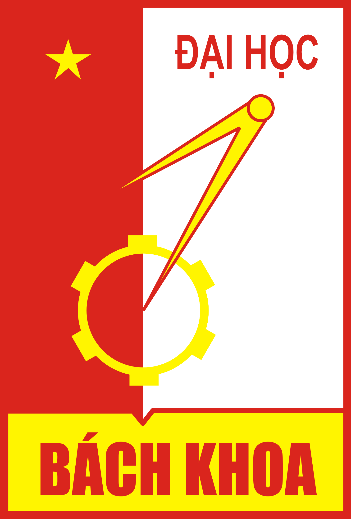
**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

****

**BÁO CÁO PROJECT I**

**ĐỀ TÀI: TRỰC QUAN HÓA CÁC THUẬT TOÁN ĐỒ THỊ**

Giảng viên hướng dẫn: PGS.TS. Cao Tuấn Dũng

Nhóm sinh viên thực hiện:

Nguyễn Thành Long - CNTT.09 - 20173245

Nguyễn Duy Hoài Lâm - CNTT.09 - 20173225

Nguyễn Đình Kiên - CNTT.11 - 20173211

**Giới thiệu**

**Mô tả bài toán**

Lý thuyết đồ thị và những thuật toán đồ thị là những kiến thức quan trọng khi học Công nghệ thông tin và có ứng dụng rất lớn trong quá trình phát triển của Công nghệ thông tin. Tuy nhiên, đây là một lĩnh vực rộng và phức tạp. Việc hiểu và cài đặt tốt các thuật toán đó đòi hỏi thời gian và công sức rất lớn. Hiện nay việc truyền đạt các thuật toán này gặp nhiều khó khăn do: Các thuật toán đó khó hình dung, việc tổ chức dữ liệu cho nó cũng phức tạp, thời gian giảng dạy trên lớp có hạn, tài liệu tham khảo có thể tự đọc, tự học vẫn còn ít …

Trong đề tài này, bọn em xây dựng một chương trình nhằm trực quan hóa hoạt động của 3 thuật toán: DFS, BFS, Dijicktra.

Với mục đích người xem có thể dễ dàng nắm bắt được cách thức hoạt động của các thuật toán đó mà không cần phải nắm rõ phần code nằm bên dưới. Từ đó tạo nên sự hứng thú học tập và tìm hiểu sâu hơn nữa.

**Yêu cầu đặt ra cho sản phẩm:**

* Tạo đồ thị bằng các sử dụng chuột:
  + Click chuột tạo các nút
  + Kéo thả chuột tạo cạnh
  + Thêm trọng số cho các cạnh…
* Khả năng chỉnh sửa đồ thị:
  + Có thể di chuyển các nút
  + Có thể xóa các nút
  + Có thể thay đổi nút bắt đầu
  + Có thể thay đổi trọng số của các cạnh
* Lựa chọn thuật toán chạy: có thể lựa chọn các thuật toán trên menu
* Lựa chọn cách thức chạy thuật toán
  + Chạy tự động
  + Chạy từng bước bằng cách nhấn nút next
* Mô tả từng bước chạy
  + Dùng bảng chi tiết các thông số của đồ thị
  + Dùng lời chỉ dẫn text
* Lưu và Mở đồ thi
  + Save đồ thị đang có trên màn hình dưới dạng file txt
  + Open file .txt đã tạo trước đó để tiết kiệm thời gian
* Thay đổi tốc độ chạy thuật toán khi chạy tự động
  + Dùng thanh trượt (thời gian tính bằng mili giây)

**MỤC LỤC**

[CHƯƠNG 1. KIẾN THỨC NỀN TẢNG 5](#_Toc28114277)

[1.1. Thuật toán 5](#_Toc28114278)

[1.1.1. Dijkstra 5](#_Toc28114279)

[1.1.2. Breadth First Search (BFS) 6](#_Toc28114280)

[1.1.3. Depth First Search (DFS) 6](#_Toc28114281)

[1.2. Cấu trúc dữ liệu 7](#_Toc28114282)

[1.2.1. Mảng 7](#_Toc28114283)

[1.2.2. Ngăn xếp (Stack) 8](#_Toc28114284)

[1.2.3. Hàng đợi (Queue) 9](#_Toc28114285)

[1.3. Ngôn ngữ lập trình 11](#_Toc28114286)

[1.3.1. Java 11](#_Toc28114287)

[1.4. Các kỹ thuật lập trình 12](#_Toc28114288)

[1.4.1. Hướng đối tượng (OOP - Object Oriented Programming) 12](#_Toc28114289)

[CHƯƠNG 2. GIẢI QUYẾT VẤN ĐỀ 13](#_Toc28114290)

[2.1. Tổng quan 13](#_Toc28114291)

[2.2. Thiết kế các chức năng chính 14](#_Toc28114292)

[2.2.1. Tạo đồ thị 14](#_Toc28114293)

[2.2.2. Sửa đồ thị 15](#_Toc28114294)

[2.2.3. Liên kết thuật toán 20](#_Toc28114295)

[2.2.4. Chạy từng bước 24](#_Toc28114296)

[2.2.5. Hướng dẫn 25](#_Toc28114297)

[2.3. Thiết kế cấu trúc, tổ chức của dữ liệu 25](#_Toc28114298)

[2.3.1. Mảng 25](#_Toc28114299)

[2.3.2. Stack và Queue 26](#_Toc28114300)

[2.4. Giải pháp thực hiện các chức năng khó, hay và quan trọng 26](#_Toc28114301)

[2.4.1. Chức năng save/open 26](#_Toc28114302)

[2.4.2. Chức năng hiển thị hướng dẫn 30](#_Toc28114303)

[2.5. Giải pháp thực hiện hiệu ứng 34](#_Toc28114304)

[2.5.1. Canvas 34](#_Toc28114305)

[2.5.2. Paint 35](#_Toc28114306)

[2.5.3. Update 36](#_Toc28114307)

[CHƯƠNG 3. KẾT QUẢ - DEMO – ĐÁNH GIÁ 37](#_Toc28114308)

[3.1. Minh họa các chức năng chương trình 37](#_Toc28114309)

[3.1.1. Giao diện phần mềm 37](#_Toc28114310)

[3.1.2. Chức năng tạo mới đồ thị 37](#_Toc28114313)

[3.1.3. Chức năng lưu đồ thị đã tạo 38](#_Toc28114314)

[3.1.4. Chức năng mở đồ thị đã lưu 39](#_Toc28114315)

[3.1.5. Chức năng thay đổi ngôn ngữ 40](#_Toc28114316)

[3.2. Demo theo một vài kịch bản 41](#_Toc28114317)

[3.2.1. Thuật toán Dijkstra 41](#_Toc28114318)

[3.2.2. Thuật toán BFS 43](#_Toc28114319)

[3.2.3. Thuật toán DFS 44](#_Toc28114320)

[3.3. Đánh giá mức độ hoàn thành 45](#_Toc28114321)

[3.3.1. Kiểm thử 45](#_Toc28114322)

[3.3.2. Đánh giá 49](#_Toc28114324)

[CHƯƠNG 4. MÔ TẢ KẾ HOẠCH THỰC HIỆN PROJECT VÀ ĐÓNG GÓP CỦA CÁC THÀNH VIÊN 50](#_Toc28114325)

[4.1. Kế hoạch thực hiện 50](#_Toc28114326)

[Tổng thời gian dự án : 15 Tuần (từ tuần 3 🡪 tuần 18) 50](#_Toc28114327)

[4.1.1. Tuần 3 🡪 5 50](#_Toc28114328)

[4.1.2. Tuần 5 🡪 9 50](#_Toc28114329)

[4.1.3. Tuần 10 🡪 15 50](#_Toc28114330)

[4.1.4. Tuần 16 🡪 18 50](#_Toc28114331)

[4.2. Đóng góp của các thành viên 50](#_Toc28114332)

[4.2.1. Thành viên Nguyễn Thành Long 50](#_Toc28114333)

[4.2.2. Thành viên Nguyễn Duy Hoài Lâm 51](#_Toc28114334)

[4.3. Tài liệu tham khảo 51](#_Toc28114335)

1. KIẾN THỨC NỀN TẢNG
   1. Thuật toán
      1. Dijkstra
         1. Giới thiệu

- Cho một đồ thị và một đỉnh gốc trên đồ thị, tìm đường đi ngắn nhất từ đỉnh gốc tới tất cả các đỉnh còn lại của đồ thị đã cho.

- Thuật toán dijkstra rất giống với thuật toán Prim’s cho cây con nhỏ nhất. Tương tự như vậy chúng ta sinh ra những đường đi nhỏ nhất với một đỉnh gốc cho trước. Chúng ta duy trì hai tập hợp, một chứa các đỉnh bao gồm trong đường đi ngắn nhất, tập hợp thứ 2 chứa các đỉnh vẫn chưa bao gồm trong đường đi ngắn nhất.

- Ở tất cả các bước của thuật toán chúng ta tìm một đỉnh (trong tập hợp thứ 2 chưa được bao gồm bởi đường đi ngắn nhất) và có một khoảng cách ngắn nhất tới đỉnh gốc.

* + - 1. Các bước tạo thuật toán

Bước 1:

- Tạo ra một tập hợp A (tập hợp đường đi cây ngắn nhất) nó sẽ lưu vết các đỉnh bên trong đường đi cây ngắn nhất. Khoảng cách tối thiểu từ gốc sẽ được tính và hoàn tất, khi khởi tạo tập hợp này rỗng.

Bước 2:

- Gán một giá trị khoảng cách cho tất cả các đỉnh trong đồ thị nhập vào. Khởi tạo tất cả các giá trị khoảng cách là vô cực. Gán giá trị khoảng cách bằng 0 cho đỉnh gốc và đó là đỉnh đầu tiên được chọn.

Bước 3: Trong khi tập hợp A chưa bao gồm tất cả các đỉnh

- Chọn một đỉnh x mà không có trong A mà giá trị khoảng cách nhỏ nhất.

- Cho đỉnh x vào trong A

- Cập nhật lại giá trị khoảng cách của tất cả các đỉnh liền kề với đỉnh x. Để cập nhật các giá trị khoảng cách, lặp lại qua tất cả các đỉnh liền kề. Đối với mỗi đỉnh liền kề y, nếu tổng giá trị khoảng cách của x (từ gốc) và giá trị của các cạnh x-y mà nhỏ hơn giá trị khoảng cách của y, sau đó cập giá trị khoảng cách của v.

* + - 1. Độ phức tạp thuật toán

Độ phức tạp thuật toán là O(V^2), nếu như đồ thị nhập vào biểu diễn dưới dạng danh sách kề thì nó có thể giảm còn O(E log V) với sự hỗ trợ của đống nhị phân

* + 1. Breadth First Search (BFS)
       1. Giới thiệu

Tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong  đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt nhìn các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt nhìn trước các đỉnh kề với nó mà chưa được quan sát trước đó và lặp lại. Xem thêm thuật toán tìm kiếm theo chiều sâu, trong đó cũng sử dụng 2 thao tác trên nhưng có trình tự quan sát các đỉnh khác với thuật toán tìm kiếm theo chiều rộng.

* + - 1. Các bước thuật toán

Từ khái niệm trên thuật toán được miêu tả các bước như sau:

1.Chèn đỉnh gốc vào hàng đợi (đang hướng tới)

2.Lấy ra đỉnh đầu tiên trong hàng đợi và quan sát nó

\* Nếu đỉnh này chính là đỉnh đích, dừng quá trình tìm kiếm và trả về kết quả.

\* Nếu không phải thì chèn tất cả các đỉnh kề với đỉnh vừa thăm nhưng chưa được quan sát trước đó vào hàng đợi.

3.Nếu hàng đợi là rỗng, thì tất cả các đỉnh có thể đến được đều đã được quan sát – dừng việc tìm kiếm và trả về "không thấy".

4.Nếu hàng đợi không rỗng thì quay về bước 2.

* + - 1. Độ phức tạp thuật toán

Độ phức tạp thời gian: O(|V|+|E|) với |V| là số đỉnh của đồ thị, |E| là số cạnh

Độ phức tạp dữ liệu: O(|V|)

* + 1. Depth First Search (DFS)
       1. Giới thiệu

Giải thuật sử dụng stack (ngăn xếp) để ghi nhớ đỉnh liền kề để bắt đầu việc tìm kiếm khi không gặp được đỉnh liền kề trong bất kỳ vòng lặp nào. Giải thuật tiếp tục cho tới khi gặp được đỉnh cần tìm hoặc tới một nút không có con. Khi đó giải thuật quay lui về đỉnh vừa mới tìm kiếm ở bước trước.

* + - 1. Các bước thuật toán

Bước 1:

* Duyệt tiếp tới đỉnh liền kề mà chưa được duyệt (đỉnh màu White). Đánh dấu đỉnh mà đã được thăm nhưng chưa duyệt xong bằng màu Gray. Hiển thị đỉnh đó và đẩy vào trong một ngăn xếp (stack).

Bước 2:

* Nếu không tìm thấy đỉnh liền kề, thì lấy một đỉnh từ trong ngăn xếp (thao tác pop up).
* (Giải thuật sẽ lấy tất cả các đỉnh từ trong ngăn xếp mà không có các đỉnh liền kề nào).

Bước 3:

* Lặp lại các bước 1 và 2 cho tới khi ngăn xếp là trống.
  + - 1. Độ phức tạp của DFS

Thuật toán thăm mỗi đỉnh v đúng một lần, do đó tổng thời gian thăm đỉnh là Θ(|V|).

Với mỗi đỉnh v duyệt qua tất cả các đỉnh kề, với mỗi đỉnh kề thực hiện thao tác với thời gian hằng số.

Do đó việc duyệt qua tất cả các đỉnh mất thời gian là: Θ(|E|).

Tổng cộng: Θ(|E|)+ Θ(|V|)= Θ(|E|+|V|), hay Θ(|V| 2 ).

* 1. Cấu trúc dữ liệu
     1. Mảng
        1. Mảng 1 chiều

Sử dụng mảng 1 chiều để lưu trữ những thành phần tập hợp đơn cấu thành như : các nút (nut[]), màu nút (colornut[]), khoảng cách giữa đỉnh đầu đến nút hiện tại (kc[]), khoảng cách cuối cùng nhỏ nhất từ nút đầu đến nút hiện tại (kcCuoi[])

Mảng 1 chiều là phương án tối ưu cho việc truy xuất nhanh chóng các phần tử trong mảng một cách trực tiếp qua chỉ số mà không cần phải duyệt qua từng phần tử như danh sách liên kết

Nhược điểm của mảng 1 chiều là không thể tối ưu bộ nhớ như danh sách liên kết do các phần tử trong mảng được lưu trữ trong các ô nhớ liên tiếp, như vậy muốn chèn một phần tử vào vị trí của mảng sẽ phải thay đổi vị trí của các phần tử phía sau. Trái ngược với đó là danh sách liên kết, có thể dễ dàng tháo liên kết và chèn 1 phần tử dễ dàng.

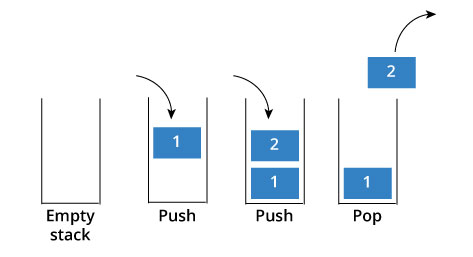
* + - 1. Mảng 2 chiều

Sử dụng mảng 2 chiều để lưu trữ những thành phần tập hợp có 2 yếu tố cấu thành như:

* Trọng số của cạnh (cạnh được xác định bởi nút 1 và nút 2)
* Mảng canh[][] ( true/false xác định sự tồn tại cạnh giữa 2 nút)
* Mảng 2 chiều là yếu tố quan trọng để tạo nên cấu trúc đồ thị: được cấu thành bởi các điểm ( là các đối tượng của lớp Point)
* startp[][] và endp[][] dùng để lưu tọa độ của điểm đầu và điểm cuối của 1 cạnh
* tbp[][] : điểm trung bình lưu trữ tọa độ giữa ở giữa mũi tên để ghi trọng số
* mtp[][]: mũi tên, xác định tọa độ để vẽ tam giác đầu mũi tên.
  + 1. Ngăn xếp (Stack)
       1. Giới thiệu

Ngăn xếp(Stack) là 1 cấu trúc dữ liệu trừu tượng được dùng phổ biên trong các ngôn ngữ lập trình hiện nay.

Do là một cấu trúc dữ liệu trừu tượng nên đặc trưng của ngăn xếp là chỉ cho phép truy xuất phần tử ở phía trên cùng của ngăn xếp thông qua nguyên tắc LIFO – Last In First Out (vào sau ra trước): Phần tử đặt vào đầu ngăn xếp sau sẽ được lấy tra trước



*hình minh họa stack*

* + - 1. Ứng dụng

Do cách thức truy xuất Last In First Out – Vào sau ra trước của Ngăn xếp nên nó được dùng để lưu trữ các nút trong giải thuật Depth First Search (DFS – Tìm kiếm theo chiều sâu).

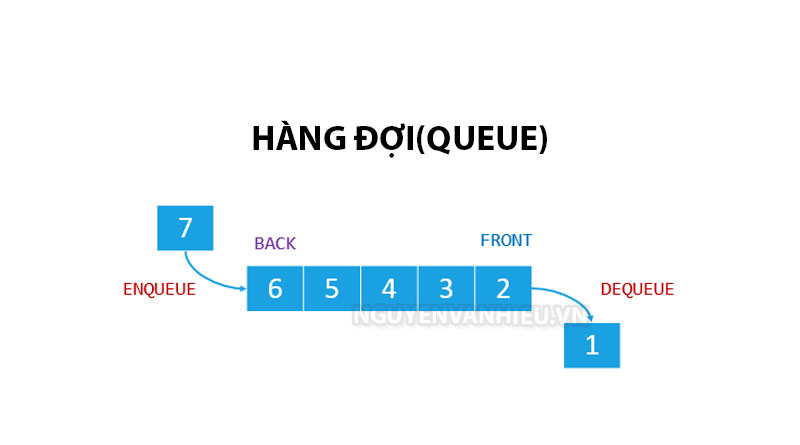
Do thuật toán DFS tìm kiếm nút sau nhất có thể sau đó thông qua các nút con và quay lui lại theo đúng nguyên tắc Last In First Out

* + 1. Hàng đợi (Queue)
       1. Giới thiệu

Hàng đợi giống như stack cũng là một kiểu cấu trúc dữ liệu trừu tượng. Nhưng nó hoạt động theo cách ngược lại so với stack

Khác với ngăn xếp (chỉ mở một đầu) thì hàng đợi mở cả hai đầu vào và ra. Một đầu luôn thêm các phần tử vào và một đầu xóa các phần tử ra

Hàng đợi hoạt động theo đúng cái tên của nó, các phần từ phải đợi để được duyệt theo nguyên tắc FIFO – First In First Out (Vào trước ra trước).

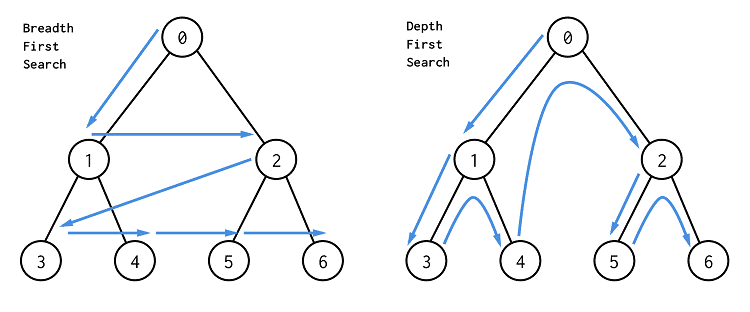
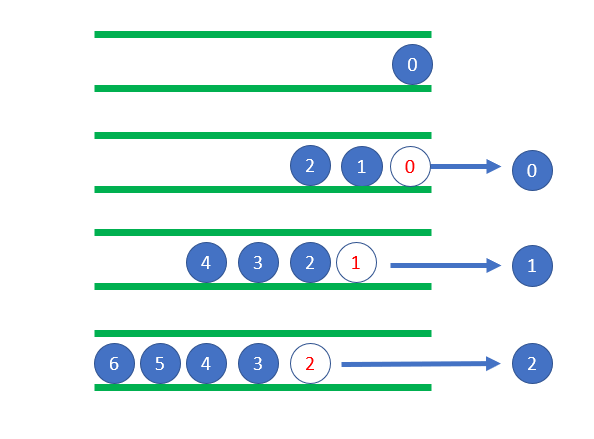


*hình minh họa hàng đợi (queue)*

* + - 1. Ứng dụng

Áp dụng tính chất vào trước ra trước của hàng đợi ta có thể dùng trong giải thuật BFS (Breath First Search) – Tìm kiếm theo chiều rộng

Với mục tiêu duyệt tất cả các đỉnh con rồi mới đến các đỉnh cháu...Bởi vậy nên các nút con sẽ được xếp vào một hàng đợi để duyệt lần lượt, với mỗi nút con khi được duyệt ra khỏi đầu hàng đợi thì sẽ thêm các nút cháu vào đuôi của hàng đợi và cứ tiếp tục cho đến khi duyệt xong hết.



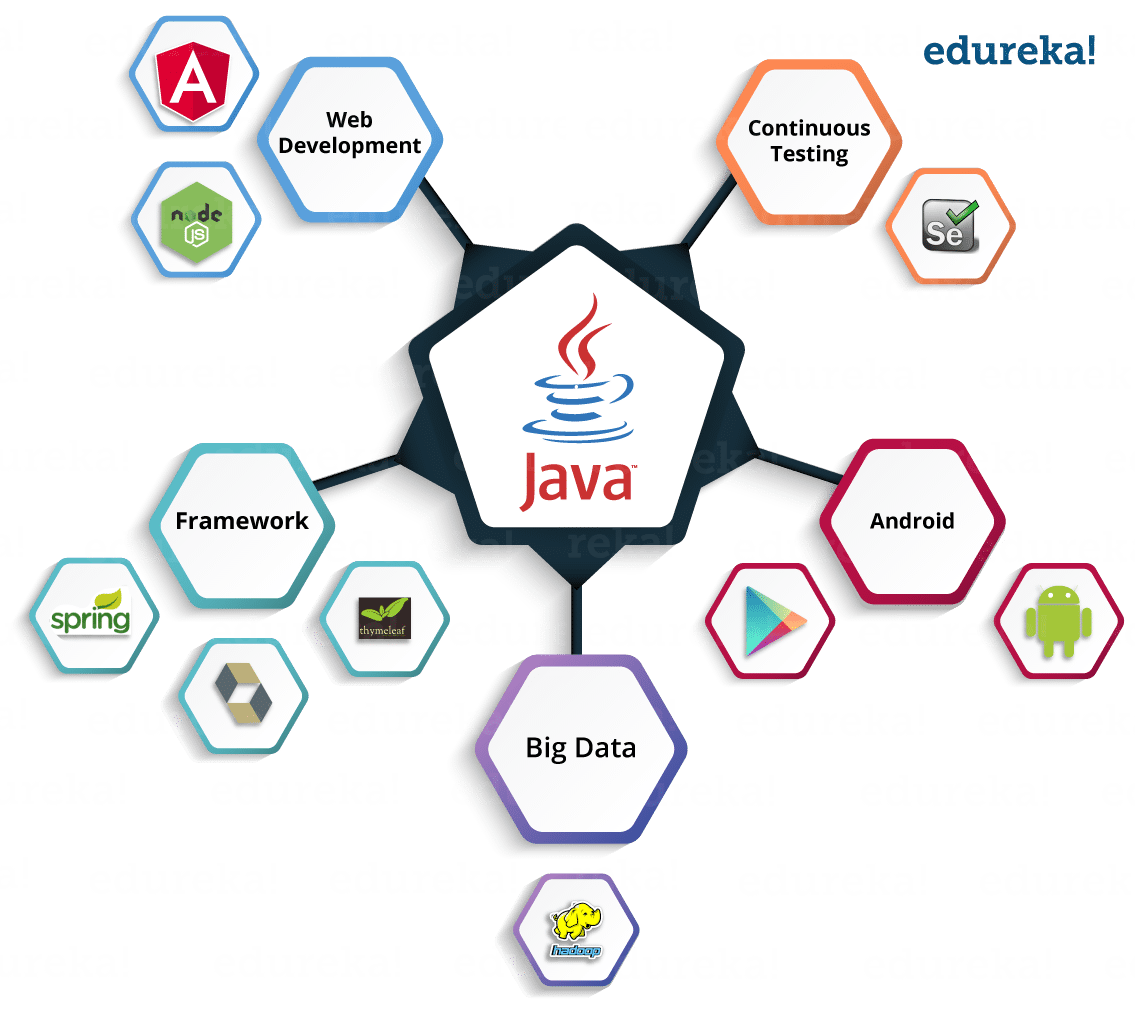
*Hình minh họa thứ tự duyệt BFS theo hàng đợi*

* 1. Ngôn ngữ lập trình
     1. Java
        1. Mục đích lựa chọn

Java là một ngôn ngữ lập trình thuần hướng đối tượng, chính vì lý do này nên chúng em đã chọn java làm ngôn ngữ để viết chương trình này với mục tiêu: “vừa củng cố kiến thức về kĩ thuật lập trình đã được học ở các kì trước và hỗ trợ mạnh mẽ cho môn học Lập Trình Hướng Đối Tượng mà chúng em học ở kỳ này”

Bên Cạnh đó java hiện nay không chỉ đơn thuần là một ngôn ngữ, mà nó đã trở thành một công nghệ, do có cộng đồng lập trình viên và nhà phát triển đông đảo, cùng với những ưu điểm vượt trội như tính bảo mật cao (các kiểu dữ liệu rõ dàng, cùng với cú pháp như các giới hạn truy cập rành mạch).

Hơn nữa java sở hữu một phạm vi thư viện rất rộng lớn và quan trọng nhất đó là khả năng chạy trên mọi môi trường (Write One Run Anywhere – Viết một lần dùng mọi nơi) của máy ảo JVM điều đó giúp khả năng bảo mật của java tăng lên trên mọi ngôn ngữ khác (khi máy ảo bị dính mã độc, thì chỉ mình máy ảo bị tấn công – còn hệ thống máy chủ không bị).



Chính vì những lý do trên nên nhóm em quyết định chọn java làm ngôn ngữ chính để triển khai trong học phần Project I này.

* + - 1. Thư việ sử dụng
         1. AWT(Abtract Window Toolkit)

Java AWT (Abtract Window Toolkit) là một API để phát triển các ứng dụng dựa trên GUI (Graphic User Interface – Giao diện người dùng đồ họa) trong java

Java.awt cung cấp các lớp hữu dụng cho API như TextFile, Text Area, RadioButton, CheckBox, Choice, List…. Để có thể dễ dàng xây dựng giao diện đồ họa

* + - * 1. Swing

Là một bộ công cụ tiện ích GUI cho java, nó đã được phát triển nâng tính năng hơn về các thành phần đồ họa so với AWT nhưng vẫn kết thừa các đặc tính hữu dụng của AWT.

Các lớp được sử dụng : JoptionPane (để xuất nội dung hộp thoại), Jtable( Tạo bảng), Jslider ( tạo thanh trượt), JscrollPane (tạo thanh cuộn) ….

* 1. Các kỹ thuật lập trình
     1. Hướng đối tượng (OOP - Object Oriented Programming)
        1. Tính kế thừa

Nhắc đến lập trình hướng đối tượng thì không thể không thể kể đến tính kế thừa, đây là một trong 4 đặc trưng cốt yếu của lập trình hướng đối tượng.

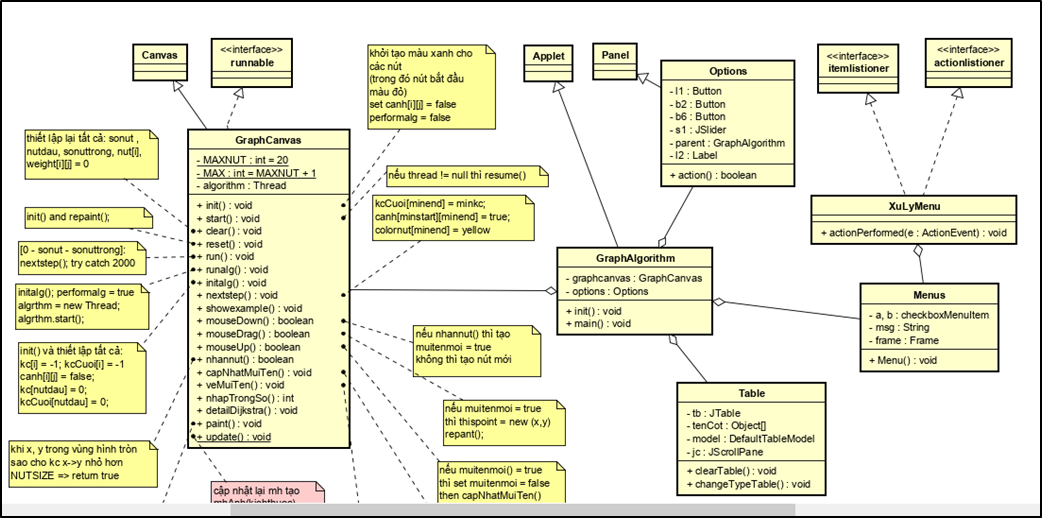
Trong project này chúng em sử dụng tính kế thừa trong việc sử dụng lại các lớp sẵn có để tạo ra những lớp mới đầy đủ chức năng hơn như việc kế thừa lại các lớp Applet, Pannel..

* + - 1. Tính đóng gói

Java sử dụng phương thức động nên mỗi khi gọi một phương thức, java sẽ truy xuất dữ liệu tùy theo từng trường hợp, không làm ảnh hưởng đến các đối tượng thuộc lớp khác.

Các thành phần dữ liệu quan trọng được giới hạn truy cập là private để ngăn chặn quyền truy cập không hợp lệ từ bên ngoài

1. GIẢI QUYẾT VẤN ĐỀ
   1. Tổng quan

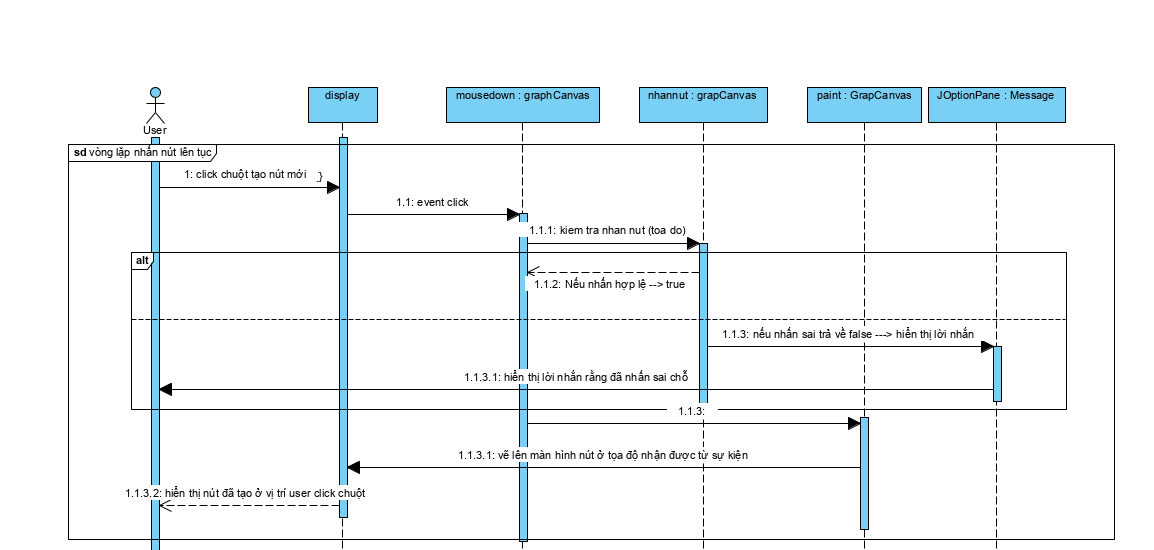
****

*Biểu đồ class tổng quát*

**- chương trình trực quan bao gồm 6 lớp chính:**

* Class GraphCanvas là lớp đồ họa chính:
* Thực hiện các chức năng vẽ,
* Tạo đồ thị đồ họa (các nút, mũi tên, trọng số…)
* Cập nhật đồ thị
* Xử lý sự kiện người dùng
* Xử lý các thuật toán
  + Class Option thực hiện các nhiệm vụ:
* Tạo các nút
* Tạo slider ( chỉnh tốc độ chạy)
* Xử lý sự kiện tác động nút từ người dùng
  + Class Menu thực hiện nhiệm vụ:
* Tạo thanh menu
* Tạo các danh sách scroll down
  + Class XuLyMenu thực hiện nhiệm vụ
* Kiểm soát lựa chọn của người dùng trên menu và phản hồi string trên console
  + Class Table thực hiện nhiệm vụ
* Tạo bảng và thanh cuộn chứa bảng
* Xử lý các thao tác xóa bảng, thay đổi cập nhật bảng.
  1. Thiết kế các chức năng chính
     1. Tạo đồ thị
        1. tạo các thành phần bằng tay

- Các nút: sử dụng cấu trúc mảng các đối tượng (Point) để xác định tọa độ các nút, được sinh ra do người dùng click chuột vào vùng đồ thị



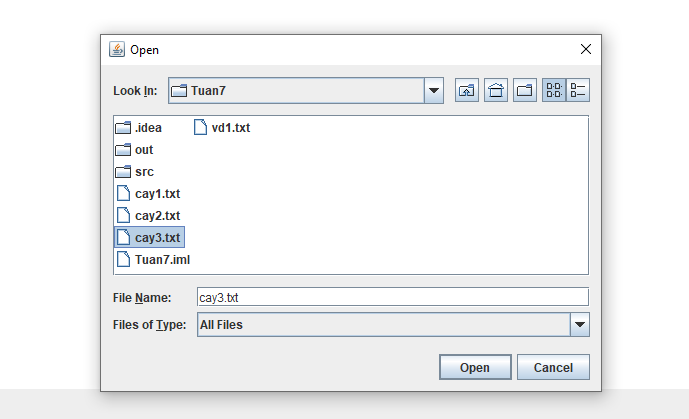
*Biểu đồ trình tự sự kiện người dùng tạo nút*

- Các mũi tên: tương tự như các nút, các mũi tên sinh ra từ sự kiện kéo chuột của người dùng, thay vì sự kiện click chuột.

- Các trọng số: giá trị trọng số của mũi tên sẽ được yêu cầu người dùng nhập vào mỗi lần kéo mũi tên

* + - 1. tạo đồ thị bằng file có sẵn

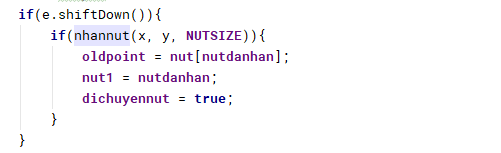
-Người dùng có thể sử dụng chức năng mở file có sẵn (file.txt)



- Phần chức năng lưu file sẽ được trình bày ở mục sau.

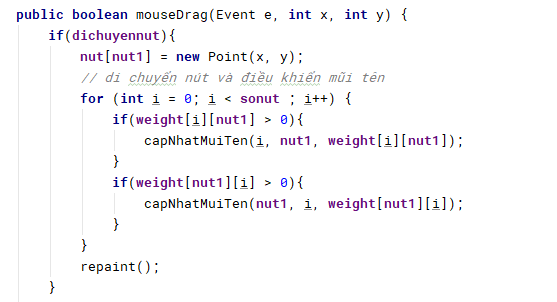
* + 1. Sửa đồ thị
       1. Di chuyển nút

Người dùng sử dụng tổ hợp phím Shift + left mouse tạo sự kiện kéo nút



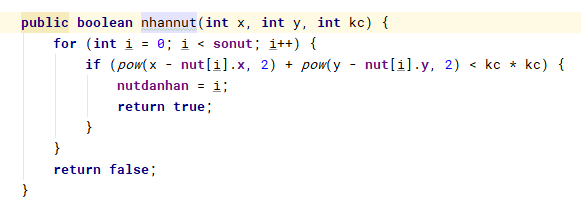
Ngay lập tức phương thức mouseDown() sẽ bắt lấy sự kiện click chuột và kiểm tra bằng cách truyền tọa độ x, y của chuột cho phương thức nhannut(x, y, NUTSIZE).

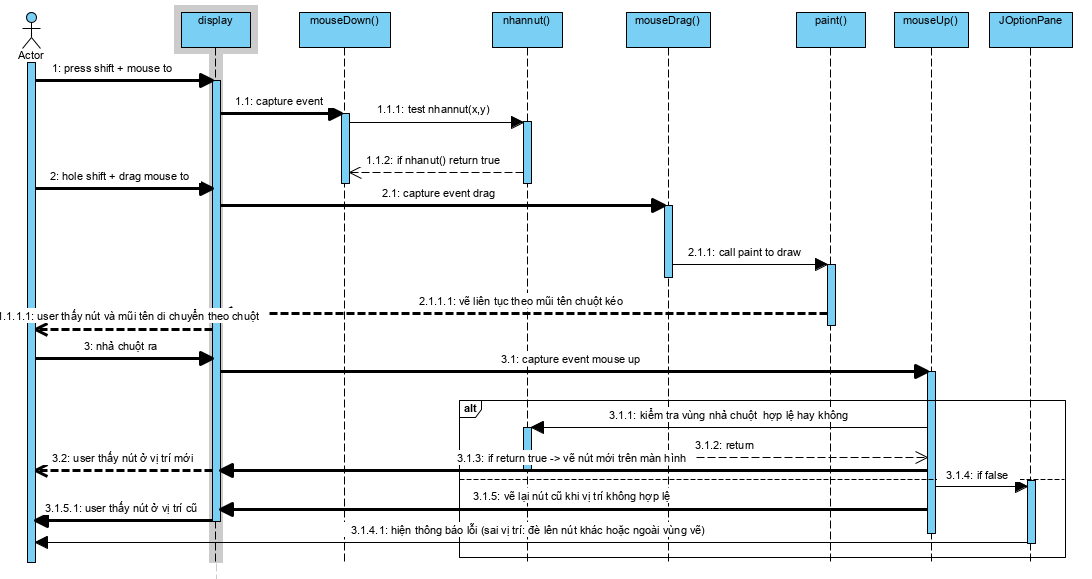
Nếu trả về true thì di chuyển



nhannut sẽ được gán bằng true

Tiếp theo phương thức mouseDrag sẽ bắt lấy sự kiện kéo chuột( xuất hiện sau khi click chuột lên nút). Và tiến hành thay đổi vị trí nút đồng thời thay đổi vị trí của mũi tên gắn với nút. Sau đó gọi đến repaint() để vẽ lại liên tục theo thời gian thực.



**

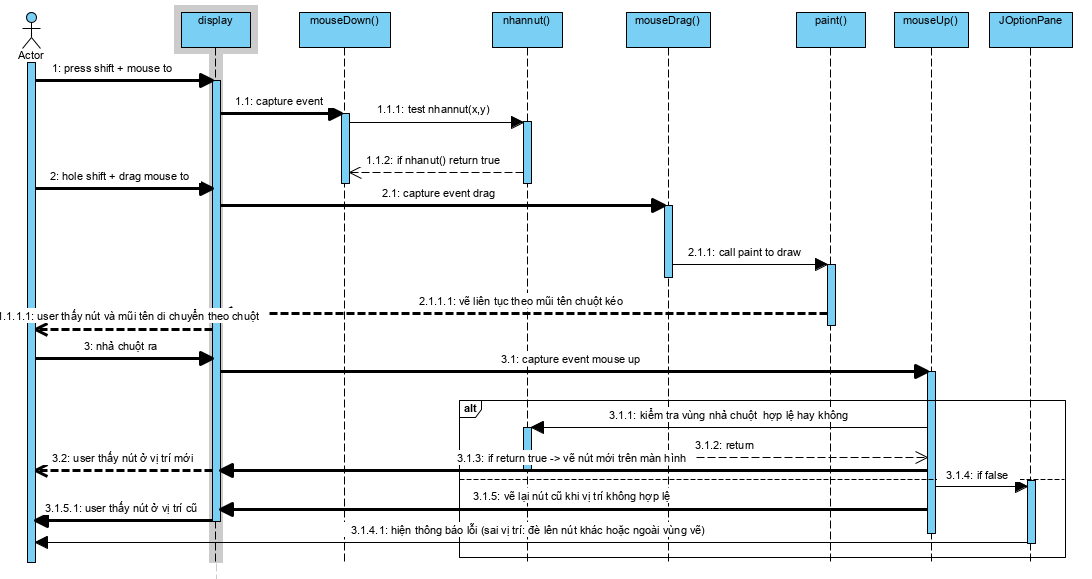
*Biểu đồ trình tự sự kiện người dùng thay đổi vị trí của nút*

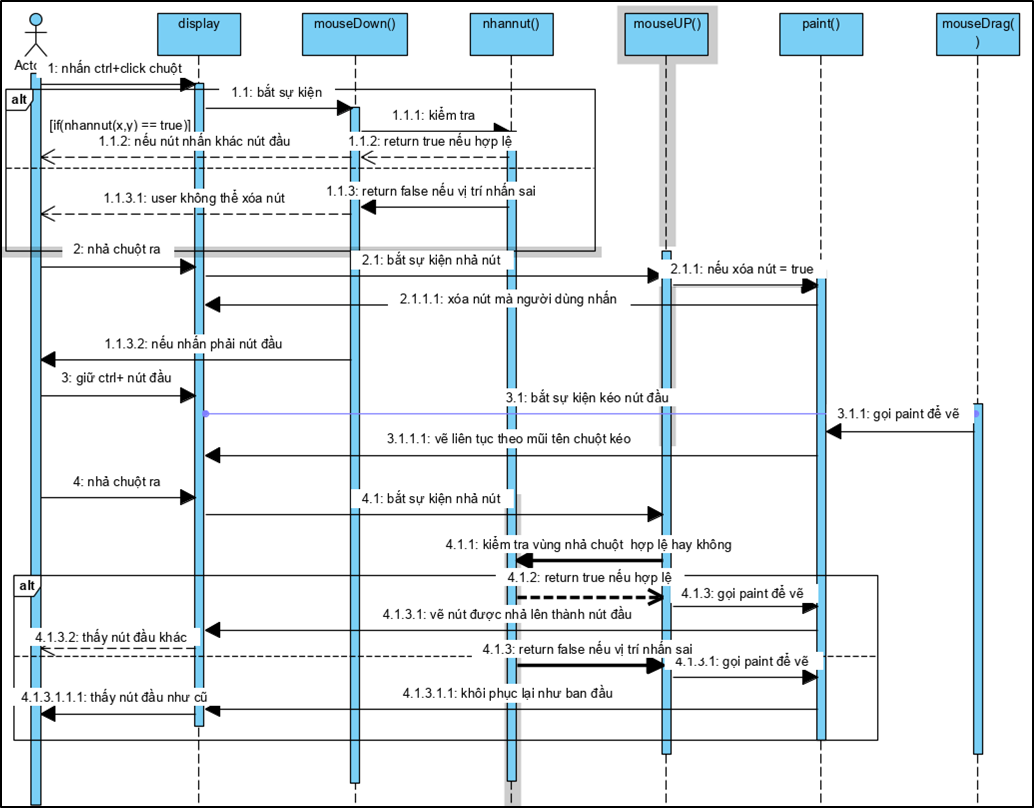
* + - 1. Xóa nút, di chuyển nút bắt đầu

Người dùng sử dụng tổ hợp phím Ctrl + left mouse để tạo sự kiện click chuột xóa nút

Sau khi nhận sự kiện ctrl+mouse phương thức mouseDown() sẽ kiểm tra sự kiện có hợp lệ (xóa nút) hay không bằng cách kiểm tra điều kiện trả về từ phương thức nhannut() (kiểm tra tọa độ click chuột có trong chu vi của nút hay không).

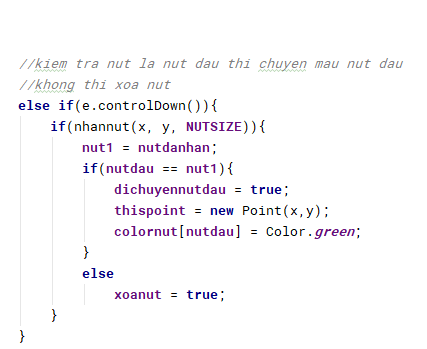
Nếu trả về true thì tiếp tục kiểm tra : nút được nhấn có bằng nút đầu hay không

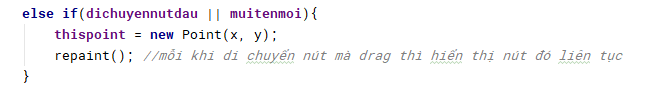
**



*Biểu đồ trình tự sự kiện xóa nút hoặc thay đổi nút đầu của người dùng*

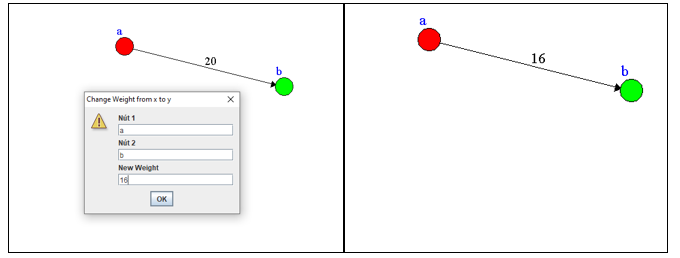
Nếu là nút đầu thì di chuyển nút đầu bằng cách drag (kéo). Phương thức bắt sự kiện kéo chuột mouseDrag() sẽ gọi tới phương thức vẽ paint() và vẽ liên tục nút trong quá trình thay đổi giống với sự kiện thay đổi vị trí nút.





* + - 1. Thay đổi trọng số nút

Sử dụng method changeWeight() để thay đổi trọng số giữa 2 nút.



Người dùng sẽ chọn chức năng change weight trên thanh menu (mục Graph) để mởi hộp thoại thay đổi trọng số.

* + 1. Liên kết thuật toán
       1. Thuật toán Djikstra

Sử dụng phương thức detailDijkstra(): để xác định nút đầu và nút cuối của từng bước

Phương thức detailDijkstra sẽ được gọi tới bởi phương thức paint(): phương thức paint sẽ được gọi bởi nextStep để vẽ lại từng bước.

for( int i = 0; i < sonut; i++)

for(int j = 0; j < sonut; j++)

if(weight[i][j] > 0)

detailDjkstra();

Phương thức paint sẽ duyệt lần lượt từng cặp nút, nếu nút nào có trọng số dương (2 nút có cạnh) thì gọi tới detailDijkstra() và truyền vào tọa độ nút i và j đang xét.

detailDijkstra() sẽ kiểm tra xem i và j có phải là trường hợp nút i đã xét mà nút j chưa xét hay không bằng điều kiện

if(kcCuoi[i] != -1 && kcCuoi[j] == -1)

Nếu thỏa mãn phương thức tiếp tục kiểm tra khoảng cách j đã xét chưa hay là nếu đã có thì nó phải lớn hơn hoặc bằng khoảng cách của nút i + trọng số giữa i và j thì mới cập nhật lại khoảng cách.

if(kc[j] != -1 || kc[j] >= kc[i] + weight[i][j]){

kc[j] = kc[i] + weight[i][j]; //cập nhật kc[j]

if(minkc == 0 || kc[j] < minkc){

minkc = kc[j]; //cập nhật lại minkc

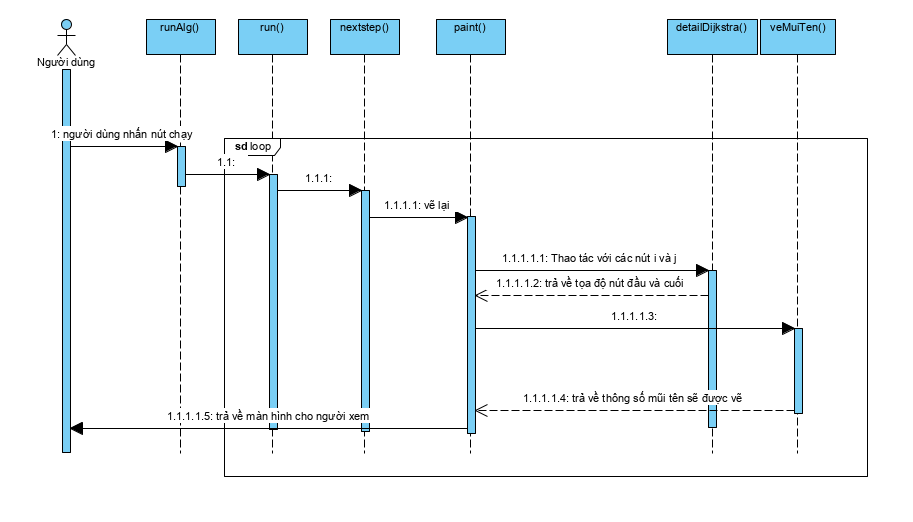
minstart = i;

minend = j;

}

}

Khoảng cách của j sau khi được cập nhật sẽ được so sánh với khoảng cách nhỏ nhất (minkc – ban đầu sẽ được gán = 0 ở mỗi lần gọi tới detailDijkstra()). Nếu như minkc = 0 có nghĩa là mới xét đến hàng xóm thứ 1 của nút i, còn nếu đã có minkc thì kiểm tra kc[j] có nhỏ hơn minkc không, nếu nhỏ hơn thì cập nhật lại minkc.



*Biều đồ trình tự quá trình chạy thuật toán*

Sau khi paint() đã gọi tới detailDijkstra() và tìm được minstart và minend, paint() sẽ gọi dến vemuiten() để tính toán tọa độ và vẽ mũi tên màu tương ứng.

Quá Trình lặp đi lặp lại cho đến khi thuật toán kết thúc.

* + - 1. Thuật toán DFS
* Sử dụng phương thức detailDFS(): để xác định nút đầu cuối của từng bước

Phương thức detailDFS sẽ được gọi tới bởi phương thức paint(): phương thức paint sẽ được gọi bởi nextStep để vẽ lại từng bước.

for( int i = 0; i < sonut; i++)

for(int j = 0; j < sonut; j++)

if(weight[i][j] > 0)

detailDFS();

detailDFS() sẽ kiểm tra xem i và j có phải là trường hợp nút i đã xét mà nút j chưa xét hay không bằng điều kiện

if(kcCuoi[i] != -1 && kcCuoi[j] == -1)

* Nếu thỏa mãn phương thức tiếp tục kiểm tra xem i có phải lá hay không. Nếu i là lá ta đánh dấu i đã thăm và đưa i vào stack.

if(checkla(i)) {  
 Visited[i] = true;  
}  
stack.push(i);

* Kiểm tra ngăn xếp có rỗng hay không xác định đỉnh liền kề đánh dấu màu đỏ đã thăm nhưng chưa duyệt xong. Hiển thị đỉnh đó đẩy vào stack. Nếu không thấy đỉnh liền kề thì lấy ra từ stack 1 đỉnh đánh dấu màu vàng. Lặp đi lặp lại đến khi stack rỗng.

while (!stack.empty()){  
 i = stack.pop();  
 if (!Visited[i]) {  
 Visited[i] = true;  
 VisitedList.add(i);  
 }  
  
 if (weight[i][j] > 0){  
 if (!Visited[j])  
 g.setColor(Color.*green*);

colornut[j] = Color.*red*;

stack.push(j);  
 minstart = i;  
 minend = j;  
}  
 }  
 while (!stack.empty())  
 stack.pop();  
 }

* + - 1. Thuật toán BFS
* Sử dụng phương thức detailBFS(): để xác định nút đầu cuối của từng bước

Phương thức detailBFS sẽ được gọi tới bởi phương thức paint(): phương thức paint sẽ được gọi bởi nextStep để vẽ lại từng bước.

for( int i = 0; i < sonut; i++)

for(int j = 0; j < sonut; j++)

if(weight[i][j] > 0)

detailBFS();

detailBFS() sẽ kiểm tra xem i và j có phải là trường hợp nút i đã xét mà nút j chưa xét hay không bằng điều kiện

if(kcCuoi[i] != -1 && kcCuoi[j] == -1)

* Nếu thỏa mãn phương thức ta đưa i vào hàng đợi và đánh dấu i đã thăm.

open.addLast(i);  
if(!Visited[i]) {  
 Visited[i] = true;

}

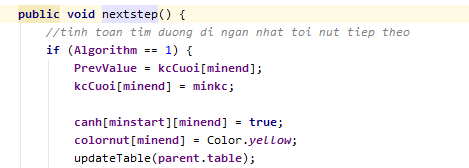
* Kiểm tra hàng đợi có rỗng hay không. Lấy ra đỉnh i đầu tiên trong hàng đợi đánh dấu vàng. Xét đỉnh j kề i nếu chưa thăm ta đánh dấu đã thăm đưa vào hàng đợi. Lặp đi lặp lại đến khi hàng đợi rỗng.

while(open.size() != 0) {  
 i = open.pollFirst();  
 if(weight[i][j] > 0){  
 if (!Visited[j]) {  
 g.setColor(Color.*green*);Visited[j] = true;  
 open.addLast(j);  
 colornut[j] = Color.*red*;  
 minstart = i;  
 minend = j;  
 }   
 }else  
 break;}

* + 1. Chạy từng bước
       1. Next step

Sử dụng phương thức nextStep() để phân chia mỗi bước của thuật toán theo sự thay đổi của khoảng cách cuổi của nút đã xét được gán bằng khoảng cách nhỏ nhất

Mỗi lần nextstep() được gọi thì sẽ xét màu vàng cho nút đã xét đồng thời gọi tới phương thức updateTable() để cập nhật bảng tương ứng.



Khi tìm được minend phương thức nextstep() sẽ gán kcCuoi[minend] = minkc để xác định nút minend đã xét và canh[minstart][minend] được gán là true để khi vẽ bằng vemuiten() sẽ tô màu đỏ. Còn nút sẽ tô màu vàng.

* + 1. Hướng dẫn

- Sử dụng 2 hình thức hướng dẫn trong khi chạy thuật toán đó là bảng và văn bản.

- Bảng sẽ được tạo nên trong panel trên bên trái của Jframe hiển thị các thông số đồ thị.

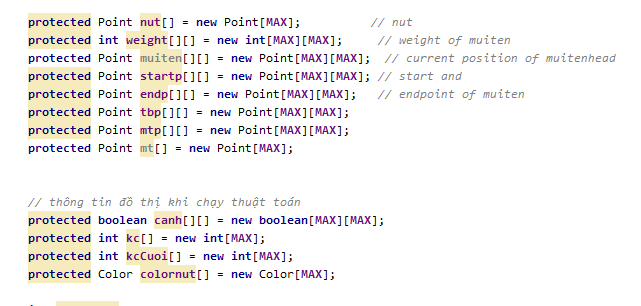
- Phần văn bản nằm bên dưới panel bảng là một phần JtextArea để hiển thị thông tin chi tiết mỗi bước thuật toán được chạy để người dùng có thể hiểu.

- Thông tin chi tiết về phần bảng và văn bản sẽ được giải thích rõ ở phần chức năng hay và khó.

* 1. Thiết kế cấu trúc, tổ chức của dữ liệu
     1. Mảng

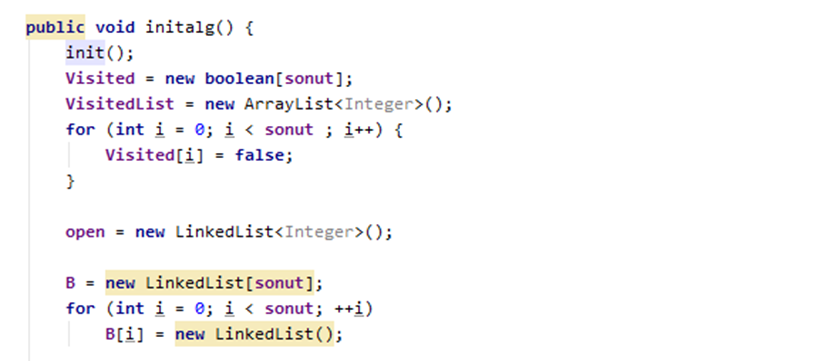
Mảng dùng để lưu trữ các thông tin của đồ thị như: số nút, tọa độ mũi tên giữa, khoảng cách, khoảng cách cuối, màu nút

Mảng 2 chiều được dùng để lưu trữ thông tin của: trọng số, cạnh, tọa độ 2 đầu mũi tên bắt đầu và kết thúc, Hướng x,y của mũi tên.



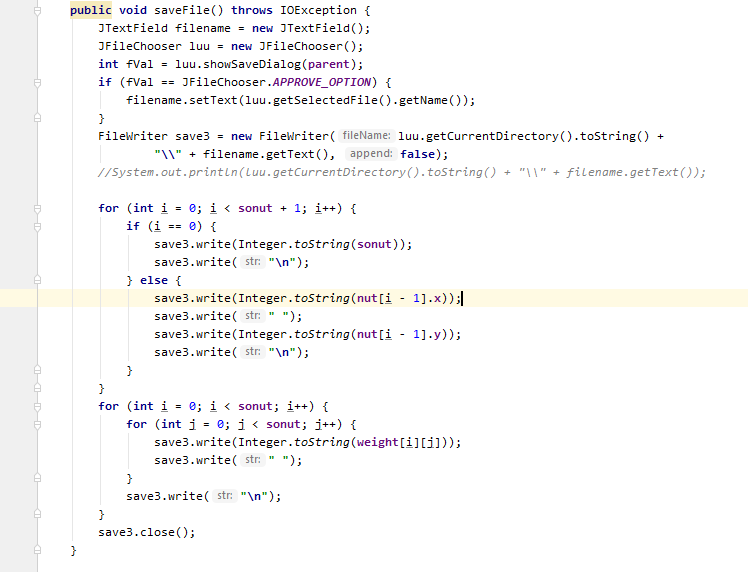
* + 1. Stack và Queue

Sử dụng hàng đợi và ngăn xếp dùng trong phương thức initalg() để khởi tạo thông tin cho các thuật toán DFS và BFS



* 1. Giải pháp thực hiện các chức năng khó, hay và quan trọng
     1. Chức năng save/open
        1. Save file

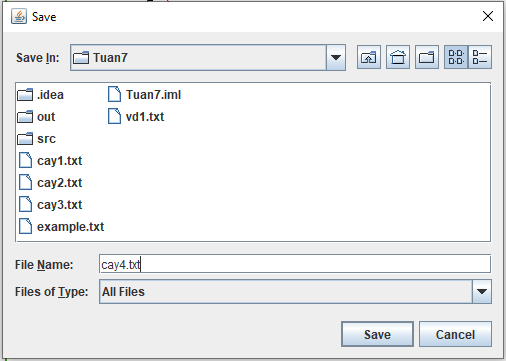
Sử dụng lớp JfileChooser của Java Swing để tạo đối tượng hiển thị hộp thoại lưu file

. 

- Sử dụng lớp FileWriter của package java.io để tạo đối tượng ghi file

- Sử dụng JtextField để tạo đối tượng filename lưu trữ tên file lấy được từ cửa sổ save file

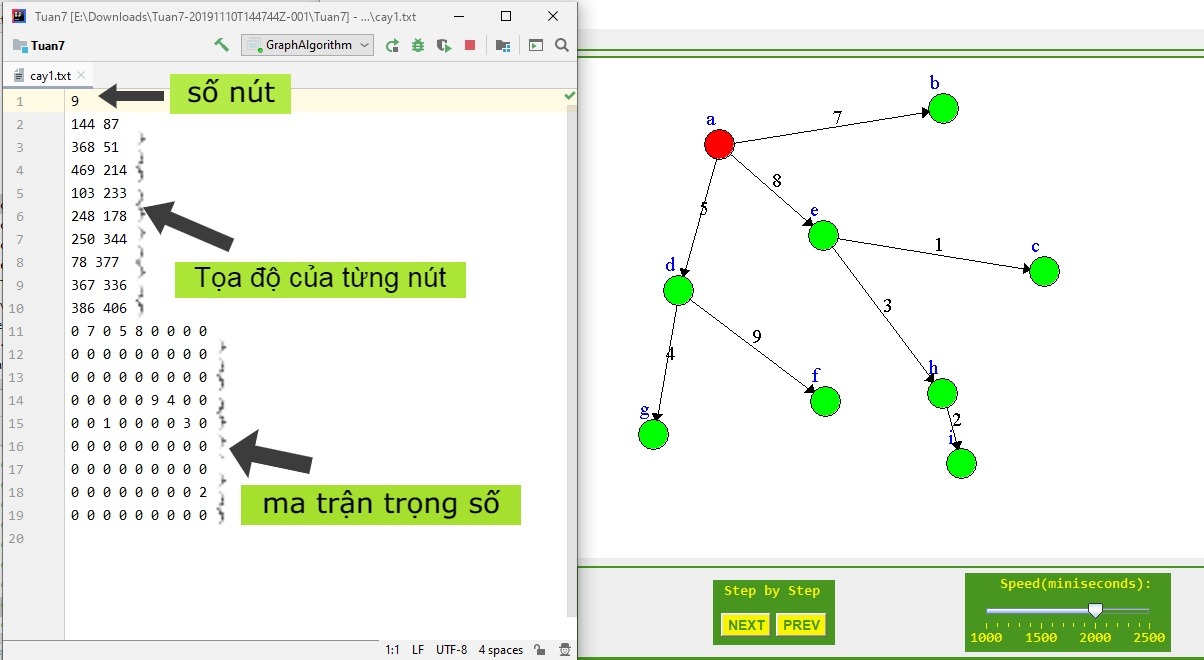
- Sau đó filename sẽ được truyền vào làm tham số cho đối tượng save3



Cơ chế lưu file sử dụng vòng lặp và điều kiện:

* Vòng lặp đầu tiên: đầu file sẽ lưu số nút, và tiếp theo sẽ lưu lần lượt tọa độ từng nút
* Vòng lặp thứ 2 sẽ lưu trọng số của các nút theo hàng

Cấu trúc tệp tin:



Tệp tin được cấu trúc theo thứ tự 3 phần:

* Phần 1 lưu số nút
* Phần 2 lưu tọa độ của nút
* Phần 3 lưu ma trận trọng số của các cặp nút
  + - 1. Open file

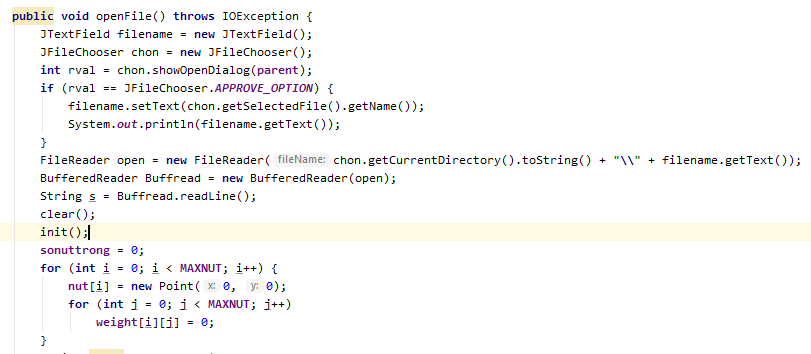
Chức năng mở file cũng sử dụng 2 lớp JtextField và JfileChooser của java swing như chức năng Save file

Sử dụng FileReader để tạo đối tượng open để mở file

Đối tượng filename dùng để lấy tên file từ hộp thoại mở file khi người dùng chọn file.

Sau đó đối tượng open sẽ được khởi tạo với tham số là chính giá trị đường dẫn đã được lưu trong filename.

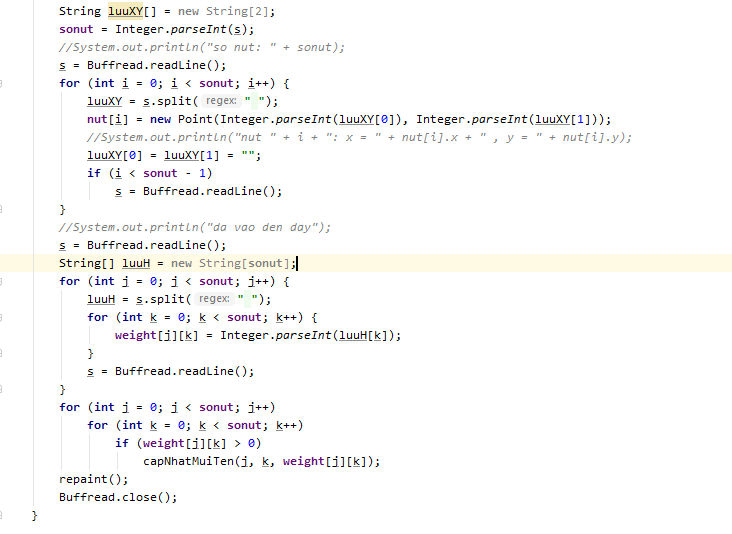
Sử dụng lớp BufferReader để đọc từng dòng trong open và lưu trong 1 string s (string này sẽ được sử dụng để phân tách từng phần của cấu trúc file )



Clear() và init() và vòng lặp lồng nhau tiếp theo dùng để khởi tạo thông số cho đồ thị mới sắp được đọc ra.

Dòng đầu tiên mà s lấy được sẽ được truyền vào sonut giống như cấu trúc file

Mảng chuỗi luuXY [2] dùng để lấy tọa độ từ chuỗi string s, luuXY[0] dùng để lấy giá trị x, luuXY[1] dùng để lưu giá trị y.



Tiếp theo vòng lặp đâu tiên truyền vào tọa độ cho các nút và đọc từng dòng.

Mảng String luuH [sonut] dùng để lưu các trọng số lấy được từ phần thứ 3 của file ( như đã trình bày ở phần save file)

Vòng lặp lồng nhau đầu tiên dùng để truyền giá trị trọng số weight[j][k], ở vòng lặp j thực hiện tách chuỗi trong dòng (split) và đọc từng dòng (Buffread.readLine()).

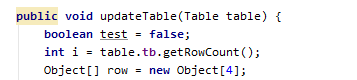
Vòng lặp lồng nhau thứ 2 dùng để cập nhật lại các mũi tên tương ứng với các trọng số và cuối cùng là repaint() để vẽ lại canvas.

* + 1. Chức năng hiển thị hướng dẫn
       1. Bảng thông số

Chức năng table dùng để hiển thị thông tin chi tiết đồ thị trong quá trình chạy thuật toán như tên các đỉnh, khoảng cách, khoảng cách cuối và đường đi ngắn nhất.

Table được thiết kế thành 1 lớp dùng để khởi tạo 1 đối tượng table trong lớp chính ( GraphAlgorithm) và gồm các phương thức thay đổi kiểu đồ thị để thích ứng với từng loại thuật toán.

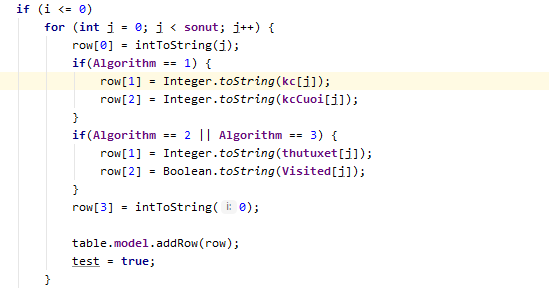
Thao tác cập nhật bảng thông số là chức năng khó nằm ở phương thức updateTable() trong class GraphCanvas.

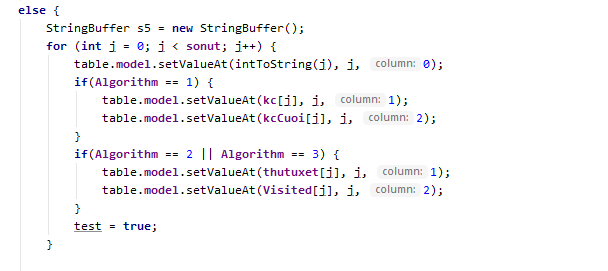


Phương thức này dùng biến i để lấy giá trị dòng cần cập nhật và một mảng Object[] row để gán giá trị ban đầu cho bảng.

Nếu i <= 0 (tức chưa có dòng trong bảng) thì sẽ gán các thông số ban đầu cho từng cột từ row[0] đến row[4]. và với row[1] và row[2] sẽ gán theo 2 trường hợp: thuật toán 1 (dijkstra) hoặc thuật toán 2,3 ( BFS hoặc DFS).

Ngược lại với i > 0 ( tức là bảng đã có hàng ) thì ta phải cập nhật cho từng cột.



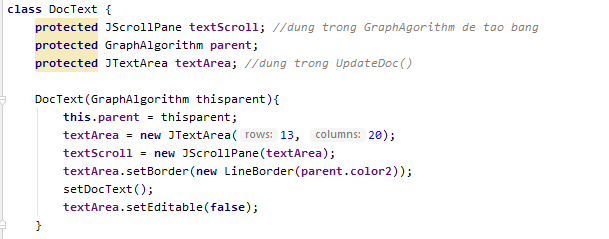


* + - 1. Text hướng hẫn

- sử dụng lớp DocText để tạo đối tượng vùng chỉ dẫn cho người dùng khi chạy thuật toán.

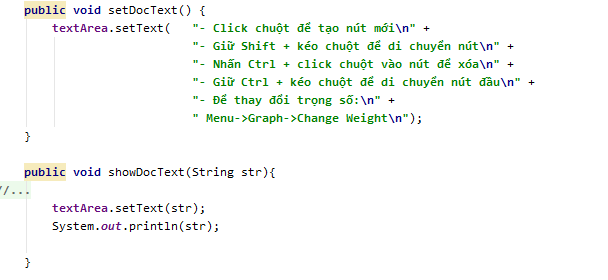
- lớp DocText đơn giản sẽ khởi tạo đối tượng textArea từ lớp JtextArea và một đối tượng textScroll từ lớp JscrollPane để chứa phần textArea bên trong có tác dụng tránh trường hợp phần chữ hiển thị vượt quá phần diện tích panel.

- lớp gồm có 2 phương thức:



+ setDocText(): dùng để hiển thị thông tin hướng dẫn khi người dùng mới bắt đầu sử dụng phần mêm.

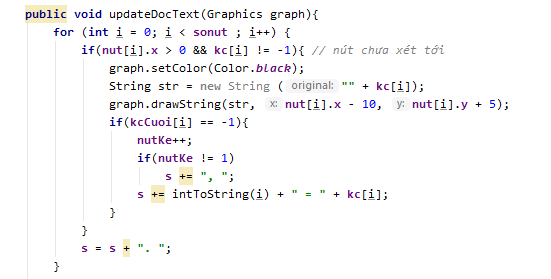
+ showDocText(String str): dùng để hiện thông tin text được lấy từ chuỗi truyền vào. Được sử dụng ở lớp updateDocText(Graph g) trong class GraphCanvas để cập nhật những thông tin mỗi bước thuật toán được chạy.



- Phương thức cập nhật updateDocText() trong class GraphCanvas:

+ phương thức sẽ cập nhật các trọng số khoảng cách của các nút nếu khoảng cách của các nút đã thay đổi và hiển thị nó ra ở chính giữa nút. Cùng với đó sẽ cập nhật chuỗi s lên DocText.

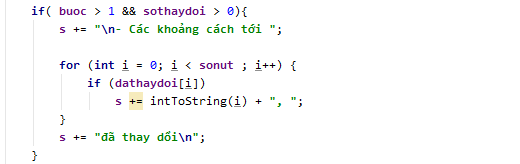
+ ở mỗi bước nextStep() thì docText sẽ được cập nhật thông qua các thông số:



nutKe( chỉ số nút hàng xóm có thể tới), buoc(chỉ số bước hiện tại), sothaydoi(chỉ số nút thay đổi khoảng cách).

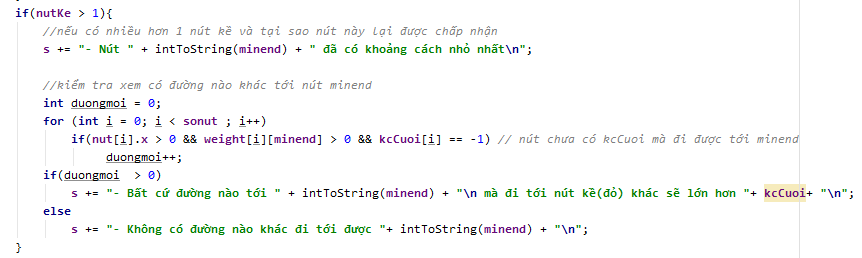
+ nếu số bước > 1 và số nút thay đổi sothaydoi > 0 thì:

thêm thông tin đã thay đổi vào trong chuỗi s.

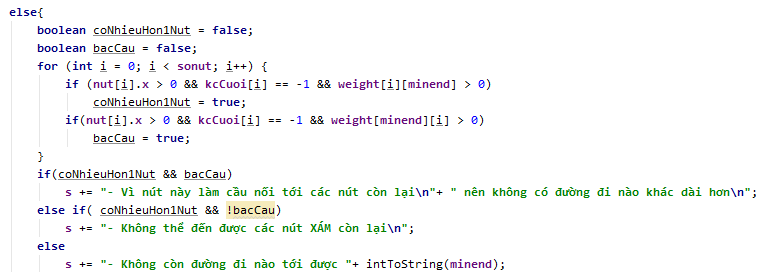


+ nếu số nút kề > 1 thì sẽ hiển thị thông tin tại sao nút có khoảng cách nhỏ nhất sẽ được

chọn.



+ ngược lại nếu mà không có nút kề nào thì hiển thị thông tin nút hiện tại là nút duy nhất làm cầu nối tới các nút còn lại, không có đường đi nào khác dài hơn hoặc là không thể đi tới được nếu không có nút bắc cầu.

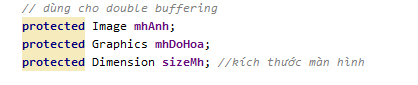


* 1. Giải pháp thực hiện hiệu ứng
     1. Canvas

- lớp GraphCanvas kết thừa từ lớp Canvas của Java AWT. Để thừa kế 2 phương thức là paint() và update()

- sử dụng các thuộc tính cho kĩ thuật double buffering, sẽ lưu lại bản sao của màn hình đồ họa trước khi vẽ lên trên canvas:

+ thuộc tính mhAnh thuộc kiểu Image sẽ tạo màn ảnh để phục vụ cho mhDoHoa



+ thuộc tính mhDoHoa kiểu Graphics sẽ lấy thuộc tính đồ họa từ mhAnh.

+ thuộc tính sizeMh sẽ lưu kích thước màn hình cần vẽ lên.

* + 1. Paint

Phương thức paint() thừa kế từ lớp Canvas dùng để vẽ lên màn hình canvas giao diện đồ họa

Phương thức paint() được bổ sung thêm các tính năng như:

+ vẽ các nút và mũi tên: vẽ các nút và cập nhật mũi tên mỗi lần repaint() được gọi tới.

+ vẽ hiệu ứng di chuyển nút đầu khi repaint() được gọi tới ở phương thức mouseDrag() khi người dùng kéo chuột.

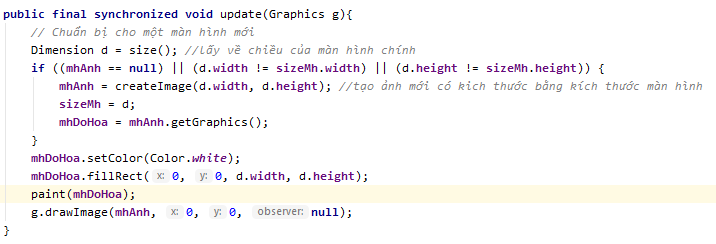
+ thực hiện gọi thuật toán xử lý và cập nhật các mũi tên khi thuật toán thay đổi.

+ hiển thị text ra doctext bằng cách gọi tới phương thức ketThucBuoc(g).

+ vẽ số thứ tự đỉnh ngay bên cạch các nút tương ứng.

* + 1. Update

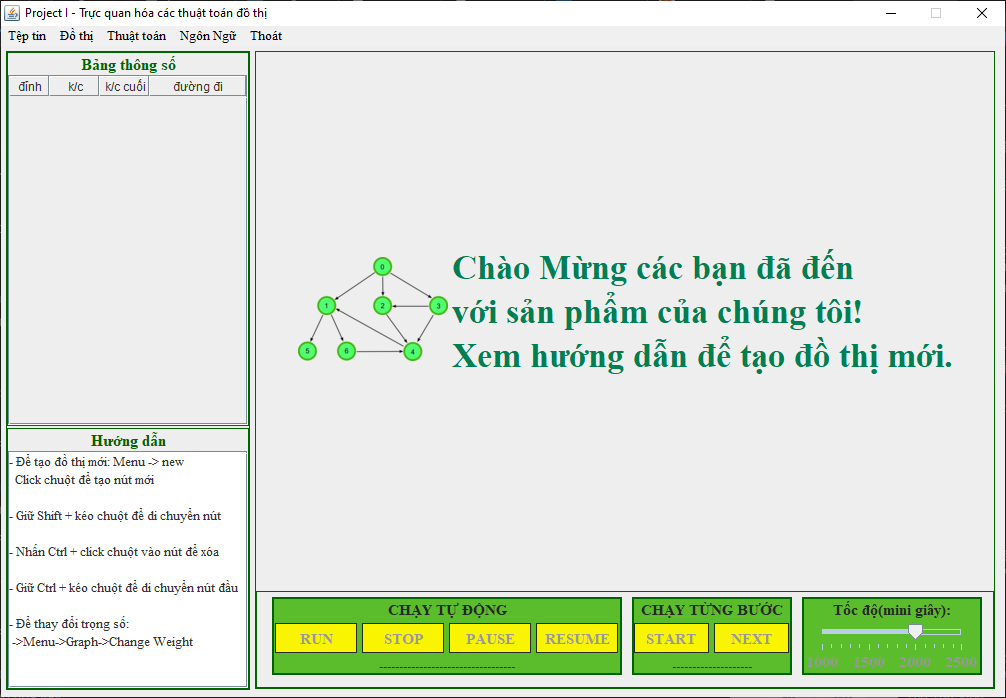
Phương thức update() dùng để cập nhật lại màn hình đồ họa. nó sẽ gọi paint() để vẽ màn hình đồ họa.



mhAnh sẽ tạo ảnh mới có kích thước bằng kích thước màn hình canvas. Sau đó mhDoHoa sẽ lấy phần đồ họa từ mhAnh và được gán màu trắng. Tiếp theo mhDoHoa sẽ vẽ một khối vuông có kích thước bằng kích thước ảnh.

Sau đó paint(mhDoHoa) để vẽ lên mhDoHoa.

1. KẾT QUẢ - DEMO – ĐÁNH GIÁ
   1. Minh họa các chức năng chương trình
      1. Giao diện phần mềm



Giao diện chính của phần mềm gồm 5 khu vực chính:

Khu vực lớn nhất là vùng vẽ đồ thị (chỗ hiển thị label chào mừng)

Khu vực tiếp bên dưới là phần điều khiển trung tâm (bao gồm các chức năng chạy dồ thị theo 2 cách: tự động và từng bước, và điều chỉnh tốc độ)

Khu vực trên bên trái là khu vực bảng hiển thị thông số và tên thuật toán bên trên

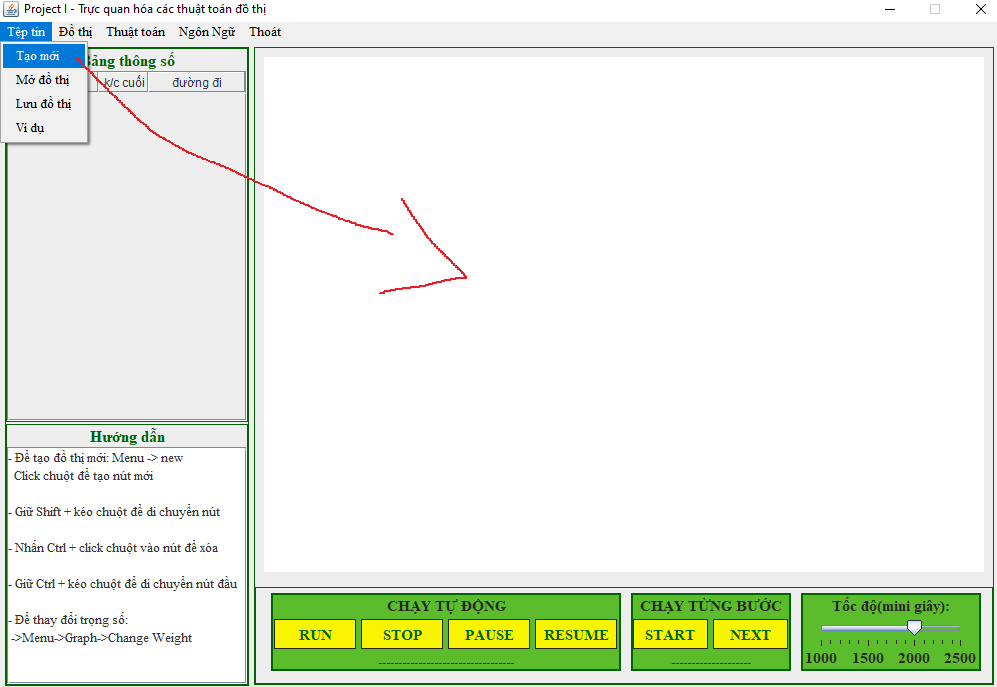
Khu vực dưới bên trái là khu vực hiển thị phần thông tin cho người dùng dễ dàng sử dụng và đồng thời hiển thị thông tin chi tiết khi chạy thuật toán

Khu vực cuối cùng là phần menu điều kiển bên trên cùng ( bao gồm các chức năng tạo file, save file, mở file chỉnh sửa đồ thị, chỉnh sửa ngôn ngữ..)

* + 1. Chức năng tạo mới đồ thị

Để tạo mới đồ thị người dùng cần làm theo hướng dẫn: vào phần menu 🡪 chọn tệp tin 🡪 chọn tạo mới

Màn hình bên trái sẽ hiển thị khu vực màu trắng thay thế cho baner chào mừng:



Sau đó người dùng làm theo hương dẫn click chuột để tạo các nút mới

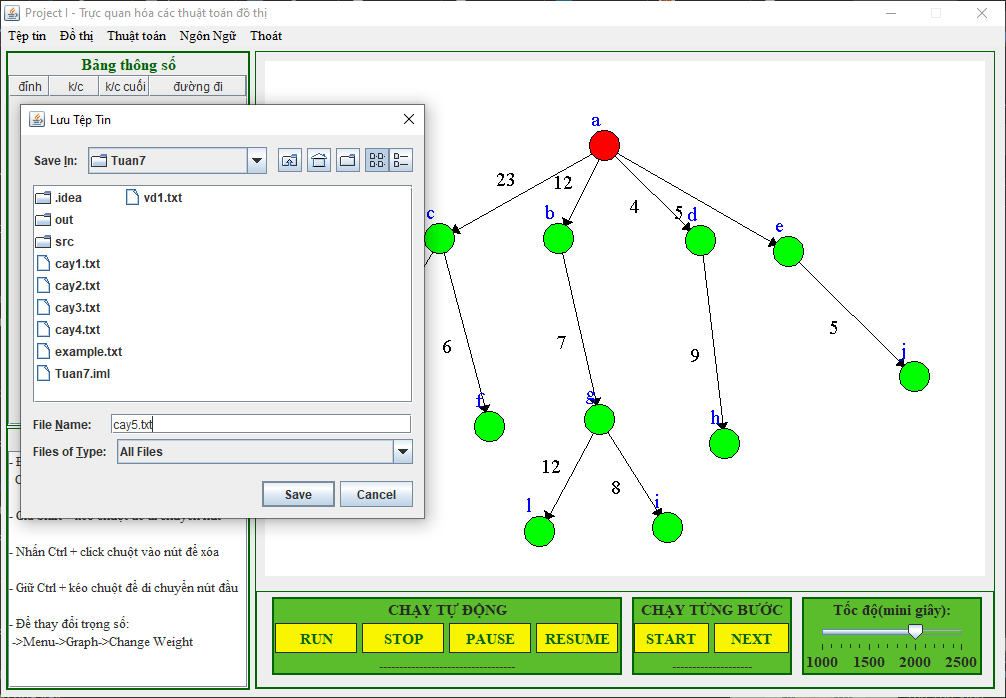
Kéo chuột từ nút này sang nút kia để tạo mũi tên mới

* + 1. Chức năng lưu đồ thị đã tạo

Sau khi đã tạo được 1 đồ thị hoàn chỉnh, người dùng có thể lưu file đồ thị để có thể mở lại lần sau nhanh chóng.

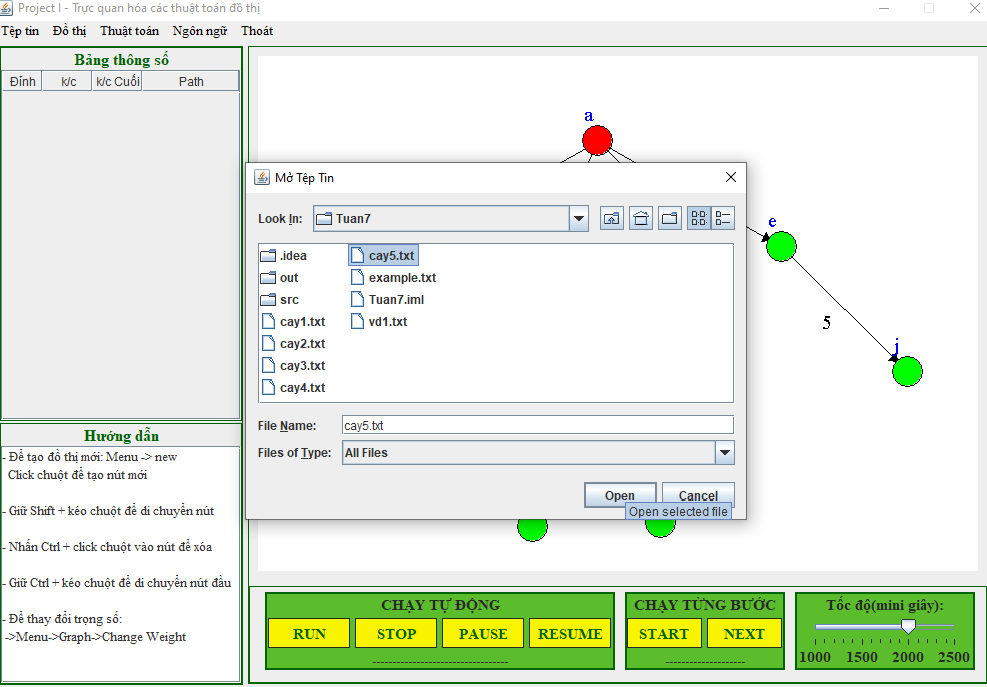
Làm tương tự như tạo mới: Vào Menu 🡪 chọn Lưu đồ thị 🡪 Một cửa sổ lưu đồ thị sẽ hiện ra, người dùng có thể chọn đường dẫn muốn lưu.

Sau khi chọn được đường dẫn và đặt tên file (với đuôi .txt) người dùng nhấn vào Save để lưu file ( ví dụ ở trên lưu trong file cay5.txt vào thư mục gốc project)



* + 1. Chức năng mở đồ thị đã lưu

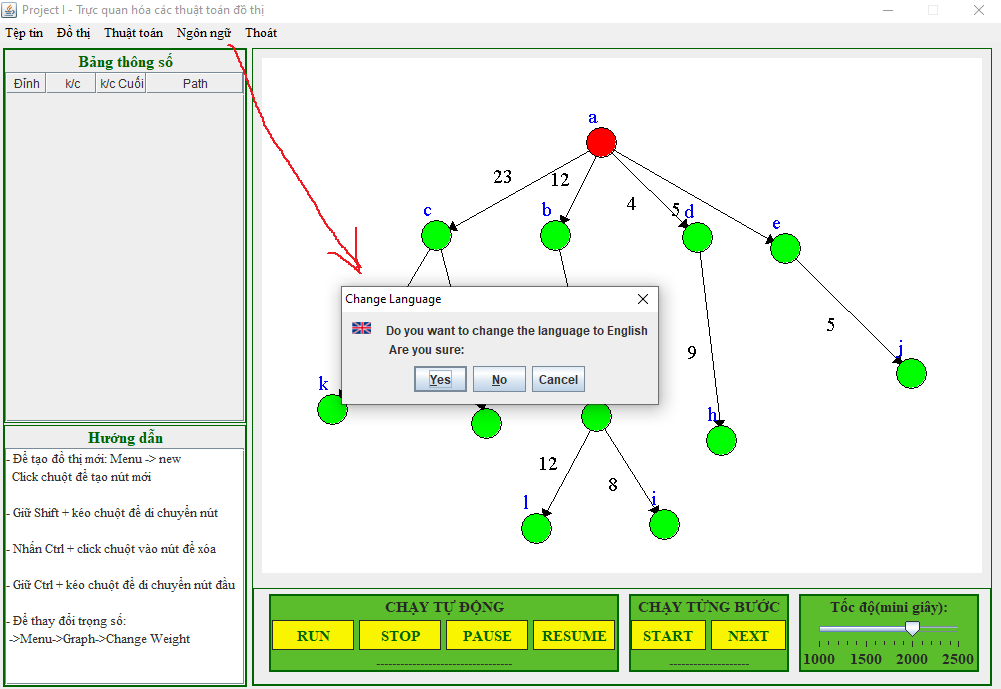
Để mở đồ thị đã lưu người dùng vào Menu 🡪 Mở đồ thị 🡪 Một cửa sổ giống với Lưu đồ thị hiện ra 🡪 chọn đúng file có đuôi .txt đã lưu lúc trước hoặc 1 file khác có định dạng tương tự



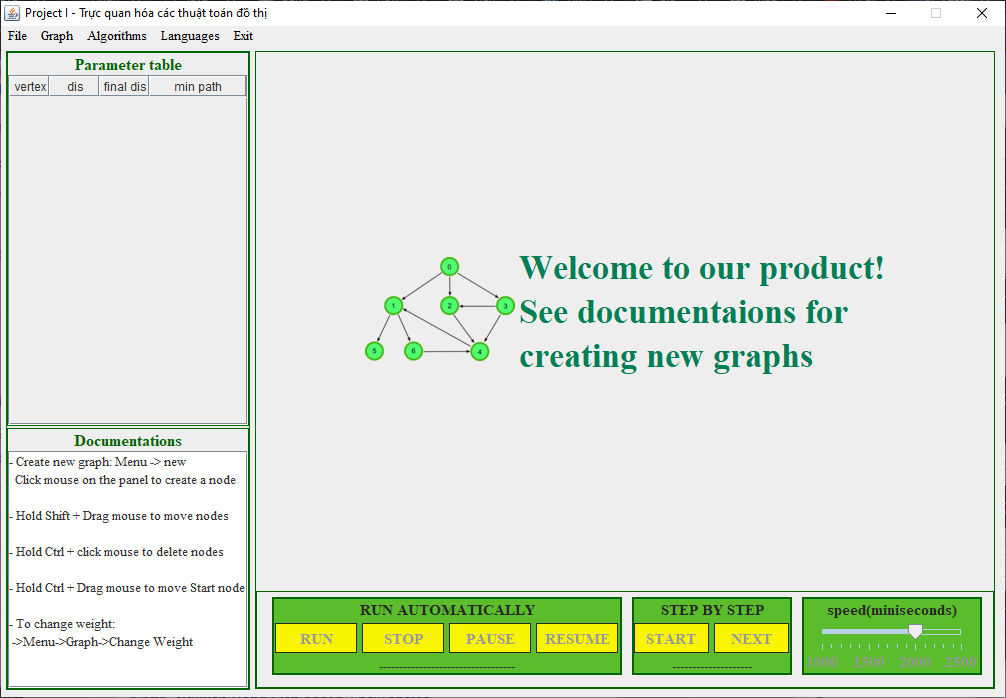
* + 1. Chức năng thay đổi ngôn ngữ

Phần mềm có cài đặt 2 ngôn ngữ Tiếng Anh và Tiếng Việt để phù hợp với đại đa số người dùng phổ thông.

Để thay đổi ngôn ngữ người dùng vào Menu 🡪 Ngôn ngữ (nếu như phần mền đang ở trạng thái tiếng việt – hoặc Language – nếu như phần mềm ở trạng thái tiếng anh) 🡪 Chọn ngôn ngữ muốn chuyển:



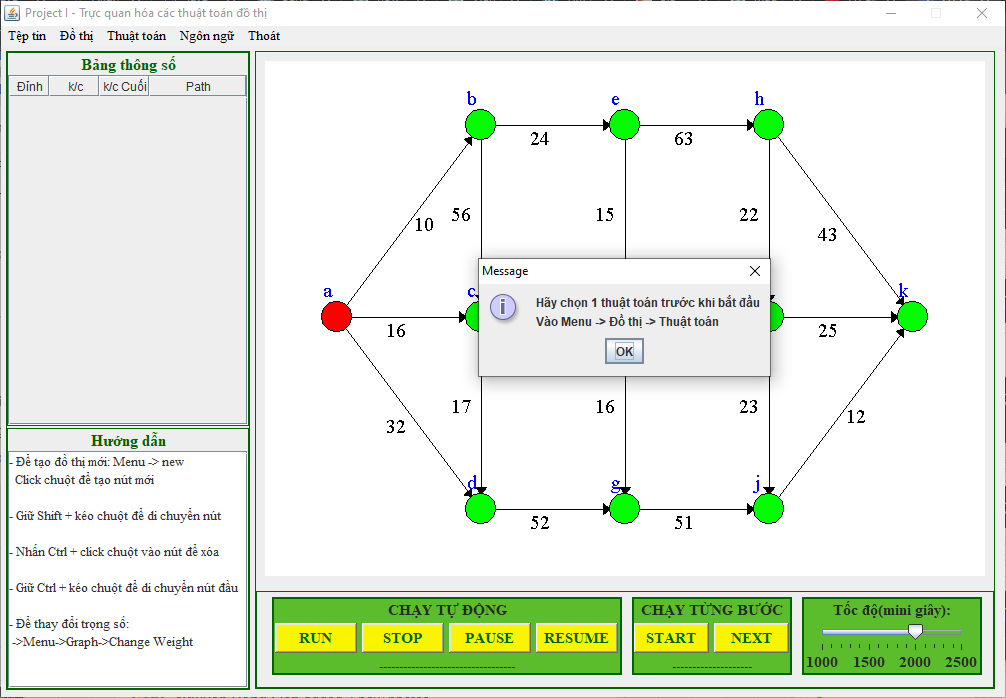
Sau khi nhấn Yes: Toàn bộ giao diện phần mềm sẽ chuyển sang tiếng anh:



* 1. Demo theo một vài kịch bản
     1. Thuật toán Dijkstra
        1. Chạy auto

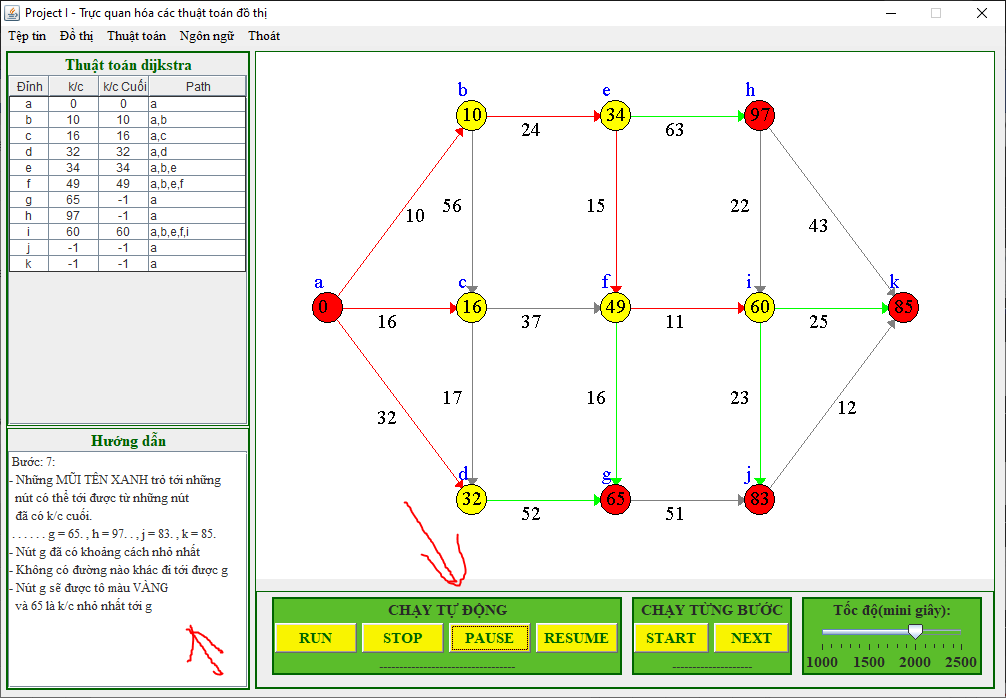
Để chạy tự động đầu tiên người dùng cần phải mở file đồ thị hoặc tạo mới

Sau đó cần phải chọn thuật toán nếu không sẽ bị lỗi:



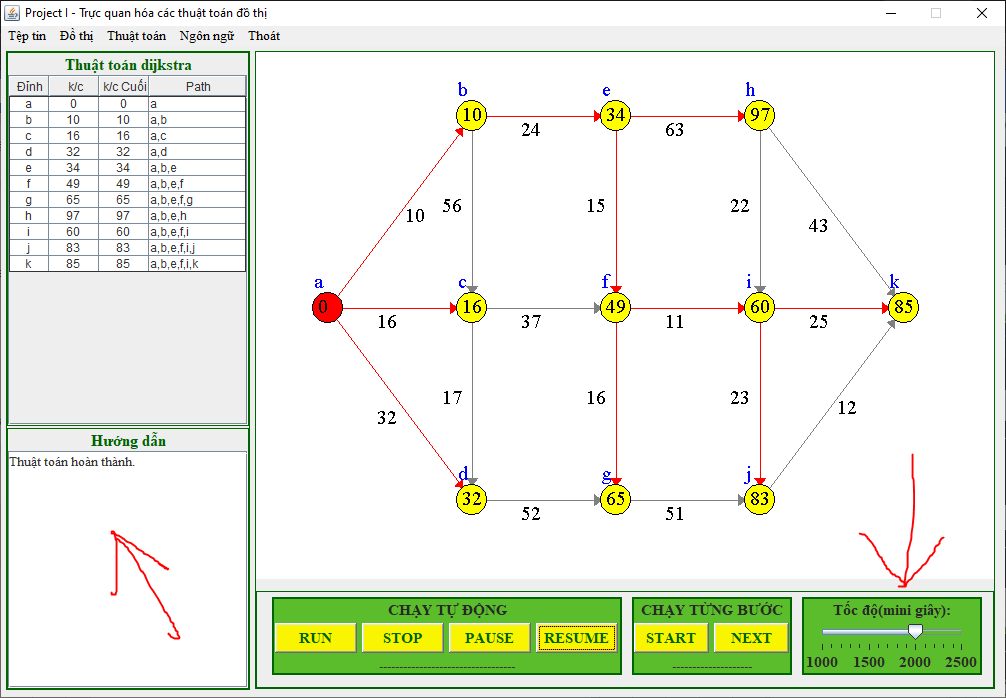
Khi đã chọn thuật toán Dijkstra vào panel chọn RUN để chạy tự động

Trong khi chạy người dùng có thể nhấn PAUSE để tạm dừng và xem hướng dẫn nếu không theo kịp, sau đó có thể nhân RESUME để tiếp tục. người dùng có thẻ nhấn STOP để dừng hẳn thuật toán:



Khi thuật toán dùng lại, ở phần hướng dẫn báo “thuật toán hoàn thành”, đồng thời tất cả các nút đều chuyển sang màu vàng (trừ nút bắt đầu) thì thuật toán đã hoàn tất

Người dùng có thể thay đổi tốc độ nếu thuật toán nhanh quá hoặc chậm quá bằng cách vào phần góc dưới dùng bên trái, kéo chuột vào thanh slider để thay đổi tốc độ



* + - 1. Chạy từng bước

Giống với chạy tự động chạy từng bước người dùng phải chọn thuật toán trước khi bắt đầu để tránh thông báo

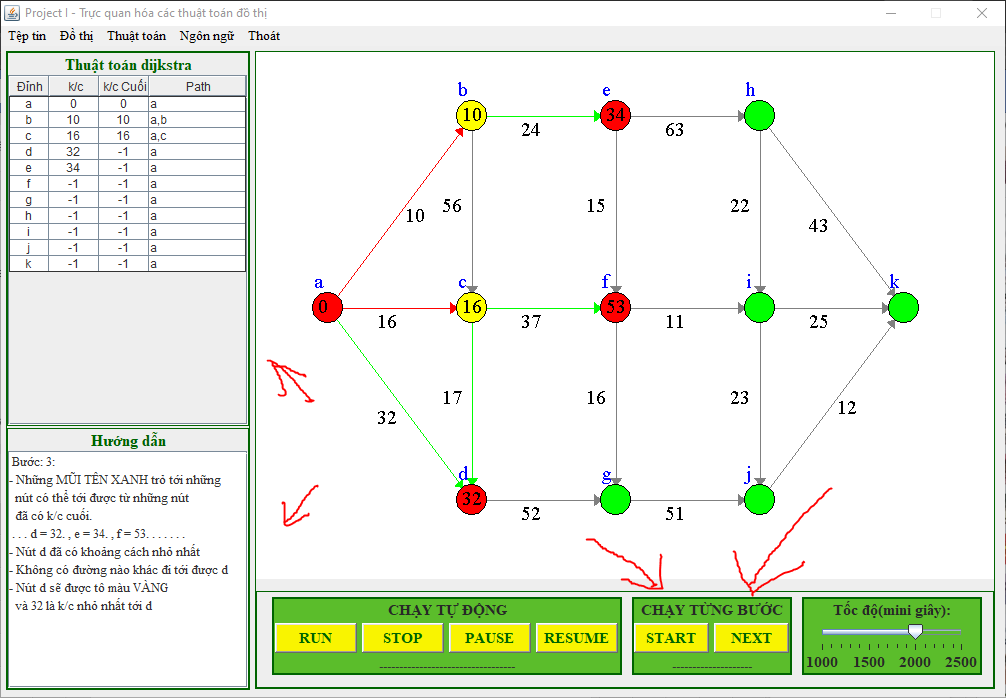
Sau khi người dùng chọn thuật toán xong. Nhấn START trong phần option   
chạy từng bước để bắt đâu:

Khi nhấn nút START lần đầu tiên, thuật toán sẽ chạy 1 bước đầu, người dùng xem xét các thông số ở bảng và phần hướng dẫn để hiểu thêm

Sau đó nhấn NEXT để thực hiện bước tiếp theo.

Cứ tiếp tục thực hiện như vậy cho đến khi thuật toán hoàn thành và nhận được kết quả giống như khi chạy tự động

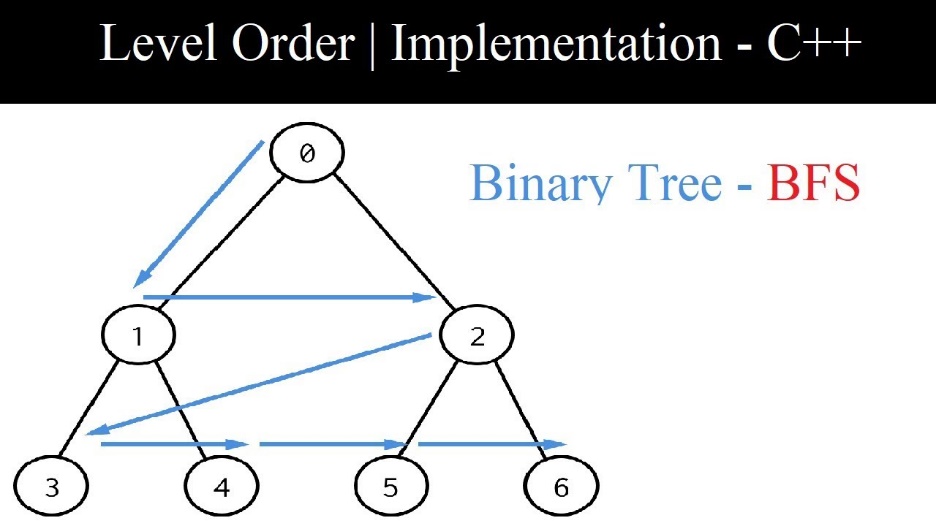
Ưu điểm của chạy từng bước là người dùng có thời gian nghiên cứu chi tiết các thông số và hướng dẫn để có thể hiểu thuật toán một cách sâu sắc.



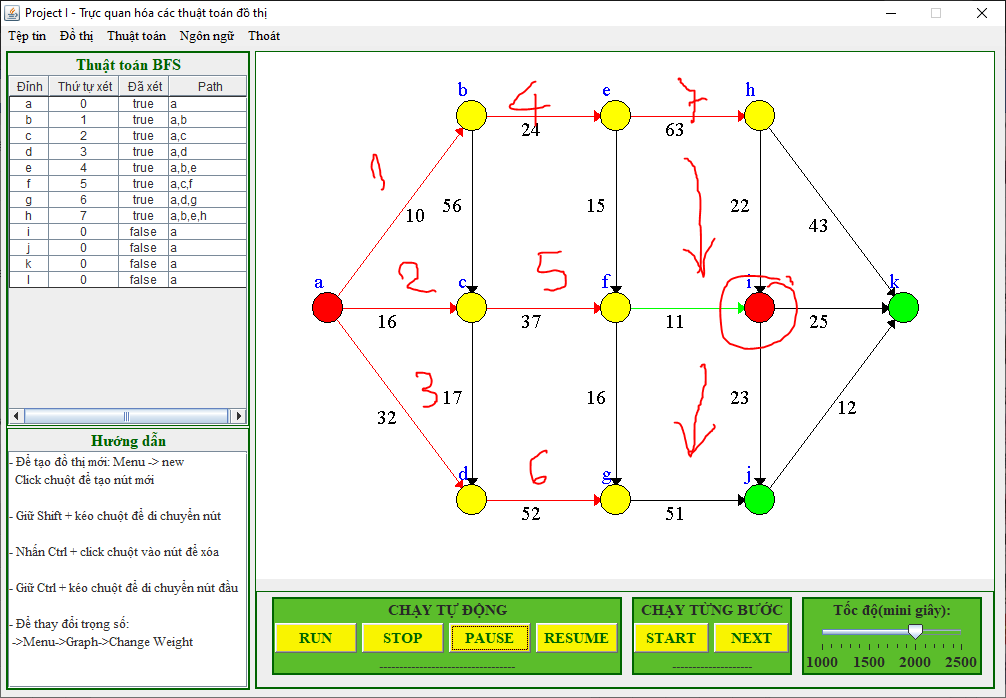
* + 1. Thuật toán BFS

Với thuật toán BFS người dùng chạy tương tự với 2 trường hợp tự động và từng bước

Thuật toán sẽ tìm kiếm theo chiều rộng theo hình zic zăc



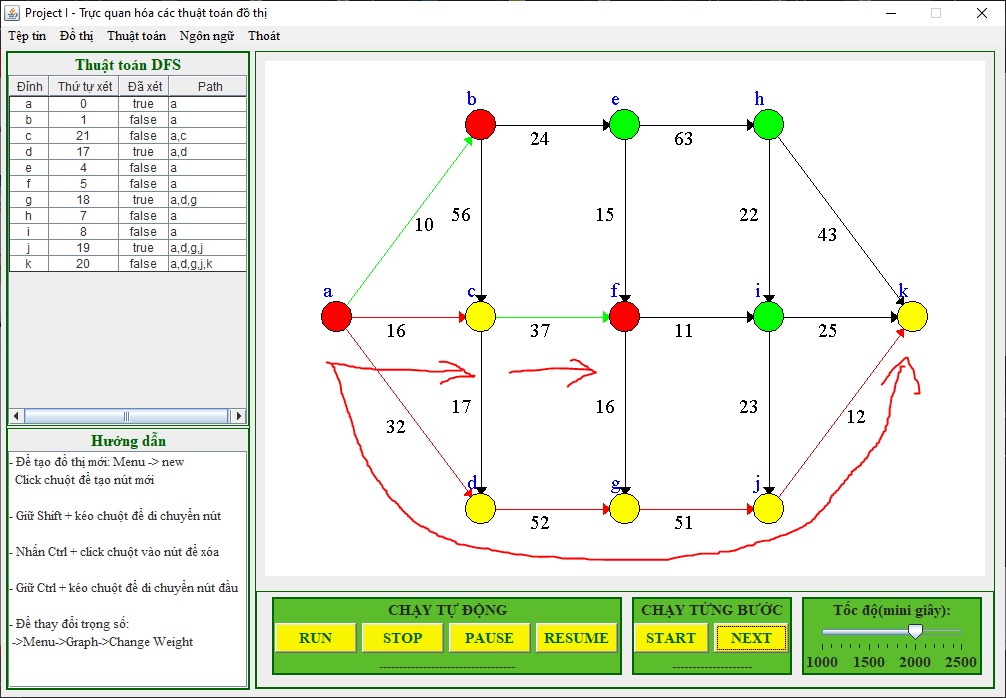
Các nút được tìm kiếm sâu dần theo theo cấp bậc (hết tầng trên rồi đến tầng dưới)



* + 1. Thuật toán DFS

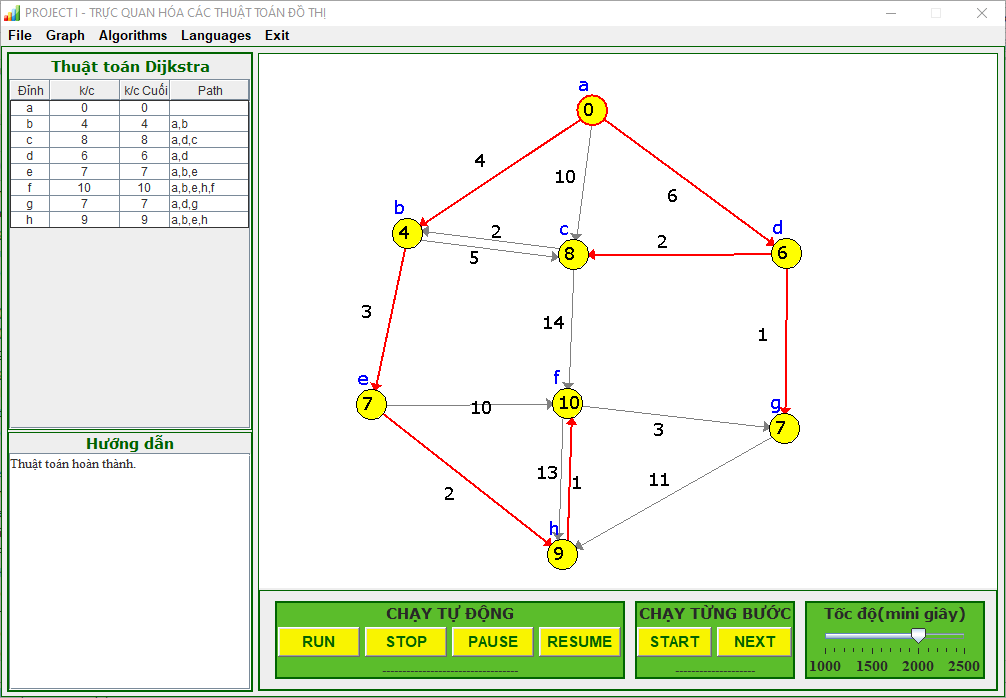
Thuật toán DFS – Tìm kiếm theo chiều sâu sẽ thực thi ngược lại hoàn toàn so với thuật toán BFS ở trên

DFS sẽ tìm đến tận cùng sau đó mới đi sang nhánh khác.

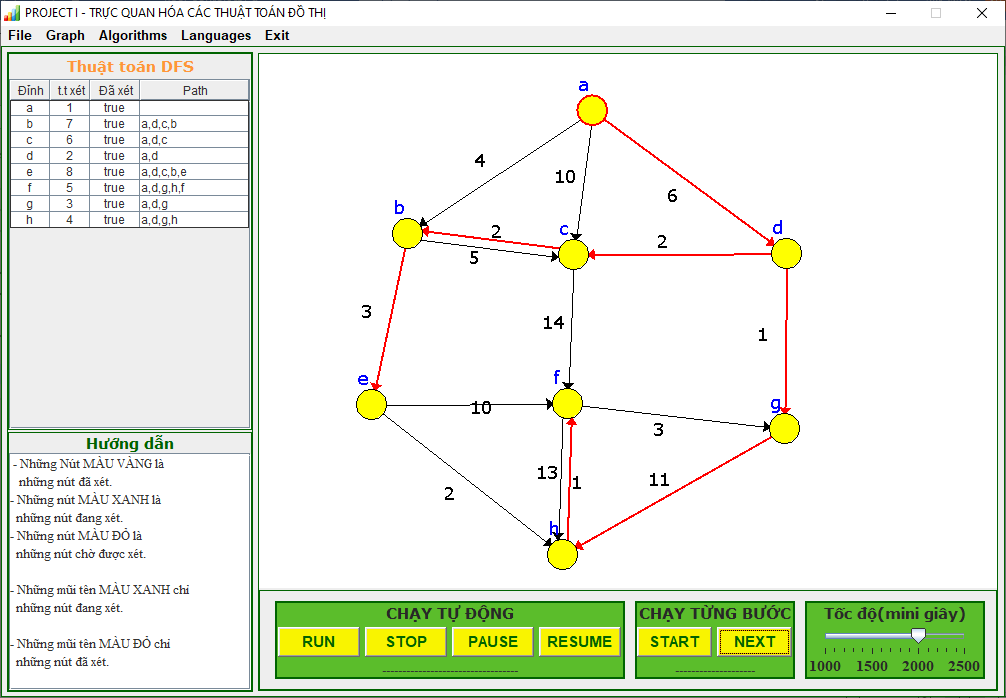


* 1. Đánh giá mức độ hoàn thành
     1. Kiểm thử
        1. Chạy với đồ thị 2 chiều

Thuật toán djkstra tìm đường đi ngắn nhất từ a 🡪 f chính xác với đường đi ngắn nhất a 🡪 b 🡪 e 🡪 h 🡪 f

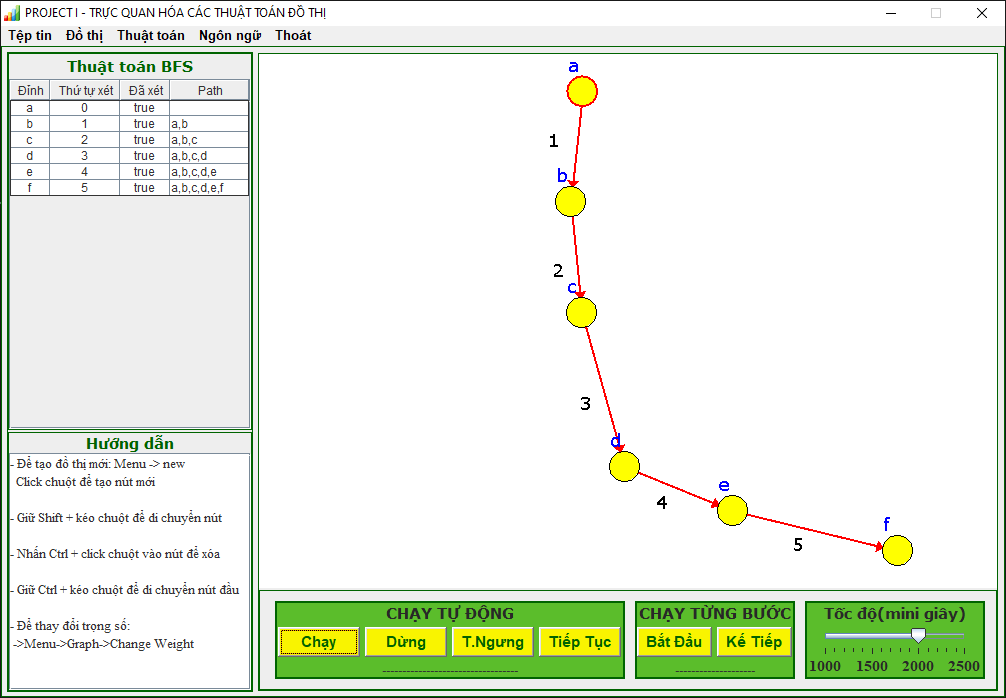


*Thuật toán dijkstra*

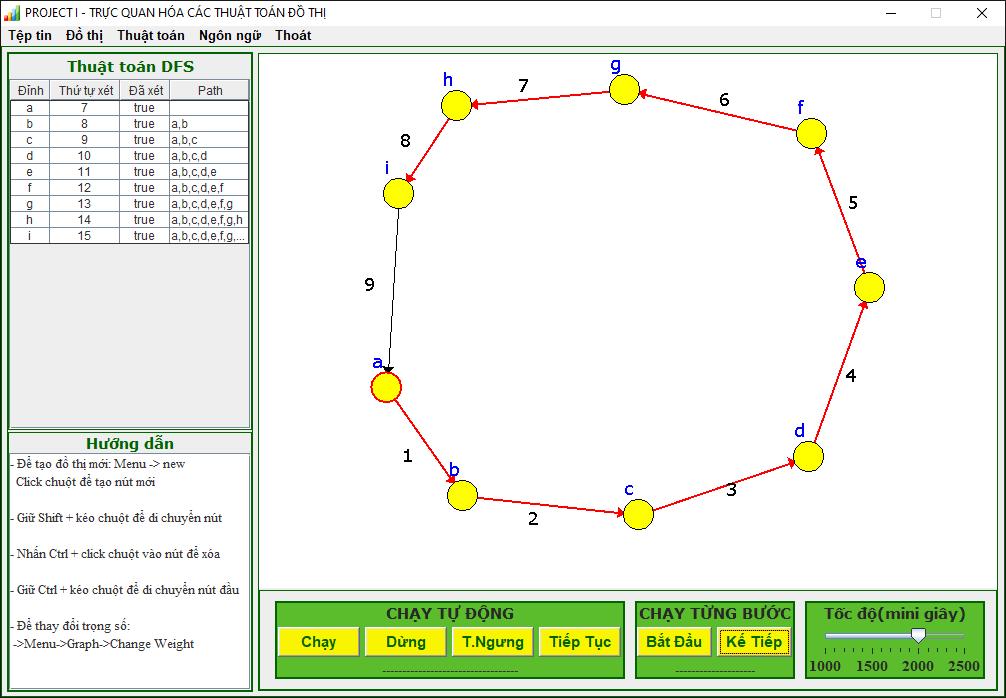


*Thuật toán DFS*

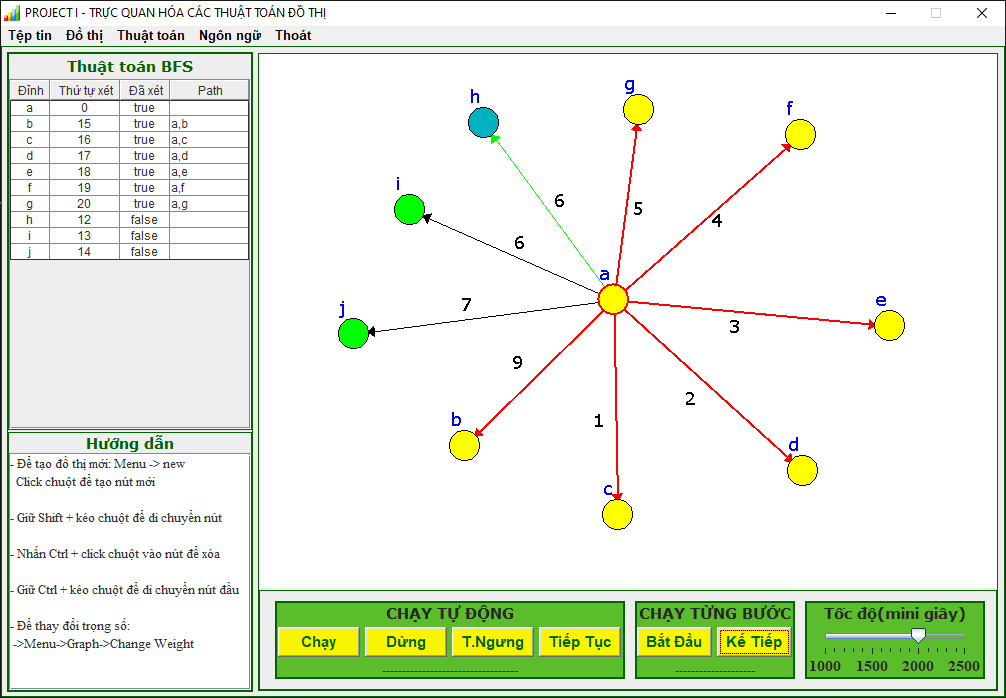
* + - 1. Chạy với các đồ thị đặc biệt
         1. Đồ thị chỉ có 1 đường



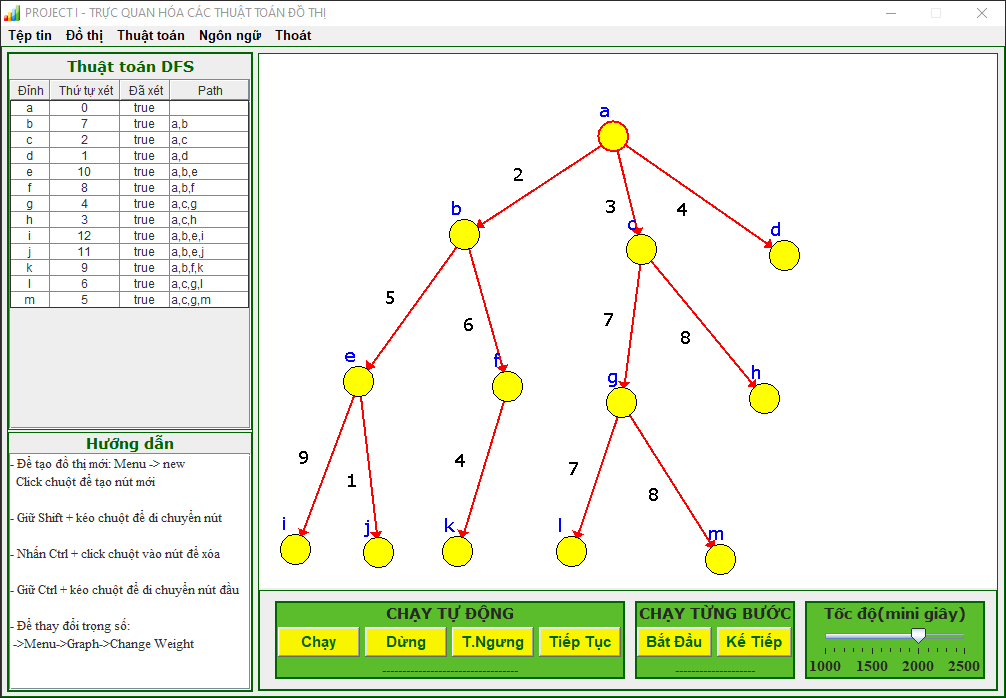
* + - * 1. Đồ thị dạng vòng



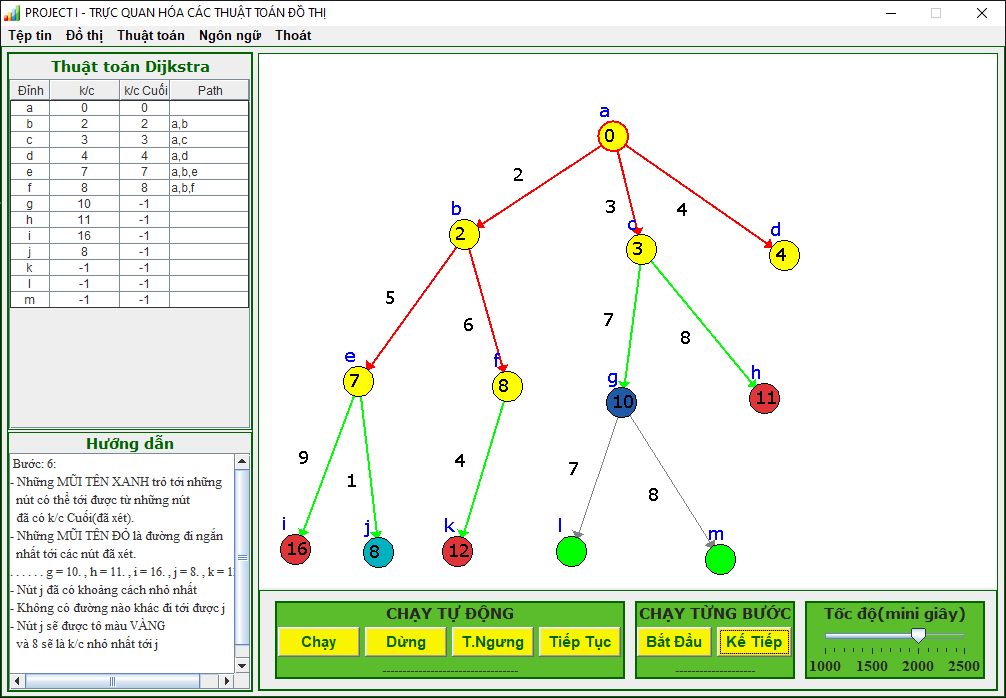
* + - * 1. Đồ thị hình sao



* + - * 1. Đồ thị hình cây



Với thuật toán DFS và BFS thuật toán chạy bình thường và cho kết quả đúng thứ tự xét.Thuật toán dijkstra đi đúng đường đi như hiển thị ở bảng thông số



* + 1. Đánh giá
       1. Mức độ hoàn thiện

Phần mềm cơ bản đã hoàn thiện về mặt đồ họa nhưng vẫn còn chưa được đẹp mắt do sử dụng java swing nên độ phân giải khi vẽ bằng canvas không được cao.

Phần mềm đã trình bày rõ ràng cho người dùng thuật toán đồ thị được chạy trực quan sinh động dễ hiểu do có cả phần hướng dẫn và thông số nhưng do hạn chế về mặt thời gian nên không thể triển khai thêm nhiều thuật toán.

* + - 1. Các chức năng còn thiếu sót

Do hạn chế về mặt thời gian và kiến thức nên phần mềm còn thiếu sót những chức năng như:

Tuy có 2 phần ngôn ngữ (Tiếng việt và Tiếng Anh) nhưng phần hướng dẫn của đồ thị hiện tại chỉ hỗ trợ tiếng việt chưa hỗ trợ được tiếng anh, Chỉ có phần giao diện thay đổi.

Chưa thực hiện được chức năng chạy song song giữa các thuật toán để so sánh hiệu năng.

Chức năng tạo hiệu ứng chuyển động sinh động hơn theo thời gian thực.

* + - 1. Hướng phát triển trong tương lai

Định hướng trong tương lai nếu có khả năng có thể phát triển phần mềm lên trên nền web, sử dụng css và các thư viện đồ họa như bootstrap để hoàn chỉnh phần đồ họa tốt hơn, thân thiện hơn. Sử dụng ReactJS để tạo hiệu ứng tốt hơn.

Tạo thêm bộ ngôn ngữ tiếng anh cho phần hướng dẫn và nếu có thể mở rộng hơn cho nhiều thứ tiếng khác

Tìm hiểu và gắn kết, tích hợp thêm các thuật toán đồ thị mới và mở rộng thêm cách trực quan các thuật toán khác như sắp xếp, hàng đợi ..

1. MÔ TẢ KẾ HOẠCH THỰC HIỆN PROJECT VÀ ĐÓNG GÓP CỦA CÁC THÀNH VIÊN
   1. Kế hoạch thực hiện

Tổng thời gian dự án : 15 Tuần (từ tuần 3 🡪 tuần 18)

* + 1. Tuần 3 🡪 5

Tìm hiểu lại các thuật toán và ôn tập kiến thức và bổ sung các kĩ năng cần có.

Viết và trình bày báo cáo kiến thức cơ bản và trình bày giải pháp.

Thực hiện cài đặt thử thuật toán và chạy với các bộ dữ liệu khác nhau.

Tìm hiểu giải pháp đồ họa và lựa chọn phương án tối ưu nhất có thể.

Cài đặt thử chương trình đồ họa đơn giản .

* + 1. Tuần 5 🡪 9

Triển khai viết mã nguồn.

Thay thế việc tạo dữ liệu bằng tay sang tạo dữ liệu từ thao tác click chuột.

Gắn kết thuật toán với phần đồ họa.

Giải pháp tạo, lưu và mở đồ thị bằng file, thiết kế cấu trúc file.

Thêm chức năng chỉnh sửa tốc độ cho thuật toán khi chạy tự động.

* + 1. Tuần 10 🡪 15

Trình bày giải pháp thực hiện các chức năng khó.

Hiển thị được thông tin đồ thị thông qua bảng thông số.

Thêm vào những chức năng hướng dẫn.

* + 1. Tuần 16 🡪 18

Viết báo cáo và chỉnh sửa các chức năng nhỏ cần thiết của phần mềm.

Hoàn thiện phần đồ họa của phần mềm cho thân thiện với người dùng.

* 1. Đóng góp của các thành viên
     1. Thành viên Nguyễn Thành Long
* Thực hiện phần viết mã chính, mã nguồn thuật toán Dijkstra
* Đưa ra giải pháp đồ họa trực quan hóa
* Viết báo cáo phần giải pháp thực hiện các chức năng của phần mềm.
  + 1. Thành viên Nguyễn Duy Hoài Lâm
* Đưa ra kế hoạch thực hiện dự án
* Thực hiện viết mã nguồn thuật toán BFS, DFS
* Viết báo cáo phần mở đầu và cơ sở lý thuyết
  1. Tài liệu tham khảo

[1] Herbert Schildt, Java: The Complete Reference - Seventh Edition.

[2] Cay S.Horstmann, Core Java Volume I - 10th Edition - 2016.

[3] Carla Laffra, GraphAlgorithm - March, 1996.