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.H MÔ PHỎNG GIẢI THUẬT SINH VÀ DUYỆT CÂY THEO THỨ TỰ SAU**

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

**Sinh viên thực hiện: Bùi Thanh Pháp**

**Mã số sinh viên: 64131768**

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.H MÔ PHỎNG GIẢI THUẬT SINH VÀ DUYỆT CÂY THEO THỨ TỰ SAU**

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

**Sinh viên thực hiện: Bùi Thanh Pháp**

**Mã số sinh viên: 64131768**

Khánh Hoà - 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 họa.h mô phỏng giải thuật sinh học và duyệt cây theo thứ tự sau

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

**Sinh viên được hướng dẫn:** Bùi Thanh Pháp

**MSSV:** 64131768

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Lần** | **Ngày** | **Nội dung** | **Nhận xét của GVHD** |
| 1 | 28/11/2024 | Nhận đề tài hướng dẫn và định hướng giải quyết vấn đề. |  |
| 2 | 6/12/2024 | Trình bày việc mô phỏng thuật toán chính dựa trên kiến thức đã được học ở những học phần trước đó. |  |
| 3 | 12/12/2024 | Sinh viên tiến hành thực hiện thuật toán đã đề ra với việc tạo ra các cách nhập dữ liệu đầu vào đa dạng như bàn phím, đọc file. |  |
| 4 | 21/12/2024 | Hoàn thành phần duyệt cây theo thứ tự sau và sửa lại giao diện cho hoàn chỉnh |  |
| 5 | 28/12/2024 | Sinh viên thực hiện demo và sửa những lỗi phát sinh trong quá trình làm. |  |
| 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 1](#_Toc186191559)

[TÓM TẮT 2](#_Toc186191560)

[CHƯƠNG 1 TỔNG QUAN 3](#_Toc186191561)

[1.1 Dev-C++ 3](#_Toc186191562)

[1.2. Thư viện Graphics.h 3](#_Toc186191563)

[1.3. Mô tả bài toán 4](#_Toc186191564)

[1.4. Cơ sở lý thuyết 4](#_Toc186191565)

[1.4.1. Khái niệm về cây 4](#_Toc186191566)

[1.4.2. Cây nhị phân 5](#_Toc186191567)

[1.4.3. Cây nhị phân tìm kiếm 6](#_Toc186191568)

[1.4.4. Duyệt theo thứ tự sau LRN (Left – Right – Node) 7](#_Toc186191569)

[1.4.5. Độ phức tạp giải thuật 13](#_Toc186191570)

[CHƯƠNG 2. PHƯƠNG PHÁP NGHIÊN CỨU 15](#_Toc186191571)

[2.1. Tổng quan về giao diện 15](#_Toc186191572)

[2.1.1. Cài đặt Dev-C++ và thư viện graphic.h 15](#_Toc186191573)

[2.1.2. Xử lý sự kiện chuột 16](#_Toc186191574)

[2.1.3. Xây dựng giao diện 19](#_Toc186191575)

[2.2. Phân tích thuật toán 22](#_Toc186191576)

[2.2.1. Nhập số lượng phần tử và giá trị phần tử từ bản phím 23](#_Toc186191577)

[2.2.2. Nhập phần tử Keyboard với số lượng phần tử mặc định 25](#_Toc186191578)

[2.2.3. Sử dụng chuột để nhập dữ liệu 27](#_Toc186191579)

[2.2.4. Tạo các phần tử bằng cách đọc file 30](#_Toc186191580)

[2.2.5. Thuật toán duyệt cây theo thứ tự LRN 32](#_Toc186191581)

[2.3. Các nút chức năng khác 36](#_Toc186191582)

[2.3.1. Nút NLR 36](#_Toc186191583)

[2.3.2. Nút LNR 39](#_Toc186191584)

[2.3.3. Nút Chiều Rộng 41](#_Toc186191585)

[2.3.4. Nút Reset 44](#_Toc186191586)

[CHƯƠNG 3. KẾT QUẢ THỰC HIỆN 46](#_Toc186191587)

[CHƯƠNG 4. KẾT LUẬN 51](#_Toc186191588)

[4.1. Ưu điểm 51](#_Toc186191589)

[4.2. Nhược điểm 51](#_Toc186191590)

[TÀI LIỆU THAM KHẢO 52](#_Toc186191591)

**DANH MỤC HÌNH ẢNH**

[Hình 1.1 Cấu trúc của các nút của cây 5](file:///C:\Users\TN\Documents\Zalo%20Received%20Files\BaoCaoTTCS_BuiThanhPhap_64131768.docx#_Toc186189252)

[Hình 1.2 Cấu trúc cây nhị phân 6](#_Toc186189253)

[Hình 1.3 Minh họa cây nhị phân 6](file:///C:\Users\TN\Documents\Zalo%20Received%20Files\BaoCaoTTCS_BuiThanhPhap_64131768.docx#_Toc186189254)

[Hình 1.4 Minh họa cây nhị phân tìm kiếm 7](#_Toc186189255)

[Hình 1.5 Minh họa cây nhị phân duyệt theo LRN 7](#_Toc186189256)

[Hình 1.6 Minh họa bước thuật toán 8](#_Toc186189257)

[Hình 2.1 Minh họa chạy graphics 16](#_Toc186189258)

[Hình 2.2 Minh họa khi nhấn Loadfile 19](#_Toc186189259)

[Hình 2.3 Giao diện demo 20](#_Toc186189260)

[Hình 2.4 Hình ảnh nhập số lượng và giá trị phần tử 24](#_Toc186189261)

[Hình 2.5 Hình ảnh giao diện khi nhập giá trị 25](#_Toc186189262)

[Hình 2.6 Hình ảnh số liệu trong file 26](#_Toc186189263)

[Hình 2.7 Hình ảnh khi người dùng thực hiện keyboard 27](#_Toc186189264)

[Hình 2.8 Giao diện xuất hiện khi thực hiện keyboard 27](#_Toc186189265)

[Hình 2.9 Giao diện khi sử dụng nút Mouse Input 30](#_Toc186189266)

[Hình 2.10. Khi nhấn vào số nó hiện bên console 30](#_Toc186189267)

[Hình 2.11 Hình ảnh tạo các phần từ bằng đọc file 32](#_Toc186189268)

[Hình 2.12 Giao diện khi nhấn vào nút Load file 32](#_Toc186189269)

[Hình 2.13 Kết quả khi duyệt cây LRN 36](#_Toc186189270)

[Hình 2.14 Kết quả khi duyệt cây NLR 39](#_Toc186189271)

[Hình 2.15 Kết quả khi chạy duyệt cây LNR 41](#_Toc186189272)

[Hình 2.16 Kết quả khi chạy duyệt cây theo chiều rộng 44](#_Toc186189273)

[Hình 3.1 Giao diện khi nhấn nút Mouse Input 46](#_Toc186189274)

[Hình 3.2 Hiện thị kết quả duyệt cây LRN khi nhấn nút Start 46](#_Toc186189275)

[Hình 3.3 Hình ảnh khi nhấn vào Keyboard 47](#_Toc186189276)

[Hình 3.4 Giao diện từ việc nhập giá trị từ nút Keyboard 47](#_Toc186189277)

[Hình 3.5 Hiển thị kết quả duyệt cây theo thứ tự giữa LNR 48](#_Toc186189278)

[Hình 3.6 Hình ảnh khi nhấn nút Input Size và nhập giá trị 48](#_Toc186189279)

[Hình 3.7 Giao diện hiển thị khi nhập giá trị từ nút Input Size 49](#_Toc186189280)

[Hình 3.8 Hiển thị kết quả khi nhấn nút NLR 49](#_Toc186189281)

[Hình 3.9 Giao diện khi nhấn nút Load file 50](#_Toc186189282)

[Hình 3.10 Hiển thị kết quả khi nhấn nút Chiều Rộng 50](#_Toc186189283)

# LỜI CẢM ƠN

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

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

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

Em xin chân thành cảm ơn!

# TÓM TẮT

Trong khoa học máy tính, cây nhị phân là một cấu trúc dữ liệu quan trọng được sử dụng để lưu trữ và xử lý dữ liệu một cách hiệu quả. Thuật toán duyệt cây là phương pháp cơ bản để truy xuất thông tin từ cây, trong đó duyệt cây theo thứ tự sau (LRN - Left, Right, Node) là một kỹ thuật quan trọng, ứng dụng trong nhiều lĩnh vực như phân tích cú pháp, xử lý biểu thức, và lập trình hệ thống.

Đề tài tập trung vào việc xây dựng một chương trình sử dụng ngôn ngữ C/C++ với phần mềm Dev-C++ và thư viện graphics.h để mô phỏng quá trình sinh cây nhị phân từ dữ liệu đầu vào và trực quan hóa thuật toán duyệt cây theo thứ tự sau. Chương trình minh họa rõ ràng từng bước trong quá trình duyệt cây, hiển thị đồ họa trực quan các nút, mối liên kết giữa chúng và thứ tự duyệt.

Sản phẩm minh họa cách cây nhị phân được khởi tạo và duyệt theo thuật toán LRN. Đồng thời, chương trình chỉ ra các ưu điểm và hạn chế của phương pháp, cũng như đề xuất cách khắc phục các nhược điểm. Ứng dụng chạy mượt mà với dữ liệu được nhập từ bàn phím hoặc file, đồng thời hỗ trợ tương tác thuận lợi với người dùng thông qua chuột.

# CHƯƠNG 1 TỔNG QUAN

## **1.1 Dev-C++**

Dev-C++ là một công cụ phát triển tích hợp (IDE) dành cho lập trình C và C++, được cung cấp miễn phí và có mã nguồn mở. Phần mềm này hoạt động trên hệ điều hành Windows và cho phép người dùng viết, biên dịch, cũng như thực thi các chương trình C/C++. Dev-C++ nổi bật với giao diện trực quan, dễ sử dụng, phù hợp cho cả người mới học và lập trình viên có kinh nghiệm. Hệ thống tích hợp của Dev-C++ bao gồm trình biên dịch MinGW hoặc TDM-GCC, khả năng quản lý dự án, và tính năng tự động gợi ý mã, giúp tối ưu hóa quá trình phát triển phần mềm. Bên cạnh đó, nó còn hỗ trợ công cụ gỡ lỗi, làm nổi bật cú pháp, và cho phép tích hợp các thư viện bên ngoài để mở rộng chức năng. Với những đặc điểm này, Dev-C++ trở thành một lựa chọn hiệu quả cho việc học tập.

## **1.2. Thư viện Graphics.h**

Thư viện graphics.h là một đồ họa thư viện được sử dụng trong trình cài đặt C/C++ để tạo ra các đồ họa ứng dụng trên nền tảng Windows. Thư viện này cung cấp các cơ sở chức năng cho phép người lập trình vẽ các hình học đối tượng như đường thẳng, hình chữ nhật, hình tròn và các đối tượng phức tạp hơn, giúp tạo ra đồ họa giao diện người dùng ( GUI) hoặc các ứng dụng đồ họa đơn giản.

Các tính năng chính của graphics.h:

(1)Vẽ hình cơ bản: Cung cấp các hàm để vẽ các hình cơ bản như đường thẳng (line), hình tròn (circle), hình chữ nhật (rectangle), hình elip (ellipse), và các giác khác nhau.

(2) Quản lý sắc màu: Cho phép thiết lập và thay đổi màu sắc của các đối tượng được vẽ, hỗ trợ nhiều màu sắc khác nhau.

(3) Quản lý vùng vẽ: Cho phép chuyển đổi giữa bản văn chế độ (chế độ văn bản) và đồ họa chế độ (chế độ đồ họa), tạo không gian để vẽ.

(4) Chuột điều khiển: Cung cấp các chức năng để theo dõi và điều khiển chuột điều khiển, giúp người dùng tương tác với ứng dụng.

(5) Đồ họa chế độ**:** graphics.h cho phép sử dụng các đồ họa chế độ khác nhau, từ đơn giản vẽ chế độ cho đến cấp độ cao hơn để hiển thị đồ họa phức tạp.

## **1.3. Mô tả bài toán**

Trong khoa học máy tính, phân tích cây nhị phân là một cơ sở dữ liệu cấu trúc và quan trọng, được sử dụng rộng rãi trong các ứng dụng như lưu trữ dữ liệu, cú pháp xử lý và tìm kiếm tối ưu. Một trong các phương pháp phổ biến để truy xuất thông tin từ cây là cây trình duyệt thuật toán. Trong đó, trình duyệt theo thứ tự sau (LRN - Left, Right, Node) trên môi trường đồ hoạ bằng các hàm đã được tích hợp sẵn trong thư viện Graphics.h.

Đề tài tập trung vào hai nhiệm vụ chính:

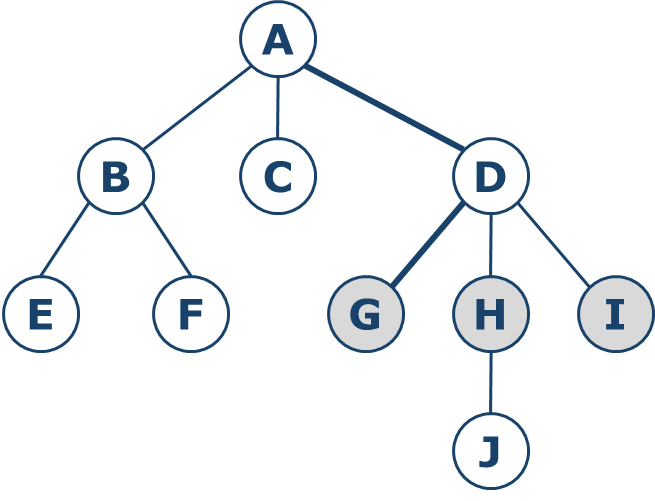
- Sinh cây nhị phân: Từ một danh sách đầu vào dữ liệu (ví dụ: số nguyên, ký tự), xây dựng cây nhị phân đồng bộ các quy tắc cụ thể. Mỗi phần tử được sắp xếp và gắn vào cây dựa trên giá trị, đảm bảo cấu trúc của hợp lệ nhị phân cây.

- Duyệt theo thứ tự sau (LRN): Thực hiện duyệt các nút trong cây theo thứ tự: Trái - Phải - Gốc . Thuật toán đảm bảo tất cả các nút đều được duyệt theo thứ tự và quá trình này được xác định rõ ràng thông tin qua đồ họa giao diện.

## **1.4. Cơ sở lý thuyết**

### **1.4.1. Khái niệm về cây**

Cây là một tập hợp các phần tử, gọi là các nút, được liên kết với nhau thông qua mối quan hệ cha con. Trong một cây, mỗi nút cha có thể có nhiều nút con, nhưng mỗi nút con chỉ có duy nhất một nút cha. Cây thường được tổ chức theo cấu trúc phân cấp, với một nút đặc biệt gọi là nút gốc (root), là nút không có cha. Mỗi nút trong cây có thể được mô tả bằng các thuộc tính như tên nút, danh sách các nút con (nếu có), và mức độ (depth), là khoảng cách từ nút gốc đến nút đó.



Hình 1.1 Cấu trúc của các nút của cây

**Cây tam phân, chiều cao 4**

**Mức 2**

**Mức 3**

**Mức 1**

**Mức 0**

**Nút gốc**

**Nút lá**

Từ hình 1 ta thấy số lượng nhánh của mỗi nút là ngẫu nhiên nên cần lưu ý một số chú ý sau:

- Bậc của một nút: là số nút con của nút đó. Ví dụ bậc của nút A là 3, bậc của nút C là 0, bậc của nút H là 1…

- Bậc của cây: là giá trị bậc lớn nhất của các nút (cây n-phân). Ví dụ cây trong hình trên có bậc 3, gọi là cây tam phân, cây có bậc 2 gọi là cây nhị phân…

- Nút lá: các nút mà không có nút con. Ví dụ các nút lá: E, F, C, G, I, J

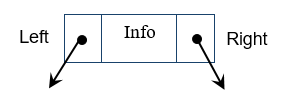
- Nút nhánh: là nút có bậc khác 0 mà không phải nút gốc (hay còn gọi là nút trung gian). Ví dụ các nút B, D, H

- Mức của nút: là số nguyên từ 0, các nút ngang hàng nhau thì có cùng mức. (nút gốc A có mức là 0, mức 1 gồm các nút B, C, D, mức 2 gồm E, F, G, H, I, mức 3 gồm J)

- Chiều cao (chiều sâu): là mức lớn nhất của các nút lá. Cây trên có chiều sâu là 4.

### **1.4.2. Cây nhị phân**

Cây nhị phân là một dạng đặc biệt của cây, trong đó mỗi nút chỉ có tối đa hai nút con, gọi là nút con trái và nút con phải. Cây nhị phân thường được sử dụng rộng rãi trong tin học và là cơ sở cho nhiều thuật toán hiệu quả. Cây nhị phân có thể được phân loại thành cây nhị phân đầy đủ (full binary tree), cây nhị phân đều (perfect binary tree), và cây nhị phân cân bằng (balanced binary tree).

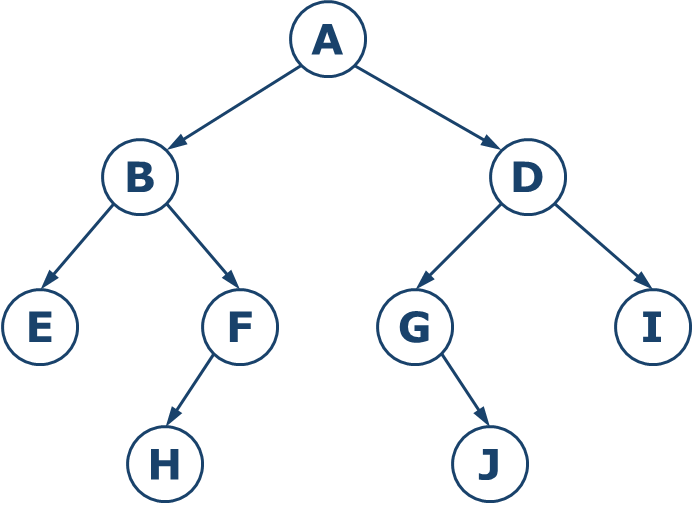


Hình 1.2 Cấu trúc cây nhị phân

- Mỗi phần tử trong cây nhị phân chứa 3 thành phần:

+ Info: phần tử chứa thông tin / giá trị của nút (node).

+ Left: phần lưu trữ địa chỉ của nút bên trái.

 + Right: phần lưu trữ địa chỉ của nút bên phải

Hình 1.3 Minh họa cây nhị phân

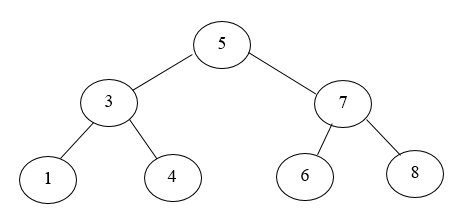
Nút con bên phải của A

Nút con bên trái của B

Đối với cây ta cần phải quan tâm giữ lại nút gốc, bởi từ nút gốc, đi theo các hướng liên kết Left, Right ta có thể duyệt mọi nút khác.

### **1.4.3. Cây nhị phân tìm kiếm**

Cây nhị phân tìm kiếm là cây nhị phân mà giá trị của phần tử bên trái của một node có giá trị nhỏ hơn giá trị của node, giá trị của các phần tử bên phải của một node thì lớn hơn giá trị của node đó.



Hình 1.4 Minh họa cây nhị phân tìm kiếm

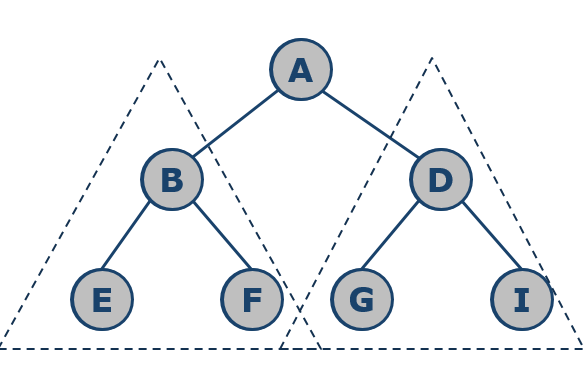
### **1.4.4. Duyệt theo thứ tự sau LRN (Left – Right – Node)**

- Phương pháp này duyệt theo thứ tự:

+ Duyệt cây con bên trái theo LRN

+ Duyệt cây con bên phải theo LRN

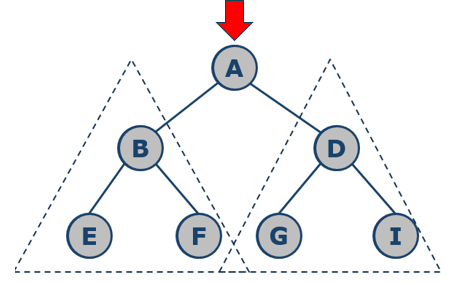
+ Thăm nút gốc



Hình 1.5 Minh họa cây nhị phân duyệt theo LRN

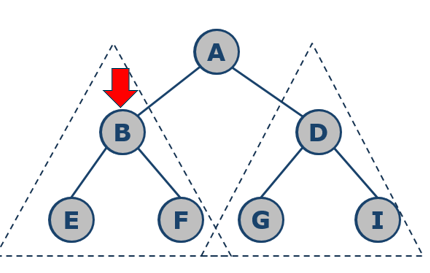
- Các bước duyệt cây theo LRN:

+ Đầu tiên tại nút góc A.

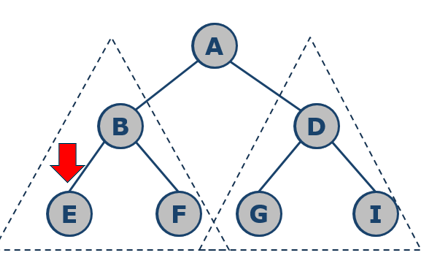


Hình 1.6 Minh họa bước thuật toán

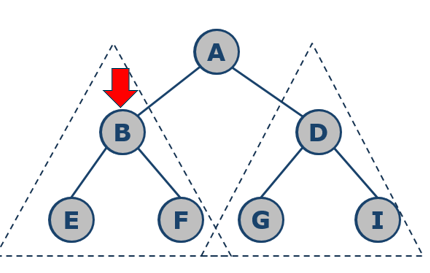
+ Theo quy tắc LRN, ta bắt đầu duyệt cây con trái của A tức là nút B.



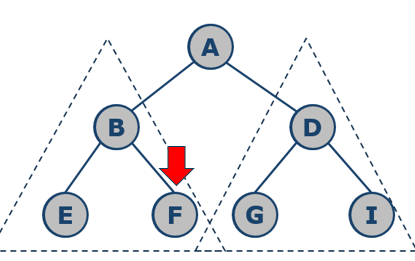
+ Tại B theo quy tắc ta duyệt cây con bên trái của B tức là nút E.



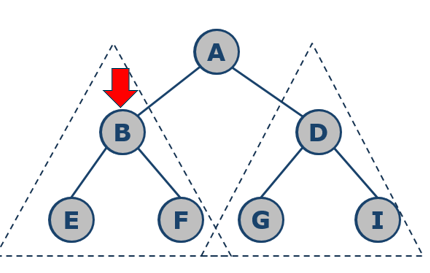
+ Lúc này nút E không có cây con trái hoặc phải nên ta ghi nhận giá trị nút E.



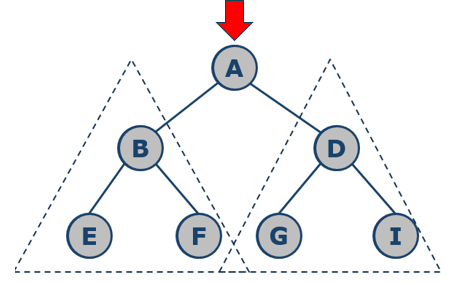
+ Ta quay lại nút B, sau khi duyệt cây con bên trái chuyển sang duyệt cây con bên phải của nút B đó là nút F.



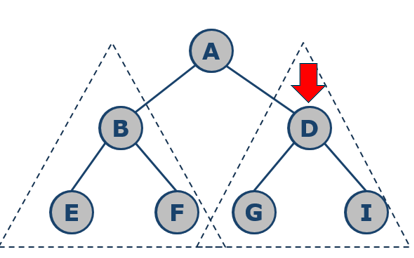
+ Tại đây, ta ghi nhận giá trị nút F, vì F không có cây con bên trái hoặc cây con bên phải.



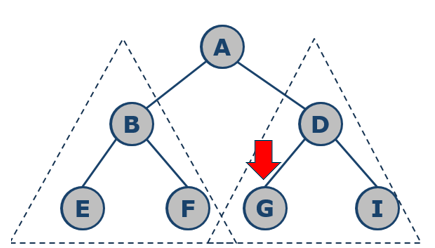
+ Ta quay lại nút B, sau khi duyệt xong cây con bên trái (E) và cây con bên phải (F), ghi nhận giá trị nút B.



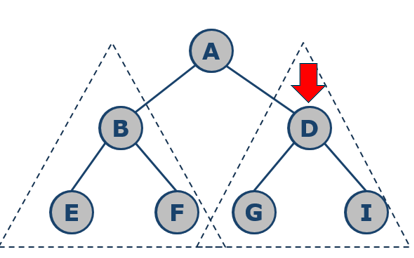
+ Quay lại nút A, Sau khi duyệt xong cây con trái của A (cây con gốc B), chuyển sang duyệt cây con phải của A (cây con gốc D).



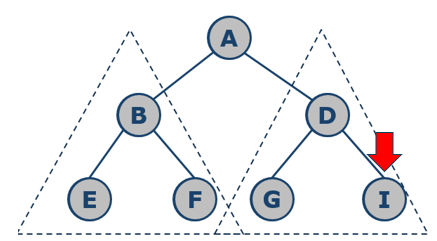
+ Tại D theo quy tắc ta duyệt cây con bên trái của D tức là nút G.



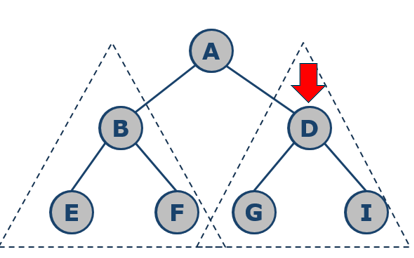
+ Lúc này nút G không có cây con trái hoặc phải nên ta ghi nhận giá trị nút G.



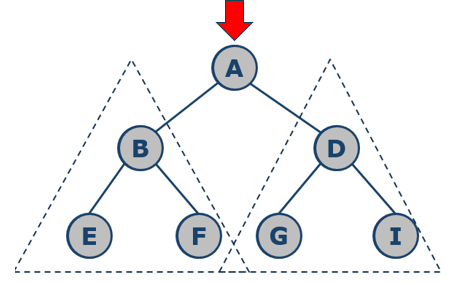
+ Ta quay lại nút D, sau khi duyệt cây con bên trái chuyển sang duyệt cây con bên phải của nút D đó là nút I.



+ Tại đây, ta ghi nhận giá trị nút I, vì I không có cây con bên trái hoặc cây con bên phải.



+ Ta quay lại nút D, sau khi duyệt xong cây con bên trái (G) và cây con bên phải (I), ghi nhận giá trị nút D.



+ Khi đã duyệt xong cả cây con trái (B) và cây con phải (D) của nút A, ta ghi nhận giá trị nút A.

Từ các bước duyệt cây theo LRN, ta thu được kết quả thứ tự duyệt LRN :

E F B G I D A

R

R

R

R

L

L

L

L

- Hàm duyệt cây theo thứ tự LRN:

void DuyetLRN(Node\* root) {

if (root != NULL) {

DuyetLRN(root->left); // Duyệt cây con bên trái

DuyetLRN(root->right); // Duyệt cây con bên phải

printf("%d ", root->info); // Thăm nút hiện tại

}

### **1.4.5. Độ phức tạp giải thuật**

- Về độ phức tạp thời gian, để duyệt một cây nhị phân, ta cần ghé thăm tất cả các nút trong cây. Với n là số lượng nút trong cây, mỗi nút chỉ được ghé thăm một lần, và việc xử lý mỗi nút (in giá trị hoặc thực hiện phép toán) mất thời gian O(1). Vì vậy, độ phức tạp thời gian của giải thuật là: O(n).

Ví dụ: Trong một cây nhị phân có n = 7 nút, nếu ta duyệt cây, mỗi nút chỉ được ghé thăm một lần (tương ứng với n = 7 lần xử lý). Điều này khẳng định độ phức tạp thời gian luôn tuyến tính với n.

- Về độ phức tạp không gian, độ phức tạp không gian phụ thuộc vào ngăn xếp (stack) được sử dụng để quản lý các lời gọi đệ quy:

+ Trường hợp tốt nhất (cây cân bằng): trong cây nhị phân cân bằng, chiều cao của cây là h = log2(n), vì vậy số khung ngăn xếp tối đa sẽ là O(log2(n)).

+ Trường hợp xấu nhất (cây suy biến): nếu cây suy biến thành một danh sách liên kết (tức là chỉ có con trái hoặc con phải), chiều cao của cây là h = n. Khi đó, ngăn xếp cần O(n) không gian. Độ phức tạp không gian trong trường hợp tổng quát: O(h) với h là chiều cao của cây.

# CHƯƠNG 2. PHƯƠNG PHÁP NGHIÊN CỨU

## **2.1. Tổng quan về giao diện**

### **2.1.1. Cài đặt Dev-C++ và thư viện graphic.h**

Để sử dụng được thư viện graphic.h tải file cài đặt phần mềm Dev-C++ theo đường dẫn (<https://drive.google.com/file/d/1qMnzjQdm2yVzmUvzxjTHV74BAj956HKd/view>). Sau đó mở file vừa tải, và tiến hành cài đặt. Thư viện graphics.h tiến hành cài đặt như sau:

**- Đối với Dev C++ 32-bit:**

+ Bước 1: Chúng ta cần copy 2 file "graphics.h" và "winbgim.h" vào thư mục: (Path)\MinGW32\include

+ Bước 2: Copy file "libbgi.a" vào thư mục: (Path)\MinGW32\lib

+ Bước 3: Chúng ta tiếp tục copy 2 file "6-ConsoleAppGraphics.template" và "ConsoleApp\_cpp\_graph.txt" vào thư mục: (Path)\Templates

**- Đối với Dev C++ 64-bit:**

+ Bước 1: Ta cần copy 2 file “graphics.h” và “winbgim.h” vào thư mục: “(Path)\MinGW64\x86\_64-w64-mingw32\include”.

+ Bước 2: Copy file "libbgi.a" vào thư mục "(Path)\MinGW64\x86\_64-w64-mingw32\lib".

+ Bước 3: Ta tiếp tục copy 2 file "6-ConsoleAppGraphics.template" và "ConsoleApp\_cpp\_graph.txt" vào thư mục "(Path)\Templates".

**- Hướng dẫn sử dụng: Sau khi cài đặt theo các bước như hướng dẫn ở trên, các bạn mở Dev C++ lên.**

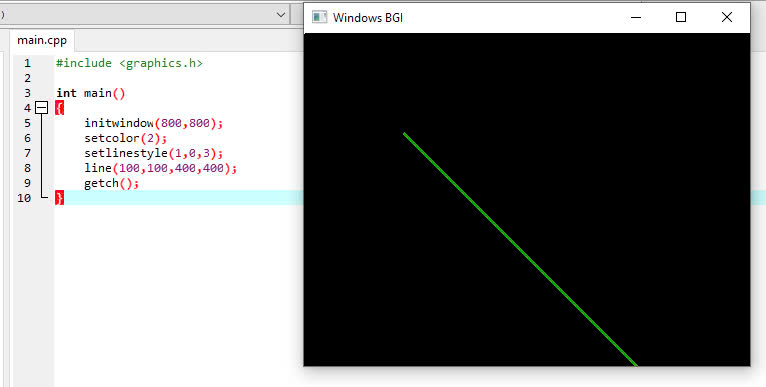
+ Bước 1: Nếu bạn cài Dev C++ 32 bit thì bỏ qua bước này, còn nếu là 64 bit thì các bạn chọn trình biên dịch là TDM-GCC 32 bit ứng với phiên bản Dev C++ bạn cài đặt (Đây là bước bắt buộc và phải làm từ đầu, nếu không thì sẽ không chạy được đồ họa).

Ví dụ: Nếu Dev C++ 5.6.3 64 bit thì bạn sẽ chọn trình biên dịch là "TDM-GCC 4.8.1 32 bit Release".

+ Bước 2: Tạo một Project mới bằng cách chọn File → New → Project...

Chọn Console Graphics Application → Tick vào ô C++ Project → Gõ tên Project vào ô Name → Click Ok và chọn vị trí lưu.

+ Bước 3: Tiến hành lập trình đồ họa và chạy thử.



Hình 2.1 Minh họa chạy graphics

### **2.1.2. Xử lý sự kiện chuột**

//ham kiem tra su kien chuot

void mouse() {

int x\_mouse, y\_mouse;

while (enable\_click) {

// Check for left mouse click event

if (ismouseclick(WM\_LBUTTONDOWN)) {

// Get mouse position

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

// Print x, y position to console

printf("Mouse clicked at (%d, %d)\n", x\_mouse, y\_mouse);

// Checking for Load file button

if (x\_mouse > 690 && x\_mouse < 800 && y\_mouse > 10 && y\_mouse < 50) {

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

readfile();

drawTree();

}

// Checking for Keyboard button

if (x\_mouse > 690 && x\_mouse < 800 && y\_mouse > 70 && y\_mouse < 110) {

printf("Input from keyboard...\n");

keyboard();

drawTree();

}

// Checking for Starting button

if (x\_mouse > 690 && x\_mouse < 800 && y\_mouse > 130 && y\_mouse < 170) {

printf("Running demonstration...\n");

demo();

}

// Checking for Input Size button

if (x\_mouse > 690 && x\_mouse < 800 && y\_mouse > 190 && y\_mouse < 230) {

printf("Input size from keyboard...\n");

inputSize();

drawTree();

}

// Checking for NLR button

if (x\_mouse > 20 && x\_mouse < 80 && y\_mouse > 470 && y\_mouse < 510) {

printf("NLR traversal...\n");

demoNLR();

}

// Checking for LNR button

if (x\_mouse > 110 && x\_mouse < 170 && y\_mouse > 470 && y\_mouse < 510) {

printf("LNR traversal...\n");

demoLNR();

}

// Checking for Breadth-First Search (Chieu Rong) button

if (x\_mouse > 200 && x\_mouse < 320 && y\_mouse > 470 && y\_mouse < 510) {

printf("Breadth-First Search (Chieu Rong) traversal...\n");

demoBFS();

}

// Checking for Reset button

if (x\_mouse > 350 && x\_mouse < 420 && y\_mouse > 470 && y\_mouse < 510) {

printf("Resetting...\n");

reset();

}

}

// Check for right mouse click to exit loop

if (ismouseclick(WM\_RBUTTONDOWN)) {

// Get mouse position

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

// Print exit message to console

printf("Right mouse click detected. Exiting...\n");

break;

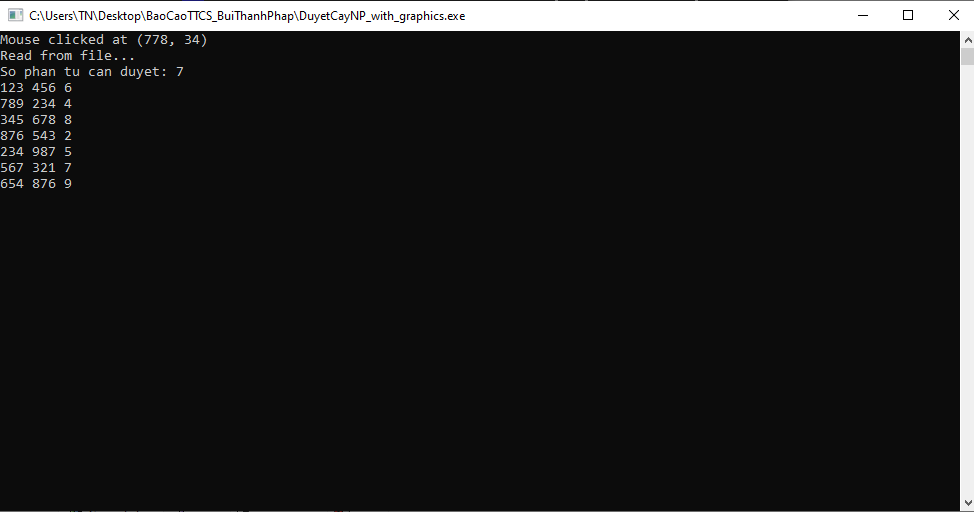
}

delay(100); // Wait for the next mouse event cycle

}

}

Hàm mouse xử lý sự kiện chuột trong giao diện đồ họa. Khi chuột trái được click, chương trình lấy tọa độ chuột và kiểm tra xem có nằm trong vùng nút bấm nào không. Nút "Load file": Khi chuột click vào vùng tọa độ xác định (từ x = 690, y = 10 đến x = 800, y = 50), chương trình sẽ gọi hàm readfile() để đọc dữ liệu từ file và sau đó gọi hàm drawTree() để vẽ lại cây dữ liệu. Nút "Keyboard": Khi chuột click vào vùng tọa độ xác định (từ x = 690, y = 70 đến x = 800, y = 110), chương trình sẽ kích hoạt hàm keyboard() để cho phép người dùng nhập dữ liệu từ bàn phím. Sau đó, cây dữ liệu sẽ được vẽ lại thông qua hàm drawTree(). Nút "Starting": Khi chuột click vào vùng tọa độ xác định (từ x = 690, y = 130 đến x = 800, y = 170), chương trình sẽ chạy chương trình demo thông qua hàm demo(). Nút "Input size": Khi chuột click vào vùng tọa độ xác định (từ x = 690, y = 190 đến x = 800, y = 230), chương trình sẽ yêu cầu người dùng nhập số lượng phần tử và giá trị các phần tử thông qua hàm inputSize(), sau đó vẽ lại cây dữ liệu bằng drawTree()…



Hình 2.2 Minh họa khi nhấn Loadfile

### **2.1.3. Xây dựng giao diện**

Giao diện được thiết kế nhằm mô tả các thông tin và chức năng của đề tài, bao gồm các phần chính như: khu vực hiển thị, là màn hình chính được sử dụng để hiển thị cây và kết quả của các thao tác duyệt cây; các nút chức năng như "Load file" để tải dữ liệu đầu vào từ tệp tin, "Keyboard" cung cấp tùy chọn nhập dữ liệu trực tiếp qua bàn phím, "Starting" để bắt đầu thực hiện quá trình mô phỏng và duyệt cây, và "Input Size" nhập số lượng và giá trị các phần tử. Ngoài ra, giao diện còn hiển thị thông tin đề tài bao gồm tên giáo viên hướng dẫn (GVHD), tên sinh viên thực hiện, mã số sinh viên (MSSV), lớp, và tên demo (ví dụ "Duyệt cây LRN"). Ngoài ra còn có chức năng duyệt khác như: duyệt cây theo thứ tự trước (NLR), giữa (LNR), chiều cao, reset và mouse input. Giao diện được thiết kế theo phong cách tối giản, đảm bảo dễ sử dụng và tập trung vào chức năng hiển thị các thao tác liên quan đến cây nhị phân.



Hình 2.3 Giao diện demo

//Ve giao dien

void GUI\_init(){

setcolor(WHITE);

//main border

setlinestyle(0, 1, 3);//setlinestyle(int linestyle, unsigned pattern, int thickness)

rectangle(10, 10, 660, 450);

//load file button

rectangle(690, 10, 800, 50);

settextstyle(2, 0, 7);

outtextxy(700, 20, "Load file");

//keyboard button: y=y+60 for top an bottom

rectangle(690, 70, 800, 110);

settextstyle(2, 0, 7);

outtextxy(700, 80, "Keyboard");

//Demo button: y=y+60 for top an bottom

rectangle(690, 130, 800, 170);

settextstyle(2, 0, 7);

outtextxy(700, 140, "Starting");

// input size button

rectangle(690, 190, 800, 230);

settextstyle(2, 0, 7);

outtextxy(700, 200, "Input Size");

// Cac nut duyet khac

rectangle(20, 510, 80, 470);

settextstyle(2, 0, 7);

outtextxy(30, 480, "NLR");

rectangle(110, 510, 170, 470);

settextstyle(2, 0, 7);

outtextxy(120, 480, "LNR");

rectangle(200, 510, 320, 470);

settextstyle(2, 0, 7);

outtextxy(210, 480, "Chieu Rong");

// Reset button

rectangle(350, 510, 420, 470);

settextstyle(2, 0, 7);

outtextxy(360, 480, "Reset");

//Dung chuot nhap du lieu

rectangle(450, 510, 570, 470);

settextstyle(2, 0, 7);

outtextxy(460, 480, "Mouse Input");

//Thong tin sinh vien

settextstyle(2, 0, 7);

outtextxy(690, 300, "GVHD: Doan Vu Thinh");

settextstyle(2, 0, 7);

outtextxy(690, 330, "Ten: Bui Thanh Phap");

settextstyle(2, 0, 7);

outtextxy(690, 360, "MSSV: 64131768");

settextstyle(2, 0, 7);

outtextxy(690, 390, "Lop: 64.CNTT-3");

settextstyle(2, 0, 7);

outtextxy(690, 420, "Demo: Duyet cay LRN");

}

Trong đoạn code trên, màu vẽ hiện tại được thiết lập là màu trắng và kiểu vẽ đường được đặt là kiểu liền nét với độ rộng là 3. Sau đó, một hình chữ nhật được vẽ từ điểm (10, 10) đến điểm (660, 470), tạo thành khung chính của giao diện. Tiếp theo, ba nút chức năng được vẽ ở các vị trí khác nhau bên phải giao diện, bao gồm các nút "Load file", "Keyboard", "Starting", và "Input Size", mỗi nút được đặt tên tương ứng bằng cách sử dụng hàm outtextxy. Dưới khung chính giao diện, còn có năm nút chức năng "NLR", "LNR", "Chieu Rong", "Reset" và "Mouse Input". Ngoài ra, ở phía dưới các nút bấm, một vùng hiển thị thông tin được thiết kế, nơi thông tin chi tiết về sinh viên, như tên, mã số sinh viên, lớp học, đề tài, và giáo viên hướng dẫn, được hiển thị rõ ràng bằng các lệnh outtextxy tại những vị trí xác định.

## **2.2. Phân tích thuật toán**

**Khai báo thư viện:**

#include<stdio.h>

#include<graphics.h>

#include<stdlib.h>

#include <time.h>//de su dung rand()

#include <queue> // Thu vien cho hang doi

//Dinh nghia so luong toi da va ten tap tin dau vao

#define MAX 7

#define INPUT "points.inp"

**Khai báo hàm và biến toàn cục:**

//khai bao protype cac ham

void inputSize();

void readfile();

void printpoints(struct toado m[MAX], int size);

void GUI\_init();

void mouse();

void keyboard();

void demo();

void drawBinaryTree(struct Node\* root, int x, int y, int offset, int circleColor, int textColor);

void drawTree();

void postOrderTraversal(struct Node\* root, int x, int y, int offset, int\* step, int \*displayX, int displayY);

void freeTree(struct Node\* root);

void reset();

void preOrderTraversal(struct Node\* root, int x, int y, int offset, int\* step, int\* displayX, int displayY);

void demoNLR();

void inOrderTraversal(struct Node\* root, int x, int y, int offset, int\* step, int\* displayX, int displayY);

void demoLNR();

void levelOrderTraversal(struct Node\* root, int\* step, int\* displayX, int displayY);

void demoBFS();

//khai bao bien toan cuc

int sodinh;//so dinh (so phan tu can duyet)

FILE \*fp;//con tro tep

char buffer[7];

int enable\_click = 1;

void GUI\_demo(); //hien thi giao dien demo

void GUI\_demoNLR();

void GUI\_demoLNR();

void GUI\_demoBFS();

struct toado{

int x;

int y;

int value;

};

### **2.2.1. Nhập số lượng phần tử và giá trị phần tử từ bản phím**

// Ham xu ly so luong va nhap gia tri tu ban phim

void inputSize() {

printf("\nNhap so luong phan tu (toi da %d): ", MAX);

scanf("%d", &sodinh);

if (sodinh > MAX) {

printf("So phan tu vuot qua gioi han cho phep. So phan tu toi da là %d\n", MAX);

sodinh = MAX;

}

srand(time(NULL));

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

printf("Nhap gia tri cho phan tu thu %d: ", i + 1);

scanf("%d", &td[i].value);

td[i].x = rand() % 640 + 10;

td[i].y = rand() % 460 + 10;

}

printf("\nNhap xong danh sach phan tu!\n");

printpoints(td, sodinh);

}

Đoạn code trên sẽ yêu cầu người dùng nhập số lượng phần tử mà người dùng muốn. Nếu người dùng nhập số lượng quá mức cho phép tức là 7 thì nó mặc định số lượng phần tử là 7. Sau đó, người dùng sẽ nhập giá trị phần tử.

Ví dụ: Nhap so luong phan tu: 5

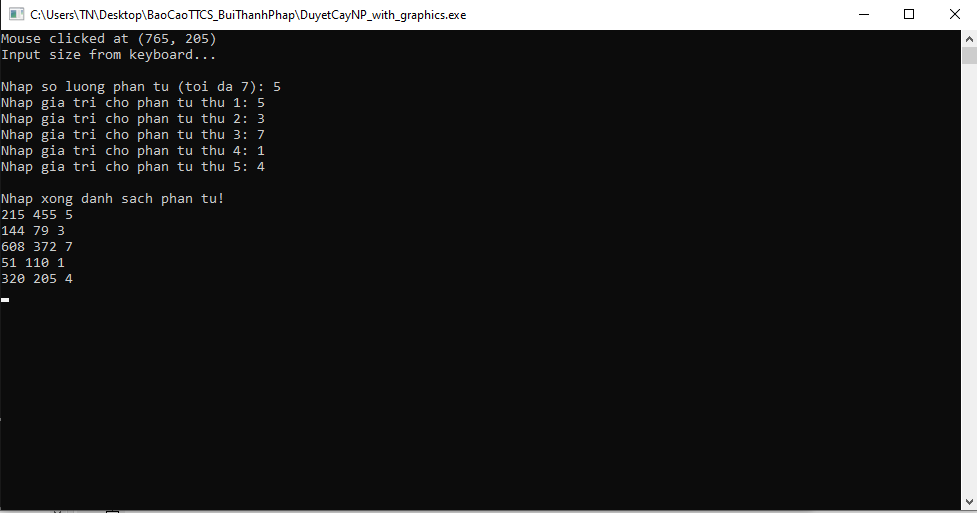
Nhap gia tri cho phan tu thu 1: 5

Nhap gia tri cho phan tu thu 2: 3

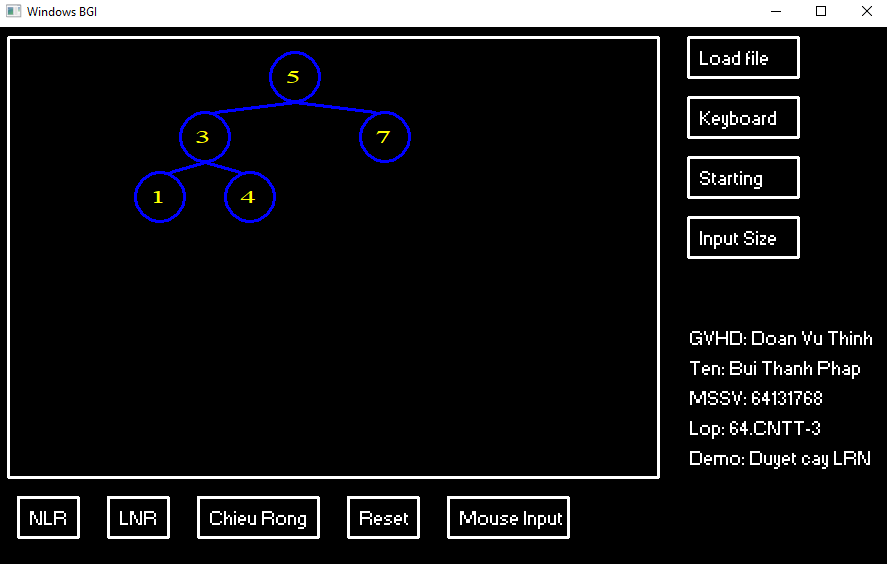
Nhap gia tri cho phan tu thu 3: 7

Nhap gia tri cho phan tu thu 4: 1

Nhap gia tri cho phan tu thu 5: 4



Hình 2.4 Hình ảnh nhập số lượng và giá trị phần tử



Hình 2.5 Hình ảnh giao diện khi nhập giá trị

### **2.2.2. Nhập phần tử Keyboard với số lượng phần tử mặc định**

//ham nhap tu ban phim

void keyboard(){

fp=fopen(INPUT,"r");

if(fp==NULL){

printf("File not found");

return;

}

//else

//doc dong dau tien cua tap tin

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

//doc lan luot tung phan tu: moi ptu co 2 tham so: toado\_x, toado\_y

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

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

//nhap gia tri cho cac diem

printf("\ntd[%d].value = ",i);

scanf("%d",&td[i].value);

}

//in ra cac diem

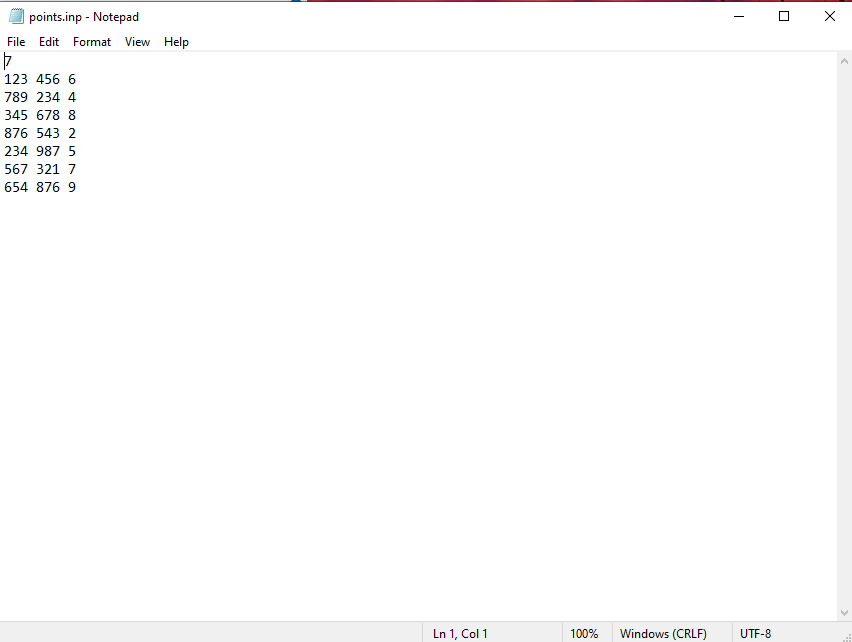
printpoints(td,sodinh);

printf("\nFinished!");

fclose(fp);

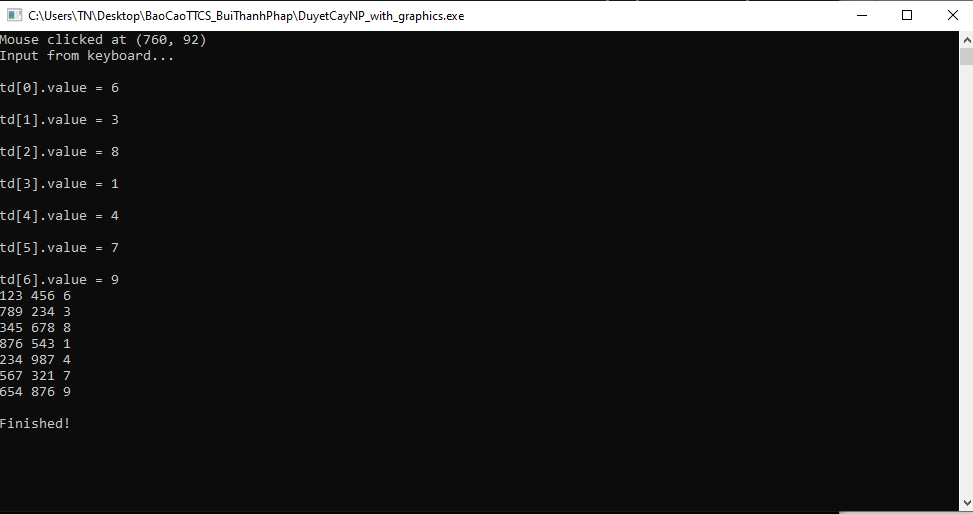
}

Đoạn code trên sẽ không yêu cầu người dùng nhập số lượng phần tử mà người dùng muốn vì nó số lượng phần tử đã được mặc định trong file là 7 phần tử. Người dùng chỉ cần nhập giá trị phần tử và giá trị phần tử sẽ được in ra kèm theo tọa độ.

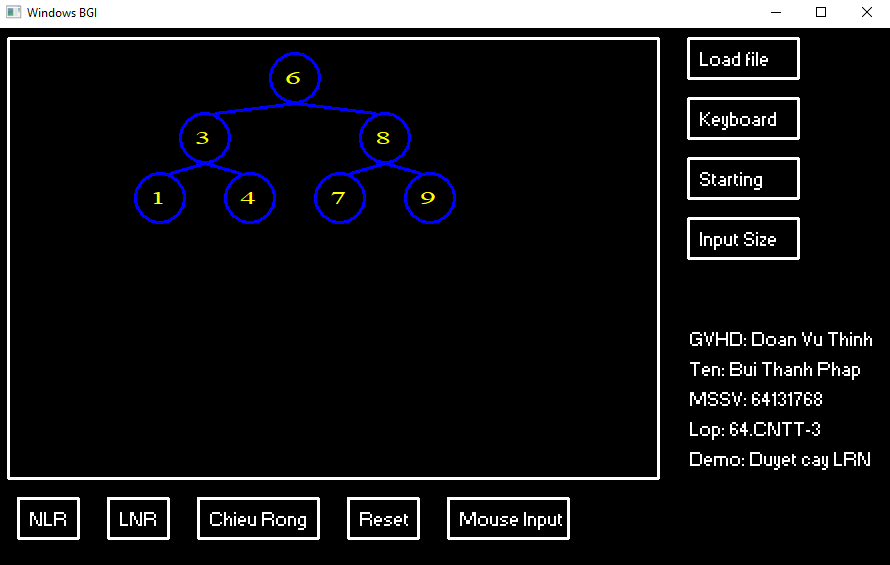


Hình 2.6 Hình ảnh số liệu trong file

Ví dụ : Người dùng nhập 7 giá trị phần tử: 6, 3, 8, 1, 4, 7, 9.



Hình 2.7 Hình ảnh khi người dùng thực hiện keyboard



Hình 2.8 Giao diện xuất hiện khi thực hiện keyboard

### **2.2.3. Sử dụng chuột để nhập dữ liệu**

// Kiem tra nut "Mouse Input"

if (x\_mouse > 450 && x\_mouse < 570 && y\_mouse > 470 && y\_mouse < 510) {

printf("Mouse Input button clicked!\n");

// Che do nhap du lieu bang chuot

printf("Click to input a number (0-9):\n");

// Can giua cac o so tu 0 den 9

int startX = 150; // Vi tri bat dau cua bang so (can giua man hinh)

int y\_pos = 300; // Toa do Y co dinh cua cac o so

// Ve cac o so tu 0 den 9

for (int i = 0; i < 10; i++) {

int x\_pos = startX + i \* 40; // Toa do X cua cac o so tu 0 den 9

rectangle(x\_pos, y\_pos, x\_pos + 40, y\_pos + 40);

char buffer[2];

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

outtextxy(x\_pos + 15, y\_pos + 15, buffer);

}

// Cho nguoi dung nhap vao cac o so da nhap

while (enteredNumbers < 7) { // Dung lai khi da nhap du 7 so

if (ismouseclick(WM\_LBUTTONDOWN)) {

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

for (int i = 0; i < 10; i++) {

int x\_pos = startX + i \* 40; // Toa do X cua cac o so tu 0 den 9

int y\_pos = 300; // Toa do Y co dinh cua cac o so

// Kiem tra neu chuot nhap vao o so

if (x\_mouse >= x\_pos && x\_mouse <= x\_pos + 40 &&

y\_mouse >= y\_pos && y\_mouse <= y\_pos + 40) {

printf("Number %d selected!\n", i);

// Them so vao cay nhi phan

td[sodinh++].value = i;

printf("Added number %d to tree\n", i)

// Ve lai cay sau khi them so

drawTree();

// Cap nhat so luong so da nhap

enteredNumbers++;

// Xoa hoac lam mo o so da chon

setfillstyle(SOLID\_FILL, BLACK); // To mau den (hoac mau khac) cho o so da

floodfill(x\_pos + 20, y\_pos + 20, WHITE); // Lam mo o so da chon

// Neu da nhap du 7 so, an bang so va quay lai giao dien chinh

if (enteredNumbers >= 7) {

// Xoa cac o so khoi man hinh

cleardevice();

// Ve lai cay voi cac so da nhap

drawTree();

GUI\_init();

break; // Thoat khoi vong lap nhap so

}

break; // Thoat vong lap kiem tra o so

}

}

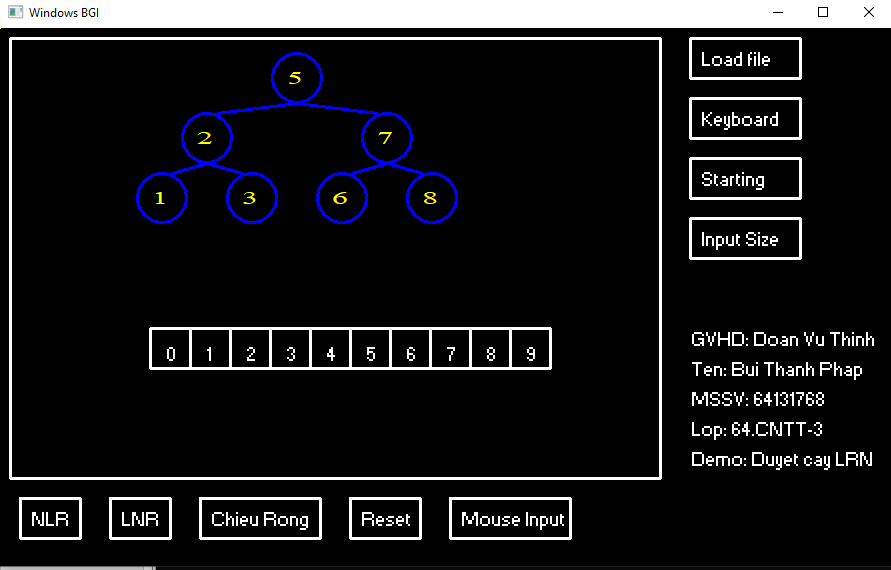
}

delay(100);

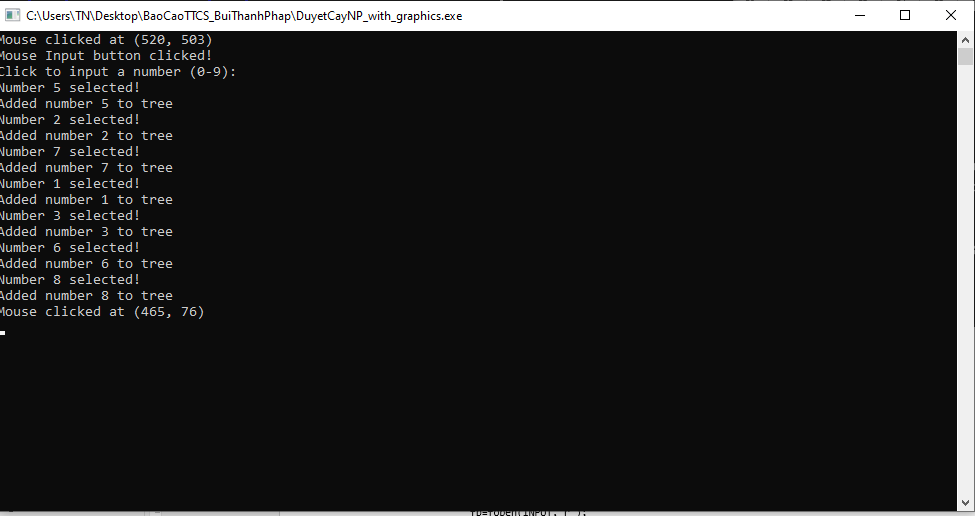
}

}

Đoạn mã trên xử lý chức năng nhập dữ liệu bằng chuột, cho phép người dùng nhấp chọn các ô số từ 0 đến 9 và thêm chúng vào cây nhị phân. Khi nút "Mouse Input" được nhấn (tọa độ xác định bởi x\_mouse > 450 && x\_mouse < 570 && y\_mouse > 470 && y\_mouse < 510), các ô số từ 0 đến 9 được vẽ ra màn hình dưới dạng các hình chữ nhật (tọa độ X được tính dựa trên startX và khoảng cách giữa các ô, với tọa độ Y cố định là 300). Trong vòng lặp, khi phát hiện nhấp chuột trái (WM\_LBUTTONDOWN), chương trình kiểm tra xem chuột có nằm trong tọa độ của bất kỳ ô số nào hay không. Nếu có, số đó được thêm vào cây nhị phân (dựa vào biến td[sodinh++].value), số lượng phần tử đã nhập được cập nhật, và cây được vẽ lại (drawTree()). Khi một số đã được chọn, ô số đó sẽ bị làm mờ hoặc xóa (sử dụng setfillstyle và floodfill). Vòng lặp tiếp tục cho đến khi nhập đủ 7 số, sau đó xóa các ô số khỏi màn hình (cleardevice()), quay lại giao diện chính (GUI\_init()), và thoát vòng lặp. Mã bao gồm kiểm tra tọa độ chuột để xử lý sự kiện chính xác và cập nhật giao diện sau mỗi lần chọn số.



Hình 2.9 Giao diện khi sử dụng nút Mouse Input



Hình 2.10. Khi nhấn vào số nó hiện bên console

### **2.2.4. Tạo các phần tử bằng cách đọc file**

//Ham doc lay thong tin cac diem

void readfile(){

fp=fopen(INPUT,"r");

if(fp==NULL){

printf("File not found");

return;

}

//else

//doc dong dau tien cua tap tin

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

printf("So phan tu can duyet: %d\n",sodinh);

//doc lan luot tung phan tu: moi ptu co 3 tham so: toado\_x, toado\_y, giatri

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

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

}

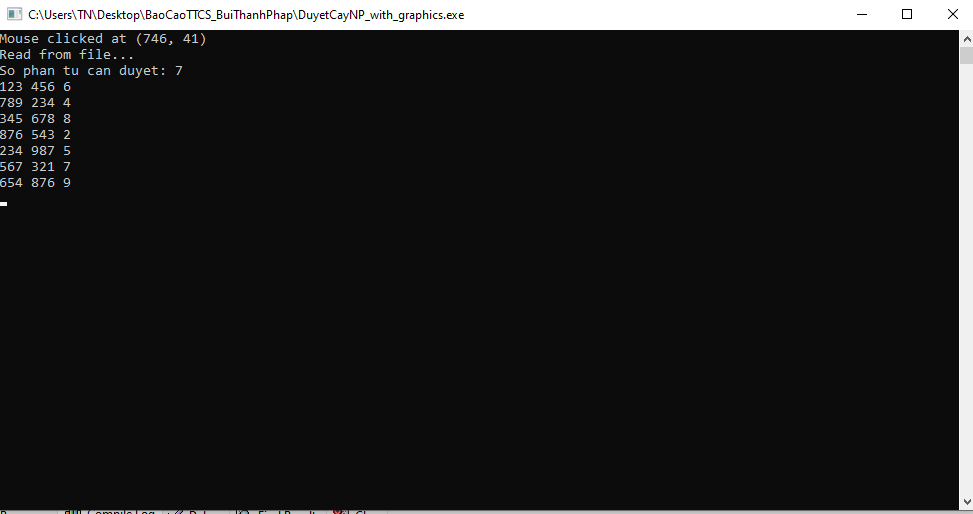
//in ra cac diem

printpoints(td,sodinh);

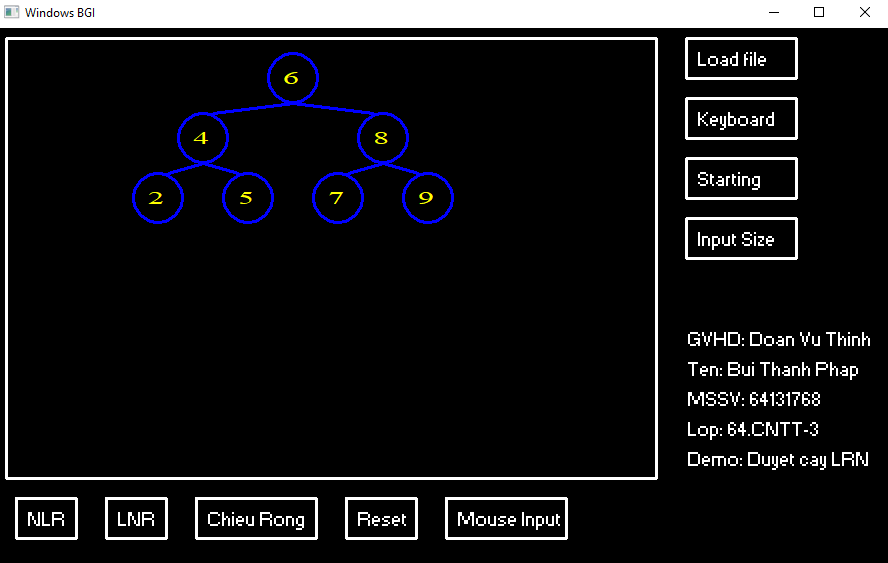
fclose(fp);

}

Hàm readfile đảm nhiệm việc đọc dữ liệu từ một tệp chứa thông tin các phần tử cần xử lý, với tên tệp được định nghĩa trước trong biến INPUT. Trước tiên, hàm mở tệp bằng fopen. Nếu tệp không tồn tại hoặc không thể mở, chương trình hiển thị thông báo "File not found" và thoát. Nếu mở thành công, dòng đầu tiên được đọc để xác định số lượng phần tử, lưu vào biến sodinh. Sau đó, vòng lặp được sử dụng để đọc thông tin từng phần tử, gồm tọa độ x, y và giá trị value, lưu vào mảng cấu trúc td. Dữ liệu này được sử dụng để hiển thị hoặc xử lý tiếp theo. Các điểm sau đó được in ra màn hình bằng hàm printpoints để xác nhận thông tin. Quá trình đọc đảm bảo tất cả các phần tử trong tệp được lưu trữ chính xác. Cuối cùng, tệp được đóng bằng fclose để giải phóng tài nguyên, đồng thời giảm thiểu nguy cơ rò rỉ tài nguyên hệ thống.



Hình 2.11 Hình ảnh tạo các phần từ bằng đọc file



Hình 2.12 Giao diện khi nhấn vào nút Load file

### **2.2.5. Thuật toán duyệt cây theo thứ tự LRN**

//cau truc cay nhi phan

struct Node {

int value;//gia tri nut

struct Node\* left;//con tro toi cay trai

struct Node\* right;//con tro toi cay phai

};

//ham them not nut vao cay nhi phan

struct Node\* insertNode(struct Node\* root, int value) {

if (root == NULL) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));//Tao nut moi

newNode->value = value;//gan gia tri cho nut moi

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

if (value < root->value) {

root->left = insertNode(root->left, value);//them vao cay con ben trai

} else {

root->right = insertNode(root->right, value);//them vao cay con ben phai

}

return root;

}

Cấu trúc cây nhị phân sử dụng struct Node gồm một giá trị value và hai con trỏ left, right trỏ đến cây con trái và phải. Hàm insertNode thực hiện thêm một nút mới vào cây: nếu cây rỗng (root == NULL), tạo một nút mới với giá trị value, trỏ cây con trái và phải về NULL, sau đó trả về nút này. Nếu value nhỏ hơn giá trị nút gốc, đệ quy thêm vào cây con trái, ngược lại thêm vào cây con phải, và cuối cùng trả về nút gốc sau khi thêm.

//ham ve cay nhi phan

void drawBinaryTree(struct Node\* root, int x, int y, int offset, int circleColor, int textColor) {

if (root == NULL) return;

setcolor(circleColor);

circle(x, y, 25);

setcolor(textColor);

settextstyle(1, 0, 1);

outtextxy(x - 10, y - 10, itoa(root->value, buffer, 10));

delay(500);

if (root->left != NULL) {

setcolor(circleColor);

line(x, y + 25, x - offset + 10, y + 35);

drawBinaryTree(root->left, x - offset, y + 60, offset / 2, circleColor, textColor);

}

if (root->right != NULL) {

setcolor(circleColor);

line(x, y + 25, x + offset - 10, y + 35);

drawBinaryTree(root->right, x + offset, y + 60, offset / 2, circleColor, textColor);

}

}

//ham ve cay nhi phan tu cac gia tri da nhap vao

void drawTree() {

struct Node\* root = NULL;//khoi tao cay rong

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

root = insertNode(root, td[i].value);

}

drawBinaryTree(root, getmaxx() / 3, 50, 90, BLUE, YELLOW);//getmax() lay chieu rong man hinh

}

Hàm drawBinaryTree là một hàm đệ quy dùng để vẽ cây nhị phân trên giao diện đồ họa. Hàm nhận vào các tham số gồm con trỏ root (gốc của cây), các tọa độ (x, y) cho vị trí vẽ, độ lệch offset để xác định khoảng cách giữa các nút con, và color để chọn màu sắc vẽ. Hàm vẽ một vòng tròn tại vị trí (x, y) với bán kính 25 và hiển thị giá trị của nút trong vòng tròn đó. Sau đó, nếu nút hiện tại có con trái, nó vẽ một đường nối từ nút cha đến nút con trái và đệ quy vẽ cây con trái. Tương tự, nếu có con phải, hàm vẽ đường nối đến con phải và đệ quy vẽ cây con phải. Việc vẽ cây diễn ra từ gốc đến các nhánh con (trái, phải), và với mỗi bước vẽ, có một độ trễ (delay(500)) giúp quá trình vẽ dễ quan sát.

Hàm drawTree khởi tạo cây nhị phân rỗng, thêm các giá trị từ mảng vào cây bằng cách gọi hàm insertNode, và cuối cùng gọi hàm drawBinaryTree để vẽ toàn bộ cây nhị phân từ gốc.

//Ham duyet cay theo thu tu sau va cac buoc

void postOrderTraversal(struct Node\* root, int x, int y, int offset, int\* step, int \*displayX, int displayY) {

if (root == NULL) return;

// Duyet cay con ben trai

if (root->left != NULL) {

postOrderTraversal(root->left, x - offset, y + 60, offset / 2, step, displayX, displayY);

}

// Duyet cay con ben phai

if (root->right != NULL) {

postOrderTraversal(root->right, x + offset, y + 60, offset / 2, step, displayX, displayY);

}

// Hien thi vong tron tai vi tri hien tai

if (\*step % 2 == 0) {

setcolor(6);

} else {

setcolor(3);

}

circle(x, y, 25);

settextstyle(1, 0, 1);

outtextxy(x - 10, y - 10, itoa(root->value, buffer, 10));

// Hien thi buoc trên console

printf("\nBuoc %d: Duyet nut %d\n", ++(\*step), root->value);

// Hien thi ket qua tai vung danh sach LRN

if (\*step % 2 == 0) {

setcolor(3);

} else {

setcolor(6);

}

circle(\*displayX, displayY, 25);

outtextxy(\*displayX - 7, displayY - 7, itoa(root->value, buffer, 10));

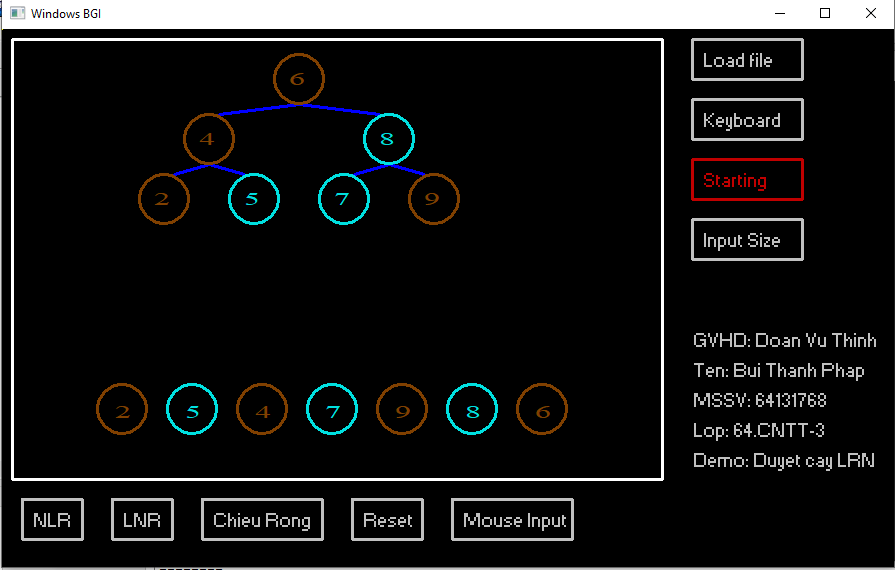
\*displayX += 70;

delay(1000);

mouse();

}

Hàm postOrderTraversal là một hàm đệ quy dùng để duyệt cây nhị phân theo thứ tự sau (Post-order Traversal), nghĩa là duyệt cây con bên trái trước, rồi đến cây con bên phải, và cuối cùng là xử lý nút gốc. Hàm nhận vào các tham số bao gồm con trỏ root (gốc của cây), các tọa độ (x, y) để vẽ nút, độ lệch offset để xác định khoảng cách giữa các nút con, và các biến step, displayX, displayY để theo dõi bước duyệt và vẽ kết quả trên màn hình. Hàm sẽ đệ quy duyệt cây con bên trái, sau đó là cây con bên phải. Sau khi duyệt xong các nút con, hàm vẽ vòng tròn tại vị trí hiện tại, hiển thị giá trị của nút trong vòng tròn đó, và in ra bước duyệt trên console. Cuối cùng, kết quả duyệt sẽ hiển thị trên màn hình.



Hình 2.13 Kết quả khi duyệt cây LRN

## **2.3. Các nút chức năng khác**

### **2.3.1. Nút NLR**

// Ham duyet cay theo thu tu truoc va cac buoc

void preOrderTraversal(struct Node\* root, int x, int y, int offset, int\* step, int\* displayX, int displayY) {

if (root == NULL) return;

// Duyet nut goc (root) truoc

if (\*step % 2 == 0) {

setcolor(12);

} else {

setcolor(10);

}

circle(x, y, 25);

settextstyle(1, 0, 1);

outtextxy(x - 10, y - 10, itoa(root->value, buffer, 10));

// Hien thi buoc tren console

printf("\nBuoc %d: Duyet nut %d\n", ++(\*step), root->value);

// Hien thi ket qua tai vung danh sach NLR

if (\*step % 2 == 0) {

setcolor(10);

} else {

setcolor(12);

}

circle(\*displayX, displayY, 25);

outtextxy(\*displayX - 7, displayY - 7, itoa(root->value, buffer, 10));

\*displayX += 70;

delay(1000);

// Duyet cay con ben trai

if (root->left != NULL) {

preOrderTraversal(root->left, x - offset, y + 60, offset / 2, step, displayX, displayY);

}

// Duyet cay con ben phai

if (root->right != NULL) {

preOrderTraversal(root->right, x + offset, y + 60, offset / 2, step, displayX, displayY);

}

}

Đoạn mã trên triển khai hàm duyệt cây theo thứ tự trước (NLR) với hiển thị đồ họa và in kết quả trên console. Hàm preOrderTraversal nhận vào cây nhị phân gốc root, tọa độ vẽ đồ họa ban đầu (x, y), khoảng cách offset, biến đếm bước step, tọa độ hiển thị danh sách NLR displayX, displayY. Nếu cây rỗng, hàm kết thúc. Bước đầu tiên là vẽ nút gốc tại vị trí (x, y) với màu thay đổi theo bước (đỏ/xanh), giá trị nút được in bằng hàm outtextxy. Tiếp theo, bước hiện tại và giá trị nút được in ra console. Nút này cũng được hiển thị trong danh sách NLR ở vị trí (\*displayX, displayY) và cập nhật tọa độ \*displayX. Sau đó, hàm gọi đệ quy duyệt cây con trái với tọa độ mới (x - offset, y + 60) và cây con phải với (x + offset, y + 60), mỗi lần giảm offset đi một nửa để đảm bảo khoảng cách vẽ phù hợp. Chức năng này sử dụng hiệu ứng đồ họa và tạm dừng 1 giây để trực quan hóa quá trình duyệt cây.

// Ham demo de duyet theo thu tu truoc

void demoNLR() {

enable\_click = 0;

GUI\_demoNLR();

struct Node\* root = NULL;

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

root = insertNode(root, td[i].value);

}

int displayX = 120;

int displayY = 380;

int step = 0;

printf("Duyet cay theo thu tu truoc (NLR): ");

preOrderTraversal(root, getmaxx() / 3, 50, 90, &step, &displayX, displayY);

printf("\n");

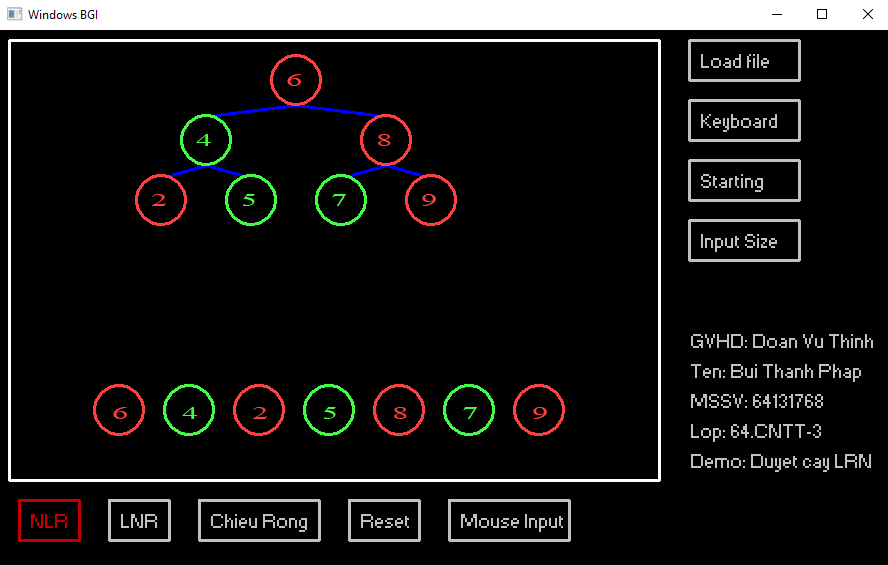
delay(1000);

enable\_click = 1;

mouse();

}

Hàm demoNLR là hàm demo thực hiện duyệt cây nhị phân theo thứ tự trước (NLR) với đồ họa trực quan, bắt đầu bằng việc vô hiệu hóa thao tác chuột để đảm bảo không có sự can thiệp của người dùng, sau đó gọi hàm GUI\_demoNLR() để thiết lập giao diện, khởi tạo cây nhị phân với gốc root bằng cách lặp qua mảng td và chèn từng giá trị vào cây bằng hàm insertNode. Các tham số hiển thị ban đầu được thiết lập gồm tọa độ displayX = 120, displayY = 380 cho danh sách NLR và step = 0 để đếm số bước duyệt. Dòng thông báo "Duyet cay theo thu tu truoc (NLR): " được in ra console trước khi gọi hàm preOrderTraversal để bắt đầu duyệt cây từ gốc, với các thông số hiển thị đồ họa gồm tọa độ (getmaxx() / 3, 50), khoảng cách offset = 90, cùng các biến tham chiếu step, displayX, displayY được cập nhật liên tục trong quá trình duyệt. Sau khi duyệt, chương trình dừng 1 giây để hiển thị kết quả cuối, kích hoạt lại thao tác chuột bằng enable\_click = 1, và gọi hàm mouse() để người dùng tiếp tục thao tác khác.



Hình 2.14 Kết quả khi duyệt cây NLR

### **2.3.2. Nút LNR**

// Ham duyet cay theo thu tu giua va cac buoc

void inOrderTraversal(struct Node\* root, int x, int y, int offset, int\* step, int\* displayX, int displayY) {

if (root == NULL) return;

// Duyet cay con ben trai

if (root->left != NULL) {

inOrderTraversal(root->left, x - offset, y + 60, offset / 2, step, displayX, displayY);

}

if (\*step % 2 == 0) {

setcolor(YELLOW);

} else {

setcolor(GREEN);

}

// Duyet nut goc (root) giua

circle(x, y, 25);

settextstyle(1, 0, 1);

outtextxy(x - 10, y - 10, itoa(root->value, buffer, 10));

// Hien thi buoc trên console

printf("\nBuoc %d: Duyet nut %d\n", ++(\*step), root->value);

if (\*step % 2 == 0) {

setcolor(GREEN);

} else {

setcolor(YELLOW);

}

circle(\*displayX, displayY, 25);

outtextxy(\*displayX - 7, displayY - 7, itoa(root->value, buffer, 10));

\*displayX += 70;

delay(1000);

// Duyet cay con ben phai

if (root->right != NULL) {

inOrderTraversal(root->right, x + offset, y + 60, offset / 2, step, displayX, displayY);

}

}

Hàm inOrderTraversal thực hiện duyệt cây nhị phân theo thứ tự giữa (LNR) với đồ họa trực quan, bắt đầu bằng việc kiểm tra nút gốc root, nếu cây rỗng thì thoát hàm, sau đó duyệt đệ quy cây con bên trái với tọa độ mới (x - offset, y + 60) và khoảng cách offset giảm một nửa, tiếp đến xử lý nút gốc bằng cách vẽ hình tròn tại vị trí (x, y) với màu thay đổi theo bước (vàng/xanh), hiển thị giá trị nút bằng outtextxy và in bước duyệt cùng giá trị lên console, sau đó hiển thị nút trong danh sách LNR tại tọa độ (\*displayX, displayY), cập nhật tọa độ \*displayX để chuẩn bị cho nút tiếp theo, cuối cùng duyệt đệ quy cây con bên phải với tọa độ mới (x + offset, y + 60) và hiệu ứng dừng 1 giây được thêm để trực quan hóa quá trình.

// Ham demo de duyet theo thu tu giua (LNR)

void demoLNR() {

enable\_click = 0;

GUI\_demoLNR();

struct Node\* root = NULL;

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

root = insertNode(root, td[i].value);

}

int displayX = 120;

int displayY = 380;

int step = 0;

printf("Duyet cay theo thu tu giua (LNR): ");

inOrderTraversal(root, getmaxx() / 3, 50, 90, &step, &displayX, displayY);

printf("\n");

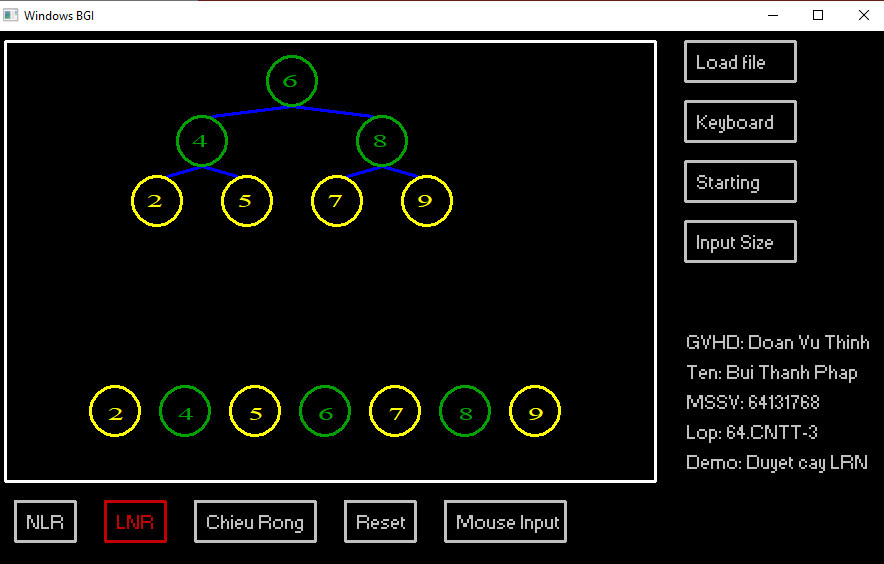
delay(1000);

enable\_click = 1;

mouse();

}

Hàm demoLNR thực hiện duyệt cây nhị phân theo thứ tự giữa (LNR) với đồ họa, bắt đầu bằng việc vô hiệu hóa thao tác chuột, thiết lập giao diện qua GUI\_demoLNR(), tạo cây từ mảng td bằng insertNode, khởi tạo tọa độ hiển thị displayX, displayY và đếm bước step. Sau đó, hàm in thông báo lên console, gọi inOrderTraversal để duyệt cây và hiển thị kết quả đồ họa. Cuối cùng, chương trình dừng 1 giây, kích hoạt lại thao tác chuột và gọi hàm mouse() để tiếp tục xử lý.



Hình 2.15 Kết quả khi chạy duyệt cây LNR

### **2.3.3. Nút Chiều Rộng**

// Ham duyet cay theo chieu rong

void levelOrderTraversal(struct Node\* root, int\* step, int\* displayX, int displayY) {

if (root == NULL) return;

// Khoi tao hang doi

std::queue<struct Node\*> q;

q.push(root);

// Duyet cay theo chieu rong

while (!q.empty()) {

struct Node\* current = q.front();

q.pop();

if (\*step % 2 == 0) {

setcolor(CYAN); // Mau do cho cac buoc chan

} else {

setcolor(MAGENTA); // Mau xanh cho cac buoc le

}

// Hien thi nut hien tai

circle(\*displayX, displayY, 25);

settextstyle(1, 0, 1);

outtextxy(\*displayX - 7, displayY - 7, itoa(current->value, buffer, 10));

\*displayX += 70;

// Hien thi buoc trên console

printf("\nBuoc %d: Duyet nut %d\n", ++(\*step), current->value);

// Duyet cac nut con

if (current->left != NULL) {

q.push(current->left);

}

if (current->right != NULL) {

q.push(current->right);

}

delay(1000);

}

}

Hàm levelOrderTraversal thực hiện duyệt cây nhị phân theo chiều rộng (Level Order Traversal) bằng cách sử dụng hàng đợi (queue). Nếu cây rỗng, hàm thoát ngay. Đầu tiên, hàm khởi tạo một hàng đợi và đẩy nút gốc root vào hàng đợi. Trong vòng lặp, nó lấy từng nút ra khỏi hàng đợi, hiển thị giá trị nút đó trên màn hình đồ họa tại tọa độ (\*displayX, displayY) với màu sắc thay đổi theo bước (xanh lơ cho bước chẵn, tím cho bước lẻ), và in thông tin nút đã duyệt ra console. Sau đó, nếu nút hiện tại có cây con bên trái hoặc phải, các nút con này sẽ được thêm vào hàng đợi để duyệt tiếp. Tọa độ \*displayX được cập nhật sau mỗi lần hiển thị để chuẩn bị cho nút tiếp theo. Vòng lặp tiếp tục cho đến khi hàng đợi rỗng, tức là toàn bộ cây đã được duyệt xong. Mỗi lần duyệt nút, hàm dừng 2 giây (delay(1000)) để hiển thị trực quan hơn.

// Ham demo de duyet theo chieu rong

void demoBFS() {

enable\_click = 0;

GUI\_demoBFS();

struct Node\* root = NULL;

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

root = insertNode(root, td[i].value);

}

int displayX = 120;

int displayY = 380;

int step = 0;

printf("Duyet cay theo chieu rong (BFS): ");

levelOrderTraversal(root, &step, &displayX, displayY);

printf("\n");

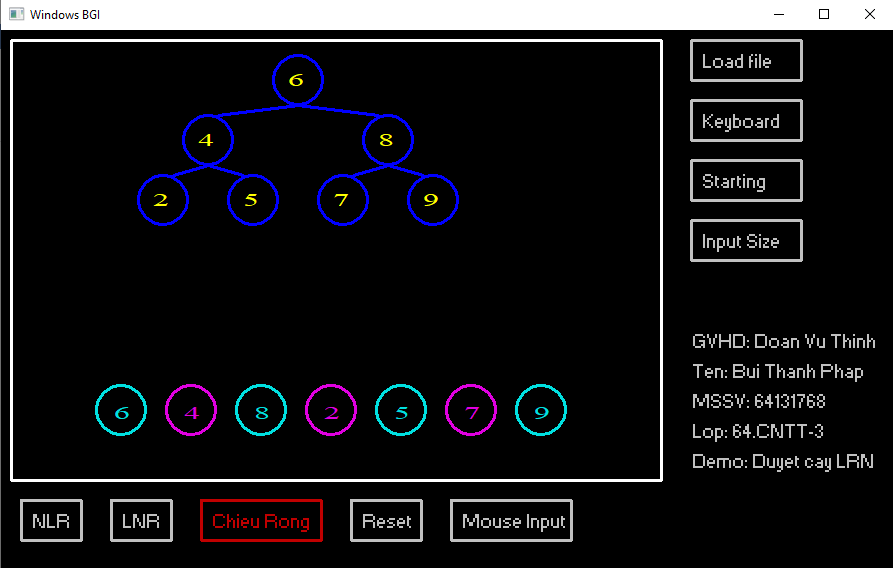
delay(1000);

enable\_click = 1;

mouse();

}

Hàm demoBFS thực hiện duyệt cây nhị phân theo chiều rộng (BFS) với hiển thị đồ họa. Đầu tiên, thao tác chuột được vô hiệu hóa (enable\_click = 0), sau đó giao diện được thiết lập qua GUI\_demoBFS(). Hàm khởi tạo cây nhị phân với gốc root = NULL và sử dụng hàm insertNode để chèn các giá trị từ mảng td vào cây. Các biến tọa độ hiển thị danh sách BFS được khởi tạo với displayX = 120 và displayY = 380, biến đếm bước duyệt step = 0. Sau đó, hàm in ra thông báo "Duyet cay theo chieu rong (BFS): " lên console và gọi levelOrderTraversal để duyệt cây, hiển thị đồ họa và thông tin các nút đã duyệt. Sau khi duyệt xong, chương trình dừng lại 1 giây để hiển thị kết quả cuối cùng, kích hoạt lại thao tác chuột (enable\_click = 1), và gọi hàm mouse() để xử lý thao tác tiếp theo.



Hình 2.16 Kết quả khi chạy duyệt cây theo chiều rộng

### **2.3.4. Nút Reset**

// Ham reset

void reset() {

cleardevice();

GUI\_init();

struct Node\* root = NULL;

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

root = insertNode(root, td[i].value);

}

drawBinaryTree(root, getmaxx() / 3, 50, 90, BLUE, YELLOW);

printf("Giao dien va cay da duoc lam moi.\n");

enable\_click = 1;

mouse();

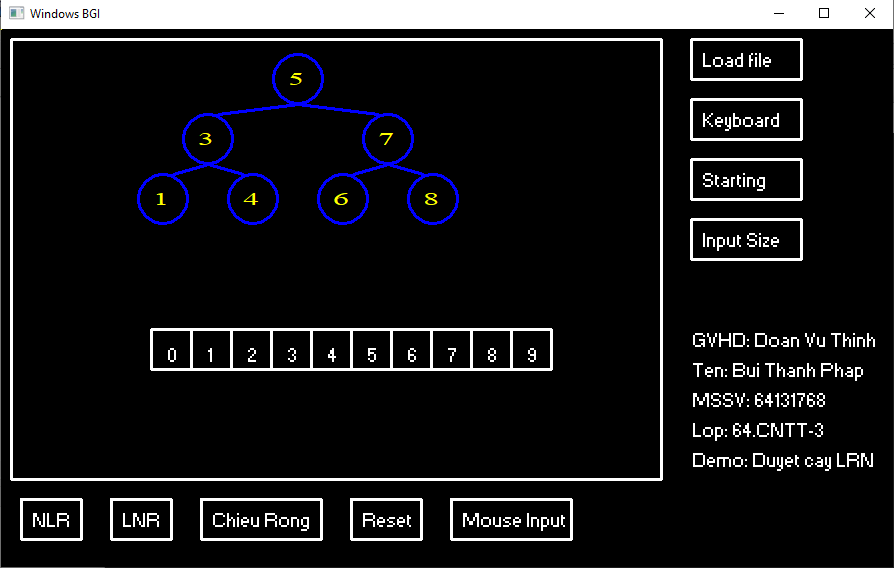
}

Hàm reset thiết lập lại giao diện và cây nhị phân về trạng thái ban đầu. Đầu tiên, hàm xóa toàn bộ nội dung màn hình hiện tại bằng cleardevice(), sau đó gọi GUI\_init() để khởi tạo lại giao diện. Tiếp theo, cây nhị phân được khởi tạo lại với gốc root = NULL và các nút được chèn từ mảng td bằng insertNode. Hàm drawBinaryTree được gọi để vẽ cây nhị phân trên giao diện, với gốc cây đặt tại tọa độ (getmaxx() / 3, 50), khoảng cách ban đầu giữa các nút là 90, và sử dụng màu xanh (BLUE) cho các cạnh và vàng (YELLOW) cho các nút. Sau khi làm mới xong, hàm in thông báo "Giao dien va cay da duoc lam moi." lên console, bật lại thao tác chuột bằng enable\_click = 1, và gọi mouse() để cho phép người dùng tiếp tục tương tác.

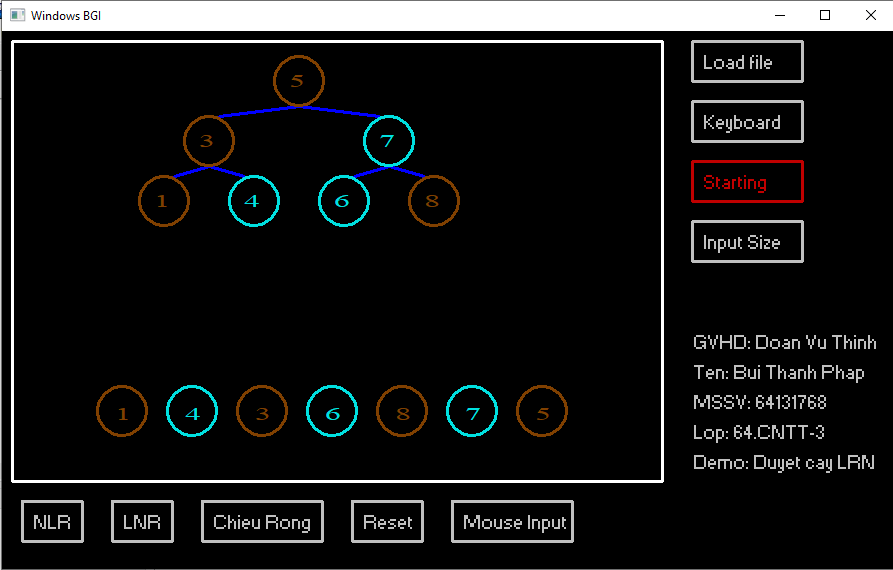
# CHƯƠNG 3. KẾT QUẢ THỰC HIỆN

Link YouTube: <https://www.youtube.com/watch?v=YjiZGrBD9vI&t=6s>

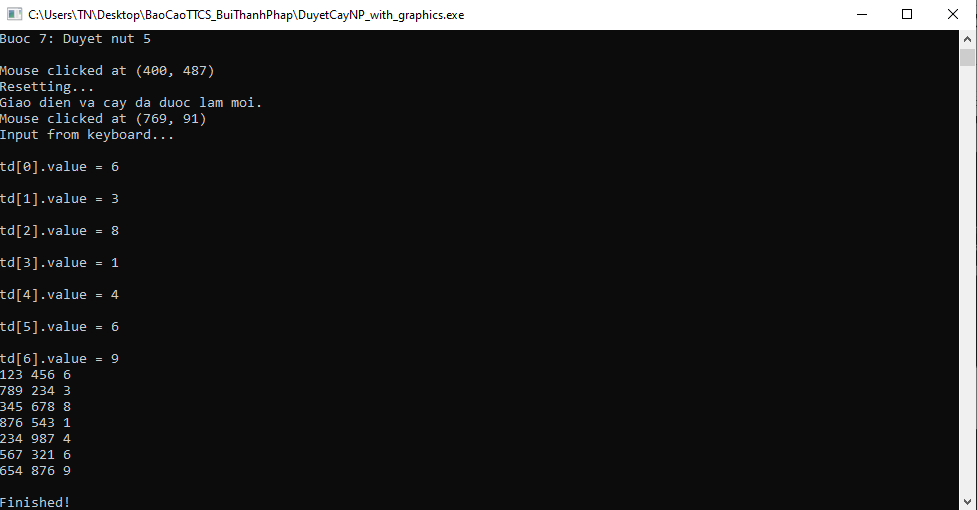
Link GitHub: <https://github.com/buithanhphap/DemoCayLRN>



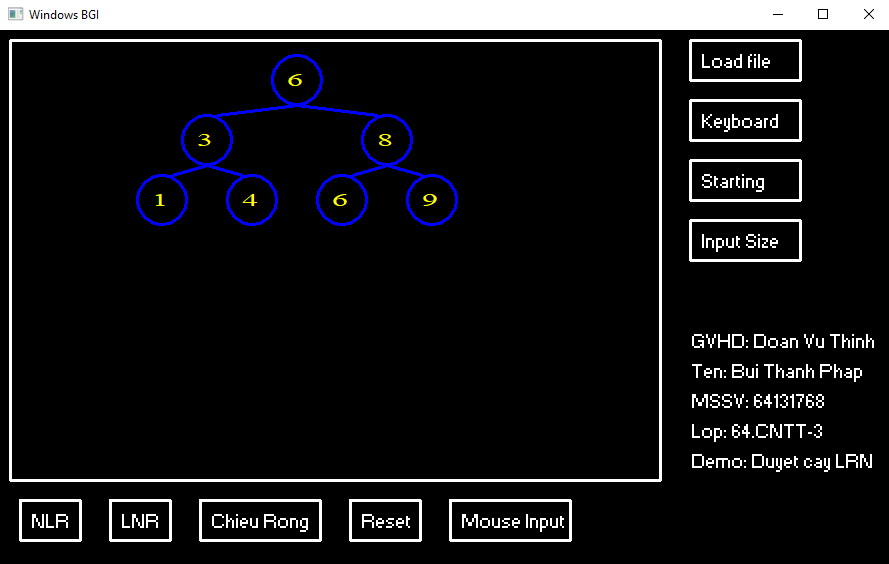
Hình 3.1 Giao diện khi nhấn nút Mouse Input



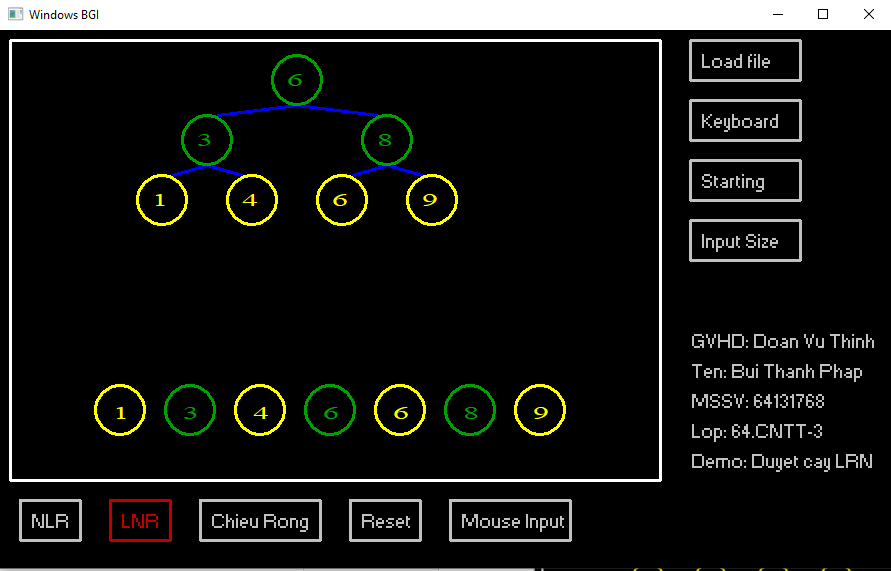
Hình 3.2 Hiện thị kết quả duyệt cây LRN khi nhấn nút Start



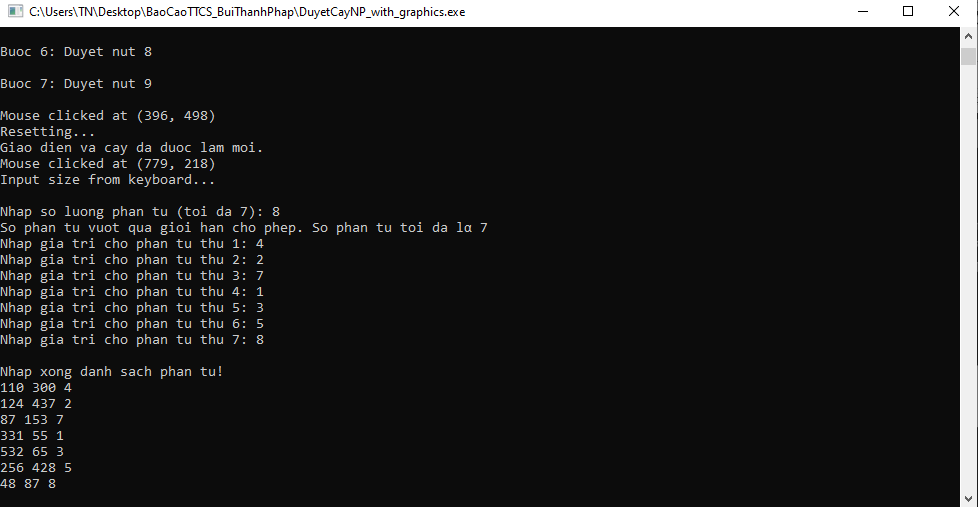
Hình 3.3 Hình ảnh khi nhấn vào Keyboard



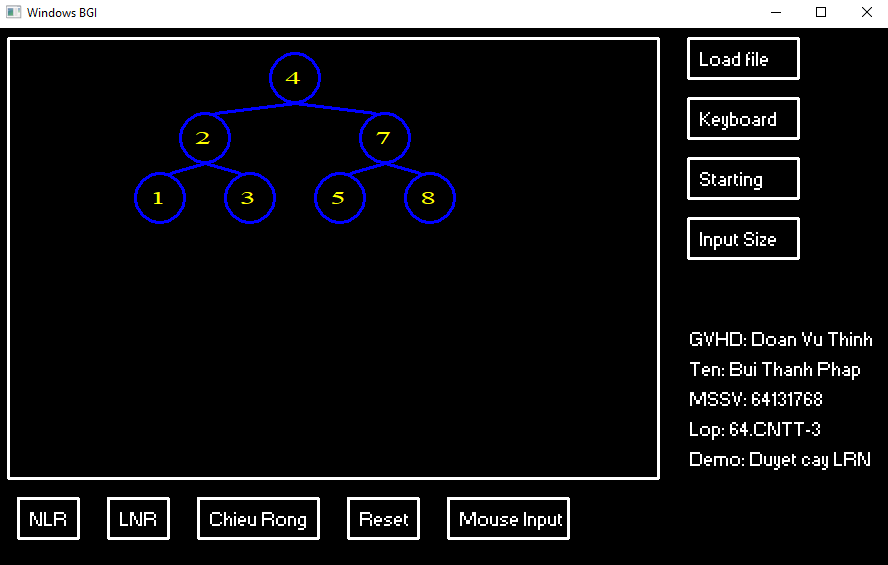
Hình 3.4 Giao diện từ việc nhập giá trị từ nút Keyboard



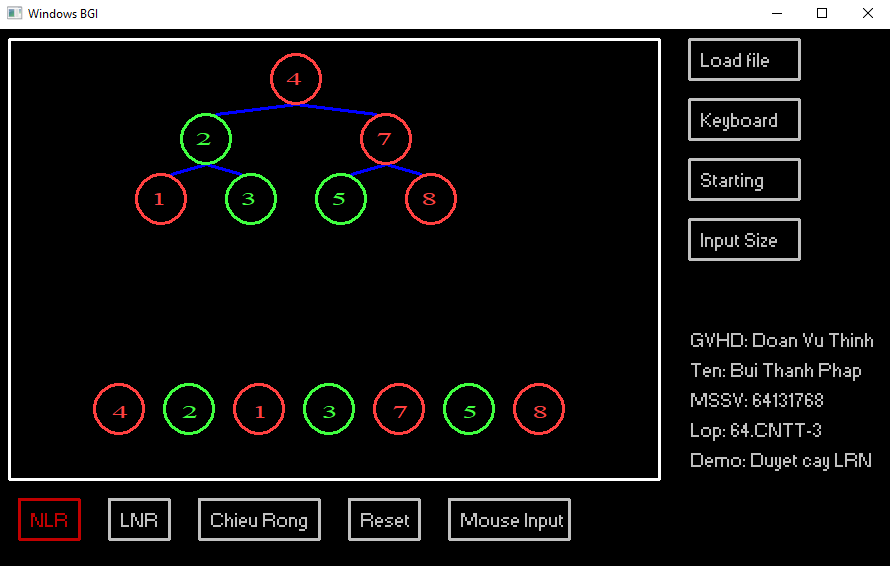
Hình 3.5 Hiển thị kết quả duyệt cây theo thứ tự giữa LNR



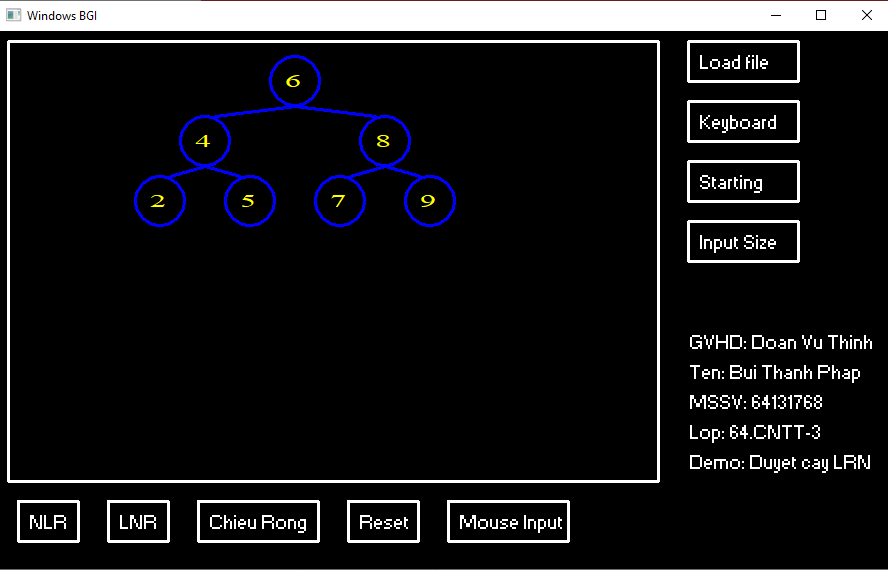
Hình 3.6 Hình ảnh khi nhấn nút Input Size và nhập giá trị



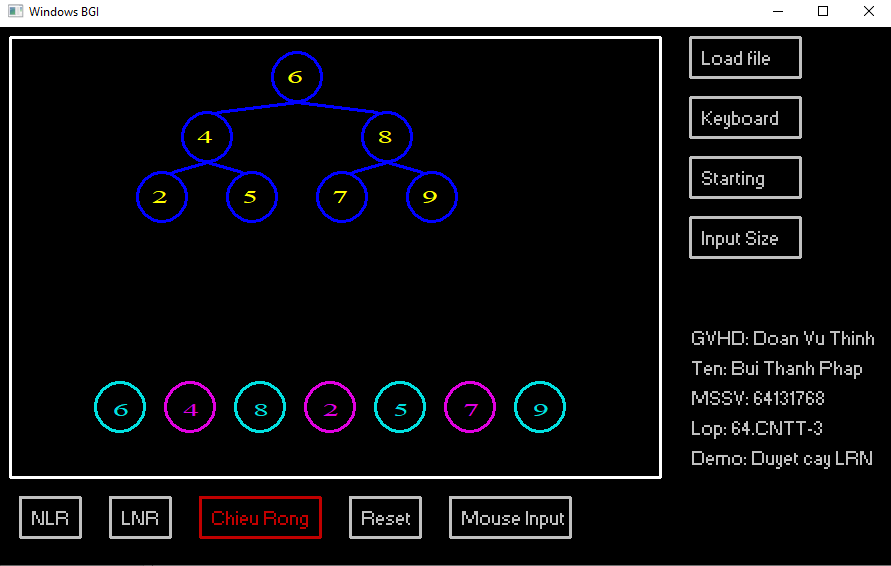
Hình 3.7 Giao diện hiển thị khi nhập giá trị từ nút Input Size



Hình 3.8 Hiển thị kết quả khi nhấn nút NLR



Hình 3.9 Giao diện khi nhấn nút Load file



Hình 3.10 Hiển thị kết quả khi nhấn nút Chiều Rộng

# CHƯƠNG 4. KẾT LUẬN

## **4.1. Ưu điểm**

Sau một thời gian tìm hiểu, nghiên cứu và lập trình, đề tài cũng đã được mô phỏng trên giao diện Dev-C++ với quá trình sinh và duyệt cây theo thứ tự sau, sử dụng công cụ đồ hoạ graphics.h. Giao diện được thiết kế để trình bày đầy đủ các chức năng và tiến trình duyệt cây như duyệt theo thứ tự trước, giữa, sau và chiều rộng. Người dùng có thể nhập dữ liệu từ bàn phím, sử dụng chuột. Các bước thực hiện được trình bày rõ ràng, cụ thể, dễ hiểu, giúp người dùng dễ dàng nắm bắt và theo dõi. Khi quá trình duyệt cây được kích hoạt, cây nhị phân sẽ được hiển thị trực quan trên giao diện, hỗ trợ người dùng quan sát và kiểm tra trật tự duyệt cây một cách trực quan và sinh động.

## **4.2. Nhược điểm**

Tuy nhiên, đề tài vẫn còn tồn tại một số nhược điểm cần khắc phục. Trong quá trình sinh và duyệt cây theo thứ tự sau, dùng xử lý reset để vẽ lại cây, nhưng điều này có thể gây hiện tượng nhấp nháy, ảnh hưởng đến trải nghiệm người dùng. Em sẽ tiếp tục nghiên cứu và cải thiện những hạn chế này trong các phiên bản tiếp theo để giao diện hoạt động mượt mà và hiệu quả hơn.

# 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. ThS Đoàn Vũ Thịnh (2023), "Bubble Sort with Graphics," YouTube video. Available at: <https://www.youtube.com/watch?v=cYSgYl3ixBQ>