BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**

---🙢🕮🙠---



**ĐỀ TÀI: SỬ DỤNG CÔNG CỤ GRAPHICS.H MÔ PHỔNG THUẬT TOÁN DIJKSTRA**

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

**Sinh viên thực hiện: Trần Khải**

**Mã số sinh viên: 63130554**

**Lớp: 63.CNTT- 2**

**Khánh Hòa, thứ tư ngày 3 tháng 1 năm 2024**

**MỤC LỤC**

[LỜI CẢM ƠN 4](#_Toc155688811)

[TÓM TẮT 5](#_Toc155688812)

[CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI 6](#_Toc155688813)

[**1.** **Giới thiệu thuật toán Dijkstra** 6](#_Toc155688814)

[**2.** **Ý tưởng thuật toán Dijkstra** 6](#_Toc155688815)

[**3.** **Ngôn ngữ lập trình C** 6](#_Toc155688816)

[**4.** **Ứng dụng thực tiễn Dijkstra** 7](#_Toc155688817)

[**4.1** **Ưu điểm** 7](#_Toc155688818)

[**4.2** **Nhược điểm** 8](#_Toc155688819)

[CHƯƠNG 2: NGHIÊN CỨU THUẬT TOÁN DIJKSTRA VÀ ÁP DỤNG VÀO SẢN PHẨM 9](#_Toc155688820)

[**1.** **Định nghĩa thuật toán Dijkstra** 9](#_Toc155688821)

[**2.** **Nguyên lý hoạt động** 9](#_Toc155688822)

[**3.** **Ứng dụng vào sản phẩm sử dụng ngôn ngữ lập trình C** 10](#_Toc155688823)

[**3.1** **Các file chứa dữ liệu** 10](#_Toc155688824)

[**3.2** **Khởi tạo màn hình** 11](#_Toc155688825)

[**3.3** **Khởi tạo giao diện** 11](#_Toc155688826)

[**3.4** **Chức năng LoadFile** 15](#_Toc155688827)

[**3.5** **Vẽ các điểm lên màn hình** 17](#_Toc155688828)

[**3.6** **Nối các đỉnh theo ma trận kề** 19](#_Toc155688829)

[**3.7**  **Bắt sự kiện khi click chuột** 20](#_Toc155688830)

[**3.8** **Chức năng Keyboard** 23](#_Toc155688831)

[**3.9** **Hàm Dijkstra** 25](#_Toc155688832)

[**3.10** **Chức năng Reset** 26](#_Toc155688833)

[**3.11** **Chức năng Random** 26](#_Toc155688834)

[**3.12** **Chức năng Exit** 29](#_Toc155688835)

[CHƯƠNG 3: KẾT LUẬN VÀ KẾ HOẠCH TƯƠNG LAI 30](#_Toc155688836)

[**1.** **Kết luận** 30](#_Toc155688837)

[**2.** **Kế hoạch tương lai** 30](#_Toc155688838)

[TÀI LIỆU THAM KHẢO 32](#_Toc155688839)

|  |  |  |  |
| --- | --- | --- | --- |
| **Trần Khải 63130554**  **Lớp 63.CNTT-2** | | | |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 27/11/2023 | Đề tài: Mô phỏng thuật toán Dijkstra bằng công cụ graphics.h  Lên kế hoạch thực hiện  Lựa chọn giải pháp để cài đặt |  |
| 2 | 7/12/2023 | Khởi tạo thành công GUI  Tạo file graph.inp được các tọa độ x và y, ma trận kề chưa đúng  Hàm vẽ các node theo tọa độ trong file graph.inp  Hàm mouse() để thực hiện chức năng và bắt sự kiện click chuột  Chạy được chức năng LoadFile và Keyboard |  |
| 3 | 14/12/2023 | Hiểu được tuy Dijkstra là thuật toán tìm đường đi ngắn nhất nhưng trong thực tế sẽ không tối ưu trong các trường hợp phát sinh, ví dụ có vật cản trên đường, đường hỏng,…  Lên ý tưởng random các node nhưng tọa độ còn xấu và sai |  |
| 4 | 21/12/2023 | Thêm các hàm hỗ trợ random node và file graph\_temp.inp  Thay đổi 1 chút thuật toán Dijkstra |  |
| 5 | 29/12/2023 | Rút gọn code tối ưu nhất nhưng vẫn giữ được tính chính xác và logic.  Sửa được ma trận kề  Sửa được thuật toán Dijkstra  Thêm các hàm sau: void initGraph(), drawShortestPath(int parent[]), selectMinVertex(int value[], int processed[]);  Hiển thị thông tin sinh viên |  |
| 6 | 9/1/2024 | Nộp báo cáo lần cuối sau khi đã chỉnh sửa và hoàn thiện yêu cầu GV đã đề ra |  |

# 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 trính 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 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

Bài báo cáo này giúp chúng ta nhận thức thêm được một phần kiến thức, tầm quan trọng về thuật toán Dijkstra và sự ứng dụng của nó trong thực tiễn. Qua sản phẩm demo giúp ta hiểu rõ hơn về bản chất của thuật toán này và cách nó hoạt động như thế nào. Dù có một số hạn chế nhưng tính ứng dụng thực tế của thuật toán này không phải bàn cãi như: Hệ thống đường đi trong mạng giao thông., Mạng Truyền Thông, Định Tuyến Mạng Internet,…

Mời tất cả mọi người xem qua bài báo cáo để hiểu rõ hơn!

# CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1. **Giới thiệu thuật toán Dijkstra**

Thuật toán Dijkstra, được phát triển bởi nhà toán học Edsger W. Dijkstra vào năm 1956, là một trong những thuật toán quan trọng trong lĩnh vực đồ thị và tối ưu hóa đường đi. Thuật toán này được thiết kế để tìm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị có trọng số dương.

1. **Ý tưởng thuật toán Dijkstra**

Các bước chính của thuật toán Dijkstra để tìm đường đi ngắn nhất từ đỉnh a đến đỉnh z trên đồ thị G=(V,E,W) được mô tả như sau:

Bước 1 : T=V; Da = 0; Di = ∞, Vi ≠ a.

Bước 2 : Lặp cho đến khi z ∉T:

* Lấy ra khỏi T đỉnh Vi có Di nhỏ nhất.
* Đánh nhãn lại cho mọi Vj kề Vi và Vj T ∈ theo công thức: Dj = min{Dj, Di+Wij}.

1. **Ngôn ngữ lập trình C**

"C" là một ngôn ngữ lập trình tự do và mạnh mẽ được phát triển từ năm 1972 bởi Dennis Ritchie tại Bell Labs (nay là Nokia Bell Labs) để thực hiện hệ điều hành UNIX. Ngôn ngữ này đã có ảnh hưởng lớn đối với nhiều ngôn ngữ lập trình khác và đóng vai trò quan trọng trong sự phát triển của hệ điều hành và ứng dụng máy tính.

Dưới đây là một số khái niệm cơ bản về ngôn ngữ lập trình C:

1. Mục Đích Ban Đầu:

Ban đầu, C được thiết kế để phát triển hệ điều hành UNIX. Tuy nhiên, do tính chất đơn giản, hiệu quả và di động của nó, C đã trở thành một ngôn ngữ lập trình phổ biến trong phát triển ứng dụng và hệ thống nhúng.

1. Tính Chất Chính:

Ngôn Ngữ Bậc Thấp: C là một ngôn ngữ bậc thấp, cung cấp kiểm soát cao với phần lớn chức năng của nó thể hiện trực tiếp các khái niệm của máy tính. Khả Năng Di Động: Chương trình C có thể chạy trên nhiều hệ điều hành và kiến trúc máy tính khác nhau mà không cần sửa đổi nhiều.

1. Cú Pháp Đơn Giản:

Cung cấp một cú pháp đơn giản với số lượng ít từ khóa và quy tắc ngữ pháp đơn giản.

1. Hỗ Trợ Đa Nhiệm:

Cung cấp hỗ trợ cho lập trình đa nhiệm và đồng thời cho các ứng dụng thời gian thực.

1. Hệ Thống Thư Viện Phong Phú:

Cung cấp nhiều thư viện chuẩn để hỗ trợ nhiều loại công việc khác nhau.

1. Kiểu Dữ Liệu:

Hỗ trợ kiểu dữ liệu nguyên thủy như int, char, float và kiểu dữ liệu phức tạp như struct và union.

1. Trình Biên Dịch:

Mã nguồn C thường được biên dịch thành mã máy trực tiếp, giúp tăng hiệu suất thực thi của chương trình.

1. Cộng Đồng Lớn:

Có một cộng đồng lập trình C lớn và tích cực, giúp hỗ trợ và phát triển ngôn ngữ.

1. Ứng Dụng:

C được sử dụng rộng rãi trong phát triển hệ điều hành, ứng dụng hệ thống nhúng, phần mềm nhúng, và nhiều loại ứng dụng khác.

1. Tiếp Tục Phát Triển:

Mặc dù đã có nhiều ngôn ngữ lập trình mới nổi lên, nhưng C vẫn tiếp tục được sử dụng và phát triển trong các lĩnh vực như trí tuệ nhân tạo, phân tích dữ liệu và phát triển phần mềm nhúng.

1. **Ứng dụng thực tiễn Dijkstra**

### **Ưu điểm**

1. Tìm Đường Đi Ngắn Nhất:

Thuật toán Dijkstra chủ yếu được sử dụng để tìm đường đi ngắn nhất giữa hai điểm trong một mạng lưới hoặc đồ thị có trọng số dương.

1. Quản Lý Mạng Giao Thông:

Ứng dụng trong quản lý giao thông để tối ưu hóa đường đi và giảm thiểu thời gian di chuyển.

1. Quản Lý Tài Nguyên Mạng:

Sử dụng để tối ưu hóa quản lý tài nguyên trong mạng máy tính, giảm độ trễ và tăng hiệu suất.

1. Lập Kế Hoạch Đường Đi Trong Robot:

Trong lĩnh vực robot tự động, thuật toán Dijkstra được sử dụng để lập kế hoạch đường đi cho robot sao cho nó di chuyển từ một vị trí đến vị trí khác một cách hiệu quả nhất.

1. Tìm Đường Đi Trong Bản Đồ:

Ứng dụng trong các ứng dụng bản đồ trực tuyến để tìm đường đi nhanh nhất từ một địa điểm đến địa điểm khác.

1. Mô Phỏng Môi Trường:

Trong lĩnh vực trí tuệ nhân tạo, thuật toán Dijkstra được sử dụng để mô phỏng và lập kế hoạch đường đi cho các tác vụ như xe tự lái.

### **Nhược điểm**

1. Không Xử Lý Trọng Số Âm:

Một trong nhược điểm lớn nhất của thuật toán Dijkstra là không thể xử lý được đồ thị có trọng số âm. Khi có trọng số âm, thuật toán có thể không đưa ra kết quả đúng.

1. Độ Phức Tạp Tăng Nhanh:

Với đồ thị lớn, độ phức tạp của thuật toán Dijkstra có thể tăng nhanh, đặc biệt là với các cài đặt sử dụng hàng đợi ưu tiên.

1. Cần Lưu Trữ Toàn Bộ Bảng Khoảng Cách:

Yêu cầu lưu trữ toàn bộ bảng khoảng cách, điều này có thể là vấn đề với đồ thị lớn, dẫn đến sự tiêu tốn bộ nhớ lớn.

1. Khả Năng Lặp Nhiều Lần:

Trong một số trường hợp, thuật toán Dijkstra có thể lặp nhiều lần, điều này có thể làm tăng độ phức tạp và làm giảm hiệu suất.

1. Không Linh Hoạt Khi Thay Đổi Đồ Thị:

Khi đồ thị thay đổi, ví dụ thêm hoặc xoá cạnh, cần phải thực hiện lại toàn bộ quá trình Dijkstra để có kết quả chính xác.

Mặc dù có nhược điểm, thuật toán Dijkstra vẫn là một công cụ quan trọng và được ứng dụng rộng rãi trong nhiều lĩnh vực do tính hiệu quả và độ chính xác của nó.

# CHƯƠNG 2: NGHIÊN CỨU THUẬT TOÁN DIJKSTRA VÀ ÁP DỤNG VÀO SẢN PHẨM

1. **Định nghĩa thuật toán Dijkstra**

Thuật toán Dijkstra là một trong những thuật toán cổ điển để giải quyết bài toán tìm đường đi ngắn nhất từ một điểm cho trước tới tất cả các điểm còn lại trong đồ thị có trọng số.

1. **Nguyên lý hoạt động**

Thuật toán Dijkstra tập trung vào việc tìm kiếm đường đi ngắn nhất từ một đỉnh nguồn đến tất cả các đỉnh còn lại trong đồ thị có trọng số dương. Dưới đây là ý tưởng cơ bản của thuật toán:

1. Khởi tạo:

* Đặt khoảng cách từ đỉnh nguồn đến chính nó là 0 và đến tất cả các đỉnh khác là vô cùng.
* Tạo một hàng đợi ưu tiên và thêm tất cả các đỉnh vào hàng đợi với khoảng cách hiện tại của chúng từ đỉnh nguồn.

1. Lặp Cho Đến Khi Hàng Đợi Trống:

* Trong mỗi lần lặp, chọn đỉnh u có khoảng cách ngắn nhất từ đỉnh nguồn.
* Duyệt qua tất cả các đỉnh kề của u.

1. Cập Nhật Khoảng Cách:

Nếu khoảng cách từ đỉnh nguồn đến u cộng với trọng số của cạnh (u, v) nhỏ hơn khoảng cách hiện tại từ đỉnh nguồn đến v, thì cập nhật khoảng cách đó.

1. Đánh Dấu Đỉnh:

Đánh dấu đỉnh u đã được xét để tránh lặp.

1. Lặp Lại:

Lặp lại bước 2 và 3 cho đến khi hàng đợi ưu tiên trống.

1. Kết Quả:

Kết quả cuối cùng sẽ là bảng khoảng cách từ đỉnh nguồn đến tất cả các đỉnh trong đồ thị.

1. Kiểm Tra Đường Đi:

Để xác định đường đi ngắn nhất từ đỉnh nguồn đến một đỉnh bất kỳ, ta có thể lặp lại từ đỉnh đó theo ngược lại, sử dụng thông tin được lưu trữ trong quá trình thực hiện thuật toán.

1. Kết Luận:

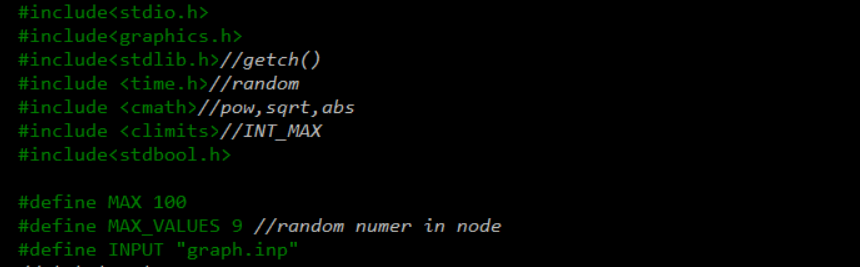
Thuật toán Dijkstra đảm bảo tìm ra đường đi ngắn nhất từ đỉnh nguồn đến tất cả các đỉnh khác, và nó dừng lại khi tất cả các đỉnh đã được xét.

1. Độ Phức Tạp:

Độ phức tạp của thuật toán Dijkstra phụ thuộc vào cách triển khai, nhưng thường là O((V + E) \* log(V)), trong đó V là số đỉnh và E là số cạnh trong đồ thị. Thuật toán Dijkstra là một phương pháp hiệu quả và rất phổ biến để giải quyết vấn đề tìm đường đi ngắn nhất trong đồ thị có trọng số dương.

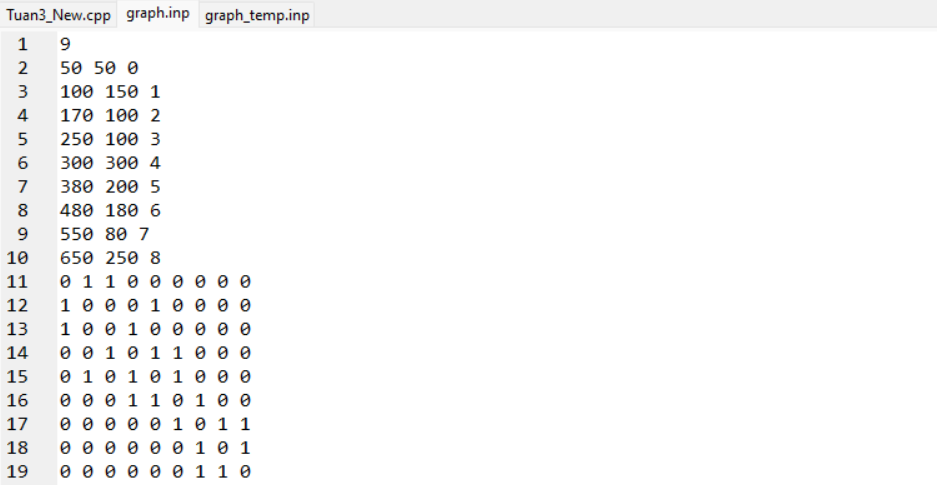
1. **Ứng dụng vào sản phẩm sử dụng ngôn ngữ lập trình C**

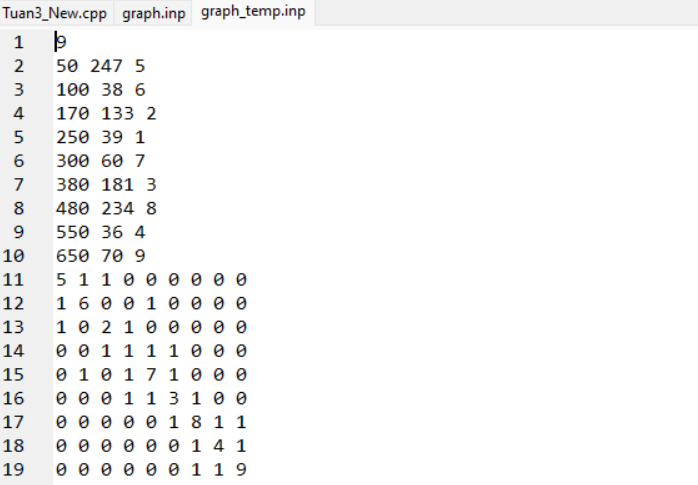
**Các thư viện cần thiết**



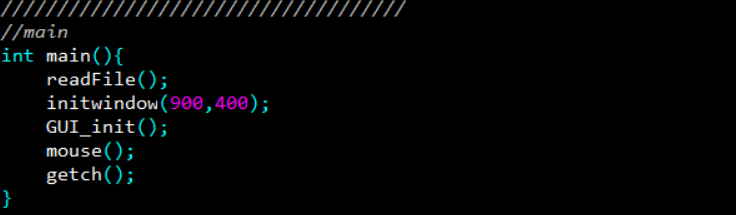
### **Các file chứa dữ liệu**

Gồm graph.inp và graph\_temp.inp





### **Khởi tạo màn hình**



### **Khởi tạo giao diện**

**Giao diện người dùng**



void GUI\_init(){

    setcolor(WHITE);

    setlinestyle(0,1,3);//main border

    rectangle(10,10,680,320);

    // button load file

    rectangle(10,380,100,330);

    settextstyle(2,0,7);

    outtextxy(20,340,"Load file");

    // keyboard

    rectangle(115,380,220,330);

    settextstyle(2,0,7);

    outtextxy(125,340,"Keyboard");

    // demo

    rectangle(235,380,340,330);

    settextstyle(2,0,7);

    outtextxy(245,340,"Dijkstra");

    // reset

    rectangle(355,380,450,330);

    settextstyle(2,0,7);

    outtextxy(370,340,"Reset");

    // random

    rectangle(465,380,570,330);

    settextstyle(2,0,7);

    outtextxy(475,340,"Random");

    // exit

    rectangle(585,380,680,330);

    settextstyle(2,0,7);

    outtextxy(610,340,"Exit");

    rectangle(690,10,890,380);

    settextstyle(2,0,7);

    outtextxy(720, 20,"P.INFORMATION");

    outtextxy(695,40,"Full Name: Tran Khai");

    outtextxy(695,60,"Class: 63.CNTT-2");

    outtextxy(695,80,"Topic: Dijkstra with graphics");

    outtextxy(695,100,"Teacher: Doan Vu Thinh");

    outtextxy(695,150,"Describe: Tim duong di");

    outtextxy(695,170,"ngan nhat tu dinh thu");

    outtextxy(695,190,"0 den dinh thu 8 trong");

    outtextxy(695,210,"do thi");

}

1. Màu Sắc và Kiểu Nét Vẽ:

setcolor(WHITE);: Đặt màu vẽ là trắng.

setlinestyle(0, 1, 3);: Đặt kiểu nét vẽ là 3 (nét đứt).

setcolor(7);: Đặt màu vẽ là màu mặc định (trắng).

1. Vẽ Hình Chữ Nhật:

rectangle(x1, y1, x2, y2);: Vẽ hình chữ nhật từ điểm (x1, y1) đến (x2, y2).

1. Vẽ Văn Bản:

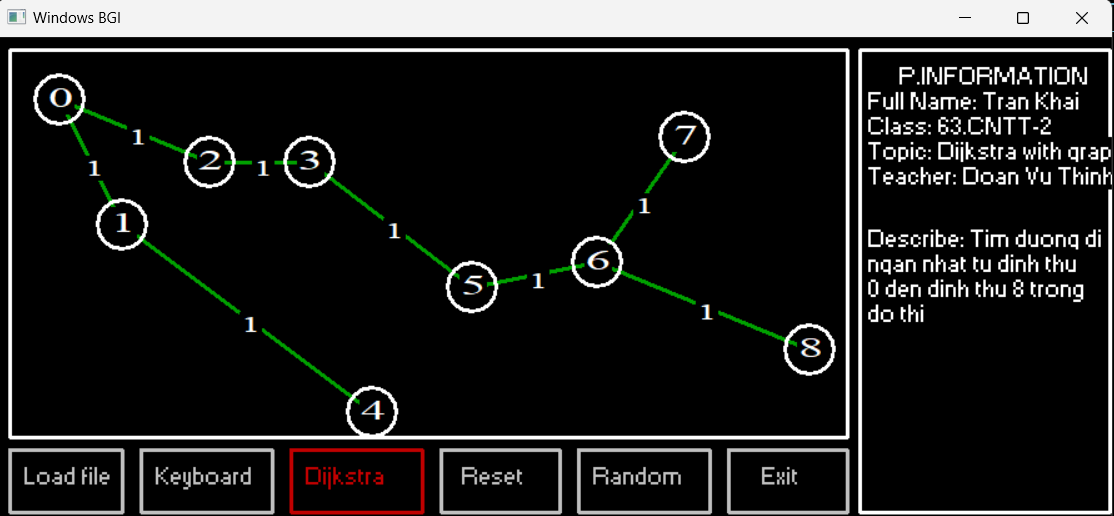
settextstyle(2, 0, 7);: Đặt kiểu văn bản với font 7, không in đậm (0), kích thước 2.

outtextxy(x, y, "Text");: Đặt văn bản ở vị trí (x, y).

1. Tọa độ:

Canh tọa độ x và y để hiển thị lên màn hình các button và text và vẽ ở trên

**Giao diện demo**



void GUI\_demo(){

    setcolor(WHITE);

    setlinestyle(0,1,3);//main border

    rectangle(10,10,680,320);

    setcolor(7);

    // button load file

    rectangle(10,380,100,330);

    settextstyle(2,0,7);

    outtextxy(20,340,"Load file");

    // keyboard

    rectangle(115,380,220,330);

    settextstyle(2,0,7);

    outtextxy(125,340,"Keyboard");

    // demo

    setcolor(4);

    rectangle(235,380,340,330);

    settextstyle(2,0,7);

    outtextxy(245,340,"Dijkstra");

    setcolor(7);

    // reset

    rectangle(355,380,450,330);

    settextstyle(2,0,7);

    outtextxy(370,340,"Reset");

    // random

    rectangle(465,380,570,330);

    settextstyle(2,0,7);

    outtextxy(475,340,"Random");

    // exit

    rectangle(585,380,680,330);

    settextstyle(2,0,7);

    outtextxy(610,340,"Exit");

}

Tương tự như phần GUI\_init nhưng có thêm setclor(4) ở dòng sau “Dijkstra” để tạo hiển thị như nút Dijkstra đang chạy và có màu đỏ

### **Chức năng LoadFile**

**Đọc File**

void readFile(){

fp = fopen(INPUT,"r");

    if(fp==NULL){

        printf("File not found");

        return;

    }

    fscanf(fp, "%d",&sodinh);

    for(int i=0;i<sodinh;i++) {

        fscanf(fp, "%d %d %d", &td[i].x, &td[i].y, &td[i].value);

    }

    // Ð?c ma tr?n k? sau khi dã d?c t?t c? các di?m

    for(int i=0;i<sodinh;i++){

        for (int j=0;j<sodinh;j++){

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

        }

    }

    printPoints(td, sodinh);

    printf("Ma tran ke:\n");

    for(int i=0;i<sodinh;i++){

        for(int j=0;j<sodinh;j++){

            printf("%d ",matrix[i][j]);

        }

        printf("\n");

    }

    fclose(fp);

}

Mở Tệp và Kiểm Tra Lỗi:

fp = fopen(INPUT, "r");

if (fp == NULL) {

printf("File not found");

return;

}

Mở tệp có tên được định nghĩa bởi INPUT với chế độ đọc ("r").

Kiểm tra xem tệp có tồn tại không (fp == NULL). Nếu không, hiển thị thông báo lỗi và thoát khỏi hàm.

Đọc Số Đỉnh và Tọa Độ Các Đỉnh:

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

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

fscanf(fp, "%d %d %d", &td[i].x, &td[i].y, &td[i].value);

}

Đọc số đỉnh (sodinh) từ tệp.

Đọc tọa độ và giá trị của mỗi đỉnh từ tệp.

Đọc Ma Trận Kề:

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

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

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

}

}

Đọc ma trận kề từ tệp.

In Thông Tin Đỉnh và Ma Trận Kề:

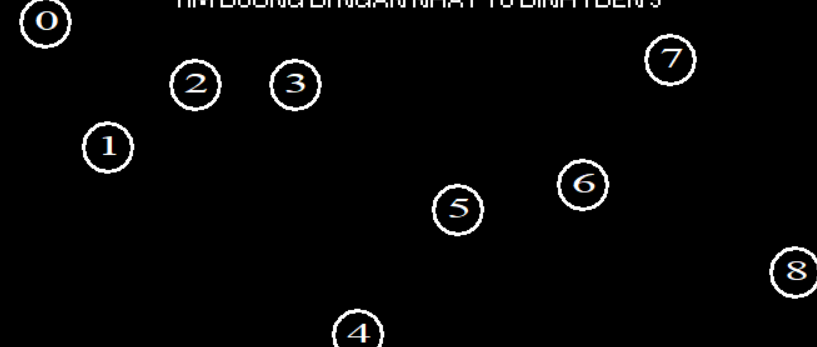
Gọi hàm printPoints() để in thông tin về các đỉnh.

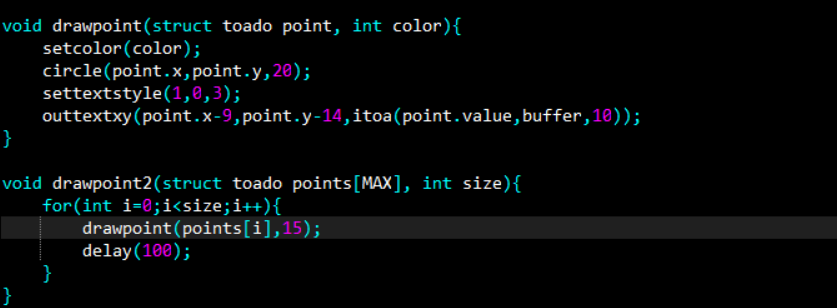
In ma trận kề ra màn hình.

fclose(fp);

Đóng tệp sau khi đã đọc xong.

### **3.5 Vẽ các điểm lên màn hình**





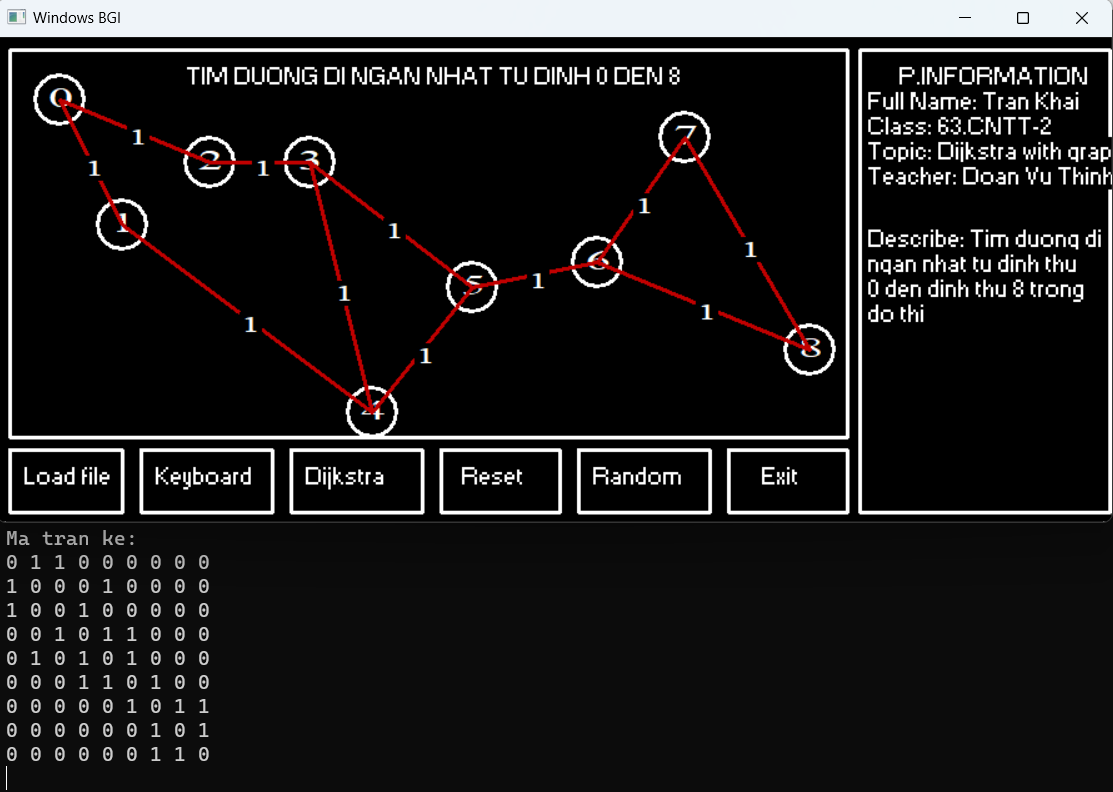
1. drawpoints()

* Hàm này vẽ một điểm tại tọa độ (point.x, point.y) với màu được đặt bởi tham số color.
* Sử dụng hàm circle để vẽ hình tròn (điểm) và settextstyle để đặt kiểu văn bản.
* Sử dụng outtextxy để hiển thị giá trị của điểm tại vị trí (point.x - 9, point.y- 14) trên màn hình.

2. drawpoints2()

* Hàm này vẽ một danh sách các điểm từ mảng points với kích thước size.
* Gọi hàm drawpoint để vẽ từng điểm trong danh sách.
* Sử dụng hàm delay(100) để tạo ra hiệu ứng độ trễ giữa các điểm khi vẽ.

### **3.6 Nối các đỉnh theo ma trận kề**



void draw\_line(int start, int end, int color, int length) {

    setcolor(color);

    line(td[start].x, td[start].y, td[end].x, td[end].y);

    // in khong cach cac node

    int mid\_x = (td[start].x + td[end].x) / 2;

    int mid\_y = (td[start].y + td[end].y) / 2;

    // display

    setcolor(WHITE);

    settextstyle(1, 0, 1);

    char length\_str[5];

    itoa(length, length\_str, 10);

    outtextxy(mid\_x - 5, mid\_y - 5, length\_str);

    delay(100);

}

void initGraph() {

    drawpoint2(td,sodinh);

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

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

            if (matrix[i][j] != 0) {

                draw\_line(i, j, 4, matrix[i][j]);

                delay(100);

            }

        }

    }

}

1. draw\_line()

Hàm này vẽ một đường thẳng giữa hai đỉnh start và end với màu color.Tính toán tọa độ trung điểm của đoạn thẳng để hiển thị độ dài của đường nối. Sử dụng itoa để chuyển độ dài thành chuỗi và outtextxy để hiển thị chuỗi trên màn hình.

2. initGraph()

Hàm này vẽ các đỉnh bằng cách gọi hàm drawpoint2. Sau đó, với mỗi cặp đỉnh có giá trị khác không trong ma trận kề, hàm gọi draw\_line để vẽ đường nối giữa chúng với màu đỏ và có độ dài tương ứng.

### **3.7 Bắt sự kiện khi click chuột**

//using mouse select button

void mouse() {

    int x\_mouse, y\_mouse;

    while(enable\_click) {

        if(ismouseclick(WM\_LBUTTONDOWN)) {

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

            if(x\_mouse>10&&x\_mouse<100&&y\_mouse>330&& y\_mouse<380){ //

                printf("Read from file ...");

                outtextxy(150,20,"TIM DUONG DI NGAN NHAT TU DINH 0 DEN

8");

                readFile();

                initGraph();

            }

            if(x\_mouse>115&&x\_mouse<220&&y\_mouse>330&& y\_mouse<380){ //

                printf("INPUT ...");

                keyboard();

                demo();

            }

            //Dijkstra

            if(x\_mouse>235&&x\_mouse<340&&y\_mouse>330&& y\_mouse<380){ //

                printf("Running ...");

                readFile();

                demo();

                enable\_click = 1;

            }

            if(x\_mouse>355&&x\_mouse<450&&y\_mouse>330&& y\_mouse<380){ //

                printf("Resting ...");

                cleardevice();

                GUI\_init();

            }

            if(x\_mouse>465&&x\_mouse<570&&y\_mouse>330&& y\_mouse<380){

                printf("Random ...");

                drawpoint\_rand(td, sodinh);

                draw\_line\_rand();

            }

            if(x\_mouse>585&&x\_mouse<680&&y\_mouse>330&& y\_mouse<380){

                printf("Exiting ...");

                exit(0);

            }

        }

        if (ismouseclick(WM\_RBUTTONDOWN)) {

            getmouseclick(WM\_RBUTTONDOWN, x\_mouse, y\_mouse);

            printf("\nExiting");

            break;

        }

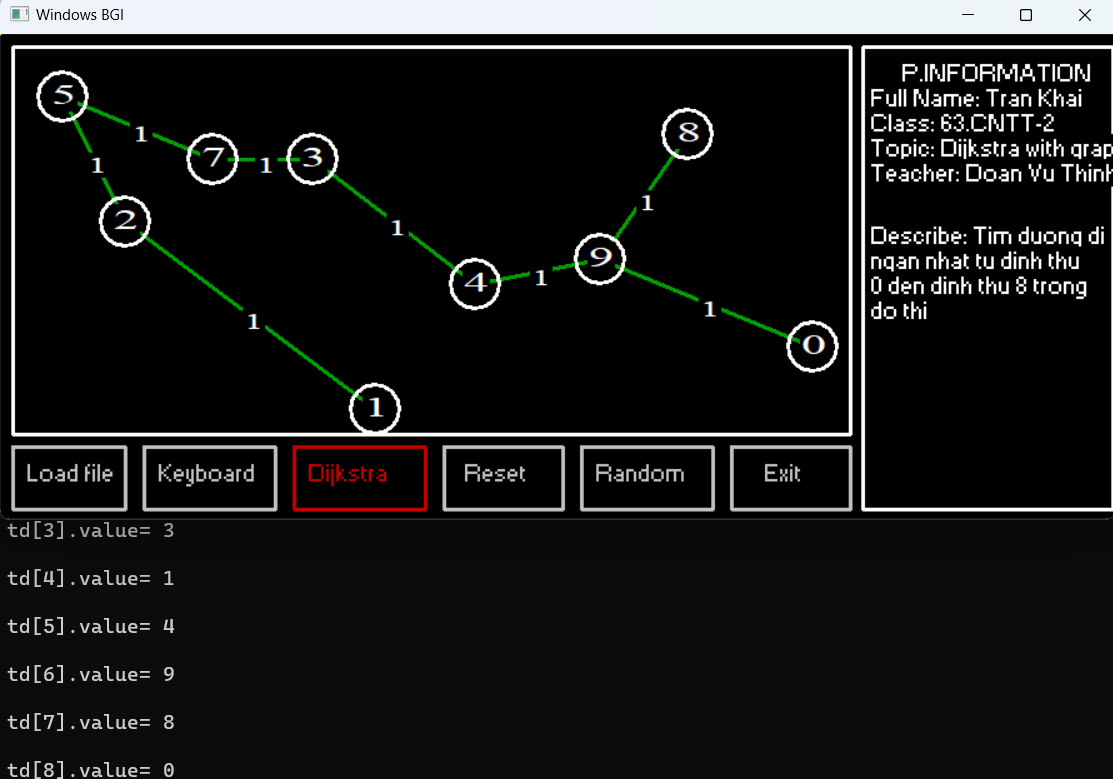
        delay(100);

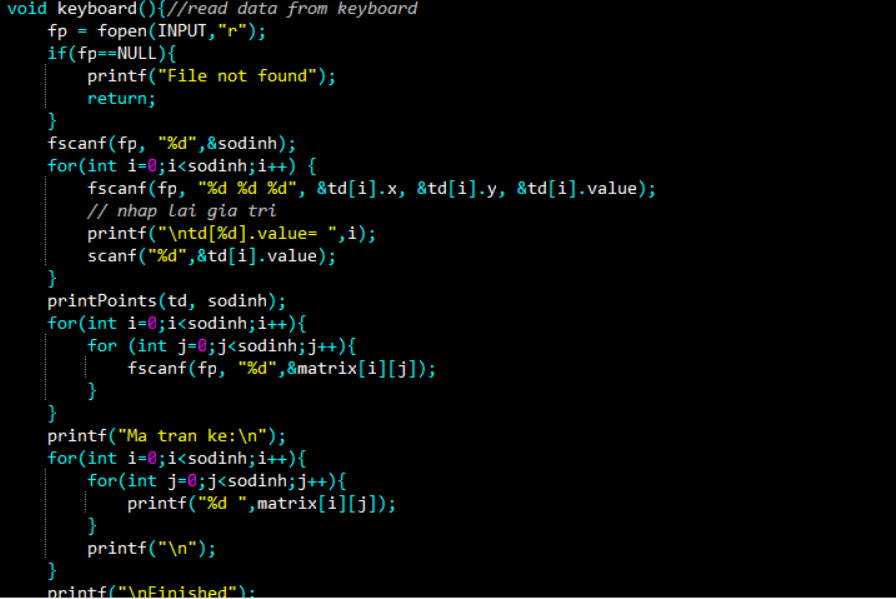
    }

}

* Dùng vòng lặp để liên tục kiểm tra sự kiện chuột trong khi enableclick = 1 là true.
* Nếu có sự kiện chuột trái (WM\_LBUTTONDOWN), lấy tọa độ chuột (x\_mouse, y\_mouse) và xử lý các sự kiện tương ứng.
* Kiểm tra xem chuột trái đã được nhấn ở vị trí nút "Load file" hay không và thực hiện các hành động tương ứng.
* Nếu có sự kiện chuột phải (WM\_RBUTTONDOWN), in ra màn hình và thoát khỏi vòng lặp.
* Sử dụng delay(100) để giảm tải CPU và giữ cho vòng lặp chạy mỗi 100 milliseconds.
* Nếu có sự kiện chuột phải, thoát khỏi vòng lặp và kết thúc hàm.

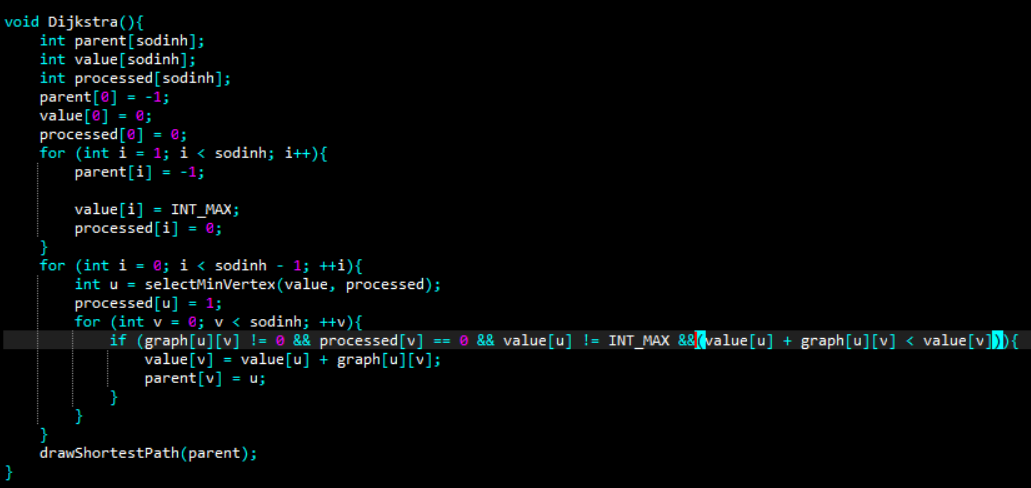
### **3.8 Chức năng Keyboard**





* Mở tệp INPUT với chế độ đọc ("r").
* Kiểm tra xem tệp có tồn tại không. Nếu không, hiển thị thông báo lỗi và thoát khỏi hàm.
* Đọc số đỉnh (sodinh) từ tệp.
* Đọc tọa độ và giá trị của mỗi đỉnh từ tệp và sau đó nhập lại giá trị từ bàn phím.
* Gọi hàm printPoints để in thông tin về các đỉnh.
* Đọc ma trận kề từ tệp.
* In ma trận kề ra màn hình và thông báo hoàn thành.
* Đóng tệp sau khi đã đọc xong.

### **3.9 Hàm Dijkstra**



1. Khởi tạo mảng và biến

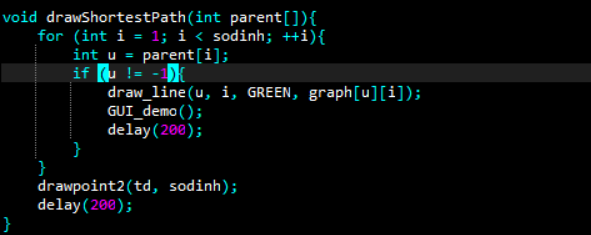
* Tạo ba mảng parent, value, và processed để lưu thông tin về đỉnh cha, giá trị ngắn nhất, và trạng thái xử lý của từng đỉnh.
* Đặt giá trị ban đầu cho đỉnh xuất phát (parent[0] = -1, value[0] = 0, processed[0] = 0).
* Đặt giá trị khởi tạo cho các đỉnh khác (parent[i] = -1, value[i] = INT\_MAX, processed[i] = 0).

2. Thuật toán Dijkstra

* Sử dụng một vòng lặp để lặp qua sodinh - 1 bước (đối với từng đỉnh, trừ đỉnh xuất phát).
* Tại mỗi bước, chọn đỉnh u chưa được xử lý có giá trị value nhỏ nhất.
* Đánh dấu u đã được xử lý.
* Duyệt qua tất cả các đỉnh v chưa được xử lý và có kết nối với u. Nếu có một đường đi ngắn hơn từ đỉnh xuất phát đến v thông qua u, cập nhật giá trị value[v] và đỉnh cha parent[v].

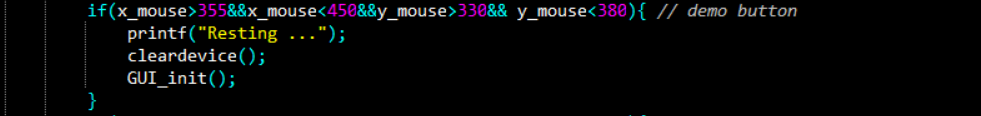
3. Vẽ đường đi ngắn nhất

* Gọi hàm drawShortestPath để vẽ đường đi ngắn nhất trên đồ thị.

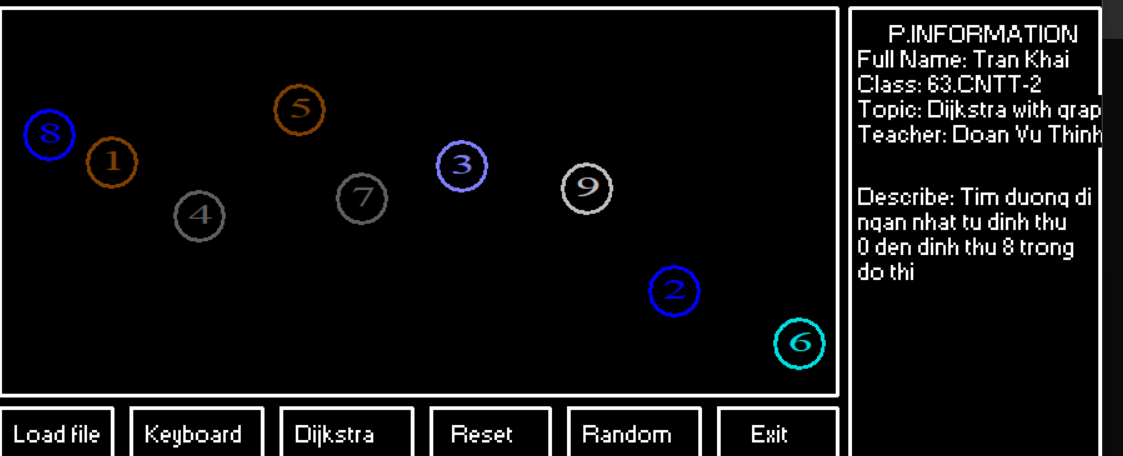


### **3.10 Chức năng Reset**

Xóa tất cả dữ liệu trên màn hình sau đó trả lại hàm GUI\_init để hiển thị giao diện người dùng



### **3.11 Chức năng Random**



**Random các đỉnh**

void drawpoint\_rand(struct toado points[MAX], int size) {

    readRandomGraph();

    srand(time(NULL));

    refreshGraph();

    FILE \*fp\_rand = fopen("graph\_temp.inp", "w");

    if (fp\_rand == NULL) {

        printf("File not found");

        return;}

    // ghi so dinh

    fprintf(fp\_rand, "%d\n", sodinh);

    for(int i = 0; i < size; i++){

        struct toado randomPoint;

        randomPoint.x = points[i].x;

        randomPoint.y = 30 + rand() % (300 - 30 + 1);

        int randomColor = rand() % 9 + 1;

        randomPoint.value = rand\_number\_in\_node();

        drawpoint(randomPoint, randomColor);

        // ghi toa do vao file

        fprintf(fp\_rand, "%d %d %d\n", randomPoint.x, randomPoint.y, randomPoint.value);

        // Cap nhat ma tran ke

        graph[i][i] = randomPoint.value;

        delay(100);

    }

    //ghi ma tran ke vao file

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

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

            // ghi ma tran ke vao file

            fprintf(fp\_rand, "%d ", graph[i][j]);

        }

        fprintf(fp\_rand, "\n");

    }

    fclose(fp\_rand);

    // Cap nhat ma tran ke random

    for (int i = 0; i < size; i++) {

        graph[i][i] = td[i].value;

    }

}

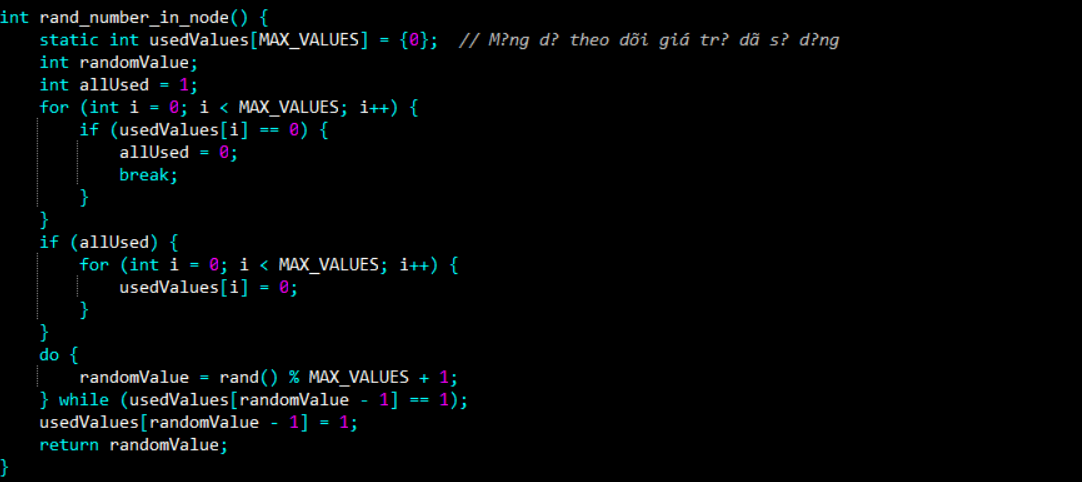
* Gọi hàm readRandomGraph để đọc thông tin của đồ thị ngẫu nhiên từ tệp.
* Sử dụng srand(time(NULL)) để khởi tạo seed ngẫu nhiên dựa trên thời gian hiện tại.
* Gọi hàm refreshGraph để làm mới đồ thị, có thể là việc thiết lập lại giá trị ban đầu hoặc đặt các giá trị mặc định.

A black screen with white text

Description automatically generated

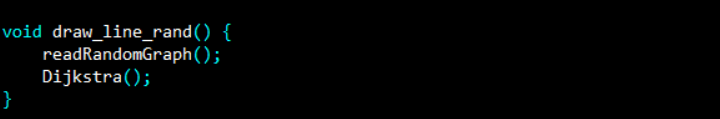
* Mở một tệp có tên là "graph\_temp.inp" để ghi thông tin về đồ thị ngẫu nhiên.
* Sử dụng vòng lặp để tạo và vẽ các điểm ngẫu nhiên.
* Tọa độ x của mỗi điểm giữ nguyên từ danh sách points, tọa độ y được chọn ngẫu nhiên trong khoảng từ 30 đến 300.
* Màu sắc cũng được chọn ngẫu nhiên và giá trị của mỗi điểm được tạo bằng cách gọi hàm rand\_number\_in\_node.
* Đồng thời, ghi tọa độ của điểm vào tệp và cập nhật ma trận kề.
* Ghi ma trận kề vào tệp và đóng tệp sau khi ghi xong.
* Ma trận kề sau đó được cập nhật với các giá trị ban đầu từ td

**Random giá trị trong đỉnh**



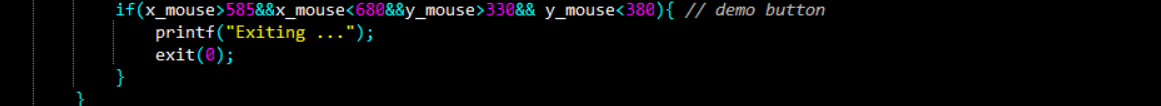
* usedValues là một mảng tĩnh được sử dụng để theo dõi giá trị nào đã được sử dụng và giá trị nào chưa được sử dụng. Kích thước của mảng là MAX\_VALUES.
* Dùng một vòng lặp để kiểm tra xem tất cả giá trị trong mảng usedValues đã được sử dụng chưa.
* Nếu có ít nhất một giá trị chưa được sử dụng (usedValues[i] == 0), đặt allUsed thành 0 để báo hiệu rằng chưa sử dụng hết.
* Nếu tất cả các giá trị đã được sử dụng (allUsed == 1), đặt lại tất cả giá trị trong mảng usedValues thành 0 để tái sử dụng chúng.
* Sử dụng một vòng lặp do-while để tạo ra một giá trị ngẫu nhiên (randomValue) trong khoảng từ 1 đến MAX\_VALUES.
* Kiểm tra xem giá trị đã được sử dụng chưa. Nếu đã sử dụng, tiếp tục lặp để chọn một giá trị mới.
* Đánh dấu giá trị đã được sử dụng trong mảng usedValues.
* Trả về giá trị ngẫu nhiên không trùng lặp.

**Vẽ các cạnh nối đỉnh sau khi random**



* Gọi hàm readRandomGraph để đọc thông tin của đồ thị ngẫu nhiên từ tệp.
* Gọi hàm Dijkstra để tìm đường đi ngắn nhất trong đồ thị.

### **3.12Chức năng Exit**



# CHƯƠNG 3: KẾT LUẬN VÀ KẾ HOẠCH TƯƠNG LAI

1. **Kết luận**

Công Cụ Quan Trọng: Dù có nhược điểm, thuật toán Dijkstra vẫn là một công cụ quan trọng và được ứng dụng rộng rãi trong thực tế. Tích Hợp Cùng Các Nghiên Cứu Mới: Dijkstra vẫn giữ vững vị thế của mình trong cộng đồng lập trình và tiếp tục được tích hợp cùng các nghiên cứu mới về tối ưu hóa đường đi.

Bài báo cáo đã cung cấp cái nhìn tổng quan về nguyên tắc hoạt động, ưu điểm và nhược điểm của thuật toán Dijkstra. Mặc dù không phải là lựa chọn hoàn hảo trong mọi tình huống, nhưng sự hiệu quả và tính ổn định của nó làm cho Dijkstra trở thành một công cụ quan trọng trong giải quyết vấn đề đường đi ngắn nhất. Sự cân nhắc giữa ưu và nhược điểm cũng giúp xác định sự ứng dụng thích hợp của thuật toán trong các tình huống cụ thể.

1. **Kế hoạch tương lai**
2. Tối Ưu Hóa và Tăng Hiệu Suất

Tập trung vào nghiên cứu và phát triển các biến thể của thuật toán Dijkstra để tối ưu hóa độ phức tạp và tăng cường hiệu suất trong các trường hợp đặc biệt.

1. Xử Lý Đồ Thị Có Trọng Số Âm

Nghiên cứu và thiết kế giải pháp để xử lý trọng số âm trong đồ thị, có thể thông qua sự kết hợp với các thuật toán khác hoặc phương pháp sáng tạo mới.

1. Tích Hợp Trí Tuệ Nhân Tạo

Kết hợp thuật toán Dijkstra với kỹ thuật trí tuệ nhân tạo, như học máy, để tối ưu hóa quá trình lựa chọn đường đi và dự đoán tình hình đường đi trong tương lai.

1. Đối Mặt với Đồ Thị Lớn

Nghiên cứu cách xử lý đồ thị lớn một cách hiệu quả, có thể thông qua các kỹ thuật phân tách và tối ưu hóa lưu trữ thông tin đồ thị.

1. Phát Triển Ứng Dụng Thực Tế

Áp dụng thuật toán Dijkstra trong các ứng dụng thực tế như hệ thống thông tin giao thông thông minh, quản lý mạng di động, và điều khiển robot tự động.

1. Nghiên Cứu Trong Cộng Đồng Lập Trình

Tham gia vào cộng đồng lập trình và tiếp tục theo dõi các tiến triển và nghiên cứu mới liên quan đến thuật toán Dijkstra và các biến thể của nó.

1. Phát Triển Thư Viện và Công Cụ

Đóng góp vào việc phát triển thư viện và công cụ hỗ trợ cho triển khai thuật toán Dijkstra, làm cho nó trở nên dễ sử dụng và tích hợp vào các dự án khác.

1. Hợp Tác Nghiên Cứu

Hợp tác với các tổ chức nghiên cứu và ngành công nghiệp để áp dụng thuật toán Dijkstra vào các vấn đề thực tế và giải quyết các thách thức cụ thể.

1. Giáo Dục và Đào Tạo

Tham gia vào hoạt động giáo dục và đào tạo để chia sẻ kiến thức và kinh nghiệm về thuật toán Dijkstra, góp phần vào việc đào tạo thế hệ mới lập trình viên và nhà nghiên cứu.

# TÀI LIỆU THAM KHẢO

[1] Bài giảng Kỹ thuật đồ họa, Đoàn Vũ Thịnh, 2019

[2] Hướng dẫn giải chi tiết và lập trình Kỹ thuật đồ hoạ, Đoàn Vũ Thịnh, 2021

[3] Computer Graphics, C Version (2nd Edition), Donald Hearn, 1996

[4] Bài giảng cấu trúc dữ liệu và giải thuật, Nguyễn Đức Thuần, 2023

[5] Giải thuật và lập trình, Lê Minh Hoàng, 2006