d788: 排名順序

內容

考試成績出爐了 , 大家開始討論自己的分數高低一個接著一個參與討論 , 新加入的那個人 , 想要知道自己目前排名是多少但是太多人了 , 導致沒辦法一時得到他的排名大家開始請求小光這個答案 ,不過小光非常討厭排名 , 一點都不想幫忙現在就交給你了

輸入說明

每組輸入的第一行有一個數字 N ( 1 ≦ N ≦ 10,0000 )代表接下來會有 N 個人陸續與討論，接下來會有 N 行，代表接下來陸續加入的人的成績 M ， ( 1 ≦ M ≦ N )而且每個人的成績都不會重複

輸出說明

對於已經知道的成績，請陸續對每個加入的輸出他的排名

範例輸入

6 1 5 6 3 4 2

範例輸出

1 1 1 3 3 5

#pragma GCC optimize(“O3”)

#include<bits/stdc++.h>

#include<ext/pb\_ds/assoc\_container.hpp>

using namespace std;

using namespace \_\_gnu\_pbds;

int main()

{

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

cin.tie(NULL);

tree<int,null\_type,greater<int>,rb\_tree\_tag,tree\_order\_statistics\_node\_update> s;

int n;

cin>>n;

int a;

while(n--)

{

cin>>a;

s.insert(a);

cout<<s.order\_of\_key(a)+1<<'\n';

}

return 0;

}

最短路徑Single Source Shortest Paths:Dijkstra's Algorithm + Priority Queue

1. // 要丟進Priority Queue的點
2. // b是點，d是可能的最短路徑長度。
3. // a可以提出來，不必放在Node裡。
4. struct Node {int b, d;};
5. bool operator<(Node n1, Node n2) {return n1.d > n2.d;}
6. int w[9][9];    // adjacency matrix
7. int d[9];
8. int parent[9];
9. bool visit[9];
10. void dijkstra\_with\_priority\_queue(int source)
11. {
12. for (int i=0; i<9; i++) visit[i] = false;
13. for (int i=0; i<9; i++) d[i] = 1e9;
14. // C++ STL的Priority Queue
15. priority\_queue<Node> pq;
16. d[source] = 0;
17. parent[source] = source;
18. pq.push((Node){source, d[source]});
19. for (int i=0; i<9; i++)
20. {
21. // 找出下一個要加入到最短路徑樹的點。
22. int a = -1;
23. while (!pq.empty() && visit[a = pq.top().b])
24. pq.pop();   // 最後少pop一次，不過無妨。
25. if (a == -1) break;
26. visit[a] = true;
27. for (int b=0; b<9; b++)
28. if (!visit[b] && d[a] + w[a][b] < d[b])
29. {
30. d[b] = d[a] + w[a][b];
31. parent[b] = a;
32. // 交由Priority Queue比較大小
33. pq.push( (Node){b, d[b]} );
34. }

    }

一點到多點的最短路徑、找出一棵最短路徑樹（adjacency matrix）

1. int w[9][9];    // 一張有權重的圖：adjacency matrix
2. int d[9];       // 記錄起點到圖上各個點的最短路徑長度
3. int parent[9];  // 記錄各個點在最短路徑樹上的父親是誰
4. bool visit[9];  // 記錄各個點是不是已在最短路徑樹之中
5. void label\_setting(int source)
6. {
7. for (int i=0; i<100; i++) visit[i] = false; // initialize
8. d[source] = 0;              // 設定起點的最短路徑長度
9. parent[source] = source;    // 設定起點是樹根（父親為自己）
10. visit[source] = true;       // 將起點加入到最短路徑樹
11. for (int k=0; k<9-1; k++)   // 將剩餘所有點加入到最短路徑樹
12. {
13. // 從既有的最短路徑樹，找出一條聯外而且是最短的邊
14. int a = -1, b = -1, min = 1e9;
15. // 找一個已在最短路徑樹上的點
16. for (int i=0; i<9; i++)
17. if (visit[i])
18. // 找一個不在最短路徑樹上的點
19. for (int j=0; j<9; j++)
20. if (!visit[j])
21. if (d[i] + w[i][j] < min)
22. {
23. a = i;  // 記錄這一條邊
24. b = j;
25. min = d[i] + w[i][j];
26. }
27. // 起點有連通的最短路徑都已找完
28. if (a == -1 || b == -1) break;
29. //      // 不連通即是最短路徑長度無限長
30. //      if (min == 1e9) break;
31. d[b] = min;         // 儲存由起點到b點的最短路徑長度
32. parent[b] = a;      // b點是由a點延伸過去的
33. visit[b] = true;    // 把b點加入到最短路徑樹之中
34. }
35. }

BFS

1. bool adj[9][9]; // 一張圖，資料結構為adjacency matrix。
2. bool visit[9];  // 記錄圖上的點是否遍歷過，有遍歷過為true。
3. void BFS()
4. {
5. // 建立一個queue。
6. queue<int> q;
7. // 全部的點都初始化為尚未遍歷
8. for (int i=0; i<9; i++)
9. visit[i] = false;
10. for (int k=0; k<9; k++)
11. if (!visit[k])
12. {
13. // 一、把起點放入queue。
14. q.push(k);
15. visit[k] = true;
16. // 二、重複下述兩點，直到queue裡面沒有東西為止：
17. while (!q.empty())
18. {
19. // 甲、從queue當中取出一點。
20. int i = q.front(); q.pop();
21. // 乙、找出跟此點相鄰的點，並且尚未遍歷的點，
22. //     依照編號順序通通放入queue。
23. for (int j=0; j<9; j++)
24. if (adj[i][j] && !visit[j])
25. {
26. q.push(j);
27. visit[j] = true;
28. }
29. }
30. }
31. }

**4.4.3 Vector**

vector<int> vi 建立一個int 型別的vector

• vi.clear() 清空vector

• vi.size() 取得元素數量

• vi.reserve(x) 預先將大小設為x，新增的元素以default constructor 初始化

• vi.empty() 判斷是否為空

• vi.push\_back(i) 從後面加入一個元素，值為i

• vi.pop\_back() 從後面刪掉一個元素

• vi[i] 取得第i 個元素(從0 開始計算)

• vi.begin() 取得指向第一個元素的iterator

• vi.end() 取得指向最後一個元素後面一格的iterator

• vi.front() 取得第一個元素的值

• vi.back() 取得最後一個元素的值

**4.4.4 Deque**

• deque<int> dq 建立一個int 型別的deque

• dq.clear() 清空deque

• dq.size() 取得元素數量

• dq.empty() 判斷是否為空

• dq.push\_front(i) 從前面加入一個元素，值為i

• dq.pop\_front() 從前面刪掉一個元素

• dq.push\_back(i) 從後面加入一個元素，值為i

• dq.pop\_back() 從後面刪掉一個元素

• dq[i] 取得第i 個元素(從0 開始計算)

• dq.begin() 取得指向第一個元素的iterator

• dq.end() 取得指向最後一個元素後面一格的iterator

• dq.front() 取得第一個元素的值

• dq.back() 取得最後一個元素的值

**4.4.5 List**

link<int> lk 建立一個int 型別的link

• lk.clear() 清空link

• lk.size() 取得元素數量

• lk.empty() 判斷是否為空

• lk.push\_front(i) 從前面加入一個元素，值為i

• lk.pop\_front() 從前面刪掉一個元素

• lk.push\_back(i) 從後面加入一個元素，值為i

• lk.pop\_back() 從後面刪掉一個元素

• lk.begin() 取得指向第一個元素的iterator

• lk.end() 取得指向最後一個元素後面一格的iterator

• lk.front() 取得第一個元素的值

• lk.back() 取得最後一個元素的值

• lk.insert(it, x) 在it 這個iterator 所指的元素前面，插入一個值為x 的元素

• lk.insert(it, first, last) 在it 所指的元素前面，插入first 到last 中的所有元素

(first、last 為iterator)

• lk.erase(it) 刪除it 所指的元素

• lk.erase(first, last) 刪除first 到last 中的所有元素(first、last 為iterator)

**4.4.6 Stack**

stack<int> st 建立一個int 型別的stack

• stack<int, vector<int> > st 以vector 作為底層容器建立stack

• sk.size() 取得元素數量

• sk.empty() 判斷是否為空

• sk.push(i) 從頂端加入一個元素，值為i

• sk.pop() 刪除頂端元素

• sk.top() 取得頂端元素(即最後一個元素)

**4.4.7 Queue**

• queue<int> q 建立一個int 型別的queue

• queue<int, deque<int> > q 以deque 作為底層容器建立queue

• q.size() 取得元素數量

• q.empty() 判斷是否為空

• q.push(i) 從後端加入一個元素，值為i

• q.pop() 從前端刪除一個元素

• q.front() 取得前端元素(即out 端第一個元素)

• q.back() 取得後端元素(即in 端第一個元素)

**4.4.8 Priority Queue**

• priority\_queue<int> pq 建立一個int 型別的queue

• priority\_queue<int, deque<int>, greater<int> > pq 以deque 作為底層容器建立

queue，以greater 作為比較函式(也就是使頂端為最小元素)

• pq.size() 取得元素數量

• pq.empty() 判斷是否為空

• pq.push(i) 加入一個元素，值為i

• pq.pop() 刪除最大元素

• pq.top() 取得最大元素的值

**4.4.9 Set, Multiset, Map, Multimap**

• set<int> s 建立元素為int 的set

• multiset<int> ms 建立元素為int 的multiset

• map<int, string> m 建立Key 為int，元素值為string 的map

• multimap<int, string> mm 建立Key 為int，元素值為string 的multimap

• size() 取得元素數量

• empty() 判斷是否為空

• clear() 清空

• insert(i) 新增一個值為i 的元素(set、multiset)

• insert(make\_pair(i,x)) 新增一個Key 值為i，元素值為x 的元素(以pair 形式插入)

(map、multimap)

• erase(i) 刪除值為i 的元素

• erase(it) 刪除it 所指的元素

• begin() 取得指向第一個元素的iterator

• end() 取得指向最後一個元素後面一格的iterator

• find(i) 取得Key 值為i 的元素的iterator(若找不到則回傳end())

• count(i) 取得Key 值為i 的元素的數量

• lower\_bound(i) 取得第一個Key 值大於等於i 的元素的iterator

• upper\_bound(i) 取得第一個Key 值大於i 的元素的iterator

• m[i] 取得Key 值為i 的元素的值(map)

• \*it 取得it 所指的元素的值(set、multiset), Key-Value Pair(map、multimap)

• it->first 取得it 所指的元素的Key 值(map、multimap)

• it->second 取得it 所指的元素的元素值(map、multimap)

**4.5 STL Algorithm**

STL 裡面除了一些資料結構以外，還內建有一些常用的演算法以及實用函式，在此稍作

介紹。

需要引入標頭檔<algorithm>。

**4.5.1 sort()**、**stable\_sort()**

sort 即是排序演算法，複雜度為*O*(*N* lg*N*)。分為兩種版本，sort() 及stable\_sort()，

後者會維持同樣大的元素的順序。

sort(first, last);

//將[first,last)間的元素排序(first和last為iterator或pointer)

//注意last本身不算，使用vector的話一般會將first=v.begin(), last=v.end()

//使用array的話將first=arr, end=arr+N (N為陣列大小)

sort(first, last, cmp);

//以自訂比較函式cmp來排序

**4.5.2 reverse()**

將序列的元素倒轉。例如*f*1*;* 2*;* 3*;* 4*g*→*f*4*;* 3*;* 2*;* 1*g*。時間複雜度*O*(*N*)。

注意不要把reverse() 跟vector 的reserve() 搞混了。

reverse(first, last);

//將[first,last)間的元素倒轉

**4.5.3 lower\_bound()**、**upper\_bound()**

lower\_bound() 以二分搜方式取得第一個大於等於(upper\_bound() 只能大於) 指定值的

元素的iterator，而複雜度為*O*(lg*N*)。注意搜尋的區間必須為已排序，否則結果無法預期。

vector<int>::iterator it = lower\_bound(first, last, i);

//在[first,last)間找第一個大於等於i的元素，注意last本身不算

it = lower\_bound(first, last, i, cmp);

//使用自訂cmp比較函式

int a = \*it;

//lower\_bound()回傳的是iterator，需要提領才能得到元素值

**4.5.4 binary\_search()**

也是二分搜，但只查詢指定元素是否存在，回傳型態為bool。同樣地序列須為已排序。

複雜度為*O*(lg*N*)。

bool b = binary\_search(first, last, i);

//在[first,last)區間內查詢是否有等於i的元素

b = binary\_search(first, last, i, cmp);

//使用自訂比較函式cmp

**4.5.5 fill()**

將一個序列的所有值初始化為指定值。效率較for 迴圈為高，且適用範圍比memset()

廣一些(例如適用於class 型別)。

fill(first, last, i);

//將[first,last)間的所有元素設定為i

**4.5.6 mismatch()**

取得兩個序列第一個不相等的元素，回傳值為一個pair，包含分別指向兩個序列該位置

的iterator。也常用於string 字串。時間複雜度*O*(*N*)。

pair<vector<int>::iterator, vector<int>::iterator> = mismatch(first1, last1, first2);

//將序列[first1,last1)與序列[first2,...比對，回傳第一個不同的位置

**4.5.7 next\_permutation()**

取得下一個排列方式， 並將它寫入原本的序列。*N*! 爆搜的時候很常用(?) 例如

*f*1*;* 2*;* 3*g*→*f*1*;* 3*;* 2*g*→*f*2*;* 1*;* 3*g*→…

成功的時候回傳true，失敗則回傳false。時間複雜度*O*(*N*)。

next\_permutation(first, last);

//將[first,last)間的序列換成下一個排序方式。

**4.5.8 random\_shuffle()**

將序列隨機重新排列。例如1*;* 2*;* 3*;* 4*;* 5*;* 6*;* 7→6*;* 1*;* 5*;* 7*;* 4*;* 2*;* 3

random\_shuffle(first, last);

//將[first,last)間的元素隨機排列

演算法(Algorithm)

count

count(first, last, val)

[first]: iterator

[last]: iterator

[val]: target value type

[回傳值]: int

針對某個container做搜尋，區間由first及last這兩個iterator指定，目標值為val，回傳container中元素值為val的個數

search

search(s\_first, s\_last, t\_first, t\_last)

[s\_first][t\_first]: iterator

[s\_last][t\_last]: iterator

[回傳值]: iterator

找尋某一資料序列是否出現在另一個容器中

merge

merge(s1\_first, s1\_last, s2\_first, s2\_last, t\_first)

[s1\_first][s2\_first]: iterator

[s1\_last][s2\_last]: iterator

[t\_first]: iterator

[回傳值]: iterator

合併s1與s2兩資料於t

sort

sort(first, last)

sort(first, last, f)

[first]: iterator

[last]: iterator

[f]: 函式

[回傳值]: void

資料排序(預設由小到大)

31for\_each

for\_each(first, last, f)

[first]: iterator

[last]: iterator

[f]: 函式

[回傳值]: 函數物件指標

對container中從first 所指的元素起到last為止，其間每一個元素做某個動作(由函數f指定)

transform

transform(first, last, output, f)

[first]: iterator

[last]: iterator

[output]: container

[f]: 函式

[回傳值]: 函數物件指標

同for\_each，但會把結果放在output容器中

**2 Flow**

**2.1 Dinic**

(a) Bounded Maxflow Construction:

1. add two node ss, tt

2. add\_edge(ss, tt, INF)

3. **for** each edge u -> v with capacity [l, r]:

add\_edge(u, tt, l)

add\_edge(ss, v, l)

add\_edge(u, v, r-l)

4. see (b), check **if** it is possible.

5. answer is maxflow(ss, tt) + maxflow(s, t)

-------------------------------------------------------

(b) Bounded Possible Flow:

1. same construction method as (a)

2. run maxflow(ss, tt)

3. **for** every edge connected with ss **or** tt:

rule: check **if** their rest flow is exactly 0

4. answer is possible **if** every edge **do** satisfy the rule

;

5. otherwise , it is NOT possible.

-------------------------------------------------------

(c) Bounded Minimum Flow:

1. same construction method as (a)

2. answer is maxflow(ss, tt)

-------------------------------------------------------

(d) Bounded Minimum Cost Flow:

\* the concept is somewhat like bounded possible flow.

1. same construction method as (a)

2. answer is maxflow(ss, tt) + (Σ l \* cost **for** every

edge)

-------------------------------------------------------

(e) Minimum Cut:

1. run maxflow(s, t)

2. run cut(s)

3. ss[i] = 1: node i is at the same side with s.

-------------------------------------------------------

#define N 5010

#define M 60010

#define ll long long

#define inf 1ll<<62

ll to[ M ] , next[ M ] , head[ M ];

ll cnt , ceng[ M ] , que[ M ] , w[ M ];

ll n , m , start , end;

void add( ll a , ll b , ll flow ){

to[ cnt ] = b , next[ cnt ] = head[ a ] , w[ cnt ] = flow , head[ a ] = cnt ++;

to[ cnt ] = a , next[ cnt ] = head[ b ] , w[ cnt ] = flow , head[ b ] = cnt ++;

}

void read(){

memset(head,-1,sizeof head);

//memset(next,-1,sizeof next);

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

ll a , b , flow;

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

scanf( "%lld%lld%lld" , &a , &b , &flow );

add( a , b , flow );

}

end = n ,start = 1;

}

bool bfs(){

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

ll h = 1 , t = 2;

ceng[ start ] = 0;

que[ 1 ] = start;

while( h < t ){

ll sta = que[ h ++ ];

for( ll i = head[ sta ] ; ~i ; i = next[ i ] )

if( w[ i ] > 0 && ceng[ to[ i ] ] < 0 ){

ceng[ to[ i ] ] = ceng[ sta ] + 1;

que[ t ++ ] = to[ i ];

}

}

return ceng[ end ] != -1;

}

ll find( ll x , ll low ){

ll tmp = 0 , result = 0;

if( x == end ) return low;

for( ll i = head[ x ] ; ~i && result < low ; i = next[ i ] )

if( w[ i ] > 0 && ceng[ to[ i ] ] == ceng[ x ] + 1 ){

tmp = find( to[ i ] , min( w[ i ] , low - result ) );

w[ i ] -= tmp;

w[ i^1 ] += tmp;

result += tmp;

}

if( !result ) ceng[ x ] = -1;

return result;

}

ll dinic(){

ll ans = 0 , tmp;

while( bfs() ) ans += find( start , inf );

return ans;

}

int main(){

read();

cout << dinic() << endl;

}

**2.3 min cost flow**

struct MinCostMaxFlow{

typedef int Tcost;

static const int MAXV = 20010;

static const int INFf = 1000000;

static const Tcost INFc = 1e9;

struct Edge{

int v, cap;

Tcost w;

int rev;

Edge(){}

Edge(int t2, int t3, Tcost t4, int t5)

: v(t2), cap(t3), w(t4), rev(t5) {}

};

int V, s, t;

vector<Edge> g[MAXV];

void init(int n){

V = n+2;

s = n+1, t = n+2;

for(int i = 0; i <= V; i++) g[i].clear();

}

void addEdge(int a, int b, int cap, Tcost w){

g[a].push\_back(Edge(b, cap, w, (int)g[b].size()));

g[b].push\_back(Edge(a, 0, -w, (int)g[a].size()-1));

}

Tcost d[MAXV];

int id[MAXV], mom[MAXV];

bool inqu[MAXV];

queue<int> q;

Tcost solve(){

int mxf = 0; Tcost mnc = 0;

while(1){

fill(d, d+1+V, INFc);

fill(inqu, inqu+1+V, 0);

fill(mom, mom+1+V, -1);

mom[s] = s;

d[s] = 0;

q.push(s); inqu[s] = 1;

while(q.size()){

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

inqu[u] = 0;

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

Edge &e = g[u][i];

int v = e.v;

if(e.cap > 0 && d[v] > d[u]+e.w){

d[v] = d[u]+e.w;

mom[v] = u;

id[v] = i;

if(!inqu[v]) q.push(v), inqu[v] = 1;

}

}

}

if(mom[t] == -1) break ;

int df = INFf;

for(int u = t; u != s; u = mom[u])

df = min(df, g[mom[u]][id[u]].cap);

for(int u = t; u != s; u = mom[u]){

Edge &e = g[mom[u]][id[u]];

e.cap -= df;

g[e.v][e.rev].cap += df;

}

mxf += df;

mnc += df\*d[t];

}

return mnc;

}

} flow;

**2.4 SW mincut 全點對最小割**

// global min cut

struct SW{ // O(V^3)

static const int MXN = 514;

int n,vst[MXN],del[MXN];

int edge[MXN][MXN],wei[MXN];

void init(int \_n){

n = \_n; FZ(edge); FZ(del);

}

void addEdge(int u, int v, int w){

edge[u][v] += w; edge[v][u] += w;

}

void search(int &s, int &t){

FZ(vst); FZ(wei);

s = t = -1;

while (true){

int mx=-1, cur=0;

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

if (!del[i] && !vst[i] && mx<wei[i])

cur = i, mx = wei[i];

if (mx == -1) break;

vst[cur] = 1;

s = t; t = cur;

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

if (!vst[i] && !del[i]) wei[i] += edge[cur][i];

}

}

int solve(){

int res = 2147483647;

for (int i=0,x,y; i<n-1; i++){

search(x,y);

res = min(res,wei[y]);

del[y] = 1;

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

edge[x][j] = (edge[j][x] += edge[y][j]);

}

return res;

}

;

**3 Matching**

**3.1 Hungarian**

// Maximum Cardinality Bipartite Matching

// Worst case O(nm)

struct Graph{

static const int MAXN = 5003;

vector<int> G[MAXN];

int n, match[MAXN], vis[MAXN];

void init(int \_n){

n = \_n;

for (int i=0; i<n; i++) G[i].clear();

}

bool dfs(int u){

for (int v:G[u]){

if (vis[v]) continue;

vis[v]=true;

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

match[v] = u;

match[u] = v;

return true;

}

}

return false;

}

int solve(){

int res = 0;

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

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

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

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

if ( dfs(i) ) res++;

}

}

return res;

}

} graph;

**3.2 KM**

const int MAX\_N = 400 + 10;

const ll INF64 = 0x3f3f3f3f3f3f3f3fLL;

int nl , nr;

int pre[MAX\_N];

ll slack[MAX\_N];

ll W[MAX\_N][MAX\_N];

ll lx[MAX\_N] , ly[MAX\_N];

int mx[MAX\_N] , my[MAX\_N];

bool vx[MAX\_N] , vy[MAX\_N];

void augment(int u) {

if(!u) return;

augment(mx[pre[u]]);

mx[pre[u]] = u;

my[u] = pre[u];

}

inline void match(int x) {

queue<int> que;

que.push(x);

while(1) {

while(!que.empty()) {

x = que.front();

que.pop();

vx[x] = 1;

REP1(y , 1 , nr) {

if(vy[y]) continue;

ll t = lx[x] + ly[y] - W[x][y];

if(t > 0) {

if(slack[y] >= t) slack[y] = t , pre[y] = x;

continue;

}

pre[y] = x;

if(!my[y]) {

augment(y);

return;

}

vy[y] = 1;

que.push(my[y]);

}

}

ll t = INF64;

REP1(y , 1 , nr) if(!vy[y]) t = min(t , slack[y]);

REP1(x , 1 , nl) if(vx[x]) lx[x] -= t;

REP1(y , 1 , nr) {

if(vy[y]) ly[y] += t;

else slack[y] -= t;

}

REP1(y , 1 , nr) {

if(vy[y] || slack[y]) continue;

if(!my[y]) {

augment(y);

return;

}

vy[y] = 1;

que.push(my[y]);

}

}

}

int main() {

int m;

RI(nl , nr , m);

nr = max(nl , nr);

while(m--) {

int x , y;

ll w;

RI(x , y , w);

W[x][y] = w;

lx[x] = max(lx[x] , w);

}

REP1(i , 1 , nl) {

REP1(x , 1 , nl) vx[x] = 0;

REP1(y , 1 , nr) vy[y] = 0 , slack[y] = INF64;

match(i);

}

ll ans = 0LL;

REP1(x , 1 , nl) ans += W[x][mx[x]];

PL(ans);

REP1(x , 1 , nl) printf("%d%c",W[x][mx[x]] ? mx[x] : 0," \n"[x == nl]);

return 0;

}

**4 Graph**

**4.1 BCC edge**

邊雙連通

任意兩點間至少有兩條不重疊的路徑連接， 找法：

1. 標記出所有的橋

2. 對全圖進行DFS ， 不走橋， 每一次DFS 就是一個新的邊雙連通

邊雙連通

任意兩點間至少有兩條不重疊的路徑連接，找法：

1. 標記出所有的橋

2. 對全圖進行 DFS，不走橋，每一次 DFS 就是一個新的邊雙連通

// from BCW

struct BccEdge {

static const int MXN = 100005;

struct Edge { int v,eid; };

int n,m,step,par[MXN],dfn[MXN],low[MXN];

vector<Edge> E[MXN];

DisjointSet djs;

void init(int \_n) {

n = \_n; m = 0;

for (int i=0; i<n; i++) E[i].clear();

djs.init(n);

}

void add\_edge(int u, int v) {

E[u].PB({v, m});

E[v].PB({u, m});

m++;

}

void DFS(int u, int f, int f\_eid) {

par[u] = f;

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

for (auto it:E[u]) {

if (it.eid == f\_eid) continue;

int v = it.v;

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

DFS(v, u, it.eid);

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

} else {

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

}

}

}

void solve() {

step = 0;

memset(dfn, -1, sizeof(int)\*n);

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

if (dfn[i] == -1) DFS(i, i, -1);

}

djs.init(n);

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

if (low[i] < dfn[i]) djs.uni(i, par[i]);

}

}

}graph;

**4.2 Dijkstra**

typedef struct Edge{

int v; long long len;

bool operator > (const Edge &b)const { return len>b.len; }

} State;

const long long INF = 1LL<<60;

void Dijkstra(int n, vector<Edge> G[], long long d[], int s, int t=-1){

static priority\_queue<State, vector<State>, greater<State> > pq;

while ( pq.size() )pq.pop();

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

d[s]=0; pq.push( (State){s,d[s]} );

while ( pq.size() ){

auto x = pq.top(); pq.pop();

int u = x.v;

if (d[u]<x.len)continue;

if (u==t)return;

for (auto &e:G[u]){

if (d[e.v] > d[u]+e.len){

d[e.v] = d[u]+e.len;

pq.push( (State) {e.v,d[e.v]} );

}

}

}

}

**4.4 max clique**

const int MAXN = 105;

int best;

int n;

int num[MAXN];

int path[MAXN];

int G[MAXN][MAXN];

bool dfs( int \*adj, int total, int cnt ){

int t[MAXN];

if (total == 0){

if( best < cnt ){

best = cnt;

return true;

}

return false;

}

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

if( cnt+(total-i) <= best ) return false;

if( cnt+num[adj[i]] <= best ) return false;

int k=0;

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

if(G[ adj[i] ][ adj[j] ])

t[k++] = adj[j];

if (dfs(t, k, cnt+1)) return true;

}

return false;

}

int MaximumClique(){

int adj[MAXN];

if (n <= 0) return 0;

best = 0;

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

int k=0;

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

if (g[i][j]) adj[k++] = j;

dfs( adj, k, 1 );

num[i] = best;

}

return best;

}

**4.5 min mean cycle**

// from BCW

/\* minimum mean cycle \*/

const int MAXE = 1805;

const int MAXN = 35;

const double inf = 1029384756;

const double eps = 1e-6;

struct Edge {

int v,u;

double c;

};

int n,m,prv[MAXN][MAXN], prve[MAXN][MAXN], vst[MAXN];

Edge e[MAXE];

vector<int> edgeID, cycle, rho;

double d[MAXN][MAXN];

inline void bellman\_ford() {

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

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

fill(d[i+1], d[i+1]+n, inf);

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

int v = e[j].v, u = e[j].u;

if(d[i][v]<inf && d[i+1][u]>d[i][v]+e[j].c) {

d[i+1][u] = d[i][v]+e[j].c;

prv[i+1][u] = v;

prve[i+1][u] = j;

}

}

}

}

double karp\_mmc() {

// returns inf if no cycle, mmc otherwise

double mmc=inf;

int st = -1;

bellman\_ford();

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

double avg=-inf;

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

if(d[n][i]<inf-eps) avg=max(avg,(d[n][i]-d[k][i])/(n-k));

else avg=max(avg,inf);

}

if (avg < mmc) tie(mmc, st) = tie(avg, i);

}

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

edgeID.clear(); cycle.clear(); rho.clear();

for (int i=n; !vst[st]; st=prv[i--][st]) {

vst[st]++;

edgeID.PB(prve[i][st]);

rho.PB(st);

}

while (vst[st] != 2) {

int v = rho.back(); rho.pop\_back();

cycle.PB(v);

vst[v]++;

}

reverse(ALL(edgeID));

edgeID.resize(SZ(cycle));

return mmc;

}

**4.6 SSSP related concepts**

最短路問題分類：

三個工具Bellman -Ford, Floyd, Dijkstra ，

1. 可以把Dijkstra Priority Queue 裡面存的東西想成「狀

態」， 他可以拿來統計甚至轉移。

2. 當遇到邊權會扣掉走的人的血量（ 或油量之類的） ， 當不能

有負值的時候， 就要使用Bellman -Ford 來做，

一開始可以把起點設為最初的血量（ 油量） ， 拿去做Bellman -

Ford ， 當做了n-1 次之後， 還能轉移， 那就是有負環或正

環（ 端看如何轉移Bellman -Ford ， 這部分的轉移式很自由

可以依照題目敘述亂改。）

3. 特別注意如果要判到某一個點的⻑度是不是無限小， 可在做

了n-1 次之後， 發現u->v 可以更新， 那我可以去看v

是否可以到另一點k， 如果是聯通的， 代表k 這個點的⻑

度是無限小。

**4.7 Tarjan.cpp**

割點

點u 為割點**if and** only **if** 滿足1. **or** 2.

1. u 爲樹根， 且u 有多於一個子樹。

2. u 不爲樹根， 且滿足存在(u,v) 爲樹枝邊(或稱父子邊，

即u 爲v 在搜索樹中的父親)， 使得DFN(u) <= Low(v)

。

-------------------------------------------------------

橋

一條無向邊(u,v) 是橋**if and** only **if** (u,v) 爲樹枝邊， 且

滿足DFN(u) < Low(v)。

// 0 base

struct TarjanSCC{

static const int MAXN = 1000006;

int n, dfn[MAXN], low[MAXN], scc[MAXN], scn, count;

vector<int> G[MAXN];

stack<int> stk;

bool ins[MAXN];

void tarjan(int u){

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

stk.push(u);

ins[u] = true;

for(auto v:G[u]){

if(!dfn[v]){

tarjan(v);

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

}else if(ins[v]){

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

}

}

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

int v;

do {

v = stk.top();

stk.pop();

scc[v] = scn;

ins[v] = false;

} while(v != u);

scn++;

}

}

void getSCC(){

memset(dfn,0,sizeof(dfn));

memset(low,0,sizeof(low));

memset(ins,0,sizeof(ins));

memset(scc,0,sizeof(scc));

count = scn = 0;

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

if(!dfn[i]) tarjan(i);

}

}

}SCC;

**4.8 2-SAT**

const int MAXN = 2020;

struct TwoSAT{

static const int MAXv = 2\*MAXN;

vector<int> GO[MAXv],BK[MAXv],stk;

bool vis[MAXv];

int SC[MAXv];

void imply(int u,int v){ // u imply v

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

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

}

int dfs(int u,vector<int>\*G,int sc){

vis[u]=1, SC[u]=sc;

for (int v:G[u])if (!vis[v])

dfs(v,G,sc);

if (G==GO)stk.push\_back(u);

}

int scc(int n=MAXv){

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

for (int i=0; i<n; i++)if (!vis[i])

dfs(i,GO,-1);

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

int sc=0;

while (!stk.empty()){

if (!vis[stk.back()])

dfs(stk.back(),BK,sc++);

stk.pop\_back();

}

}

}SAT;

int main(){

SAT.scc(2\*n);

bool ok=1;

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

if (SAT.SC[2\*i]==SAT.SC[2\*i+1])ok=0;

}

if (ok){

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

if (SAT.SC[2\*i]>SAT.SC[2\*i+1]){

cout << i << endl;

}

}

}

else puts("NO");

}

void warshall(){

bitset<2003> d[2003];

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

for (int i=0; i<n; i++) if (d[i][k]) {

d[i] |= d[k];

}

}

}

**4.9 平面圖判定**

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <iomanip>

#include <vector>

#include <cstring>

#include <string>

#include <queue>

#include <deque>

#include <stack>

#include <map>

#include <set>

#include <utility>

#include <list>

#include <cmath>

#include <algorithm>

#include <cassert>

#include <bitset>

#include <complex>

#include <climits>

#include <functional>

using namespace std;

typedef long long ll;

typedef pair<int, int> ii;

typedef pair<ll, ll> l4;

#define mp make\_pair

#define pb push\_back

#define debug(x) cerr << #x << " = " << x << " "

const int N=400+1;

struct Planar

{

int n,m,hash[N],fa[N],deep[N],low[N],ecp[N];

vector<int> g[N],son[N];

set< pair<int,int> > SDlist[N],proots[N];

int nxt[N][2],back[N],rev[N];

deque<int> q;

void dfs(int u)

{

hash[u]=1; q.pb(u);

ecp[u]=low[u]=deep[u];

int v;

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

if(!hash[v=g[u][i]])

{

fa[v]=u;

deep[v]=deep[u]+1;

dfs(v);

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

SDlist[u].insert(mp(low[v],v));

}

else ecp[u]=min(ecp[u],deep[v]);

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

}

int visited[N];

void addtree(int u,int t1,int v,int t2)

{

nxt[u][t1]=v; nxt[v][t2]=u;

}

void findnxt(int u,int v,int& u1,int& v1)

{

u1=nxt[u][v^1];

if(nxt[u1][0]==u) v1=0;

else v1=1;

}

void walkup(int u,int v)

{

back[v]=u;

int v1=v,v2=v,u1=1,u2=0,z;

for (;;)

{

if(hash[v1]==u || hash[v2]==u) break;

hash[v1]=u;hash[v2]=u; z=max(v1,v2);

if(z>n)

{

int p=fa[z-n];

if(p!=u)

{

proots[p].insert(mp(-low[z-n], z));

v1=p,v2=p,u1=0,u2=1;

}

else break;

}

else

{

findnxt(v1,u1,v1,u1);

findnxt(v2,u2,v2,u2);

}

}

}

int topstack;

pair<int,int> stack[N];

int outer(int u,int v)

{

return ecp[v]<deep[u] || (SDlist[v].size() && SDlist[v].begin()->first<deep[u]);

}

int inside(int u,int v)

{

return proots[v].size()>0 || back[v]==u;

}

int active(int u,int v)

{

return inside(u,v) || outer(u,v);

}

void push(int a,int b)

{

stack[++topstack]=mp(a,b);

}

void mergestack()

{

int v1,t1,v2,t2,s,s1;

v1=stack[topstack].first;t1=stack[topstack].second;

topstack--;

v2=stack[topstack].first;t2=stack[topstack].second;

topstack--;

s=nxt[v1][t1^1];

s1=(nxt[s][1]==v1);

nxt[s][s1]=v2;

nxt[v2][t2]=s;

SDlist[v2].erase( make\_pair(low[v1-n],v1-n) );

proots[v2].erase( make\_pair(-low[v1-n],v1) );

}

void findnxtActive(int u,int t,int& v,int& w1,int S)

{

findnxt(u,t,v,w1);

while(u!=v && !active(S,v))

findnxt(v,w1,v,w1);

}

void walkdown(int S,int u)

{

topstack=0;

int t1,v=S,w1,x2,y2,x1,y1,p;

for(t1=0;t1<2;++t1)

{

findnxt(S,t1^1,v,w1);

while(v!=S)

{

if(back[v]==u)

{

while(topstack>0) mergestack();

addtree(S,t1,v,w1); back[v]=0;

}

if(proots[v].size())

{

push(v,w1);

p=proots[v].begin()->second;

findnxtActive(p,1,x1,y1,u);

findnxtActive(p,0,x2,y2,u);

if(active(u,x1) && !outer(u,x1))

v=x1,w1=y1;

else if(active(u,x2) && !outer(u,x2))

v=x2,w1=y2;

else if(inside(u,x1) || back[x1]==u)

v=x1,w1=y1;

else v=x2,w1=y2;

push(p,v==x2);

}

else if(v>n || ( ecp[v]>=deep[u] && !outer(u,v) ))

findnxt(v,w1,v,w1);

else if(v<=n && outer(u,v) && !topstack)

{

addtree(S,t1,v,w1); break;

}

else break;

}

}

}

int work(int u)

{

int v;

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

if(fa[v=g[u][i]]==u)

{

son[u].push\_back(n+v);

proots[n+v].clear();

addtree(n+v,1,v,0);

addtree(n+v,0,v,1);

}

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

if(deep[v=g[u][i]]>deep[u]+1)

walkup(u,v);

topstack=0;

for (int i = 0; i < son[u].size(); ++i) walkdown(son[u][i], u);

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

if(deep[v=g[u][i]]>deep[u]+1 && back[v])

return 0;

return 1;

}

void init(int \_n)

{

n = \_n;

m = 0;

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

{

g[i].clear();

SDlist[i].clear();

son[i].clear();

proots[i].clear();

nxt[i][0]=nxt[i][1]=0;

fa[i]=0;

hash[i]=0;low[i]=ecp[i]=deep[i]=back[i]=0;

q.clear();

}

}

void add(int u, int v)

{

++m;

g[u].pb(v); g[v].pb(u);

}

bool check\_planar()

{

if(m>3\*n-5)

return false;

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

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

if(!hash[i])

{

deep[i]=1;

dfs(i);

}

memset(hash,0,sizeof hash);

//memset(hash, 0, (2\*n+1)\*sizeof(hash[0]));

// originally only looks at last n element

assert(q.size() == n);

while (!q.empty())

{

if (!work(q.back()))

return false;

q.pop\_back();

}

return true;

}

} base, \_new;

vector<ii> edges;

int n, m;

inline void build(int n, Planar &\_new)

{

\_new.init(n);

for (auto e : edges)

\_new.add(e.first, e.second);

}

void end()

{

puts("-1");

exit(0);

}

bool vis[N];

const int maxp = 5;

int path[maxp], tp=0;

void dfs(int cur)

{

vis[cur] = true;

path[tp++] = cur;

if (tp == maxp)

{

auto it = lower\_bound(base.g[cur].begin(), base.g[cur].end(), path[0]);

if ( it != base.g[cur].end() && \*it == path[0])

{

//a cycle

int x = n+1;

for (int i = 0; i < 5; ++i) edges.pb(mp(x, path[i]));

build(x, \_new);

if (\_new.check\_planar())

{

for (int i = 0; i < maxp; ++i) printf("%d%c", path[i], i==maxp-1?'\n':' ');

exit(0);

}

for (int i = 0; i < 5; ++i) edges.pop\_back();

}

}

else

{

for (auto e : base.g[cur]) if (!vis[e]) dfs(e);

}

vis[cur] = false;

--tp;

}

int main()

{

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

if (n <= 4)

{

assert(false);

puts("0"); return 0;

}

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

{

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

edges.pb(mp(u, v));

}

build(n, base);

if (!base.check\_planar()) end();

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

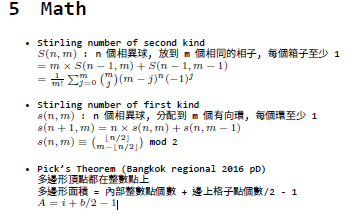
sort(base.g[i].begin(), base.g[i].end());

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

dfs(i);

end();

}



**5.1 ax+by=gcd(a,b)**

pair<int,int> extgcd(int a, int b){

if (b==0) return {1,0};

int k = a/b;

pair<int,int> p = extgcd(b,a-k\*b);

return { p.second, p.first - k\*p.second };

}

**5.2 FFT**

// use llround() to avoid EPS

typedef double Double;

const Double PI = acos(-1);

// STL complex may TLE

typedef complex<Double> Complex;

#define x real()

#define y imag()

template<typename Iter> // Complex\*

void BitReverse(Iter a, int n){

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

for (int k = n>>1; k>(j^=k); k>>=1);

if (i<j) swap(a[i],a[j]);

}

}

template<typename Iter> // Complex\*

void FFT(Iter a, int n, int rev=1){ // rev = 1 or -1

assert( (n&(-n)) == n ); // n is power of 2

BitReverse(a,n);

Iter A = a;

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

int m = (1<<s);

Complex wm( cos(2\*PI\*rev/m), sin(2\*PI\*rev/m) );

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

Complex w(1,0);

for (int j=0; j<(m>>1); j++){

Complex t = w \* A[k+j+(m>>1)];

Complex u = A[k+j];

A[k+j] = u+t;

A[k+j+(m>>1)] = u-t;

w = w\*wm;

}

}

}

if (rev==-1){

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

A[i] /= n;

}

}

}

**5.3 GaussElimination**

/ by bcw\_codebook

const int MAXN = 300;

const double EPS = 1e-8;

int n;

double A[MAXN][MAXN];

void Gauss() {

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

bool ok = 0;

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

if(fabs(A[j][i]) > EPS) {

swap(A[j], A[i]);

ok = 1;

break;

}

}

if(!ok) continue;

double fs = A[i][i];

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

double r = A[j][i] / fs;

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

A[j][k] -= A[i][k] \* r;

}

}

}

}

**5.4 inverse**

const int MAXN = 1000006;

int inv[MAXN];

void invTable(int bound, int p){

inv[1] = 1;

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

inv[i] = (long long)inv[p%i] \* (p-p/i) %p;

}

}

int inv(int b, int p){

if (b==1) return 1;

return (long long)inv(p%b,p) \* (p-p/b) %p;

}

**5.5 Miller-Rabin**

typedef long long LL;

inline LL bin\_mul(LL a, LL n,const LL& MOD){

LL re=0;

while (n>0){

if (n&1) re += a;

a += a; if (a>=MOD) a-=MOD;

n>>=1;

}

return re%MOD;

}

inline LL bin\_pow(LL a, LL n,const LL& MOD){

LL re=1;

while (n>0){

if (n&1) re = bin\_mul(re,a,MOD);

a = bin\_mul(a,a,MOD);

n>>=1;

}

return re;

}

bool is\_prime(LL n){

//static LL sprp[3] = { 2LL, 7LL, 61LL};

static LL sprp[7] = { 2LL, 325LL, 9375LL,

28178LL, 450775LL, 9780504LL,

1795265022LL };

if (n==1 || (n&1)==0 ) return n==2;

int u=n-1, t=0;

while ( (u&1)==0 ) u>>=1, t++;

for (int i=0; i<3; i++){

LL x = bin\_pow( sprp[i]%n, u, n);

if (x==0 || x==1 || x==n-1)continue;

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

x=x\*x%n;

if (x==1 || x==n-1)break;

}

if (x==n-1)continue;

return 0;

}

return 1;

}

**5.6 Mobius**

void mobius() {

fill(isPrime, isPrime + MAXN, 1);

mu[1] = 1, num = 0;

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

if (isPrime[i]) primes[num++] = i, mu[i] = -1;

static int d;

for (int j = 0; j < num && (d = i \* primes[j]) < MAXN; ++j) {

isPrime[d] = false;

if (i % primes[j] == 0) {

mu[d] = 0; break;

} else mu[d] = -mu[i];

}

}

}

**5.8 SG**

Anti Nim (取走最後一個石子者敗)

先手必勝 if and only if

1. 「所有」堆的石子數都為 1 且遊戲的 SG 值為 0。

2. 「有些」堆的石子數大於 1 且遊戲的 SG 值不為 0。

-------------------------------------------------------

Anti-SG (決策集合為空的遊戲者贏)

定義 SG 值為 0 時，遊戲結束，

則先手必勝 if and only if

1. 遊戲中沒有單一遊戲的 SG 函數大於 1 且遊戲的 SG 函數為 0。

2. 遊戲中某個單一遊戲的 SG 函數大於 1 且遊戲的 SG 函數不為 0。

-------------------------------------------------------

Sprague-Grundy

1. 雙人、回合制

2. 資訊完全公開

3. 無隨機因素

4. 可在有限步內結束

5. 沒有和局

6. 雙方可採取的行動相同

SG(S) 的值為 0：後手(P)必勝

不為 0：先手(N)必勝

int mex(set S) {

// find the min number >= 0 that not in the S

// e.g. S = {0, 1, 3, 4} mex(S) = 2

}

state = []

int SG(A) {

if (A not in state) {

S = sub\_states(A)

if( len(S) > 1 ) state[A] = reduce(operator.xor, [SG(B) for B in S])

else state[A] = mex(set(SG(B) for B in next\_states(A)))

}

return state[A]

}

**5.9 theorem**

/\*Lucas's Theorem

For non-negative integer n,m and prime P,

C(m,n) mod P = C(m/M,n/M) \* C(m%M,n%M) mod P

= mult\_i ( C(m\_i,n\_i) )

where m\_i is the i-th digit of m in base P.

-------------------------------------------------------

Kirchhoff's theorem

A\_{ii} = deg(i), A\_{ij} = (i,j) \in E ? -1 : 0

Deleting any one row, one column, and cal the det(A)

-------------------------------------------------------

Nth Catalan recursive function:

C\_0 = 1, C\_{n+1} = C\_n \* 2(2n + 1)/(n+2)

-------------------------------------------------------

Mobius Formula

u(n) = 1 , if n = 1

(-1)^m , 若 n 無平方數因數，且 n = p1\*p2\*p3\*...\*pk

0 , 若 n 有大於 1 的平方數因數

- Property

1. (積性函數) u(a)u(b) = u(ab)

2. ∑\_{d|n} u(d) = [n == 1]

-------------------------------------------------------

Mobius Inversion Formula

if f(n) = ∑\_{d|n} g(d)

then g(n) = ∑\_{d|n} u(n/d)f(d)

= ∑\_{d|n} u(d)f(n/d)

- Application

the number/power of gcd(i, j) = k

- Trick

分塊, O(sqrt(n))

-------------------------------------------------------

Chinese Remainder Theorem (m\_i 兩兩互質)

x = a\_1 (mod m\_1)

x = a\_2 (mod m\_2)

....

x = a\_i (mod m\_i)

construct a solution:

Let M = m\_1 \* m\_2 \* m\_3 \* ... \* m\_n

Let M\_i = M / m\_i

t\_i = 1 / M\_i

t\_i \* M\_i = 1 (mod m\_i)

solution x = a\_1 \* t\_1 \* M\_1 + a\_2 \* t\_2 \* M\_2 + ... + a\_n \* t\_n \* M\_n + k \* M

= k\*M + ∑ a\_i \* t\_i \* M\_i, k is positive integer.

under mod M, there is one solution x = ∑ a\_i \* t\_i \* M\_i

-------------------------------------------------------

Burnside's lemma

|G| \* |X/G| = sum( |X^g| ) where g in G

總方法數: 每一種旋轉下不動點的個數總和 除以 旋轉的方法數

\*/

**6 Geometry**

**6.1 2D point template**

typedef double Double;

struct Point {

Double x,y;

bool operator < (const Point &b)const{

//return tie(x,y) < tie(b.x,b.y);

//return atan2(y,x) < atan2(b.y,b.x);

assert(0 && "choose compare");

}

Point operator + (const Point &b)const{

return (Point){x+b.x,y+b.y};

}

Point operator - (const Point &b)const{

return (Point){x-b.x,y-b.y};

}

Point operator \* (const Double &d)const{

return Point(d\*x,d\*y);

}

Double operator \* (const Point &b)const{

return x\*b.x + y\*b.y;

}

Double operator % (const Point &b)const{

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

}

friend Double abs2(const Point &p){

return p.x\*p.x + p.y\*p.y;

}

friend Double abs(const Point &p){

return sqrt( abs2(p) );

}

};

typedef Point Vector;

struct Line{

Point P; Vector v;

bool operator < (const Line &b)const{

return atan2(v.y,v.x) < atan2(b.v.y,b.v.x);

}

};

**6.2 circumcentre**

#include "2Dpoint.cpp"

Point circumcentre(Point &p0, Point &p1, Point &p2){

Point a = p1-p0;

Point b = p2-p0;

Double c1 = abs2(a)\*0.5;

Double c2 = abs2(b)\*0.5;

Double d = a % b;

Double x = p0.x + ( c1\*b.y - c2\*a.y ) / d;

Double y = p0.y + ( c2\*a.x - c1\*b.x ) / d;

return {x,y};

}

**6.3 ConvexHull**

#include "2Dpoint.cpp"

// retunr H, 第一個點會在 H 出現兩次

void ConvexHull(vector<Point> &P, vector<Point> &H){

int n = P.size(), m=0;

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

H.clear();

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

while (m>=2 && (P[i]-H[m-2]) % (H[m-1]-H[m-2]) <0)H.pop\_back(), m--;

H.push\_back(P[i]), m++;

}

for (int i=n-2; i>=0; i--){

while (m>=2 && (P[i]-H[m-2]) % (H[m-1]-H[m-2]) <0)H.pop\_back(), m--;

H.push\_back(P[i]), m++;

}

}

**6.4 half plane intersection**

bool OnLeft(const Line& L,const Point& p){

return Cross(L.v,p-L.P)>0;

}

Point GetIntersection(Line a,Line b){

Vector u = a.P-b.P;

Double t = Cross(b.v,u)/Cross(a.v,b.v);

return a.P + a.v\*t;

}

int HalfplaneIntersection(Line\* L,int n,Point\* poly){

sort(L,L+n);

int first,last;

Point \*p = new Point[n];

Line \*q = new Line[n];

q[first=last=0] = L[0];

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

while(first < last && !OnLeft(L[i],p[last-1])) last--;

while(first < last && !OnLeft(L[i],p[first])) first++;

q[++last]=L[i];

if(fabs(Cross(q[last].v,q[last-1].v))<EPS){

last--;

if(OnLeft(q[last],L[i].P)) q[last]=L[i];

}

if(first < last) p[last-1]=GetIntersection(q[last-1],q[last]);

}

while(first<last && !OnLeft(q[first],p[last-1])) last--;

if(last-first<=1) return 0;

p[last]=GetIntersection(q[last],q[first]);

int m=0;

for(int i=first;i<=last;i++) poly[m++]=p[i];

return m;

}

**6.5 Intersection of two circle**

vector<Double> interCircle(Double o1, Double r1, Double o2, Double r2) {

Double d2 = abs2(o1 - o2);

Double d = sqrt(d2);

if (d < fabs(r1-r2) || r1+r2 < d) return {};

Double u = 0.5\*(o1+o2) + ((r2\*r2-r1\*r1)/(2.0\*d2))\*(o1-o2);

Double A = sqrt((r1+r2+d) \* (r1-r2+d) \* (r1+r2-d) \* (-r1+r2+d));

Double v = A / (2.0\*d2) \* Double(o1.S-o2.S, -o1.F+o2.F);

return {u+v, u-v};

}

**6.6 Intersection of two lines**

Point interPnt(Point p1, Point p2, Point q1, Point q2, bool &res){

Double f1 = cross(p2, q1, p1);

Double f2 = -cross(p2, q2, p1);

Double f = (f1 + f2);

if(fabs(f) < EPS) {

res = false;

return {};

}

res = true;

return (f2 / f) \* q1 + (f1 / f) \* q2;

}

**6.7 Smallest Circle**

#include "circumcentre.cpp"

pair<Point,Double> SmallestCircle(int n, Point \_p[]){

Point \*p = new Point[n];

memcpy(p,\_p,sizeof(Point)\*n);

random\_shuffle(p,p+n);

Double r2=0;

Point cen;

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

if ( abs2(cen-p[i]) <= r2)continue;

cen = p[i], r2=0;

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

if ( abs2(cen-p[j]) <= r2)continue;

cen = (p[i]+p[j])\*0.5;

r2 = abs2(cen-p[i]);

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

if ( abs2(cen-p[k]) <= r2)continue;

cen = circumcentre(p[i],p[j],p[k]);

r2 = abs2(cen-p[k]);

}

}

}

delete[] p;

return {cen,r2};

}

// auto res = SmallestCircle(,);

**7 String**

**7.1 AC automaton**

// remember make\_fail() !!!

// notice MLE

const int sigma = 62;

const int MAXC = 200005;

inline int idx(char c){

if ('A'<= c && c <= 'Z')return c-'A';

if ('a'<= c && c <= 'z')return c-'a' + 26;

if ('0'<= c && c <= '9')return c-'0' + 52;

}

struct ACautomaton{

struct Node{

Node \*next[sigma], \*fail;

int cnt; // dp

Node(){

memset(next,0,sizeof(next));

fail=0;

cnt=0;

}

} buf[MAXC], \*bufp, \*ori, \*root;

void init(){

bufp = buf;

ori = new (bufp++) Node();

root = new (bufp++) Node();

}

void insert(int n, char \*s){

Node \*ptr = root;

for (int i=0; s[i]; i++){

int c = idx(s[i]);

if (ptr->next[c]==NULL)

ptr->next[c] = new (bufp++) Node();

ptr = ptr->next[c];

}

ptr->cnt=1;

}

Node\* trans(Node \*o, int c){

while (o->next[c]==NULL) o = o->fail;

return o->next[c];

}

void make\_fail(){

static queue<Node\*> que;

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

ori->next[i] = root;

root->fail = ori;

que.push(root);

while ( que.size() ){

Node \*u = que.front(); que.pop();

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

if (u->next[i]==NULL)continue;

u->next[i]->fail = trans(u->fail,i);

que.push(u->next[i]);

}

u->cnt += u->fail->cnt;

}

}

} ac;

**7.2 KMP**

template<typename T>

void build\_KMP(int n, T \*s, int \*f){ // 1 base

f[0]=-1, f[1]=0;

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

int w = f[i-1];

while (w>=0 && s[w+1]!=s[i])w = f[w];

f[i]=w+1;

}

}

template<typename T>

int KMP(int n, T \*a, int m, T \*b){

build\_KMP(m,b,f);

int ans=0;

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

while ( w>=0 && b[w+1]!=a[i] )w = f[w];

w++;

if (w==m){

ans++;

w=f[w];

}

}

return ans;

}

**7.3 palindromic tree**

// remember init() !!!

// remember make\_fail() !!!

// insert s need 1 base !!!

// notice MLE

const int sigma = 62;

const int MAXC = 1000006;

inline int idx(char c){

if ('a'<= c && c <= 'z')return c-'a';

if ('A'<= c && c <= 'Z')return c-'A'+26;

if ('0'<= c && c <= '9')return c-'0'+52;

}

struct PalindromicTree{

struct Node{

Node \*next[sigma], \*fail;

int len, cnt; // for dp

Node(){

memset(next,0,sizeof(next));

fail=0;

len = cnt = 0;

}

} buf[MAXC], \*bufp, \*even, \*odd;

void init(){

bufp = buf;

even = new (bufp++) Node();

odd = new (bufp++) Node();

even->fail = odd;

odd->len = -1;

}

void insert(char \*s){

Node\* ptr = even;

for (int i=1; s[i]; i++){

ptr = extend(ptr,s+i);

}

}

Node\* extend(Node \*o, char \*ptr){

int c = idx(\*ptr);

while ( \*ptr != \*(ptr-1-o->len) )o=o->fail;

Node \*&np = o->next[c];

if (!np){

np = new (bufp++) Node();

np->len = o->len+2;

Node \*f = o->fail;

if (f){

while ( \*ptr != \*(ptr-1-f->len) )f=f->fail;

np->fail = f->next[c];

}

else {

np->fail = even;

}

np->cnt = np->fail->cnt;

}

np->cnt++;

return np;

}

} PAM;

**7.4 SAM**

// par : fail link

// val : a topological order ( useful for DP )

// go[x] : automata edge ( x is integer in [0,26) )

struct SAM{

struct State{

int par, go[26], val;

State () : par(0), val(0){ FZ(go); }

State (int \_val) : par(0), val(\_val){ FZ(go); }

};

vector<State> vec;

int root, tail;

void init(int arr[], int len){

vec.resize(2);

vec[0] = vec[1] = State(0);

root = tail = 1;

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

extend(arr[i]);

}

void extend(int w){

int p = tail, np = vec.size();

vec.PB(State(vec[p].val+1));

for ( ; p && vec[p].go[w]==0; p=vec[p].par)

vec[p].go[w] = np;

if (p == 0){

vec[np].par = root;

} else {

if (vec[vec[p].go[w]].val == vec[p].val+1){

vec[np].par = vec[p].go[w];

} else {

int q = vec[p].go[w], r = vec.size();

vec.PB(vec[q]);

vec[r].val = vec[p].val+1;

vec[q].par = vec[np].par = r;

for ( ; p && vec[p].go[w] == q; p=vec[p].par)

vec[p].go[w] = r;

}

}

tail = np;

}

};

**7.5 smallest rotation**

string mcp(string s){

int n = s.length();

s += s;

int i=0, j=1;

while (i<n && j<n){

int k = 0;

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

if (s[i+k] <= s[j+k]) j += k+1;

else i += k+1;

if (i == j) j++;

}

int ans = i < n ? i : j;

return s.substr(ans, n);

}

**7.6 suffix array**

/\*he[i]保存了在後綴數組中相鄰兩個後綴的最長公共前綴長度

\*sa[i]表示的是字典序排名為i的後綴是誰（字典序越小的排名越靠前）

\*rk[i]表示的是後綴我所對應的排名是多少 \*/

const int MAX = 1020304;

int ct[MAX], he[MAX], rk[MAX];

int sa[MAX], tsa[MAX], tp[MAX][2];

void suffix\_array(char \*ip){

int len = strlen(ip);

int alp = 256;

memset(ct, 0, sizeof(ct));

for(int i=0;i<len;i++) ct[ip[i]+1]++;

for(int i=1;i<alp;i++) ct[i]+=ct[i-1];

for(int i=0;i<len;i++) rk[i]=ct[ip[i]];

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

for(int j=0;j<len;j++){

if(j+i>=len) tp[j][1]=0;

else tp[j][1]=rk[j+i]+1;

tp[j][0]=rk[j];

}

memset(ct, 0, sizeof(ct));

for(int j=0;j<len;j++) ct[tp[j][1]+1]++;

for(int j=1;j<len+2;j++) ct[j]+=ct[j-1];

for(int j=0;j<len;j++) tsa[ct[tp[j][1]]++]=j;

memset(ct, 0, sizeof(ct));

for(int j=0;j<len;j++) ct[tp[j][0]+1]++;

for(int j=1;j<len+1;j++) ct[j]+=ct[j-1];

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

sa[ct[tp[tsa[j]][0]]++]=tsa[j];

rk[sa[0]]=0;

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

if( tp[sa[j]][0] == tp[sa[j-1]][0] &&

tp[sa[j]][1] == tp[sa[j-1]][1] )

rk[sa[j]] = rk[sa[j-1]];

else

rk[sa[j]] = j;

}

}

for(int i=0,h=0;i<len;i++){

if(rk[i]==0) h=0;

else{

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

h=max(0,h-1);

for(;ip[i+h]==ip[j+h];h++);

}

he[rk[i]]=h;

}

}

**7.7 Z value**

z[0] = 0;

for ( int bst = 0, i = 1; i < len ; i++ ) {

if ( z[bst] + bst <= i ) z[i] = 0;

else z[i] = min(z[i - bst], z[bst] + bst - i);

while ( str[i + z[i]] == str[z[i]] ) z[i]++;

if ( i + z[i] > bst + z[bst] ) bst = i;

}

// 回文版

void Zpal(const char \*s, int len, int \*z) {

// Only odd palindrome len is considered

// z[i] means that the longest odd palindrom centered at

// i is [i-z[i] .. i+z[i]]

z[0] = 0;

for (int b=0, i=1; i<len; i++) {

if (z[b] + b >= i) z[i] = min(z[2\*b-i], b+z[b]-i);

else z[i] = 0;

while (i+z[i]+1 < len and i-z[i]-1 >= 0 and

s[i+z[i]+1] == s[i-z[i]-1]) z[i] ++;

if (z[i] + i > z[b] + b) b = i;

}

}

**7.8 BWT (Burrows-Wheeler Transform)**

string BWT(string); // by suffix array

string iBWT(string &s, int start=0){

int n = (int) s.size();

string ret(n,' ');

vector<int> next(n,0), box[256];

for (int i=0; i<n; i++) // bucket sort

box[ (int)s[i] ].push\_back(i);

for (int i=0, j=0; i<256; i++)

for (int x:box[i])

next[j++] = x;

for (int i=0, p=start; i<n; i++)

ret[i] = s[ p=next[p] ];

return ret;

}

**8 Data structure**

**8.3 KD tree**

const int MXN = 100005;

struct KDTree {

struct Nd {

int x,y,x1,y1,x2,y2;

int id,f;

Nd \*L, \*R;

}tree[MXN];

int n;

Nd \*root;

LL dis2(int x1, int y1, int x2, int y2) {

LL dx = x1-x2; LL dy = y1-y2;

return dx\*dx+dy\*dy;

}

static bool cmpx(Nd& a, Nd& b){ return a.x<b.x; }

static bool cmpy(Nd& a, Nd& b){ return a.y<b.y; }

void init(vector<pair<int,int>> ip) {

n = ip.size();

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

tree[i].id = i;

tree[i].x = ip[i].first;

tree[i].y = ip[i].second;

}

root = build\_tree(0, n-1, 0);

}

Nd\* build\_tree(int L, int R, int dep) {

if (L>R) return nullptr;

int M = (L+R)/2;

tree[M].f = dep%2;

nth\_element(tree+L, tree+M, tree+R+1,

tree[M].f ? cmpy : cmpx);

tree[M].x1 = tree[M].x2 = tree[M].x;

tree[M].y1 = tree[M].y2 = tree[M].y;

tree[M].L = build\_tree(L, M-1, dep+1);

if (tree[M].L) {

tree[M].x1 = min(tree[M].x1, tree[M].L->x1);

tree[M].x2 = max(tree[M].x2, tree[M].L->x2);

tree[M].y1 = min(tree[M].y1, tree[M].L->y1);

tree[M].y2 = max(tree[M].y2, tree[M].L->y2);

}

tree[M].R = build\_tree(M+1, R, dep+1);

if (tree[M].R) {

tree[M].x1 = min(tree[M].x1, tree[M].R->x1);

tree[M].x2 = max(tree[M].x2, tree[M].R->x2);

tree[M].y1 = min(tree[M].y1, tree[M].R->y1);

tree[M].y2 = max(tree[M].y2, tree[M].R->y2);

}

return tree+M;

}

int touch(Nd\* r, int x, int y, LL d2){

LL dis = sqrt(d2)+1;

if (x<r->x1-dis || x>r->x2+dis ||

y<r->y1-dis || y>r->y2+dis)

return 0;

return 1;

}

void nearest(Nd\* r, int x, int y, int &mID, LL &md2){

if (!r || !touch(r, x, y, md2)) return;

LL d2 = dis2(r->x, r->y, x, y);

if (d2 < md2 || (d2 == md2 && mID < r->id)) {

mID = r->id; md2 = d2;

}

// search order depends on split dim

if ((r->f == 0 && x < r->x) ||

(r->f == 1 && y < r->y)) {

nearest(r->L, x, y, mID, md2);

nearest(r->R, x, y, mID, md2);

} else {

nearest(r->R, x, y, mID, md2);

nearest(r->L, x, y, mID, md2);

}

}

int query(int x, int y) {

int id = 1029384756;

LL d2 = 102938475612345678LL;

nearest(root, x, y, id, d2);

return id;

}

}tree;

**8.4 Link-Cut tree**

const int MXN = 100005;

const int MEM = 100005;

struct Splay {

static Splay nil, mem[MEM], \*pmem;

Splay \*ch[2], \*f;

int val, rev, size;

Splay (int \_val=-1) : val(\_val), rev(0), size(1)

{ f = ch[0] = ch[1] = &nil; }

bool isr()

{ return f->ch[0] != this && f->ch[1] != this; }

int dir()

{ return f->ch[0] == this ? 0 : 1; }

void setCh(Splay \*c, int d){

ch[d] = c;

if (c != &nil) c->f = this;

pull();

}

void push(){

if( !rev ) return;

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

if (ch[0] != &nil) ch[0]->rev ^= 1;

if (ch[1] != &nil) ch[1]->rev ^= 1;

rev=0;

}

void pull(){

size = ch[0]->size + ch[1]->size + 1;

if (ch[0] != &nil) ch[0]->f = this;

if (ch[1] != &nil) ch[1]->f = this;

}

} Splay::nil, Splay::mem[MEM], \*Splay::pmem = Splay::mem;

Splay \*nil = &Splay::nil;

void rotate(Splay \*x){

Splay \*p = x->f;

int d = x->dir();

if (!p->isr()) p->f->setCh(x, p->dir());

else x->f = p->f;

p->setCh(x->ch[!d], d);

x->setCh(p, !d);

p->pull(); x->pull();

}

vector<Splay\*> splayVec;

void splay(Splay \*x){

splayVec.clear();

for (Splay \*q=x;; q=q->f){

splayVec.push\_back(q);

if (q->isr()) break;

}

reverse(begin(splayVec), end(splayVec));

for (auto it : splayVec) it->push();

while (!x->isr()) {

if (x->f->isr()) rotate(x);

else if (x->dir()==x->f->dir())

rotate(x->f),rotate(x);

else rotate(x),rotate(x);

}

}

int id(Splay \*x) { return x - Splay::mem + 1; }

Splay\* access(Splay \*x){

Splay \*q = nil;

for (;x!=nil;x=x->f){

splay(x);

x->setCh(q, 1);

q = x;

}

return q;

}

void chroot(Splay \*x){

access(x);

splay(x);

x->rev ^= 1;

x->push(); x->pull();

}

void link(Splay \*x, Splay \*y){

access(x);

splay(x);

chroot(y);

x->setCh(y, 1);

}

void cut\_p(Splay \*y) {

access(y);

splay(y);

y->push();

y->ch[0] = y->ch[0]->f = nil;

}

void cut(Splay \*x, Splay \*y){

chroot(x);

cut\_p(y);

}

Splay\* get\_root(Splay \*x) {

access(x);

splay(x);

for(; x->ch[0] != nil; x = x->ch[0])

x->push();

splay(x);

return x;

}

bool conn(Splay \*x, Splay \*y) {

x = get\_root(x);

y = get\_root(y);

return x == y;

}

Splay\* lca(Splay \*x, Splay \*y) {

access(x);

access(y);

splay(x);

if (x->f == nil) return x;

else return x->f;

}

**9.9 java cheat sheet**

import java.util.\*;

import java.math.\*;

import java.io.\*;

public class java{

static class Comp implements Comparator<Integer>{

public int compare(Integer lhs, Integer rhs){

return lhs - rhs;

}

}

static class Yee implements Comparable<Yee>{

public int compareTo(Yee y){

return 0;

}

}

static class Reader{

private BufferedReader br;

private StringTokenizer st;

public Reader(){

br = new BufferedReader(new InputStreamReader(System.in));

}

boolean hasNext() throws IOException{

String s;

while (st == null || !st.hasMoreElements()){

if ((s = br.readLine())==null) return false;

st = new StringTokenizer(s);

}

return true;

}

String next() throws IOException{

while (st == null || !st.hasMoreElements())

st = new StringTokenizer(br.readLine());

return st.nextToken();

}

int nextInt() throws IOException{

return Integer.parseInt(next());

}// Long.parseLong, Double.parseDouble, br.readLine

}

public static void main(String args[])throws IOException{

Reader cin = new Reader();

//Scanner cin = new Scanner(System.in);

PrintWriter cout = new PrintWriter(System.out);

//Scanner cin = new Scanner(new File("t.in"));

//PrintWriter cout = new PrintWriter(new File("t.out"));

// \*\*\*\*\* cout.close() or cout.flush() is needed \*\*\*\*\*

// 2D array: int[][] a = new int[10][10];

// input, EOF, Graph

int n = cin.nextInt();

// nextFloat, nextLine, next

ArrayList<ArrayList<Integer>> G = new ArrayList<>();

for (int i=0; i<n; i++) G.add(new ArrayList<>());

while (cin.hasNext()){ // EOF

int u = cin.nextInt(), v = cin.nextInt();

G.get(u).add(v);

}

// Math: E, PI, min, max, random(double 0~1), sin...

// Collections(List a): swap(a,i,j), sort(a[,comp]), min(a), binarySearch(a,val[,comp])

// set

Set<Integer> set = new TreeSet<>();

set.add(87); set.remove(87);

if (!set.contains(87)) cout.println("no 87");

// map

Map<String, Integer> map = new HashMap<>();

map.put("0", 1); map.put("2", 3);

for ( Map.Entry<String,Integer> i : map.entrySet() )

cout.println(i.getKey() + " " + i.getValue() + " wry");

cout.println( map.get("1") );

// Big Number: TEN ONE ZERO, modInverse isProbablePrime modInverse modPow

// add subtract multiply divide remainder, and or xor not shiftLeft shiftRight

// queue: add, peek(==null), poll

PriorityQueue<Integer> pq = new PriorityQueue<Integer>(Collections.reverseOrder());

Queue<Integer> q = new ArrayDeque<Integer>();

// stack: push, empty, pop

Stack<Integer> s = new Stack<Integer>();

cout.close();

}

}

欧拉函数的用处，在于[欧拉定理]。"欧拉定理"指的是：

如果两个正整数a和n互质，则n的欧拉函数 φ(n) 可以让下面的等式成立：

2015-08-04/55c059520a3e8

也就是说，a的φ(n)次方被n除的余数为1。或者说，a的φ(n)次方减去1，可以被n整除。比如，3和7互质，而7的欧拉函数φ(7)等于6，所以3的6次方（729）减去1，可以被7整除（728/7=104）。

欧拉定理的证明比较复杂，这里就省略了。我们只要记住它的结论就行了。

欧拉定理可以大大简化某些运算。比如，7和10互质，根据欧拉定理，

2015-08-04/55c05962e83e9

已知 φ(10) 等于4，所以马上得到7的4倍数次方的个位数肯定是1。

2015-08-04/55c0596cb6dd4

因此，7的任意次方的个位数（例如7的222次方），心算就可以算出来。

欧拉定理有一个特殊情况。

假设正整数a与质数p互质，因为质数p的φ(p)等于p-1，则欧拉定理可以写成

2015-08-04/55c0597866ff0

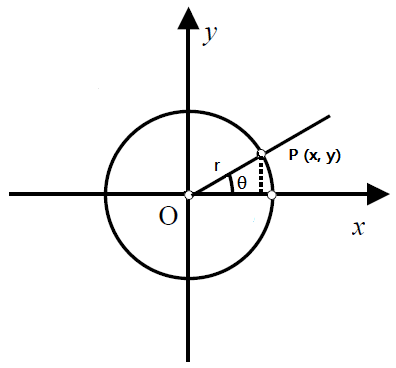
这就是著名的[费马小定理](http://zh.wikipedia.org/wiki/%E8%B4%B9%E9%A9%AC%E5%B0%8F%E5%AE%9A%E7%90%86)。它是欧拉定理的特例。

欧拉定理是RSA算法的核心。理解了这个定理，就可以理解RSA。

一、廣義角三角函數：   
sin(θ) 與 cos(θ) 的定義：   
在坐標平面上，以原點 O 為圓心，有一個半徑等於 r 的圓，給定一個廣義角 θ，規定 θ 的起始邊為 x 軸的正方向，θ 角的頂點為原點，依逆時針旋轉，則根據 θ 的旋轉量，可決定終邊的位置。   
假設終邊這條射線與圓交於P(x,y)，則定義：

sin(θ) = y / r

cos(θ) = x / r

   
其它三角函數的定義： 

tan(θ) = sin(θ) / cos(θ) = y / x

cot(θ) = cos(θ) / sin(θ) = x / y

sec(θ) = 1 / cos(θ) = r / x

csc(θ) = 1 / sin(θ) = r / y  
三角函數的關係：   
平方關係：

sin2(θ) + cos2(θ) = 1  
商數關係：

tan(θ) = sin(θ) / cos(θ) ；

cot(θ) = cos(θ) / sin(θ)  
餘角關係：

sin(90°-θ) = cos(θ) ； cos(90°-θ) = sin(θ) ；

tan(90°-θ) = cot(θ) ； cot(90°-θ) = tan(θ) ；

sec(90°-θ) = csc(θ) ； csc(90°-θ) = sec(θ)  
三角函數的化簡：   
  
化簡到 360° 內：

sin(n\*360°+θ) = sin(θ)

cos(n\*360°+θ) = cos(θ)

tan(n\*360°+θ) = tan(θ)  
化簡到 180° 內：

sin(180°+θ) = -sin(θ)；

sin(180°-θ) = sin(θ)；

cos(180°+θ) = -cos(θ)；

cos(180°-θ) = -cos(θ)；

tan(180°+θ) = tan(θ)；

tan(180°-θ) = -tan(θ)  
餘角關係

sin(90°+θ) = cos(θ)；

cos(90°+θ) = -sin(θ)；

tan(90°+θ) = -cot(θ)；

sin(270°-θ) = -cos(θ)；

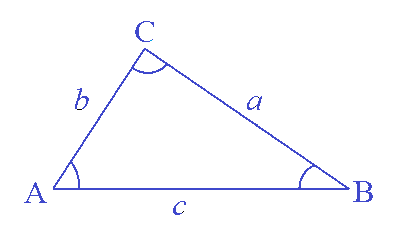
cos(270°-θ) = -sin(θ)；

tan(270°-θ) = cot(θ)  
負角關係：

sin(-θ) = -sin(θ)；

cos(-θ) = cos(θ)；

tan(-θ) = -tan(θ)

二、三角函數在平面幾何上的應用：   
  
   
  
正弦定理：(Sine Rule)   
在 ∆ABC 中，以 a, b, c 分別為 ∠A, ∠B, ∠C 的對邊長，則：

a / sin(A) = b / sin(B) = c / sin(C) = 2R

其中 R 為 ∆ABC 的外接圓半徑。  
餘弦定理：(Cosine Rule)   
在 ∆ABC 中，以 a, b, c 分別為 ∠A, ∠B, ∠C 的對邊長，則：

a2 = b2 + c2 - 2bc \* cos(A)

b2 = a2 + c2 - 2ac \* cos(B)

c2 = a2 + b2 - 2ab \* cos(C)   
三、三角函數的公式：   
複角公式：(合、分角公式) (Angle-Sum and -Difference Identities)

sin(α + β) = sin(α)cos(β) + cos(α)sin(β)

sin(α – β) = sin(α)cos(β) – cos(α)sin(β)

cos(α + β) = cos(α)cos(β) – sin(α)sin(β)

cos(α – β) = cos(α)cos(β) + sin(α)sin(β)

tan(a + b) = [tan(a) + tan(b)] / [1 - tan(a)tan(b)]

tan(a - b) = [tan(a) - tan(b)] / [1 + tan(a)tan(b)]   
倍角公式：(Double-Angle Identities)

sin(2x) = 2sin(x)cos(x)

cos(2x) = cos2(x) – sin2(x)

= 1 – 2sin2(x)

= 2cos2(x) – 1

tan(2x) = [2 tan(x)] / [1 - tan2(x)]  
半角公式：(Half-Angle Identities)

sin(x/2) = +/- sqrt[(1 - cos(x))/2]

cos(x/2) = +/- sqrt[(1 + cos(x))/2]

tan(x/2) = +/- sqrt[(1 - cos(x))/(1 + cos(x))]

= (1 - cos(x)) / sin(x)

= sin(x) / (1 + cos(x))  
和差化積：(Sum Identities)

sin(x) + sin(y) = 2 sin[(x+y)/2] cos[(x-y)/2]

sin(x) - sin(y) = 2 cos[(x+y)/2] sin[(x-y)/2]

cos(x) + cos(y) = 2 cos[(x+y)/2] cos[(x-y)/2]

cos(x) - cos(y) = -2 sin[(x+y)/2] sin[(x-y)/2]  
積化和差：(Product Identities)

sin(x) cos(y) = (1/2) [sin(x+y) + sin(x-y)]

cos(x) sin(y) = (1/2) [sin(x+y) - sin(x-y)]

cos(x) cos(y) = (1/2) [cos(x-y) + cos(x+y)]

sin(x) sin(y) = (1/2) [cos(x-y) - cos(x+y)]