BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**SỬ DỤNG CÔNG CỤ ĐỒ HỌA GRAPHICS.H ĐỂ MÔ PHỎNG GIẢI THUẬT TÌM KIẾM THEO CHIỀU SÂU DEPTH FIRST SEARCH**

GVHD : Th.S. Đoàn Vũ Thịnh

SVTH : Đặng Hoàng Lâm

MSSV : 63134410

Lớp : 63.CNTT-3

Khánh Hòa, tháng 01 năm 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**SỬ DỤNG CÔNG CỤ ĐỒ HỌA GRAPHICS.H ĐỂ MÔ PHỎNG GIẢI THUẬT TÌM KIẾM THEO CHIỀU SÂU DEPTH FIRST SEARCH**

GVHD : Th.S. Đoàn Vũ Thịnh

SVTH : Đặng Hoàng Lâm

MSSV : 63134410

Lớp : 63.CNTT-3

Khánh Hòa, tháng 01 năm 2024

i

**MỤC LỤC**

**Chương 1. TỔNG QUAN 6**

1.1 MỘT SỐ KIẾN THỨC CƠ BẢN VỀ THUẬT TOÁN 6

1.1.1 Khái niệm bài toán Tin học 6

1.1.2 Khái niệm thuật toán 6

1.1.3 Các tính chất của thuật toán 7

1.1.4 Thuật toán tìm kiếm 7

1.2 CÔNG CỤ LẬP TRÌNH DEV C++ VÀ THƯ VIỆN GRAPHICS.H 8

1.2.1 Dev C++ IDE 8

1.2.2 Thư viện đồ họa graphics.h 9

1.3 MỤC ĐÍCH CỦA ĐỀ TÀI 10

**Chương 2. CƠ SỞ LÝ THUYẾT 11**

2.1 MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ ĐỒ THỊ 11

2.1.1 Khái niệm đồ thị 11

2.1.2 Các khái niệm cơ bản 11

2.2 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU DEPTH FIRST SEARCH(DFS ) 13

2.2.1 Phát biểu bài toán 13

2.2.2 Cách thức hoạt động của DFS và ứng dụng 14

2.2.3 Độ phức tạp thuật toán DFS 15

2.2.4 Ưu nhược điểm của giải thuật DFS 15

2.3 MÔ PHỎNG THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU( DFS) ĐỂ TÌM ĐƯỜNG ĐI TỪ ĐỈNH NÀY ĐẾN ĐỈNH KHÁC 16

2.3.1 Tạo khung chương trình mô phỏng 16

2.3.2 Khai báo các biến, file đồ thị để cài đặt và mô phỏng thuật toán 17

2.3.3 Đọc dữ liệu từ file để khởi tạo đồ thị 19

2.3.4 Thuật toán và ứng dụng đồ họa để thể hiện thuật toán 20

**Chương 3: KẾT QUẢ NGHIÊN CỨU 22**

3.1 KẾT QUẢ MÔ PHỎNG TRÊN DEV C++ BẰNG THƯ VIỆN GRAPHICS.H 23

3.1.1 Hiển thị khung 23

3.1.2 Hiển thị đồ thị 24

3.1.3 Hiển thị mô phỏng kết quả duyệt DFS theo yêu cầu bài toán 26

3.1.4 Các chức năng khác 28

**Chương 4: KẾT LUẬN 30**

**TÀI LIỆU THAM KHẢO 32**

**PHỤ LỤC 32**

ii

**DANH MỤC HÌNH**

Hình 2.1 Đồ thị vô hướng 11

Hình 2.2 Đồ thị có hướng 11

Hình 2.3 Đơn đồ thị 12

Hình 2.4 Đa đồ thị 12

Hình 2.5 Đồ thị vô hướng và đồ thị có hướng 13

Hình 2.6 Tạo khung chương trình mô phỏng 17

Hình 2.7 Các biến phục vụ cho thuật toán và đồ họa 18

Hình 2.8 Cấu trúc file dữ liệu đồ thị 19

Hình 2.9 Đọc dữ liệu từ file và khởi tạo đồ thị 20

Hình 2.10 Hàm mô phỏng và sử dụng thuật toán 22

Hình 3.1 Các biến đánh dấu 23

Hình 3.2 Hiển thị khung mô phỏng 23

Hình 3.3 Hàm main 24

Hình 3.4 Hiển thị đồ thị 24

Hình 3.5 Code nạp file từ cửa sổ file explore 25

Hình 3.6 Thông báo không tồn tại đồ thị trong khung 26

Hình 3.7 Nhập dữ liệu từ bàn phím 26

Hình 3.8 Mô phỏng duyệt DFS 27

Hình 3.9 Mô phỏng chính xác yêu cầu bài toán 27

Hình 3.10 Code thực hiện nút Starting 28

Hình 3.11 Tác giả 29

Hình 3.12 Trước khi Reset 29

Hình 3.13 Sau khi Reset 29

Hình 3.14 Trước khi thực hiên delete 30

Hình 3.15 Sau khi thực hiện delete 30

**Chương 1. TỔNG QUAN**

**1.1 MỘT SỐ KIẾN THỨC CƠ BẢN VỀ THUẬT TOÁN**

**1.1.1 Khái niệm bài toán tin học**

Trong phạm vi tin học, người ta quan niệm bài toán là một công việc nào đó mà con người muốn máy tính thực hiện.

Khi dùng máy tính để giải bài toán, ta cần quan tâm tới 2 vấn đề: Dữ liệu cần được đưa vào máy tính (Input) là gì và cần lấy ra (Output) thông tin gì? Nói một cách khác, cho một bài toán là việc mô tả rõ Input và Output của bài toán.

Vấn đề còn lại là: Làm thế nào để từ Input ta có được Output ? [1]

**1.1.2 Khái niệm thuật toán**

Khác với Toán học (các yêu cầu của bài toán thường là chứng minh sự tồn tại đáp án chứ không yêu cầu tìm một cách chi tiết để tìm ra đáp án đó), giải một bài toán Tin học là việc đi tìm một lời giải cụ thể, tường minh để đưa ra Output của bài toán dựa trên Input đã cho. Việc chỉ ra một cách tìm Output của bài toán được gọi là một thuật toán. Có nhiều cách phát biểu khái niệm về thuật toán. Dưới đây là cách phát biểu được chọn để đưa vào sách giáo khoa Tin học phổ thông:

***Khái niệm về thuật toán: thuật toán là một dãy hữu hạn các thao tác được sắp xếp theo một trình tự nhất định để sau khi thực hiện dãy các thao tác đó, từ input ta có output cần tìm***

Trong lĩnh vực máy tính, cụm từ “thuật toán” đôi khi người ta dùng bằng một từ khác: “giải thuật”.

Ví dụ về một thuật toán: Nhập vào một số nguyên dương N, kiểm tra số đó có là số nguyên tố hay không?

*Lời giải:*

Input: Số nguyên dương N.

Output: Có/không tương ứng với N có là nguyên tố hay không?

Ý tưởng: Một số nguyên gọi là nguyên tố khi nó chỉ có ước là 1 và chính nó. Từ định nghĩa suy ra:

* Nếu N = 1 thì thông báo là N không nguyên tố rồi kết thúc;
* Nếu 1< N < 4 thì thông báo N là số nguyên tố rồi kết thúc;
* Nếu N ≥ 4 và không có ước trong khoảng từ 2 đến [ N] thì N là nguyên tố.

Thuật toán:Có nhiều cách mô phỏng khác nhau. Dưới đây là cách mô phỏng thuật toán dạng liệt kê các bước:

Bước 1. Nhập số nguyên dương N;

Bước 2. Nếu N = 1 thì thông báo là N không nguyên tố rồi kết thúc;

Bước 3. Nếu N < 4 thì thông báo N là số nguyên tố rồi kết thúc;

Bước 4. i ← 2;

Bước 5. Nếu (\*) i > [ N ] thì thông báo N là nguyên tố rồi kết thúc;

Bước 6. Nếu N chia hết cho i thì thông báo N không nguyên tố rồi kết thúc;

Bước 7. i ← i + 1 rồi quay lại bước 5; [1]

**1.1.3 Các tính chất của thuật toán**

Dựa trên khái niệm về thuật toán và ví dụ ở trên ta thấy các thao tác trong thuật toán phải được mô tả đủ chi tiết để một đối tượng cứ tiến hành thực hiện theo đúng thứ tự các thao tác đó là có thể cho ra output dựa trên input tương ứng. Một thuật toán phải đảm bảo được các tính chất sau:

Tính xác định: Sau khi thực hiện một thao tác thì hoặc là thuật toán kết thúc hoặc là có đúng một thao tác xác định để thực hiện tiếp theo.

Tính đúng đắn: Sau khi thực hiện thuật toán ta phải nhận được đúng Output cần tìm.

Tính dừng: Thuật toán phải kết thúc sau một số hữu hạn lần thực hiện.

Tính tổng quát: Thuật toán là đúng đắn với mọi bộ dữ liệu đầu vào của bài toán.

Tính hiệu quả:

- Hiệu quả về thời gian: Ta quan tâm tới thời gian cần thiết để thực hiện xong thuật toán đó. Thời gian đó phải nằm trong giới hạn cho phép.

- Hiệu quả về không gian: Dung lượng bộ nhớ cần thiết để lưu trữ các đối tượng như bộ Input, bộ Output, kết quả trung gian và chương trình được dùng để thực hiện thuật toán.

- Dễ cài đặt: thuật toán đó liệu có chuyển được thành chương trình bằng một ngôn ngữ lập trình nào đó hay không. Trước khi xây dựng thuật toán cho một bài toán nào đó, trước tiên phải xác định được Input và Output là gì, thử trên một số ví dụ cụ thể để định hướng cho việc xây dựng thuật toán.[1]

**1.1.4 Thuật toán tìm kiếm**

Thuật toán tìm kiếm là một quy trình hoặc phương pháp được thiết kế để tìm kiếm một giá trị hoặc một tập giá trị trong một tập dữ liệu có cấu trúc nhất định. Thuật toán tìm kiếm thường được áp dụng trong các bài toán như tìm kiếm một phần tử trong mảng, tìm kiếm đường đi trong đồ thị, tìm kiếm từ khóa trong cơ sở dữ liệu, v.v.

Có nhiều thuật toán tìm kiếm khác nhau được sử dụng tùy thuộc vào đặc điểm của dữ liệu và mục tiêu tìm kiếm. Dưới đây là một số khái niệm cơ bản về một số thuật toán tìm kiếm phổ biến:

1. Tìm kiếm tuần tự (Sequential Search): Phương pháp tìm kiếm tuần tự duyệt qua từng phần tử của dữ liệu cho đến khi tìm thấy giá trị cần tìm hoặc duyệt hết toàn bộ dữ liệu. Đây là phương pháp đơn giản nhưng không hiệu quả cho các tập dữ liệu lớn.
2. Tìm kiếm nhị phân (Binary Search): Áp dụng cho các tập dữ liệu đã được sắp xếp theo thứ tự tăng dần (hoặc giảm dần). Phương pháp này chia tập dữ liệu thành hai phần, so sánh giá trị cần tìm với giá trị ở phần giữa, và tiếp tục chia nhỏ phần tìm kiếm cho đến khi tìm thấy giá trị cần tìm hoặc xác định không có trong tập dữ liệu.
3. Tìm kiếm theo chiều sâu (Depth-First Search - DFS): Là một thuật toán tìm kiếm trong đồ thị, bắt đầu từ một đỉnh gốc và duyệt qua các đỉnh kề với đỉnh hiện tại. Thuật toán tiếp tục tìm kiếm theo chiều sâu cho đến khi không còn đỉnh nào để duyệt hoặc tìm thấy đỉnh cần tìm. DFS thông thường được thực hiện bằng cách sử dụng đệ quy hoặc ngăn xếp (stack) để lưu trữ các đỉnh.
4. Tìm kiếm theo chiều rộng (Breadth-First Search - BFS): Tương tự như DFS, BFS là thuật toán tìm kiếm trong đồ thị, nhưng thay vì duyệt qua các đỉnh kề trước, BFS duyệt qua tất cả các đỉnh cùng cấp trước khi đi xuống cấp tiếp theo. BFS thường được thực hiện bằng cách sử dụng hàng đợi (queue) để lưu trữ các đỉnh.

Các thuật toán tìm kiếm khác nhau có đặc điểm và ứng dụng riêng. Trong đề tài này tôi lựa chọn thuật toán Depht First Search(DFS ) để mô phỏng thông qua thư viện đồ họa graphics.h [1]

**1.2 CÔNG CỤ LẬP TRÌNH DEV C++ VÀ THƯ VIỆN GRAPHICS.H**

**1.2.1 DEV C++ IDE**

Dev C++ là một môi trường phát triển tích hợp (Integrated Development Environment – IDE ) được sử dụng chủ yếu để lập trình bằng ngôn ngữ C++. Đây là một IDE miễn phí và mã nguồn mở, cung cấp một loạt các công cụ hỗ trợ viết và biên dịch mã nguồn C++.

Ưu điểm của Dev C++ trong việc phát triển phần mềm C++:

1. Dễ sử dụng: Dev C++ có giao diện người dùng thân thiện và dễ sử dụng, đặc biệt phù hợp cho người mới bắt đầu học lập trình C++.
2. Hỗ trợ biên dịch: Dev C++ đi kèm với trình biên dịch MinGW, cho phép biên dịch và chạy mã nguồn C++ một cách dễ dàng.
3. Tích hợp nhiều công cụ: Dev C++ cung cấp nhiều công cụ hữu ích như trình biên dịch, trình gỡ lỗi, trình soạn thảo mã nguồn và bộ biên dịch đồ họa.
4. Cú pháp màu: IDE cung cấp cú pháp màu cho mã nguồn, giúp làm nổi bật các thành phần của ngôn ngữ và dễ dàng đọc và sửa lỗi.
5. Hỗ trợ đa nền tảng: Dev C++ có thể chạy trên nhiều hệ điều hành như Windows, Linux và macOS.

Các tính năng chính của Dev C++:

1. Trình soạn thảo mã nguồn: Dev C++ cung cấp một trình soạn thảo mã nguồn với các tính năng như kiểm tra cú pháp, tự động xếp dòng, gợi ý từ khóa và dấu ngoặc đóng mở.
2. Trình biên dịch: Dev C++ sử dụng bộ biên dịch MinGW, cho phép biên dịch và chạy mã nguồn C++.
3. Trình gỡ lỗi: IDE hỗ trợ chế độ gỡ lỗi để tìm và sửa lỗi trong mã nguồn.
4. Quản lý dự án: Dev C++ cho phép tạo và quản lý các dự án lập trình, giúp tổ chức mã nguồn và tài liệu liên quan.

Cách sử dụng Dev C++ để viết và biên dịch mã nguồn C++:

1. Tạo một tệp mới: Chọn "File" -> "New" -> "Source file" để tạo một tệp mã nguồn mới.
2. Viết mã nguồn: Sử dụng trình soạn thảo mã nguồn để viết mã nguồn C++.
3. Lưu tệp mã nguồn: Sử dụng "File" -> "Save" để lưu tệp mã nguồn với phần mở rộng ".cpp".
4. Biên dịch mã nguồn: Chọn "Execute" -> "Compile" để biên dịch mã nguồn thành tệp thực thi.
5. Chạy chương trình: Chọn "Execute" -> "Run" để chạy chương trình đã biên dịch.
6. Gỡ lỗi: Nếu có lỗi trong mã nguồn, sử dụng chế độ gỡ lỗi để xác định và sửa lỗi.

Lưu ý: Dev C++ là một IDE phổ biến và dễ sử dụng. Tuy nhiên, nó đã không được cập nhật trong một thời gian dài và có thể có một số vấn đề liên quan đến tính ổn định và bảo mật. Do đó, khi phát triển phần mềm C++, bạn cũng có thể xem xét sử dụng các IDE khác như Visual Studio, Code::Blocks, hoặc CLion, tùy thuộc vào nhu cầu và yêu cầu của dự án.

**1.2.2 Thư viện đồ họa graphics.h**

Thư viện đồ họa graphics.h là một thư viện đồ họa cơ bản trong C++, được sử dụng để tạo các ứng dụng đồ họa đơn giản trên các nền tảng Windows. Nó cung cấp các chức năng để vẽ các hình học, văn bản và các đối tượng đồ họa khác trên màn hình.

Vai trò của thư viện graphics.h trong việc tạo các ứng dụng đồ họa C++:

1. Đơn giản hóa việc vẽ đồ họa: graphics.h cung cấp các hàm đơn giản để vẽ các hình học cơ bản như đường thẳng, hình tròn, hình chữ nhật và hình tam giác. Điều này giúp đơn giản hóa quá trình vẽ đồ họa trong ứng dụng của bạn.
2. Tạo giao diện đồ họa người dùng: Với graphics.h, bạn có thể tạo các giao diện đồ họa người dùng đơn giản bằng cách vẽ các nút, ô nhập liệu và các thành phần khác trên màn hình. Điều này cho phép người dùng tương tác với ứng dụng của bạn thông qua các phím và chuột.
3. Hiệu ứng đồ họa và hoạt hình: Thư viện graphics.h cung cấp các chức năng để tạo hiệu ứng đồ họa đơn giản như di chuyển, xoay, phóng to và thu nhỏ các đối tượng đồ họa. Điều này cho phép bạn tạo ra các hoạt hình đơn giản và tạo ra hiệu ứng đồ họa đẹp mắt trong ứng dụng của bạn.

Các chức năng cơ bản và cách sử dụng trong graphics.h để vẽ đồ thị và các đối tượng đồ họa khác:

1. Khởi tạo môi trường đồ họa: Sử dụng hàm initgraph để khởi tạo môi trường đồ họa. Ví dụ: int gdriver = DETECT, gmode; initgraph(&gdriver, &gmode, "");.
2. Vẽ các hình học cơ bản: graphics.h cung cấp các hàm như line, circle, rectangle, ellipse để vẽ các hình học cơ bản trên màn hình. Ví dụ: line(x1, y1, x2, y2);.
3. Hiển thị văn bản: Sử dụng hàm outtext hoặc outtextxy để hiển thị văn bản lên màn hình đồ họa. Ví dụ: outtext("Hello, World!");.
4. Tạo màu sắc: Sử dụng các hàm như setcolor, setbkcolor để đặt màu cho các hình học và nền của màn hình. Ví dụ: setcolor(RED);.
5. Tạo đối tượng đồ họa: Sử dụng các hàm như bar, floodfill để tạo các đối tượng đồ họa như hình chữ nhật, đa giác và điền màu trong các hình dạng. Ví dụ: bar(left, top, right, bottom);.
6. Ghi chú thư viện: Sử dụng hàm closegraph để đóng môi trường đồ họa khi kết thúc ứng dụng. Ví dụ: closegraph();.

**1.3 MỤC ĐÍCH CỦA ĐỀ TÀI**

Mục đích của đề tài "Sử dụng công cụ đồ họa graphics.h để mô phỏng giải thuật tìm kiếm theo chiều sâu (Depth First Search)" là tạo ra một ứng dụng đồ họa hấp dẫn và trực quan, cho phép người dùng hiểu rõ hơn về cách hoạt động của giải thuật tìm kiếm theo chiều sâu.

Bằng cách sử dụng thư viện graphics.h, chúng ta có thể biểu diễn đồ thị và các nút trong đồ thị dưới dạng hình ảnh và hình học trên màn hình. Mục tiêu là tạo ra một giao diện trực quan mô phỏng quá trình tìm kiếm theo chiều sâu, cho phép người dùng thấy được cách các nút trong đồ thị được khám phá theo thứ tự và cách kết quả tìm kiếm được truyền lại.

Việc sử dụng công cụ đồ họa graphics.h giúp chúng ta tạo ra một môi trường thân thiện và dễ hiểu, nơi người dùng có thể tương tác trực tiếp với đồ thị và quan sát quá trình tìm kiếm diễn ra. Điều này giúp cải thiện sự tương tác và tăng cường sự hiểu biết về giải thuật tìm kiếm theo chiều sâu.

Mục đích cuối cùng là truyền tải kiến thức và khám phá quá trình tìm kiếm theo chiều sâu một cách thú vị và dễ dàng nhất. Bằng cách kết hợp giữa khả năng biểu diễn đồ thị và tương tác đồ họa, tôi hy vọng rằng đề tài này sẽ giúp người dùng hiểu rõ hơn về giải thuật tìm kiếm theo chiều sâu và áp dụng nó vào các vấn đề thực tế khác trong lĩnh vực khoa học máy tính và trí tuệ nhân tạo

**Chương 2. CƠ SỞ LÝ THUYẾT**

**2.1 MỘT SỐ KHÁI NIỆM CƠ BẢN VỀ ĐỒ THỊ**

**2.1.1 Khái niệm đồ thị**

Là một cấu trúc rời rạc gồm các đỉnh và các cạnh nối các đỉnh đó. Được mô tả hình thức: G = (V, E). Trong đó:

V gọi là tập các đỉnh (Vertices) và E gọi là tập các cạnh (Edges). Có thể coi E là tập các cặp (u, v) với u và v là hai đỉnh của V. Một số hình ảnh của đồ thị:

A diagram of a network

Description automatically generatedA hexagon with colored circles and lines

Description automatically generated

**Hình 2.1 Đồ thị vô hướng**  **Hình 2.2 Đồ thị có hướng**

**2.1.2 Các khái niệm cơ bản**

Có thể phân loại đồ thị theo đặc tính và số lượng của tập các cạnh E: Cho đồ thị G = (V, E). Ta có một số khái niệm sau:

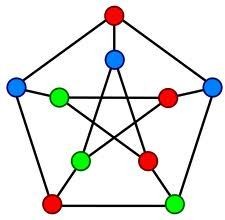
**-** Đơn đồ thị:G được gọi là đơn đồ thị nếu giữa hai đỉnh u, v của V có nhiều nhất là 1 cạnh trong E nối từ u tới v.

**-** Đa đồ thị: G được gọi là đa đồ thị nếu giữa hai đỉnh u, v của V có thể có nhiều hơn 1 cạnh trong E nối từ u tới v.

**-** Đồ thị vô hướng:G được gọi là đồ thị vô hướng nếu các cạnh trong E là không định hướng, tức là cạnh nối hai đỉnh u, v bất kỳ cũng là cạnh nối hai đỉnh v, u. Hay nói cách khác, tập E gồm các cặp (u, v) không tính thứ tự (u, v)  (v, u)

**-** Đồ thị có hướng:G được gọi là đồ thị có hướng nếu các cạnh trong E là có định hướng, có thể có cạnh nối từ đỉnh u tới đỉnh v nhưng chưa chắc đã có cạnh nối từ đỉnh v tới đỉnh u. Hay nói cách khác, tập E gồm các cặp (u, v) có tính thứ tự: (u, v)  (v, u). Trong đồ thị có hướng, các cạnh được gọi là các cung. Đồ thị vô hướng cũng có thể coi là đồ thị có hướng nếu như ta coi cạnh nối hai đỉnh u, v bất kỳ tương đương với hai cung (u, v) và (v, u).

Ví dụ:



3

1

2

5

4

Vô hướng Có hướng

A diagram of a triangle

Description automatically generated**Hình 2.3 Đơn đồ thị**

A triangle with red dots

Description automatically generated

Có hướng Vô hướng

**Hình 2.4 Đa đồ thị**

Cạnh liên thuộc, đỉnh kề, bậc:

Đối với đồ thị vô hướng G = (V, E). Xét một cạnh e  E, nếu e = (u, v) thì ta nói hai đỉnh u và v là kề nhau (adjacent) và cạnh e này liên thuộc (incident) với đỉnh u và đỉnh v.

Với một đỉnh v trong đồ thị, ta định nghĩa bậc (degree) của v, ký hiệu deg(v) là số cạnh liên thuộc với v. Dễ thấy rằng trên đơn đồ thị thì số cạnh liên thuộc với v cũng là số đỉnh kề với v.

Đối với đồ thị có hướng G = (V, E). Xét một cung e  E, nếu e = (u, v) thì ta nói u nối tới v và v nối từ u, cung e là đi ra khỏi đỉnh u và đi vào đỉnh v. Đỉnh u khi đó được gọi là đỉnh đầu, đỉnh v được gọi là đỉnh cuối của cung e.

Với mỗi đỉnh v trong đồ thị có hướng, ta định nghĩa: Bán bậc ra của v ký hiệu deg+(v) là số cung đi ra khỏi nó; bán bậc vào ký hiệu deg-(v) là số cung đi vào đỉnh đó

Đường đi:Một đường đi độ dài *k* từ đỉnh u đến đỉnh v là dãy (u = *x0*, *x1*, ..., *xk* = v) thoả mãn (xi, xi+1)  E (là 1 cạnh của đồ thị) với i: (0  i  k). Đỉnh u gọi là đỉnh xuất phát, v gọi là đỉnh kết thúc của đường đi. Đường đi không có cạnh nào đi qua hơn 1 lần gọi là đường đi đơn.

Chu trình:Đường đi có đỉnh xuất phát trùng với đỉnh kết thúc gọi là chu trình.tương tự ta có khái niệm chu trình đơn.[1]

**2.2 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU DEPTH FIRST SEARCH(DFS)**

**2.2.1 Phát biểu bài toán**

Cho đồ thị G = (V, E) và s và t là hai đỉnh của đồ thị.

Yêu cầu: Hãy chỉ ra một đường đi từ s đến t (nếu có).

A diagram of a network

Description automatically generatedA diagram of a triangle with green circles and arrows

Description automatically generated*Ví dụ: Xét một đồ thị vô hướng và một đồ thị có hướng dưới đây:*

**Hình 2.5 Đồ thị vô hướng và đồ thị có hướng**

Trên cả hai đồ thị, (1, 2, 3, 4) là đường đi đơn độ dài 3 từ đỉnh 1 tới đỉnh

4. Bởi (1, 2) (2, 3) và (3, 4) đều là các cạnh (hay cung).

Làm sao để duyệt tất cả các đỉnh có thể đến được từ một đỉnh xuất phát nào đó? Vấn đề này đưa về một bài toán liệt kê mà yêu cầu của nó là không được bỏ sót hay lặp lại bất kỳ đỉnh nào. Vì vậy, cần phải xây dựng những thuật toán cho phép duyệt một cách hệ thống các đỉnh, những thuật toán như vậy gọi là những thuật toán tìm kiếm trên đồ thị. Trong lý thuyết đồ thị, người ta quan tâm đến hai thuật toán cơ bản nhất: thuật toán tìm kiếm theo chiều sâu và thuật toán tìm kiếm theo chiều rộng và trong đề tài này tôi sẽ trình bày bài toán cũng như hướng giải quyết bài toán theo thuật toán tìm kiếm theo chiều sâu Depth First Search.[1]

**2.2.2 Cách thức hoạt động của DFS và ứng dụng**

Thuật toán tìm kiếm theo chiều sâu (DFS - Depth-First-Search) là một thuật toán dùng để tìm kiếm thông qua việc khám phá tất cả các đỉnh của đồ thị theo một hướng cụ thể. Nó bắt đầu từ một đỉnh gốc, liên tục truy cập các đỉnh con kế tiếp và tiếp tục quá trình này cho đến khi không còn đỉnh con nào để truy cập. Khi không có đỉnh con kế tiếp, thuật toán quay lại các đỉnh trước đó và tiếp tục tìm kiếm đỉnh con chưa được thăm.

Dưới đây là cách hoạt động của thuật toán DFS:

Dưới đây là các bước để hiểu cách thuật toán DFS hoạt động bằng cách sử dụng đệ quy:

Bước 1: Chọn một đỉnh khởi đầu và đánh dấu nó là đã thăm.

Bước 2: Thăm đỉnh hiện tại và thực hiện các hoạt động liên quan đến đỉnh đó.

Bước 3: Tìm một đỉnh kế tiếp chưa được thăm:

- Nếu tìm thấy đỉnh kế tiếp chưa được thăm, đánh dấu đỉnh đó là đã thăm và thực hiện đệ quy DFS trên đỉnh đó (quay lại Bước 2).

- Nếu không tìm thấy đỉnh kề kế tiếp nào chưa được ghé thăm, quay lại đỉnh trước đó và tiếp tục tìm kiếm xem có đỉnh kề nào khác chưa được ghé thăm hay không.

Bước 4: Lặp lại Bước 3 cho đến khi tất cả các đỉnh của đồ thị đã được khám phá.

Thuật toán tìm kiếm theo chiều sâu có thể được hiện thực bằng cách sử dụng đệ quy hoặc sử dụng một ngăn xếp để lưu trữ các đỉnh chưa được duyệt. Khi sử dụng đệ quy, thuật toán sẽ tự động duyệt các đỉnh kề với đỉnh hiện tại. Khi sử dụng ngăn xếp, thuật toán sẽ lưu trữ các đỉnh kề chưa được duyệt và duyệt chúng theo thứ tự.

Ứng dụng: Thuật toán DFS (Depth-First-Search) có nhiều ứng dụng trong lĩnh vực khoa học máy tính. Nó được sử dụng để tìm kiếm đường đi giữa các đỉnh trong đồ thị, kiểm tra tính liên thông của đồ thị, kiểm tra chu trình và tìm kiếm các cấu trúc dữ liệu như cây hoặc đồ thị. DFS là một công cụ quan trọng và đa dụng trong việc giải quyết nhiều bài toán thực tế.

**2.2.3 Độ phức tạp thuật toán DFS**

Quá trình tìm kiếm trên đồ thị bắt đầu từ một đỉnh có thể thăm tất cả các đỉnh còn lại, khi đó cách biểu diễn đồ thị có ảnh hưởng lớn tới chi phí về thời gian thực hiện giải thuật:

Trong trường hợp ta biểu diễn đồ thị bằng danh sách kề, DFS có độ phức tạp tính toán là O(n + m) = O(max(n, m)). Đây là cách cài đặt tốt nhất. Nếu ta biểu diễn đồ thị bằng ma trận kề thì độ phức tạp tính toán trong trường hợp này là O(n + n2) = O(n2). Nếu ta biểu diễn đồ thị bằng danh sách cạnh, thao tác duyệt những đỉnh kề với đỉnh u sẽ dẫn tới việc phải duyệt qua toàn bộ danh sách cạnh, đây là cài đặt tồi nhất, nó có độ phức tạp tính toán là O(n.m). [1]

**2.2.4 Ưu nhược điểm của giải thuật DFS**

Giải thuật Depth First Search (DFS) có những ưu nhược điểm sau:

- Ưu điểm của DFS:

Đơn giản và dễ hiểu: DFS là một thuật toán đơn giản và dễ hiểu, đặc biệt khi triển khai bằng đệ quy.

Hiệu quả trong việc tìm kiếm các đường đi: DFS tìm kiếm theo chiều sâu trong đồ thị, giúp nhanh chóng tìm ra một đường đi từ đỉnh xuất phát đến đỉnh đích nếu tồn tại.

Tiết kiệm không gian bộ nhớ: DFS chỉ sử dụng một ngăn xếp (stack) để lưu trữ các đỉnh chờ duyệt, do đó tiết kiệm không gian bộ nhớ so với thuật toán tìm kiếm theo chiều rộng (BFS).

Dễ dàng mở rộng cho các vấn đề khác: DFS có thể dễ dàng mở rộng và áp dụng cho các vấn đề khác nhau như tìm kiếm các thành phần liên thông, tìm kiếm chu trình, kiểm tra tính liên thông của đồ thị, v.v.

- Nhược điểm của DFS:

Không đảm bảo tìm kiếm đường đi ngắn nhất: DFS không đảm bảo tìm kiếm đường đi ngắn nhất từ đỉnh xuất phát đến đỉnh đích. Do thuật toán tìm kiếm theo chiều sâu, nó có thể đi vào một nhánh mà không biết có nhánh khác ngắn hơn.

Có thể rơi vào vòng lặp vô hạn: Nếu đồ thị không hạn chế, tức là có chu trình vô hạn, và không có cơ chế để kiểm tra các đỉnh đã được thăm, DFS có thể rơi vào vòng lặp vô hạn và không kết thúc.

Phụ thuộc vào cấu trúc đồ thị: Kết quả của DFS có thể phụ thuộc vào cấu trúc của đồ thị, bao gồm cả thứ tự các đỉnh và thứ tự duyệt các đỉnh kề. Điều này có thể dẫn đến kết quả không nhất quán khi áp dụng DFS trên các đồ thị khác nhau.

Tóm lại, DFS có những ưu điểm như đơn giản, hiệu quả trong tìm kiếm đường đi và tiết kiệm không gian bộ nhớ. Tuy nhiên, nó cũng có nhược điểm như không đảm bảo tìm kiếm đường đi ngắn nhất, có thể rơi vào vòng lặp vô hạn và phụ thuộc vào cấu trúc đồ thị.

**2.3 MÔ PHỎNG THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU( DFS) ĐỂ TÌM ĐƯỜNG ĐI TỪ ĐỈNH NÀY ĐẾN ĐỈNH KHÁC**

Để làm việc được với thư viện graphics.h trong Dev C++ thì ta cần khai báo thư viện này ngay từ đầu chương trình:

*#include <graphics.h>*

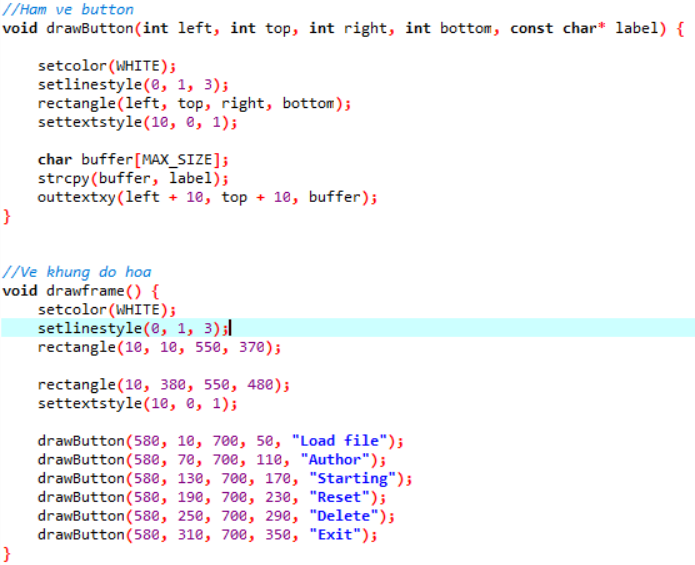
**2.3.1 Tạo khung chương trình mô phỏng**

1. Hàm drawButton:

* Chức năng: Vẽ một nút bấm (button) lên giao diện đồ họa với các thông số được chỉ định.
* Tham số đầu vào:
  + left, top, right, bottom: Tọa độ xác định vị trí và kích thước của nút bấm.
  + label: Chuỗi văn bản chứa nhãn hiển thị trên nút bấm.
* Thao tác thực hiện:
  + Thiết lập màu vẽ thành màu trắng (WHITE).
  + Thiết lập kiểu đường vẽ có độ dày 3 pixel.
  + Vẽ hình chữ nhật với các tọa độ đã cho để tạo khung cho nút bấm.
  + Thiết lập kiểu chữ cho nhãn nút bấm.
  + Sao chép chuỗi label vào bộ đệm buffer.
  + Vẽ chuỗi buffer (nhãn nút bấm) lên giao diện tại vị trí thích hợp.

2. Hàm drawframe:

* Chức năng: Vẽ khung đồ họa tổng thể và các nút bấm chức năng chính cho chương trình.
* Thao tác thực hiện:
  1. Thiết lập màu vẽ thành màu trắng (WHITE).
  2. Thiết lập kiểu đường vẽ có độ dày 3 pixel.
  3. Vẽ hai hình chữ nhật lớn để tạo khung nền cho giao diện đồ họa.
  4. Thiết lập kiểu chữ cho nhãn nút bấm.
  5. Gọi hàm drawButton để vẽ 6 nút bấm chức năng với các nhãn tương ứng: "Load file", "Author", "Starting", "Reset", "Delete", "Exit". Các nút bấm này được sắp xếp ở vị trí bên phải khung nền.



**Hình 2.6 Tạo khung chương trình mô phỏng**

**2.3.2 Khai báo các biến, file đồ thị để cài đặt và mô phỏng thuật toán**

1. Biến cho thuật toán DFS:

* dothi: Mảng 2 chiều lưu trữ biểu diễn đồ thị dưới dạng ma trận kề.
* visited: Mảng đánh dấu các đỉnh đã thăm trong quá trình tìm kiếm.
* path: Mảng lưu trữ đường đi tìm được trong quá trình DFS.
* len: Biến lưu độ dài hiện tại của đường đi.

2. Biến và cấu trúc cho đồ họa:

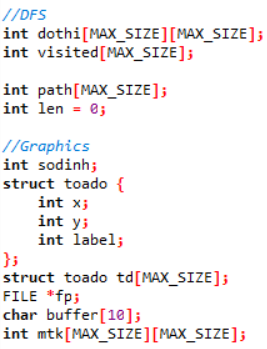
* sodinh: Biến lưu trữ tổng số đỉnh (nút) trong đồ thị.
* td: Mảng các cấu trúc toado, mỗi phần tử đại diện cho tọa độ và nhãn của một đỉnh trong đồ thị.
* fp: Con trỏ tệp, có thể được sử dụng để đọc hoặc ghi dữ liệu đồ thị từ/đến tệp.
* buffer: Mảng ký tự, có thể được sử dụng để thao tác chuỗi tạm thời.
* mtk: Mảng 2 chiều biểu diễn ma trận kề của đồ thị (tương tự như dothi).
* toado: Cấu trúc bao hàm tọa độ và nhãn của một đỉnh trong đồ thị.

Mục đích:

* Các biến cho DFS được sử dụng để thực hiện thuật toán tìm kiếm theo chiều sâu trên đồ thị, nhằm tìm đường đi giữa các đỉnh.
* Các biến và cấu trúc cho đồ họa được sử dụng để lưu trữ, quản lý và hiển thị biểu diễn đồ thị một cách trực quan, giúp người dùng dễ dàng hình dung và thao tác với đồ thị.

Giải thích thêm:

* Ma trận kề (dothi hoặc mtk) là một cách phổ biến để biểu diễn đồ thị trong lập trình. Các phần tử của ma trận kề thể hiện mối liên hệ giữa các đỉnh: nếu dothi[i][j] = 1 thì có cạnh nối từ đỉnh i đến đỉnh j.
* Cấu trúc toado giúp lưu trữ thông tin về từng đỉnh, bao gồm vị trí và nhãn của nó, phục vụ cho việc hiển thị đồ thị trên giao diện đồ họa.



**Hình 2.7 Các biến phục vụ cho thuật toán và đồ họa**

Cấu trúc của một file đồ thị được đọc vào có dạng như sau:

A screenshot of a computer

Description automatically generated

**Hình 2.8 Cấu trúc file dữ liệu đồ thị**

**2.3.3 Đọc dữ liệu từ file để khởi tạo đồ thị**

1. Hàm openAndReadFile [2]:

* Chức năng: Mở và đọc dữ liệu tọa độ và ma trận kề từ một tệp văn bản.
* Tham số đầu vào:
  + filePath: Đường dẫn đến tệp văn bản chứa dữ liệu đồ thị.
* Thao tác thực hiện:
  + Mở tệp văn bản với chế độ đọc.
  + Nếu mở tệp thất bại, thông báo lỗi "File not found".
  + Đọc số đỉnh từ tệp.
  + Đọc tọa độ x, y và nhãn của từng đỉnh từ tệp.
  + Đọc các phần tử của ma trận kề từ tệp.
  + Đánh dấu trạng thái đồ thị đã được tải (isGraphLoaded = true).
  + Đóng tệp văn bản.

2. Hàm drawgraph:

* Chức năng: Vẽ đồ thị lên giao diện đồ họa dựa trên dữ liệu đã đọc.
* Thao tác thực hiện:
  1. Thiết lập kiểu chữ cho nhãn đỉnh.
  2. Vẽ các đỉnh:
     + Vẽ đường tròn tại tọa độ của mỗi đỉnh.
     + Vẽ nhãn của đỉnh bên cạnh đường tròn.
  3. Vẽ các cạnh:
     + Duyệt ma trận kề, nếu phần tử mtk[i][j] bằng 1 (tức là có cạnh nối đỉnh i và j), vẽ đoạn thẳng nối hai đỉnh đó.

A screenshot of a computer code

Description automatically generated

**Hình 2.9 Đọc dữ liệu từ file và khởi tạo đồ thị**

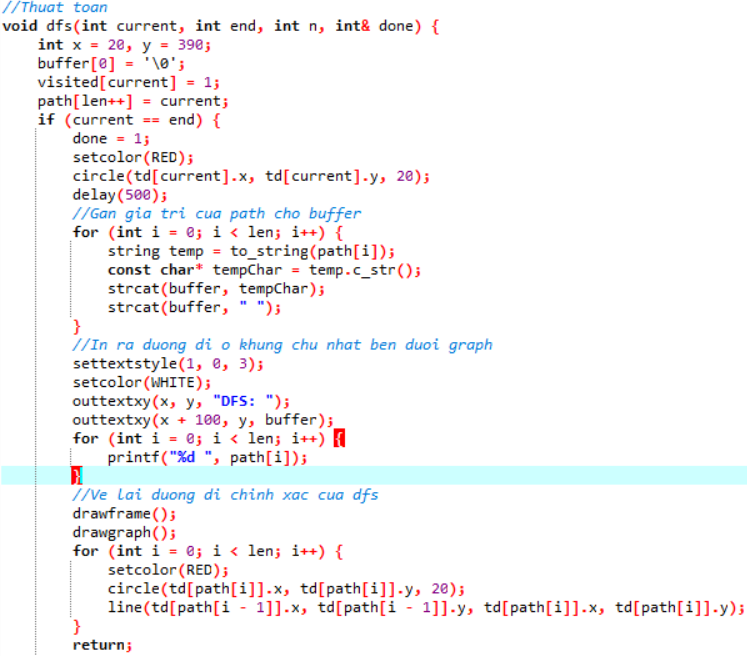
**2.3.4 Thuật toán và ứng dụng đồ họa để thể hiện thuật toán**

1. Hàm dfs:

* Chức năng: Thực hiện tìm kiếm theo chiều sâu (DFS) trên đồ thị từ một đỉnh bắt đầu cho đến khi tìm thấy đỉnh kết thúc. Vẽ trực quan quá trình tìm kiếm và đường đi tìm được. [2]
* Các bước thực hiện:
  1. Đánh dấu đỉnh hiện tại đã thăm và thêm vào đường đi:
     + Đánh dấu đỉnh current là đã thăm (visited[current] = 1).
     + Thêm đỉnh current vào danh sách đường đi (path[len++] = current).
  2. Kiểm tra nếu đã đến đỉnh kết thúc:
     + Nếu current là đỉnh kết thúc (current == end):
       - Đánh dấu tìm kiếm thành công (done = 1).
       - Tô màu đỏ đỉnh kết thúc trên đồ thị.
       - Xây dựng chuỗi mô tả đường đi tìm được và in ra màn hình.
       - Vẽ lại đồ thị với đường đi được tô màu đỏ.
       - Kết thúc tìm kiếm.
  3. Duyệt các đỉnh kề chưa thăm:
     + Lặp lại với tất cả các đỉnh i kề với đỉnh hiện tại:
       - Nếu i chưa được thăm và có cạnh nối từ current đến i (mtk[current][i] == 1 && visited[i] == 0):
         * Tô màu đỏ cạnh nối current và i.
         * Gọi đệ quy hàm dfs để tiếp tục tìm kiếm từ đỉnh i.
  4. Lùi lại nếu không tìm thấy đường đi:
     + Nếu không tìm thấy đường đi từ đỉnh hiện tại (done vẫn bằng 0), loại bỏ đỉnh hiện tại khỏi danh sách đường đi (len--).

2. Hàm DFS\_display:

* Chức năng: Hiển thị giao diện đồ họa và cho phép người dùng nhập đỉnh bắt đầu và đỉnh kết thúc để thực hiện tìm kiếm DFS.
* Các bước thực hiện:
  1. Nhập đỉnh bắt đầu và đỉnh kết thúc từ người dùng.
  2. Gọi hàm dfs để thực hiện tìm kiếm DFS từ đỉnh bắt đầu đến đỉnh kết thúc.

A screenshot of a computer code

Description automatically generated

**Hình 2.10 Hàm mô phỏng và sử dụng thuật toán**

**Chương 3. KẾT QUẢ NGHIÊN CỨU**

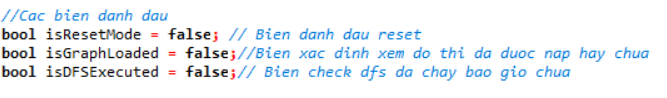
Để có thể sử dụng các hàm và cấu trúc dữ liệu liên quan đến Windows API nhằm hỗ trợ thực hiện các thao tác như mở file từ file explore thì ta cần khai báo thư viện windows.h.

Mặc dù bản thân máy tính có thể đã có sẵn thư viện windows.h, tuy nhiên, bạn vẫn cần khai báo nó trong mã nguồn của bạn để cho trình biên dịch biết rằng bạn đang sử dụng các hàm và cấu trúc dữ liệu từ thư viện đó

Để khai báo thư viện windows.h trong chương trình của bạn, bạn có thể sử dụng lệnh sau:

*#include<windows.h>*

Các biến đánh dấu được khai báo để sử dụng cho việc thực hiện các chức năng của chương trình:



**Hình 3.1 Các biến đánh dấu**

**3.1 KẾT QUẢ MÔ PHỎNG TRÊN DEV C++ BẰNG THƯ VIỆN GRAPHICS.H**

**3.1.1 Hiển thị khung**

A screenshot of a computer

Description automatically generated

**Hình 3.2 Hiển thị khung mô phỏng**

Sau khi chạy chương trình hàm *main()* sẽ được gọi để tạo cửa sổ đồ họa và vẽ khung mô phỏng bằng hàm *drawframe()*

A white background with blue text

Description automatically generated

**Hình 3.3 Hàm main**

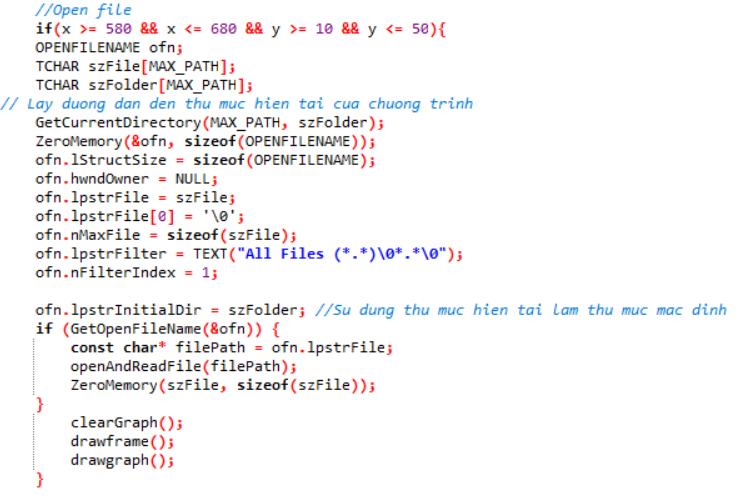
**3.1.2 Hiển thị đồ thị**

A screenshot of a computer

Description automatically generated

**Hình 3.4 Hiển thị đồ thị**

Khi click chuột vào nút **Load File** trên khung mô phỏng, hàm *hanldeClick()* sẽ được gọi để kiểm tra sự kiện chuột và thực thi đoạn code để thực hiện việc lấy file dữ liệu từ cửa sổ file explore để nạp đồ thị vào khung mô phỏng.



**Hình 3.5 Code nạp file từ cửa sổ file explore**

Các bước thực hiện:

1. Kiểm tra điều kiện mở tệp:
   * Nếu tọa độ chuột hiện tại nằm trong vùng nút mở tệp (x >= 580 && x <= 680 && y >= 10 && y <= 50), bắt đầu tiến trình mở tệp.
2. Chuẩn bị cấu trúc chọn tệp:
   * Khởi tạo cấu trúc OPENFILENAME để sử dụng hàm GetOpenFileName.
   * Đặt các thuộc tính như tên cửa sổ, tên tệp, bộ lọc tệp, thư mục mặc định.

3. Mở hộp thoại chọn tệp:

* + Gọi hàm GetOpenFileName để hiển thị hộp thoại chọn tệp.
  + Nếu người dùng chọn tệp:
    - Lấy đường dẫn tệp đã chọn.
    - Gọi hàm openAndReadFile để đọc dữ liệu đồ thị từ tệp.

4. Vẽ lại đồ thị:

* + Xóa nội dung đồ thị cũ trên màn hình.
  + Vẽ lại khung đồ thị và đồ thị mới được đọc từ tệp.

**3.1.3 Hiển thị mô phỏng kết quả duyệt DFS**

Khi click chuột vào nút **Starting** trên khung mô phỏng, hàm *hanldeClick()* sẽ được gọi để kiểm tra sự kiện chuột. Nếu không tồn tại đồ thị trong khung, chương trình sẽ hiển thị thông báo nhắc nhở người dùng nạp đồ thị để thực hiện thuật toán. Nếu đồ thị đang tồn tại trong khung, chương trình thực thi đoạn code để thực hiện mô phỏng thuật toán DFS sau đó hiển hiển thị đường đi chính xác từ điểm bắt đầu đến điểm kết thúc được nhập từ bàn phím ở của sổ console bằng cách tô đỏ các đỉnh và các cạnh của chúng đồng thời hiển thị đường đi chính xác ở khung kết quả bên dưới. Sau khi chạy xong DFS lần đầu tiên người dùng có thể click lại vào nút **Starting** để chạy lại thuật toán:

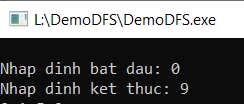
A screenshot of a computer

Description automatically generated

**Hình 3.6 Thông báo không tồn tại đồ thị trong khung**

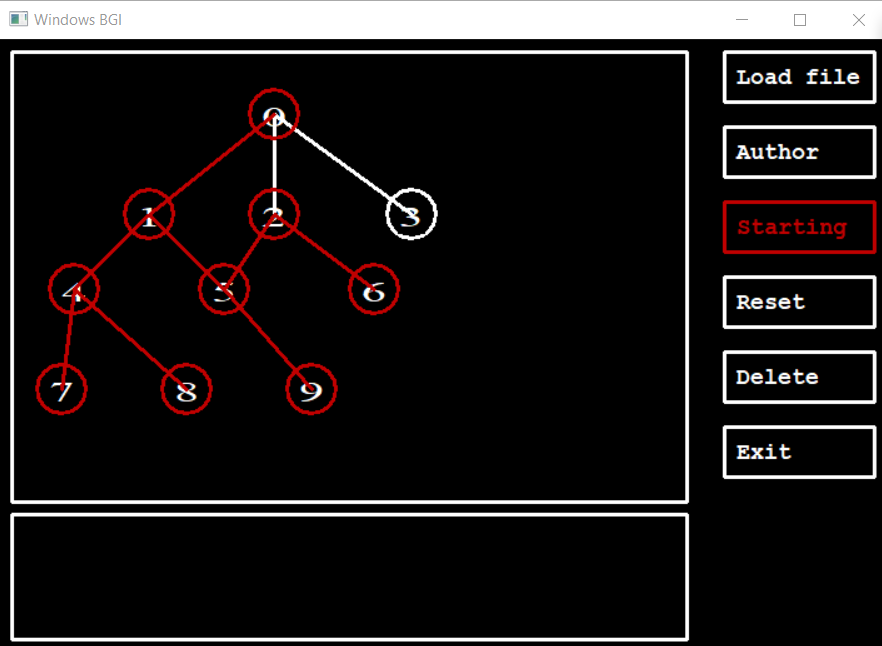
Ví dụ: Chạy chương trình và hiển thị kết quả từ đường đi từ 0 đến 9 bằng thuật toán DFS

- INPUT: Nhập dữ liệu từ bàn phím trong cửa sổ console:



**Hình 3.7 Nhập dữ liệu từ bàn phím**

- Hiển thị mô phỏng sau khi đã duyệt qua tất cả các đỉnh và cạnh theo INPUT



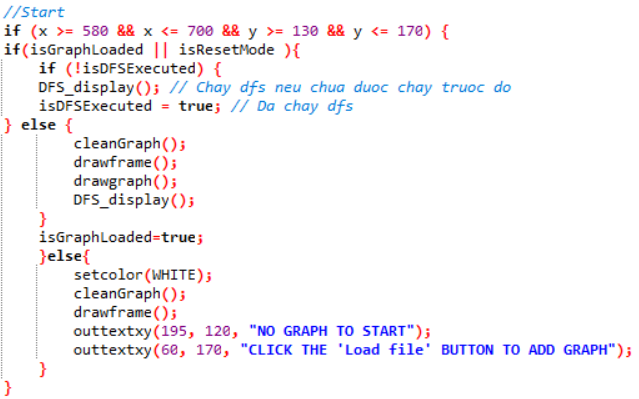
**Hình 3.8 Mô phỏng duyệt DFS**

- Hiển thị mô phỏng thứ tự đường đi chính xác theo yêu cầu của bài toán:

A screenshot of a computer

Description automatically generated

**Hình 3.9 Mô phỏng chính xác yêu cầu bài toán**



**Hình 3.10 Code thực hiện nút Starting**

Các bước thực hiện:

1. Kiểm tra điều kiện bắt đầu:
   * Nếu tọa độ chuột hiện tại nằm trong vùng nút bắt đầu (x >= 580 && x <= 700 && y >= 130 && y <= 170), tiến hành xử lý.
2. Kiểm tra trạng thái DFS:
   * Nếu chưa từng thực hiện DFS (isDFSExecuted bằng false):
     + Gọi hàm DFS\_display để hiển thị giao diện và thực hiện tìm kiếm DFS.
     + Đánh dấu đã thực hiện DFS (isDFSExecuted = true).
   * Nếu đã thực hiện DFS trước đó:
     + Xóa các đường đi đã tìm kiếm trước đó (cleanGraph).
     + Vẽ lại khung đồ thị và đồ thị ban đầu (drawframe, drawgraph).
     + Gọi lại hàm DFS\_display để thực hiện tìm kiếm DFS mới.
3. Đánh dấu đồ thị đã được tải:
   * Đặt isGraphLoaded thành true, cho biết đã có đồ thị được tải.

**3.1.4 Các chức năng khác**

- Hiển thị tác giả: Khi click chuột vào nút **Author** trên khung mô phỏng, hàm *hanldeClick()* sẽ được gọi để kiểm tra sự kiện chuột và thực thi đoạn code hiển thị thông tin tác giả:

A screenshot of a computer

Description automatically generated

**Hình 3.11 Tác giả**

- Đặt đồ thị về trạng thái ban đầu: Khi click chuột vào nút **Reset** trên khung mô phỏng, hàm *hanldeClick()* sẽ được gọi để kiểm tra sự kiện chuột và thực thi đoạn code đặt đồ thị về lại trạng thái ban đầu :

Ví dụ: Sau khi chạy xong thuật toán DFS, bạn không muốn tiếp tục chạy thêm lần nữa nhưng cũng không muốn xóa đồ thị thì nút Reset sẽ là giải pháp để thực hiện mong muốn của bạn:

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Hình 3.12 Trước khi Reset**  **Hình 3.13 Sau khi Reset**

- Xóa đồ thị và các thao tác đang thực hiện: Khi click chuột vào nút **Delete** trên khung mô phỏng, hàm *hanldeClick()* sẽ được gọi để kiểm tra sự kiện chuột và thực thi đoạn code xóa đồ thị và các kết quả đang được hiển thị( nếu có) sau đó đặt khung mô phỏng về trạng thái ban đầu và hiển thị thông báo:

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

**Hình 3.14 Trước khi thực hiên delete**

A screenshot of a computer

Description automatically generated

**Hình 3.15 Sau khi thực hiện delete**

- Thoát khỏi chương trình: Khi click chuột vào nút **Exit** trên khung mô phỏng, hàm *hanldeClick()* sẽ được gọi để kiểm tra sự kiện chuột và thực hiển đóng cửa sổ đồ họa.

**Chương 4. KẾT LUẬN**

Báo cáo với đề tài "Sử dụng công cụ đồ họa graphics.h để mô phỏng giải thuật tìm kiếm theo chiều sâu (depth-first search)" đã trình bày một ứng dụng cụ thể của giải thuật tìm kiếm theo chiều sâu và sử dụng thư viện đồ họa graphics.h để tạo một giao diện đồ họa cho việc mô phỏng.

Trong báo cáo, đã trình bày một sự giới thiệu về giải thuật tìm kiếm theo chiều sâu (DFS), một thuật toán quan trọng trong lĩnh vực trí tuệ nhân tạo và lý thuyết đồ thị. Đã mô tả cách thức hoạt động của giải thuật DFS, ưu nhược điểm và cách nó được sử dụng để tìm kiếm đường đi trong đồ thị.

Tiếp theo, báo cáo đã giới thiệu thư viện graphics.h và cách nó được sử dụng để tạo một giao diện đồ họa. Đã trình bày các hàm và phương pháp để vẽ các đối tượng đồ họa như hình chữ nhật và nút bấm trên màn hình. Bằng cách sử dụng thư viện này, đã thiết kế một giao diện đồ họa cho việc mô phỏng giải thuật DFS.

Sau đó, báo cáo đã mô tả cách giải thuật DFS được mô phỏng trên giao diện đồ họa. Đã trình bày cách thức tạo và hiển thị đồ thị, cách thức thực hiện giải thuật DFS và cách thức hiển thị quá trình tìm kiếm trên giao diện đồ họa.

Cuối cùng, báo cáo đã trình bày một ứng dụng cụ thể và có ý nghĩa của giải thuật tìm kiếm theo chiều sâu (DFS) và công cụ đồ họa graphics.h để mô phỏng. Báo cáo này cung cấp một cái nhìn đầy đủ về việc áp dụng giải thuật DFS vào thực tế và làm thế nào để tạo ra một giao diện đồ họa cho việc mô phỏng này.

**TÀI LIỆU THAM KHẢO**

[1]. Nguyễn Thị Chinh (2011), *Mô phỏng một số thuật toán trên đồ thị*,Luận văn thạc sĩ.

[2]. Đoàn Vũ Thịnh (2023), *AI*, GitHub, Truy cập vào 21/12/2023, từ [AI/code/DFS.cpp at main · thinhdoanvu/AI · GitHub](https://github.com/thinhdoanvu/AI/blob/main/code/DFS.cpp).

**PHỤ LỤC**

1. Hàm xóa đồ thị

A computer code with text

Description automatically generated

2. Hàm tạo hiệu ứng khi hover vào các nút của chương trình

A screenshot of a computer program

Description automatically generated

3. Hàm xử lý con trỏ chuột

A computer screen shot of a computer code

Description automatically generated

4. Code hiển thị thông tin tác giả

A close-up of a number

Description automatically generated

5. Code của nút Reset

A computer screen shot of a code

Description automatically generated

6. Code của nút Delete

A computer code with numbers and symbols

Description automatically generated with medium confidence

7. Code của nút Exit

A close up of numbers

Description automatically generated