**Dept First Search**

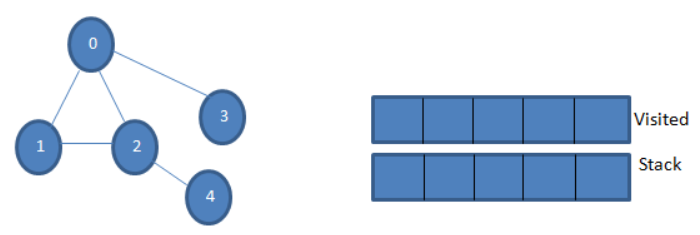
1. **Giới thiệu thuật toán :**

Giải thuật tìm kiếm theo chiều sâu (Depth First Search – viết tắt là DFS), còn được gọi là giải thuật tìm kiếm ưu tiên chiều sâu, là giải thuật duyệt hoặc tìm kiếm trên một cây hoặc một đồ thị và sử dụng stack (ngăn xếp) ghi nhớ nút liền kề, bắt đầu việc tìm kiếm khi không gặp được nút liền kề trong bất kỳ vòng lặp nào. Giải thuật tiếp tục cho tới khi gặp được nút cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về nút vừa mới tìm kiếm ở bước trước.

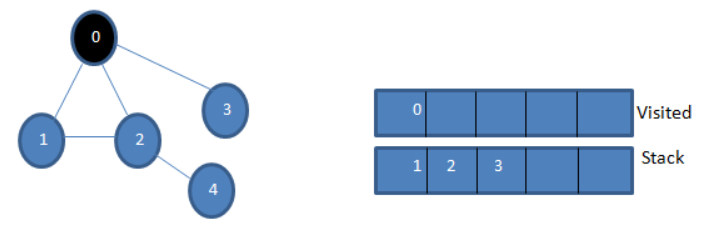
**Thuật toán DFS**

* **Bước 1:** Chèn nút gốc hoặc nút bắt đầu của cây hoặc biểu đồ trong ngăn xếp.
* **Bước 2:** Bật mục trên cùng từ ngăn xếp và thêm nó vào danh sách đã truy cập.
* **Bước 3:** Tìm tất cả các nút lân cận của nút được đánh dấu đã thăm và thêm những nút chưa được truy cập vào ngăn xếp.
* **Bước 4** : Lặp lại bước 2 và 3 cho đến khi chồng trống.

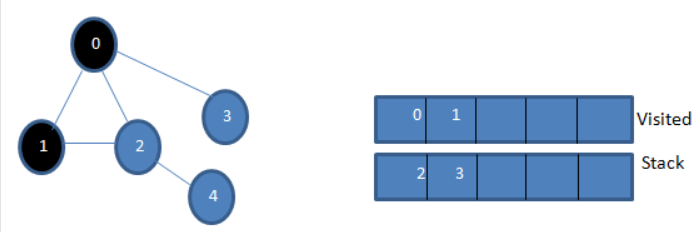
**Ví dụ :** Cho đồ thị như bên dưới, ta sẽ tạo 2 nhóm, visited lưu các nút đã duyệt và ngăn xếp stack chứa các nút liền kề của nó



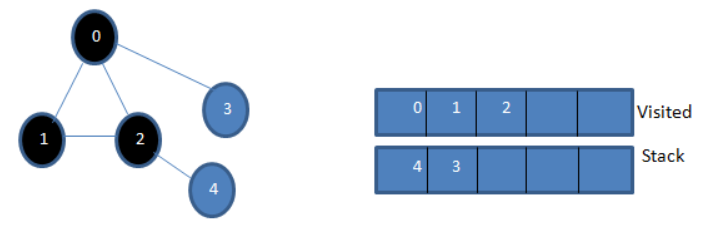
Gọi 0 là nút bắt đầu. Đầu tiên, đánh dấu nó là đã duyệt và thêm nó vào danh sách visited. Sau đó, đẩy tất cả các nút liền kề của nó trong ngăn xếp ( 1, 2 , 3)



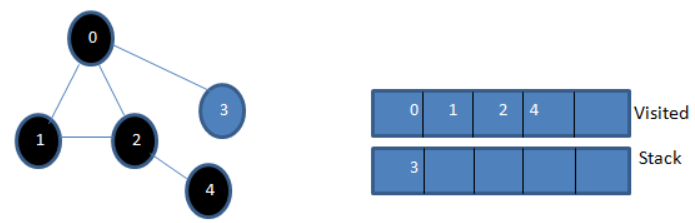
Tiếp theo, lấy một trong các nút liền kề để xử lý, hay đỉnh của ngăn xếp lúc này, tức là 1. Đánh dấu nó là đã duyệt bằng cách thêm nó vào danh sách visited. Bây giờ hãy tìm các nút liền kề của 1. Vì 0 đã có trong danh sách được truy cập, ta bỏ qua nó và truy cập đỉnh của ngăn xếp lúc này, tức là 2



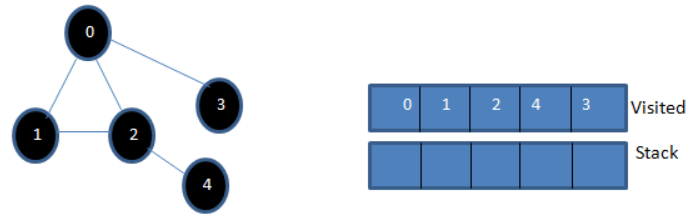
Tiếp theo, đánh dấu nút 2 là đã duyệt. Nút 4 liền kề của nó được thêm vào ngăn xếp.



Tiếp theo, đánh dấu 4 là đỉnh của ngăn xếp như đã duyệt. Nút 4 chỉ có nút 2 là nút liền kề của nó và nút 2 đã duyệt, do đó ta bỏ qua nó.



Ở giai đoạn này, chỉ có nút 3 hiện diện trong ngăn xếp. Nút 0 liền kề của nó đã được truy cập, do đó ta bỏ qua nó. Bây giờ ta đánh dấu 3 là đã duyệt.



Bây giờ ngăn xếp trống và danh sách đã duyệt hiển thị trình tự của đường đi duyệt theo chiều sâu của biểu đồ đã cho.

Nếu chúng ta quan sát đồ thị đã cho và trình tự duyệt, chúng ta nhận thấy rằng đối với thuật toán DFS, chúng ta thực sự duyệt qua độ sâu của đồ thị và sau đó quay ngược lại nó một lần nữa để khám phá các nút mới.

1. **Viết chương trình C++ Demo**

* **Tạo lớp đồ thị cho duyệt đồ thị theo DFS**

**//Lớp đồ thị để duyệt theo DFS**

**class DFSGraph**

**{**

**int V; // Số đỉnh của đồ thị**

**list<int> \*adjList; // Danh sách liền kề**

**void DFS\_util(int v, bool visited[]); //Hàm được dùng bởi DFS**

**public:**

**// Hàm tạo Đồ thị**

**DFSGraph(int V)**

**{**

**this->V = V;**

**adjList = new list<int>[V];**

**}**

**// Hàm thêm cạnh vào đồ thị**

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

**adjList[v].push\_back(w); // Thêm w vào danh sách của v**

**}**

**void DFS(); // Hàm duyệt đồ thị theo DFS**

**};**

* **Gọi hàm duyệt đồ thị theo DFS**

**void DFSGraph::DFS\_util(int v, bool visited[])**

**{**

**// Nút hiện tại đã được duyệt**

**visited[v] = true;**

**cout << v << " ";**

**// Xử lý đệ quy tất cả các nút liền kề của nút hiện tại**

**list<int>::iterator i;**

**for(i = adjList[v].begin(); i != adjList[v].end(); ++i)**

**if(!visited[\*i])**

**DFS\_util(\*i, visited);**

**}**

**// Gọi hàm duyệt DFS**

**void DFSGraph::DFS()**

**{**

**// Ban đầu không có nút nào được duyệt**

**bool \*visited = new bool[V];**

**for (int i = 0; i < V; i++)**

**visited[i] = false;**

**// Duyệt từng nút bằng cách gọi đệ quy DFS\_util**

**for (int i = 0; i < V; i++)**

**if (visited[i] == false)**

**DFS\_util(i, visited);**

**}**

* **Main** : Nhập đồ thị như ví dụ ở trên để kiểm tra kết quả có trùng khớp với kết quả lý thuyết

**int main()**

**{**

**// Tạo đồ thị**

**DFSGraph gdfs(5);**

**gdfs.addEdge(0, 1);**

**gdfs.addEdge(0, 2);**

**gdfs.addEdge(0, 3);**

**gdfs.addEdge(1, 2);**

**gdfs.addEdge(2, 4);**

**gdfs.addEdge(3, 3);**

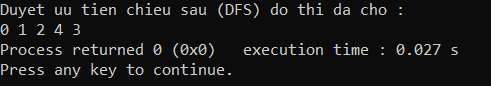
**gdfs.addEdge(4, 4);**

**cout << "Duyet uu tien chieu sau (DFS) do thi da cho :"<<endl;**

**gdfs.DFS();**

**return 0;**

**}**

1. **Kết quả chạy chương trình : với**

Trùng khớp với kết quả ở trên

**Breadth First Search**

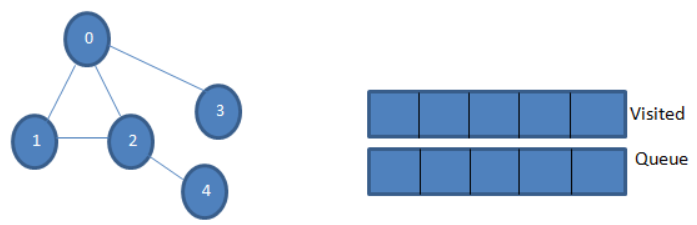
1. **Giới thiệu thuật toán :**

Giải thuật tìm kiếm theo chiều rộng (Breadth First Search – viết tắt là BFS) duyệt qua một đồ thị theo chiều rộng và sử dụng hàng đợi (queue) để ghi nhớ đỉnh liền kề, bắt đầu việc tìm kiếm khi không gặp được đỉnh liền kề trong bất kỳ vòng lặp nào.

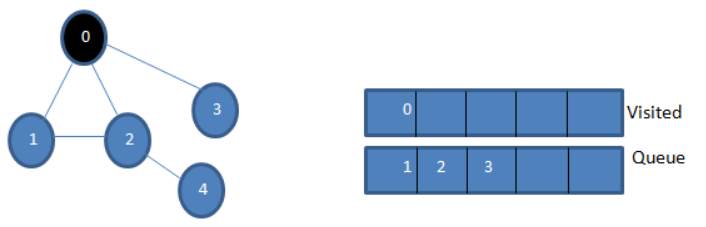
Hãy coi G là một đồ thị mà chúng ta sẽ duyệt qua bằng thuật toán BFS.

* Gọi S là nút gốc / nút bắt đầu của đồ thị.
* Bước 1: Bắt đầu với nút S và xếp nó vào hàng đợi.
* Bước 2: Lặp lại các bước sau cho tất cả các nút trong biểu đồ.
* Bước 3: Dequeue S và xử lý nó.
* Bước 4: Ghép tất cả các nút lân cận của S và xử lý chúng.
* [KẾT THÚC LOOP]
* Bước 6: THOÁT

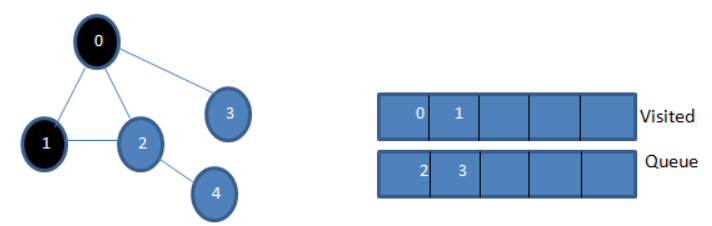
**Ví dụ :** Cho đồ thị như hình dưới đây, tạo 2 nhóm , visited dùng để chứa các nút đã duyệt, queue làmhàng đợi để chứa các nút liền kề



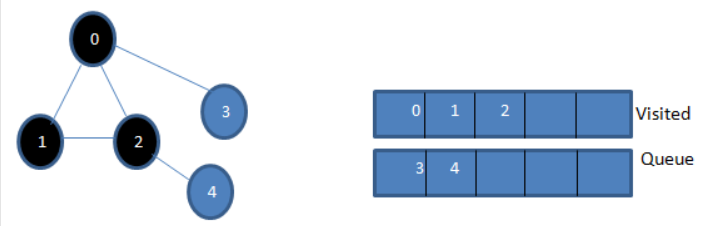
Gọi 0 là nút bắt đầu hoặc nút nguồn. Đầu tiên, xếp nó vào hàng đợi đã duyệt visited và tất cả các nút lân cận của nó trong hàng đợi queue.



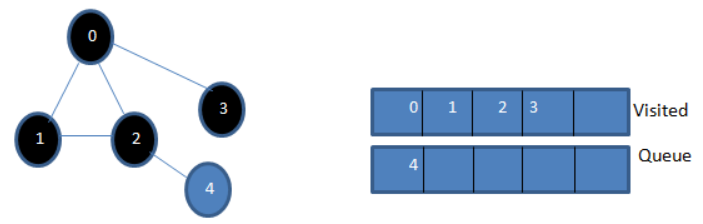
Tiếp theo, lấy một trong các nút liền kề để xử lý ở đây là 1. Đánh dấu nó là đã duyệt bằng cách xóa nó khỏi hàng đợi và đưa các nút liền kề của nó vào hàng đợi (2 và 3 đã có trong hàng đợi). Vì 0 đã được truy cập, ta bỏ qua nó.



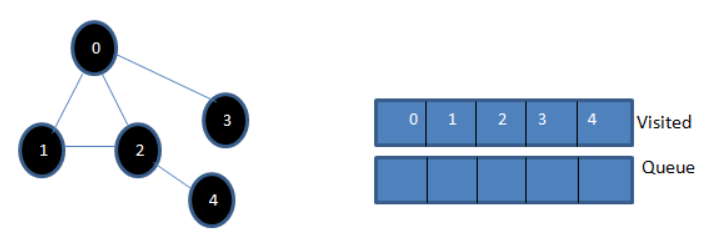
Tiếp theo, xóa nút 2 trong hàng đợi và đánh dấu nó là duyệt. Sau đó, nút 4 liền kề của nó được thêm vào hàng đợi.



Tiếp theo, ta xét nút 3 từ hàng đợi và đánh dấu nó là đã duyệt. Nút 3 chỉ có một nút liền kề, tức là nút 0, đã được duyệt. Do đó, ta bỏ qua nó.



Ở giai đoạn này, chỉ có nút 4 hiện diện trong hàng đợi. Nút 2 liền kề của nó đã duyệt, do đó ta bỏ qua nó. Bây giờ, ta đánh dấu 4 là đã duyệt.



Tiếp theo, trình tự hiện diện trong danh sách được duyệt visited là trình tự tìm kiếm ưu tiên chiều rộng của đồ thị đã cho

Nếu chúng ta quan sát đồ thị đã cho và trình tự duyệt, chúng ta có thể nhận thấy rằng đối với thuật toán BFS, chúng ta thực sự duyệt qua bề rộng của đồ thị và sau đó chuyển sang cấp độ tiếp theo.

1. **Viết chương trình C++ Demo**

* **Tạo lớp đồ thị để duyệt theo BFS**

**//Lớp này đại diện cho một đồ thị có hướng bằng cách sử dụng**

**// biểu diễn danh sách liềnkề**

**class Graph**

**{**

**int V; // Số đỉnh**

**// Con trỏ trỏ đến một mảng chứa danh sách liền kề**

**list<int> \*adj;**

**public:**

**Graph(int V); // Hàm tạo để thêm một cạnh vào đồ thị**

**void addEdge(int v, int w);**

**// in duyệt BFS từ một nguồn nhất định s**

**void BFS(int s);**

**};**

* **Gọi hàm tìm kiếm ưu tiên chiều rộng (BFS)**

**Graph::Graph(int V)**

**{**

**this->V = V;**

**adj = new list<int>[V];**

**}**

**void Graph::addEdge(int v, int w)**

**{**

**adj[v].push\_back(w); // Thêm w vào danh sách của v.**

**}**

**void Graph::BFS(int s)**

**{**

**// Đánh dấu tất cả các đỉnh là chưa được duyệt**

**bool \*visited = new bool[V];**

**for(int i = 0; i < V; i++)**

**visited[i] = false;**

**// Tạo hàng chờ cho BFS**

**list<int> queue;**

**// Đánh dấu các đỉnh hiện tại là đã được duyệt và thêm nó vào hàng đợi**

**visited[s] = true;**

**queue.push\_back(s);**

**// 'i' sẽ được sử dụng để lấy tất cả các đỉnh liền kề của một đỉnh**

**list<int>::iterator i;**

**while(!queue.empty())**

**{**

**// Xếp hàng đợi một đỉnh từ hàng đợi và in nó**

**s = queue.front();**

**cout << s << " ";**

**queue.pop\_front();**

**// Lấy tất cả các đỉnh liền kề của đỉnh s. Nếu một vùng liền kề chưa được truy cập, đánh dấu nó đã được truy cập và xếp vào hàng chờ**

**for (i = adj[s].begin(); i != adj[s].end(); ++i)**

**{**

**if (!visited[\*i])**

**{**

**visited[\*i] = true;**

**queue.push\_back(\*i);**

**}**

**}**

**}**

**}**

* **Main :** Nhập đồ thị như ví dụ ở trên để kiểm tra kết quả có trùng khớp với kết quả lý thuyết

**//Chương trình điều khiển để kiểm tra các phương thức của lớp đồ thị**

**int main()**

**{**

**// Tạo một đồ thị đã cho như trong sơ đồ trên**

**Graph g(5);**

**g.addEdge(0, 1);**

**g.addEdge(0, 2);**

**g.addEdge(0, 3);**

**g.addEdge(1, 2);**

**g.addEdge(2, 4);**

**g.addEdge(3, 3);**

**g.addEdge(4, 4);**

**cout << "Duyet do thi da cho theo thuat toan Tim kiem uu tien chieu rong (BFS) "**

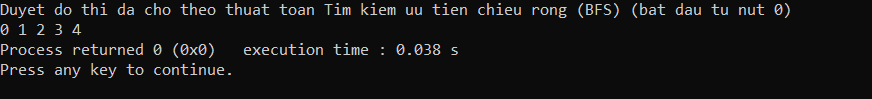
**<< "(bat dau tu nut 0) \n";**

**g.BFS(0);**

**return 0;**

**}**

1. **Kết quả chạy trương trình :**

****

Trùng khớp với kết quả ở trên

**So sánh BFS Và DFS**

| **BFS** | **DFS** |
| --- | --- |
| Viết tắt của “Tìm kiếm theo chiều rộng” | Viết tắt của "Tìm kiếm theo chiều sâu" |
| Các nút được khám phá theo chiều rộng theo cấp độ. | Các nút được khám phá theo chiều sâu cho đến khi chỉ còn các nút lá và sau đó được kiểm tra lại để khám phá các nút chưa được thăm dò khác. |
| BFS được thực hiện với sự trợ giúp của cấu trúc dữ liệu hàng đợi. | DFS được thực hiện với sự trợ giúp của cấu trúc dữ liệu ngăn xếp. |
| Hiệu suất chậm hơn. | Nhanh hơn BFS. |
| Hữu ích trong việc tìm đường đi ngắn nhất giữa hai nút. | Được sử dụng chủ yếu để phát hiện các chu trình trong đồ thị. |

**Các Ứng Dụng Của DFS**

* **Phát hiện các chu trình trong đồ thị:** Nếu chúng ta tìm thấy một cạnh phía sau trong khi thực hiện DFS trong một đồ thị thì chúng ta có thể kết luận rằng đồ thị có một chu trình. Do đó DFS được sử dụng để phát hiện các chu trình trong đồ thị.
* **Tìm đường :** Cho hai đỉnh x và y, chúng ta có thể tìm đường đi giữa x và y bằng DFS. Chúng ta bắt đầu với đỉnh x và sau đó đẩy tất cả các đỉnh trên đường đến ngăn xếp cho đến khi chúng ta gặp y. Nội dung của ngăn xếp cung cấp đường dẫn giữa x và y.
* **Cây bao trùm tối thiểu và đường đi ngắn nhất:** Việc duyệt DFS của đồ thị không có trọng số cho chúng ta một cây bao trùm tối thiểu và đường đi ngắn nhất giữa các nút.
* **Sắp xếp theo cấu trúc liên kết:** Chúng tôi sử dụng tính năng sắp xếp theo cấu trúc liên kết khi chúng tôi cần lập lịch trình các công việc từ các phụ thuộc đã cho giữa các công việc. Trong lĩnh vực khoa học máy tính, chúng tôi chủ yếu sử dụng nó để giải quyết các phụ thuộc ký hiệu trong trình liên kết, tuần tự hóa dữ liệu, lập lịch hướng dẫn, v.v. DFS được sử dụng rộng rãi trong sắp xếp theo cấu trúc liên kết.