I.Duyệt theo chiều sâu – DFS (Depth First Search)

1. Thuật toán:

- Bước 1. Xuất phát từ 1 đỉnh cho trước nào đó.
- Bước 2. Xử lý đỉnh này và đánh dấu đã duyệt để không xử lý lần sau.
- Bước 3. Đưa tất cả các đỉnh kề với nó vào danh sách xử lý và chọn 1 đỉnh để xử lý tiếp theo.
- Bước 4. Quay lại Bước 2 cho đến khi không còn đỉnh trong danh sách.

2. Cài đặt:

```
//hàm xét từ đỉnh s bất kỳ
void DFS(int s, GRAPH &g)
{
	//Đánh dấu đỉnh s đã duyệt
	g.visited[s] = 1;

	//Tìm xem từ đỉnh s có đỉnh i nào chưa duyệt và nối trực tiếp với s
	for(int i = 0; i < g.sodinh; i++)
		if(g.visited[i] == 0 && g.A[s][i] != 0)
		{
			g.LuuVet[i] = s; //Lưu trước đỉnh i là đỉnh s
			DFS(i, g);//gọi đệ quy tiến hành xét tiếp
		}
}
//Hàm tìm đường đi từ đỉnh s đến đỉnh f trong đồ thị g
void duyetDFS(int s, int f, GRAPH &g)
{
	//Khởi tạo giá trị ban đầu, tất cả các đỉnh chưa được duyệt và chưa lưu vết
	for(int i = 0; i < g.sodinh; i++)
	{
		g.visited[i] = 0;
		g.LuuVet[i] = -1;
	}

	//Gọi hàm DFS
	DFS(s, g);

	if(g.visited[f] == 1)
	{
		//In ket qua
		int j = f;
		while(j != s)
		{
			cout<<j<<"<---";
			j = g.LuuVet[j];
		}
		cout<<s;
		cout<<endl;
	}
	else
		cout<<"Khong co duong di tu dinh "<<s<<" den dinh "<<f;
```

```
        cout<<endl;
}
```

II. Duyệt theo chiều rộng – BFS (Bread First Search)

1. Ý tưởng:

- Bước 1. Xuất phát từ 1 đỉnh cho trước nào đó.
- Bước 2. Xử lý đỉnh này và đánh dấu để không xử lý lần sau.
- Bước 3. Đưa tất cả các đỉnh kề với nó vào danh sách xử lý và lần lượt xử lý các đỉnh kề với đỉnh đang xét
- Bước 4. Quay lại Bước 2 cho đến khi không còn đỉnh trong danh sách.

2. QUEUE

Sử dụng cấu trúc dữ liệu hàng đợi QUEUE để khử đệ quy.

Định nghĩa QUEUE với một số phép toán thêm, lấy phần tử và khởi tạo.

```
struct QUEUE{
        int size;
        int a[MAX];
};
void KhoiTao(QUEUE &q){ q.size = 0;}
bool Them(int k, QUEUE &q)
{
        if(q.size + 1 > MAX) return false;
        q.a[q.size] = k;
        q.size++;
        return true;
}
bool KiemTraRong(QUEUE q)
{
        return (q.size == 0);
}
bool Lay(int &v, QUEUE &q)
{
        if(KiemTraRong(q)) return false;
        v = q.a[0];
        for(int i = 0; i < q.size - 1; i++)
                q.a[i] = q.a[i+1];
        q.size--;
        return true;
}
```

3. Hàm BFS dùng để tìm đường đi từ đỉnh s

```
void BFS(int s, GRAPH &g)
{
        QUEUE q;
        KhoiTao(q);
        Them(s, q);//Thêm đỉnh s vào q

        while(!KiemTraRong(q))
```

```
        {
                Lay(s, q);
                //Đánh dấu đỉnh s đã suyệt
                g.visited[s] = 1;
                for(int i = 0; i < g.sodinh; i++)
                if(g.visited[i] == 0 && g.A[s][i] != 0)
{
                        Them(i, q);
                        g.LuuVet[i] = s;
                }
        }
}
```

4. Hàm duyệt đồ thị tìm đường đi từ đỉnh s đến f bằng BFS

```
void duyetBFS(int s, int f, GRAPH &g)
{
        //Khởi tạo giá trị ban đầu, tất cả các đỉh chư được duyệt và chưa lưu vết
        for(int i = 0; i < g.sodinh; i++)
        {
                g.visited[i] = 0;
                g.LuuVet[i] = -1;
        }

        //Gọi hàm BFS
        BFS(s, g);

        if(g.visited[f] == 1)
        {
                //In ket qua
                int j = f;
                while(j != s)
                {
                        cout<<j<<"<---";
                        j = g.LuuVet[j];
                }
                cout<<s;
                cout<<endl;
        }
        else
                cout<<"Khong co duong di tu dinh "<<s<<" den dinh "<<f;
        cout<<endl;
}
```

III. Demo

Trong hàm **main()** gọi hàm duyetDFS() hoặc duyetBFS() với các tham số truyền vào tương ứng.

Lưu ý: Ma trận kề trong đồ thị chỉ số đánh từ 0.