}

m += eul;

ll ans = ksm(a, m, mod);

return ans;

}

//使用

ll a;

ll mod;

string b;

cin>>a;

cin>>b;

cin>>mod;

ll ans = eularksm(a, b, mod);

cout << ans << endl;

链接：[欧拉降幂 - 逃往火星的猫 - 博客园 (cnblogs.com)](https://www.cnblogs.com/l1l1/p/9658131.html)

63.约瑟夫问题（n 个人标号从0到（n - 1），逆时针站成一圈，从0号开始，每一次从当前的人逆时针数k个，然后让这个人出局。问最后剩下的人是谁。）

ll josephus(int n, ll k){

ll res = 0;

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

res = (res + k) % i;

return res;

}

64.kruskal重构树

struct edge{

int x, y w;

};

int find(int x, vector<int>& fa){

if (fa[x] == x)

return fa[x];

return fa[x] = find(fa[x], fa);

}

void kruskal(int n, int m, vector<edge>& ed, vector<vector<int>>& g, vector<int>& val){

vector<int> fa(n + 10);

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

fa[i] = i;

sort(ed.begin() + 1, ed.begin() + m + 1);

int now = n + 1;

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

auto [x, y, w] = ed[i];

int fx = find(x, fa), fy = find(y, fa);

if (fx != fy){

val[now++] = w;

fa[fx] = fa[fy] = fa[now] = now;

g[now].push\_back(fx);

g[now].push\_back(fy);

}

}

}

65.最小环

int val[maxn + 1][maxn + 1]; // 原图的邻接矩阵

inline int floyd(const int &n) {

static int dis[maxn + 1][maxn + 1]; // 最短路矩阵

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

for (int j = 1; j <= n; ++j) dis[i][j] = val[i][j]; // 初始化最短路矩阵

int ans = inf;

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

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

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

ans = std::min(ans, dis[i][j] + val[i][k] + val[k][j]); // 更新答案

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

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

dis[i][j] = std::min(

dis[i][j], dis[i][k] + dis[k][j]); // 正常的 floyd 更新最短路矩阵

}

return ans;

}

66.一般图最大匹配（带花树，求最大匹配数，并输出匹配边）

struct node {

int to, nxt;

} g[M];

int head[N], cnt;

int vis[N], match[N], f[N], pre[N], Id, id[N];

// vis[i]: 0(未染色) 1(黑色) 2(白色)

// match[i]: i的匹配点

// f[i]: i在带花树中的祖先

// pre[i]: i的非匹配边的另一点

// id: 找LCA用

// g数组: 存关系

int n, m, ans, u, v;

queue<int> q;

void init() {

Id = ans = cnt = 0;

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

head[i] = -1, id[i] = match[i] = 0;

}

}

void add(int u, int v) { g[cnt].to = v, g[cnt].nxt = head[u], head[u] = cnt++; }

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

int lca(int x, int y) {

for (++Id;; swap(x, y)) {

if (x) {

x = find(x);

if (id[x] == Id)

return x;

else

id[x] = Id, x = pre[match[x]];

}

}

}

void blossom(int x, int y, int l) { // l为x，y的最近公共祖先（同一个花）

while (find(x) != l) {

pre[x] = y, y = match[x];

if (vis[y] == 2) vis[y] = 1, q.push(y);

if (find(x) == x) f[x] = l;

if (find(y) == y) f[y] = l;

x = pre[y];

}

}

bool aug(int s) {

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

vis[i] = pre[i] = 0;

f[i] = i;

}

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

q.push(s), vis[s] = 1;

while (!q.empty()) {

u = q.front();

q.pop();

for (int i = head[u]; ~i; i = g[i].nxt) {

v = g[i].to;

if (find(u) == find(v) || vis[v] == 2) continue;

if (!vis[v]) {

vis[v] = 2, pre[v] = u;

if (!match[v]) {

for (int x = v, last; x; x = last)

last = match[pre[x]], match[x] = pre[x], match[pre[x]] = x;

return true;

}

vis[match[v]] = 1, q.push(match[v]);

} else {

int LCA = lca(u, v);

blossom(u, v, LCA), blossom(v, u, LCA);

}

}

}

return false;

}

int main() {

cin >> n >> m;

init();

while (m--) {

cin >> u >> v;

add(u, v);

add(v, u);

}

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

if (!match[i] && aug(i)) ans++;

}

cout << ans << endl;

for (int i = 1; i <= n; i++) cout << match[i] << ' ';

return 0;

}

67.快读，快写（scan，print版本的快读快写只需要改变函数的参数类型就可以实现\_\_int128的读入输出，威子版本可读任意整数类型，无需调整）

void scan(int& x){

x = 0;

int f = 1;

char ch;

while (((ch = getchar()) > '9' || ch < '0') && ch != '-');

if (ch == '-')

f = -f;

else

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

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

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

x \*= f;

}

void print(int x){

if (x < 0){

x = -x;

putchar('-');

}

if (x > 9)

print(x / 10);

putchar(x % 10 + '0');

}

威子版本

template<typename T> inline void read(T& num) {

num = 0;

char digit = nc();

bool negtive = false;

for (; !isdigit(digit); digit = nc())

negtive ^= !(digit ^ 45);

for (; isdigit(digit); digit = nc())

num = num \* 10 + digit - '0';

if (negtive)

num = -num;

}

template<typename T, typename... Ts>

inline void read(T& t, Ts&... ts) {

read(t); read(ts...);

}

} // namespace IO

68.待定