BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ**

**SỬ DỤNG THƯ VIỆN ĐỒ HỌA GRAPHIC.H MÔ PHỎNG GIẢI THUẬT TÌM ĐƯỜNG ĐI VỚI GIẢI THUẬT BREADTH-FIRST SEARCH**

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

**Sinh viên thực hiện : Trần Đức Ngạn**

**Mã số sinh viên : 64131460**

**Lớp : 64.CNTT-3**

Khánh Hòa - 2024

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO THỰC TẬP CƠ SỞ**

**SỬ DỤNG THƯ VIỆN ĐỒ HỌA GRAPHIC.H MÔ PHỎNG GIẢI THUẬT TÌM ĐƯỜNG ĐI VỚI GIẢI THUẬT BREADTH-FIRST SEARCH**

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

**Sinh viên thực hiện : Trần Đức Ngạn**

**Mã số sinh viên : 64131460**

**Lớp : 64.CNTT-3**

Khánh Hòa - 2024

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**Khoa: Công nghệ Thông tin**

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

**Tên đề tài:** Sử dụng thư viện đồ hoạ graphics.h mô phỏng giải thuật tìm đường đi với giải thuật BFS.

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

**Sinh viên được hướng dẫn:** Trần Đức Ngạn

**MSSV:** 64131460

**Khóa:** 64 **Ngành:** Công nghệ Thông tin

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 28/11/2024 | Sinh viên nhận đề tài hướng dẫn và định hướng giải quyết vấn đề. Sinh viên trình bày kế hoạch thực hiện. |  |
| 2 | 02/12/2024 | Sinh viên xác định và phát biểu bài toán, tìm hiểu, phân tích và ví dụ thuật toán tìm kiếm theo chiều rộng, trình bày sơ đồ thuật toán tìm kiếm theo chiều rộng. |  |
| 3 | 15/12/2024 | Sinh viên thực hiện phân tích chương trình, thiết kế giao diện người dung, cài đặt chức năng nhập dữ liệu và chức năng đồ họa. |  |
| 4 | 19/12/2024 | Sinh viên thực hiện cải tiến thuật toán, giao diện, hoàn thiện các tính năng và viết báo cáo. |  |
| 5 | 28/12/2024 | Sinh viên thực hiện demo và sửa những lỗi phát sinh trong quá trình thực hiện thuật toán. |  |
| 6 | 5/1/2024 | Sinh viên nộp bản thảo lần cuối sau khi đã chỉnh sửa các yêu cầu như đã đề ra. |  |

**MỤC LỤC**

[LỜI CẢM ƠN 4](#_Toc186360050)

[KẾT QUẢ THỰC HIỆN 5](#_Toc186360051)

[LỜI NÓI ĐẦU 6](#_Toc186360052)

[CHƯƠNG I. TỔNG QUAN 7](#_Toc186360053)

[1.1 Dev-C++ 7](#_Toc186360054)

[1.2 Thư viện Graphics.h 8](#_Toc186360055)

[1.3 Cơ sở lý thuyết 8](#_Toc186360056)

[1.3.1. Định nghĩa cơ bản về đồ thị 8](#_Toc186360057)

[1.3.2 Biểu diễn đồ thị trong lập trình 9](#_Toc186360058)

[1.3.2.1 Ma trận kề với đồ thị 9](#_Toc186360059)

[1.3.2.2 Ưu điểm, nhược điểm khi biểu diễn đồ thị bằng ma trận kề 9](#_Toc186360060)

[1.3.3 Thuật toán tìm kiếm theo chiều rộng 10](#_Toc186360061)

[1.3.3.1 Ý tưởng thuật toán 10](#_Toc186360062)

[1.3.3.2 Phân tích thuật toán 10](#_Toc186360063)

[1.3.3.3 Lưu đồ giải thuật BFS 13](#_Toc186360064)

[1.3.3.4 Ví dụ giải thuật BFS trên đồ thi 14](#_Toc186360065)

[CHƯƠNG II. PHƯƠNG PHÁP NGHIÊN CỨU 16](#_Toc186360066)

[2.1 Phân tích thuật toán 16](#_Toc186360067)

[2.1.1 Khai báo thư viện và định nghĩa hằng số 16](#_Toc186360068)

[2.1.2 Cấu trúc dữ liệu cho đồ thị 16](#_Toc186360069)

[2.1.3 Các hàm vẽ đồ thị 17](#_Toc186360070)

[2.1.4 Các hàm đặt trạng thái đồ thị 21](#_Toc186360071)

[2.1.5 Hàm đọc file 22](#_Toc186360072)

[2.1.6 Thuật toán BFS 23](#_Toc186360073)

[2.2 Thiết kế giao diện người dùng 26](#_Toc186360074)

[2.2.1. Xây dựng giao diện 26](#_Toc186360075)

[2.2.2 Xử lý sự kiện chuột 28](#_Toc186360076)

[CHƯƠNG III. KẾT QUẢ THỰC HIỆN 33](#_Toc186360077)

[3.1 Giao diện chính 33](#_Toc186360078)

[3.2 Nút “Infor” 34](#_Toc186360079)

[3.3 Nút “Load file” 35](#_Toc186360080)

[3.4 Nút “Start” 38](#_Toc186360081)

[3.5 Nút “Reset” 41](#_Toc186360082)

[3.6 Nút “Delete” 43](#_Toc186360083)

[3.7 Nút “Exit” 43](#_Toc186360084)

[CHƯƠNG IV. KẾT LUẬN 44](#_Toc186360085)

[4.1 Ưu điểm 44](#_Toc186360086)

[4.2 Nhược điểm 44](#_Toc186360087)

[TÀI LIỆU THAM KHẢO 45](#_Toc186360088)

**DANH MỤC HÌNH ẢNH**

[Hình 1. Đồ thị vô hướng 8](#_Toc186358288)

[Hình 2. Đồ thị có hướng 8](#_Toc186358289)

[Hình 4. Ma trận kề đồ thị có hướng 9](#_Toc186358290)

[Hình 3. Ma trận kề đồ thị vô hướng 9](#_Toc186358291)

[Hình 5. Lưu đồ giải thuật BFS 13](#_Toc186358292)

[Hình 6. Đồ thị ví dụ 14](#_Toc186358293)

[Hình 7. Tệp tin chứa dữ liệu đồ thị 22](#_Toc186358294)

[Hình 8. Giao diện chính 33](#_Toc186358295)

[Hình 9. Giao diện khi bấm nút “Infor” 34](#_Toc186358296)

[Hình 10. Giao diện khi bấm nút “Load file” 35](#_Toc186358297)

[Hình 11. Giao diện sau khi đã “Load file” 35](#_Toc186358298)

[Hình 12. Giao diện khi đang đặt đỉnh của đồ thị 36](#_Toc186358299)

[Hình 13. Giao diện đồ thị có hướng 37](#_Toc186358300)

[Hình 14. Giao diện đồ thị vô hướng 37](#_Toc186358301)

[Hình 15. Giao diện bấm nút “Start” khi chưa “Load file” 38](#_Toc186358302)

[Hình 16. Giao diện bấm nút “Start” khi đã “Load file” và đặt xong đồ thị 39](#_Toc186358303)

[Hình 17. Giao diện hiển thị khi thuật toán BFS đang tìm kiếm đường đi 39](#_Toc186358304)

[Hình 18. Giao diện hiển thị khi thuật toán BFS hoàn thành tìm kiếm đường đi 40](#_Toc186358305)

[Hình 19. Giao diện bấm nút “Reset” khi chưa “Load file” 41](#_Toc186358306)

[Hình 20. Giao diện bấm nút “Reset” khi đã “Load file” 42](#_Toc186358307)

[Hình 21. Giao diện bấm nút “Delete” 43](#_Toc186358308)

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 thực hiện đề tài này.

Em xin chân thành gửi lời cảm ơn đến Thầy Đoàn Vũ Thịnh đã hướng dẫn và giúp đỡ cho em trong học phần Thực tập cơ sở.

Do thời gian và kiến thức còn có hạn nên em sẽ không thể tránh khỏi những thiếu sót nhất định, kính mong sự góp ý của quý Thầy Cô.

KẾT QUẢ THỰC HIỆN

Đề tài được thực hiện với mục tiêu xây dựng một ứng dụng trực quan hóa thuật toán tìm kiếm theo chiều rộng (Breadth-First Search - BFS) sử dụng ngôn ngữ lập trình C++ và thư viện graphics.h. Các kết quả chính:

1. Link sản phẩm

* Lần 1:
  + Source code: <https://github.com/TranDucNgan05/TTCS_BFS/tree/afcec0c3ec7678531048cade9b357c069bc2224c>
  + Video demo: <https://www.youtube.com/watch?v=eJx_bDkzad0>
* Lần 2 (sau khi chỉnh sửa, bổ sung tính năng):
  + Source code: <https://github.com/TranDucNgan05/TTCS_BFS/tree/55f321363407048fe6bb3cf9319373a68f4cd83d>
  + Video demo: <https://www.youtube.com/watch?v=6hInNDh2lmE>
* Lần 3 (Final):
  + Source code: <https://github.com/thinhdoanvu/DOAN_TTCS_TTCN/tree/main/TranDucNganBFS>
  + Video demo: <https://www.youtube.com/watch?v=E6pwGw9xrPM>

2. Nội dung báo cáo

* Chương 1: Tổng quan về công cụ Dev-C++, thư viện graphics.h, các khái niệm lý thuyết liên quan đến đồ thị và thuật toán BFS.
* Chương 2: Quy trình nghiên cứu và phát triển ứng dụng, tập trung vào phân tích thuật toán, cài đặt cấu trúc dữ liệu đồ thị và xây dựng giao diện đồ họa.
* Chương 3: Các kết quả đạt được, bao gồm giao diện chính của ứng dụng và các chức năng chính.
* Chương 4: Kết luận tổng quan về kết quả.

LỜI NÓI ĐẦU

Lý thuyết đồ thị là một nhánh quan trọng trong toán học, với lịch sử phát triển lâu đời và ứng dụng đa dạng trong nhiều lĩnh vực khoa học và đời sống hiện đại, từ tin học, viễn thông, đến kinh tế và quản lý. Một trong những nội dung cốt lõi của lý thuyết đồ thị là nghiên cứu các thuật toán duyệt qua các đỉnh và cạnh của đồ thị để tìm kiếm thông tin hoặc giải quyết bài toán tối ưu. Các thuật toán này được thiết kế sao cho mỗi đỉnh chỉ được thăm đúng một lần, nhằm đảm bảo tính hệ thống và hiệu quả. Trong số đó, thuật toán tìm kiếm theo chiều rộng (Breadth-First Search - BFS) nổi bật với khả năng tìm kiếm ngắn nhất trong đồ thị không trọng số, góp phần quan trọng vào nhiều ứng dụng thực tiễn như định tuyến mạng, trí tuệ nhân tạo, và xử lý dữ liệu lớn.

Đề tài này được thực hiện với mục tiêu xây dựng một ứng dụng trực quan hóa thuật toán BFS bằng ngôn ngữ lập trình C++ và thư viện đồ họa graphics.h. Thông qua ứng dụng này, người dùng có thể dễ dàng hình dung cách thức hoạt động của thuật toán, từ việc chọn điểm bắt đầu chương trình sẽ duyệt qua từng lớp đỉnh, đến xác định kết quả.

Bên cạnh đó, nghiên cứu cũng trình bày chi tiết các khái niệm cơ bản của lý thuyết đồ thị và thuật toán BFS. Các kết quả đạt được từ ứng dụng bao gồm giao diện chính và khả năng trực quan hóa quá trình tìm kiếm trên đồ thị.

CHƯƠNG I. TỔNG QUAN

## 1.1 Dev-C++

Dev-C++ là một môi trường phát triển tích hợp (IDE) mã nguồn mở dành cho ngôn ngữ lập trình C và C++. Phần mềm này được phát triển bởi Bloodshed Software và hiện nay đã có nhiều phiên bản cải tiến do cộng đồng đóng góp. Với giao diện đơn giản, dễ sử dụng, Dev-C++ được đánh giá là công cụ học lập trình C/C++ hiệu quả, đặc biệt phù hợp với sinh viên và những người mới bắt đầu. Hướng dẫn cài đặt Dev-C++:

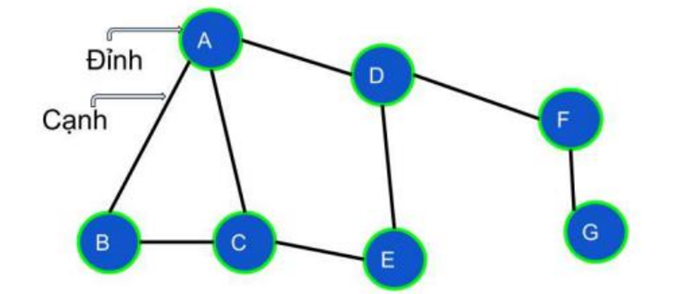
1. Tải xuống Dev-C++:
   * Truy cập trang web chính thức hoặc trang phân phối phần mềm:  
     <https://sourceforge.net/projects/orwelldevcpp/>
   * Nhấn Download để tải xuống tệp cài đặt.
2. Cài đặt phần mềm:
   * Mở tệp cài đặt (.exe) vừa tải về.
   * Chọn ngôn ngữ và nhấn Next.
   * Đọc và chấp nhận điều khoản sử dụng bằng cách nhấn I Agree.
   * Chọn đường dẫn cài đặt (có thể để mặc định) và nhấn Next.
   * Bấm Install để bắt đầu quá trình cài đặt.
3. Hoàn tất cài đặt:
   * Sau khi cài đặt hoàn tất, nhấn Finish.
   * Khởi động Dev-C++ và kiểm tra trình biên dịch: vào Tools > Compiler Options để đảm bảo GCC đã được thiết lập.
4. Cấu hình và sử dụng cơ bản:
   * Để viết chương trình đầu tiên, chọn File > New > Project hoặc File > New > Source File.
   * Lưu file với phần mở rộng .c hoặc .cpp, viết mã và nhấn F11 để biên dịch và chạy chương trình.

## 1.2 Thư viện Graphics.h

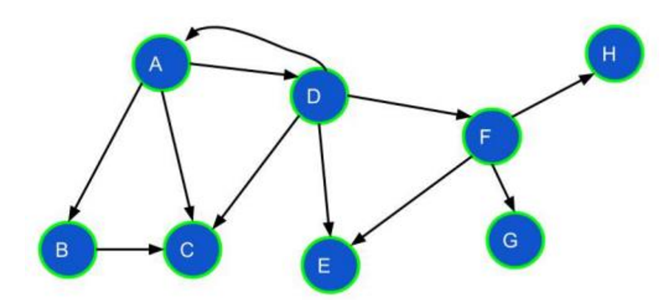
Thư viện graphics.h là một phần của thư viện đồ họa (Borland Graphics Interface - BGI) được sử dụng trong lập trình C/C++. Với graphics.h, cho phép vẽ các đối tượng hình học cơ bản như đường thẳng, đường tròn, hình elip, hình chữ nhật và đa giác. Ngoài ra, thư viện còn hỗ trợ hiển thị văn bản đồ họa với kiểu dáng và kích thước tùy chỉnh, cùng khả năng sử dụng các màu sắc cơ bản để tạo các hiệu ứng. Hướng dẫn cài đặt thư viện graphics.h: <https://github.com/thinhdoanvu/ComputerGraphics/blob/master/README.md>

## 1.3 Cơ sở lý thuyết

### 1.3.1. Định nghĩa cơ bản về đồ thị



Hình . Đồ thị vô hướng



Hình 2. Đồ thị có hướng

Những vòng tròn được gọi là các đỉnh (vertices) và những đường thẳng nối những vòng tròn được gọi là các cạnh (edges). Về cơ bản: Đồ thị là một tập hợp hữu hạn gồm các đỉnh và được nối với nhau bởi các cạnh. Một đồ thị G sẽ được kí hiệu: G = (V, E).

### 1.3.2 Biểu diễn đồ thị trong lập trình

Trong các chương trình, đồ thị thường được biểu diễn qua ba cách phổ biến: ma trận kề, danh sách kề và danh sách cạnh. Mỗi cách lưu trữ đều có ưu và nhược điểm riêng, phù hợp với từng bài toán cụ thể. Để đơn giản hóa việc nhập và xuất dữ liệu: sử dụng ma trận kề, vì dữ liệu đầu vào có kích thước nhỏ, thuận tiện cho việc xử lý mô phỏng đường đi.

#### 1.3.2.1 Ma trận kề với đồ thị

Với đồ thị ma trận kề của đồ thị có n đỉnh là ma trận vuông vỡ n x n có các phần tử là 0 hoặc 1.A[][] = {Aij = 1 nếu cạnh (i, j) là một cạnh của đồ thị, Aij = 0 nếu cạnh (i, j) không là một cạnh của đồ thị}.

**Ảnh có chứa biểu đồ, văn bản, hàng, ảnh chụp màn hình

Mô tả được tạo tự động**

Ảnh có chứa biểu đồ, hàng, ảnh chụp màn hình, Sơ đồ

Mô tả được tạo tự động

Hình 4. Ma trận kề đồ thị có hướng

Hình 3. Ma trận kề đồ thị vô hướng

#### 1.3.2.2 Ưu điểm, nhược điểm khi biểu diễn đồ thị bằng ma trận kề

**Ưu điểm:** Đơn giản, dễ cài đặt, dễ dàng kiểm tra 2 đỉnh có kề nhau hay không trong O(1) bằng cách kiểm tra giá trị của A[i, j].

**Nhược điểm:** Tốn bộ nhớ.

### 1.3.3 Thuật toán tìm kiếm theo chiều rộng

#### 1.3.3.1 Ý tưởng thuật toán

Tìm kiếm theo chiều rộng (BFS) là một thuật toán quan trọng trong lý thuyết đồ thị, được sử dụng để duyệt hoặc tìm kiếm trên cây hoặc đồ thị. Thuật toán bắt đầu từ một đỉnh gốc (hoặc một đỉnh bất kỳ được chọn làm gốc) và lần lượt duyệt qua tất cả các đỉnh lân cận. Sau đó, đối với từng đỉnh lân cận, thuật toán tiếp tục kiểm tra các đỉnh con chưa được thăm. Mỗi khi một đỉnh được kiểm tra, nó sẽ được đánh dấu là đã thăm. Quá trình này tiếp diễn cho đến khi tìm được đường đi ngắn nhất hoặc duyệt xong toàn bộ đồ thị.

Một điểm khác biệt cơ bản giữa BFS và tìm kiếm theo chiều sâu (DFS) nằm ở cách xử lý thứ tự duyệt các đỉnh. Trong DFS, các đỉnh được thăm muộn hơn sẽ sớm được duyệt xong nhờ việc sử dụng ngăn xếp (STACK), hoạt động theo cơ chế LIFO (Last In, First Out). Ngược lại, BFS thay thế ngăn xếp bằng hàng đợi (QUEUE), hoạt động theo cơ chế FIFO (First In First Out).

Với sự thay đổi này, trong BFS, các đỉnh được thăm sớm hơn sẽ được xử lý trước. Một đỉnh sẽ được xem là đã duyệt xong ngay sau khi tất cả các đỉnh kề chưa được thăm của nó đều được kiểm tra. Điều này giúp BFS trở thành một công cụ hiệu quả để tìm kiếm đường đi ngắn nhất trong đồ thị không trọng số.

#### 1.3.3.2 Phân tích thuật toán

Thuật toán sử dụng một cấu trúc dữ liệu hàng đợi (queue) để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng.

Bước 1: Khởi tạo

* Đặt tất cả các đỉnh ở trạng thái *chưa thăm*.
* Đánh dấu đỉnh xuất phát S là *đã thăm*.
* Thêm đỉnh S vào hàng đợi để bắt đầu duyệt theo thứ tự ưu tiên chiều rộng.

Bước 2: Lặp cho đến khi hàng đợi rỗng

* Lấy một đỉnh u ra khỏi hàng đợi và thông báo thăm u (bắt đầu duyệt đỉnh u).
* Xét tất cả các đỉnh v kề với u mà chưa được đánh dấu:
  + Đánh dấu v.
  + Ghi nhận vết đường đi từ u đến v.
  + Thêm v vào hàng đợi để chờ duyệt ở các bước tiếp theo.

Bước 3: Truy vết đường đi

* Sử dụng thông tin vết đã ghi nhận để tìm đường đi cụ thể.

BFS(S) {

// Bước 1: Khởi tạo

queue = ∅; // Tạo một hàng đợi rỗng

push(queue, S); // Đẩy đỉnh S vào hàng đợi

visited[S] = true; // Đánh dấu đỉnh S đã được thăm

parent[S] = -1; // S không có đỉnh cha (đỉnh gốc)

// Bước 2: Lặp khi mà hàng đợi rỗng

while (queue != ∅) {

u = queue.front(); // Lấy ra đỉnh ở đầu hàng đợi

queue.pop(); // Xóa đỉnh khỏi đầu hàng đợi

<Thăm đỉnh u>

// Duyệt các đỉnh kề với u mà chưa được thăm và đẩy vào hàng đợi

for (int v : ke[u]) {

if (!visited[v]) { // Nếu v chưa được thăm

push(queue, v);

visited[v] = true;

parent[v] = u; //Truy vết: Ghi nhận u là cha của v.

}

}

}

}

//Bước 3: Truy vết

truyVet(int T) {

int path[]; // Lưu đường đi

// Truy ngược từ T về S qua mảng parent[]

while (T != -1) {

push(path, T);

T = parent[T];

}

// Đảo ngược để có đường đi từ S đến T

reverse(path);

// In đường đi

for (int node : path) {

<In ra node>

}

}

#### 1.3.3.3 Lưu đồ giải thuật BFS

Ảnh có chứa văn bản, biểu đồ, Bản vẽ kỹ thuật, hàng

Mô tả được tạo tự động

Hình 5. Lưu đồ giải thuật BFS

#### 1.3.3.4 Ví dụ giải thuật BFS trên đồ thi

Ảnh có chứa vòng tròn, hàng, biểu đồ, hình vẽ

Mô tả được tạo tự động

Hình 6. Đồ thị ví dụ

| Hàng đợi | Đỉnh u  (lấy ra từ hàng đợi) | Hàng đợi  (sau khi lấy u ra) | Các đỉnh v kề u mà chưa thăm | Hàng đợi sau khi đẩy những đỉnh v vào |
| --- | --- | --- | --- | --- |
| (1) | 1 | ∅ | 2, 3 | (2, 3) |
| (2, 3) | 2 | (3) | 4 | (3, 4) |
| (3, 4) | 3 | (4) | 5 | (4, 5) |
| (4, 5) | 4 | (5) | 6 | (5, 6) |
| (5, 6) | 5 | (6) | ∅ | (6) |
| (6) | 6 | ∅ | ∅ | ∅ |

Khi quan sát thứ tự các phần tử được lấy ra khỏi hàng đợi:

* Đầu tiên là đỉnh 1 (đỉnh xuất phát).
* Tiếp theo là các đỉnh 2, 3 (các đỉnh kề trực tiếp với 1).
* Sau đó là các đỉnh 4, 5 (các đỉnh kề với 2 hoặc 3 ).
* Cuối cùng là đỉnh 6 (kề với 4 hoặc 5).

Điều này cho thấy, các đỉnh gần đỉnh xuất phát S hơn sẽ được duyệt trước, theo đúng nguyên tắc duyệt theo chiều rộng. Do đó, nếu kết hợp với việc lưu vết đường đi trong quá trình duyệt, ta có thể rút ra kết luận: Đường đi từ S đến một điểm khác trên đồ thị không trọng số trong thuật toán BFS luôn là đường đi ngắn nhất.

CHƯƠNG II. PHƯƠNG PHÁP NGHIÊN CỨU

## 2.1 Phân tích thuật toán

### 2.1.1 Khai báo thư viện và định nghĩa hằng số

#include <stdio.h>

#include <math.h>

#include <queue>

#include <graphics.h>

#define MAX\_SIZE 20

#define M\_PI 3.14159265358979323846

#define RADIUS 20

Đầu tiên, thư viện <stdio.h> được sử dụng để thao tác với input và output cơ bản. Thư viện <math.h> cung cấp các hàm toán học đtính toán góc và khoảng cách và tính toán khoảng cách giữa các điểm. Thư viện <queue> được sử dụng để triển khai hàng đợi (queue), phục vụ cho thuật toán BFS (Breadth-First Search). Thư viện <graphics.h> được sử dụng để vẽ các hình học và đồ họa trên màn hình, phục vụ cho việc tạo giao diện đồ họa người dùng. Các hằng số được khai báo bao gồm:

* MAX\_SIZE: Được định nghĩa là 20, là kích thước tối đa của đồ thị, chỉ ra số lượng đỉnh tối đa trong đồ thị.
* M\_PI: Đại diện cho giá trị của số Pi (π).
* RADIUS: Là bán kính của các đỉnh trong đồ thị khi vẽ hình tròn, giúp xác định kích thước của các điểm trong đồ thị.

### 2.1.2 Cấu trúc dữ liệu cho đồ thị

// Do thi

int numV; // So dinh

struct Coord { // Toa do

int x;

int y;

int label;

};

struct Coord vertices[MAX\_SIZE];

FILE \*fp;

char buffer[10];

int adjMatrix[MAX\_SIZE][MAX\_SIZE]; // Ma tran ke

Đoạn mã trên cung cấp các khai báo cơ bản cần thiết để xây dựng và xử lý đồ thị trong chương trình. Các cấu trúc dữ liệu được sử dụng, bao gồm mảng các đỉnh (vertices), ma trận kề (adjMatrix), biến số đỉnh (numV).

### 2.1.3 Các hàm vẽ đồ thị

// Ham ve hinh tron

void drawCircle(int x, int y, int label) {

setcolor(WHITE);

circle(x, y, RADIUS);

char buffer[10];

sprintf(buffer, "%d", label);

outtextxy(x - 10, y - 10, buffer);

}

Hàm drawCircle có chức năng vẽ một hình tròn và hiển thị một giá trị số nguyên (label) gần vị trí tâm của hình tròn đó. Đầu tiên, hàm sử dụng setcolor(WHITE) để thiết lập màu vẽ là trắng. Sau đó, hàm circle(x, y, RADIUS) được gọi để vẽ một hình tròn tại tọa độ (x, y) với bán kính RADIUS. Tiếp theo, giá trị label được chuyển thành chuỗi ký tự thông qua hàm sprintf và lưu vào mảng buffer. Cuối cùng, hàm outtextxy(x - 10, y - 10, buffer) được sử dụng để hiển thị giá trị label gần tâm của hình tròn, tại vị trí (x - 10, y - 10) để tránh bị che khuất.

// Ham ve duong thang

void drawLine(int x1, int y1, int x2, int y2) {

const int arrow\_size = 10; // Kich thuoc cua mui ten

// Tinh goc giua hai diem

double angle = atan2(y2 - y1, x2 - x1);

x1 = x1 + RADIUS \* cos(angle);

y1 = y1 + RADIUS \* sin(angle);

x2 = x2 - RADIUS \* cos(angle);

y2 = y2 - RADIUS \* sin(angle);

// Ve duong thang

line(x1, y1, x2, y2);

// Ve mui ten neu do thi co huong

if(!isAdjSymmetry) {

// Tinh toan cac diem cua mui ten

int x3 = x2 - arrow\_size \* cos(angle - M\_PI / 6);

int y3 = y2 - arrow\_size \* sin(angle - M\_PI / 6);

int x4 = x2 - arrow\_size \* cos(angle + M\_PI / 6);

int y4 = y2 - arrow\_size \* sin(angle + M\_PI / 6);

// Ve mui ten

line(x2, y2, x3, y3);

line(x2, y2, x4, y4);

}

}

Hàm drawLine có chức năng vẽ một đường thẳng giữa hai điểm (x1, y1) và (x2, y2), đồng thời vẽ mũi tên ở cuối đường thẳng nếu đồ thị có hướng. Đầu tiên, hàm tính toán góc giữa hai điểm bằng cách sử dụng hàm atan2, sau đó điều chỉnh tọa độ của các điểm sao cho chúng không bị che khuất bởi các hình tròn có bán kính RADIUS. Tiếp theo, hàm vẽ một đường thẳng từ điểm (x1, y1) đến điểm (x2, y2) bằng hàm line. Nếu đồ thị có hướng (được kiểm tra qua biến isAdjSymmetry), hàm tính toán các điểm để vẽ mũi tên tại vị trí (x2, y2). Mũi tên được vẽ bằng cách vẽ hai đoạn thẳng nhỏ từ điểm cuối của đường thẳng đến các điểm tính toán trước đó, tạo thành hình mũi tên.

// Ham ve canh

void drawEdges() {

for (int i = 0; i < numV; i++) {

for (int j = 0; j < numV; j++) {

// Vo huong (ma tran doi xung)

if (isAdjSymmetry) {

if (adjMatrix[i][j] == 1 && i < j) {

drawLine(vertices[i].x, vertices[i].y, vertices[j].x, vertices[j].y);

}

}

// Co huong

else {

if (adjMatrix[i][j] == 1) {

drawLine(vertices[i].x, vertices[i].y, vertices[j].x, vertices[j].y);

}

}

}

}

}

Hàm drawEdges có nhiệm vụ vẽ các cạnh của đồ thị dựa trên ma trận kề adjMatrix. Hàm duyệt qua tất cả các cặp đỉnh (i, j) trong đồ thị và kiểm tra điều kiện để vẽ cạnh giữa chúng. Nếu đồ thị là vô hướng (được xác định bởi biến isAdjSymmetry là true), cạnh chỉ được vẽ khi adjMatrix[i][j] == 1 và i < j để tránh vẽ lại các cạnh đã được vẽ, do ma trận kề đối xứng. Trong trường hợp đồ thị có hướng (khi isAdjSymmetry là false), cạnh được vẽ nếu adjMatrix[i][j] == 1 mà không phân biệt thứ tự của i và j. Để vẽ các cạnh, hàm sử dụng hàm drawLine, truyền vào tọa độ của các đỉnh tương ứng trong mảng vertices.

void drawgraph() {

int vertexCount = 0;

settextstyle(10, 0, 1);

setcolor(WHITE);

outtextxy(20, 390, "CLICK LEFT MOUSE BUTTON TO ADD A NODE");

// Mang luu tru cac dinh (chi ve cac dinh truoc)

while (vertexCount < numV && !kbhit()) {

if (ismouseclick(WM\_LBUTTONDOWN)) {

int x = mousex();

int y = mousey();

if (x >= 10 + RADIUS && x <= 550 - RADIUS && y >= 10 + RADIUS && y <= 370 - RADIUS) {

bool canPlace = true;

// Kiem tra khoang cach voi cac dinh da co

for (int i = 0; i < vertexCount; i++) {

double distance = sqrt(pow(vertices[i].x - x, 2) + pow(vertices[i].y - y, 2));

if (distance < 3 \* RADIUS) { // Khoang cach toi thieu de them dinh moi

canPlace = false;

break;

}

}

if (canPlace) {

// Gan vi tri va nhan cho dinh moi

vertices[vertexCount].x = x;

vertices[vertexCount].y = y;

vertices[vertexCount].label = vertexCount; // Nhan dinh bat dau tu 0

drawCircle(x, y, vertexCount); // Ve dinh

vertexCount++; // Tang so dinh

}

}

clearmouseclick(WM\_LBUTTONDOWN); // Xoa su kien chuot

}

delay(10);

}

setcolor(BLACK);

outtextxy(20, 390, "CLICK LEFT MOUSE BUTTON TO ADD A NODE");

setcolor(WHITE);

drawEdges(); // Ve cac canh sau khi ve tat ca cac dinh

}

Hàm drawgraph cho phép người dùng thêm các đỉnh vào đồ thị bằng cách nhấp chuột. Đầu tiên, hàm hiển thị thông báo yêu cầu người dùng nhấn chuột trái để thêm đỉnh. Sau đó, trong vòng lặp, hàm kiểm tra xem người dùng có nhấp chuột hay không bằng cách sử dụng ismouseclick(WM\_LBUTTONDOWN). Khi có sự kiện nhấp chuột, hàm lấy tọa độ chuột và kiểm tra xem tọa độ đó có nằm trong phạm vi cho phép (không vượt quá ranh giới của khung hiển thị và không trùng với các đỉnh đã tồn tại). Nếu vị trí hợp lệ hàm sẽ thêm đỉnh mới vào mảng vertices và vẽ nó lên màn hình bằng hàm drawCircle. Sau khi vẽ tất cả các đỉnh, hàm vẽ các cạnh giữa các đỉnh đã được thêm vào bằng cách gọi drawEdges.

// Ham ve lai do thi

void redrawgraph() {

for (int i = 0; i < numV; i++) {

drawCircle(vertices[i].x, vertices[i].y, vertices[i].label);

}

drawEdges();

}

Hàm redrawgraph có chức năng vẽ lại toàn bộ đồ thị. Mục đích của hàm này là để tái tạo lại đồ thị từ đầu.

### 2.1.4 Các hàm đặt trạng thái đồ thị

// Xoa do thi

void clearGraph() {

isBFSExecuted = false;

memset(visited, 0, sizeof(visited));

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

len = 0;

cleardevice();

}

Hàm clearGraph() có chức năng xóa đồ thị và thiết lập lại các trạng thái ban đầu.

// Xoa do thi va dat lai trang thai

void cleanGraph() {

memset(visited, 0, sizeof(visited));

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 xóa đồ thị, thiết lập lại các trạng thái về ban đầu và đặt biến isGraphLoaded về false, báo hiệu rằng đồ thị đã bị xóa và chưa được nạp.

### 2.1.5 Hàm đọc file

Ảnh có chứa ảnh chụp màn hình, thiết kế, mẫu, thuật in máy

Mô tả được tạo tự động

Hình 7. Tệp tin chứa dữ liệu đồ thị

Cấu trúc của file nhập vào bao gồm hai phần chính: số đỉnh và ma trận kề của đồ thị. Phần đầu tiên là số đỉnh của đồ thị, cho biết có bao nhiêu đỉnh trong đồ thị đó. Phần thứ hai là ma trận kề, thể hiện mối quan hệ giữa các đỉnh trong đồ thị. Ma trận kề là một ma trận vuông, với số dòng và số cột bằng số đỉnh của đồ thị.

// 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", &numV);

if (numV < 2 || numV > 15) {

printf("Invalid number of vertices. It must be between 2 and 15.\n");

return;

}

for (int i = 0; i < numV; i++) {

for (int j = 0; j < numV; j++) {

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

}

}

// Kiem tra doi xung

isAdjSymmetry = true;

for (int i = 0; i < numV; i++) {

for (int j = 0; j < numV; j++) {

if (adjMatrix[i][j] != adjMatrix[j][i]) {

isAdjSymmetry = false;

break;

}

}

}

}

isGraphLoaded = true; // Dat co gia tri la true khi do thi duoc load

fclose(fp);

}

Hàm openAndReadFile mở tệp tin chứa dữ liệu đồ thị và thực hiện các bước sau: Đầu tiên, tệp được mở bằng fopen. Nếu không thành công, thông báo lỗi sẽ được in ra. Ngược lại, chương trình đọc số lượng đỉnh numV từ tệp và kiểm tra tính hợp lệ của giá trị này (phải nằm trong khoảng từ 2 đến 15). Nếu không hợp lệ, chương trình dừng. Tiếp theo, ma trận kề được đọc vào mảng adjMatrix[i][j]. Sau khi đó, chương trình kiểm tra tính đối xứng của ma trận kề để xác định đồ thị có phải vô hướng không.

### 2.1.6 Thuật toán BFS

void bfs(int start, int end, int n) {

if (start < 0 || start >= n || end < 0 || end >= n) {

setcolor(WHITE);

settextstyle(10, 0, 1);

outtextxy(20, 390, "INVALID START OR END NODE");

return;

}

std::queue<int> q;

memset(visited, 0, sizeof(visited));

memset(parent, -1, sizeof(parent));

q.push(start);

visited[start] = 1;

// Hien thi duong tim kiem BFS

while (!q.empty()) {

int current = q.front();

q.pop();

setcolor(RED);

circle(vertices[current].x, vertices[current].y, 20);

delay(500);

if (current == end) {

break;

}

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

if (adjMatrix[current][i] == 1 && !visited[i]) {

q.push(i);

visited[i] = 1;

parent[i] = current;

// Ve duong noi tu diem tiep tuyen va mui ten mau do

setcolor(RED);

drawLine(vertices[current].x, vertices[current].y, vertices[i].x, vertices[i].y);

delay(400);

}

}

}

// Hien thi duong di

if (visited[end]) {

// Xay dung duong di

int path[MAX\_SIZE];

int len = 0;

for (int v = end; v != -1; v = parent[v]) {

path[len++] = v;

}

setcolor(YELLOW); // Duong di mau vang

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

// Ve lai duong mui ten

drawLine(vertices[path[i]].x, vertices[path[i]].y, vertices[path[i - 1]].x,

vertices[path[i - 1]].y);

delay(200);

}

for (int i = 0; i < len; i++) {

setcolor(YELLOW);

circle(vertices[path[i]].x, vertices[path[i]].y, 20);

delay(100);

}

settextstyle(10, 0, 1);

setcolor(WHITE);

outtextxy(20, 390, "PATH: ");

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

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

outtextxy(120 + (len - 1 - i) \* 30, 390, buffer);

}

}

else {

settextstyle(10, 0, 1);

setcolor(WHITE);

outtextxy(20, 390, "NO PATH FOUND");

}

}

Hàm bfs(int start, int end, int n) thực hiện thuật toán duyệt đồ thị theo chiều rộng (BFS) để tìm đường đi ngắn nhất giữa hai đỉnh start và end trong đồ thị. Trước tiên, chương trình kiểm tra tính hợp lệ của hai đỉnh đầu và cuối. Nếu không hợp lệ, thông báo "INVALID START OR END NODE" sẽ được hiển thị.

Nếu hợp lệ, các mảng visited và parent được khởi tạo để đánh dấu các đỉnh đã duyệt và lưu thông tin truy ngược. Thuật toán bắt đầu từ đỉnh start, sử dụng hàng đợi để quản lý các đỉnh chờ duyệt. Từng bước, các đỉnh kề được kiểm tra, nếu chưa duyệt, sẽ được thêm vào hàng đợi, đánh dấu và lưu thông tin cha. Đồng thời, chương trình vẽ trực quan các đỉnh và cạnh bằng màu đỏ, giúp người dùng theo dõi quá trình tìm kiếm.

Nếu tìm được đường đi đến end, chương trình sử dụng mảng parent để truy ngược và vẽ lại đường đi bằng màu vàng. Danh sách các đỉnh trong đường đi được hiển thị rõ ràng trên giao diện. Ngược lại, nếu không tìm thấy, thông báo "NO PATH FOUND" sẽ xuất hiện.

void BFS\_display() {

int start, end;

printf("Enter the starting vertex: ");

scanf("%d", &start);

printf("Enter the ending vertex: ");

scanf("%d", &end);

bfs(start, end, numV);

}

Hàm BFS\_display() được sử dụng để giao tiếp với người dùng, cho phép họ nhập vào đỉnh bắt đầu và đỉnh kết thúc của quá trình duyệt BFS. Khi được gọi, hàm sẽ hiển thị thông báo yêu cầu người dùng nhập đỉnh bắt đầu và đỉnh kết thúc, sau đó nhận giá trị đầu vào từ bàn phím thông qua scanf. Sau khi lấy được các giá trị này, hàm sẽ gọi bfs(start, end, numV) để thực hiện thuật toán BFS trên đồ thị. Tham số numV là số lượng đỉnh của đồ thị, được xác định từ trước.

## 2.2 Thiết kế giao diện người dùng

### 2.2.1. Xây dựng giao diện

// Ham ve button

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\_SIZE];

strcpy(buffer, label);

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

}

// Ve khung do hoa

void drawframe() {

setcolor(WHITE);

setlinestyle(0, 1, 3);

rectangle(10, 10, 550, 370);

rectangle(10, 380, 550, 480);

settextstyle(10, 0, 1);

drawButton(567, 10, 700, 50, "Infor");

drawButton(567, 70, 700, 110, "Load file");

drawButton(567, 130, 700, 170, "Start");

drawButton(567, 190, 700, 230, "Reset");

drawButton(567, 250, 700, 290, "Delete");

drawButton(567, 310, 700, 350, "Exit");

}

Hàm drawButton chịu trách nhiệm vẽ các nút bấm trên giao diện với các thông số đầu vào bao gồm tọa độ góc trên trái, góc dưới phải, và nhãn hiển thị của nút. Màu sắc và kiểu đường viền của các nút được thiết lập bằng các hàm đồ họa như setcolor và setlinestyle, đảm bảo các nút được thể hiện rõ ràng và nổi bật trên giao diện. Nội dung nhãn của mỗi nút được hiển thị tại vị trí phù hợp bên trong khung nút bằng hàm outtextxy.

Hàm drawframe chịu trách nhiệm vẽ toàn bộ khung giao diện chính, bao gồm khu vực hiển thị đồ thị, các nút chức năng và vùng thông tin. Khung đồ họa chính được biểu diễn bằng một hình chữ nhật nằm ở trung tâm, từ tọa độ (10, 10) đến (550, 370), dùng để hiển thị đồ thị và kết quả thuật toán BFS. Phía dưới khung chính là một vùng thông tin bổ sung, được vẽ từ tọa độ (10, 380) đến (550, 480), dành cho việc hiển thị các thông tin hoặc kết quả chi tiết.

Các nút chức năng được đặt ở bên phải khung chính, bao gồm các nút "Infor", "Load file", "Start", "Reset", "Delete", và "Exit". Những nút này cung cấp các chức năng chính như tải dữ liệu, bắt đầu thuật toán, đặt lại trạng thái, và thoát chương trình.

### 2.2.2 Xử lý sự kiện chuột

// Ham xu ly thao tac

void handleClick(int x, int y) {

// Infor

if(x >= 567 && x <= 700 && y >= 10 && y <= 50) {

cleanGraph();

drawframe();

settextstyle(10, 0, 2);

setcolor(RED);

outtextxy(80, 50, "BFS Visualization for Graphs");

settextstyle(8, 0, 1);

setcolor(WHITE);

outtextxy(80, 110, "Supervisor: M.Sc. DOAN VU THINH");

outtextxy(80, 160, "Student: TRAN DUC NGAN");

outtextxy(80, 210, "Student ID: 64131460");

settextstyle(6, 0, 1);

setcolor(YELLOW);

outtextxy(80, 270, "Thank you for reviewing the project!");

}

// Load file

if(x >= 567 && x <= 700 && y >= 70 && y <= 110) {

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;

isAdjSymmetry = true;

openAndReadFile(filePath);

ZeroMemory(szFile, sizeof(szFile));

}

clearGraph();

drawframe();

drawgraph();

}

// Start

if (x >= 567 && x <= 700 && y >= 130 && y <= 170) {

// Neu do thi da duoc tai

if (isGraphLoaded) {

// Neu BFS chua duoc chay, thi chay BFS

if (!isBFSExecuted) {

BFS\_display(); // Chay BFS

isBFSExecuted = true; // Danh dau la BFS da chay

} else {

clearGraph(); // Xoa do thi hien tai

drawframe(); // Ve lai khung

redrawgraph(); // Ve lai do thi

BFS\_display(); // Chay lai BFS

}

} else {

setcolor(WHITE);

cleanGraph(); // Xoa do thi

drawframe(); // Ve lai khung

outtextxy(195, 120, "NO GRAPH TO START"); // Khong co do thi de bat dau

outtextxy(60, 170, "CLICK THE 'Load file' BUTTON TO ADD GRAPH"); // Nhan nut 'Load file' de tai do thi

}

}

// Reset

if (x >= 567 && x <= 700 && y >= 190 && y <= 230) {

if (isGraphLoaded) {

isBFSExecuted = false; // Dat lai trang thai BFS

cleanGraph(); // Xoa do thi hien tai

drawframe(); // Ve lai khung

drawgraph(); // Ve lai do thi

} else {

setcolor(WHITE);

settextstyle(10, 0, 1);

outtextxy(195, 120, "NO GRAPH TO RESET"); // Khong co do thi de dat lai

}

}

// Delete

if (x >= 567 && x <= 700 && y >= 250 && y <= 290) {

cleanGraph(); // Xoa do thi

drawframe(); // Ve khung

outtextxy(180, 120, "DATA HAS BEEN DELETED"); // Du lieu da bi xoa

outtextxy(60, 170, "CLICK THE 'Load file' BUTTON TO ADD GRAPH"); // Click vao nut 'Load file' de them do thi

}

// Exit

if (x >= 567 && x <= 700 && y >= 310 && y <= 350) {

closegraph(); // Dong cua so do thi

}

}

Đoạn mã trên là một hàm handleClick dùng để xử lý các thao tác khi người dùng click chuột vào các vùng khác nhau trong giao diện đồ họa. Cụ thể:

* Infor: Khi người dùng nhấp vào nút "Infor" trong giao diện đồ họa vùng tọa độ (567, 10) đến (700, 50), chương trình sẽ gọi hàm cleanGraph() để xóa đồ thị hiện tại và thiết lập lại trạng thái ban đầu của chương trình. Sau đó, hàm drawframe() được gọi để vẽ lại toàn bộ khung giao diện. Tiếp theo, thông tin về dự án được hiển thị.
* Load file: Khi người dùng nhấn vào nút "Load file" trong giao diện đồ họa vùng tọa độ (567, 70) đến (700, 110), chương trình sẽ mở hộp thoại chọn file để người dùng chọn tệp đồ thị, mã sử dụng OPENFILENAME để thiết lập và hiển thị hộp thoại chọn file. Đầu tiên, chương trình lấy đường dẫn của thư mục hiện tại bằng hàm GetCurrentDirectory. Sau đó, cấu hình hộp thoại chọn file với các tham số như bộ lọc loại file (\*.\* cho tất cả các loại file). Khi người dùng chọn một file và nhấn "Open", chương trình sẽ lưu đường dẫn của file vào biến filePath. Hàm openAndReadFile(filePath) sẽ được gọi để mở và đọc nội dung file đồ thị đã chọn. Sau khi tải file xong, chương trình sẽ gọi clearGraph() để xóa đồ thị cũ, sau đó vẽ lại khung giao diện với drawframe() và vẽ đồ thị mới với drawgraph().
* Start: Khi người dùng nhấn vào nút "Start" trong giao diện đồ họa vùng tọa độ từ (567, 130) đến (700, 170), chương trình sẽ kiểm tra xem đồ thị đã được tải hay chưa. Nếu đồ thị đã được tải thành công, chương trình sẽ kiểm tra xem thuật toán BFS đã được chạy hay chưa. Nếu chưa, chương trình sẽ thực thi thuật toán BFS bằng cách gọi hàm BFS\_display() và đánh dấu trạng thái là đã chạy thuật toán BFS. Nếu thuật toán BFS đã được thực hiện trước đó, chương trình sẽ vẽ lại đồ thị, sau đó thực thi lại thuật toán BFS trên đồ thị đó. Trong trường hợp đồ thị chưa được tải, chương trình sẽ hiển thị thông báo yêu cầu người dùng tải đồ thị trước khi bắt đầu thuật toán, đồng thời nhắc nhở người dùng nhấn nút "Load file" để thêm đồ thị.
* Reset: Khi người dùng nhấn vào nút "Reset" trong giao diện đồ họa vùng tọa độ từ (567, 190) đến (700, 230), chương trình sẽ kiểm tra xem đồ thị đã được tải hay chưa. Nếu đồ thị đã được tải, chương trình sẽ đặt lại trạng thái của thuật toán BFS (bằng cách gán isBFSExecuted thành false), sau đó xóa đồ thị hiện tại bằng hàm cleanGraph() và yêu cầu người dùng vẽ lại đồ thị bằng cách bấm vào chuột trái mà không cần tải lại tệp tin đồ thị, Nếu đồ thị chưa được tải, chương trình sẽ hiển thị thông báo "NO GRAPH TO RESET" yêu cầu người dùng tải đồ thị trước khi có thể thực hiện thao tác reset.
* Delete: Khi người dùng nhấn vào nút "Delete" trong giao diện đồ họa vùng tọa độ từ (567, 250) đến (700, 290), chương trình sẽ thực hiện xóa đồ thị hiện tại bằng hàm cleanGraph(), sau đó vẽ lại khung giao diện bằng hàm drawframe(). Đồng thời, chương trình sẽ hiển thị thông báo "DATA HAS BEEN DELETED" để thông báo cho người dùng biết rằng dữ liệu đã bị xóa và yêu cầu người dùng nhấn nút "Load file" để tải đồ thị mới.
* Exit: Khi người dùng nhấn vào nút "Exit" trong giao diện đồ họa vùng tọa độ từ (567, 310) đến (700, 350), chương trình sẽ gọi hàm closegraph() để đóng cửa sổ đồ họa và kết thúc chương trình.

CHƯƠNG III. KẾT QUẢ THỰC HIỆN

## 3.1 Giao diện chính

Ảnh có chứa ảnh chụp màn hình

Mô tả được tạo tự động

Hình 8. Giao diện chính

Giao diện được thiết kế đơn giản bao gồm hai khu vực chính. Phần khung chính nằm ở trung tâm màn hình dùng để hiển thị nội dung chính. Bên dưới khung chính là một khung phụ nhỏ hơn, đóng vai trò khu vực hiển thị thông báo của chương trình. Phía bên phải giao diện là bảng điều khiển với 6 nút bấm, các nút bấm này bao gồm: "Infor", "Load file", "Start", "Reset", "Delete", và "Exit".

## 3.2 Nút “Infor”



Hình 9. Giao diện khi bấm nút “Infor”

Khi người dùng nhấn vào nút "Infor", giao diện hiển thị một màn hình giới thiệu thông tin đồ án.

## 3.3 Nút “Load file”

Ảnh có chứa văn bản, phần mềm, Biểu tượng máy tính, Phần mềm đa phương tiện

Mô tả được tạo tự động

Hình 10. Giao diện khi bấm nút “Load file”

Khi người dùng nhấn vào “Load file” chương trình sẽ mở hộp thoại chọn tệp, cho phép người dùng chọn một tệp từ hệ thống của mình.

Ảnh có chứa ảnh chụp màn hình, văn bản

Mô tả được tạo tự động

Hình 11. Giao diện sau khi đã “Load file”

Khi tệp đã được tải lên thành công, chương trình sẽ hiển thị thông báo “CLICK LEFT MOUSE BUTTON TO ADD A NODE”. Người dùng sẽ được yêu cầu sử dụng chuột trái để vẽ các đỉnh của đồ thị trên giao diện.

Ảnh có chứa ảnh chụp màn hình, văn bản, biểu đồ, thiết kế

Mô tả được tạo tự động

Hình 12. Giao diện khi đang đặt đỉnh của đồ thị

Người dùng cần lần lượt nhấn chuột trái để đặt từng nút tương ứng với các đỉnh của đồ thị. Mỗi lần nhấp chuột trái, một nút sẽ được vẽ lên màn hình tại vị trí con trỏ chuột. Quá trình này sẽ tiếp tục cho đến khi người dùng đặt đủ tất cả các nút theo số lượng đỉnh của đồ thị. Khi người dùng hoàn thành việc đặt tất cả các nút, thông báo sẽ biến mất, và chương trình sẽ hiện thị các cạnh của đồ thị có trong ma trận kề.

Ảnh có chứa ảnh chụp màn hình, biểu đồ

Mô tả được tạo tự động

Hình 13. Giao diện đồ thị có hướng

Ảnh có chứa ảnh chụp màn hình, biểu đồ, hàng, vòng tròn

Mô tả được tạo tự động

Hình 14. Giao diện đồ thị vô hướng

Nếu ma trận kề là đối xứng, chương trình sẽ vẽ đồ thị vô hướng. Ngược lại, nếu ma trận không đối xứng, đồ thị được vẽ sẽ là có hướng, với các mũi tên chỉ chiều của các cạnh.

## 3.4 Nút “Start”

Ảnh có chứa văn bản, ảnh chụp màn hình, màn hình, phần mềm

Mô tả được tạo tự động

Hình 15. Giao diện bấm nút “Start” khi chưa “Load file”

Khi người dùng nhấn vào nút "Start". Nếu đồ thị chưa được tải lên, chương trình sẽ hiển thị thông báo lỗi trên giao diện, yêu cầu người dùng nhấn nút "Load file" để tải đồ thị trước khi thực thi.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

Hình 16. Giao diện bấm nút “Start” khi đã “Load file” và đặt xong đồ thị

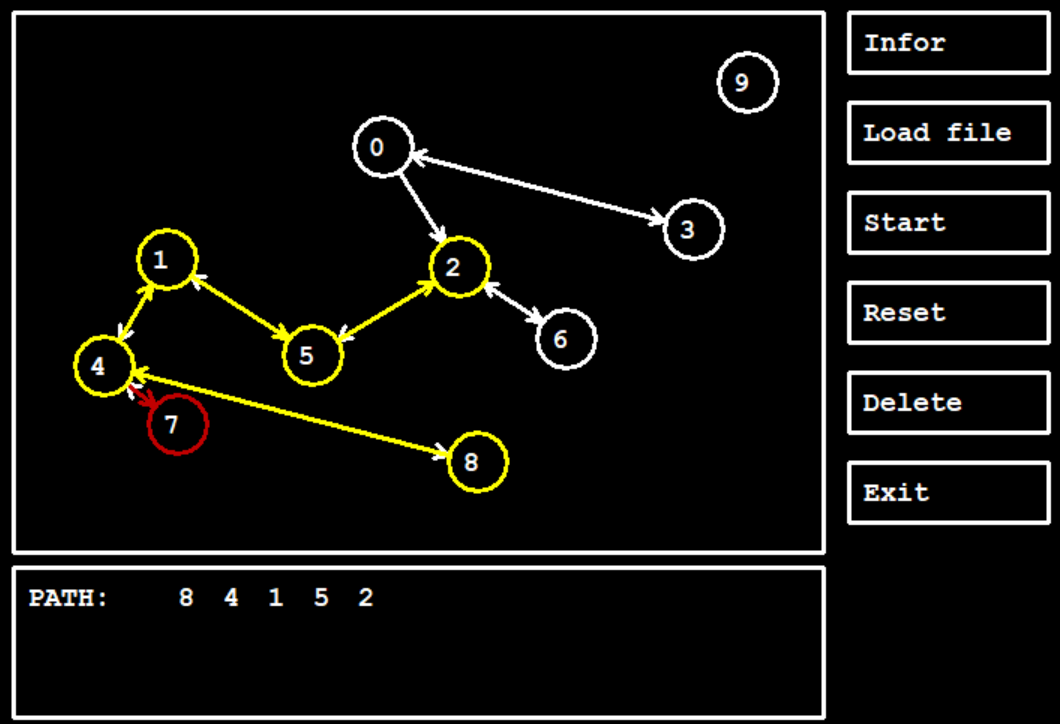
Sau khi người dùng nhấn nút "Start", nếu người dùng hoàn thành việc đặt các đỉnh, chương trình sẽ hiển thị một giao diện nhập yêu cầu người dùng nhập vào đỉnh bắt đầu và đỉnh kết thúc.

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, Phần mềm đa phương tiện

Mô tả được tạo tự động

Hình 17. Giao diện hiển thị khi thuật toán BFS đang tìm kiếm đường đi

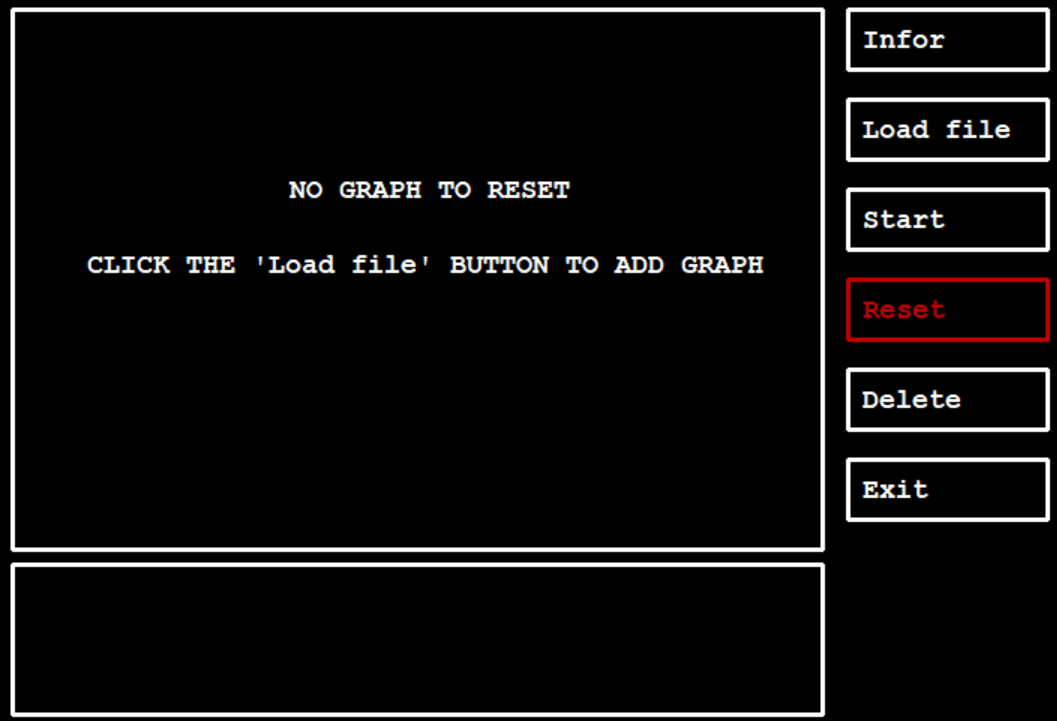
Khi người dùng nhập đỉnh bắt đầu và đỉnh kết thúc thì thuật toán BFS thực hiện quá trình tìm kiếm đường đi, các đỉnh và đường nối trên đồ thị sẽ được hiển thị với màu đỏ. Mỗi lần thuật toán khám phá một đỉnh hoặc một cạnh, chúng sẽ được tô màu đỏ, thể hiện bước đi tiếp theo của thuật toán trong quá trình tìm kiếm theo chiều rộng.



Hình 18. Giao diện hiển thị khi thuật toán BFS hoàn thành tìm kiếm đường đi

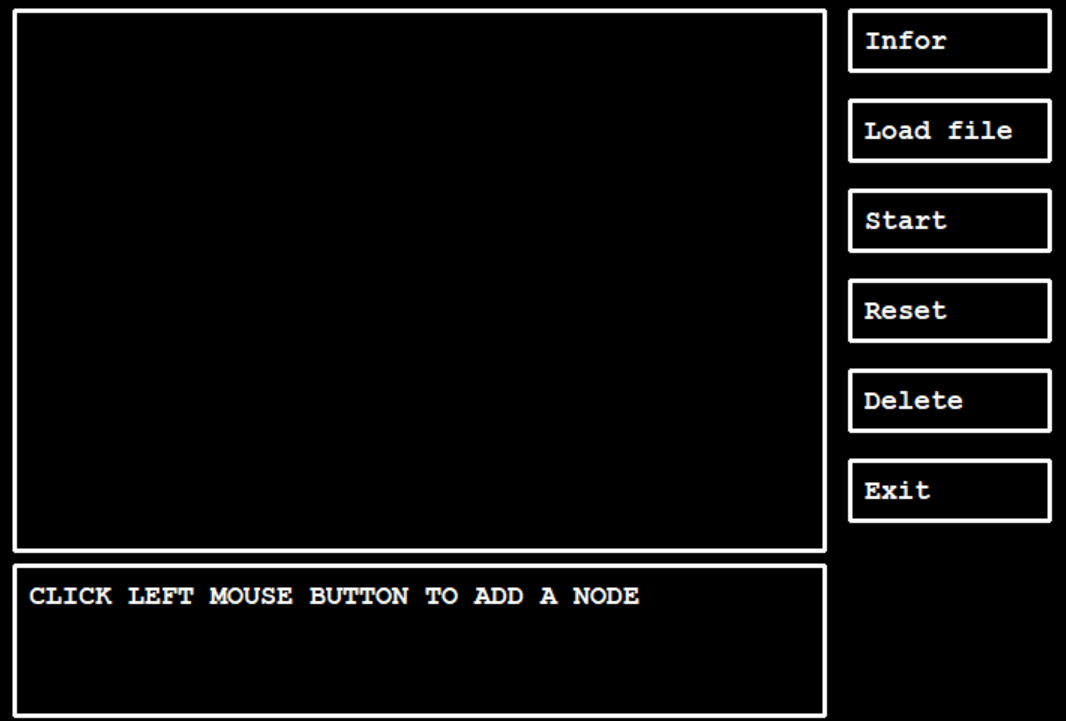
Khi thuật toán BFS hoàn thành, các đỉnh và cạnh trên đường đi này sẽ được tô màu vàng để nổi bật, thể hiện rõ quá trình tìm kiếm của thuật toán. Ngoài ra, một ô thông báo sẽ xuất hiện trên ô thông báo để hiển thị kết quả của thuật toán BFS, bao gồm đường đi từ đỉnh bắt đầu đến đỉnh kết thúc.

## 3.5 Nút “Reset”



Hình 19. Giao diện bấm nút “Reset” khi chưa “Load file”

Khi người dùng nhấn vào nút "Reset", nếu đồ thị chưa được tải lên, chương trình sẽ hiển thị thông báo lỗi yêu cầu người dùng tải đồ thị trước khi có thể thực hiện thao tác reset.



Hình 20. Giao diện bấm nút “Reset” khi đã “Load file”

Nếu đồ thị đã được tải lên, khi người dùng nhấn nút "Reset", chương trình sẽ xóa toàn bộ đồ thị hiện tại và khởi động lại quá trình vẽ đồ thị. Điều này có nghĩa là người dùng sẽ đặt lại các đỉnh của đồ thị, giúp người dùng có thể thực hiện lại các thao tác đặt lại các đỉnh mà không cần phải tải lại tệp đồ thị.

## 3.6 Nút “Delete”

Ảnh có chứa văn bản, ảnh chụp màn hình, phần mềm, màn hình

Mô tả được tạo tự động

Hình 21. Giao diện bấm nút “Delete”

Khi người dùng nhấn nút "Delete", chương trình sẽ xóa toàn bộ đồ thị hiện tại khỏi giao diện. Điều này có nghĩa là tệp vừa tải lên và tất cả các đỉnh và các cạnh của đồ thị sẽ bị loại bỏ, giúp người dùng bắt đầu lại từ đầu.

## 3.7 Nút “Exit”

Khi người dùng nhấn nút "Exit", chương trình sẽ đóng cửa sổ đồ thị và thoát khỏi giao diện đồ họa. Tất cả các hoạt động liên quan đến giao diện đồ họa sẽ được kết thúc, và chương trình sẽ kết thúc phiên làm việc với giao diện đồ hoạ.

CHƯƠNG IV. KẾT LUẬN

## 4.1 Ưu điểm

Ứng dụng trực quan hóa thuật toán tìm kiếm theo chiều rộng (BFS) đã đáp ứng đầy đủ các yêu cầu về chức năng và giao diện, cụ thể:

* Cho phép người dùng dễ dàng thao tác tải dữ liệu đồ thị từ file và bấm chuột trái để thêm đỉnh cho đồ thị.
* Các bước thực hiện thuật toán BFS được hiển thị rõ ràng trên giao diện đồ họa, giúp người dùng dễ dàng theo dõi quá trình duyệt và kiểm tra kết quả.
* Giao diện thân thiện, dễ sử dụng, hỗ trợ tương tác trực tiếp thông qua chuột và các phím chức năng.

## 4.2 Nhược điểm

Bên cạnh các ưu điểm nổi bật, đồ án vẫn còn một số hạn chế cần khắc phục trong tương lai:

* Hạn chế về khả năng hiển thị đồ thị có kích thước lớn với số lượng đỉnh và cạnh nhiều, dẫn đến việc hiển thị bị rối.
* Khi người dùng thêm các đỉnh vào đồ thị thì các nút chức năng khác không thể bấm vào được cho đến khi quá trình thêm đỉnh hoàn tất.

TÀI LIỆU THAM KHẢO

[1] VNOI Wiki. (2024). *Lý thuyết đồ thị*. Truy cập tại <https://wiki.vnoi.info/algo/graph-theory/graph>

[2] CP-Algorithms. (2024). *Thuật toán tìm kiếm theo chiều rộng - Algorithms for competitive programming*. Truy cập tại <https://cp-algorithms.com/graph/breadth-first-search.html>

[3] Đoàn Vũ Thịnh. (2019). *Bài giảng Kỹ thuật đồ họa*. Đại học Nha Trang.

[4] Đoàn Vũ Thịnh. (2024). Computer Graphics. Truy cập tại <https://github.com/thinhdoanvu/ComputerGraphics>