BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue and white circle with text and a red star

Description automatically generated

**BÁO CÁO THỰC TẬP CƠ SỞ**

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

**Giảng viên hướng dẫn: ThS. Đoàn Vũ Thịnh**

**Sinh viên thực hiện: Nguyễn Minh Tài**

**Mã số sinh viên: 64132077**

**Lớp: 64.CNTT-3**

Khánh Hòa - 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

A blue and white circle with text and a red star

Description automatically generated

**BÁO CÁO THỰC TẬP CƠ SỞ**

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

**Giảng viên hướng dẫn: ThS. Đoàn Vũ Thịnh**

**Sinh viên thực hiện: Nguyễn Minh Tài**

**Mã số sinh viên: 64132077**

**Lớp: 64.CNTT-3**

Khánh Hòa - 2024

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**Khoa: Công Nghệ Thông Tin**

**BÁO CÁO TIẾN ĐỘ VÀ ĐÁNH GIÁ THỰC TẬP CƠ SỞ**

**Tên đề tài:** : Sử dụng công cụ đồ họa graphics.h để mô phỏng giải thuật tìm đường đi theo chiều sâu Depth First Search

**Giáo viên hướng dẫn:** ThS. Đoàn Vũ Thịnh

**Sinh viên được hướng dẫn:** Nguyễn Minh Tài

**MSSV:** 64132077

**Lớp:** 64.CNTT-3

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 28/11/2024 | Nhận đề tài và lên kế hoạch thực hiện  Tìm hiểu lý thuyết thuật toán và mục đích  phát triển đề tài  Lựa chọn giải pháp để cài đặt |  |
| 2 | 2/12/2024 | Phát biểu bài toán  Lưu dồ thuật toán  Code thuật toán tìm kiếm theo chiều sâu DFS |  |
| 3 | 9/12/2024 | Thực hiện thiết kế giao diện  Thực hiện code chức năng load file  Thực hiện code chức năng nhập từ bàn phím  Thực hiện code chức năng đồ họạ |  |
| 4 | 16/12/2024 | Chỉnh sửa lại thuật toán  Chỉnh lại giao diện  Hoàn thiện các tính năng  Viết báo cáo |  |
| 5 | 23/12/2024 | Chỉnh sửa và hoàn thiện báo cáo  Hoàn thiện demo |  |
| 6 | 31/12/2024 | Sinh viên nộp bản thảo lần cuối sau khi đã chỉnh sửa các yêu cầu như đã đề ra. |  |

**MỤC LỤC**

[LỜI CẢM ƠN 6](#_Toc186358732)

[TÓM TẮT 7](#_Toc186358733)

[CHƯƠNG 1: TỔNG QUAN 8](#_Toc186358734)

[1.1.1 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU DFS 8](#_Toc186358735)

[1.1.1 Sơ lược về thuật toán 8](#_Toc186358736)

[1.1.2 Ý tưởng chính của thuật toán 8](#_Toc186358737)

[1.1.3 Đánh giá thuật toán DFS 13](#_Toc186358738)

[1.1.4 Lưu đồ thuật toán 15](#_Toc186358739)

[1.2 DEV C++ IDE 15](#_Toc186358740)

[1.3 THƯ VIỆN ĐỒ HỌA GRAPHICS.H 16](#_Toc186358741)

[CHƯƠNG 2: PHƯƠNG PHÁP NGHIÊN CỨU 18](#_Toc186358742)

[2.1 CÀI ĐẶT VÀ SỬ DỤNG MÔI TRƯỜNG ĐỒ HỌA 18](#_Toc186358743)

[2.1.1 Cài đặt Dev-C++ 18](#_Toc186358744)

[2.1.2 Cài đặt thư viện graphic.h 18](#_Toc186358745)

[2.2 TỔNG QUAN VỀ THIẾT KẾ GIAO DIỆN 20](#_Toc186358746)

[2.2.1 Sơ lược giao diện 20](#_Toc186358747)

[2.2.2 Thiết kế giao diện 22](#_Toc186358748)

[2.2.2.1 Khởi tạo cửa sổ đồ hoạ BGI 22](#_Toc186358749)

[2.2.2.2 Xây dựng các thực thể trên cửa sổ BGI 22](#_Toc186358750)

[2.2.2.3 Các tác vụ xử lý đồ thị 29](#_Toc186358751)

[2.3 XỬ LÝ SỰ KIỆN CHUỘT 31](#_Toc186358752)

[2.3.1 Xử lý sự kiện click vào đỉnh đồ thị 31](#_Toc186358753)

[2.3.2 Xử lý sự kiện hover chuột: 34](#_Toc186358754)

[2.3.3 Xử lý thao tác chuột khi click 36](#_Toc186358755)

[2.3.4 Click chọn phương thức input: 41](#_Toc186358756)

[2.3.5 Click tạo node: 42](#_Toc186358757)

[2.3.6 Các hàm bổ trợ xử lý: 44](#_Toc186358758)

[2.4 XỬ LÝ VÀ PHÂN TÍCH THUẬT TOÁN 46](#_Toc186358759)

[2.4.1 Khởi tạo chương trình 46](#_Toc186358760)

[2.4.1.1 Khai báo 46](#_Toc186358761)

[2.4.1.3 Input dữ liêu thực thi 48](#_Toc186358762)

[2.4.2 Thuật toán và ứng dụng đồ họa để thể hiện thuật toán 50](#_Toc186358763)

[2.4.2.1 Thuật toán tìm kiếm chiều sâu DFS 50](#_Toc186358764)

[2.4.2.2 Thuật toán DFS với đồ họa 58](#_Toc186358765)

[CHƯƠNG 3: KẾT QUẢ THỰC HIỆN 66](#_Toc186358766)

[CHƯƠNG 4: KẾT LUẬN 74](#_Toc186358767)

**DANH MỤC HÌNH**

[Hình 1.1 Ảnh minh họa thuật toán DFS 1 9](#_Toc186358902)

[Hình 1.2 Ảnh minh họa thuật toán DFS 2 10](#_Toc186358903)

[Hình 1.3 Ảnh minh họa thuật toán DFS 3 10](#_Toc186358904)

[Hình 1.4 Ảnh minh họa thuật toán DFS 4 10](#_Toc186358905)

[Hình 1.5 Ảnh minh họa thuật toán DFS 5 11](#_Toc186358906)

[Hình 1.6 Ảnh minh họa thuật toán DFS 6 11](#_Toc186358907)

[Hình 1.7 Hình ảnh mã giải thuật toán DFS 11](#_Toc186358908)

[Hình 1.8 Đồ thị tìm đường đi từ A đến G 13](#_Toc186358909)

[Hình 1.9 Ví dụ minh họa quá trình tìm đường đi từ 1 đến 4 13](#_Toc186358910)

[Hình 1.10 Lưu đồ thuật toán DFS 15](#_Toc186358911)

[Hình 1.11 Hình ảnh sử dụng Dev C++ 16](#_Toc186358912)

[Hình 1.12 Minh hoa hình ảnh đồ họa graphic.h 17](#_Toc186358913)

[Hình 2.1 Cài đăt môi trường đồ họa graphic.h 19](#_Toc186358914)

[Hình 2.2 Hình ảnh cấu hình thư viện graphic.h vào Dev C++ 19](#_Toc186358915)

[Hình 2.3 Giao diên bắt đàu mô tả thuật toán DFS 20](#_Toc186358916)

[Hình 2.4 Hình ảnh đồ thị vô hướng minh họa DFS 21](#_Toc186358917)

[Hình 2.5 Hình ảnh thiết kế cửa số đồ họa 22](#_Toc186358918)

[Hình 2.6 Hình ảnh xử lý vẽ các nút 23](#_Toc186358919)

[Hình 2.7 Hình ảnh đồ thị vô hướng được vẽ 26](#_Toc186358920)

[Hình 2.8 Hình ảnh đồ thị có hướng được vẽ 26](#_Toc186358921)

[Hình 2.9 Hình ảnh khi click các đỉnh bắt đầu và kết thúc 33](#_Toc186358922)

[Hình 2.10 Hình ảnh hover vào button 36](#_Toc186358923)

[Hình 2.11 Hình ảnh hiển thị phương thức nhập 42](#_Toc186358924)

[Hình 2.12 Hình ảnh hiển thị phương thức nhập 44](#_Toc186358925)

[Hình 2.13 Hình ảnh khai báo các thư viện 46](#_Toc186358926)

[Hình 2.14 Hình ảnh khai báo biến xử lý thuật toán 47](#_Toc186358927)

[Hình 2.15 Ma trận kề đồ thị vô hướng của các đỉnh trong tệp 48](#_Toc186358928)

[Hình 2.16 Ma trận kề đồ thị vô hướng của các đỉnh trong tệp 48](#_Toc186358929)

[Hình 2.17 Kết quả thuật toán DFS khi xử lí 58](#_Toc186358930)

[Hình 2.18 Hình ảnh vẽ đỉnh và cạnh khi xử lý thuật toán 60](#_Toc186358931)

[Hình 2.19 Hình ảnh vẽ lại đồ thị khi tìm thấy đường đi 62](#_Toc186358932)

[Hình 2.20 Hình ảnh hiển thị đường đi 63](#_Toc186358933)

[Hình 2.21 Hiển thị thông báo không tìm thấy đường đi 65](#_Toc186358934)

[Hình 3.1 Giao diện khi nhấn nút load file 66](#_Toc186358935)

[Hình 3.2 Giao diện sau khi chọn file và tải đồ thị 66](#_Toc186358936)

[Hình 3.3 Giao diện khi bắt đầu nhấn button Start 67](#_Toc186358937)

[Hình 3.4 Giao diện khi click vào “Enter” 67](#_Toc186358938)

[Hình 3.5 Giao diện khi nhập các đỉnh từ bàn phím và xử lý DFS 68](#_Toc186358939)

[Hình 3.6 Giao diện khi chọn phương thức nhập là Click vào đỉnh 68](#_Toc186358940)

[Hình 3.7 Giao diện khi click vào đỉnh bắt đầu 69](#_Toc186358941)

[Hình 3.8 Giao diện khi click vào đỉnh kết thúc 69](#_Toc186358942)

[Hình 3.9 Giao diện khi xử lý thuật toán bằng phương pháp nhập click chuột 70](#_Toc186358943)

[Hình 3.10 Giao diện khi nhấn button Author 70](#_Toc186358944)

[Hình 3.11 Giao diện khi nhấn button Reset 71](#_Toc186358945)

[Hình 3.12 Giao diện khi nhấn button Delete 71](#_Toc186358946)

[Hình 3.13 Giao diện khi nhấn button Exit 72](#_Toc186358947)

[Hình 3.14 Giao diện màn hình console khi xử lý thuật toán DFS 72](#_Toc186358948)

[Hình 3.15 Giao diện khi load file xong chuẩn bị tạo node 73](#_Toc186358949)

[Hình 3.16 Giao diện khi tạo node 73](#_Toc186358950)

# LỜI CẢM ƠN

Để có thể hoàn thành đợt thực tập lần này, em xin chân thành cảm ơn đến quý thầy cô khoa Công nghệ Thông tin đã tạo điều kiện hỗ trợ và giúp đỡ em trong quá trình học tập và nghiên cứu đề tài này.

Qua đây, em xin chân thành cảm ơn thầy Đoàn Vũ Thịnh, người đã trực tiếp quan tâm và hướng dẫn chúng em hoàn thành tốt đợt thực tập trong thời gian qua.

Do kiến thức còn hạn chế và thời gian thực hiện còn ngắn nên bài báo cáo của em còn nhiều thiếu sót, kính mong sự góp ý của quý thầy cô.

Em xin chân thành cảm ơn!

# TÓM TẮT

Đồ họa máy tính là một lĩnh vực của Công nghệ Thông tin, ở đó việc nghiên cứu xây dựng và tập hợp các công cụ (mô hình lý thuyết và phần mềm) khác nhau để kiến tạo, xây dựng, lưu trữ và xử lý các mô hình và hình ảnh của các đối tượng, sự vật, hiện tượng trong cuộc sống, sản xuất. Thuật toán tô màu bằng dòng quét và tô màu bằng đường biên trong đồ họa máy tính có tầm quan trọng rất lớn và được sử dụng rộng rãi trong các phần mềm đồ họa phổ biến hiện nay như Adobe Photoshop, Corel Draw, Microsoft Paint.

Quy trình thực hiện được trải qua các bước từ cài đặt thuật toán, hiển thị kết quả đầu ra trên màn hình đều được thực hiện trên môi trường C/C++ thông qua ứng dụng Dev C/C++ có kết hợp với thư viện graphics.h.

Sản phẩm đã minh họa được từng bước giải thuật tìm đường đi theo chiều sâu Depth First Search . Đồng thời cũng chỉ ra các trường hợp hạn chế của mỗi thuật toán và cách khắc phục các nhược điểm đó. Sản phẩm chạy tốt với dữ liệu được lấy từ file, nhập từ bàn phím và tưởng tác thuận lợi với sự hỗ trợ của chuột.

# CHƯƠNG 1: TỔNG QUAN

## THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU DFS

### Sơ lược về thuật toán

Thuật toán Tìm kiếm theo Chiều sâu (Depth First Search – DFS) là một trong những thuật toán cơ bản và quan trọng trong lý thuyết đồ thị. Nó được sử dụng để duyệt qua tất cả các đỉnh trong đồ thị, đi từ một đỉnh xuất phát và khám phá các đỉnh lân cận theo chiều sâu trước khi quay lại để tìm các nhánh khác. Thuật toán này có ứng dụng rộng rãi trong các bài toán về đồ thị, như tìm kiếm, phát hiện chu trình, tìm thành phần liên thông, và nhiều ứng dụng khác trong lĩnh vực khoa học máy tính.

Thuật toán bắt đầu tại một node, và tiếp tục đi đến các node khác từ node bắt đầu bằng các cạnh của đồ thị. Tìm kiếm theo chiều sâu luôn theo một con đường duy nhất trong đồ thị để tìm các node mới. Sau đó, nó quay lại node liền trước và bắt đầu khai phá các con đường khác của đồ thị. Thuật toán lưu trữ các node đã viếng thăm, để nó đi đến mỗi node một lần duy nhất.

### Ý tưởng chính của thuật toán

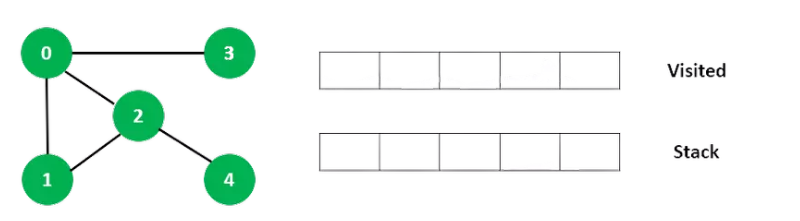
Thuật toán Tìm kiếm theo chiều sâu (Depth First Search – DFS) là một trong những thuật toán nền tảng trong lý thuyết đồ thị. Mục tiêu của DFS là duyệt qua tất cả các đỉnh của đồ thị hoặc cây bằng cách đi "sâu" vào các nhánh của đồ thị trước khi quay lại các nhánh khác. Điều này có nghĩa là, bắt đầu từ một đỉnh, thuật toán sẽ tiếp tục đi tới các đỉnh kề chưa được thăm cho đến khi không thể tiếp tục. Khi đạt đến đỉnh không còn đỉnh nào chưa được duyệt, thuật toán quay lại đỉnh trước đó và tiếp tục thăm các đỉnh kề chưa được thăm. Quá trình này diễn ra cho đến khi tất cả các đỉnh của đồ thị được thăm.

Quá trình hoạt động của DFS có thể mô tả như sau:

* Khởi đầu từ một đỉnh ban đầu: Thuật toán bắt đầu từ một đỉnh bất kỳ của đồ thị và đánh dấu đỉnh này là đã thăm.
* Duyệt chiều sâu: Sau khi thăm một đỉnh, thuật toán tiếp tục đi "sâu" vào các đỉnh kề chưa được thăm, tiếp tục đi theo chiều sâu cho đến khi không thể tiến xa hơn.
* Quay lại và kiểm tra lại các nhánh: Khi không còn đỉnh nào có thể tiếp tục duyệt trong nhánh hiện tại, thuật toán quay lại đỉnh trước đó (nơi đã thăm ít nhất một đỉnh kề), để kiểm tra các đỉnh kề chưa được thăm. Quá trình này tiếp tục lặp lại cho đến khi toàn bộ các đỉnh của đồ thị đã được duyệtThuật toán DFS có thể được mô tả qua hai phương pháp:
* Sử dụng đệ quy: Đây là cách phổ biến và đơn giản nhất để triển khai DFS. Mỗi lần thăm một đỉnh, thuật toán sẽ gọi đệ quy để thăm các đỉnh kề chưa được duyệt. Đệ quy giúp thuật toán dễ dàng quay lại đỉnh trước đó khi không còn đỉnh nào chưa được thăm trong nhánh hiện tại.
* Sử dụng ngăn xếp (stack): Thay vì sử dụng đệ quy, DFS cũng có thể được triển khai bằng cách sử dụng ngăn xếp để lưu trữ các đỉnh cần duyệt. Mỗi lần thăm một đỉnh, thuật toán sẽ thêm các đỉnh kề của nó vào ngăn xếp. Quá trình này sẽ tiếp tục cho đến khi ngăn xếp rỗng, tức là tất cả các đỉnh đã được thăm..

Trong bài viết này, chúng ta sẽ tập trung vào việc triển khai thuật toán DFS bằng phương pháp ngăn xếp. Với phương pháp này, thuật toán sử dụng một cấu trúc ngăn xếp để quản lý quá trình duyệt đồ thị. Mỗi đỉnh được lấy ra từ ngăn xếp sẽ được thăm và sau đó các đỉnh kề của nó, nếu chưa được duyệt, sẽ được thêm vào ngăn xếp. Khi không còn đỉnh nào để đi sâu hơn, thuật toán sẽ quay lại các đỉnh trước đó nhờ cơ chế LIFO (Last In, First Out) của ngăn xếp và tiếp tục quá trình duyệt đồ thị.

Hãy xem thuật toán Tìm kiếm theo chiều sâu hoạt động như thế nào với một ví dụ. Chúng ta dùng đồ thị vô hướng có 5 đỉnh.



Hình 1.1 Ảnh minh họa thuật toán DFS 1

Chúng ta bắt đầu từ đỉnh 0, thuật toán DFS bắt đầu bằng cách đứa nó vào danh sách Visited và đưa tất cả các cạnh liền kề đỉnh đang xét vào ngăn xếp.

A screenshot of a white background

Description automatically generated

Hình 1.2 Ảnh minh họa thuật toán DFS 2

Tiếp theo, chúng ta truy cập phần tử ở đầu ngăn xếp tức là 1 và đi đến các nút liền kề của nó. Vì 0 đã được truy cập, nên 2 là số được xét.

A screenshot of a computer game

Description automatically generated

Hình 1.3 Ảnh minh họa thuật toán DFS 3

Đỉnh 2 có một đỉnh liền kề chưa được thăm là 4, vì vậy chúng ta thêm đỉnh đó vào vị trí đầu của ngăn xếp và duyệt nó.

A close-up of a number

Description automatically generated

Hình 1.4 Ảnh minh họa thuật toán DFS 4

A screenshot of a computer game

Description automatically generated

Hình 1.5 Ảnh minh họa thuật toán DFS 5

Sau khi chúng ta duyệt phần tử 3 cuối cùng, nó không có bất kỳ nút liền kề nào chưa được duyệt, vì vậy chúng tôi đã hoàn thành tìm kiếm theo chiều sâu trong đồ thị trên.

A screenshot of a computer game

Description automatically generated

Hình 1.6 Ảnh minh họa thuật toán DFS 6

**Biểu diễn thuật toán:**

**A close up of text

Description automatically generated**

Hình 1.7 Hình ảnh mã giải thuật toán DFS

**Thuật giải**

Một số quy ước:

* Open: Là tập các đỉnh có thể được xét tiếp theo. Trong thuật giải này, Open được quản lý bằng mảng, thêm phần tử vào đầu mảng và lấy phần tử từ đầu mảng (tương tự cơ chế ngăn xếp).
* Close: Là tập các đỉnh đã được xét (quản lý bằng mảng, thêm phần tử vào cuối mảng).
* s: Đỉnh xuất phát.
* g: Đỉnh kết thúc.

Trình bày thuật giải:

Khởi tạo:

1. Đưa đỉnh xuất phát s vào mảng Open.
2. Gán mảng Close là rỗng.

Duyệt đồ thị:

1. Lặp lại cho đến khi Open rỗng:
   * Lấy phần tử đầu tiên n ra khỏi Open để xét.
   * Nếu nnn là đỉnh đích g: Kết thúc thuật giải, trả về True.
   * Nếu nnn không thuộc tập Close, thêm nnn vào cuối mảng Close.
   * Tìm tất cả các đỉnh kề Γ(n) của nnn.
   * Với mỗi đỉnh m∈Γ(n): Nếu mmm không thuộc Open∪Close thêm m vào đầu mảng Open.
2. Nếu hết các bước mà không tìm thấy đỉnh ggg: Thuật giải kết thúc, trả về False.

**Ví dụ minh hoạ thuật toán DFS**

Cho hình ảnh minh hoạ về cây ta có: 1 là đỉnh bắt đầu, 4 là đỉnh kết thúc

Yêu cầu bài toán: tìm đường đi từ đỉnh 1 đến đỉnh 4 bằng giải thuật tìm kiếm theo chiều sâu.

A diagram of a number

Description automatically generated

Hình 1.8 Đồ thị tìm đường đi từ A đến G

Ta có các bước để giải thuật toán DFS như trên

Qua đó tìm được đường đi từ đỉnh 1 đến đỉnh 4:

A table with numbers and letters

Description automatically generated

Hình 1.9 Ví dụ minh họa quá trình tìm đường đi từ 1 đến 4

### Đánh giá thuật toán DFS

Thuật toán Tìm kiếm theo chiều sâu (DFS) có độ phức tạp tính toán khá đơn giản, tuy nhiên, nó phụ thuộc vào cấu trúc của đồ thị mà thuật toán đang làm việc. Dưới đây là đánh giá độ phức tạp của thuật toán DFS dựa trên các yếu tố như số lượng đỉnh (vertices) và số lượng cạnh (edges) trong đồ thị.

Độ phức tạp thời gian của thuật toán DFS là **O(V + E)**, trong đó:

* **V** là số lượng đỉnh trong đồ thị.
* **E** là số lượng cạnh trong đồ thị.

Lý do là trong quá trình duyệt, thuật toán sẽ thăm tất cả các đỉnh (V đỉnh) và duyệt qua tất cả các cạnh (E cạnh). Mỗi đỉnh và mỗi cạnh sẽ được kiểm tra tối đa một lần.

* **Thăm đỉnh:** Khi duyệt qua đồ thị, mỗi đỉnh sẽ được thăm một lần. Việc thăm một đỉnh (cộng với việc đánh dấu đỉnh đó là đã thăm) tốn thời gian O(1).
* **Duyệt cạnh:** Thuật toán sẽ kiểm tra tất cả các cạnh kề với đỉnh hiện tại. Trong một đồ thị không có trọng số, mỗi cạnh cũng sẽ được kiểm tra một lần khi thăm đỉnh kề, do đó số lần kiểm tra cạnh là E.

Do đó, tổng thời gian duyệt của DFS là tổng thời gian thăm tất cả các đỉnh cộng với tổng thời gian kiểm tra tất cả các cạnh, dẫn đến độ phức tạp thời gian tổng quát là **O(V + E)**. Độ phức tạp không gian của DFS cũng phụ thuộc vào cấu trúc đồ thị và cách triển khai. Cụ thể:

* **Sử dụng đệ quy:** Khi triển khai DFS theo phương pháp đệ quy, mỗi lời gọi đệ quy sẽ tạo ra một khung ngăn xếp (stack frame) mới. Vì vậy, độ sâu của đệ quy (cũng là chiều sâu của đồ thị) có ảnh hưởng đến độ phức tạp không gian. Trong trường hợp xấu nhất (đồ thị có dạng chuỗi dài), độ phức tạp không gian là O(V), vì chiều sâu đệ quy có thể đạt đến số lượng đỉnh trong đồ thị.
* **Sử dụng ngăn xếp:** Nếu DFS được triển khai bằng cách sử dụng ngăn xếp thay vì đệ quy, ta cần một ngăn xếp để lưu trữ các đỉnh đang được thăm. Kích thước của ngăn xếp sẽ phụ thuộc vào chiều sâu của đồ thị và cũng có thể đạt đến O(V) trong trường hợp xấu nhất.

Vì thế, độ phức tạp không gian của thuật toán DFS là **O(V)**, vì chúng ta chỉ cần lưu trữ các đỉnh cần duyệt trong ngăn xếp hoặc ngăn xếp đệ quy.

### Lưu đồ thuật toán

A diagram of a flowchart

Description automatically generated

Hình 1.10 Lưu đồ thuật toán DFS

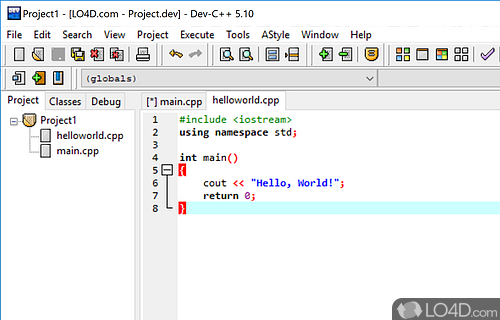
## 1.2 DEV C++ IDE

Dev C++ là một môi trường phát triển tích hợp (IDE) được thiết kế để hỗ trợ lập trình viên trong việc phát triển phần mềm sử dụng ngôn ngữ C++ và các ngôn ngữ lập trình khác như C. IDE này được biết đến là một công cụ miễn phí và mã nguồn mở, cung cấp đầy đủ các tính năng cơ bản cần thiết cho việc viết mã, biên dịch, và gỡ lỗi trong lập trình C++. Dev C++ là một lựa chọn phổ biến, đặc biệt đối với những lập trình viên mới bắt đầu, nhờ vào giao diện dễ sử dụng và các công cụ tích hợp mạnh mẽ.

Các tính năng chính của Dev C++:

Dev C++ cung cấp các tính năng hữu ích giúp lập trình viên dễ dàng phát triển phần mềm C++:

* Trình soạn thảo mã nguồn: Cung cấp các tính năng như kiểm tra cú pháp, tự động xếp dòng, gợi ý từ khóa, giúp lập trình viên viết mã nhanh chóng và chính xác.
* Trình biên dịch MinGW: Dev C++ sử dụng bộ biên dịch MinGW để biên dịch mã nguồn C++ thành các tệp thực thi mà không cần cài đặt thêm phần mềm bên ngoài.
* Chế độ gỡ lỗi: Cung cấp công cụ gỡ lỗi mạnh mẽ giúp lập trình viên tìm và sửa lỗi trong chương trình, đảm bảo chất lượng phần mềm.
* Quản lý dự án: Dev C++ cho phép người dùng tạo và quản lý các dự án lập trình, giúp tổ chức mã nguồn và tài liệu liên quan một cách hiệu quả.



Hình 1.11 Hình ảnh sử dụng Dev C++

## 1.3 THƯ VIỆN ĐỒ HỌA GRAPHICS.H

Để triển khai thuật toán đồ họa trên môi trường Windows, việc sử dụng Dev C++ có thể gặp khó khăn do sự hạn chế của hệ điều hành này trong việc hỗ trợ thư viện đồ họa cổ điển. Tuy nhiên, Michael đã sáng tạo một giải pháp hiệu quả bằng cách phát triển một môi trường giả lập đồ họa dựa trên Borland C, đồng thời cải tiến và chuyển đổi thư viện BGI (Borland Graphics Interface) thành một thư viện mới có tên là WinBGIm, được tối ưu hóa để hoạt động mượt mà trên Windows. Với WinBGIm, người lập trình có thể dễ dàng sử dụng các hàm đồ họa đặc trưng của Borland trong môi trường Dev C++.

Thư viện graphics.h, hay còn gọi là Borland Graphics Interface (BGI), đã trở thành một công cụ phổ biến trong lập trình đồ họa, đặc biệt trên các hệ điều hành DOS và Windows phiên bản cũ như Windows 95 và 98. Được phát triển và tích hợp vào IDE Borland C++ 3.1 vào năm 1992, BGI bao gồm hai thành phần chính: graphics.h và graphics.lib, giúp lập trình viên sử dụng dễ dàng trong môi trường C/C++ hoặc Pascal thông qua module graph.

A group of shapes on a black background

Description automatically generated

Hình 1.12 Minh hoa hình ảnh đồ họa graphic.h

Các chức năng chính của thư viện graphics.h bao gồm:

* Khởi tạo và kết thúc cửa sổ đồ họa.
* Vẽ các hình dạng như đường thẳng, hình chữ nhật, hình tròn, và các đối tượng đồ họa khác.
* Thêm văn bản vào cửa sổ đồ họa.
* Quản lý màu sắc và độ sáng của các yếu tố đồ họa.

# CHƯƠNG 2: PHƯƠNG PHÁP NGHIÊN CỨU

## 2.1 CÀI ĐẶT VÀ SỬ DỤNG MÔI TRƯỜNG ĐỒ HỌA

### 2.1.1 Cài đặt Dev-C++

Để cài đặt Dev C++, làm theo các bước sau:

Truy cập trang web của Dev C++ (<https://www.bloodshed.net/devcpp.html>) hoặc từ các nguồn tải phần mềm đáng tin cậy để tải phiên bản Dev C++ phù hợp với hệ điều hành của bạn

### 2.1.2 Cài đặt thư viện graphic.h

Do thư viện đồ họa graphics và winbgim không được tích hợp sẵn trong thư viện của phần mềm Dev C/ C++ nên ta phải tự thêm 2 thư viện đồ họa nếu muốn sử dụng để lập trình đồ họa.

Cài đặt **Graphics** theo đường link:

**(**<https://github.com/thinhdoanvu/ComputerGraphics/tree/master/Coding/library>) hoặc

<https://drive.google.com/file/d/1UK0316yZEZorN-MWCqxgIGBH79os6vvC/view>

*Thêm thư viện đồ họa*

Đường dẫn (Path):

* Với Windows 32 bit: Path=”C:\Program Files\Dev-Cpp”
* Windows 64 bit: Path=”C:\Program Files (x86)\Dev-Cpp”

*Hướng dẫn cài đặt:*

Sau khi giải nén file gốc, ta cần làm theo các bước sau:

Bước 1: Sao chép tệp tin đầu vào:

* Sao chép tệp tin graphics.h và winbgim.h từ thư mục đã giải nén vào thư mục include của Dev-C++.
* Đường dẫn thư mục: C:\Program Files (x86)\Dev-Cpp\MinGW64\include.
* Nếu cài đặt tại vị trí khác, tìm đến thư mục Dev-Cpp\MinGW64\include là được.

Bước 2: Sao chép tệp tin thư viện:

Sao chép tệp tin libbgi.a từ thư mục đã giải nén vào thư mục lib của Dev-C++.

* Đường dẫn thư mục: C:\Program Files (x86)\Dev-Cpp\MinGW64\lib.
* Nếu cài đặt tại vị trí khác, tìm đến thư mục Dev-Cpp\MinGW64\lib là được.

Bước 3: Cấu hình Dev C++:

Vì chương trình graphics.h chỉ hỗ trợ 32bit, nên để chạy được cần chuyển sang TDM-GCC 32bit, như hình:

A screenshot of a computer

Description automatically generated

Hình 2.1 Cài đăt môi trường đồ họa graphic.h

* Chọn **Tools** -> **Compiler Options**, tick chọn **Add the following commands when calling the linker**
* Copy các biến sau vào ô nhập:

*-static-libgcc -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32*

Kết quả như hình là được:

A screenshot of a computer

Description automatically generated

Hình 2.2 Hình ảnh cấu hình thư viện graphic.h vào Dev C++

* Chọn OK

Bước 4: Chạy chương trình.

## 2.2 TỔNG QUAN VỀ THIẾT KẾ GIAO DIỆN

### 2.2.1 Sơ lược giao diện

A screenshot of a computer

Description automatically generated

Hình 2.3 Giao diên bắt đàu mô tả thuật toán DFS

Giao diện mô phỏng DFS sử dụng thư viện đồ họa C++ (Graphics.h) để tạo ra giao diện trực quan, giúp người dùng có thể theo dõi và điều khiển quá trình tìm kiếm một cách dễ dàng. Giao diện của ứng dụng bao gồm các thành phần sau:

- Tiêu đề: Phía trên cùng của màn hình hiển thị dòng chữ "DFS GRAPH VISUALIZATION", báo hiệu đây là một công cụ mô phỏng thuật toán DFS.

- Khu vực đồ họa: Phần lớn màn hình là khu vực để người dùng có thể xem thông tin và vẽ đồ thị hoặc quan sát quá trình DFS. Các đỉnh và cạnh của đồ thị có thể được tạo ra và hiển thị trong khu vực này.

Các nút điều khiển:

* Load file: Cho phép người dùng tải tệp đồ thị từ bên ngoài vào chương trình. Điều này rất hữu ích khi người dùng muốn thử nghiệm với nhiều đồ thị khác nhau mà không cần phải tạo lại từ đầu.
* Starting: Bắt đầu thuật toán DFS từ một đỉnh xuất phát đã được người dùng chọn. Điều này sẽ kích hoạt thuật toán và bắt đầu quá trình tìm kiếm.
* Delete: Cho phép người dùng xóa các đỉnh hoặc cạnh trong đồ thị. Tính năng này hữu ích khi người dùng muốn thay đổi cấu trúc đồ thị để thử nghiệm với các tình huống khác nhau.
* Reset: Đặt lại tất cả, đưa đồ thị và trạng thái ứng dụng trở lại ban đầu, giúp người dùng bắt đầu lại từ đầu mà không cần khởi động lại chương trình.
* Author: Hiển thị thông tin về tác giả của ứng dụng, giúp người dùng biết được nguồn gốc và người phát triển công cụ này.
* Exit: Đóng ứng dụng và thoát khỏi chương trình.
* Giao diện giúp người dùng dễ dàng tương tác với thuật toán DFS bằng cách cung cấp các điều khiển để bắt đầu, xóa, hoàn tác, và quan sát từng bước trong quá trình tìm kiếm đồ thị.

Giao diện đồ thị được bao gồm:

A screenshot of a computer

Description automatically generated

Hình 2.4 Hình ảnh đồ thị vô hướng minh họa DFS

Hình ảnh hiển thị một đồ thị vô hướng, trong đó các đỉnh được đánh nhãn số, và các cạnh nối giữa chúng tạo thành các kết nối giữa các đỉnh.Các đỉnh được vẽ dưới dạng các vòng tròn, với nhãn số nằm trong mỗi vòng tròn. Các cạnh nối giữa các đỉnh được vẽ dưới dạng các đường thẳng nối giữa các vòng tròn, thể hiện mối quan hệ giữa các đỉnh.

Đồ thị biểu diễn của một mạng lưới với các kết nối giữa các điểm, và nó có thể được sử dụng để mô phỏng thuật toán tìm kiếm như DFS (Depth First Search). Giao diện đồ họa cho phép người dùng quan sát và tương tác với đồ thị.

### 2.2.2 Thiết kế giao diện

#### 2.2.2.1 Khởi tạo cửa sổ đồ hoạ BGI

Tại hàm chính main(), ta sử dụng hàm initwindow(width, height) với width =950 và height = 600 được sử dụng để khởi tạo một cửa sổ đồ họa có độ rộng là 950 và chiều cao là 600 trên màn hình do thư viện đồ hoạ graphics.h cung cấp.

A screenshot of a computer

Description automatically generated

Hình 2.5 Hình ảnh thiết kế cửa số đồ họa

#### 2.2.2.2 Xây dựng các thực thể trên cửa sổ BGI

**Xây dựng button xử lý**

Hàm drawButton được thiết kế để tạo ra một nút bấm (button) với hình chữ nhật và nhãn (label) hiển thị bên trong. Đây là một thành phần giao diện đồ họa cơ bản, thường được sử dụng trong các chương trình tương tác để biểu diễn các thao tác như bấm chọn.

*void drawButton(int left, int top, int right, int bottom, const char\* label) {*

*setcolor(WHITE);*

*setlinestyle(0, 1, 3);*

*rectangle(left, top, right, bottom);*

*settextstyle(10, 0, 1);*

*char buffer[MAX];*

*strcpy(buffer, label);*

*outtextxy(left + 10, top + 10, buffer);}*

Hàm thực hiện các bước chính như sau:

* Đặt màu của viền nút là màu trắng thông qua lệnh setcolor(WHITE);.
* Chọn kiểu nét liền và độ dày nét là 3 pixel bằng setlinestyle(0, 1, 3);.
* Sử dụng lệnh rectangle(left, top, right, bottom); để tạo một khung hình chữ nhật với các tọa độ được cung cấp.
* Đặt kiểu chữ đơn giản với kích thước nhỏ qua settextstyle(10, 0, 1);
* Vẽ chuỗi buffer (nhãn nút bấm) lên giao diện tại vị trí thích hợp. In nội dung của nút tại vị trí cách góc trên bên trái của hình chữ nhật 10 pixel bằng outtextxy(left + 10, top + 10, buffer).

A number with red and blue dots

Description automatically generated with medium confidence

Hình 2.6 Hình ảnh xử lý vẽ các nút

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.

**Xây dựng khung đồ họa:**

Hàm drawframe được sử dụng để vẽ khung giao diện đồ họa cho chương trình trực quan hóa thuật toán DFS. Đầu tiên, hàm đặt màu hiển thị là trắng và thiết lập kiểu chữ lớn để in tiêu đề “DFS GRAPH VISUALIZATION” tại vị trí (250, 20).

*//Ve khung do hoa*

*void drawframe() {*

*setcolor(WHITE);*

*settextstyle(3, 0, 3);*

*outtextxy(250, 20, (char\*)"DFS GRAPH VISUALIZATION");*

*setlinestyle(0, 1, 3);*

*rectangle(10, 10, 770, 50);*

*setlinestyle(0, 1, 3);*

*rectangle(10, 50, 770, 470);*

*setlinestyle(0, 1, 3);*

*rectangle(10, 410, 770, 470);*

*rectangle(10, 480, 770, 580);*

*settextstyle(10, 0, 1);*

*// Ve cac nut*

*drawButton(800, 10, 925, 60, "Load file");*

*drawButton(800, 90, 925, 140, "Starting");*

*drawButton(800, 170, 925, 220, "Delete");*

*drawButton(800, 250, 925, 300, "Reset");*

*drawButton(800, 330, 925, 380, "Author");*

*drawButton(800, 410, 925, 460, "Exit");}*

Hàm drawframe được sử dụng để vẽ khung giao diện đồ họa cho chương trình trực quan hóa thuật toán DFS. Đầu tiên, hàm đặt màu hiển thị là trắng và thiết lập kiểu chữ lớn để in tiêu đề “DFS GRAPH VISUALIZATION” tại vị trí (250, 20).

Sau đó, hàm vẽ bốn khung hình chữ nhật: khung trên cùng (10, 10, 770, 50) dùng làm thanh tiêu đề, khung chính giữa (10, 50, 770, 470) để hiển thị đồ thị hoặc thuật toán, khung phụ chính giữa (10, 410, 770, 470) để hiển thị menu nhập dữ liệu và khung dưới cùng (10, 480, 770, 580) để hiển thị thông tin bổ sung hoặc các nút điều khiển. Cuối cùng, kiểu chữ nhỏ được đặt cho khung dưới cùng để chuẩn bị hiển thị nội dung chi tiết kết quả.

**Xây dựng đồ thị:**

*//Ve do thi*

*void drawgraph(){*

*//Hien thi toa do cua dinh va mtk*

*settextstyle(1, 0, 3);*

*for (int i = 0; i < sodinh; i++) {*

*drawVertex(i, WHITE);*

*outtextxy(td[i].x - 10, td[i].y - 10, itoa(td[i].label, buffer, 10));*

*}*

*for (int i = 0; i < sodinh; i++) {*

*for (int j = 0; j < sodinh; j++) {*

*if (mtk[i][j] == 1) {*

*if (!isDirectedGraph()) {*

*drawEdge(i, j, WHITE);*

*} else {*

*drawEdgeCH(i, j, WHITE);*

*}}*

*}}}*

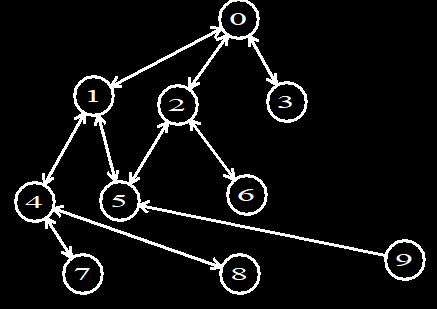
Hàm drawgraph dùng để vẽ đồ thị dựa trên tọa độ các đỉnh và ma trận kề (mtk). Trước tiên, hàm thiết lập kiểu chữ để hiển thị. Sau đó, vòng lặp đầu tiên duyệt qua các đỉnh (số lượng là sodinh), vẽ từng đỉnh dưới dạng hình tròn tại tọa độ (td[i].x, td[i].y) với bán kính 20 và hiển thị nhãn đỉnh (label) bằng lệnh outtextxy. Vòng lặp thứ hai kiểm tra ma trận kề, nếu mtk[i][j] == 1 (tức có cạnh nối giữa đỉnh i và đỉnh j), hàm sẽ vẽ đường thẳng nối hai đỉnh bằng hàm vẽ cạnh *drawEdge().* Trong trường hợp nếu đọc file đó là đồ thị đối xứng thì ta vẽ đồ thị vô hướng:

A white lines with numbers on it

Description automatically generated with medium confidence

Hình 2.7 Hình ảnh đồ thị vô hướng được vẽ

Trong trường hợp đồ thị load file là ma trận kề không đối xứng sẽ vẽ đồ thị có hướng:



Hình 2.8 Hình ảnh đồ thị có hướng được vẽ

**Xây dựng nút đỉnh:**

Hàm drawVertex thực hiện việc vẽ một đỉnh đồ họa dưới dạng hình tròn tại vị trí xác định, với màu sắc được chỉ định. Mỗi đỉnh trong đồ thị sẽ được đại diện bởi một hình tròn, giúp trực quan hóa các đỉnh trong đồ thị khi thực thi thuật toán tìm kiếm hoặc vẽ đồ thị.

*// Ve dinh*

*void drawVertex(int vertex, int color) {*

*setcolor(color);*

*circle(td[vertex].x, td[vertex].y, 20);}*

* setcolor(color);: Hàm này thiết lập màu sắc cho đối tượng đồ họa tiếp theo, trong trường hợp này là màu của đỉnh. Màu được xác định thông qua tham số color.
* circle(td[vertex].x, td[vertex].y, 20);: Hàm circle vẽ một hình tròn (đại diện cho đỉnh) tại tọa độ (td[vertex].x, td[vertex].y) với bán kính là 20. Tọa độ (td[vertex].x, td[vertex].y) được lấy từ mảng td chứa thông tin vị trí của các đỉnh trong đồ thị, với vertex là chỉ số của đỉnh cần vẽ.

**Xây dựng cạnh:**

*Cạnh vô hướng*:

Hàm drawEdge thực hiện việc vẽ một cạnh giữa hai đỉnh trong đồ thị dưới dạng đoạn thẳng, với màu sắc và kiểu đường được chỉ định. Hàm này giúp hiển thị các kết nối giữa các đỉnh trong đồ thị khi thuật toán đồ họa được thực thi, như trong trường hợp của thuật toán DFS.

*// Ve canh vo huong*

*void drawEdge(int vertex1, int vertex2, int color) {*

*setcolor(color);*

*setlinestyle(SOLID\_LINE, 0, 3);*

*// Tinh toan vector huong giua hai dinh*

*int dx = td[vertex2].x - td[vertex1].x;*

*int dy = td[vertex2].y - td[vertex1].y;*

*// Tinh toan do dài cua canh (quang duong giua 2 dinh)*

*float length = sqrt(dx \* dx + dy \* dy);*

*int radius = 20;*

*// tinh toan diem bat dau tren duong tron cua vertex1*

*float ratio1 = radius / length;*

*int startX = td[vertex1].x + ratio1 \* dx;*

*int startY = td[vertex1].y + ratio1 \* dy;*

*// Tinh toan diem ket thuc tren duong tròn cua vertex2*

*float ratio2 = radius / length;*

*int endX = td[vertex2].x - ratio2 \* dx;*

*int endY = td[vertex2].y - ratio2 \* dy;*

*line(startX, startY, endX, endY);}*

Hàm drawEdge được sử dụng để vẽ các cạnh không có hướng, tức là các cạnh không có một hướng cụ thể, mà chỉ đơn giản là một liên kết giữa hai đỉnh trong đồ thị. Các đỉnh được lưu trữ trong một mảng td[], với mỗi đỉnh có tọa độ (x, y). Cạnh không có hướng chỉ cần vẽ một đoạn thẳng giữa hai đỉnh, và không yêu cầu thêm thông tin nào về hướng. Ta tính sự chênh lệch về tọa độ giữa hai đỉnh, với vector hướng được xác định bằng cách tính dx = td[vertex2].x - td[vertex1].x và dy = td[vertex2].y - td[vertex1].y. Chiều dài của cạnh được tính bằng công thức Pythagoras: length = sqrt(dx \* dx + dy \* dy). Để tránh vẽ cạnh cắt qua chính các đỉnh, ta sẽ tính toán các điểm bắt đầu và kết thúc của đoạn thẳng sao cho chúng nằm trên đường tròn có bán kính cố định từ các đỉnh. Điểm bắt đầu tại vertex1 và điểm kết thúc tại vertex2 được xác định bằng cách điều chỉnh tọa độ sao cho chúng nằm cách xa trung tâm của các đỉnh một khoảng phù hợp. Cuối cùng, một đoạn thẳng (line) được vẽ giữa hai điểm bắt đầu và kết thúc.

*Cạnh có hướng:*

Hàm drawEdgeCH được sử dụng để vẽ các cạnh có hướng, tức là các cạnh này có một hướng đi cụ thể từ một đỉnh đến đỉnh kia. Cạnh có hướng không chỉ đơn giản là một đoạn thẳng mà còn có thêm một mũi tên để chỉ ra hướng của cạnh.

*// Ve canh huong di*

*void drawEdgeCH(int vertex1, int vertex2, int color) {*

*setcolor(color);*

*setlinestyle(SOLID\_LINE, 0, 3);*

*int dx = td[vertex2].x - td[vertex1].x;*

*int dy = td[vertex2].y - td[vertex1].y;*

*float length = sqrt(dx \* dx + dy \* dy);*

*int radius = 20;*

*// Tinh ty le ve canh sao cho tranh ve cat dinh*

*float ratio1 = radius / length;*

*int startX = td[vertex1].x + ratio1 \* dx;*

*int startY = td[vertex1].y + ratio1 \* dy;*

*float ratio2 = radius / length;*

*int endX = td[vertex2].x - ratio2 \* dx;*

*int endY = td[vertex2].y - ratio2 \* dy;*

*line(startX, startY, endX, endY);*

*int arrowSize = 10; // Kich thuoc mui ten*

*float arrowAngle = atan2(dy, dx); // Goc cua duong thang*

*float angle1 = arrowAngle + M\_PI / 6; // Goc mat phia cua mui ten*

*float angle2 = arrowAngle - M\_PI / 6; // Goc phia con lai*

*// Tinh toan cac diem cua mui ten*

*int x1 = endX - arrowSize \* cos(angle1);*

*int y1 = endY - arrowSize \* sin(angle1);*

*int x2 = endX - arrowSize \* cos(angle2);*

*int y2 = endY - arrowSize \* sin(angle2);*

*// Ve mui ten*

*line(endX, endY, x1, y1); // Mot canh cua mui ten*

*line(endX, endY, x2, y2); // Canh còn lai cua mui ten*

*}*

Cũng như trong hàm drawEdge, các điểm đầu và cuối của đoạn thẳng được tính toán từ tọa độ của các đỉnh. Ta tính toán chiều dài của cạnh và các điểm bắt đầu, kết thúc của đoạn thẳng sao cho không cắt vào các đỉnh. Đoạn thẳng giữa hai điểm bắt đầu và kết thúc được vẽ giống như trong hàm drawEdge. Để thể hiện hướng của cạnh, một mũi tên được vẽ ở cuối đoạn thẳng. Mũi tên được xác định bằng cách tính toán góc của cạnh với trục hoành (atan2(dy, dx)), từ đó tìm ra các góc phụ (angle1 và angle2) để tạo ra hai cạnh của mũi tên. Các điểm của mũi tên được tính toán và vẽ bằng hai đoạn thẳng nối từ điểm cuối của cạnh đến các đỉnh của mũi tên. Hai đoạn thẳng được vẽ từ điểm cuối của cạnh đến hai điểm của mũi tên, tạo thành một mũi tên chỉ ra hướng của cạnh.

#### 2.2.2.3 Các tác vụ xử lý đồ thị

**Xóa đồ thị:**

Hàm clearGraph được sử dụng để xóa đồ thị và các thông tin liên quan đến thuật toán DFS. Cụ thể, hàm này sẽ đặt biến isDFSExecuted về false, chỉ ra rằng thuật toán DFS chưa được thực thi. Sau đó, nó dọn dẹp các mảng CLOSE và path, thiết lập lại chiều dài đường đi (len) về 0, và cuối cùng xóa toàn bộ đồ họa trên màn hình bằng hàm cleardevice().

*//Xoa do thi*

*void clearGraph(){*

*isDFSExecuted=false;*

*memset(CLOSE, 0, sizeof(CLOSE));*

*memset(path, 0, sizeof(path));*

*len = 0;*

*cleardevice();}*

*//Xoa do thi*

*void cleanGraph(){*

*memset(CLOSE, 0, sizeof(CLOSE));*

*memset(path, 0, sizeof(path));*

*len = 0;*

*cleardevice();*

*isGraphLoaded = false;}//Gan ve false sau khi xoa do thi the hien chua nap*

Hàm cleanGraph có chức năng tương tự trong việc dọn dẹp các mảng CLOSE và path, cũng như thiết lập lại chiều dài đường đi. Tuy nhiên, điểm khác biệt là hàm này đặt biến isGraphLoaded về false, chỉ ra rằng đồ thị chưa được nạp vào chương trình. Sau đó, nó cũng xóa đồ họa trên màn hình.

**Xử lý tạo cạnh sau khi tạo nút**

Hàm drawEdges chịu trách nhiệm vẽ toàn bộ các cạnh của đồ thị dựa trên dữ liệu ma trận kề được load từ file. Ma trận kề là một cấu trúc dữ liệu biểu diễn đồ thị, trong đó phần tử tại vị trí [i][j] cho biết sự tồn tại của cạnh nối từ đỉnh i đến đỉnh j. Nếu phần tử này bằng 1, nghĩa là có một cạnh nối giữa hai đỉnh.

*void drawEdges() {*

*for (int i = 0; i < sodinh; i++) {*

*for (int j = 0; j < sodinh; j++) {*

*if (mtk[i][j] == 1) {*

*if (!isDirectedGraph()) {*

*drawEdge(i, j, WHITE);*

*} else {*

*drawEdgeCH(i, j, WHITE);*

*} }*

*}}}*

Hàm sử dụng hai vòng lặp for lồng nhau để duyệt qua toàn bộ các phần tử trong ma trận kề. Với mỗi cặp đỉnh (i, j), nếu ma trận chỉ ra rằng có cạnh nối giữa chúng, chương trình sẽ xác định loại đồ thị (có hướng hoặc không có hướng) bằng cách gọi hàm isDirectedGraph(). Nếu đồ thị không có hướng, hàm drawEdge được gọi để vẽ cạnh dưới dạng một đoạn thẳng đơn giản giữa hai đỉnh. Ngược lại, nếu đồ thị có hướng, hàm drawEdgeCH được gọi để vẽ cạnh dưới dạng đoạn thẳng có kèm mũi tên chỉ rõ hướng đi từ đỉnh i đến đỉnh j.

Trong quá trình này, màu của cạnh được đặt mặc định là WHITE. Đối với đồ thị không có hướng, chỉ những cạnh nằm trong một nửa ma trận (theo quy tắc ma trận đối xứng) sẽ được vẽ, đảm bảo rằng mỗi cạnh chỉ xuất hiện một lần. Còn với đồ thị có hướng, các cạnh từ đỉnh i đến đỉnh j và ngược lại được xử lý độc lập, vì chúng có thể có hướng khác nhau.

## 2.3 XỬ LÝ SỰ KIỆN CHUỘT

### 2.3.1 Xử lý sự kiện click vào đỉnh đồ thị

Xử lý sự kiện click chuột để người dùng chọn hai đỉnh trong đồ thị: đỉnh bắt đầu và đỉnh kết thúc. Quá trình này được chia thành hai phần rõ rệt: chọn đỉnh bắt đầu và chọn đỉnh kết thúc.

*start = -1, end = -1;*

*settextstyle(1, 0, 2);*

*setcolor(WHITE);*

*outtextxy(20, 420, (char\*)"Click dinh bat dau:");*

*while (start == -1) {*

*if (ismouseclick(WM\_LBUTTONDOWN)) {*

*int x, y;*

*getmouseclick(WM\_LBUTTONDOWN, x, y);*

*for (int i = 0; i < sodinh; i++) {*

*if (sqrt(pow(x - td[i].x, 2) + pow(y - td[i].y, 2)) <= 20) {// cong thuc khoang cach tu chuot den toa do dinh euclid*

*start = i;*

*setcolor(YELLOW);*

*circle(td[i].x, td[i].y, 20);*

*sprintf(buffer, "Dinh: %d", td[i].label);*

*outtextxy(420, 420, buffer);*

*break;}}}*

*setcolor(WHITE);*

*outtextxy(20, 440, (char\*)"Click dinh ket thuc:");*

*while (end == -1) {*

*if (ismouseclick(WM\_LBUTTONDOWN)) {*

*int x, y;*

*getmouseclick(WM\_LBUTTONDOWN, x, y);*

*for (int i = 0; i < sodinh; i++) {*

*if (sqrt(pow(x - td[i].x, 2) + pow(y - td[i].y, 2)) <= 20) {// cong thuc khoang cach tu chuot den toa do dinh euclid*

*end = i;*

*setcolor(YELLOW);*

*circle(td[i].x, td[i].y, 20);*

*sprintf(buffer, "Dinh: %d", td[i].label);*

*outtextxy(420, 440, buffer);*

*break;*

*}}}}*

**Chọn đỉnh bắt đầu**: Đầu tiên, chương trình yêu cầu người dùng chọn một đỉnh làm điểm bắt đầu cho thuật toán. Vòng lặp while (start == -1) sẽ tiếp tục cho đến khi người dùng chọn một đỉnh hợp lệ. Khi người dùng click chuột vào một đỉnh trong đồ thị, chương trình sẽ lấy tọa độ của chuột qua hàm getmouseclick và kiểm tra khoảng cách giữa chuột và các đỉnh trong đồ thị.

* ismouseclick(WM\_LBUTTONDOWN): Kiểm tra xem có sự kiện nhấn chuột trái (click) không.
* getmouseclick(WM\_LBUTTONDOWN, x, y);: Lấy tọa độ (x, y) của vị trí chuột khi nhấn chuột trái.

Khoảng cách này được tính theo công thức Euclid: sqrt(pow(x - td[i].x, 2) + pow(y - td[i].y, 2)), trong đó (x, y) là tọa độ chuột và (td[i].x, td[i].y) là tọa độ của đỉnh thứ i. Nếu khoảng cách này nhỏ hơn hoặc bằng 20 (bán kính của đỉnh), chương trình sẽ xác định rằng người dùng đã click vào đỉnh đó. Khi đó, chỉ số của đỉnh đó sẽ được lưu vào biến start, và đỉnh được tô màu vàng để hiển thị rõ ràng cho người dùng. Thông tin về đỉnh cũng được hiển thị trên màn hình.

**Chọn đỉnh kết thúc**: Sau khi người dùng chọn đỉnh bắt đầu, chương trình yêu cầu người dùng chọn một đỉnh khác làm đỉnh kết thúc. Quy trình chọn đỉnh kết thúc tương tự như chọn đỉnh bắt đầu. Khi người dùng click chuột vào một đỉnh, chương trình sẽ tính toán khoảng cách giữa chuột và các đỉnh, và nếu chuột click vào một đỉnh, chương trình sẽ gán chỉ số của đỉnh đó vào biến end. Đỉnh kết thúc sẽ được tô màu xanh để phân biệt với đỉnh bắt đầu. Thông tin về đỉnh kết thúc cũng được hiển thị tương tự như đỉnh bắt đầu.

A screenshot of a computer game

Description automatically generated

Hình 2.9 Hình ảnh khi click các đỉnh bắt đầu và kết thúc

### 2.3.2 Xử lý sự kiện hover chuột:

Hàm handleMouseHover dùng để xử lý hiệu ứng khi chuột di chuyển vào vùng của một nút bấm trên giao diện đồ họa.

*//Xu ly khi hover vao button*

*void handleMouseHover(int left, int top, int right, int bottom, const char\* label, bool &hovered) {*

*char buffer[MAX]; // Khai báo buffer luu du lieu chuoi*

*if (isMouseHover(left, top, right, bottom)) {*

*setcolor(RED);*

*rectangle(left, top, right, bottom);*

*settextstyle(10, 0, 1);*

*sprintf(buffer, "%s", label); //Dinh dang chuoi duoc luu vao buffer*

*outtextxy(left + 10, top + 10, buffer); // Hien thi noi dung cua buffer*

*hovered = true;*

*} else if (hovered) {*

*setcolor(WHITE);*

*rectangle(left, top, right, bottom);*

*settextstyle(10, 0, 1);*

*sprintf(buffer, "%s", label);*

*outtextxy(left + 10, top + 10, buffer);*

*hovered = false;*

*}}*

Hàm handleMouseHover xử lý việc thay đổi màu sắc của nút khi chuột di chuyển vào hoặc ra khỏi vùng nút. Hàm này nhận vào các tham số tương tự như isMouseHover cùng với một tham số hovered để theo dõi trạng thái của nút. Nếu chuột đang "hover" trong vùng nút (kiểm tra bằng isMouseHover), hàm sẽ thay đổi màu viền nút thành đỏ (setcolor(RED)) và vẽ lại nút bằng lệnh rectangle. Nội dung của nút (được lưu trong biến label) sẽ được hiển thị bên trong nút, và trạng thái hovered được cập nhật thành true.

Nếu chuột di chuyển ra khỏi vùng nút, hàm sẽ vẽ lại nút với màu trắng (setcolor(WHITE)) và hiển thị lại nhãn cũ. Trạng thái hovered được đặt lại thành false để phản ánh rằng chuột không còn "hover" trên nút nữa.

*void mouse() {*

*int x;int y;*

*//Bien danh dau hover*

*bool hovered = false; bool hovered1 = false;*

*bool hovered2 = false; bool hovered3 = false;*

*bool hovered4 = false; bool hovered5 = false;*

*while (!kbhit()) {*

*if (ismouseclick(WM\_MOUSEMOVE)) {*

*getmouseclick(WM\_MOUSEMOVE, x, y);*

*handleMouseHover(800, 10, 925, 60, "Load file", hovered);*

*handleMouseHover(800, 90, 925, 140, "Starting", hovered1);*

*handleMouseHover(800, 170, 925, 220, "Delete", hovered2);*

*handleMouseHover(800, 250, 925, 300, "Reset", hovered3);*

*handleMouseHover(800, 330, 925, 380, "Author", hovered4);*

*handleMouseHover(800, 410, 925, 460, "Exit", hovered5);*

*clearmouseclick(WM\_MOUSEMOVE);}*

*if (ismouseclick(WM\_LBUTTONDOWN)) {*

*getmouseclick(WM\_LBUTTONDOWN, x, y);*

*handleClick(x, y);*

*clearmouseclick(WM\_LBUTTONDOWN);*

*}}}*

Hàm mouse quản lý việc xử lý sự kiện chuột, bao gồm việc theo dõi chuyển động của chuột (hover) và nhận click chuột để thực hiện các hành động tương ứng.

Đầu tiên, các biến x và y được khai báo để lưu tọa độ của con trỏ chuột. Các biến hovered, hovered1, ... hovered5 được sử dụng để theo dõi trạng thái hover của từng nút bấm trên giao diện. Mỗi nút bấm có một biến hovered riêng biệt để kiểm tra chuột có di chuyển vào vùng nút đó hay không.

Vòng lặp while (!kbhit()) giữ cho hàm chạy liên tục cho đến khi có phím được nhấn. Bên trong vòng lặp, hàm kiểm tra nếu chuột di chuyển (ismouseclick (WM\_MOUSEMOVE)). Nếu có, hàm lấy tọa độ chuột bằng getmouseclick (WM\_MOUSEMOVE, x, y) và sau đó gọi hàm handleMouseHover cho từng nút bấm (Load file, Starting, Delete, Reset, Author, Exit) để kiểm tra và xử lý hiệu ứng hover cho mỗi nút. Hàm handleMouseHover sẽ thay đổi màu sắc và hiển thị nội dung của nút khi chuột di chuyển vào vùng của nút.

Khi người dùng nhấn chuột trái (ismouseclick (WM\_LBUTTONDOWN)), hàm sẽ lấy tọa độ chuột và gọi hàm handleClick (x, y) để xử lý hành động tương ứng với vị trí nút người dùng đã click. Sau đó, sự kiện click chuột được xóa bằng clearmouseclick (WM\_LBUTTONDOWN) để chuẩn bị cho lần nhấn chuột tiếp theo.



Hình 2.10 Hình ảnh hover vào button

### 2.3.3 Xử lý thao tác chuột khi click

Hàm handleClick xử lý các sự kiện click chuột trong ứng dụng đồ họa, với mục đích thực hiện các hành động khi người dùng nhấn vào các nút trên giao diện, tương ứng với các chức năng: mở file, bắt đầu thuật toán DFS, reset đồ thị, xóa dữ liệu, và thoát ứng dụng.

*if(x >= 800 && x <= 925 && y >= 10 && y <= 60){*

*OPENFILENAME ofn;*

*TCHAR szFile[MAX\_PATH];*

*TCHAR szFolder[MAX\_PATH];*

*// Lay duong dan den thu muc hien tai cua chuong trinh*

*GetCurrentDirectory(MAX\_PATH, szFolder);*

*ZeroMemory(&ofn, sizeof(OPENFILENAME));*

*ofn.lStructSize = sizeof(OPENFILENAME);*

*ofn.hwndOwner = NULL;*

*ofn.lpstrFile = szFile; ofn.lpstrFile[0] = '\0';*

*ofn.nMaxFile = sizeof(szFile);*

*ofn.lpstrFilter = TEXT("All Files (\*.\*)\0\*.\*\0");*

*ofn.nFilterIndex = 1; ofn.lpstrInitialDir = szFolder;*

*//Su dung thu muc hien tai lam thu muc mac dinh*

*if (GetOpenFileName(&ofn)) {*

*const char\* filePath = ofn.lpstrFile;*

*openAndReadFile(filePath);*

*ZeroMemory(szFile, sizeof(szFile));*

*clearGraph();*

*drawframe();*

*nodeCount = 0;*

*isCreatingNodes = true;*

*settextstyle(1, 0, 2);*

*setcolor(WHITE);*

*outtextxy(20, 490, (char\*)"CLICK CHUOT VAO DE TAO NUT.");*

*}}*

Khi người dùng nhấn vào nút "Load file", hàm kiểm tra xem tọa độ click có nằm trong vùng của nút này (từ x = 800 đến 925, y = 10 đến 60). Đầu tiên, chương trình sử dụng các cấu trúc dữ liệu và hàm trong Windows API để hiển thị hộp thoại chọn file(thư viện window.h). Hai mảng ký tự szFile và szFolder được khai báo để lưu đường dẫn của file được chọn và thư mục hiện tại của chương trình. Biến OPENFILENAME ofn được sử dụng để thiết lập và truyền các thông tin cấu hình cần thiết cho hộp thoại.

Sau khi khởi tạo, hàm GetCurrentDirectory được gọi để lấy đường dẫn của thư mục hiện tại, lưu vào mảng szFolder. Điều này đảm bảo rằng hộp thoại chọn file sẽ mặc định mở ở thư mục hiện tại của chương trình. Tiếp theo, toàn bộ cấu trúc OPENFILENAME được khởi tạo bằng ZeroMemory, và các tham số như kích thước, bộ lọc file (chỉ hiển thị tất cả các loại file), và thư mục mặc định được thiết lập.

Khi người dùng chọn một file và nhấn nút "Open", hàm GetOpenFileName sẽ trả về true. Lúc này, đường dẫn file được lưu trong trường lpstrFile của OPENFILENAME. Chương trình lấy đường dẫn này thông qua con trỏ filePath và gọi hàm openAndReadFile để đọc nội dung file và xử lý theo logic riêng. Sau khi đọc file xong, chương trình thực hiện một số thao tác dọn dẹp , làm mới giao diện đồ họa và số lượng đỉnh cần tạo, rồi thông báo click mouse để tạo các đỉnh đồ thị.

Các thao tác khác:

*//Author*

*if(x >= 800 && x <= 925 && y >= 330 && y <= 380){*

*cleanGraph(); drawframe();*

*settextstyle(10, 0, 1);*

*outtextxy(250, 130, (char\*)"GVHD: Th.S. DOAN VU THINH");*

*outtextxy(250, 180, (char\*)"SVTH: NGUYEN MINH TAI");*

*outtextxy(250, 230, (char\*)"MSSV: 64132077");}*

Khi người dùng nhấn vào nút "Author", chương trình sẽ hiển thị thông tin về giảng viên và sinh viên (bao gồm tên và mã sinh viên) tại vị trí cố định trên giao diện. Các lệnh outtextxy được sử dụng để hiển thị văn bản này.

*//Start*

*if (x >= 800 && x <= 925 && y >= 90 && y <= 140) {*

*if ((isGraphLoaded || isResetMode) ) {*

*if(nodeCount == sodinh){*

*if (!isDFSExecuted) {*

*DFS\_display();*

*isDFSExecuted = true;*

*} else {*

*cleanGraph(); drawframe();*

*drawgraph(); DFS\_display();*

*isGraphLoaded = true; }*

*}}*

*else{*

*setcolor(WHITE);*

*cleanGraph(); drawframe();*

*outtextxy(295, 140, (char\*)"NO GRAPH TO START");*

*outtextxy(160, 190, (char\*)"CLICK THE 'Load file' BUTTON TO ADD GRAPH");*

*}}*

Nút "Start, khi nhấp vào vùng này, chương trình sẽ kiểm tra xem đồ thị đã được tải lên hay chưa bằng cách kiểm tra giá trị của biến isGraphLoaded. Nếu đồ thị đã sẵn sàng, chương trình tiếp tục kiểm tra biến isDFSExecuted để đảm bảo thuật toán DFS chưa được chạy trước đó. Nếu tất cả điều kiện đều thỏa mãn, thuật toán DFS được thực thi và trực quan hóa bằng hàm DFS\_display(). Đồng thời, trạng thái của chương trình được cập nhật: biến isDFSExecuted được đặt thành true để ngăn ngừa việc chạy lại DFS không cần thiết, và isGraphLoaded được đặt thành true để ghi nhận rằng đồ thị đã sẵn sàng. Trong trường hợp đồ thị chưa được nạp, chương trình sẽ hiển thị thông báo lỗi trên giao diện, chẳng hạn như “NO GRAPH TO START” và “CLICK THE 'Load file' BUTTON TO ADD GRAPH”.

*//Reset*

*if (x >= 800 && x <= 925 && y >= 250 && y <= 300) {*

*if (nodeCount > 0) {*

*cleanGraph();drawframe();*

*isCreatingNodes = true;*

*isDFSExecuted = false; isResetMode = true;*

*// Xoa toàn bo toa do cac nut*

*for (int i = 0; i < sodinh; i++) {*

*td[i].x = 0;*

*td[i].y = 0;*

*td[i].label = -1; }// Dat gia tri mac dinh cho label*

*nodeCount = 0;*

*settextstyle(1, 0, 2);*

*setcolor(WHITE);*

*outtextxy(20, 490, (char\*)"SUCCESS! CLICK CHUOT VAO DE TAO NUT.");}*

*isGraphLoaded = false;}*

Chức năng "Reset" trong chương trình, cho phép người dùng xóa toàn bộ dữ liệu của đồ thị hiện tại và bắt đầu tạo lại từ đầu. Khi người dùng nhấp chuột vào vùng tương ứng với nút Reset (tọa độ từ (800, 250) đến (925, 300)), chương trình kiểm tra nếu đồ thị hiện tại có ít nhất một đỉnh (nodeCount > 0). Nếu điều kiện thỏa mãn, các bước xử lý để thiết lập lại đồ thị được thực hiện. Hàm cleanGraph() được gọi để xóa toàn bộ các phần tử đồ thị đã vẽ trên giao diện, và hàm drawframe() được sử dụng để vẽ lại khung chương trình, chuẩn bị cho việc tạo đồ thị mới. Đồng thời, các biến trạng thái trong chương trình cũng được đặt lại: isCreatingNodes được đặt thành true để cho phép người dùng bắt đầu tạo lại các đỉnh, isDFSExecuted đặt thành false để khởi tạo lại trạng thái thuật toán DFS, và isResetMode đặt thành true để xác định rằng chương trình đang ở chế độ reset. Dữ liệu của các đỉnh trong đồ thị được xóa hoàn toàn. Vòng lặp duyệt qua danh sách các đỉnh (sodinh), đặt tọa độ của mỗi đỉnh về giá trị mặc định là 0, đồng thời nhãn của các đỉnh (td[i].label) được đặt về -1, cho thấy rằng chưa có đỉnh nào được gán nhãn. Biến nodeCount cũng được đặt lại bằng 0, biểu thị rằng đồ thị hiện tại không còn đỉnh nào.

Cuối cùng, một thông báo được hiển thị trên giao diện với nội dung: "SUCCESS! CLICK CHUOT VAO DE TAO NUT." để hướng dẫn người dùng rằng họ có thể bắt đầu lại việc tạo đồ thị mới. Biến isGraphLoaded được đặt thành false, báo hiệu rằng đồ thị hiện tại không được tải từ file mà sẽ được tạo thủ công.

*//Delete*

*if (x >= 800 && x <= 925 && y >= 170 && y <= 220) {*

*isResetMode= false;*

*cleanGraph();*

*drawframe();*

*outtextxy(280, 140, (char\*)"DATA HAS BEEN DELETED");*

*outtextxy(160, 190, (char\*)"CLICK THE 'Load file' BUTTON TO ADD GRAPH");}*

Chức năng Delete được kích hoạt khi người dùng nhấp chuột vào vùng tọa độ 800 ≤ x ≤ 925 và 170 ≤ y ≤ 220. Chức năng này cho phép người dùng xóa toàn bộ dữ liệu đồ thị hiện tại. Sau khi tắt isResetMode, chương trình thực hiện xóa giao diện với cleanGraph(), vẽ lại khung bằng drawframe(), và hiển thị thông báo trên màn hình.

*//Exit*

*if (x >= 800 && x <= 925 && y >= 410 && y <= 460) {*

*int msgboxID = MessageBox(NULL, TEXT("Ban có chac chan muon thoat?"), TEXT("Xac nhan"), MB\_ICONQUESTION | MB\_OKCANCEL);*

*if (msgboxID == IDOK) {*

*closegraph();*

*exit(0); }}*

Chức năng Exit được kích hoạt khi người dùng nhấp chuột vào vùng tọa độ 800 ≤ x ≤ 925 và 410 ≤ y ≤ 460. Đây là chức năng thoát chương trình, được thiết kế để ngăn ngừa các thao tác thoát nhầm. Khi kích hoạt, một hộp thoại xác nhận xuất hiện bằng hàm MessageBox() với nội dung "Bạn có chắc chắn muốn thoát?". Người dùng có thể chọn "OK" để xác nhận hoặc "Cancel" để hủy thao tác. Nếu "OK" được chọn, chương trình sẽ đóng giao diện đồ họa bằng closegraph() và kết thúc với exit(0)

### 2.3.4 Click chọn phương thức input:

Đoạn mã sau đây được sử dụng để tạo một giao diện đồ họa, trong đó người dùng được yêu cầu chọn một trong hai phương thức nhập liệu:

1. Nhập dữ liệu bằng bàn phím.
2. Nhập dữ liệu bằng cách nhấp chuột.

*int choice = -1;*

*while (choice == -1) {*

*if (ismouseclick(WM\_LBUTTONDOWN)) {*

*int x, y;*

*getmouseclick(WM\_LBUTTONDOWN, x, y);*

*if (x >= 20 && x <= 120 && y >= 440 && y <= 460) {*

*choice = 1; // Chon nhap phim*

*} else if (x >= 340 && x <= 500 && y >= 440 && y <= 460) {*

*choice = 2; // Chon mouse click*

*} } }*

*setcolor(BLACK);*

*outtextxy(20, 420, (char\*)"Chon cach nhap: ");*

*outtextxy(20, 440, (char\*)"1. Enter");*

*outtextxy(340, 440, (char\*)"2. Click");*

Các dòng chữ này được đặt ở các vị trí xác định trên màn hình, đóng vai trò như các nút bấm để người dùng tương tác. Chương trình sau đó sử dụng biến choice với giá trị ban đầu là -1 để theo dõi trạng thái lựa chọn của người dùng. Vòng lặp while được sử dụng để duy trì trạng thái "chờ" cho đến khi người dùng nhấn chuột vào một trong hai tùy chọn. Khi có sự kiện nhấn chuột trái (được kiểm tra bằng hàm ismouseclick), chương trình lấy tọa độ vị trí nhấn bằng hàm getmouseclick.



Hình 2.11 Hình ảnh hiển thị phương thức nhập

Dựa trên tọa độ này, chương trình xác định xem người dùng đã nhấn vào vùng nào trên màn hình. Nếu tọa độ nằm trong vùng từ (20, 420) đến (100, 440), chương trình gán giá trị 1 cho biến choice, tương ứng với lựa chọn nhập bằng bàn phím. Nếu tọa độ nằm trong vùng từ (340, 420) đến (430, 440), giá trị 2 được gán cho choice, tương ứng với lựa chọn nhập bằng chuột. Khi một lựa chọn hợp lệ được thực hiện, vòng lặp dừng và chương trình tiếp tục xử lý các thao tác tiếp theo. Sau khi chọn thì thông báo chọn cách nhập sẽ xóa đi.

### 2.3.5 Click tạo node:

Để mô tả một chức năng trong quá trình tạo đồ thị, nơi người dùng có thể nhấp chuột để tạo các nút (nodes) trên màn hình đồ họa. Chức năng này chỉ được kích hoạt khi người dùng đang ở chế độ tạo nút và chưa đạt đến số lượng nút tối đa được định sẵn. Mục đích của đoạn mã là cho phép người dùng tự do tạo ra các nút trên một không gian đồ họa, đảm bảo các nút không bị chồng lấn và được sắp xếp hợp lý.

*if (isCreatingNodes && nodeCount < sodinh) {*

*int xCenter = mousex();*

*int yCenter = mousey();*

*if (xCenter >= 30 && xCenter <= 750 && yCenter >= 70 && yCenter <= 385){*

*// Neu dang tao node và chua du so luong node*

*if (isCreatingNodes && nodeCount < sodinh) {*

*if (isPositionValid(xCenter, yCenter)) {*

*// Luu taa do và label cua node*

*td[nodeCount].x = xCenter;*

*td[nodeCount].y = yCenter;*

*td[nodeCount].label = nodeCount;*

*drawVertex(nodeCount, WHITE);*

*settextstyle(1, 0, 2);*

*char buffer[10];*

*itoa(td[nodeCount].label, buffer, 10);*

*outtextxy(xCenter - 10, yCenter - 10, buffer);*

*nodeCount++;*

*// Neu da tao du so luong node, dung tao node*

*if (nodeCount >= sodinh) {*

*isCreatingNodes = false;*

*drawEdges();*

*}}}} }*

Đầu tiên kiểm tra điều kiện isCreatingNodes && nodeCount < sodinh, nghĩa là chương trình chỉ cho phép tạo nút khi người dùng đang ở chế độ tạo nút (isCreatingNodes = true) và số lượng nút hiện tại (nodeCount) chưa đạt đến số lượng tối đa (sodinh), là giới hạn đã được định trước. Nếu một trong hai điều kiện này không thỏa mãn, quá trình tạo nút sẽ không tiếp tục. Khi điều kiện trên được thỏa mãn, chương trình lấy tọa độ hiện tại của chuột trên màn hình bằng cách sử dụng hàm mousex() và mousey(), lưu chúng vào các biến xCenter và yCenter. Đây là tọa độ nơi người dùng nhấp chuột và dự kiến sẽ là vị trí của nút mới.

Kiểm tra xem vị trí chuột có nằm trong vùng cho phép tạo nút hay không. Điều kiện if (xCenter >= 30 && xCenter <= 750 && yCenter >= 70 && yCenter <= 385) đảm bảo rằng người dùng chỉ có thể tạo nút trong một khu vực giới hạn trên màn hình, từ tọa độ (30, 70) đến (750, 385). Điều này giúp tránh việc người dùng vô tình tạo nút ngoài khu vực đồ thị.

Sau khi xác nhận chuột nằm trong khu vực hợp lệ, hàm isPositionValid(xCenter, yCenter) được gọi để kiểm tra xem vị trí mà người dùng nhấp chuột có hợp lệ hay không. Hàm này đảm bảo rằng nút mới không bị chồng lấn lên các nút đã tồn tại trước đó và khoảng cách giữa các nút đủ lớn để dễ dàng nhận diện.

Nếu vị trí chuột hợp lệ, tọa độ của nút mới và nhãn của nó sẽ được lưu vào mảng td[nodeCount], trong đó xCenter và yCenter là tọa độ của nút, còn nodeCount là nhãn của nút (được sử dụng để đánh số thứ tự các nút). Sau khi lưu thông tin, hàm drawVertex(nodeCount, WHITE) được gọi để vẽ nút tại vị trí đã xác định. Đồng thời, nhãn của nút (số thứ tự) sẽ được hiển thị gần nút bằng cách sử dụng hàm itoa() để chuyển số nhãn thành chuỗi và hàm outtextxy() để hiển thị chuỗi nhãn đó trên màn hình.

Sau khi vẽ nút, biến nodeCount được tăng lên một đơn vị, phản ánh số lượng nút đã được tạo. Nếu số lượng nút đã đủ (nodeCount >= sodinh), quá trình tạo nút sẽ kết thúc. Lúc này, cờ isCreatingNodes sẽ được đặt thành false, dừng chế độ tạo nút mới. Đồng thời, hàm drawEdges() được gọi để vẽ các cạnh của đồ thị nối giữa các nút đã được tạo.

A screenshot of a video game

Description automatically generated

Hình 2.12 Hình ảnh hiển thị phương thức nhập

### 2.3.6 Các hàm bổ trợ xử lý:

Kiểm tra khoảng cách các đỉnh:

*const int MIN\_DISTANCE = 2 \* 20 + 10; // Khoang cach toi thieu giua hai nut (bao gom khoang trang)( r=20)*

*// Ham kiem tra khoang cach giua cac nut*

*bool isPositionValid(int x, int y) {*

*for (int i = 0; i < nodeCount; i++) {*

*int dx = td[i].x - x;*

*int dy = td[i].y - y;*

*int distanceSquared = dx \* dx + dy \* dy;*

*if (distanceSquared < MIN\_DISTANCE \* MIN\_DISTANCE) {*

*return false;*

*} }*

*return true; }*

Hàm isPositionValid kiểm tra xem vị trí của một nút mới có hợp lệ hay không dựa trên khoảng cách tối thiểu giữa các nút trong đồ thị. Hằng số MIN\_DISTANCE xác định khoảng cách tối thiểu giữa các nút, bao gồm đường kính của nút (20) và một khoảng trống thêm (10). Hàm duyệt qua các nút đã tồn tại và tính khoảng cách bình phương giữa vị trí mới và các nút cũ. Nếu khoảng cách này nhỏ hơn bình phương của MIN\_DISTANCE, hàm trả về false, cho thấy vị trí không hợp lệ. Nếu không có vấn đề, hàm trả về true, cho phép thêm nút mới vào vị trí đó. Hàm này giúp đảm bảo các nút trong đồ thị không chồng lấn và có khoảng cách hợp lý, tạo ra bố cục trực quan dễ nhìn.

Kiểm tra hover

*//Kiem tra hover*

*bool isMouseHover(int left, int top, int right, int bottom) {*

*int x = mousex();*

*int y = mousey();*

*if (x >= left && x <= right && y >= top && y <= bottom) {*

*return true; // Chuot di chuyen vao trong vung cua button}*

*return false; // Chuot di chuyen ra khoi vung cua button}*

Hàm isMouseHover kiểm tra xem chuột có đang ở trong vùng của một nút hay không. Nó lấy tọa độ chuột bằng mousex() và mousey(), sau đó so sánh tọa độ chuột với tọa độ của vùng nút được xác định bởi các tham số left, top, right, và bottom. Nếu chuột nằm trong vùng nút (tức là x và y nằm trong khoảng từ left đến right, từ top đến bottom), hàm trả về true, nghĩa là chuột đang "hover" (di chuyển vào) vùng nút; nếu không, trả về false.

## 2.4 XỬ LÝ VÀ PHÂN TÍCH THUẬT TOÁN

### 2.4.1 Khởi tạo chương trình

#### 2.4.1.1 Khai báo

Những khai báo đảm bảo rằng chương trình có đầy đủ công cụ để thực hiện các chức năng cần thiết, từ xử lý toán học, nhập/xuất dữ liệu, thao tác đồ họa, đến tương tác với hệ điều hành. Sau khi khai báo, chương trình sẽ tiếp tục với việc khởi tạo và triển khai các hàm cụ thể để thực hiện các mục tiêu đã đề ra.

Phần khai báo chương trình là bước đầu tiên và quan trọng trong việc viết một ứng dụng sử dụng đồ họa trong ngôn ngữ lập trình C. Để thực hiện các thao tác đồ họa, chương trình phải bao gồm các thư viện cần thiết.

**Khai báo thư viện:**

Các thư viện C++ và Windows để xây dựng ứng dụng đồ họa và xử lý dữ liệu. stdio.h và stdlib.h hỗ trợ nhập/xuất dữ liệu và quản lý bộ nhớ. iostream giúp nhập và xuất dữ liệu trong C++, trong khi windows.h tương tác với hệ điều hành Windows. graphics.h cung cấp công cụ vẽ đồ họa 2D, và math.h cung cấp các hàm toán học. Tất cả các thư viện này phối hợp giúp xây dựng ứng dụng đồ họa và tính toán trong môi trường Windows.

A screenshot of a computer code

Description automatically generated

Hình 2.13 Hình ảnh khai báo các thư viện

**Khai báo biến dữ liệu:**

Các biến cờ được khai báo để theo dõi các trạng thái của chương trình. Biến isResetMode được dùng để xác định xem chương trình có đang ở chế độ reset hay không, tức là liệu đồ thị có bị xóa đi và bắt đầu lại không. Biến isGraphLoaded kiểm tra xem đồ thị có được tải vào chưa, giúp chương trình yêu cầu người dùng tải đồ thị trước khi thực hiện các thao tác khác. Biến isDFSExecuted giúp theo dõi việc thuật toán DFS đã được thực hiện hay chưa, ngăn không cho thuật toán chạy lại nếu không cần thiết.

A screenshot of a computer code

Description automatically generated

Hình 2.14 Hình ảnh khai báo biến xử lý thuật toán

Để triển khai thuật toán DFS, một số mảng và biến được khai báo. Mảng dothi[MAX][MAX] dùng để biểu diễn đồ thị dưới dạng ma trận, lưu trữ các cạnh nối giữa các đỉnh. Biến path[MAX] lưu trữ đường đi tìm được từ đỉnh nguồn đến đỉnh đích trong quá trình tìm kiếm. Biến len lưu trữ độ dài của đường đi tìm thấy. Các mảng OPEN[MAX] và CLOSE[MAX] dùng để theo dõi các đỉnh trong đồ thị: OPEN chứa các đỉnh chưa được xét, còn CLOSE chứa các đỉnh đã được xét. Mảng CHA[MAX] lưu trữ chỉ số đỉnh cha của mỗi đỉnh trong quá trình tìm kiếm. Mảng OPENED[MAX] giúp theo dõi các đỉnh đã được đưa vào danh sách xét duyệt.

Trong phần đồ họa, biến sodinh lưu số lượng đỉnh trong đồ thị, còn cấu trúc toado td[MAX] lưu tọa độ (x, y) và nhãn của từng đỉnh để phục vụ việc hiển thị đồ thị.Để xử lý các thao tác với file, con trỏ fp được khai báo để đọc hoặc ghi dữ liệu từ/to file, và mảng buffer[10] dùng làm bộ đệm tạm thời khi nhập dữ liệu từ file. Ma trận kề mtk[MAX][MAX] được sử dụng để lưu trữ thông tin kết nối giữa các đỉnh trong đồ thị, phục vụ việc xác định các cạnh nối giữa các đỉnh trong quá trình thực thi

**Khai báo tệp tin dữ liêu:**

Đây là một tập tin chứa dữ liệu mô tả một đồ thị không có hướng.

A screenshot of a computer screen

Description automatically generated

Hình 2.15 Ma trận kề đồ thị vô hướng của các đỉnh trong tệp

Đây là một tập tin chứa dữ liệu mô tả một đồ thị có hướng:

A screenshot of a computer code

Description automatically generated

Hình 2.16 Ma trận kề đồ thị vô hướng của các đỉnh trong tệp

Ma trận kề của đồ thị, với mỗi dòng và mỗi cột đại diện cho một đỉnh. Các giá trị trong ma trận là 0 hoặc 1, trong đó 1 biểu thị sự tồn tại của một cạnh giữa hai đỉnh, còn 0 biểu thị không có cạnh. Ví dụ, dòng đầu tiên "0 1 1 1 0 0 0 0 0 0" có nghĩa là đỉnh 0 có cạnh nối với các đỉnh 1, 2, 3, còn lại không có cạnh nào. Tương tự, các dòng tiếp theo mô tả các mối quan hệ kết nối giữa các đỉnh trong đồ thị. Ma trận kề này giúp xác định các đỉnh nào được kết nối với nhau trong đồ thị.

#### 2.4.1.3 Input dữ liêu thực thi

**Đọc file khởi tạo đồ thị:**

Hàm openAndReadFile có chức năng mở và đọc dữ liệu từ một tệp tin chứa thông tin về đồ thị, bao gồm số đỉnh và ma trận kề.

*//Doc file toa do va ma tran ke*

*void openAndReadFile(const char\* filePath) {*

*fp = fopen(filePath, "r");*

*if (fp == NULL) {*

*printf("File not found");*

*} else {*

*fscanf(fp, "%d", &sodinh);*

*for (int i = 0; i < sodinh; i++) {*

*for (int j = 0; j < sodinh; j++) {*

*fscanf(fp, "%d", &mtk[i][j]);*

*}} }*

*isGraphLoaded = true;*

*fclose(fp);}*

Đầu tiên, hàm sử dụng fopen(filePath, "r") để mở tệp tin với chế độ đọc ("r"). Nếu tệp không tồn tại hoặc không thể mở được, hàm sẽ thông báo lỗi "File not found" và kết thúc quá trình đọc. Nếu tệp được mở thành công, hàm sẽ tiếp tục thực hiện các bước đọc dữ liệu từ tệp. Hàm fscanf(fp, "%d", &sodinh) đọc số lượng đỉnh (sodinh) từ tệp. Trong một vòng lặp hai chiều, hàm sẽ đọc từng phần tử của ma trận kề và lưu vào mảng mtk[i][j] bằng cách sử dụng fscanf(fp, "%d", &mtk[i][j]). Ma trận kề này mô tả các kết nối giữa các đỉnh trong đồ thị, với giá trị 1 biểu thị có cạnh nối giữa hai đỉnh và giá trị 0 biểu thị không có cạnh nối. Sau khi hoàn tất việc đọc dữ liệu, hàm sẽ đặt biến isGraphLoaded = true để đánh dấu rằng đồ thị đã được tải thành công từ tệp. Sau đó, tệp được đóng lại bằng cách gọi fclose(fp).Toàn bộ quá trình này giúp tải dữ liệu về đồ thị từ tệp tin và chuẩn bị cho các bước xử lý tiếp theo trong chương trình.

**Nhập dữ liệu từ bàn phím:**

Người dùng nhập vào hai giá trị từ bàn phím: đỉnh bắt đầu (start) và đỉnh kết thúc (end) cho thuật toán DFS.

* printf("Nhap dinh bat dau: ");: Hiển thị thông báo yêu cầu người dùng nhập vào đỉnh bắt đầu.
* scanf("%d", &start);:Đọc giá trị số nguyên nhập vào từ bàn phím và lưu vào biến start, đại diện cho đỉnh bắt đầu.
* printf("Nhap dinh ket thuc: ");: Hiển thị thông báo yêu cầu người dùng nhập vào đỉnh kết thúc.
* scanf("%d", &end);:Đọc giá trị số nguyên nhập vào từ bàn phím và lưu vào biến end, đại diện cho đỉnh kết thúc.

Mục đích của đoạn mã là cho phép người dùng xác định đỉnh bắt đầu và kết thúc trong đồ thị để thuật toán DFS có thể thực hiện tìm kiếm từ đỉnh bắt đầu đến đỉnh kết thúc.

### 2.4.2 Thuật toán và ứng dụng đồ họa để thể hiện thuật toán

#### 2.4.2.1 Thuật toán tìm kiếm chiều sâu DFS

Thuật toán **Tìm kiếm theo chiều sâu (DFS)** là một phương pháp dùng để tìm kiếm hoặc duyệt qua các đỉnh trong đồ thị. Quá trình này đi từ đỉnh bắt đầu, đi sâu vào từng nhánh cho đến khi không thể đi tiếp được (tức là gặp đỉnh không có đỉnh con chưa được thăm), sau đó quay lại và tiếp tục tìm kiếm các nhánh khác.

Quy trình mô tả thuật toán DFS:

**Khởi tạo :**

Khi bắt đầu thuật toán, các mảng và biến cần được khởi tạo:

* Mảng CLOSE[]: Mảng này lưu trạng thái các đỉnh trong đồ thị. Khi một đỉnh được duyệt, giá trị của nó trong mảng này sẽ được thay đổi từ chỉ số của đỉnh thành -1 (đánh dấu đã thăm).
* Mảng OPEN[]: Đây là danh sách các đỉnh đang chờ được xét. Đỉnh đầu tiên trong OPEN[] là đỉnh bắt đầu và các đỉnh khác được thêm vào khi có liên kết từ đỉnh hiện tại.
* Mảng CHA[]: Đây là mảng lưu chỉ số của đỉnh cha tương ứng với mỗi đỉnh trong quá trình duyệt
* Mảng OPENED[]: Mảng này được sử dụng để đánh dấu các đỉnh đã được đưa vào danh sách OPEN. Nếu một đỉnh đã được đưa vào OPEN, nó sẽ không được thêm vào lần nữa.
* Biến path[]: Mảng này lưu trữ đường đi từ đỉnh bắt đầu đến đỉnh đích.

Các đỉnh trong đồ thị sẽ được khởi tạo như sau:

- Đặt tất cả các đỉnh vào OPEN để chuẩn bị duyệt qua.

- Đưa đỉnh bắt đầu vào OPEN, đồng thời đánh dấu nó là chưa được xét trong CLOSE.

*for (int i = 0; i < n; i++) {*

*CLOSE[i] = i; // Chua xet*

*OPENED[i] = i;*

*CHA[i] = -1; } // Khong co cha*

OPEN[dem++] = start;: Tại bước này, đỉnh bắt đầu (start) được đưa vào danh sách OPEN, và đồng thời biến dem được tăng lên, cho phép thêm các đỉnh tiếp theo vào danh sách OPEN trong các bước tiếp theo của thuật toán.

*OPEN[dem++] = start; // Them dinh bat dau vao OPEN*

Để minh họa rõ hơn về cách thuật toán DFS hoạt động, cách lưu trữ dữ liệu và truy vết đường đi, đây một ví dụ cụ thể.

*Giả sử*: Chúng ta có một đồ thị với 5 đỉnh (0, 1, 2, 3, 4) và các cạnh nối như sau:

* Đỉnh 0 nối với đỉnh 1 và 4.
* Đỉnh 1 nối với đỉnh 0 và 2.
* Đỉnh 2 nối với đỉnh 1 và 3.
* Đỉnh 3 nối với đỉnh 2 và 4.
* Đỉnh 4 nối với đỉnh 0 và 3.

Giả sử chúng ta sẽ tìm kiếm đường đi từ đỉnh 0 đến đỉnh 3 bằng thuật toán DFS.

Sau khi khởi tạo, các mảng sẽ có giá trị ban đầu như sau:

* OPEN = [0] (Chỉ có đỉnh 0 là chưa xét).
* CLOSE = [0, 1, 2, 3, 4] (Các đỉnh đều chưa xét).
* OPENED = [0, 1, 2, 3, 4] (Tất cả các đỉnh đều chưa được đưa vào OPEN).
* CHA = [-1, -1, -1, -1, -1] (Không có đỉnh cha nào).

**Duyệt qua các đỉnh**

Khi thuật toán bắt đầu duyệt qua các đỉnh trong danh sách OPEN, nó sẽ lấy ra đỉnh cuối cùng (vì là DFS, ta sẽ xét các đỉnh theo thứ tự chiều sâu). Sau đó, các bước tiếp theo sẽ diễn ra như sau:

**a. Kiểm tra đỉnh đích**

Mỗi khi xét một đỉnh, thuật toán kiểm tra xem đỉnh đó có phải là đỉnh đích (goal) hay không. Nếu đúng, thuật toán sẽ kết thúc và thực hiện truy vết đường đi từ đỉnh đích về đỉnh bắt đầu.

*if (current == goal) {*

*printf("\nTim thay duong di tu %d den %d.\n", start, goal);*

*// Truy vet duong di*

*int temp = goal;*

*while (temp != -1) {*

*path[len++] = temp;*

*temp = CHA[temp]; }*

Để truy vết đường đi, thuật toán sẽ sử dụng mảng CHA[] để lần lượt tìm đỉnh cha của mỗi đỉnh, bắt đầu từ đỉnh đích và quay lại đỉnh bắt đầu. Khi truy vết, ta sẽ thu được một chuỗi các đỉnh từ đích về nguồn.

Khi DFS tìm thấy đỉnh đích (goal), ta cần xác định đường đi từ đỉnh bắt đầu (start) đến đỉnh đích. Thuật toán truy vết ngược từ đích về nguồn bằng cách sử dụng mảng CHA[], trong đó, CHA[i] lưu thông tin về cha (parent) của đỉnh i. Nghĩa là:

- Nếu ta đang ở đỉnh i, thì đỉnh cha của i (đỉnh mà từ đó ta đi đến i) là CHA[i].

- Bằng cách lặp qua mảng CHA[] từ goal và lần lượt lấy cha của mỗi đỉnh, ta sẽ truy ngược được đường đi từ đích về nguồn.

Các truy vết:

Khi DFS tìm thấy đích (current == goal), chương trình sẽ thông báo đã tìm thấy đường đi từ đỉnh bắt đầu (start) đến đỉnh đích (goal). Biến len được khởi tạo bằng 0 để theo dõi chiều dài của đường đi. Quá trình truy vết bắt đầu từ đỉnh đích (goal), và ta sẽ lần lượt tìm đỉnh cha của mỗi đỉnh cho đến khi gặp đỉnh bắt đầu.

Sử dụng một vòng lặp for để truy vết ngược từ đích về nguồn. Điều kiện dừng của vòng lặp là khi ta đã đến đỉnh bắt đầu (CHA[i] == -1).

Trong mỗi lần lặp, ta làm như sau:

* Thêm đỉnh hiện tại (i) vào mảng path[] để lưu lại đường đi.
* Tăng biến len để đếm số lượng đỉnh trên đường đi.
* Cập nhật i bằng CHA[i], tức là chuyển đến đỉnh cha của đỉnh hiện tại để tiếp tục truy vết.

Quá trình lặp sẽ tiếp tục cho đến khi CHA[i] == -1, tức là đã tìm được đỉnh bắt đầu, lúc này vòng lặp dừng lại. Mảng path[] sẽ chứa các đỉnh từ đích về nguồn theo thứ tự ngược lại. Để có đường đi từ nguồn đến đích, ta có thể đảo ngược mảng path[].

Ví dụ:

Đỉnh bắt đầu: start = 0, nếu đỉnh 4 là đỉnh đích và ta đi từ đỉnh 4 về đỉnh bắt đầu:

Mảng CHA[] sau khi DFS chạy: CHA = [-1, 0, 1, 1, 3]

* CHA[1] = 0: Đỉnh 1 có cha là đỉnh 0.
* CHA[3] = 1: Đỉnh 3 có cha là đỉnh 1.
* CHA[4] = 3: Đỉnh 4 có cha là đỉnh 3.
* CHA[0] = -1: Đỉnh 0 là đỉnh bắt đầu, không có cha.

*Quá trình truy vết*:

Bước 1:

* + - * i = 4
      * Thêm 4 vào path[].
      * Cập nhật i = CHA[4] = 3.

Bước 2:

* + - * i = 3
      * Thêm 3 vào path[].
      * Cập nhật i = CHA[3] = 1.

Bước 3:

* + - * i = 1
      * Thêm 1 vào path[].
      * Cập nhật i = CHA[1] = 0.

Bước 4:

* + - * i = 0
      * Thêm 0 vào path[].
      * Cập nhật i = CHA[0] = -1 (kết thúc).

Sau khi vòng lặp kết thúc, mảng path[] chứa các đỉnh từ đích về nguồn, tức là: [4, 3, 1, 0].

Đây là đường đi từ goal về start. Để in ra đường đi theo chiều thuận, chỉ cần đảo ngược:

Đường đi: 0 -> 1 -> 3 -> 4.

**b. Thêm các đỉnh kề vào OPEN**

Nếu đỉnh đang xét không phải là đỉnh đích, thuật toán sẽ tiếp tục duyệt qua các đỉnh kề của đỉnh hiện tại. Đối với mỗi đỉnh kề, nếu đỉnh đó chưa được xét và có cạnh nối từ đỉnh hiện tại, đỉnh kề đó sẽ được thêm vào danh sách OPEN và mảng CHA[] sẽ được cập nhật để lưu trữ đỉnh cha của đỉnh kề đó.

*// Them cac dinh ke vao OPEN*

*for (int i = n - 1; i >= 0; i--) {*

*if (mtk[current][i] == 1 && CLOSE[i] != -1 && OPENED[i] != -1) {*

*OPEN[dem++] = i; // Them dinh vào OPEN*

*OPENED[i] = -1; // Ðanh dau dã tham*

*CHA[i] = current; // Luu cha cua dinh này*

*lastNeighbor = i; // luu de ve canh*

*}}*

Với mỗi đỉnh kề i của đỉnh hiện tại (current), thuật toán kiểm tra xem đỉnh i có hợp lệ để xét hay không. OPENED[i] == -1 nghĩa là đỉnh này đã được xét hoặc đang chờ xét, do đó ta không thêm lại vào danh sách OPEN. CLOSE[i] == -1 nghĩa là đỉnh này đã bị duyệt rồi, nên ta không xét lại

- Nếu đỉnh i hợp lệ (là đỉnh kề, chưa được xét và chưa được thêm vào danh sách chờ) thì:

+ Thêm đỉnh i vào danh sách OPEN để duyệt trong các bước tiếp theo.

+ Đánh dấu đỉnh i là đã được thêm vào OPEN.

+ Ghi lại cha của đỉnh i để sau này có thể truy vết đường đi.

**Ví dụ**:

Khi xét đỉnh 0, đỉnh kề là 1 và 2.

* Đỉnh 1 được thêm vào OPEN và CHA[1] = 0.
* Đỉnh 2 được thêm vào OPEN và CHA[2] = 0.

Sau khi duyệt qua đỉnh 0, các mảng sẽ được cập nhật như sau:

* OPEN[] = [1, 2] (đỉnh 1 và 2 được thêm vào danh sách chờ)
* CLOSE[] = [-1, 1, 2, 3, 4] (đỉnh 0 đã được xét)
* CHA[] = [-1, 0, 0, -1, -1] (đỉnh 1 và 2 có cha là 0)

**c. Lặp lại bước này cho đến khi tìm thấy đỉnh đích**

Thuật toán sẽ tiếp tục duyệt qua các đỉnh trong danh sách OPEN, theo chiều sâu, và kiểm tra các đỉnh kề của chúng. Quá trình này tiếp diễn cho đến khi đỉnh đích được tìm thấy.

Sau mỗi vòng lặp, thuật toán quay lại bước đầu tiên:

* **Lấy đỉnh cuối trong OPEN**.
* Kiểm tra đỉnh này có phải đích (goal) không.
* Nếu không, duyệt các đỉnh kề của nó và thêm chúng vào OPEN.

Quá trình này lặp lại cho đến khi một trong hai điều kiện xảy ra:

1. **Tìm thấy đỉnh đích**: Truy vết đường đi bằng mảng CHA[].
2. **Danh sách OPEN rỗng**: Có nghĩa là không tồn tại đường đi từ start đến goal.

**Ví dụ**:

* Khi xét đỉnh 1, đỉnh kề của 1 là 0 và 3.
* Đỉnh 0 đã được xét rồi, nhưng đỉnh 3 chưa được xét và có cạnh nối với 1, do đó đỉnh 3 sẽ được thêm vào OPEN và CHA[3] = 1.

Sau khi duyệt qua đỉnh 1, các mảng sẽ được cập nhật như sau:

* OPEN[] = [2, 3]
* CLOSE[] = [-1, -1, 2, 3, 4]
* CHA[] = [-1, 0, 0, 1, -1]

Trong cách cài đặt của thuật toán DFS, danh sách OPEN[] được thực hiện như một **stack (ngăn xếp)**, tức là các đỉnh được thêm vào cuối danh sách và sẽ được lấy ra theo thứ tự "LIFO" (Last In, First Out - Vào sau ra trước). Điều này có nghĩa là đỉnh nào được thêm vào OPEN[] sau cùng sẽ là đỉnh đầu tiên được lấy ra để xét.

*// Bat dau duyet*

*while (dem > 0) {*

*int current = OPEN[--dem]; // Lay dinh cuoi cung trong OPEN*

*printf("\nDinh dang xet: %d", current);*

*CLOSE[current] = -1; // Dua dinh nay vao CLOSE, Danh dau dinh bat dau da duoc tham*

Câu lệnh --dem thực hiện việc giảm giá trị của dem trước khi sử dụng nó. Điều này có nghĩa là khi lấy một đỉnh từ danh sách OPEN[], ta sẽ lấy đỉnh cuối cùng (đỉnh vừa được thêm vào gần nhất), vì OPEN[dem] sẽ là phần tử cuối cùng trong danh sách khi dem có giá trị bằng số lượng đỉnh hiện có trong OPEN[].

Khi một đỉnh được lấy ra từ OPEN[] theo cách này, thuật toán đảm bảo rằng sẽ duyệt vào các đỉnh kề của đỉnh đó trước tiên. Vì vậy, việc sử dụng --dem và lấy phần tử cuối cùng của OPEN[] giúp thuật toán thực hiện việc duyệt theo chiều sâu, luôn ưu tiên xét đỉnh gần nhất đã được thêm vào (theo nguyên tắc LIFO).

**Ví dụ:**

Giả sử ta có một đồ thị với các đỉnh được đánh số từ 0 đến 4, và danh sách OPEN[] có trạng thái: OPEN[] = [0, 1, 2, 3] // Các đỉnh được thêm vào OPEN theo thứ tự

Lúc này, dem = 4, và đỉnh cuối cùng được thêm vào là đỉnh 3.

Khi câu lệnh int current = OPEN[--dem]; được thực hiện:

* dem giảm xuống 3, vì vậy current = OPEN[3] sẽ lấy ra đỉnh 3 (đỉnh cuối cùng trong danh sách).
* Đỉnh 3 được xét và các đỉnh kề của nó sẽ được thêm vào OPEN[] (nếu có), rồi thuật toán tiếp tục lấy đỉnh cuối cùng của OPEN[] để xét.

Quá trình này tiếp tục cho đến khi danh sách OPEN[] rỗng, nghĩa là không còn đỉnh nào để xét.

**Kết thúc và in ra kết quả**

Khi thuật toán tìm thấy đỉnh đích, nó sẽ in ra đường đi từ đỉnh bắt đầu đến đỉnh đích, đồng thời vẽ đường đi trên đồ thị (nếu có giao diện đồ họa).

*// In ra duong di*

*printf("\nDuong di: ");*

*for (int i = len - 1; i >= 0; i--) {*

*printf("%d ", path[i]);}*

Nếu thuật toán không tìm thấy đỉnh đích (tức là danh sách OPEN đã hết mà không có đỉnh đích), nó sẽ thông báo rằng không tìm thấy đường đi.

*// Neu khong tim thay duong di*

*printf("\nKhong tim thay duong di tu %d den %d.\n", start, goal);*

**Ví du minh họa:**

Chúng ta có một đồ thị với 5 đỉnh (0, 1, 2, 3, 4) và các cạnh nối như sau:

mtk = {

{0, 1, 0, 0, 1},

{1, 0, 1, 0, 0},

{0, 1, 0, 1, 0},

{0, 0, 1, 0, 1},

{1, 0, 0, 1, 0}

}

Ma trận kề này biểu diễn rằng:

* Đỉnh 0 nối với đỉnh 1 và 4.
* Đỉnh 1 nối với đỉnh 0 và 2.
* Đỉnh 2 nối với đỉnh 1 và 3.
* Đỉnh 3 nối với đỉnh 2 và 4.
* Đỉnh 4 nối với đỉnh 0 và 3.

Giả sử chúng ta sẽ tìm kiếm đường đi từ đỉnh 0 đến đỉnh 3 bằng thuật toán DFS.

Khi khởi tạo, các mảng sẽ có giá trị ban đầu như sau:

* OPEN = [0] (Chỉ có đỉnh 0 là chưa xét).
* CLOSE = [0, 1, 2, 3, 4] (Các đỉnh đều chưa xét).
* OPENED = [0, 1, 2, 3, 4] (Tất cả các đỉnh đều chưa được đưa vào OPEN).
* CHA = [-1, -1, -1, -1, -1] (Không có đỉnh cha nào).

Bước 1: Xét đỉnh 0

- Ta lấy đỉnh 0 từ danh sách OPEN và đưa vào danh sách CLOSE. Sau đó, kiểm tra các đỉnh kề của 0.

- Đỉnh 0 có 2 đỉnh kề: 1 và 4.

+ Đỉnh 1 chưa được thăm (CLOSE[1] != -1) và chưa được đưa vào OPEN (OPENED[1] != -1), do đó ta thêm đỉnh 1 vào OPEN và cập nhật CHA[1] = 0.

+ Đỉnh 4 chưa được thăm (CLOSE[4] != -1) và chưa được đưa vào OPEN (OPENED[4] != -1), ta thêm đỉnh 4 vào OPEN và cập nhật CHA[4] = 0.

Sau bước này, các mảng sẽ có giá trị như sau:

* OPEN = [4, 1] (Đỉnh 4 và 1 đã được đưa vào OPEN).
* CLOSE = [-1, 1, 2, 3, 4] (Đỉnh 0 đã được thăm).
* OPENED = [0, -1, -1, -1, -1] (Đỉnh 0 đã được xét).
* CHA = [-1, 0, -1, -1, 0] (Đỉnh 1 và 4 có cha là 0).

Bước 2: Xét đỉnh 4

Tiếp theo, ta xét đỉnh 4 từ danh sách OPEN.

- Đỉnh 4 có 2 đỉnh kề: 0 và 3.

+ Đỉnh 0 đã được thăm rồi (CLOSE[0] = -1), không làm gì.

+ Đỉnh 3 chưa được thăm (CLOSE[3] != -1) và chưa được đưa vào OPEN (OPENED[3] != -1), do đó ta thêm đỉnh 3 vào OPEN và cập nhật CHA[3] = 4.

Sau bước này, các mảng sẽ có giá trị như sau:

* OPEN = [1, 3] (Đỉnh 1 và 3 đã được đưa vào OPEN).
* CLOSE = [-1, 1, 2, -1, -1] (Đỉnh 4 đã được xét).
* OPENED = [0, -1, -1, -1, -1] (Đỉnh 4 đã được xét).
* CHA = [-1, 0, -1, 4, 0] (Đỉnh 3 có cha là 4).

Bước 3: Xét đỉnh 3 (Đỉnh đích)

Bây giờ, ta xét đỉnh 3 từ danh sách OPEN. Đỉnh 3 là đỉnh đích (goal = 3), do đó thuật toán kết thúc.

- Truy vết đường đi từ đỉnh 3 đến đỉnh 0 bằng cách sử dụng mảng CHA với CHA khác -1:

+ Đỉnh 3 có cha là 4 (CHA[3] = 4).

+ Đỉnh 4 có cha là 0 (CHA[4] = 0).

+ Đỉnh 0 là đỉnh bắt đầu, không có cha (CHA[0] = -1).

Đường đi từ 0 đến 3 là: 0 -> 4 -> 3.

Sau khi truy vết, thuật toán in ra đường đi: Đường đi từ 0 đến 3: 0 4 3

A screenshot of a computer

Description automatically generated

Hình 2.17 Kết quả thuật toán DFS khi xử lí

#### 2.4.2.2 Thuật toán DFS với đồ họa

Thuật toán Tìm kiếm theo chiều sâu (DFS) được minh họa rõ ràng qua đồ họa, cho phép người dùng trực quan hóa quá trình duyệt đồ thị. Trong quá trình thực thi thuật toán, các bước tìm kiếm, từ việc chọn đỉnh để duyệt, kiểm tra các đỉnh kề, đến việc truy vết lại đường đi, đều được thể hiện qua đồ họa. Chúng ta sẽ đi vào chi tiết từng bước của thuật toán và cách thức mà đồ họa hoạt động cùng với thuật toán DFS.

**Khởi tạo Ban đầu và Vẽ Đỉnh**

Khi bắt đầu thuật toán, các mảng cần thiết được khởi tạo để theo dõi các đỉnh đã được duyệt và các đỉnh đang chờ duyệt:

* CLOSE[]: Theo dõi các đỉnh đã được duyệt.
* OPEN[]: Lưu trữ các đỉnh đang chờ duyệt.
* CHA[]: Mảng lưu trữ cha của mỗi đỉnh, giúp truy vết lại đường đi sau khi tìm thấy đỉnh đích
* visitedEdges[MAX][MAX] = {0}: Mảng lưu cách đỉnh đã duyệt để vẽ màu khi tìm thấy đường đi.

Sau khi khởi tạo, đỉnh bắt đầu (start) được thêm vào mảng OPEN[], sau đó hàm drawVertex(current, RED) sẽ vẽ đỉnh bắt đầu lên màn hình với màu **đỏ**. Màu đỏ biểu thị rằng đỉnh này đang được duyệt.

*while (dem > 0) {*

*int current = OPEN[--dem]; // Lay dinh cuoi cung trong OPEN*

*printf("\nDinh dang xet: %d", current);*

*CLOSE[current] = -1; // Dua dinh nay vao CLOSE, Danh dau dinh bat dau da duoc tham*

*// Ve dinh hien tai*

*drawVertex(current, RED);*

*delay(500);*

Hàm drawVertex(current, RED) sẽ vẽ đỉnh có chỉ số current bằng màu đỏ, biểu thị đỉnh này đang được thuật toán DFS xét đến.

A network with numbers and lines

Description automatically generated

Hình 2.18 Hình ảnh vẽ đỉnh và cạnh khi xử lý thuật toán

**Truy Vết Đường Đi**

Khi thuật toán tìm thấy đỉnh đích (goal), nó sẽ bắt đầu truy vết lại đường đi từ đỉnh đích về đỉnh bắt đầu thông qua mảng CHA[], mảng lưu trữ cha của các đỉnh. Quá trình truy vết này là để tìm đường đi ngắn nhất từ đỉnh bắt đầu đến đỉnh đích, và đồng thời thể hiện đường đi này trên đồ họa.

*// ve lai do thi*

*drawframe();*

*drawgraph();*

*// ve lai cac canh da xet*

*for (int i = 0; i < n; i++) {*

*for (int j = 0; j < n; j++) {*

*if (visitedEdges[i][j] && CLOSE[i] == -1 && CLOSE[j] == -1){*

*drawVertex(j, RED);*

*if (!isDirectedGraph()) {*

*drawEdge(i, j, RED);*

*} else {*

*drawEdgeCH(i, j, RED);*

*if (mtk[j][i] == 1) {*

*drawEdgeCH(j, i, RED);*

*} } }}}*

*// ve lai duong di dung*

*for (int i = len - 1; i > 0; i--) {*

*if (!isDirectedGraph()) {*

*drawEdge(path[i], path[i - 1], BLUE);*

*} else {*

*drawEdgeCH(path[i], path[i - 1], BLUE);*

*if (mtk[path[i - 1]][path[i]] == 1) {*

*drawEdgeCH(path[i - 1], path[i], BLUE);*

*}}*

*drawVertex(path[i], BLUE);}*

*drawVertex(path[0], BLUE); // ve dinh dau tien*

Hàm drawframe() được gọi để vẽ lại khung của đồ thị. Khung này có thể bao gồm các yếu tố như đường biên, các khung chứa các đỉnh và cạnh, hoặc các yếu tố trang trí khác giúp làm rõ cấu trúc của đồ thị. Sau khi khung được vẽ, hàm drawgraph() được gọi để vẽ lại tất cả các đỉnh và cạnh của đồ thị.

Sau khi vẽ lại đồ thị, đoạn mã tiếp theo tập trung vào việc vẽ lại các cạnh đã xét trong quá trình tìm kiếm. Vòng lặp for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { ... } } duyệt qua tất cả các cặp đỉnh trong ma trận kề của đồ thị. Đối với mỗi cặp đỉnh i và j, đoạn mã kiểm tra điều kiện visitedEdges[i][j] để xác định xem cạnh giữa hai đỉnh này đã được xét trong quá trình tìm kiếm hay chưa. Nếu cạnh đã được xét và cả hai đỉnh i và j chưa được đóng lại (tức là vẫn đang trong quá trình tìm kiếm), các đỉnh và cạnh này sẽ được vẽ lại.Đỉnh j được vẽ lại với màu đỏ, thể hiện rằng đỉnh này đã được xét trong quá trình tìm kiếm.

Nếu đồ thị không có hướng (trong trường hợp này được kiểm tra với !isDirectedGraph ()), hàm drawEdge (i, j, RED) vẽ lại cạnh giữa đỉnh i và đỉnh j bằng màu đỏ. Nếu đồ thị có hướng, hàm drawEdgeCH(i, j, RED) vẽ lại cạnh có hướng từ i đến j. Nếu cạnh ngược lại tồn tại (tức là mtk[j][i] == 1), cạnh ngược lại này cũng được vẽ lại bằng màu đỏ.

Sau khi vẽ lại các cạnh đã xét, đoạn mã tiếp tục vẽ lại đường đi đúng từ đỉnh bắt đầu đến đỉnh đích, nếu thuật toán đã tìm thấy đường đi. Đoạn mã sử dụng vòng lặp for (int i = len - 1; i > 0; i--) để duyệt qua mảng path[], chứa các đỉnh của đường đi từ đích về nguồn. Với mỗi đỉnh trong đường đi, một cạnh giữa các đỉnh liên tiếp trong mảng path[] được vẽ lại với màu xanh dương, thể hiện rằng đó là một phần của đường đi đúng.

* drawEdge( path[i], path[i - 1], BLUE);: Các cạnh giữa các đỉnh trong đường đi được vẽ lại với màu xanh dương. Nếu đồ thị có hướng, hàm drawEdgeCH (path[i], path[i - 1], BLUE) vẽ lại cạnh có hướng từ path[i] đến path[i - 1]. Nếu có cạnh ngược lại (mtk[path[i - 1]][path[i]] == 1), cạnh ngược lại cũng được vẽ lại với màu xanh dương.
* drawVertex(path[i], BLUE);: Các đỉnh trong đường đi được vẽ lại với màu xanh dương để làm nổi bật các đỉnh này là một phần của đường đi đúng.
* drawVertex(path[0], BLUE);: Cuối cùng, đỉnh đầu tiên trong đường đi (đỉnh bắt đầu) được vẽ lại với màu xanh dương, hoàn thành quá trình vẽ đường đi.

Sau khi tất cả các đỉnh và cạnh trong đường đi được vẽ xong, người dùng có thể dễ dàng nhìn thấy lộ trình mà thuật toán DFS đã tìm được.

A network with numbers and lines

Description automatically generated

Hình 2.19 Hình ảnh vẽ lại đồ thị khi tìm thấy đường đi

**Hiển thị đường đi:**

*// Hien thi duong di len man hinh do hoa*

*char buffer[MAX];*

*buffer[0] = '\0';*

*for (int i = len - 1; i >= 0; i--) {*

*char temp[10];*

*sprintf(temp, "%d ", path[i]);*

*strcat(buffer, temp);}*

*settextstyle(1, 0, 2);*

*setcolor(WHITE);*

*outtextxy(20, 490, (char\*)"DFS:");*

*outtextxy(100, 490, buffer);*

Tạo chuỗi đường đi: Lấy các đỉnh từ mảng path (đã tìm được) và ghép lại thành chuỗi, mỗi đỉnh cách nhau một khoảng trắng. Sử dụng settextstyle để định dạng kiểu chữ và setcolor để đặt màu trắng.

Hiển thị lên màn hình: Dòng chữ "DFS:" được vẽ tại tọa độ (20, 490) và chuỗi đường đi từ nguồn đến đích (lưu trong buffer) được hiển thị bên cạnh tại (100, 490).

A black background with white text

Description automatically generated

Hình 2.20 Hình ảnh hiển thị đường đi

**Vẽ cạnh khi duyệt qua đỉnh kề:**

Khi thuật toán duyệt qua các đỉnh kề của đỉnh hiện tại, một cạnh màu đỏ sẽ được vẽ nối từ đỉnh hiện tại đến đỉnh kề đang xét. Điều này giúp minh họa trực quan quá trình tìm kiếm.

*if(!isDirectedGraph()){*

*// Them các dinh ke vào OPEN*

*for (int i = n - 1; i >= 0; i--) {*

*if (mtk[current][i] == 1 && CLOSE[i] != -1 && OPENED[i] != -1) {*

*OPEN[dem++] = i; // Thêm dinh vào OPEN*

*OPENED[i] = -1; // Ðánh dau dã tham*

*CHA[i] = current; // Luu cha cua dinh nay*

*drawEdge(current, i, RED);*

*visitedEdges[current][i] = 1;*

*delay(500);*

*}}}*

Khi đồ thị không có hướng, thuật toán duyệt qua tất cả các đỉnh của đồ thị từ n-1 đến 0. Với mỗi đỉnh current, thuật toán kiểm tra tất cả các đỉnh i có cạnh kề với đỉnh current (thông qua ma trận kề mtk[current][i] == 1). Nếu đỉnh i chưa bị đóng (CLOSE[i] != -1) và chưa được mở (OPENED[i] != -1), nghĩa là đỉnh này vẫn còn trong trạng thái chưa được duyệt Sử dụng hàm drawEdge(current, i, RED), cạnh giữa hai đỉnh này sẽ được vẽ với màu đỏ, giúp người dùng dễ dàng nhận diện các cạnh mà thuật toán đang xét.

*for (int i = n - 1; i >= 0; i--) {*

*if ((mtk[current][i] == 1) && CLOSE[i] != -1 && OPENED[i] != -1) {*

*OPEN[dem++] = i; // Them dinh vao OPEN*

*OPENED[i] = -1; // Ðanh dau da tham*

*CHA[i] = current; // Luu cha cua dinh nay drawEdgeCH(current, i, RED);*

*if (mtk[i][current] == 1) {*

*drawEdgeCH(i, current, RED); }*

*visitedEdges[current][i] = 1;*

*delay(500); }}*

Đối với đồ thị có hướng, quy trình tương tự được thực hiện, nhưng với một sự khác biệt quan trọng. Trong đồ thị có hướng, cạnh giữa hai đỉnh current và i chỉ có hướng từ current đến i, vì vậy thuật toán vẽ cạnh có hướng. Hàm drawEdgeCH(current, i, RED) sẽ vẽ cạnh có hướng này với màu đỏ, giúp người dùng nhận diện các cạnh đang được xét. Nếu có cạnh ngược lại từ i đến current (tức là mtk[i][current] == 1), thuật toán sẽ vẽ thêm cạnh ngược lại bằng cách gọi hàm drawEdgeCH(i, current, RED). Điều này giúp người dùng dễ dàng nhận diện các cạnh ngược trong đồ thị có hướng.

Kết quả: Một đường nối màu đỏ sẽ xuất hiện giữa hai đỉnh, minh họa trực quan mối liên kết giữa các đỉnh kề được xét.

**Thông Báo Không Tìm Thấy Đường Đi**

Nếu thuật toán không tìm thấy đường đi từ đỉnh bắt đầu đến đỉnh đích, chương trình sẽ hiển thị thông báo trên màn hình đồ họa, giúp người dùng biết rằng không có đường đi giữa hai đỉnh này trong đồ thị.

*// Hien thi thong bao len man hinh do hoa*

*settextstyle(1, 0, 2);*

*setcolor(WHITE);*

*outtextxy(20, 490, (char\*)"DFS: Khong tim thay duong di");*

*printf("\n");*

Hàm outtextxy(20, 490, (char\*)"DFS: Khong tim thay duong di") sẽ hiển thị một thông báo chữ màu **trắng** trên màn hình đồ họa. Đây là một phần quan trọng của quá trình giao tiếp với người dùng, giúp họ hiểu rõ kết quả của thuật toán.

A black background with white text

Description automatically generated

Hình 2.21 Hiển thị thông báo không tìm thấy đường đi

# CHƯƠNG 3: KẾT QUẢ THỰC HIỆN

A screenshot of a computer

Description automatically generated

Hình 3.1 Giao diện khi nhấn nút load file

A screenshot of a computer

Description automatically generated

Hình 3.2 Giao diện sau khi chọn file và tải đồ thị

A screen shot of a computer

Description automatically generated

Hình 3.3 Giao diện khi bắt đầu nhấn button Start

A screenshot of a computer

Description automatically generated

Hình 3.4 Giao diện khi click vào “Enter”

A screenshot of a computer

Description automatically generated

Hình 3.5 Giao diện khi nhập các đỉnh từ bàn phím và xử lý DFS

A screenshot of a computer

Description automatically generated

Hình 3.6 Giao diện khi chọn phương thức nhập là Click vào đỉnh

A screenshot of a computer game

Description automatically generated

Hình 3.7 Giao diện khi click vào đỉnh bắt đầu

A screenshot of a video game

Description automatically generated

Hình 3.8 Giao diện khi click vào đỉnh kết thúc

A screenshot of a computer

Description automatically generated

Hình 3.9 Giao diện khi xử lý thuật toán bằng phương pháp nhập click chuột

A screenshot of a computer

Description automatically generated

Hình 3.10 Giao diện khi nhấn button Author

A screenshot of a computer

Description automatically generated

Hình 3.11 Giao diện khi nhấn button Reset

A screenshot of a computer

Description automatically generated

Hình 3.12 Giao diện khi nhấn button Delete

A screenshot of a computer

Description automatically generated

Hình 3.13 Giao diện khi nhấn button Exit

A screenshot of a computer

Description automatically generated

Hình 3.14 Giao diện màn hình console khi xử lý thuật toán DFS

A black screen with white text

Description automatically generated

Hình 3.15 Giao diện khi load file xong chuẩn bị tạo node

A screenshot of a computer

Description automatically generated

Hình 3.16 Giao diện khi tạo node

# 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 - DFS)" đã trình bày một ứng dụng cụ thể và sinh động của giải thuật DFS, kết hợp với thư viện đồ họa graphics.h để xây dựng giao diện đồ họa cho việc mô phỏng.

Mở đầu bằng phần giới thiệu về giải thuật tìm kiếm theo chiều sâu, một trong những thuật toán quan trọng trong lĩnh vực trí tuệ nhân tạo và lý thuyết đồ thị. Nó làm rõ nguyên lý hoạt động của DFS, các ưu điểm và nhược điểm của thuật toán, đồng thời nêu ra ứng dụng của DFS trong việc tìm kiếm đường đi trong đồ thị.

Báo cáo giới thiệu chi tiết về thư viện đồ họa graphics.h, cùng với cách sử dụng nó để xây dựng giao diện đồ họa. Các hàm và phương thức vẽ đối tượng đồ họa, như hình chữ nhật, nút bấm và các yếu tố đồ họa khác, được mô tả rõ ràng, giúp tạo ra một giao diện sinh động cho việc mô phỏng DFS. Báo cáo mô tả quá trình triển khai giải thuật DFS trên giao diện đồ họa. Các bước tạo và hiển thị đồ thị, thực thi thuật toán DFS, cùng với cách thức minh họa quá trình tìm kiếm trên màn hình, đều được trình bày chi tiết. Quá trình này giúp người dùng dễ dàng quan sát và hiểu rõ cách thức hoạt động của DFS trong thực tế.

Báo cáo khẳng định giá trị ứng dụng của giải thuật DFS kết hợp với thư viện graphics.h, không chỉ trong việc nghiên cứu lý thuyết mà còn trong việc tạo ra một công cụ trực quan giúp người dùng dễ dàng tương tác và theo dõi quá trình tìm kiếm theo chiều sâu. Tổng thể, báo cáo mang lại cái nhìn toàn diện về việc áp dụng DFS và xây dựng giao diện đồ họa để mô phỏng thuật toán này, đồng thời nhấn mạnh tính thực tiễn và hiệu quả của ứng dụng.

**TÀI LIỆU KHAM KHẢO**

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

[2]. Đoàn Vũ Thịnh (2024), *AI*, GitHub, Truy cập vào 18/12/2024, từ [*AI/code/DFS.cpp at main · thinhdoanvu/AI · GitHub*](https://github.com/thinhdoanvu/AI/blob/main/code/DFS.cpp)*.*

[3]. *Niên luận giải thuật prim - Luận văn, đồ án, đề tài tốt nghiệp*. (2024). Luanvan.net.vn. https://luanvan.net.vn/luan-van/nien-luan-giai-thuat-prim-27194/

‌[4]. *TÌM KIẾM TRÊN ĐỒ THỊ Toán rời rạc 2*. (n.d.). Retrieved December 8, 2024, from https://fit.ptithcm.edu.vn/wp-content/uploads/2018/03/TRRch03.pdf

[5]. (*Study\_Semester4\_2023-2024/ComputerGraphics/InstallGraphicsLib\_DevCpp at Master · Vinhveer/Study\_Semester4\_2023-2024*, 2023)

[6] Thái Thanh Hải. (2023, February 26). *CTDL> - Graph Algorithms - Depth First Search (DFS)*. Viblo. [*https://viblo.asia/p/data-structure-algorithm-graph-algorithms-depth-first-search-dfs-qPoL7zyXJvk*](https://viblo.asia/p/data-structure-algorithm-graph-algorithms-depth-first-search-dfs-qPoL7zyXJvk)

[7] thinhdoanvu. (2023). *AI/pdf/Chuong2\_Timkiem.pdf at main · thinhdoanvu/AI*. GitHub. [*https://github.com/thinhdoanvu/AI/blob/main/pdf/Chuong2\_Timkiem.pdf*](https://github.com/thinhdoanvu/AI/blob/main/pdf/Chuong2_Timkiem.pdf)

‌

‌