Có thể chưa đủ hết kiế nthuwsc đâu mọi người ạ nhưng các bạn cứ tham khảo nhá , có nhiều cái hay lắm , code siêu ngắn đúng form luôn ( trr này ko phải lớp thầy thỏa nhá)

## Contents

**1.1 : Viết hàm dfs(int u) mô tả thuật toán duyệt theo chiều sâu…:**

Quá trình tìm kiếm ưu tiên "chiều sâu" hơn "chiều rộng"
Đi xuống sâu nhất có thể trước khi quay lại
DFS($u$){//$u$ là đỉnh bắt đầu duyệt
    <Thăm đỉnh $u$>;//duyệt đỉnh $u$
    $chuaxet[u]$ = fa$lse$; //xác nhận đỉnh $u$ đã duyệt
    for($v \in Ke(u)$){
        if($chuaxet[v]$)//nếu $v$ chưa được duyệt
        DFS($v$);//duyệt theo chiều sâu từ $v$
    }
}
C2 : stack :
DFS($u$){
**Bước 1: Khởi tạo**
    $stack = \emptyset$;  //khởi tạo $stack$ là $\emptyset$
    p$ush(stack, u)$;//đưa đỉnh $u$ vào stack
    <Thăm đỉnh $u$>;//duyệt đỉnh $u$
    $chuaxet[u] = false$;//xác nhận đã duyệt $u$
**Bước 2: Lặp**
while($stack \neq \emptyset$){
    $s$ = p$op(stack)$;//lấy 1 đỉnh ở đầu stack
    for($t \in Ke(s)$){
        if($chuaxet[t]$){//nếu chưa duyệt $t$
            <Thăm đỉnh $t$>;//duyệt đỉnh $t$
            $chuaxet[t] = false$;//$t$ đã được duyệt
            p$ush(stack, s)$;//đưa $s$ vào stack
            p$ush(stack, t)$;//đưa $t$ vào stack
            break;//chỉ lấy một đỉnh $t$
        }
    }
}

Bước 3: Trả lại kết quả:

return <tập đỉnh đã duyệt>;

### Code:quay lui

```
#include<bits/stdc++.h>

using namespace std;
int V,E,u,check[1005];
int a[1005][1005];
void dfs(int u){
   cout<<u<<' ';
   check[u]= 1;
   for(int v=1;v<=V;v++)
      if(a[u][v]==1&&check[v]==0) dfs(v);
}
void initwsolve(){
     cin>>V>>E>>u;
     memset(a,0,sizeof(a));
     memset(check,0,sizeof(check));
/*1*/ for(int i=1;i<=E;i++){
          int x,y;cin>>x>>y;
          a[x][y]=1;
          a[y][x]=1;
     }
     dfs(u);
     cout<<endl;
}
int main(){
     int t;cin>>t;
     while(t--) initwsolve();
}
```

### code :stack

```
#include<bits/stdc++.h>
using namespace std;
int V,E,u,check[1005];
int a[1005][1005];
void dfs(int u){
   stack<int> nganxep;
   cout<<u<<' ';
```

```
    check[u]=1;
    nganxep.push(u);
    while(!nganxep.empty()){
        int s=nganxep.top();
        nganxep.pop();
        for(int t=1;t<=V;t++){
            if(a[s][t]==1&&check[t]==0){
                cout<<t<<' ';
                check[t]=1;
                nganxep.push(s);
                nganxep.push(t);
                break;
            }
        }
    }
}
void initwsolve(){
    cin>>V>>E>>u;
    memset(a,0,sizeof(a));
    memset(check,0,sizeof(check));
/*1*/ for(int i=1;i<=E;i++){
            int x,y;cin>>x>>y;
            a[x][y]=1;
            a[y][x]=1;
    }
    dfs(u);
    cout<<endl;
}
int main(){
    int t;cin>>t;
    while(t--) initwsolve();
}
```



**1.2 : Viết hàm bfs(int u) mô tả thuật toán duyệt theo chiều sâu…:**

Giả mã :

Thuật toán

BFS($u$){

Bước 1: Khởi tạo

$queue = \emptyset$;

p$ush(queue, u)$;

$chuaxet[u] = false$;

Bước 2: Lặp

while($queue \neq \emptyset$){

$s$ = p$op(queue)$;

<Thăm đỉnh $s$>;

for($t \in Ke(s)$){

if($chuaxet[t]$){

p$ush(queue, t)$;

$chuaxet[t] = false$;

}

}

}

Bước 3: Trả lại kết quả

return <tập đỉnh đã duyệt>;

}

Code :

```
#include<bits/stdc++.h>
using namespace std;
int V,E,u,check[1005];
int a[1005][1005];
void bfs(int u){
   queue<int> q;
   check[u]=1;
   q.push(u);
   while(q.size()){
      int s=q.front();
      cout<<s<<' ';
      q.pop();
      for(int t=1;t<=V;t++){
         if(a[s][t]==1&&check[t]==0){
            check[t]=1;
            q.push(t);
```

```
        }
      }

   }
}
void initwsolve(){
    cin>>V>>E>>u;
    memset(a,0,sizeof(a));
    memset(check,0,sizeof(check));
/*1*/ for(int i=1;i<=E;i++){
        int x,y;cin>>x>>y;
        a[x][y]=1;
        a[y][x]=1;
    }
    bfs(u);
    cout<<endl;
}
int main(){
    int t;cin>>t;
    while(t--) initwsolve();
}
```

**Xác định số thành phần liên thông**

Thuật toán

```
Duyet-TPLT(){ //duyệt thành phần liên thông
    Bước 1: Khởi tạo
        soTPLT = 0; //khởi tạo số thành phần liên thông bằng 0
    Bước 2: Lặp
        for(u ∈ V){ //lặp trên tập đỉnh
            if( chuaxet[u]){
                soTPLT = soTPLT + 1;//ghi nhận số TPLT
                BFS(u); // có thể gọi DFS u
                <Ghi nhận các đỉnh thuộc TPLT>;
            }
        }
Bước 3: Trả lại kết quả
```

```
        return <các TPLT>;
}
```

### Code: dfs

```
#include<bits/stdc++.h>
#include<bits/stdc++.h>
using namespace std;
int V,E,check[1005];
int a[1005][1005];
void dfs(int u){
   check[u]= 1;
   for(int v=1;v<=V;v++)
      if(a[u][v]==1&&check[v]==0) dfs(v);
}
void initwsolve(){
/*1init*/int res=0;
     cin>>V>>E;
     memset(a,0,sizeof(a));
     memset(check,0,sizeof(check));
     for(int i=1;i<=E;i++){
          int x,y;cin>>x>>y;
           a[x][y]=1;
           a[y][x]=1;
     }
/*2*/ for(int i=1;i<=V;i++){
          if(check[i]==0){
               dfs(i);res++;
          }
     }cout<<res<<endl;
}
int main(){
     int t;cin>>t;
     while(t--) initwsolve();
}
```

### Code bfs:

```
#include<bits/stdc++.h>
#include<bits/stdc++.h>
using namespace std;
int V,E,check[1005];
```

```
int a[1005][1005];
void bfs(int u){
    queue<int> q;
    check[u]=1;
    q.push(u);
    while(q.size()){
        int s=q.front();
        q.pop();
        for(int t=1;t<=V;t++){
            if(a[s][t]==1&&check[t]==0){
                check[t]=1;
                q.push(t);
            }
        }

    }
}

void initwsolve(){
/*1init*/int res=0;
      cin>>V>>E;
      memset(a,0,sizeof(a));
      memset(check,0,sizeof(check));
      for(int i=1;i<=E;i++){
            int x,y;cin>>x>>y;
             a[x][y]=1;
             a[y][x]=1;
      }
/*2*/ for(int i=1;i<=V;i++){
            if(check[i]==0){
                bfs(i);res++;
            }
      }cout<<res<<endl;
}
int main(){
      int t;cin>>t;
      while(t--) initwsolve();
}
```

**Đường đi từ dỉnh s đến đỉnh t**

**Code stack :**

```
DFS(s){
        Bước 1: Khởi tạo
        stack = ∅;  push(stack, s);    chuaxet[s] = false;
Bước 2: Lặp
while(stack ≠ ∅){
        u = pop(stack);   // lấy 1 đỉnh ở stack
        for(v ∈ Ke(u)){
            if(chuaxet[v]){    // nếu chưa duyệt v
                chuaxet[v] = false;      // v đã duyệt
                push(stack, u);    // đưa u vào stack
                push(stack, v);    // đưa v vào stack
                truoc[v] = u;      // lưu truoc[v] là u
                break;// chỉ lấy 1 đỉnh
            }
        }
}
Bước 3: Trả lại kết quả
return <tập đỉnh đã duyệt>;
```

Thuật toán dùng BFS - đường đi ít cạnh nhất:

```
BFS(s){
    Bước 1: Khởi tạo
        queue = ∅;  push(queue, s);    chuaxet[s] = false;
    Bước 2: Lặp:
    while(queue ≠ ∅){
        u = pop(queue);   // lấy 1 đỉnh ở queue
        for(v ∈ Ke(u)){
            if(chuaxet[v]){    // nếu chưa duyệt v
                push(queue, v);    // đưa v vào queue
                chuaxet[v] = false;      // v đã duyệt
                truoc[v] = u;      // lưu truoc[v] là u
            }
        }
    }
Bước 3: Trả lại kết quả return <tập đỉnh đã duyệt>;

Ghi_Nhan_Duong_Di(s, t){
    if(truoc[t] == 0){
        <Không có đường đi từ s tới t>;
```

```
        }
        else{
                <Đưa ra đỉnh t>;    // đưa ra đỉnh t trước
                u = truoc[t];       // u là đỉnh trước khi đến được t
                while(u ≠ s){
                        <Đưa ra đỉnh u>;
                        u = truoc[u];       // lần ngược lại đỉnh trước u
                }
                <Đưa ra đỉnh s>;
        }
 }// in ngược lại lớn đến nhỏ
```

## Code dfs quay lui

```
#include<bits/stdc++.h>
using namespace std;
int V,E,s,t,check[1005],b[1005];//b: truoc s=> = b[t];
int a[1005][1005];
void dfs(int u){//2
    check[u]=1;
/*3*/ for(int v=1;v<=V;v++){
        if(a[u][v]==1&&check[v]==0) {
            b[v]=u;
            dfs(v);
        }
    }
}
void initwsolve(){
    cin>>V>>E>>s>>t;
    memset(a,0,sizeof(a));
    memset(check,0,sizeof(check));
/*1*/  for(int i=1;i<=E;i++){
        int x,y;cin>>x>>y;
         a[x][y]=1;
         a[y][x]=1;
    }
    dfs(s);
    if(check[t]==0) cout<<-1<<endl;
/*4*/ else {
        vector <int> res;
```

```
        int i=t;cout<<s<<' ';
        while(i!=s){res.push_back(i); i=b[i];}
        for(int i=res.size()-1;i>=0;i--) cout<<res[i]<<' ';
        cout<<endl;
    }
}
int main(){
    int t;cin>>t;
    while(t--) initwsolve();
}
```

**Code bfs:**

```
#include<bits/stdc++.h>
using namespace std;
int V,E,s,t,check[1005],b[1005];//b: truoc s=> = b[t];
int a[1005][1005];
void bfs(int u){
   queue<int> q;
   check[u]=1;
   q.push(u);
   while(q.size()){
      int s=q.front();
      q.pop();
      for(int t=1;t<=V;t++){
         if(a[s][t]==1&&check[t]==0){
            check[t]=1;
            b[t]=s;
            q.push(t);
         }
      }

   }
}

void initwsolve(){
    cin>>V>>E>>s>>t;
    memset(a,0,sizeof(a));
    memset(check,0,sizeof(check));
    memset(b,0,sizeof(b));//note
/*1*/  for(int i=1;i<=E;i++){
```

```
        int x,y;cin>>x>>y;
         a[x][y]=1;
         a[y][x]=1;
    }
    bfs(s);
    if(check[t]==0) cout<<-1<<endl;
/*4*/ else {
        vector <int> res;
        int i=t;cout<<s<<' ';
        while(i!=s){res.push_back(i); i=b[i];}
        for(int i=res.size()-1;i>=0;i--) cout<<res[i]<<' ';
        cout<<endl;
    }
}
int main(){
    int t;cin>>t;
    while(t--) initwsolve();
}
```

**Kiểm tra đỉnh trụ:**

Đỉnh $u \in V$ của đồ thị vô hướng $G = < V, E >$ được gọi là trụ nếu loại bỏ đỉnh $u$ cùng với các cạnh nối với $u$ làm tăng thành phần liên thông của $G$. Cho trước đồ thị vô hướng (liên thông) $G = <V, E >$, tìm các đỉnh trụ của $G$?

Duyet_Tru($G = < V, E >$){
    $ReInit(\ )$; // $\forall u \in V$: $chuaxet[u] = true$;
    for($u \in V$){// lấy mỗi đỉnh $u$
        $chuaxet[u] = false$;// cấm BFS hoặc DFS duyệt $u$
        if($\boldsymbol{BFS(v)} \neq V\backslash\{u\}$)// có thể kiểm tra $\boldsymbol{DFS(v)} \neq V \setminus \{u\}$
            < $u$ là trụ>;// với $\boldsymbol{v}$ bất kỳ, $\boldsymbol{v} \neq$ u
    $ReInit()$;// khởi tạo lại mảng $chuaxet[\ ]$
    }
}

**Code Dfs:**

```
#include<bits/stdc++.h>
using namespace std;
int V,E,check[1005];//b: truoc s=> = b[t];
int a[1005][1005];
```

```
void dfs(int u){
   check[u]= 1;
   for(int v=1;v<=V;v++)
      if(a[u][v]==1&&check[v]==0) dfs(v);
}


void initwsolve(){
     cin>>V>>E;
     int res=0;
     memset(a,0,sizeof(a));
     memset(check,0,sizeof(check));
/*1*/  for(int i=1;i<=E;i++){
          int x,y;cin>>x>>y;
           a[x][y]=1;
           a[y][x]=1;
     }
     for(int i=1;i<=V;i++){
          memset(check,0,sizeof(check));
          check[i]=1;res=0;
          for(int j=1;j<=V;j++){
               if(check[j]==0){dfs(j);res++;}
          }
          if(res>1) cout<<i<<' ';
     }
     cout<<endl;
}
int main(){
     int t;cin>>t;
     while(t--) initwsolve();
}
```

**Kiểm tra tính liên thông mạnh :**

bool Strongly_Connected($G = < V, E >$){ // kt tính liên thông mạnh của G

$ReInit( )$;  // $\forall u \in V$: $chuaxet[u] = true$;

for($u \in V$){  // lặp trên tập đỉnh

if($\boldsymbol{BFS(u)} \neq V$)  // có thể kiểm tra $\boldsymbol{DFS(u)} \neq V$

return false;  // đồ thị không liên thông mạnh

else

$ReInit()$;  // khởi tạo lại mảng $chuaxet$[]

```
        }
        return true;   // đồ thị liên thông mạnh
}
```

### Code bfs vs dfs

```
#include<bits/stdc++.h>
using namespace std;
int V,E,check[1002];
int a[1002][1002];
void bfs(int u){
    queue<int> q;
    check[u]=1;
    q.push(u);
    while(q.size()){
        int s=q.front();
        q.pop();
        for(int t=1;t<=V;t++){
            if(a[s][t]==1&&check[t]==0){
                check[t]=1;
                q.push(t);
            }
        }

    }
}
void dfs(int u){
    check[u]= 1;
    for(int v=1;v<=V;v++)
        if(a[u][v]==1&&check[v]==0) dfs(v);
}
void initwsolve(){
/*init*/cin>>V>>E;
      memset(a,0,sizeof(a));
      memset(check,0,sizeof(check));
      for(int i=1;i<=E;i++){
            int x,y;cin>>x>>y;
             a[x][y]=1;
      }
/*2*/ for(int i=1;i<=V;i++){
            dfs(i);
            for(int j=1;j<=V;j++)
```

```
            if(check[j]==0){
                cout<<"NO"<<endl; return;
            }
            memset(check,0,sizeof(check));
        } cout<<"YES"<<endl;
}
int main(){
    int t;cin>>t;
    while(t--) initwsolve();
}
```

**Kiểm tra cạnh cầu: codeptit dùng ma trận kề mới đúng**

Cạnh $e \in E$ của đồ thị vô hướng $G = < V, E >$ được gọi là cạnh cầu nếu loại $e$ làm tăng thành phần liên thông của $G$. Cho trước đồ thị vô hướng (liên thông) $G = < V, E >$, tìm các cạnh cầu của $G$?

```
Duyet_Cau(G = < V, E >){
        ReInit( ); // ∀u ∈ V: chuaxet[u] = true;
        for(e ∈ E){// lấy mỗi cạnh của đồ thị
                E = E \ {e};// loại bỏ cạnh e ra khỏi đồ thị
                if(BFS(1) ≠ V)< e là cầu>;// có thể kiểm tra DFS(1) ≠ V
                E = E ∪ {e};// hoàn trả lại cạnh e
                ReInit();// khởi tạo lại mảng chuaxet[ ]
        }
}
```

**dfs:**

```
#include<bits/stdc++.h>
#include<bits/stdc++.h>
#define ii pair<int,int>
using namespace std;
int V,E,check[1001];
int a[1001][1001];
vector<pair<int,int> >canh;
void dfs(int u){
   check[u]= 1;
   for(int v=1;v<=V;v++)
      if(a[u][v]==1&&check[v]==0) dfs(v);
```

```
}
void initwsolve(){
    cin>>V>>E;
    canh.clear();
    memset(a,0,sizeof(a));
    memset(check,0,sizeof(check));
    for(int i=1;i<=E;i++){
        int x,y;cin>>x>>y;
        a[x][y]=1;
        a[y][x]=1;
        canh.push_back(ii(x,y) );
    }
    for(int i=0;i<E;i++){
            a[canh[i].first][canh[i].second]=0;
            int res=0;
            memset(check,0,sizeof(check));
            for(int k=1;k<=V;k++){
               if(check[k]==0)res++;dfs(k);
            }
            if(res>1)
              cout<<canh[i].first<<' '<<canh[i].second<<' ';
            a[canh[i].first][canh[i].second]=1;
   }
    cout<<endl;
}
int main(){
    int t;cin>>t;
    while(t--) initwsolve();
}
```

## EULER

### Điều kiện cần và đủ để đồ thị là Euler

**Với đồ thị vô hướng**

Đồ thị vô hướng liên thông G =< V, E > là đồ thị Euler khi và chỉ khi mọi đỉnh của G đều có bậc chẵn

**Với đồ thị có hướng**

Đồ thị có hướng liên thông yếu G =< V, E > là đồ thị Euler khi và

chỉ khi tất cả các đỉnh của nó đều có bán bậc ra bằng bán bậc vào
(điều này làm cho đồ thị là liên thông mạnh)

## Chứng minh đồ thị là Euler

**Với đồ thị vô hướng**

-Kiểm tra đồ thị có liên thông hay không?
Kiểm tra DFS(u) = V hoặc BFS(u) = V?
-Kiểm tra bậc của tất cả cả đỉnh có phải số chẵn hay không?
Với ma trận kề, tổng các phần tử của hàng u (cột u) là bậc của đỉnh u

**Với đồ thị có hướng**

-Kiểm tra đồ thị có liên thông yếu hay không?
+Kiểm tra đồ thị vô hướng tương ứng là liên thông, hoặc
+Kiểm tra nếu tồn tại đỉnh u ∈ V để DFS(u) = V hoặc BFS (u) = V?
-Kiểm tra tất cả các đỉnh có thỏa mãn bán bậc ra bằng bán bậc vào hay không?
Với ma trận kề, bán bậc ra của đỉnh u là deg+(u) là số các số 1 của hàng u, bán bậc vào của đỉnh u là deg−(u) là số các số 1 của cột u

## Điều kiện cần và đủ để đồ thị là nửa Euler

**Với đồ thị vô hướng**

-Đồ thị vô hướng liên thông G =< V, E > là đồ thị nửa Euler khi và chỉ khi G có 0 hoặc 2 đỉnh bậc lẻ
+ G có 2 đỉnh bậc lẻ: đường đi Euler xuất phát tại một đỉnh bậc lẻ và kết thúc tại đỉnh bậc lẻ còn lại
+ G có 0 đỉnh bậc lẻ: G chính là đồ thị Euler

**Với đồ thị có hướng**

Đồ thị có hướng liên thông yếu G =< V, E > là đồ thị nửa Euler khi và chỉ khi:
+ Tồn tại đúng hai đỉnh u, v ∈ V sao cho
deg+(u) − deg−(u) = deg−(v) − deg+(v) = 1
+ Các đỉnh s ≠ u, s ≠ v còn lại có deg+(s) = deg−(s)
+Đường đi Euler sẽ xuất phát tại đỉnh u và kết thúc tại đỉnh v

## Chứng minh đồ thị là nửa Euler

**Với đồ thị vô hướng**

-Chứng tỏ đồ thị đã cho liên thông

Sử dụng hai thủ tục DFS(u) hoặc BFS(u)

-Có 0 hoặc 2 đỉnh bậc lẻ

Sử dụng tính chất của các phương pháp biểu diễn đồ thị để tìm ra bậc của mỗi đỉnh

**Với đồ thị có hướng**

-Chứng tỏ đồ thị đã cho liên thông yếu

Sử dụng hai thủ tục DFS(u) hoặc BFS(u)

-Có hai đỉnh u, v ∈ V thỏa mãn

deg+(u) − deg−(u) = deg−(v) − deg+(v) = 1

Các đỉnh s ≠ u, s ≠ v còn lại có deg+(s) = deg−(s)

## Thuật toán tìm đường đi Euler

Thuật toán tìm đường đi Euler gần giống hệt thuật toán tìm chu trình Euler

-Tìm chu trình Euler

Đầu vào thuật toán là đỉnh u ∈ V bất kỳ

-Tìm đường đi Euler

+Đồ thị vô hướng: Đầu vào thuật toán là đỉnh u ∈ V có bậc lẻ đầu tiên (trường hợp có 0 bậc lẻ thì dùng đỉnh bất kỳ)

+Đồ thị có hướng : Đầu vào thuật toán là đỉnh u ∈ V thỏa mãn deg+(u) − deg−(u) = 1

## Thuật toán tìm chu trình Euler

Euler-Cycle($u$){

Bước 1: Khởi tạo

$stack = \emptyset$;//khởi tạo $stack$ là $\emptyset$

$CE = \emptyset$; //khởi tạo mảng $CE$ là $\emptyset$

$push(stack, u)$;//đưa đỉnh $u$ vào ngăn xếp

Bước 2: Lặp

while($stack \neq \emptyset$){

$s = \boldsymbol{get}(stack)$;//lấy đỉnh ở đầu ngăn xếp

if($Ke(s) \neq \emptyset$){

$t$=<đỉnh đầu tiên trong $Ke(s)$>;

push($stack$, $t$); //đưa đỉnh $t$ vào ngăn xếp
$E = E\backslash\{(s, t)\}$;//loại bỏ cạnh $(s, t)$;
}
else{
$s$ = pop($stack$); //loại bỏ s khỏi ngăn xếp
$s \Rightarrow CE$; //đưa s sang $CE$
}
}
Bước 3: Trả lại kết quả
<lật ngược lại các đỉnh trong $CE$ ta được chu trình Euler>;
}

## Cây khung:

 Bài toán:
Cho đồ thị vô hướng $G = < V, E >$.
Hãy xây dựng một cây khung của đồ thị bắt đầu tại đỉnh $u \in V$.
 Cách làm:
o Sử dụng thuật toán duyệt DFS hoặc BFS;
o Mỗi khi ta đến được đỉnh $v$ (tức là $chuaxet[v] = true$) từ đỉnh $u$ thì cạnh $(u, v)$ được kết nạp vào cây khung.
thuật toán :
Tree-DFS($u$){
c$huaxet[u] = false$; // đánh dấu đỉnh $u$ đã duyệt
for($v \in Ke(u)$){
if($chuaxet[v]$){ // nếu $v$ chưa được duyệt
$T = T \cup \{u, v\}$; // hợp cạnh (u,$v$) vào cây khung T
Tree-DFS($v$); // duyệt theo chiều sâu từ $v$
}
}
}
.

root = <đỉnh $u$ bất kỳ $\in V$>; // Lấy một đỉnh bất kỳ làm gốc
Tree-Graph-DFS(root){
for($u \in V$)
$chuaxet[u] = true$; // Khởi tạo mọi đỉnh: chưa xét
$T = \emptyset$; // Cây ban đầu chưa có cạnh nào
Tree-DFS(root); // Gọi hàm tạo cây khung từ 1 đỉnh
if($T < n - 1$)
<đồ thị không liên thông>;
else
<ghi nhận tập cạnh của cây khung $T$>; }
.

## Thuật toán Kruskal (1/2)

```
Kruskal( ){
    Bước 1 (khởi tạo):
        T = ∅; // Ban đầu tập cạnh cây khung là rỗng
        d(H) = 0; // Ban đầu độ dài cây khung là 0
    Bước 2 (sắp xếp):
        <sắp xếp các cạnh đồ thị theo thứ tự tăng dần của trọng số>;
    Bước 3 (lặp):
        while(|T| < n − 1 && E ≠ ∅ ){
        e = <Cạnh có độ dài nhỏ nhất>;
        E = E \ {e}; // Loại cạnh e ra khỏi tập cạnh
        if(T ∪ {e} không tạo nên chu trình){
            T = T ∪ {e}; // Đưa e vào cây khung
            d(H) = d(H) + d(e); // Cập nhật độ dài cây khung
        }
    }
Bước 4 (trả lại kết quả):
    if(|T| < n − 1) <Đồ thị không liên thông>;
    else return (T, d(H));
}
```

Vd: 3.7:
0 4 1 1 2 9 0 5 4 7
4 0 2 0 9 1 5 0 6 0

1 2 0 7 0 6 6 1 1 9
1 0 7 0 1 7 0 6 0 0
2 9 0 1 0 3 4 3 1 2
9 1 6 7 3 0 3 1 1 5
0 5 6 0 4 3 0 4 5 0
5 0 1 6 3 1 4 0 4 2
4 6 1 0 1 1 5 4 0 4
7 0 9 0 2 5 0 2 4 0
Giai :
dH= 12
1 3
1 4
2 6
3 8
3 9
4 5
6 8
5 10
6 7

## Thuật toán Prim (1/2):

```
Prim(s){
    Bước 1 (khởi tạo):
        V_H = {s}; // Ban đầu V_H chỉ chứa s
        V = V \ {s}; // Loại s ra khỏi V
        T = Ø; // Cây khung ban đầu chưa có cạnh nào
        d(H) = 0; // Độ dài cây khung ban đầu bằng 0
    Bước 2 (lặp):
        while(V ≠ Ø){
            e = (u, v); // Cạnh độ dài nhỏ nhất với u ∈ V, v ∈ V_H
            if(không tìm được e)
                return <Đồ thị không liên thông>;
            T = T ∪ {e}; // Đưa e vào cây khung
            d(H) = d(H) + d(e); // Cập nhật độ dài cây khung
```

$V_H = V_H \cup \{u\}$; // Đưa u vào VH
V = V \ u; // Loại u ra khỏi V
}
Bước 3 (trả lại kết quả):
return (T, d(H));
}

Vd :3.8
0 4 1 1 2 9 0 5 4 7
4 0 2 0 9 1 5 0 6 0
1 2 0 7 0 6 6 1 1 9
1 0 7 0 1 7 0 6 0 0
2 9 0 1 0 3 4 3 1 2
9 1 6 7 3 0 3 1 1 5
0 5 6 0 4 3 0 4 5 0
5 0 1 6 3 1 4 0 4 2
4 6 1 0 1 1 5 4 0 4
7 0 9 0 2 5 0 2 4 0
Giai :
dH = 12
1 3
1 4
3 8
3 9
4 5
8 6
6 2
5 10
6 7