

## CHƯƠNG 5: ĐỒ THỊ

# MỤC TIÊU

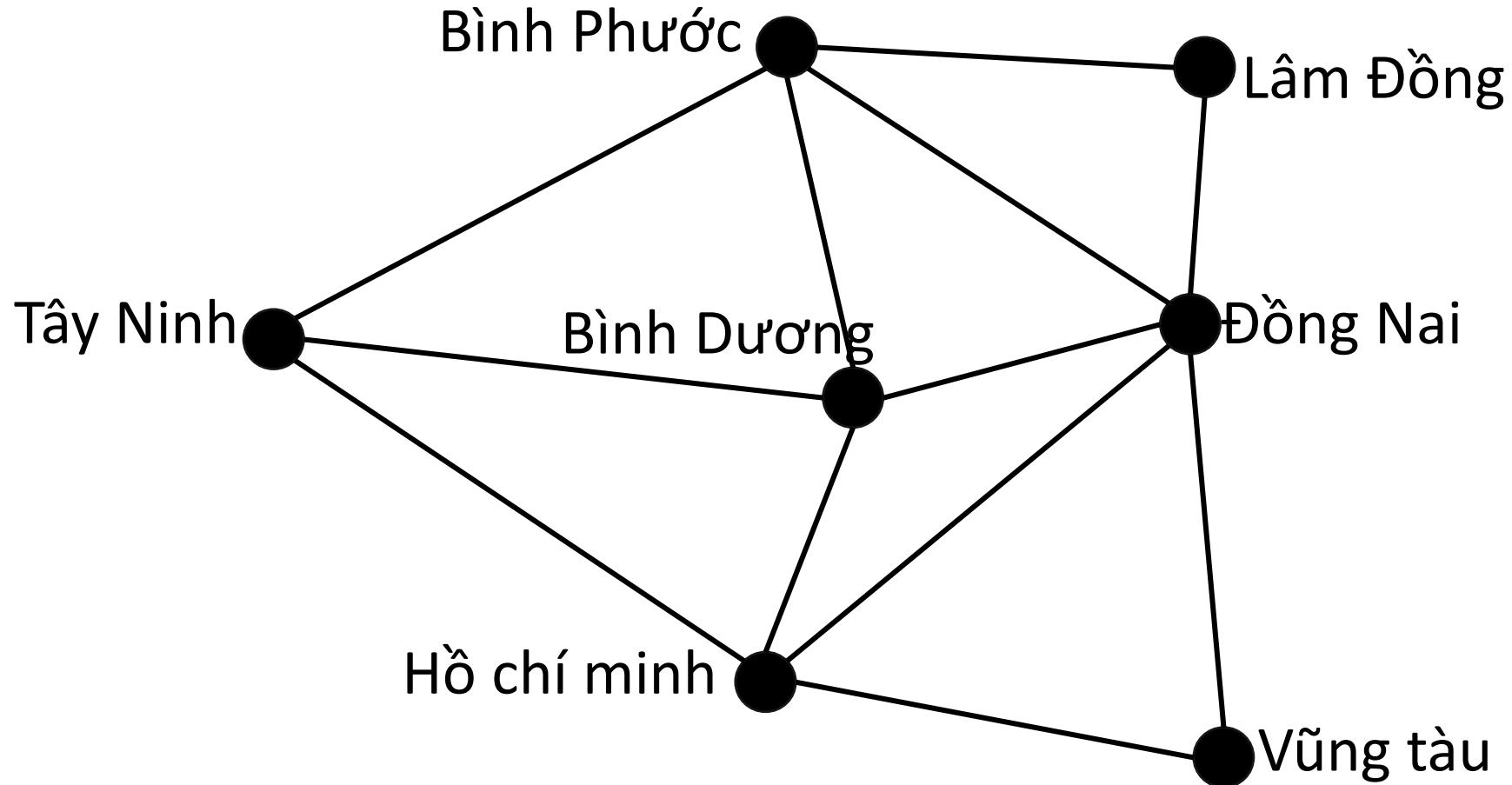
- ─ Nắm được khái niệm “**Đồ thị**”.
- ─ Biết cách **biểu diễn** đồ thị trên máy tính
- ─ Bài toán **duyệt** đồ thị
- ─ Bài toán **tìm kiếm** trên đồ thị

# NỘI DUNG

- 📖 Đồ thị và một số khái niệm liên quan
- 📖 Một số phương pháp biểu diễn đồ thị trên máy tính
- 📖 Một số phương pháp duyệt đồ thị

## **5.1 ĐỒ THỊ**

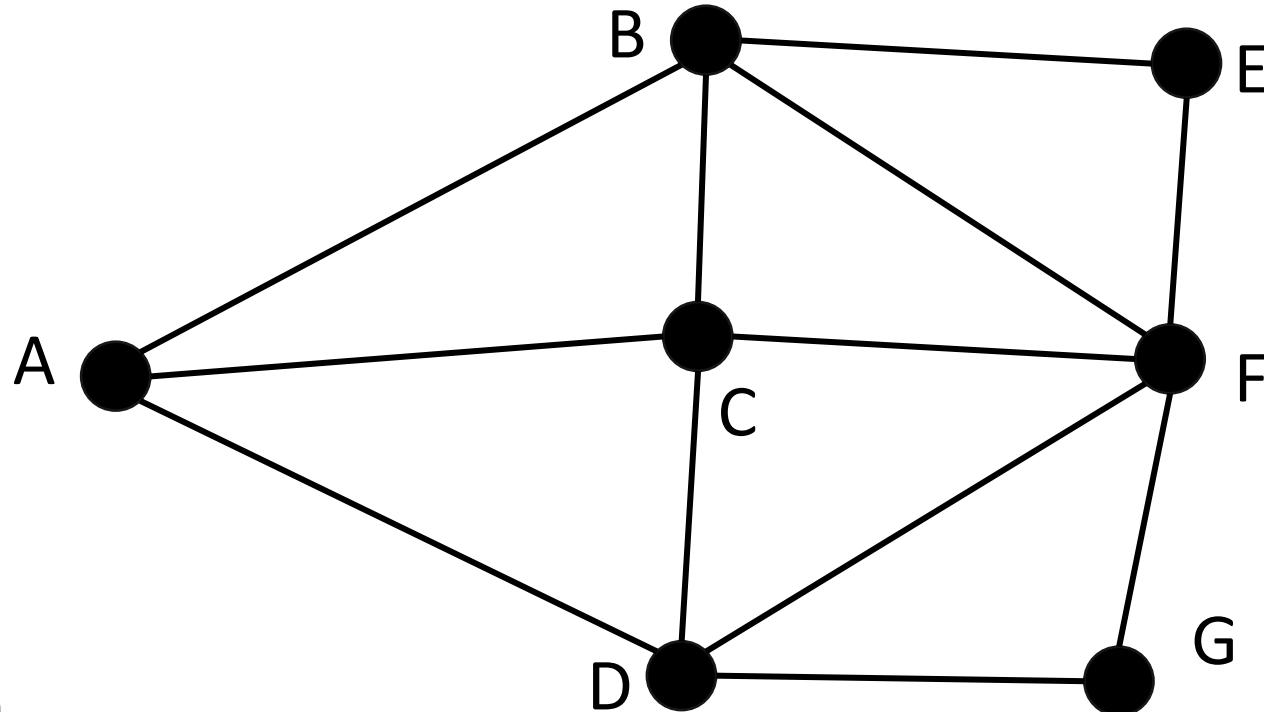
# BIỂU ĐỒ



# ĐỊNH NGHĨA

- 
- Đồ thị (**Graph**)  $G = (V, E)$  là một bộ gồm 2 thành phần:
- Các phần tử của  $V$  gọi là các **đỉnh** (**Vertises**) ( $V \neq \emptyset$ ),
  - Các phần tử của  $E$  gọi là các **cạnh** (**Edges**), mỗi cạnh tương ứng với 2 đỉnh.

## Ví dụ 5.1



$$V = \{A, B, C, D, E, F, G\}$$

$$E = \{ (A, B), (A, C), (A, D), (B, C), (B, E), (B, F), (C, F), (C, D), (D, F), (D, G) \}$$

# MỘT SỐ KHÁI NIỆM LIÊN QUAN

# ĐỈNH KỀ (Adjacent)

- Trong đồ thị  $G$ , nếu cạnh  $e \in E$ ,  $e$  nối 2 đỉnh  $u, v$  ( $u, v \in V$ ) thì ta nói  $u$  và  $v$  là hai đỉnh kề với nhau, kí hiệu  $e(u, v)$  hoặc  $e = \overline{uv}$

Ví dụ: Cho đồ thị  $G = (V, E)$  (như hình 6.1)

$$V = \{A, B, C, D, E, F, G\}$$

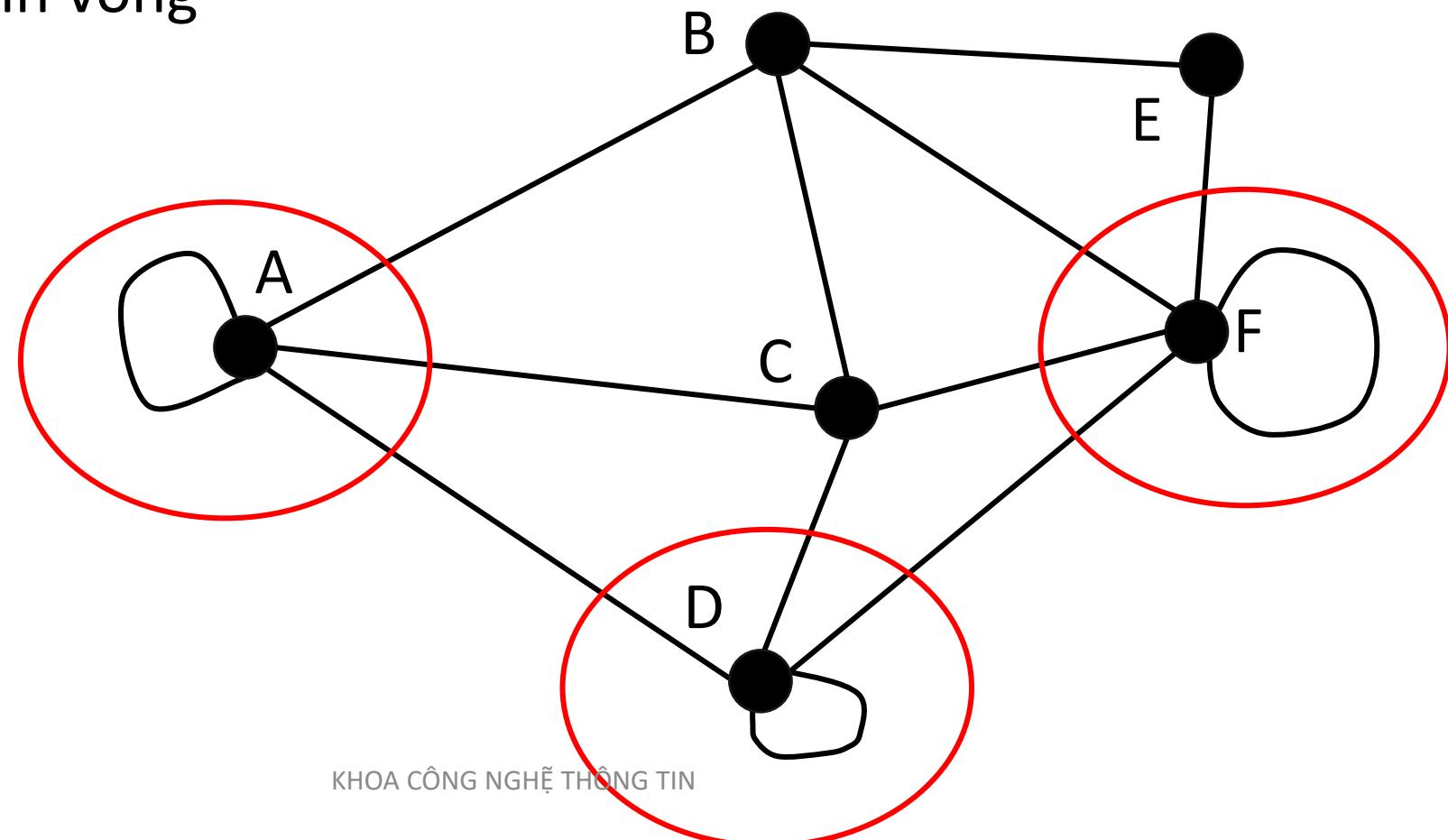
$$E = \{ (A, B), (A, C), (A, D), (B, C), (B, E), (B, F), (C, F), (C, D), (D, F), (D, G) \}$$

Ta nói A kề B, B kề A, A kề C, C kề A, A kề D, D kề A, ..., D kề G, G kề D;

# CẠNH VÒNG (loop)

- Trong đồ thi  $G$ , nếu cạnh  $e \in E$ ,  $e$  nối 2 đỉnh  $u, u'$  ( $u$  trùng với  $u'$ ) thì ta nói  $e$  là cạnh vòng

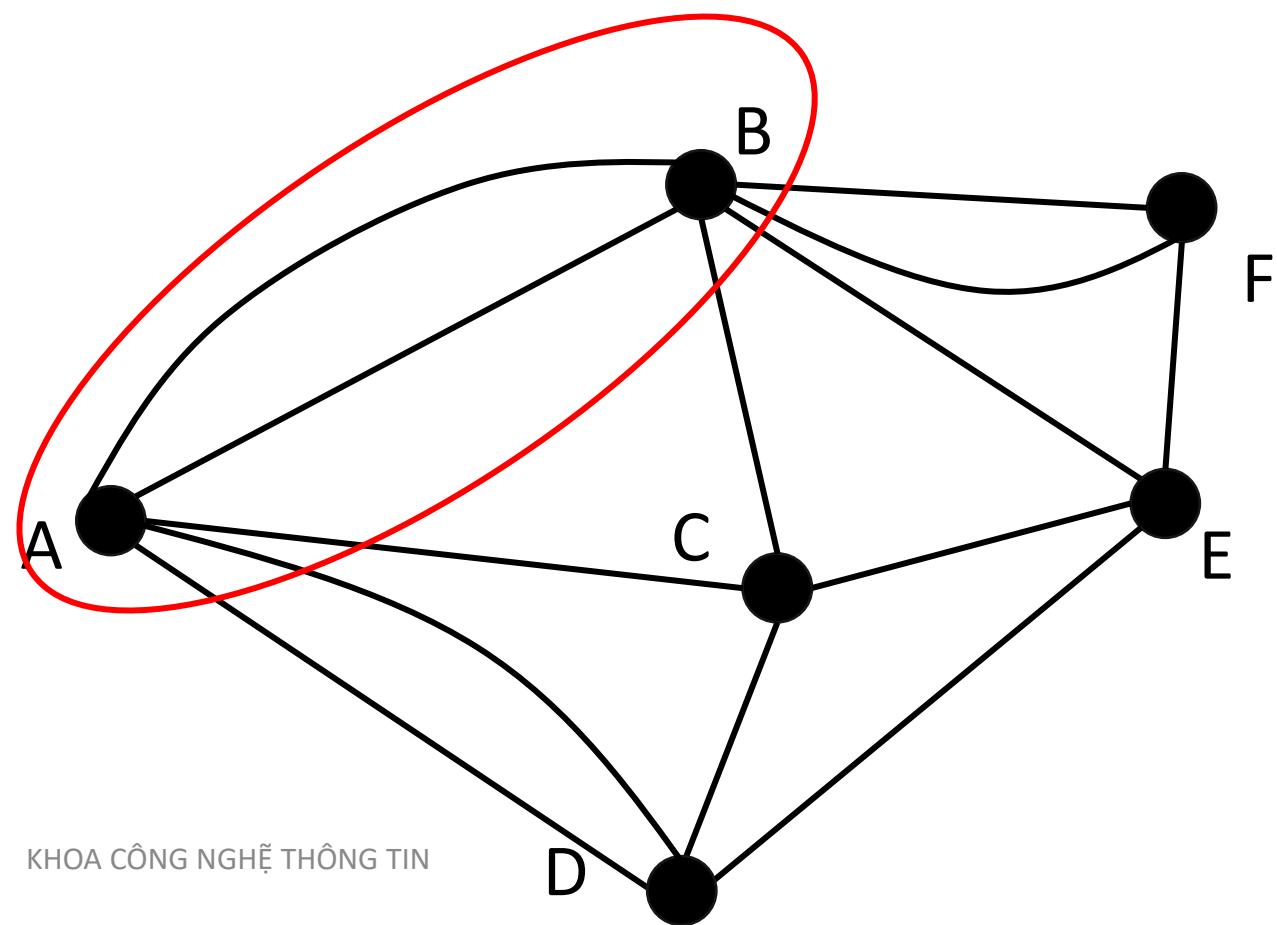
Ví dụ 5.2:



# CẠNH SONG SONG (parallel arcs)

- Trong đồ thi  $G$ , nếu cạnh  $e_1, e_2 \in E$ ,  $e_1$  nối 2 đỉnh  $u, v$ ,  $e_2$  cũng nối hai đỉnh  $u, v$  thì ta nói  $e_1, e_2$  là 2 cạnh song song

Ví dụ 5.3:



# BẬC CỦA ĐỈNH (degree of vertex)

- Trong đồ thị  $G$ , xét một đỉnh  $v \in V$ , ta nói bậc của đỉnh  $v$  được xác định bằng số cạnh tới  $v$ , trong đó mỗi vòng (loop) tại  $v$  được tính là 2 cạnh tới  $v$ , kí hiệu  $d(v)$

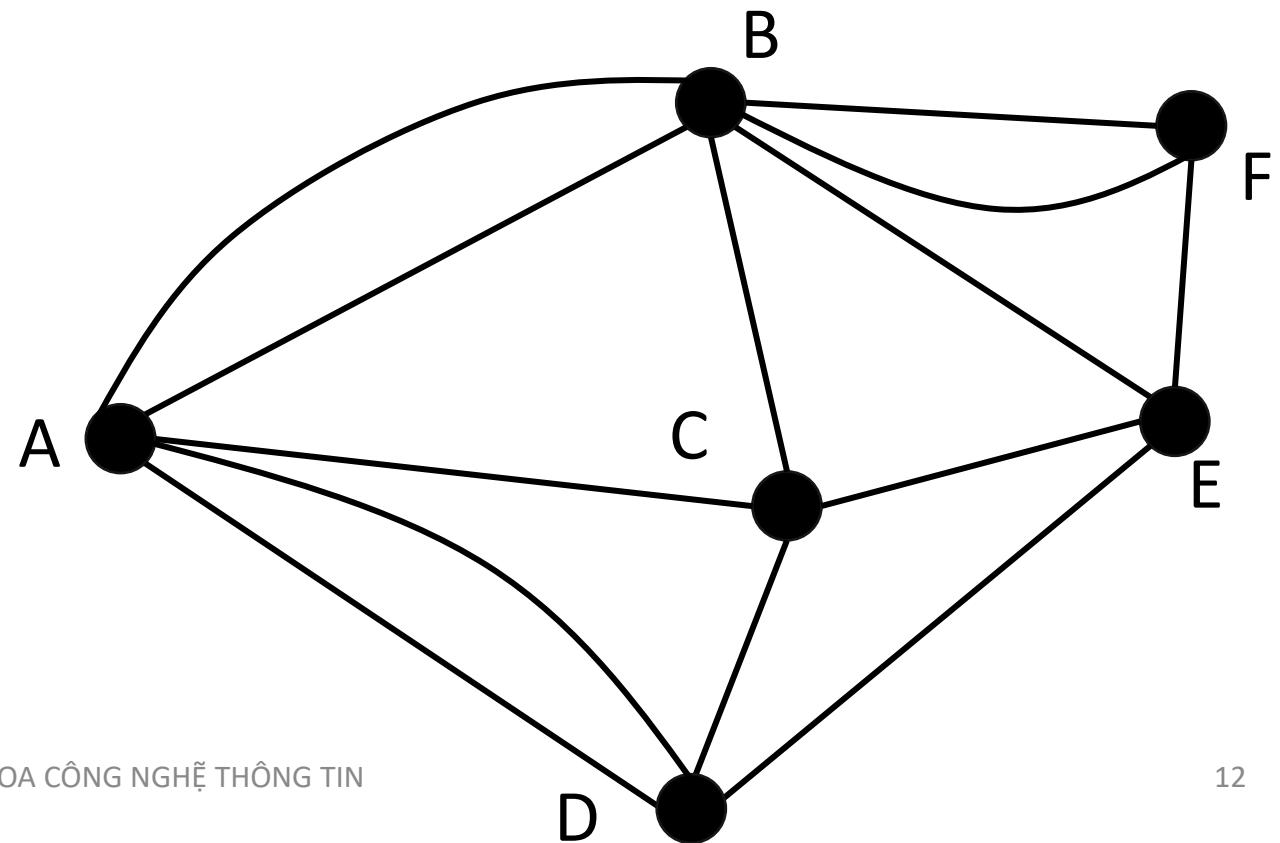
Ví dụ : xét lại ví dụ 5.3

$$d(A) = 5;$$

$$d(B) = 6;$$

...

$$d(F) = 3;$$



# ĐỒ THỊ CÓ TRỌNG SỐ (Weighted Graph)

Cho  $G = (V, E)$ , nếu  $\forall e \in E$ , e được đặt tương ứng với một trọng số (**weight**), kí hiệu  $w(e)$ , thì  $G$  là một *đồ thi có trọng số*.

**Ví dụ 5.7:** Cho  $G = (V, E)$

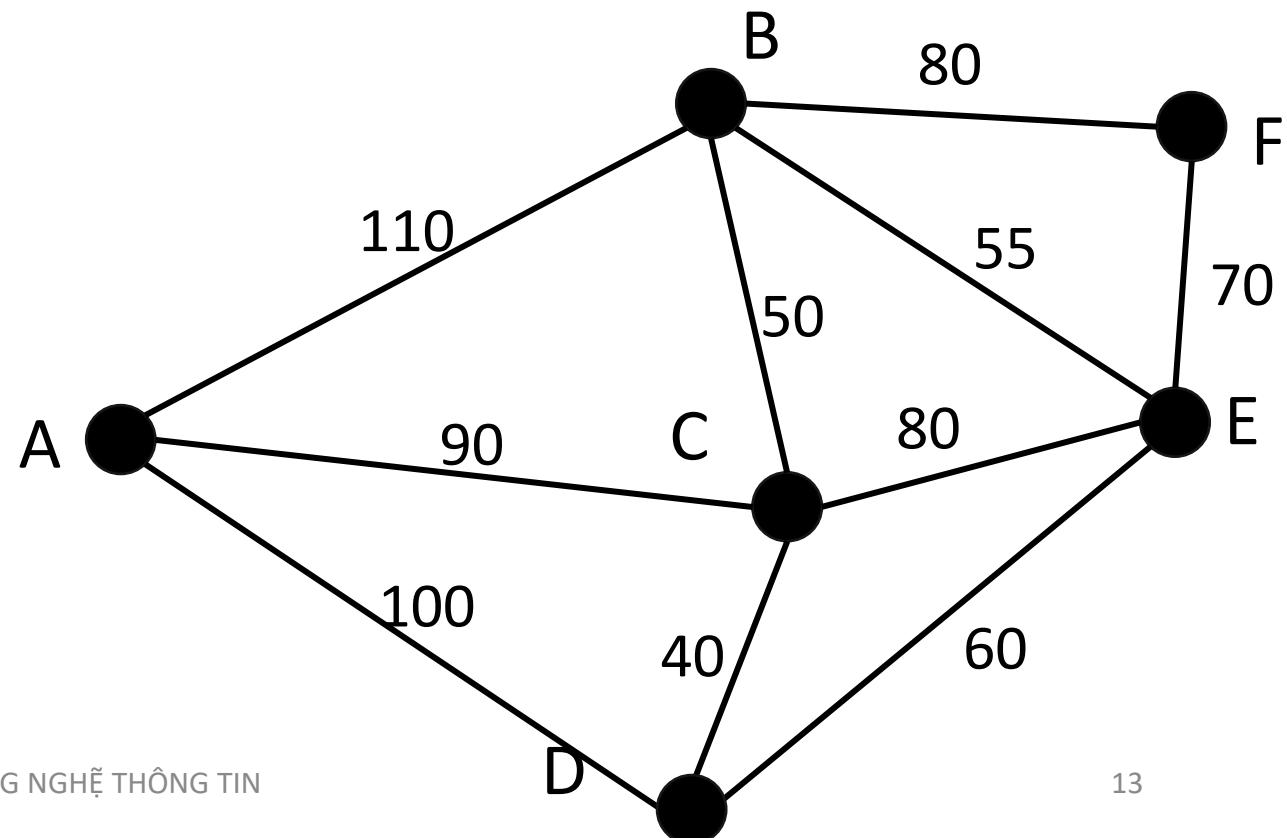
$$V = \{A, B, C, D, E, F\}$$

$$E = \{(A, B), (A, C), (A, D) \dots\}$$

$$w(A, B) = 110$$

$$w(A, C) = 90$$

....

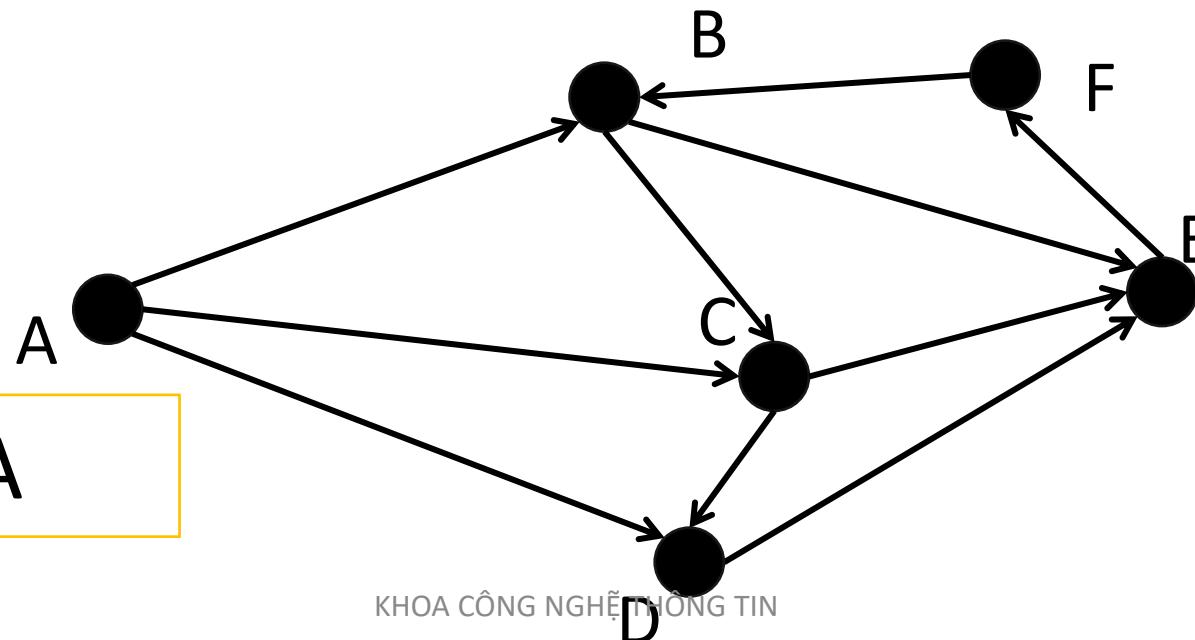


# ĐỒ THỊ CÓ HƯỚNG (Directed graph)

Cho đồ thị  $G = (V, E)$ , nếu mỗi cạnh  $e \in E$  được cho tương ứng với một **cặp thứ tự**  $(u, v)$  của 2 đỉnh  $u, v \in V$  thì ta nói  $e$  là một cạnh có hướng từ  $u$  đến  $v$ , và  $G$  được gọi là đồ thị có hướng.

Ví dụ: 5.6

A kề B  
B không kề A



# Một số loại đồ thị khác

-  Đồ thị đơn giản (Simple graph)
-  Đa đồ thị (Multiple graph)
-  Đồ thị có hướng (Directed Graph)
-  Đồ thị có hướng có trọng số (Weighted Directed Graph)
-  Đồ thị hỗn hợp (Mixed Graph)
-  ....

## 5.2. MỘT SỐ PHƯƠNG PHÁP BIỂU DIỄN ĐỒ THỊ TRÊN MÁY TÍNH



Ma Trận kè



Danh sách kè

## **5.2.1. MA TRẬN KÈ**

# MA TRẬN KỀ (Adjacency matrix)

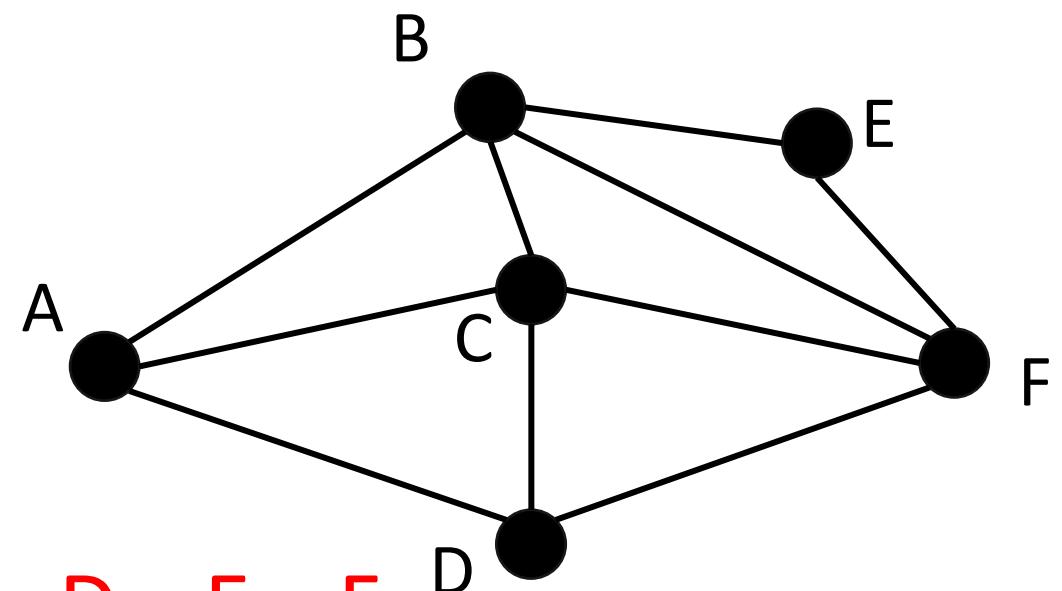
 Cho đồ thị  $G = (V, E)$  vô hướng không có trọng số, ta đánh các số các đỉnh của đồ thị bằng một số tự nhiên: 1, 2, ..., n.

Xây dựng ma trận vuông biểu diễn đồ thị như sau:

 Ma trận vuông  $A_{n \times n}$  được gọi là ma trận kề của G sao cho

$$A_{[i,j]} = \begin{cases} 1 & (\text{nếu } i \text{ kề với } j) \\ 0 & (i \text{ không kề } j) \end{cases}$$

## Ví dụ: 5.9



|   | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 0 | 0 |
| B | 1 | 0 | 1 | 0 | 1 | 1 |
| C | 1 | 1 | 0 | 1 | 0 | 1 |
| D | 1 | 0 | 1 | 0 | 0 | 1 |
| E | 0 | 1 | 0 | 0 | 0 | 1 |
| F | 0 | 1 | 1 | 1 | 1 | 0 |

# CÀI ĐẶT

# KHAI BÁO CẤU TRÚC MA TRẬN KÈ

```
#define MAX 20
int A[MAX][MAX]; // mảng hai chiều
int n; // số đỉnh của đồ thị
```

# KHỞI TẠO MẢNG RỖNG

```
void init()  
{  
    n=0;  
}
```

# NHẬP MA TRẬN

```
void input()
{
    cout<<"nhap so dinh do thi n: ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cout<<"nhap vao dong thu "<<i+1<<": ";
        for(int j=0;j<n;j++)
            cin>>A[i][j];
    }
}
```

# XUẤT MA TRẬN

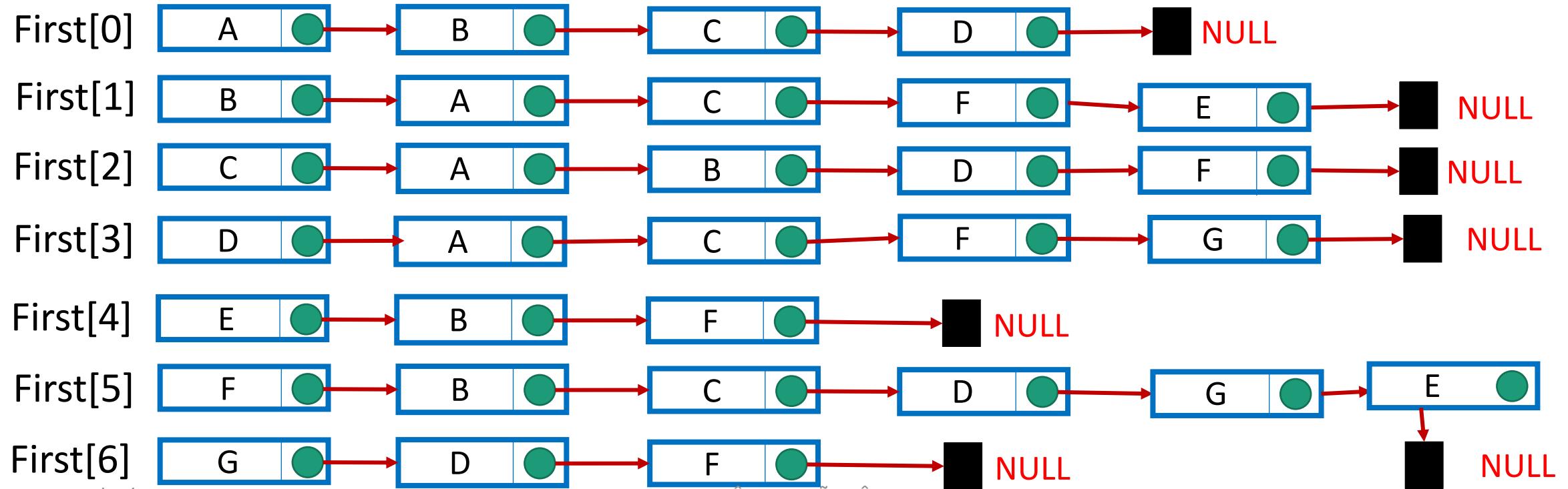
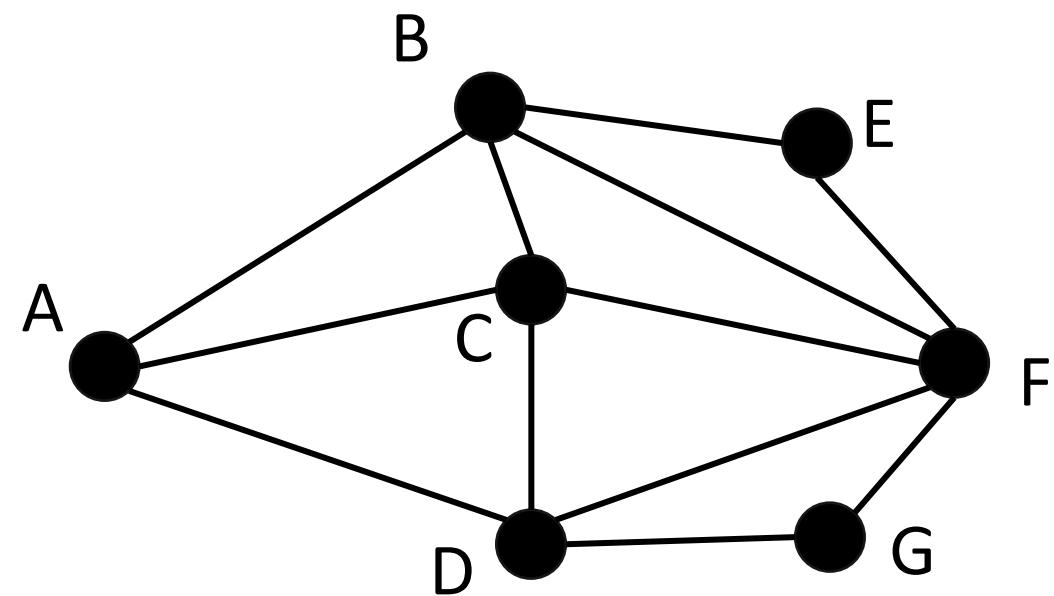
```
void output()
{
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
            cout<<A[i][j]<<" ";
        cout<<endl;
    }
}
```

## **5.2.2. DANH SÁCH KỀ**

# DANH SÁCH KỀ

- 📖 Với mỗi đỉnh **u** của đồ thị ta sẽ xây dựng **một danh sách** (danh sách liên kết đơn).
- 📖 Mỗi danh sách gồm phần tử đầu tiên là các đỉnh **u** (các đỉnh đồ thị), các phần tử trong danh sách là các đỉnh **v** (**u** kề **v**).
- 📖 **Một đồ thị** được biểu diễn bằng một mảng danh sách kèm.

## Ví dụ: 5.8



# CÀI ĐẶT

# KHAI BÁO CẤU TRÚC CHO MỘT MẢNG DANH SÁCH KÊ

```
#define MAX 20
struct Node
{
    int info;
    Node *link;
};

Node *first[MAX]; // mảng danh sách
int n; // so dinh tren do thi
```

# KHỞI TẠO MẢNG DANH SÁCH

```
void init()
{
    for(int i=0;i<n;i++)
        first[i] = NULL;
}
```

# THÊM MỘT PHẦN TỬ VÀO DANH SÁCH

```
void insert_first(Node *&f, int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->link = f;
    f = p;
}
```

# NHẬP MẢNG DANH SÁCH GỒM 'n' DANH SÁCH

```
void input()
{
    int d,x, m;
    cout<<"nhap so dinh do thi n: ";
    cin>>n;
    for(int i=0;i<n;i++)
    {
        cout<<"\nnhap dinh thu
"<<i+1<<" : ";
        cin>>d;
        insert_first(first[i],d);
        cout<<"nhap vao so dinh ke cua
"<<d<<" : ";
        cin>>m;
        for(int j=0;j<m;j++)
        {
            cin>>x;
            insert_first(first[i],x);
        }
    }
}
```

# XUẤT THÔNG TIN CỦA MỘT DANH SÁCH

```
void output_list(Node *f)
{
    if(f!=NULL)
    {
        Node * p=f;
        while (p != NULL)
        {
            cout<<p->info<<" ";
            p=p->link;
        }
    }
}
```

# XUẤT THÔNG TIN CỦA MẢNG DANH SÁCH

```
void output()
{
    if(n>0)
        for(int i=0;i<n;i++)
    {
        cout<<endl<<"Danh sach thu "<<i+1<<": ";
        output_list(first[i]);
    }
    else
        cout<<"rong";
}
```

## 5.3. MỘT SỐ PHƯƠNG PHÁP DUYỆT ĐỒ THỊ

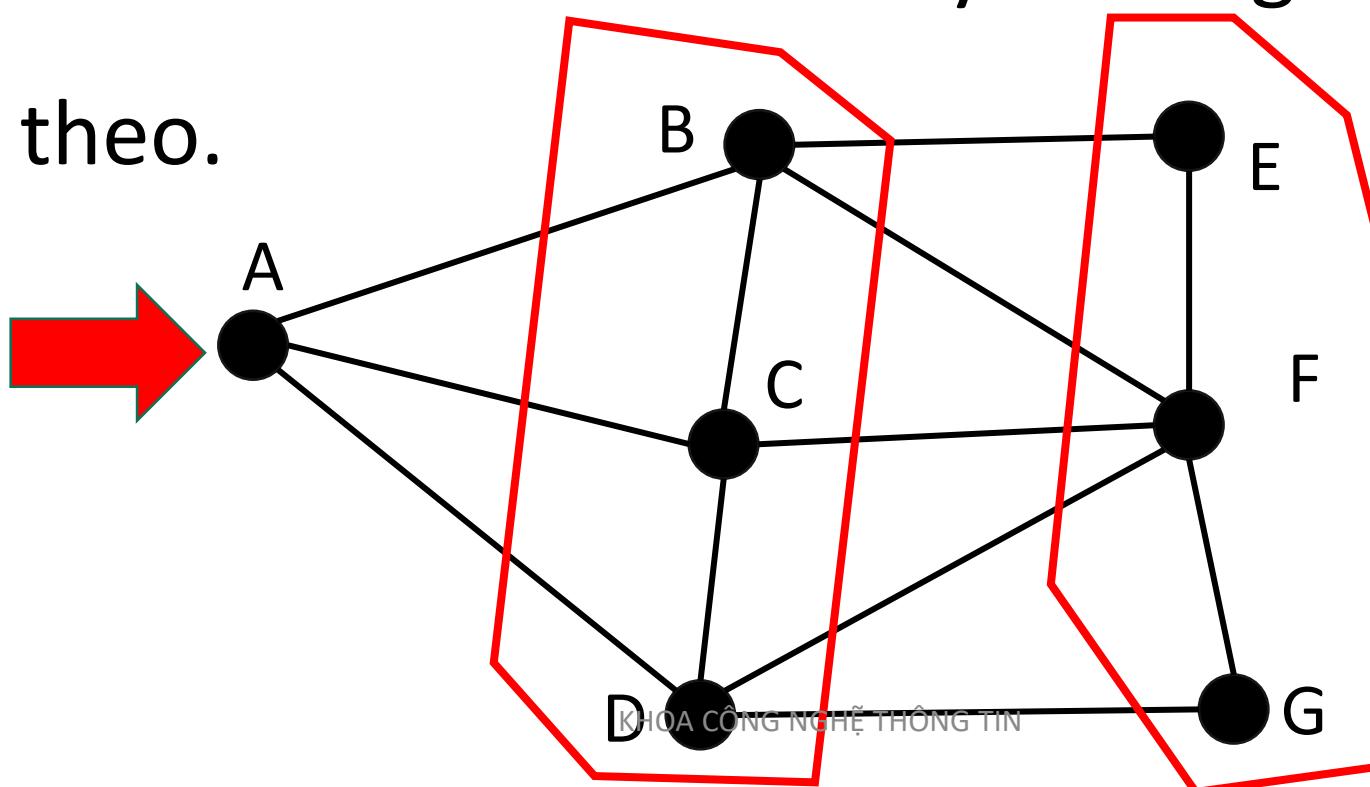
 Breadth First Search - BFS

 Depth First Search - DFS

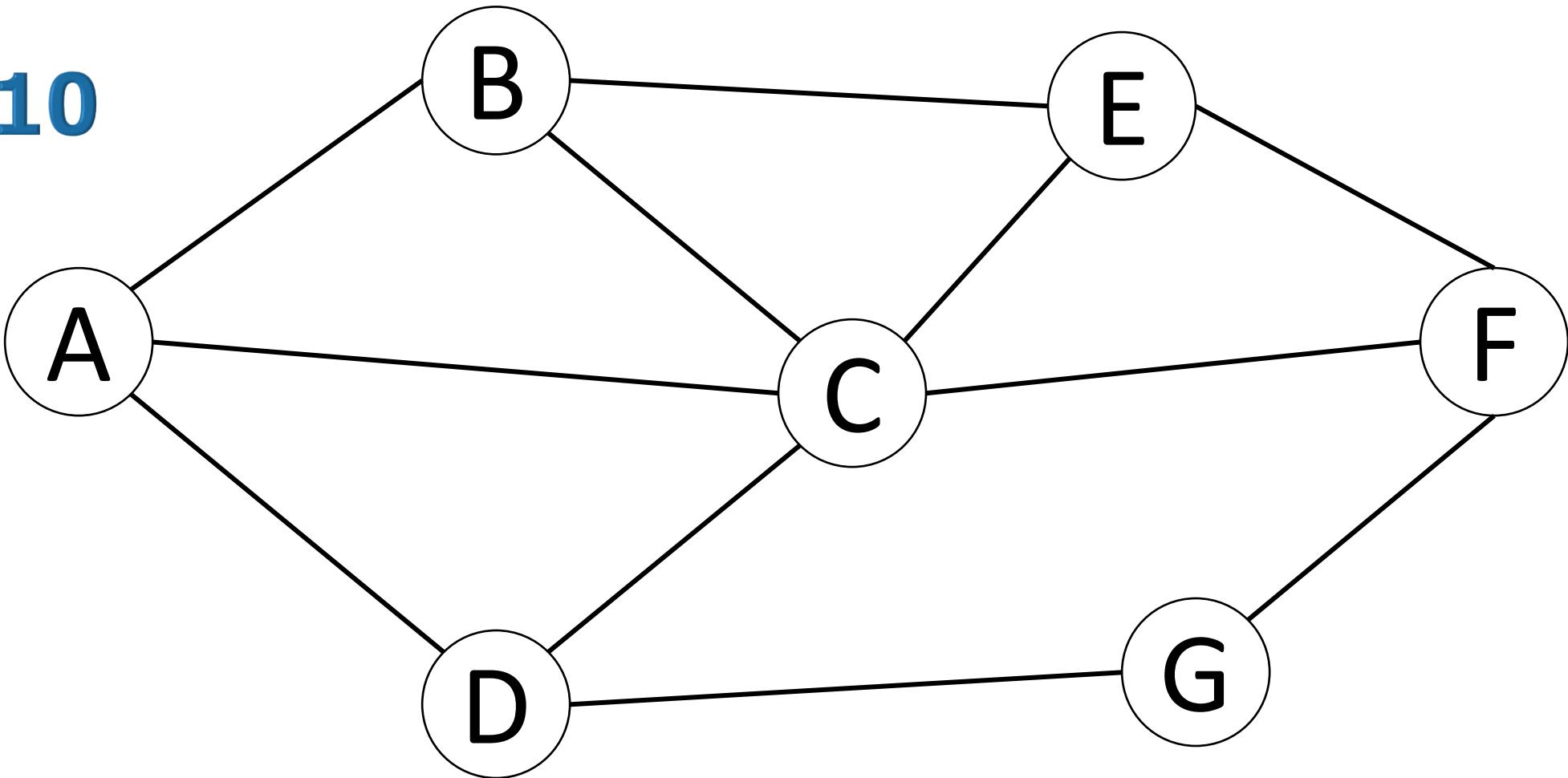
# **DUYỆT THEO CHIỀU RỘNG - BFS**

# BFS (Breadth First Search)

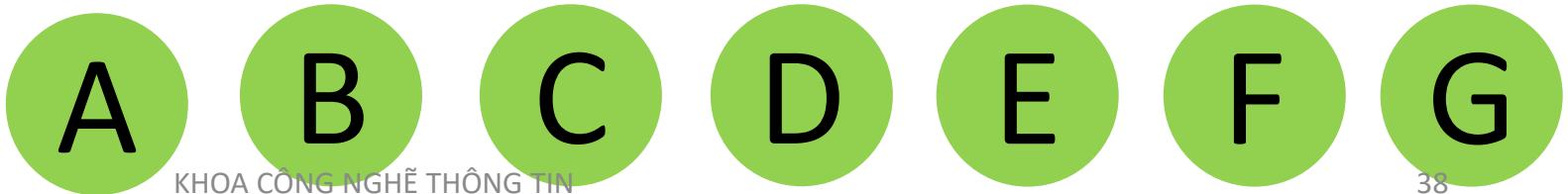
📖 Duyệt trên tất cả các nút của một mức trong không gian bài toán trước khi chuyển sang các nút ở mức tiếp theo.



## Ví dụ: 5.10



Kết Quả duyệt theo BFS:



# CÀI ĐẶT

# THUẬT GIẢI BFS

-  **Bước 0:** Giả sử  $s$  là đỉnh **bắt đầu**, cho  $s$  vào trong Q (Queue)
-  **Bước 1:** Nếu  $Q$  khác **rỗng**, gọi  $u$  là phần tử được lấy từ  $Q$ , và  
xuất  $u$  ra ngoài màn hình;
-  **Bước 2:** Tìm các đỉnh  $v$  kề với  $u$ , cho  $v$  vào trong  $Q$  quay lại  
bước 1.

# MỘT SỐ LƯU Ý KHI CÀI ĐẶT

- 📖 Quá trình xử lý dựa trên đồ thị được *biểu diễn* bằng *ma trận kề*. Các khai báo *cấu trúc* và *hàm nhập/xuất tham khảo* lại mục 5.2.2.
- 📖 Cấu trúc khai báo, quá trình thêm/ lấy phần tử trong *Queue* đã được trình bày trong *môn Cấu trúc dữ liệu*(cài đặt Queue bằng danh sách liên kết đơn)

# Ý TƯỞNG CÀI ĐẶT

- 📖 Dùng một danh sách đặc (mảng) **bfs** để lưu tập đỉnh đã duyệt.
- 📖 Dùng tập Q (**Queue**) để lưu các đỉnh kề với đỉnh vừa duyệt.
- 📖 Dùng tập C (**chuaxet**) để lưu các đỉnh chưa được duyệt (mới bắt đầu).

# THUẬT GIẢI BFS SỬ DỤNG QUEUE

```
int C[100]; // lưu trữ đỉnh chưa xét;  
// 1 là chưa xét; 0 là đã xét  
  
int bfs[100];// lưu danh sách phần tử đã duyệt  
int nbfs=0; // chỉ số lưu đỉnh đã xét  
  
Queue Q; // chỉ số lưu đỉnh đã xét  
void BFS(int v) // v là đỉnh bắt đầu  
{  
    int w, p;  
    push(v);  
    C[v]=0;  
    while(front!=NULL)
```

```
{  
    pop(p);  
    bfs[nbfs]=p;  
    nbfs++;  
    for(w=0; w<n; w++)  
        if(C[w]&&a[p][w]==1)  
        {  
            push(w);  
            C[w]=0;  
        }  
}
```

# HÀM KHỞI TẠO ĐỈNH CHƯA XÉT

```
void khoitaochuaxet()
{
    for(int i=0; i<n; i++) // n là số đỉnh của đồ thị
        C[i]=1;
}
```

# HÀM XUẤT CÁC ĐỈNH TRONG TẬP bfs

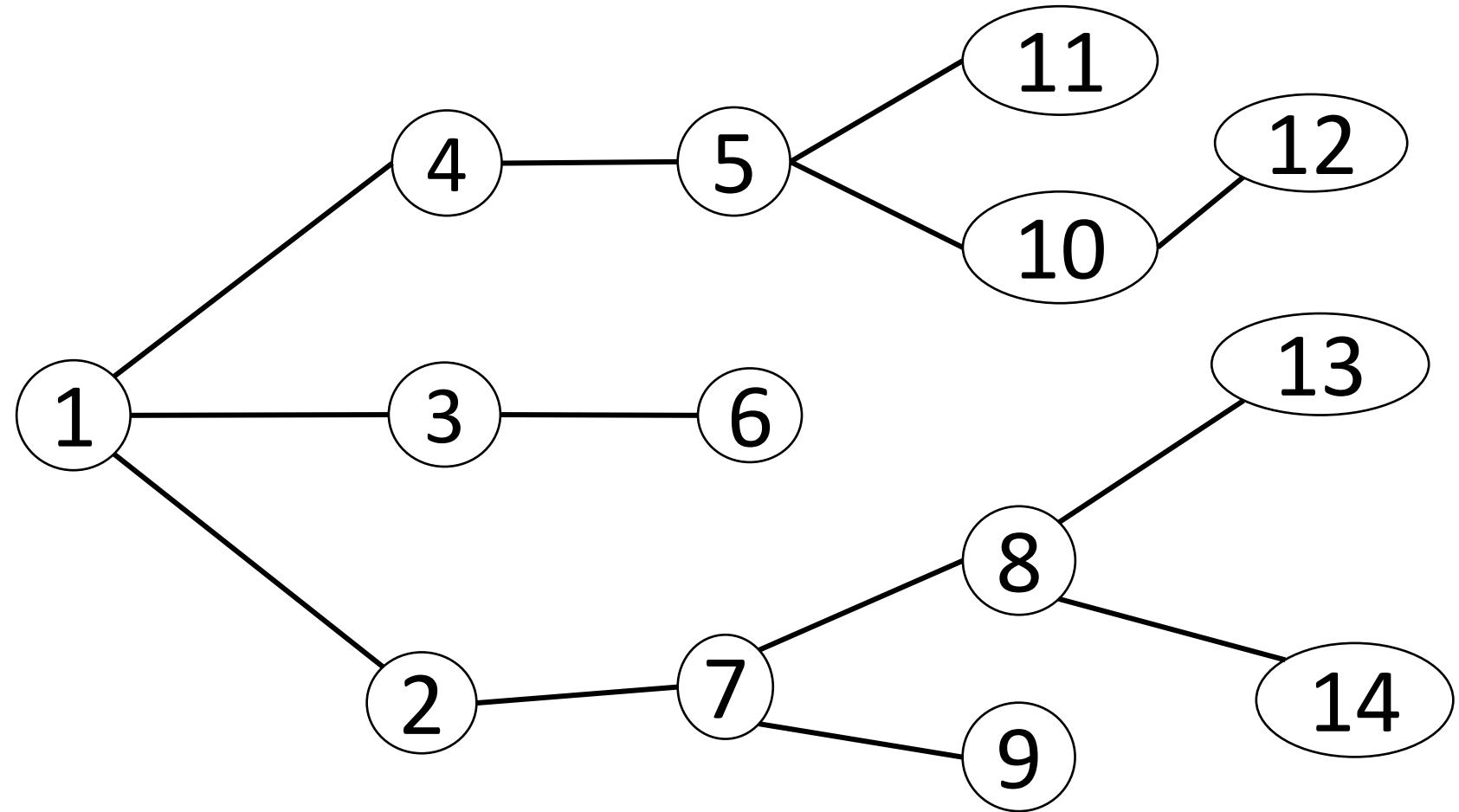
```
void output()
{
    for(int i=0; i<nbfs; i++) // n là số đỉnh của đồ thị
        cout<<bfs[i]<<" ";
}
```

## **5.3.2. DUYỆT THEO CHIỀU SÂU - DFS**

# Ý TƯỞNG DFS

- 📖 Bắt đầu từ một đỉnh **u**
- 📖 Từ **u** đi theo cạnh (cung) xa nhất có thể
- 📖 Nếu hết đường đi, trở lại đỉnh trước của cạnh xa nhất tiếp tục duyệt như trước, cho đến đỉnh cuối cùng.

## Ví dụ: 5.11



Kết Quả duyệt theo DFS:



|   | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| B | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| C | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| E | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| G | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| H | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

# CÀI ĐẶT

# THUẬT GIẢI DFS

-  **Bước 0:** Giả sử  $v$  là đỉnh bắt đầu, cho  $v$  vào Stack và cho  $v$  vào tập DFS;
-  **Bước 1:** Nếu Stack chưa rỗng, thì lấy  $x$  từ trong Stack ra;
-  **Bước 2:** Tìm  $u$  (gần nhất) kề với  $x$  và  $u$  chưa có trong tập DFS;
  - 2.1. cho  $x$  vào trong Stack,
  - 2.2.  $u$  vào trong Stack;
  - 2.3. Cho  $u$  vào tập DFS;
  - 2.4. Sau đó quay lại bước 1;

# MỘT SỐ LƯU Ý KHI CÀI ĐẶT

- Quá trình xử lý dựa trên đồ thị được biểu diễn bằng ma trận kề. Các khai báo cấu trúc và hàm nhập xuất đồ thị tham khảo lại [mục 5.2.2](#).
- Cấu trúc khai báo, quá trình thêm/ lấy phần tử trong Stack đã được trình bày trong [chương 2 mục 2.4.4](#) (cài đặt Stack bằng danh sách liên kết đơn)

# Ý TƯỞNG CÀI ĐẶT

- 📖 Dùng một danh sách **dfs** để lưu tập đỉnh đã duyệt.
- 📖 Dùng sp (**Stack**) để lưu các đỉnh kề với đỉnh vừa duyệt.
- 📖 Dùng tập C (**chuaxet**) để lưu các đỉnh chưa được duyệt (lúc mới bắt đầu).

# THỦ TỤC DFS DÙNG STACK

```
int C[100]; // lưu trữ đỉnh chưa xét;  
// 1 là chưa xét; 0 là đã xét  
int dfs[100];// lưu danh sách phần tử đã duyệt  
int ndfs=0; // chỉ số lưu đỉnh đã xét  
STACK sta=0; // chỉ số lưu đỉnh đã xét  
void DFS(int s)  
{  
    push(s);  
    dfs[ndfs]=s;  
    ndfs++;  
    C[s]=0;  
    int v=-1, u=s;  
    while(!isEmpty())  
    {
```

```
        if(v==n)  
            pop(u);  
        for(v=0;v<n;v++)  
            if(a[u][v]!=0 && C[v]==1)  
            {  
                push(u);  
                push(v);  
                dfs[ndfs]=v;  
                ndfs++;  
                C[v] = 0;  
                u=v;  
                break;  
            }
```



# HÀM XUẤT TẬP **dfs**

```
void output()
{
    for(int i=0;i<nDFS;i++)
        cout<<DFS[i]<<" ";
}
```

# HÀM KHỞI TẠO ĐỈNH CHƯA XÉT

```
void khoitaochuaxet()
{
    for(int i=0; i<n; i++) // n là số đỉnh của đồ thị
        C[i]=1;
}
```

## 5.4. TÌM KIẾM TRÊN ĐỒ THỊ

-  Dựa trên phép duyệt BFS
-  Dựa trên phép duyệt DFS

# PHÁT BIỂU BÀI TOÁN

 Cho đồ thị vô hướng đơn giản  $G = (V, E)$ , và một đỉnh X được nhập vào từ bàn phím. Hãy kiểm tra X có tồn tại trong G hay không?

# MÔ HÌNH HÓA BÀI TOÁN

-  **Đầu vào:** một đồ thị G, và một đỉnh X;
-  **Đầu ra:** TRUE/FALSE

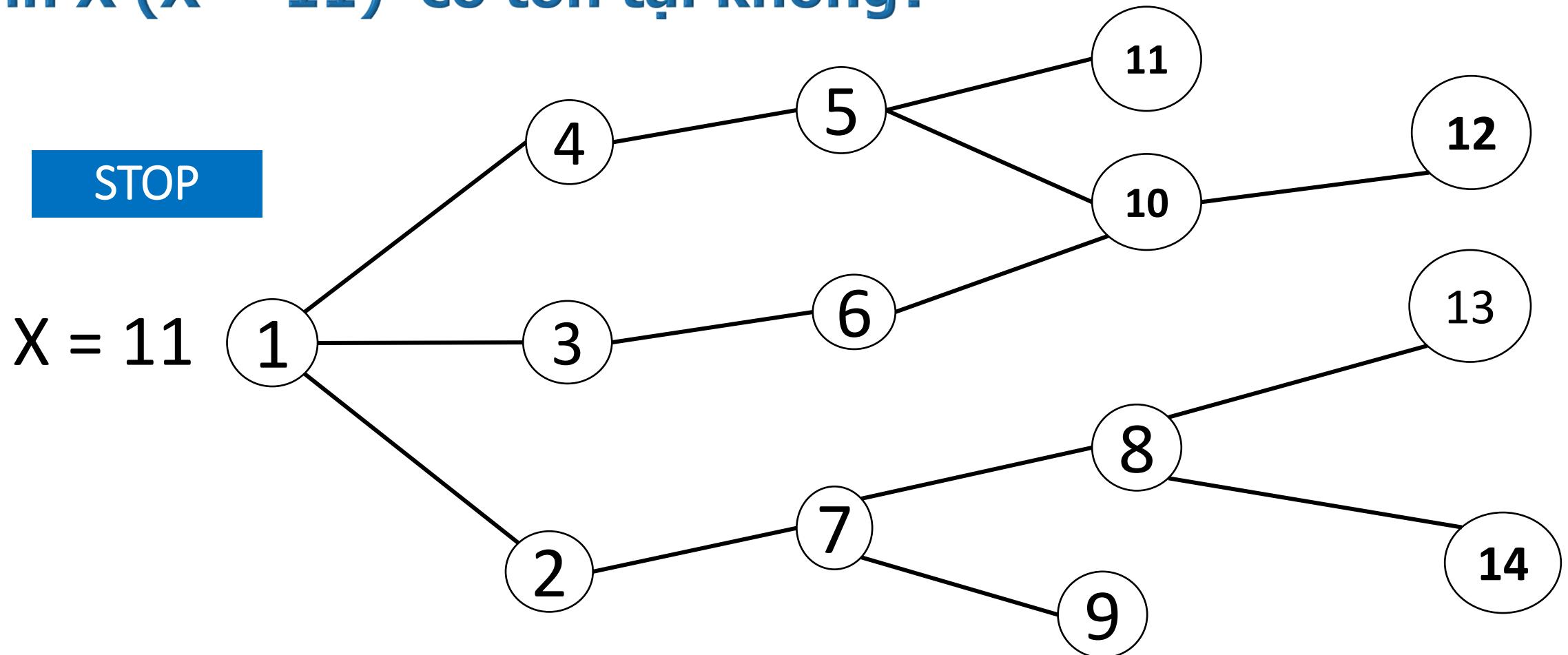
# BIỂU DIỄN BÀI TOÁN LÊN MÁY TÍNH

 Ta biểu diễn đồ thị  $G$  lên máy tính bằng Ma  
trận kề (hoặc danh sách kề).

# Ý TƯỞNG TÌM KIẾM

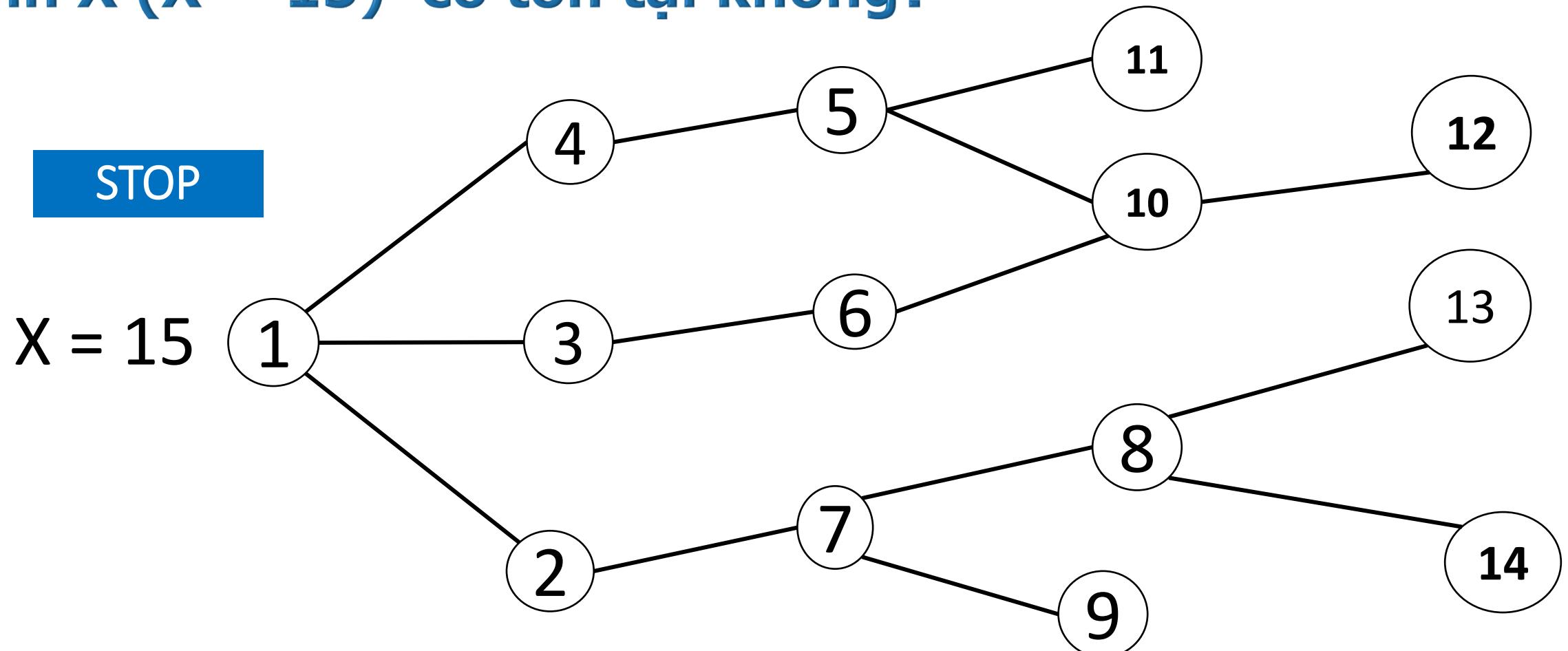
-  Ta thấy rằng, phương pháp duyệt đồ thị DFS, BFS sẽ cho phép ta duyệt qua tất cả các đỉnh của đồ thị.
-  Vì vậy, thay vì ta chỉ xuất ra ngoài màn hình đỉnh của đồ thị được duyệt. Mà ta có thể thực hiện lệnh kiểm X với đỉnh được duyệt.
-  Nếu đỉnh được duyệt bằng (giống giá trị) X thì ta thực hiện dừng phép duyệt và thông báo rằng X tìm thấy.

## Ví dụ 5.12 cho đồ thị như hình bên dưới, hãy kiểm tra đỉnh X ( $X = 11$ ) có tồn tại không?



Minh họa thuật giải tìm kiếm bằng phương pháp duyệt theo chiều rộng

## Ví dụ 5.13 cho đồ thị như hình bên dưới, hãy kiểm tra đỉnh X ( $X = 15$ ) có tồn tại không?



Minh họa thuật giải tìm kiếm bằng phương pháp duyệt theo chiều rộng

# CÀI ĐẶT THUẬT TOÁN TÌM KIẾM DỰA TRÊN PHÉP DUYỆT BFS

```
int C[100]; // lưu trữ đỉnh chưa xét;  
void Search_by_BFS(int x, int v) // v là  
đỉnh bắt đầu  
{  
    int w, p;  
    push(v);  
    C[v]=0;  
    while(front!=NULL)  
    {  
        pop(p);  
        if (x == p)  
        {  
            cout<<x<<"ton tai" ;  
            return;  
        }  
        for(w=0; w<n; w++)  
            if(C[w]&&a[p][w]==1)  
            {  
                push(w);  
                C[w]=0;  
            }  
    }  
}
```

# TỔNG KẾT CHƯƠNG

-  Đồ thị và một số khái niệm liên quan
-  Một số phương pháp biểu diễn đồ thị
-  Một số phương pháp duyệt đồ thị
-  Bài toán tìm kiếm trên đồ thị

# TÀI LIỆU THAM KHẢO

1. Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Cliffrod Stein, (Chapter 22) *Introduction to Algorithms*, Third Edition, 2009.
2. judith l. gersting, (Chapter 6) Mathematical structures for computer science-w. h. freeman (475-603)

# Bài tập chương 5

-  Bài 1: Thực hiện đếm bậc các đỉnh tại ví dụ 5.1, 5.2, 5.3, 5.6
-  Bài 2: Cho biết ma trận kề của đồ thị tại ví dụ 5.1, 5.6, 5.7, 5.11
-  Bài 3: Cho biết danh sách kề của đồ thị tại ví dụ 5.1, 5.7, 5.8, 5.11

## Bài 4: Viết chương trình với các yêu cầu sau:

-  Viết hàm nhập ma trận kề của đồ thị 5.1
-  Viết hàm xuất ma trận kề này ra ngoài màn hình.
-  Viết hàm duyệt đồ thị theo chiều rộng (dựa trên Queue bằng kỹ thuật cài đặt danh sách liên kết đơn)
-  Viết hàm duyệt đồ thị theo chiều sâu (dựa trên Stack bằng kỹ thuật cài đặt danh sách liên kết đơn)
-  Nhập vào một đỉnh X, hãy kiểm tra X có tồn tại trên đồ thị hay không?  
(dựa trên phép duyệt BFS)

## Bài 5: Viết chương trình với các yêu cầu sau:

- █ Viết hàm nhập danh sách kề của đồ thị tại ví dụ 5.1;
- █ Viết hàm xuất danh sách kề này ra ngoài màn hình.
- █ Viết hàm duyệt đồ thị theo chiều rộng (dựa trên Queue bằng kỹ thuật cài đặt danh sách liên kết đơn)
- █ Viết hàm duyệt đồ thị theo chiều sâu (dựa trên Stack bằng kỹ thuật cài đặt danh sách liên kết đơn)
- █ Nhập vào một đỉnh X, hãy kiểm tra X có tồn tại trên đồ thị hay không?  
(dựa trên phép duyệt BFS)

# Bài tập làm thêm

# Bài 6: Viết chương trình với các yêu cầu sau:

-  Viết hàm nhập ma trận kề của đồ thị 5.1,
-  Viết hàm xuất ma trận kề này ra ngoài màn hình.
-  Viết hàm duyệt đồ thị theo chiều rộng (dựa trên Queue bằng kỹ thuật cài đặt danh sách đặc)
-  Viết hàm duyệt đồ thị theo chiều sâu (dựa trên Stack bằng kỹ thuật cài đặt danh sách đặc)
-  Nhập vào một đỉnh X, hãy kiểm tra X có tồn tại trên đồ thị hay không?  
(dựa trên phép duyệt DFS)

## Bài 7: Viết chương trình với các yêu cầu sau:

-  Viết hàm nhập danh sách kề của đồ thị tại ví dụ 5.1;
-  Viết hàm xuất danh sách kề này ra ngoài màn hình.
-  Viết hàm duyệt đồ thị theo chiều rộng (dựa trên Queue bằng kỹ thuật cài đặt danh sách đặc)
-  Viết hàm duyệt đồ thị theo chiều sâu (dựa trên Stack bằng kỹ thuật cài đặt danh sách đặc)
-  Nhập vào một đỉnh X, hãy kiểm tra X có tồn tại trên đồ thị hay không?  
(dựa trên phép duyệt DFS)

Bài 8: Biểu diễn G ở ví dụ 5.11 (bằng ma trận kề) lên trên máy tính và đặt tên là do\_thi\_1.txt (dạng file TEXT) sau đó thực hiện viết chương trình với các hàm sau:

- 8.1. Hàm đọc file do\_thi\_1.txt và lưu vào mảng hai chiều;
- 8.2. Thực hiện xuất các đỉnh đồ thị theo phép duyệt DFS (dung Queue với kỹ thuật cài đặt bằng dách sách liên kết);
- 8.3. Thực hiện xuất các đỉnh đồ thị theo phép duyệt BFS (dung Stack với kỹ thuật cài đặt bằng danh sách liên kết);
- 8.3. Thực hiện xuất các đỉnh đồ thị theo phép duyệt BFS (dung kỹ thuật đệ quy)
- 8.4. Thực hiện nhập vào một đỉnh x, kiểm tra x có phải là một đỉnh của đồ thị hay không?