

Team Notebook

PTIT1



# Xử lý số nguyên lớn:

#include <cstring>

#include <cctype>

#include <cstdlib>

#include <cstdio>

#include <cmath>

#include <iostream>

#include <algorithm>

#include <vector>

#include <map>

#include <set>

#include <queue>

#include <stack>

#include <utility>

typedef unsigned long long ULL;

typedef long long LL;

#define FOR(i,a,b) for(int i=a;i<=b;i++)

#define FO(i,a,b) for(int i=a;i<b;i++)

#define FORD(i,a,b) for(int i=a;i>=b;i--)

#define FOD(i,a,b) for(int i=a;i>b;i--)

#define FORV(i,a) for(typeof(a.begin()) i = a.begin(); i != a.end(); i++)

#define Set(a,c) memset(a, c, sizeof(a))

#define pb push\_back

#define mp make\_pair

#define fi first

#define se second

using namespace std;

class bigNum {

private:

vector <int> num;

/\* Lưu ý: Các chữ số được xếp theo thứ tự ngược (tức là: chữ số đầu tiên là chữ số hàng đơn vị, rồi đến hàng chục,…) \*/

public:

bigNum() { //khởi tạo

num.clear();

}

bigNum(string s) { //khởi tạo số bigNum từ xâu C++

FORD (i,s.size()-1,0)

num.pb(s[i]-'0');

}

bigNum (int x) { //tạo số bigNum từ số nguyên

while (x>0) {

num.pb(x%10);

x/=10;

}

}

void push\_back(int x) { //thêm vào sau số bigNum

num.pb(x);

}

void del\_0() { //xóa bỏ các chữ số 0 vô nghĩa

while (num.size()>1 && num.back()==0) num.pop\_back();

}

int size() { // số các số của bigNum

return num.size();

}

friend void add\_0(bigNum &a,bigNum &b) {//cho a,b có cùng số chữ số

if (a.size()<b.size())

while (a.size()!=b.size()) a.pb(0);

if (a.size()>b.size())

while (a.size()!=b.size()) b.pb(0);

}

friend istream &operator>>(istream &in,bigNum &x) { //nhập

string s;

in >> s;

x = bigNum(s);

x.del\_0();

return in;

}

friend ostream &operator<<(ostream &out,bigNum x) { //xuất

FORD (i,x.size()-1,0)

out << x.num[i];

return out;

}

friend bigNum operator+(bigNum a,bigNum b) { //cộng

bigNum ans;

add\_0(a,b);

int memo=0;

FO (i,0,a.size()) {

ans.pb((a.num[i]+b.num[i]+memo)%10);

memo = (a.num[i]+b.num[i]+memo)/10;

}

if (memo) ans.pb(memo);

return ans;

}

friend bigNum operator-(bigNum a,bigNum b) { //trừ

bigNum ans;

int memo=0;

add\_0(a,b);

FO (i,0,a.size()) {

ans.pb((a.num[i]-b.num[i]-memo+10)%10);

if (a.num[i]-b.num[i]-memo<0) memo=1; else memo=0;

}

ans.del\_0();

return ans;

}

friend bigNum operator\*(bigNum a,bigNum b) { //nhân

bigNum ans;

int memo=0,tmp,jj;

FO (i,0,b.size()) a.pb(0);

FO (i,0,a.size()) {

tmp=memo;

jj=i;

FO (j,0,b.size()) {

tmp+=a.num[jj--]\*b.num[j];

if (jj<0) break;

}

ans.pb(tmp%10);

memo=tmp/10;

}

ans.del\_0();

return ans;

}

};

# Sàng số nguyên tố:

// sàng Eratosthenes: O(nloglogn)

// sieve: sàng số nguyên tố từ 2 đến n

void sieve(int n) {

memset(prime, 1, sizeof(prime));

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

if (prime[i]) for (int j=i\*i; j<=n; j+=i) prime[j]=false;

//chu y: for j=i\*i

}

# Thuật toán so sánh chuỗi con KMP:

void initKMP() {

next[0] = -1;

int j = -1;

FO (i,1,M) {

while (j>-1 && P[i]!=P[j+1]) j=next[j];

if (P[i] == P[j+1]) j++;

next[i] =j;

}

}

void process() { //T=Text; P=Pattern

int j = -1;

FO (i,0,N) {

//tim xau P[1..j] la suffix cua T[1..i-1] va P[j+1] == T[i]

while (j>-1 && T[i]!=P[j+1]) j = next[j];

if (T[i] == P[j+1]) j++;

if (j>=M-1) {

kq[++dem] = i-j+1;

//khi tim thay roi thi dich luon

j = next[j];

}

}

}

# Tìm thành phần liên thông mạnh:

vector <int> a[maxn]; // danh sách kề

stack <int> s;

int n,m,t;

int number[maxn],low[maxn],Free[maxn],Count,ans;

void visit(int u) {

low[u]=number[u]=++Count;

s.push(u);

FO (i,0,a[u].size()) {

int v = a[u][i];

if (Free[v]==0) {

if (number[v]==0) {

visit(v);

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

} else low[u]=min(low[u],number[v]);

}

}

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

ans++; //tăng số thành phần liên thông mạnh

set <int> t; //tập các đỉnh trong thành phần lt này

set <int> :: iterator it;

while (1) {

t.insert(s.top());

Free[s.top()]=1;

if (s.top()==u) {

s.pop();

break;

};

s.pop();

}

// in ra các đỉnh thuộc tplt

for (it=t.begin();it!=t.end();it++) cout << \*it;

cout << endl;

}

}

main() {

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

int u,v;

FOR (i,1,m) {

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

a[u].pb(v); // danh sách kề

}

FOR (i,1,n)

if (Free[i]==0) visit(i);

cout << ans;

}

# Tìm khớp và cầu:

*low*=1

*low*=1

*low*=1

*low*=2

*low*=2

*low*=2

*low*=+∞

*low*=4

*low*=4

*low*=4

Hình 5‑4. Cách đánh số và ghi nhận cung ngược lên cao nhất

Hãy để ý một cung DFS ( là nút cha của nút trên cây DFS)

* Nếu từ nhánh DFS gốc không có cung nào ngược lên phía trên có nghĩa là từ một đỉnh thuộc nhánh DFS gốc đi theo các cung định hướng chỉ đi được tới những đỉnh nội bộ trong nhánh DFS gốc mà thôi chứ không thể tới được , suy ra là một cầu. Cũng dễ dàng chứng minh được điều ngược lại. Vậy là cầu nếu và chỉ nếu . Như ví dụ ở Hình 5‑4, ta có và là cầu.
* Nếu từ nhánh DFS gốc không có cung nào ngược lên phía trên , tức là nếu bỏ đi thì từ không có cách nào lên được các tiền bối của . Điều này chỉ ra rằng nếu không phải là nút gốc của một cây DFS thì là khớp. Cũng không khó khăn để chứng minh điều ngược lại. Vậy nếu không là gốc của một cây DFS thì là khớp nếu và chỉ nếu . Như ví dụ ở Hình 5‑4, ta có B, C, E và F là khớp.
* Gốc của một cây DFS thì là khớp nếu và chỉ nếu nó có từ hai 2 nhánh con trở lên. Như ví dụ ở Hình 5‑4, gốc A không là khớp vì nó chỉ có một nhánh con.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

set <int> a[maxn];

int n,m,dem,parent[maxn],num[maxn],low[maxn],numChild[maxn];

int soCau,soKhop;

bool laKhop[maxn];

/\*

- set <int> a[u]: là tập các đỉnh kề đỉnh u

- parent[u] = đỉnh cha của đỉnh u trong cây DFS

- num[u] = thứ tự duyệt đến

- low[u] = giá trị num[.] nhỏ nhất của những đỉnh đến được từ nhánh DFS gốc u bằng một cung ngược

- numChild[u] = số nhánh con của cây DFS gốc u

- soCau = số cầu của đồ thị

- soKhop = số khớp của đồ thị

- laKhop[u] = 1 nếu u là khớp, ngược lại là 0.

\*/

void dfs(int u) {

num[u] = low[u] = dem++;

int v;

FORV (it,a[u]) {

v = \*it;

a[v].erase(u); //xóa cung v->u

if (parent[v]==0) {

parent[v] = u;

dfs(v);

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

} else

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

}

}

void process() {

Set(parent,0);

FOR (i,1,n)

if (parent[i]==0) {

parent[i] = -1;

dfs(i);

}

}

void timCau() {

FOR (v,1,n) {

int u = parent[v];

if (u!=-1 && low[v]>=num[v]) {

// u-v la cau

soCau++;

}

}

}

void timKhop() {

//đếm số con của đỉnh u trên cây DFS

FOR (v,1,n) {

int u = parent[v];

if (u!=-1) numChild[u]++;

}

//u là gốc của 1 cây DFS thì là khớp nếu nó có nhiều hơn 1 nhánh con

FOR (u,1,n)

if (parent[u]==-1 && numChild[u]>1) laKhop[u]=1;

// u ko pải là gốc là khớp nếu có đỉnh v là con của u và low[v] >= num[u]

FOR (v,1,n) {

int u = parent[v];

if (u!=-1 && parent[u]!=-1 && low[v]>=num[u]) laKhop[u]=1;

}

//đánh dấu xong các đỉnh là khớp

FOR (u,1,n)

if (laKhop[u]) soKhop++;

}

int main() {

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

//freopen("test.out","w",stdout);

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

int u,v;

FOR (i,1,m) {

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

a[u].insert(v);

a[v].insert(u);

}

process();

timCau();

timKhop();

printf("%d %d",soKhop,soCau);

return 0;

}

# Disjoin – set:

long findset(long i) {

if (parent[i]<0) return i; //Nếu i là gốc

parent[i]=findset(parent[i]); // Heuristic Path Compression

return parent[i];

}

void Union(long x,long y) {

long u,v;

u=findset(x); //Tìm gốc của x

v=findset(y); //Tìm gốc của y

if (u==v) return; //Cùng gốc

//Heuristic Union by rank

if (parent[u]<parent[v]) { //u nhiều con hơn v

parent[u]+=parent[v];

parent[v]=u; //Nối nhánh gốc v vào u

} else {

parent[v]+=parent[u];

parent[u]=v;

}

}

Khởi tạo: Tất cả các parent[]=-1.

# Floyd:

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

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

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

if (c[u][v]>c[u][k]+c[k][v]) {

c[u][v]=c[u][k]+c[k][v]; // Tối ưu

trace[u][v]=trace[u][k]; //Lưu vết

}

# Dijkstra:

vector <pair<int,int> > a[maxn];

pair<int,int> d[maxn];

set <pair<int,int> > se;

// ->first : do dai

// ->second: dinh?

int trace[maxn];

bool Free[maxn];

void init(int u) {

FOR (i,1,n) {

d[i] = mp(oo,i);

Free[i] = 1;

}

d[u] = mp(0,u);

se.clear();

se.insert(d[u]);

trace[u] = -1;

}

void process(int type,int s,int t) {

init(s);

int u,v;

while (!se.empty()) {

u = se.begin() -> se; //lay ra dinh u co trong so be nhat chua co dinh nhan~

Free[u] = 0; //co dinh nhan~ dinh u

se.erase(d[u]);

if (u==t) break;

FO (i,0,a[u].size()) {

v = a[u][i].se; //v la dinh ke u

if (Free[v] && d[v].fi > d[u].fi + a[u][i].fi) { //Dung dinh u toi uu cac dinh (chua co dinh) ke u

se.erase(d[v]);

d[v].fi = d[u].fi + a[u][i].fi;

se.insert(d[v]);

trace[v] = u;

}

}

}

}

# Prim - set:

vector <pair<ll,ll> > a[maxn];

vector <pair<ll,ll> >::iterator it;

ll m,n,ans;

pair<ll,ll> d[maxn];

set <pair<ll,ll> > s;

bool Free[maxn];

void input() {

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

FOR (i,1,m) {

ll u,v,c;

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

a[u].push\_back(make\_pair(c,v));

a[v].push\_back(make\_pair(c,u));

}

}

void init() {

FOR (i,1,n) {

d[i].first=maxmax;

d[i].second=i;

}

d[1].first=0;

FOR (i,1,n) s.insert(d[i]);

}

void prim() {

ll u,v,dd;

while (!s.empty()) {

u=s.begin()->second; //Lay khoi set dinh? co trong so be nhat

dd=s.begin()->first;

Free[u]=1;

ans+=dd;

s.erase(s.begin());

for (it=a[u].begin();it!=a[u].end();it++) {

v=it->second; //canh ke voi u

dd=it->first; //do dai u-->v

if (!Free[v] && d[v].first>dd) {

s.erase(d[v]); //Xoa bo khoi set

d[v].first=dd; //Cap nhat lai d[v]

s.insert(d[v]); //Them vao set d[v] moi

}

}

}

}

# Sắp xếp Topo:

void dfs\_visit(ll u) {

FO (i,0,a[u].size()) { //Thăm các đỉnh kề u

ll v=a[u][i];

if (Free[v]==0) {

Free[v]=1; // Ðánh dấu v đã thăm

dfs\_visit(v); // Thăm v

}

}

//Duyệt xong nhánh DFS gốc u - Thứ tự đỉnh u trong đthị mới là ‘dem’

list[dem]=u;

dem--;

}

void numbering() { // đánh số các đỉnh

FOR (i,1,n) Free[i]=0;

dem=n;

FOR (i,1,n)

if (Free[i]==0) dfs\_visit(i);

}

void topo() {

FOR (i,1,n) {

ll u=list[i]; //Ánh xạ lại đỉnh u

FO (j,0,a[u].size()) { //Duyệt các đỉnh kề u

ll v=a[u][j]; //Ánh xạ lại

d[v]=min(d[v],d[u]+c[u][v] ;

}

}

}

# BIT:

void update(ll x) {

while (x<maxn) {

d[x]++;

x+=(x&(-x));

}

}

ll get(ll x) {

ll res=0;

while (x>0) {

res+=d[x];

x-=(x&(-x));

}

return res;

}

# IT:

 void build\_it\_tree(long i,long l,long r) {

 if (l==r) {

 it[i]=a[l];

 return;

 }

 long mid=(l+r)/2;

 build\_it\_tree(i\*2,l,mid);

 build\_it\_tree(i\*2+1,mid+1,r);



 it[i]=max(it[i\*2],it[i\*2+1]);

 }

 long get(long i,long u,long v,long l,long r) {

 if (l>v || r<u) return -1;

 if (u<=l && v>=r) return it[i];

 long mid=(l+r)/2;

 return max(get(2\*i,u,v,l,mid),get(2\*i+1,u,v,mid+1,r));

 }

# Dãy con tăng dài nhất O(nLogn):

int n;

vector <int> v;

vector <int>:: iterator it;

//v[i] la gia tri phan tu ket thuc cua day con tang dai nhat co do dai la i

//De thay: v[1]<v[2]<...<v[]

//Xet a[i]. Ta can tim vi tri j thuoc v sao cho:

// - j lon nhat

// - v[j] >= a[i]

// - Neu khong tim thay, tuc la moi phan tu trong v deu < a[i] --> v.push\_back(a[i])

// - Neu tim thay o vi tri j, thi tuc la day con tang dai nhat ket thuc la a[i] se co do dai la j. Ta cap nhat lai v[j] = min(v[j],a[i])

int main() {

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

//freopen("test.out","w",stdout);

scanf("%d",&n);

int j,x;

FOR (i,1,n) {

cin >> x;

it = lower\_bound(v.begin(),v.end(),x);

if (it==v.end()) v.pb(x); else \*it = min(\*it,x);

}

cout << v.size();

return 0;

}

# LCA:

int n,k; // number of node & queries

vector <int> adj[maxn]; //danh sach ke

int parent[maxn]; //parent[i] = parent of node number i

bool Free[maxn]; //Free[i]=1 neu dinh i chua tham, nguoc lai la 0

int f[maxn][20]; //f[u][k] = cha thu 2^k cua dinh u

//f[u][0] = cha truc tiep cua node u

//f[u][k] = f[v,k-1] voi v = f[u][k-1]

//Tuc la cha bac 2^k la cha bac 2^(k-1) cua cha bac 2^(k-1)

int h[maxn]; //chieu cao (height) cua node i

void visit(int u,int height) {

Free[u]=0;

h[u]=height;

int v;

FO (i,0,(int)adj[u].size()) {

v = adj[u][i];

if (Free[v]) {

parent[v] = u;

visit(v,height+1);

}

}

}

void initParent() { //tim cha cua cac dinh

SET(Free,1);

parent[1]=0; //dinh 1 la root

visit(1,1);

//FOR (i,1,n) cout << h[i] << " ";

}

void initF() { //tao LCA

FOR (u,1,n) f[u][0]=parent[u]; //f[u][0] = cha truc tiep cua node u

int m = log2(n);

FOR (k,1,m)

FOR (u,1,n) {

int v = f[u][k-1];

f[u][k] = f[v][k-1];

//printf("%d %d %d\n",u,k,f[u][k]);

}

}

void jump(int &u,int height) { //jump from "u" up "height" - nhay? den node cha thu "height" cua node u

int m = log2(n);

FORD (k,m,0) {

int v = f[u][k];

if (h[v] >= height) {

u = v;

}

}

}

int lca(int u,int v) {

if (h[u]<h[v]) swap(u,v);

jump(u,h[v]);

if (u==v) return u;

int u1,v1,m=log2(n);

FORD (i,m,0) {

u1 = f[u][i];

v1 = f[v][i];

if (u1 != v1) {

u = u1;

v = v1;

}

}

return parent[u];

}

int main() {

freopen("RENDEZVOUS.\_c","r",stdin);

freopen("test.out","w",stdout);

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

int u,v;

FO (i,1,n) {

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

adj[u].pb(v);

adj[v].pb(u);

}

//finish input tree

initParent();

initF();

//finish init lca

FOR (i,1,k) {

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

printf("%d\n",lca(u,v));

}

return 0;

}

# Hashing

#define base 101010101

#define maxn 100005

using namespace std;

char c1[maxn],c2[maxn];

LL f1[maxn],f2[maxn],Pow[maxn];

main() {

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

//freopen("test.out","w",stdout);

gets(c1);gets(c2);

int m = strlen(c1), n = strlen(c2);

string s1 = c1; s1 = " " + s1;

string s2 = c2; s2 = " " + s2;

Pow[0]=1;

FOR (i,1,max(m,n)) Pow[i]=Pow[i-1]\*base;

FOR (i,1,m) f1[i] = f1[i-1]\*base+s1[i];

FOR (i,1,m) f2[i] = f2[i-1]\*base+s2[i];

int p;

for (p=min(m,n) ; p>=1 && f1[m]-f1[m-p]\*Pow[p-1]\*base != f2[p];p--);

printf("%s",c1);

FOR (i,p+1,n) putchar(c2[i-1]);

return 0;

}

# Trie:

struct node {

bool is\_end;

int cWord;

node \*child[26];

} \*root;

void init() {

root = new node();

root->is\_end=0;

root->cWord=0;

//cout << root->is\_end << " " << root->cWord;

}

void insert(char s[]) {

node\* current = root;

FO (i,0,strlen(s)) {

if (current->child[s[i]-'a']==NULL)

current->child[s[i]-'a'] = new node();

current = current->child[s[i]-'a'];

}

if (current->is\_end==1) {

current -> cWord++;

printf("%s%d\n",s,current -> cWord);

} else {

current->is\_end=1;

current->cWord=0;

printf("OK\n");

}

}}

# RMQ:

void initRMQ() {

FOR (i,1,n) f[i][0]=a[i];

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

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

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

}

int get(int i,int j) {

int k = log2(j-i+1);

return max(f[i][k],f[j-(1<<k)+1][k]);

}

# Halmiton:

procedure Attempt(i: Integer); *//Thuật toán quay lui*

var

v: Integer;

begin

for v := 2 to n do

if avail[v] and a[x[i - 1], v] then *//Xét các đỉnh v kề x[i - 1] chưa đi qua*

begin

x[i] := v; *//Thử đi sang v*

if i = n then *//Nếu đã qua đủ n đỉnh, đến đỉnh thứ n*

begin

if a[v, 1] then found := True; *//Đỉnh thứ n quay về được 1 thì tìm ra nghiệm*

Exit; *//Thoát luôn*

end

else *//Qua chưa đủ n đỉnh*

begin

avail[v] := False; *//Đánh dấu đỉnh đã qua*

Attempt(i + 1); *//Đi tiếp*

if found then Exit; *//Nếu đã tìm ra nghiệm thì thoát ngay*

avail[v] := True;

end;

end;

end;

# Euler:

stack := (1); *//Ngăn xếp ban đầu chỉ chứa một đỉnh bất kỳ, chẳng hạn đỉnh 1*

repeat

u := Get; *//Đọc phần tử ở đỉnh ngăn xếp*

if ∃(u, v) ∈E then *//Từ u còn đi tiếp được*

begin

Push(v);

E := E – {(u, v)}; *//Xóa cạnh (u, v) khỏi đồ thị*

end;

else *//Từ u không đi đâu được nữa*

begin

u := Pop; *//Lấy u khỏi ngăn xếp*

Output ← u; *//In ra u*

end;

until stack = ∅; *//Lặp tới khi ngăn xếp rỗng*

# C:\Users\John\Desktop\phi-ham-euler.pngMột số công thức toán:

void init() {

for (int i=2;i<=maxn;i++) phi[i] = i;

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

if (phi[i] == i) //i la so nguyen to

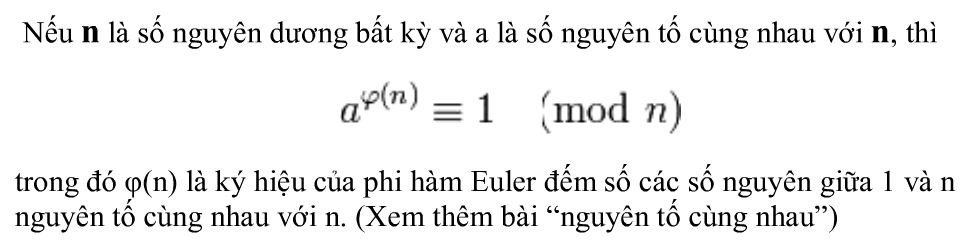
for (int j=i;j<=maxn;j+=i)

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

//FOR (i,1,20) cout << i << " : " << phi[i] << endl;

FOR (i,1,maxn)

sum[i] = sum[i-1]+sqr(phi[i]);

}

Tổ hợp: C(k,n) = C(k-1,n) + C(k-1,n-1) = n! / (k! x (n-k)!)

Chỉnh hợp ko lặp: A(k,n) = n! / (n-k)! = n x (n-1) x … x (n-k+1)

# Một số tính chất của đồ thị:

Điều kiện cần và đủ để một đồ thị vô hướng liên thông có thể định chiều được là mỗi cạnh của đồ thị nằm trên ít nhất một chu trình đơn (hay nói cách khác mọi cạnh của đồ thị đều không phải là cầu).

đồ thị định chiều được (không có cầu) và đồ thị song liên thông (không có khớp).

Một đồ thị vô hướng liên thông có chu trình Euler khi và chỉ khi mọi đỉnh của nó đều có bậc chẵn.

Một đồ thị vô hướng liên thông có đường đi Euler khi và chỉ khi nó có đúng 2 đỉnh bậc lẻ.

Một đồ thị có hướng liên thông yếu có đường đi Euler nhưng không có chu trình Euler nếu tồn tại đúng hai đỉnh s sao cho:

còn tất cả những đỉnh còn lại của đồ thị đều có bán bậc ra bằng bán bậc vào.

Xét đơn đồ thị vô hướng có đỉnh. Nếu mọi đỉnh đều có bậc không nhỏ hơn thì là đồ thị Hamilton.

Xét đơn đồ thị có hướng liên thông mạnh có đỉnh. Nếu trên phiên bản vô hướng của , mọi đỉnh đều có bậc không nhỏ hơn thì là đồ thị Hamilton.

Xét đơn đồ thị vô hướng có đỉnh. Với mọi cặp đỉnh không kề nhau có tổng bậc thì là đồ thị Hamilton.

# Bài toán lập lịch:

## Thuật toán Johnson:

*Bài toán 1:*   
'Mỗi một chi tiết D1,D2,...Dn cần phải được lần lượt gia công trên 2 máy A,B. Thời gian gia công chi tiết Di trên máy A là ai trên máy B là bi (i=1,2..n). Hãy tìm lịch (trình tự gia công) các chi tiết trên hai máy sao cho việc hoàn thành gia công tất cả các chi tiết là sớm nhất'.

***Thuật Johnson:***   
Chia các chi tiết thành 2 nhóm: nhóm N1, gồm các chi tiết Di thoả mãn a1 < b1, tức là min(ai,bi) = ai và nhóm N2 gồm các chi tiết Di thoả mãn ai >bi tức là min(ai,bi)=bi. Các chi tiết Di thoả mãn ai =bi xếp vào nhóm nào cũng được.   
Sắp xếp các chi tiết trong N1 theo chiều tăng của các ai và sắp xếp các chi tiết trong N2 theo chiều giảm của các bi   
Nối N2 vào đuôi N1, dãy thu được (đọc từ trái sang phải) sẽ là lịch gia công.

*Bài toán 2:*   
'Xét bài toán gia công N chi tiết trên 3 máy theo thứ tự A,B,C với bảng thời gian ai, bi, ci (i=1,2,..n) thoả mãn: max bi ≤ min ai hoặc max bi ≤ min ci

***Thuật toán:***   
Lịch gia công tối ưu trên 3 máy sẽ trùng với lịch gia công tối ưu trên 2 máy: máy thứ nhất với thời gian ai + bi và máy thứ hai với thời gian bi + c i.

## Thuật toán More:

*Bài toán:*   
'Có n ôtô đưa vào xưởng sửa chữa được đánh số thứ tự 1,2..,n. Ôtô phải sửa chữa trong thời gian ti và thời điểm phải bàn giao là di. Mỗi thời điểm xưởng chỉ sửa chữa một cái, xưởng sửa chữa không ngừng và thời điểm bắt đầu sửa chữa là 0. Hãy đưa ra một thứ tự sữa chữa sao cho số lượng ôtô đúng hạn là lớn nhất.'

Sắp xếp theo thứ tự tăng dần của thời điểm bàn giao   
Duyệt từ đầu cho đến khi gặp ôtô quá hạn đầu tiên (Giả sử là ôtô thứ k)   
Tìm từ đầu cho đến ôtô thứ k, ôtô nào có ti lớn nhất (Giả sử đó là ôtô thứ m). Nếu ôtô này đã được chuyển một lần rồi thì dừng chương trình, còn nếu không thì ta chuyển ôtô này xuống cuối. Rồi trở lại bước 2.

# Các thành phần song liên thông

#### Các khái niệm và thuật toán

Đồ thị vô hướng liên thông được gọi là đồ thị song liên thông nếu nó không có khớp, tức là việc bỏ đi một đỉnh bất kỳ của đồ thị không ảnh hưởng tới tính liên thông của các đỉnh còn lại. Ta quy ước rằng đồ thị chỉ gồm một đỉnh và không có cạnh nào cũng là một đồ thị song liên thông.

Cần phân biệt hai khái niệm đồ thị định chiều được (không có cầu) và đồ thị song liên thông (không có khớp). Nếu như đồ thị không định chiều được thì *tập đỉnh* của có thể phân hoạch thành các tập con rời nhau để đồ thị hạn chế trên các tập con đó là các đồ thị định chiều được. Còn nếu đồ thị không phải đồ thị song liên thông thì *tập cạnh* của có thể phân hoạch thành các tập con rời nhau để trên mỗi tập con, các cạnh và các đỉnh đầu mút của chúng trở thành một đồ thị song liên thông. Hai thành phần song liên thông có thể có chung một điểm khớp nhưng không có cạnh nào chung

Hình 5‑5. Đồ thị và hai thành phần song liên thông có chung khớp

* BCC.PAS ✓ Liệt kê các thành phần song liên thông

{$MODE OBJFPC}

program BiconnectedComponents;

const

maxN = 1000;

type

TStack = record

x, y: array[1..maxN - 1] of Integer;

top: Integer;

end;

var

a: array[1..maxN, 1..maxN] of Boolean;

number, low: array[1..maxN] of Integer;

stack: TStack;

BCC, old, count, n, u: Integer;

procedure Enter; *//Nhập dữ liệu*

var

i, m, u, v: Integer;

begin

FillChar(a, SizeOf(a), False);

ReadLn(n, m);

for i := 1 to m do

begin

ReadLn(u, v);

a[u, v] := True; a[v, u] := True;

end;

end;

procedure Push(u, v: Integer); *//Đẩy một cung (u, v) vào ngăn xếp*

begin

with stack do

begin

Inc(top);

x[top] := u; y[top] := v;

end;

end;

procedure Pop(var u, v: Integer); *//Lấy một cung (u, v) khỏi ngăn xếp*

begin

with stack do

begin

u := x[top]; v := y[top];

Dec(top);

end;

end;

*//Hàm cực tiểu hóa: Target := min(Target, Value)*

procedure Minimize(var Target: Integer; Value: Integer);

begin

if Value < Target then Target := Value;

end;

procedure DFSVisit(u: Integer); *//Thuật toán tìm kiếm theo chiều sâu*

var

v, p, q: Integer;

begin

Inc(count); number[u] := count;

low[u] := maxN + 1;

for v := 1 to n do

if a[u, v] then *//Xét mọi cạnh (u, v)*

begin

a[v, u] := False; *//Định chiều luôn*

if number[v] <> 0 then *//v đã thăm*

Minimize(low[u], number[v])

else *//v chưa thăm*

begin

Push(u, v); *//Đẩy cung DFS (u, v) vào Stack*

DFSVisit(v); *//Tiếp tục quá trình DFS từ v*

Minimize(low[u], low[v]);

if low[v] >= number[u] then *//Nếu (u, v) là cung chốt*

begin *//Liệt kê thành phần song liên thông với cung chốt (u, v)*

Inc(BCC);

WriteLn('Biconnected component: ', BCC);

repeat

Pop(p, q); *//Lấy một cung DFS (p, q) khỏi Stack*

Write(q, ', '); *//Chỉ in ra một đầu cung, tránh in lặp*

until (p = u) and (q = v); *//Đến khi lấy ra cung (u, v) thì dừng*

WriteLn(u); *//In nốt ra đỉnh u*

end;

end;

end;

end;

begin

Enter;

FillDWord(number[1], n, 0);

stack.top := 0;

count := 0;

BCC := 0;

for u := 1 to n do

if number[u] = 0 then

begin

old := count; *//old là số đỉnh đã thăm*

DFSVisit(u);

if count = old + 1 then *//u là đỉnh cô lập vì lời gọi DFSVisit(u) chỉ thăm 1 đỉnh*

begin

Inc(BCC);

WriteLn('Biconnected component: ', BCC);

WriteLn(u);

end;

end;

end.

# Cặp ghép:

procedure Visit(x ∈X); *//Thuật toán DFS*

begin

for ∀y: (x, y)∈E do *//Quét các Y\_đỉnh kề x*

if avail[y] then *//y chưa thăm, chú ý (x, y) chắc chắn là cạnh chưa ghép*

begin

avail[y] := False; *//Đánh dấu thăm y*

if match[y] = 0 then Found := True *//y chưa ghép, dựng cờ báo tìm thấy đường mở*

else Visit(match[y]); *//y đã ghép, gọi đệ quy tiếp tục DFS*

if Found then *// Ngay khi đường mở được tìm thấy*

begin

match[y] := x; *//Chỉnh lại bộ ghép theo đường mở*

Exit; *//Thoát luôn, lệnh Exit đặt ở đây sẽ thoát cả dây chuyền đệ quy*

end;

end;

end;

begin *//Thuật toán tìm bộ ghép cực đại trên đồ thị hai phía*

«Khởi tạo một bộ ghép bất kỳ, chẳng hạn ∅»;

X\* := «Tập các đỉnh chưa ghép»;

repeat *//Lặp các pha xử lý theo lô*

Old := |X\*|; *//Lưu số đỉnh chưa ghép khi bắt đầu pha*

for ∀y∈Y do avail[y] := True; *//Đánh dấu mọi Y\_đỉnh chưa thăm*

for ∀x∈X\* do

begin

Found := False; *//Cờ báo chưa tìm thấy đường mở*

Visit(x); *//Tìm đường mở bằng DFS*

if Found then X\* := X\* - {x}; *//x đã được ghép, loại bỏ x khỏi X\**

end;

until |X\*| = Old; *//Lặp cho tới khi không thể ghép thêm*

end;

* BMATCH.PAS ✓ Tìm bộ ghép cực đại trên đồ thị hai phía

{$MODE OBJFPC}

program MaximumBipartiteMatching;

const

maxN = 10000;

maxM = 1000000;

var

p, q, m: Integer;

adj: array[1..maxM] of Integer;

link: array[1..maxM] of Integer;

head: array[1..maxN] of Integer;

match: array[1..maxN] of Integer;

avail: array[1..maxN] of Boolean;

List: array[1..maxN] of Integer;

nList: Integer;

procedure Enter; *//Nhập dữ liệu*

var

i: Integer;

x, y: Integer;

begin

ReadLn(p, q, m);

FillChar(head[1], p \* SizeOf(head[1]), 0);

for i := 1 to m do

begin

ReadLn(x, y); *//Đọc một cạnh (x, y), đưa y vào danh sách kề của x*

adj[i] := y;

link[i] := head[x]; head[x] := i;

end;

end;

procedure Init; *//Khởi tạo bộ ghép rỗng*

var

i: Integer;

begin

FillChar(match[1], q \* SizeOf(match[1]), 0);

for i := 1 to p do List[i] := i; *//Mảng List chứa nList X\_đỉnh chưa ghép*

nList := p;

end;

procedure SuccessiveAugmentingPaths;

var

Found: Boolean;

Old, i: Integer;

procedure Visit(x: Integer); *//Thuật toán DFS từ x∈X*

var

i: Integer;

y: Integer;

begin

i := head[x]; *//Từ đầu danh sách kề của x*

while i <> 0 do

begin

y := adj[i]; *//Xét một đỉnh y∈Y kề x*

if avail[y] then *//y chưa thăm, hiển nhiên (x, y) là cạnh chưa ghép*

begin

avail[y] := False; *//Đánh dấu thăm y*

if match[y] = 0 then Found := True *//y chưa ghép thì báo hiệu tìm thấy đường mở*

else Visit(match[y]); *//Thăm luôn match[y]∈X (thăm liền 2 bước)*

if Found then *//Tìm thấy đường mở*

begin

match[y] := x; *//Chỉnh lại bộ ghép*

Exit; *//Thoát dây chuyền đệ quy*

end;

end;

i := link[i]; *//Chuyển sang đỉnh kế tiếp trong danh sách các đỉnh kề x*

end;

end;

begin

repeat

Old := nList; *//Lưu lại số X\_đỉnh chưa ghép*

FillChar(avail[1], q \* SizeOf(avail[1]), True);

for i := nList downto 1 do

begin

Found := False;

Visit(List[i]); *//Cố ghép List[i]*

if Found then *//Nếu ghép được*

begin *//Xóa List[i] khỏi danh sách các X\_đỉnh chưa ghép*

List[i] := List[nList];

Dec(nList);

end;

end;

until Old = nList; *//Không thể ghép thêm X\_đỉnh nào nữa*

end;

procedure PrintResult; *//In kết quả*

var

j, k: Integer;

begin

k := 0;

for j := 1 to q do

if match[j] <> 0 then

begin

Inc(k);

WriteLn(k, ': x[', match[j], '] - y[', j, ']');

end;

end;

begin

Enter;

Init;

SuccessiveAugmentingPath;

PrintResult;

end.

# Biểu thức:

Tính giá trị bt hậu tố:

1. {$MODE OBJFPC}
2. program CalculatingRPNExpression;
3. type
4. TStackNode = record *// Ngăn xếp được cài đặt bằng danh sách móc nối kiểu LIFO*
5. value: Real;
6. link: Pointer;
7. end;
8. PStackNode = ^TStackNode;
9. var
10. RPN: AnsiString;
11. top: PStackNode;
12. procedure Push(const v: Real); *//Đẩy một toán hạng là số thực v vào ngăn xếp*
13. var
14. p: PStackNode;
15. begin
16. New(p);
17. p^.value := v;
18. p^.link := top;
19. top := p;
20. end;
21. function Pop: Real; *//Lấy một toán hạng ra khỏi ngăn xếp*
22. var
23. p: PStackNode;
24. begin
25. Result := top^.value;
26. p := top^.link;
27. Dispose(top);
28. top := p;
29. end;
30. procedure ProcessToken(const token: AnsiString); *//Xử lý một phần tử trong biểu thức RPN*
31. var
32. x, y: Real;
33. err: Integer;
34. begin
35. if token[1] in ['+', '-', '\*', '/'] then *//Nếu phần tử token là toán tử*
36. begin *//Lấy ra hai phần tử khỏi ngăn xếp, thực hiện toán tử và đẩy giá trị vào ngăn xếp*
37. y := Pop; x := Pop;
38. case token[1] of
39. '+': Push(x + y);
40. '-': Push(x - y);
41. '\*': Push(x \* y);
42. '/': Push(x / y);
43. end;
44. end
45. else *//Nếu phần tử token là toán hạng thì đẩy giá trị của nó vào ngăn xếp*
46. begin
47. Val(token, x, err);
48. Push(x);
49. end;
50. end;
51. procedure Parsing; *//Xử lý biểu thức RPN*
52. var
53. i, j: Integer;
54. begin
55. j := 0; *//j là vị trí đã xử lý xong*
56. for i := 1 to Length(RPN) do *//Quét biểu thức từ trái sang phải*
57. if RPN[i] in [' ', '+', '-', '\*', '/'] then *//Nếu gặp toán tử hoặc dấu phân cách toán hạng*
58. begin
59. if i > j + 1 then *//Trước vị trí i có một toán hạng chưa xử lý*
60. ProcessToken(Copy(RPN, j + 1, i - j - 1)); *//Xử lý toán hạng đó*
61. if RPN[i] in ['+', '-', '\*', '/'] then *//Nếu vị trí i chứa toán tử*
62. ProcessToken(RPN[i]); *//Xử lý toán tử đó*
63. j := i; *//Đã xử lý xong đến vị trí i*
64. end;
65. if j < Length(RPN) then *//Trường hợp có một toán hạng còn sót lại (biểu thức chỉ có 1 toán hạng)*
66. ProcessToken(Copy(RPN, j + 1, Length(RPN) - j)); *//Xử lý nốt*
67. end;
68. begin
69. ReadLn(RPN); *//Đọc biểu thức*
70. top := nil; *//Khởi tạo ngăn xếp rỗng,*
71. Parsing; *//Xử lý*
72. Write(Pop:0:4); *//Lấy ra phần tử duy nhất còn lại trong ngăn xếp và in ra kết quả.*
73. end.

2. Chuyển từ trung tố sang hậu tố:

Stack := Ø;

for «phần tử token đọc được từ biểu thức trung tố» do

case token of *//token có thể là toán hạng, toán tử, hoặc dấu ngoặc được đọc lần lượt theo thứ tự từ trái qua phải*

'(': Push(token); *//Gặp dấu ngoặc mở thì đẩy vào ngăn xếp*

')': *//Gặp dấu ngoặc đóng thì lấy ra và hiển thị các phần tử trong ngăn xếp cho tới khi lấy tới dấu ngoặc mở*

repeat

x := Pop;

if x ≠ '(' then Output ← x;

until x = '(';

'+', '-', '\*', '/': *//Gặp toán tử*

begin

*//Chừng nào đỉnh ngăn xếp có phần tử với mức ưu tiên lớn hơn hay bằng token, lấy phần tử đó ra và hiển thị*

while (Stack ≠ Ø) and (Priority(token) ≤ Priority(Get)) do

Output ← Pop;

Push(token); *//Đẩy toán tử token vào ngăn xếp*

end;

else *//Gặp toán hạng thì hiển thị luôn*

Output ← token;

end;

*//Khi đọc xong biểu thức, lấy ra và hiển thị tất cả các phần tử còn lại trong ngăn xếp*

while Stack ≠ Ø do

Output ← Pop