ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN CUỐI KÌ

XÂY DỰNG WEBSITE TRỰC QUAN HÓA CẤU TRÚC DỮ LIỆU CÂY ĐỎ ĐEN

Môn học: Cấu trúc dữ liệu và giải thuật

Giảng viên: ThS. Nguyễn Thanh Sơn

Sinh viên thực hiện: Võ Thị Phương Anh - 21522883

Lớp: IT003.M21.KHTN

Thành phố Hồ Chí Minh, ngày 1 tháng 6 năm 2022

Mục lục

LỜI CẢM ƠN

Lời đầu tiên, em xin gửi lời cảm ơn đến thầy Nguyễn Thanh Sơn đã đồng hành xuyên suốt cùng em cũng như lớp KHTN2021 trong môn học Cấu trúc dữ liệu và giải thuật.

Sự tận tình và tạo điều kiện của thầy trong toàn bộ quá trình học là một trong những động lực lớn giúp em hoàn thành được đồ án này. Em mong nhận được những nhận xét của thầy về đồ án.

Trân trọng,

Võ Thị Phương Anh.

TỔNG QUAN VỀ ĐỒ ÁN

1. Giới thiệu về đồ án

Cấu trúc dữ liệu cây là một nội dung trong môn học Cấu trúc dữ liệu và giải thuật và là một cáu trúc dữ liệu được dùng khá phổ biến trong việc tổ chức các cấu trúc dữ liệu trong thực tế. Việc tổ chức dữ liệu trong cây được tạo ra với một logic tuần tự và mang đến sự tối ưu trong rất nhiều ứng dụng hiện đại Chính vì thế với mong muốn trực quan hóa việc xử lý và tổ chức các dữ liệu trong cây, tác giả xây dựng một trang web cho phép người dùng "trải nghiệm" cấu trúc dữ liệu cây, cụ thể là cây đỏ đen, với các thao tác cơ bản: thêm node, tìm kiếm và xóa node của cây.

2. Nền tảng kỹ thuật

2.1. Cấu trúc dữ liệu

- Cây Đỏ đen (Red-Black Tree).
- Cây Nhị phân tìm kiếm.

2.2. Ngôn ngữ lập trình

- Javascript: Xây dựng luồng xử lý back-end và tổ chức cấu trúc dữ liệu cây đỏ đen.
- HTML: Tổ chức các cấu trúc nền của trang web.
- CSS: Đồ họa web.

2.3. Hosting và Domain

Trang web có thể được deploy bằng hai phương thức:

- Chạy trên localhost.

- Chạy với tên miền rbtreevisualizeby21522883.netlify.app.

2.4. Source code và các tài liệu liên quan

Bên cạnh bản báo cáo, các tài liệu liên quan và source code của đồ án được đăng tải và mô tả cụ thể trên Github của tác giả.

Link project trên Github: https://github.com/UIT-21522883/rb-tree-visualize

3. Kết quả thực hiện

- Cài đặt và xây dựng được luồng xử lý cấu trúc dữ liệu cây đỏ đen bằng ngôn ngữ lập trình web.
- Hiện thực hóa được các thao tác cơ bản của cây đỏ đen (xoay cây tự cân bằng, duyệt cây) trên các nút tìm kiếm, thêm các node trên trang web.
- Thao tác xóa thực hiện được đối với các node lá có các node con là null, chưa thực hiện được với các node cha hoặc cao hơn.

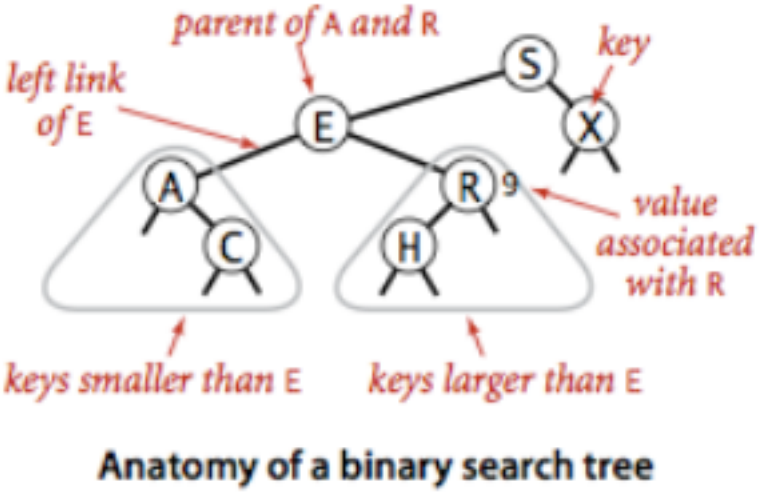
CƠ SỞ LÝ THUYẾT

1. Cây nhị phân tìm kiếm

Cây tìm kiếm nhị phân là cây nhị phân trong đó mỗi nút có một khóa (key) có thể so sánh được (và một giá trị được liên kết) và thỏa mãn điều kiện là khóa trong bất kỳ nút nào lớn hơn khóa trong tất cả các nút trong cây con bên trái của nút đó và nhỏ hơn các khóa trong tất cả các nút trong cây con bên phải của nút đó.

Tổ chức dữ liệu trong cây nhị phân tìm kiếm phải thỏa mãn các yêu cầu:

- Một node chỉ có thể có tối đa hai node con.
- Tất cả các nút của cây con bên trái đều nhỏ hơn nút gốc.
- Tất cả các nút của cây con bên phải đều hơn nút gốc.
- Các cây con của mỗi nút cũng là cây nhị phân tìm kiếm kế thừa các thuộc tính trên.



Anatomy of a binary search tree

Các phần tử trong một cây nhị phân tìm kiếm và các mối quan hệ giữa chúng. Nguồn ảnh: Princeton University.

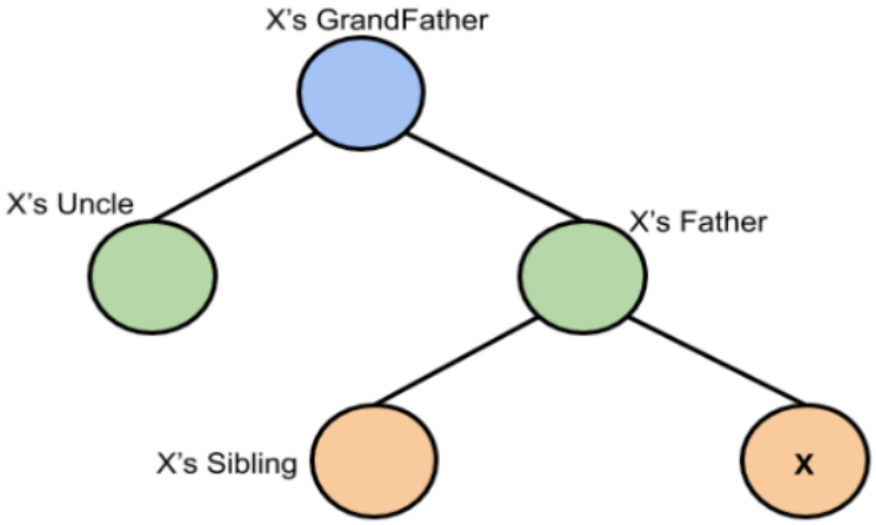
Cây nhị phân tìm kiếm được xây dựng dựa trên ý tưởng của thuật toán tìm kiếm nhị phân, cho phép duyệt tìm, chèn và loại bỏ các nodes nhanh chóng. Các thao tác cơ bản trên cây nhị phân tìm kiếm bao gồm: tạo cây nhị phân tìm kiếm, thêm các node, xóa node ra khỏi cây, duyệt cây, tìm node trong cây. Các thao tác này sẽ được nói cụ thể hơn ở phần cơ sở lý thuyết về cây đỏ đen, cấu trúc dữ liệu chính của đồ án.

2. Cây Đỏ Đen

Cây đỏ đen là một loại cây nhị phân tìm kiếm tự cân bằng trong đó mỗi nút có thêm một bit và bit đó được biểu diễn là màu đỏ hoặc đen. Hai màu này được sử dụng để đảm bảo rằng cây vẫn cân bằng trong quá trình chèn và xóa. Mặc dù sự cân bằng của cây không hoàn hảo, nhưng nó giúp giảm thời gian tìm kiếm và duy trì nó trong khoảng thời gian O(log n), trong đó n là tổng số phần tử trong cây. Cây đỏ đen được phát minh vào năm 1972 bởi Rudolf Bayer.

Nguyên tắc hoạt động của một cây đỏ đen:

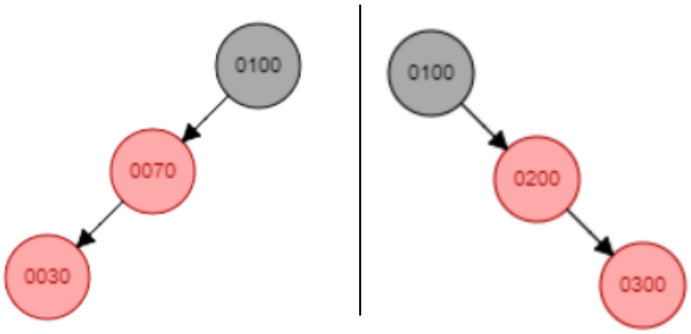
- Mỗi nút đều có màu đỏ hoặc đen.
- Nút gốc của cây luôn có màu đen.
- Không có hai nút màu đỏ liền kề (Một nút màu đỏ không thể có nút cha màu đỏ hoặc nút con màu đỏ).
- Mọi đường dẫn từ một nút (bao gồm cả gốc) đến bất kỳ nút NULL nào trong số các nút con của nó đều có cùng số lượng nút đen.
- Tất cả các nút lá đều là nút đen.



Cách gọi tên các nút trong cây đỏ đen. Nguồn ảnh: GeeksforGeeks.

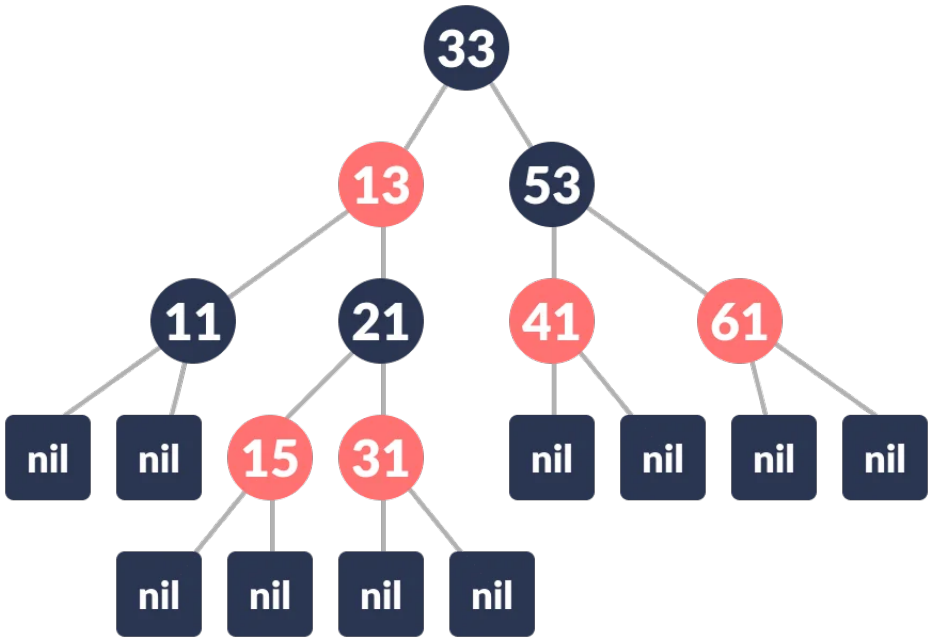
2.1. Cơ chế tự cân bằng trong cây đỏ đen

Cây đỏ đen mang thuộc tính của cây nhị phân tự cân bằng, và tính chất này thể hiện rõ nhất trong trường hợp thêm ít nhất hai nút đều nhỏ hơn hoặc lớn hơn nút đầu tiên được thêm vào. Điều này dẫn đến thứ tự ban đầu của các nút sẽ là lệch hoàn toàn sang trái hoặc sang phải. Tuy nhiên trong cây đỏ đen khi có trường hợp này xảy ra, cây sẽ bắt đầu quá trình tự cân bằng bằng cách chọn lại nút giữa là nút mang giá trị lớn hơn nút con nhỏ nhất và nhỏ hơn nút cha (ngoài cùng).



Ví dụ hai trong số các trường hợp cây chưa cân bằng và sẽ không được xem là một cây đỏ đen

Dựa theo cơ chế trên, cây đỏ đen sẽ hoàn toàn cân bằng trong trường hợp số nút là số lẻ. Đối với trường hợp số nút là số chẵn, cây sẽ lệch một nút sang bên trái hoặc phải.



Một cây đỏ đen hoàn chỉnh. Nguồn ảnh: Programmiz.

2.2. Các thao tác cơ bản trên cây đỏ đen

Các thao tác cơ bản đồng thời cũng là các thao tác được trực quan hóa trong đồ án này bao gồm: Khởi tạo cây, Thêm nút mới, tìm nút trong cây, xóa nút trong cây, duyệt nút theo chiều ngang.

2.2.1. Khởi tạo cây đỏ đen

Bước đầu tiên trong quá trình cài đặt cây đỏ đen chính là khởi tạo cây đỏ đen cùng với các thành phần. Các thành phần chính của cây được khởi tạo bao gồm:

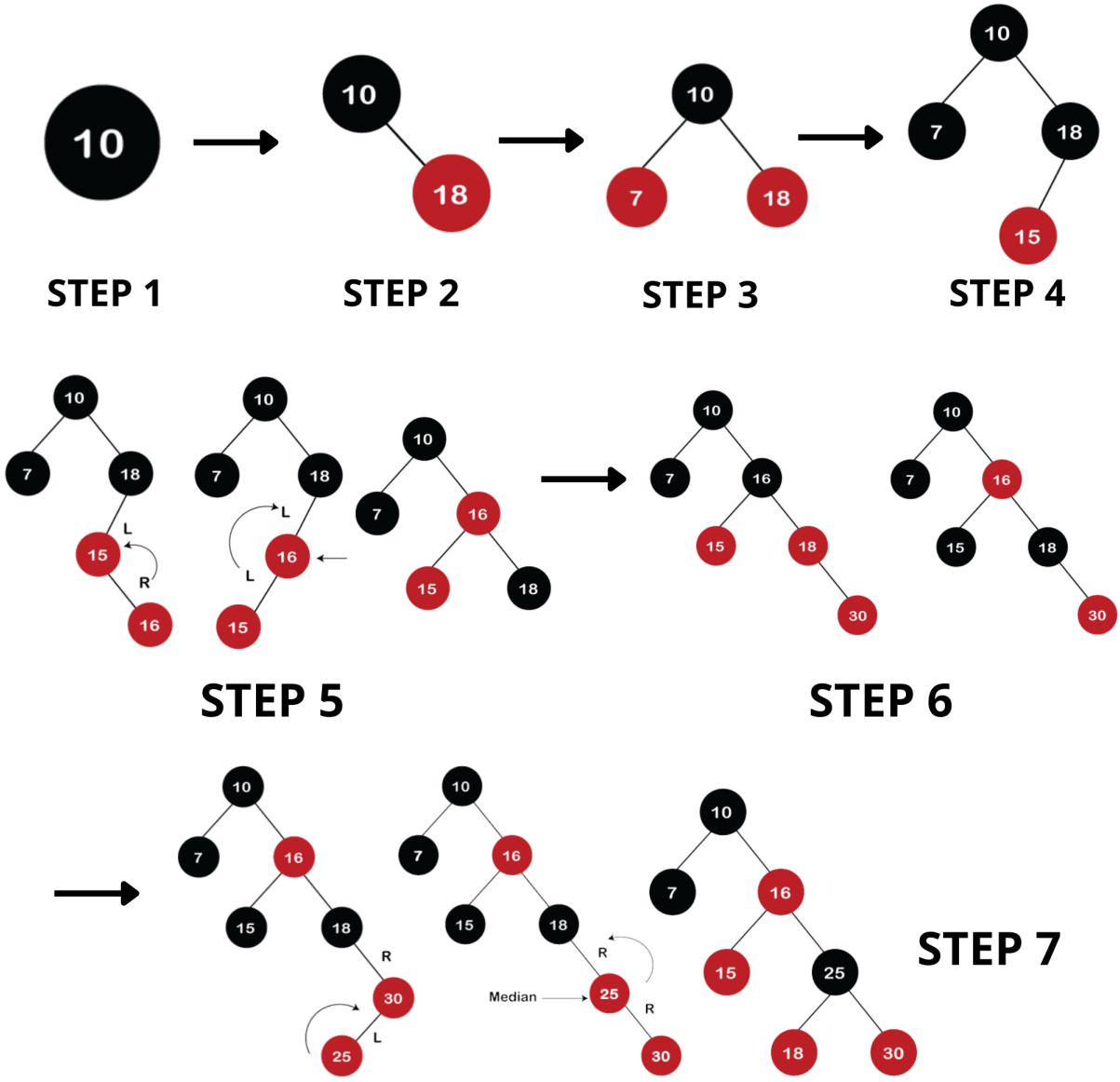
- Nút: bao gồm hai loại nút trống và nút có giá trị. Mỗi nút có giá trị (nhận từ user input). Mỗi nút sẽ có hai nút con và đối với mỗi nút con bậc thấp nhất

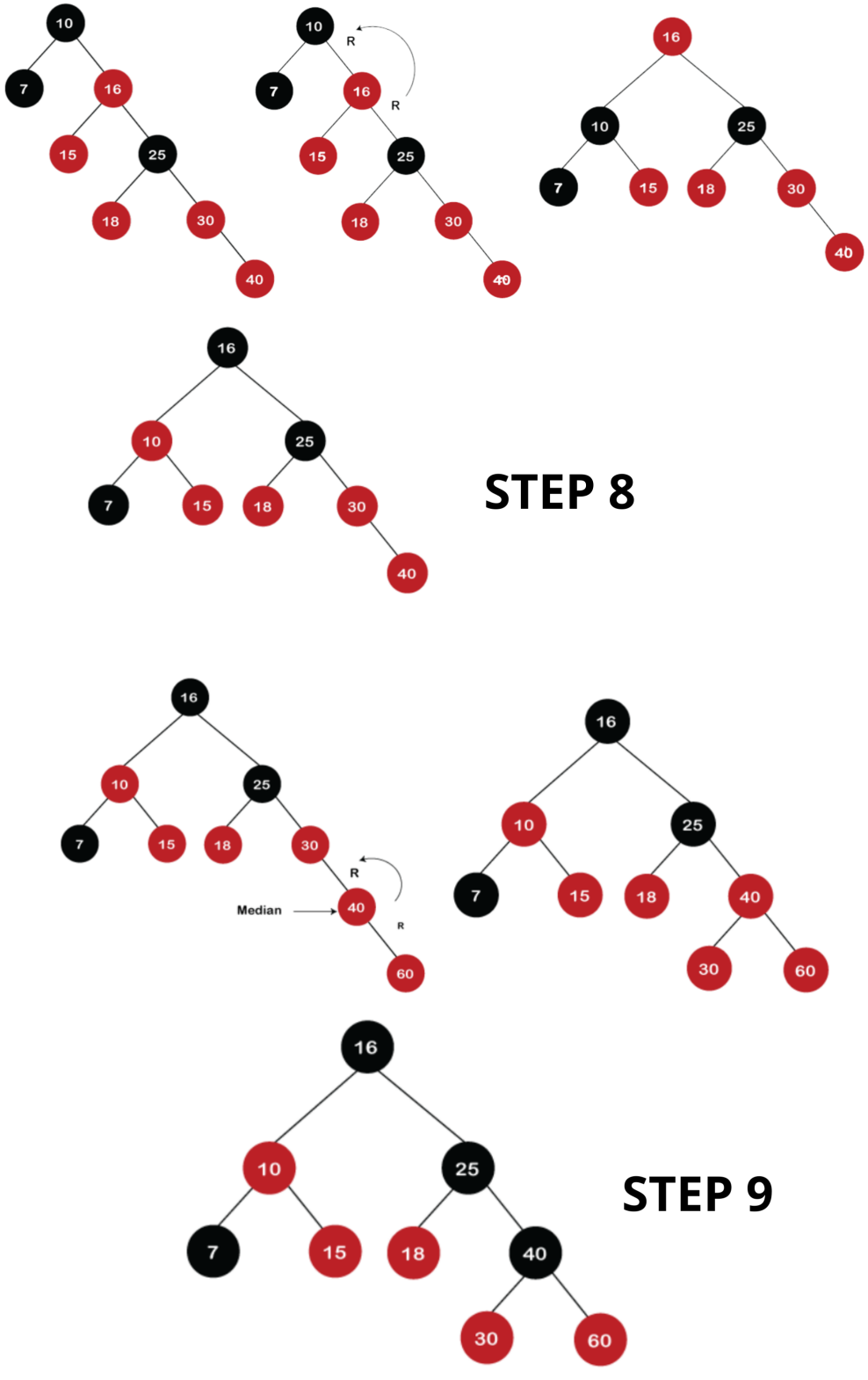
có giá trị sẽ có hai nút trống làm nút con của nó.

- Cây: bao gồm các nút và các thao tác hiển thị toàn cây.

2.2.2. Thêm nút cho cây

Trong sản phẩm cuối của đồ án là trang web trực quan hóa cấu trúc dữ liệu cây đỏ đen, các nút của cây sẽ nhận giá trị được nhập từ người dùng. Các giá trị của nút sẽ được so sánh và sắp xếp, tô màu theo đúng quy tắc tổ chức của cây đỏ đen được thể hiện cụ thể ở quy trình dưới.





Thêm các nút và xử lý xoay cây, đổi màu trong cây đỏ đen. Nguồn ảnh: Tổng hợp từ Javatpoint.

Các bước xử lý thêm nút vào cây đỏ đen (dựa trên các giá trị và thứ tự quy trình như ảnh trên):

- Bước 1: Thêm nút đầu tiên có giá trị là 10 vào cây. Nút đầu tiên sẽ đóng vai trò là nút gốc và luôn được biểu diễn bằng màu đen.
- Bước 2: Thêm nút thứ hai có giá trị là 18 vào cây.
  - Nút tiếp theo là 18. Vì 18 lớn hơn 10 nên nó sẽ được sắp xếp vào bên phải nút 10.
  - Theo quy tắc của cây đỏ đen, nếu cây không trống thì nút mới tạo sẽ có màu Đỏ. Do đó, nút 18 sẽ có màu đỏ.
  - Cuối cùng, ta xác minh nút cha của nút 18 có màu đen hay không. Trong quy trình trên, nút cha là nút 10 đã có màu đen. Do đó, đây đã đúng định dạng của cây đỏ đen.
- Bước 3: Thêm nút là một số nhỏ hơn nút giữa là 10 để tạo nhánh trái cho cây. Ở đây ta chọn 7 và vì 7 nhỏ hơn 10, nút 7 được di chuyển về bên trái của cây và có màu đỏ.
- Bước 4: Thêm nút có giá trị là 15 vào cây (lúc này có 3 nút 7, 10 và 18).
  - Vì 15 lớn hơn 10 nhưng nhỏ hơn 18, nút mới sẽ được tạo ở bên trái của nút 18. Nút 15 sẽ có màu Đỏ vì cây không trống.
  - Tuy nhiên, cây trên vi phạm tính chất của cây Đỏ - Đen vì nó có mối quan hệ nút cha mẹ - con là Đỏ - đỏ (thay vì Đen-Đỏ). Nếu nút cha của nút mới là Đỏ, thì ta phải kiểm tra màu của nút cha của nút mới. Nút mới là nút 15; nút cha của nút mới là nút 18 và nút anh em của nút cha là nút 7. Vì màu của nút cha là màu Đỏ, nên cây cần được tô màu lại cả nút 7 và 15 như thể hiện ở Step 4 của hình.

- Bước 5: Thêm một node nữa ở giữa khoảng [7;18], ở đây ta chọn 16. Tương tự với cách xử lý ở bước 4, vì 16 lớn hơn 10 và nhỏ hơn 18, nút 16 sẽ được đặt vào bên phải nút 15. Và một lần nữa, ta thấy cây trên vi phạm tính chất của cây đỏ đen, ta lại tiến hành xoay và đổi màu các nút.
  - Vì nút 16 ở bên phải của nút 15 và là cha của nút 15 là nút 18. Nút 15 là bên trái của nút 18. Ở đây chúng ta có một mối quan hệ LR, vì vậy cây cần thực hiện hai phép xoay.
  - Đầu tiên, chương trình sẽ thực hiện xoay cây sang trái, và sau đó là xoay phải. Việc xoay trái sẽ được thực hiện trên các nút 15 và 16, trong đó nút 16 sẽ di chuyển lên trên và nút 15 sẽ di chuyển xuống dưới.
  - Tuy nhiên, cây trên có xung khắc màu đỏ giữa hai nút liên tiếp nên ta thực hiện phép quay bên phải. Lúc này, nút 16 sẽ trở thành nút gốc, và các nút 15 và 18 sẽ là nút con bên trái và nút con bên phải.
  - Sau khi xoay hoàn tất, nút 16 sẽ được đổi sang màu đen và nút 18 sẽ được đổi thành màu đỏ.
- Bước 6: Thêm một nút có giá trị lớn hơn tất cả giá trị của nút trong cây. Ở đây ta chọn 30. Nút 30 lớn hơn phần tử lớn nhất trong cây lúc bấy giờ là 18, vì vậy nút 30 được thêm vào bên phải nút 18 với màu gốc là màu đỏ. Tuy nhiên, việc thêm nút 30 sẽ tạo ra một nhánh con mới trong cây và các nút ở bậc cha của nút 30 (nút 15, 18) cần được định dạng lại thành màu đen.
- Bước 7: Thêm một node có giá trị nhỏ hơn 30 nhưng lớn hơn hơn tất cả giá trị của các nút còn lại trong cây. Ở đây ta chọn 25. Vì 25 lớn hơn 10, 16, 18 nhưng nhỏ hơn 30; vì vậy, nó sẽ được thêm vào bên trái của nút 30. Vì cây không trống, nút 25 sẽ có màu Đỏ. Ở đây xung đột hai nút liền kề có màu

đỏ xảy ra giữa hai nút 25 và 30.

  - Xét thấy nút 30 không có anh chị em, thao tác cần thực hiện sẽ là xoay cây. Vì nút mới 25 được tạo ở bên trái nút cha 30 và nút cha ở bên phải nút cha của nó là 18, do đó mối quan hệ trái phải được hình thành. Đầu tiên, xoay phải được thực hiện trong đó nút 25 đi lên, trong khi nút 30 đi xuống.
  - Sau lần xoay đầu tiên, ta sẽ có một mối quan hệ phải-phải, do đó, phép quay trái được thực hiện. Sau khi xoay phải, phần tử trung vị, tức là, 25 sẽ là nút gốc; nút 30 sẽ ở bên phải của 25 và nút 18 sẽ ở bên trái của nút 25.
  - Cuối cùng, ta thực hiện đổi màu các nút 25 và 18; nút 25 có màu đen và nút 18 có màu đỏ.

- Bước 8: Ta tiếp tục "lặp lại" bước 6 là thêm một nút có giá trị lớn hơn tất cả giá trị của nút trong cây. Ở đây ta chọn 40. Vì 40 lớn hơn 10, 16, 18, 25 và 30 nên nút 40 sẽ ở bên phải nút 30. Vì cây không trống nên nút 40 sẽ có màu Đỏ. Ta dễ dàng nhận thấy ó xung đột Red-red giữa các nút liền kề 40 và 30 và nút 30 có các nút ngang bậc (anh chị em) với nó, vì thế ta thực hiện đổi màu.

  - Vì màu của nút cha mẹ và nút anh chị em của nút mới là Đỏ nên việc tô màu lại sẽ được thực hiện. Màu của cả hai nút sẽ trở thành màu đen. Tuy nhiên sau khi tô màu lại, ta cũng thấy rằng 25 không phải là nút gốc, do đó, việc đổi màu sẽ được thực hiện và màu của nút 25 chuyển thành đỏ.
  - Sau khi tô màu lại, xung đột màu đỏ xảy ra giữa hai nút 25 và 16. Bây giờ nút 25 sẽ được coi là nút mới. Vì nút cha của nút 25 có màu đỏ và

nút cha của nút cha có màu đen, và nút 25 nằm ở bên phải của nút 16 và 16 ở bên phải của nút cha của nó, do đó có một mối quan hệ phải-phải giữa các nút. Trong mối quan hệ này, quay trái được thực hiện. Sau khi xoay trái, phần tử trung vị 16 sẽ là nút gốc.

  - Sau khi xoay, việc đổi màu được thực hiện trên các nút 16 và 10. Màu của nút 10 và nút 16 lần lượt chuyển thành đỏ và đen.

- Bước 9: Một lần nữa, ta thêm một nút có giá trị lớn hơn tất cả giá trị của nút trong cây, cụ thể lần này là 60. Vì 60 lớn hơn 16, 25, 30 và 40, nên nút 60 sẽ ở bên phải nút 40. Vì cây không trống nên màu của nút 60 sẽ là Đỏ.

  - Như các bước 6 và 8, xung đột màu đỏ sẽ tiếp tục xảy ra. Nút cha có màu đỏ và không có nút cha nào tồn tại trong cây, vì vậy ta cần tiến hành xoay cây. Sau khi xoay cây lần đầu tiên sẽ ,mối quan hệ phải-phải tồn tại giữa các nút, vì vậy xoay trái sẽ được thực hiện.

  - Khi xoay trái được thực hiện, nút 40 sẽ đi lên trên và nút 30 sẽ đi xuống. Sau khi xoay, việc đổi màu được thực hiện trên các nút 30 và 40. Màu của nút 30 sẽ trở thành Đỏ, trong khi màu của nút 40 sẽ trở thành đen.

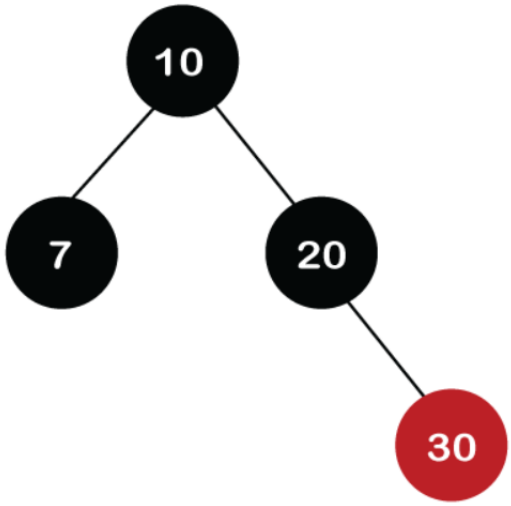
  - Đây là một ví dụ hoàn chỉnh về các thao tác thêm trên mọi trường hợp giá trị của nút.

2.2.3. Xóa nút trong cây

Việc xóa các nút trong cây kết hợp cả thao tác tìm kiếm, xoay và đổi màu các nút của cây sao cho phù hợp với thứ tự sau khi xóa một nút bất kỳ.

Việc xóa các nút trong cây kết sẽ đi từ đơn giản đến phức tạp dựa vào vị trí và bậc của nút đó trong cây đỏ đen được khởi tạo. Tại đây ta khái quát tất cả các trường hợp có thể xảy ra trong quá trình xóa nút.

- Trường hợp 1: Nút cần xóa là nút con có màu đỏ.



Ví dụ minh họa: Trong trường hợp 1, người dùng muốn xóa nút 30. Nguồn ảnh: Javatpoint.

Áp dụng cây nhị phân tìm kiếm để tìm nút 30 trong cây cho trước, ta thấy rằng 30 lớn hơn 10 và 20, có nghĩa là 30 là con bên phải của nút 20. Nút 30 là nút lá và có màu Đỏ, vì vậy nó sẽ được xóa khỏi cây một cách dễ dàng và không làm ảnh hưởng đến nút khác.

- Trường hợp 2: Xóa nút bên trong cây có nút cha là nút gốc và chỉ có một nút con. Ta dùng lại cây đỏ đen được ví dụ trong trường hợp 1 với yêu cầu mới là xóa nút 20 trong cây.

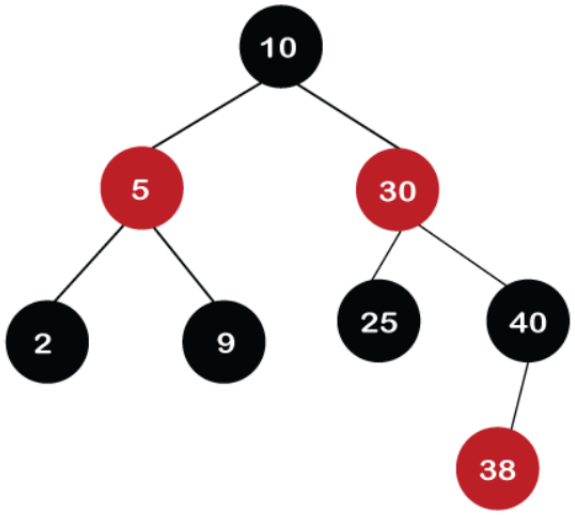
  Cách tiếp cận:

  – Ta không thể thực hiện xóa trực tiếp nút 20 ra khỏi cây vì nút 20 còn được bao bọc bởi các nút xung quanh vì thế ta chỉ có thể thay thế giá trị của nút đó bằng một giá trị khác.

  – Có thể thấy rằng nút 20 nằm ở bên phải của nút gốc và nó chỉ có một nút con là nút 30. Vì vậy, nút 20 được thay thế bằng một giá trị 30, nhưng

màu của nút sẽ được giữ nguyên, tức là màu đen. Cuối cùng, nút 20 (nút lá) bị xóa khỏi cây.

- Trường hợp 3: Xóa một nút có hai nút con Để xóa nút bên trong có hai nút con, việc cần làm là thay thế giá trị của nút bên trong (hoặc cây con bên trái hoặc cây con bên phải). Có hai cách tiếp cận cho vấn đề này:
  - Inorder predecessor: thay thế bằng giá trị lớn nhất tồn tại trong cây con bên trái.
  - Inorder successor: thay thế bằng giá trị nhỏ nhất tồn tại trong cây con bên phải.



Bài toán đặt ra: Xóa node 30 trong cây trên. Nguồn ảnh: Javatpoint.

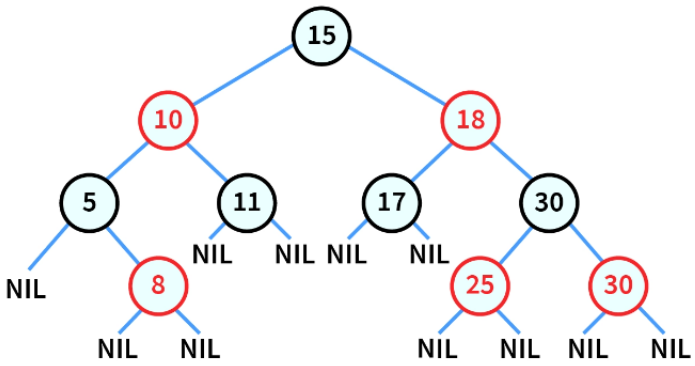
Phương pháp thực hiện: Nút 30 nằm ở bên phải của nút gốc. Trong trường hợp này,phương pháp được lựa chọn là Inorder successor. Giá trị 38 là giá trị nhỏ nhất trong cây con bên phải, vì vậy chúng tôi sẽ thay thế giá trị 30 bằng 38, nhưng nút sẽ giữ nguyên, tức là, màu đỏ. Sau khi thay thế, nút lá, tức là 30, sẽ bị xóa khỏi cây. Vì nút 30 là nút lá và có màu Đỏ, chúng ta cần xóa nó (chúng ta không phải thực hiện bất kỳ phép xoay hay tô màu nào).

•

2.2.4. Tìm kiếm phần tử trong cây

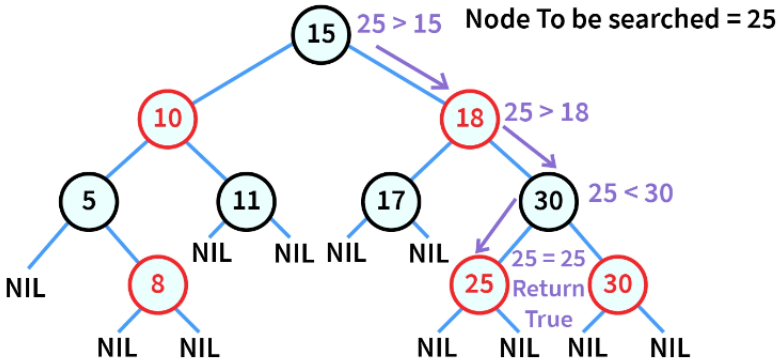
Kĩ thuật tìm kiếm trong cây đỏ đen hoàn toàn giống với tìm kiếm trên cây nhị phân tìm kiếm. Giải thuật tìm kiếm đó được mô tả như sau với nút đang được tìm là x:

- Bắt đầu từ nút gốc của cây đã cho.
- Nếu xx nhỏ hơn khóa của gốc thì đệ quy cho cây con bên trái, ngược lại nếu xx lớn hơn khóa của gốc thì đệ quy cho cây con bên phải.
- Nếu xx được tìm thấy ở bất kỳ đâu trong cây, trả về True và trả về False trong trường hợp còn lại.



Ví dụ cụ thể: Tìm nút với giá trị 25 trong cây trên. Nguồn ảnh: Scaler

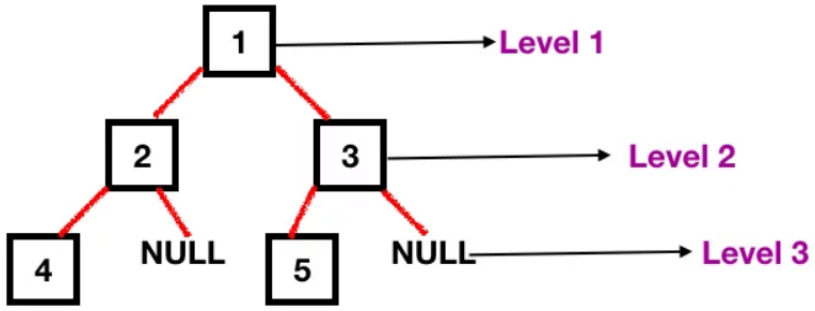
Giải thuật tìm kiếm trên cây sẽ được thực hiện tuần tự như hình sau:



Nguồn ảnh: Scaler

2.2.5.1 Tìm kiếm theo chiều ngang

Trong giải thuật được xây dựng trong đồ án này, tác giả sử dụng phương pháp duyệt cây theo chiều rộng (Breadth First Search). Thuật toán này sẽ vận hành chương trình theo cách đi qua tất cả các nút ở mỗi bậc trước khi chuyển sang cấp độ tiếp theo (độ sâu).



Thứ tự duyệt các nút trong Breadth First Search. Nguồn ảnh: Sơn Dương-VNTalking

3. Javascript

JavaScript là một ngôn ngữ lập trình mã nguồn mở được thiết kế để tạo các ứng dụng tập trung vào web. Nó có trọng lượng nhẹ và được thông dịch nên nhanh hơn nhiều so với các ngôn ngữ khác và được tích hợp với HTML giúp dễ dàng triển khai hơn trong các ứng dụng web.

JavaScript là một ngôn ngữ lập trình có kịch bản được sử dụng để tạo và quản lý các trang web động,và bất kỳ thứ gì di chuyển trên màn hình mà không yêu cầu người dùng làm mới trình duyệt của mình.

JavaScript- ngôn ngữ lập trình bậc cao dành chuyên cho phát triển web.

JavaScript cùng với HTML và CSS cùng nhau tạo thành xương sống của sự phát triển web.

4. HTML

HTML là một ngôn ngữ markup được sử dụng để tạo các phần của một trang web. HTML tổ chức các phần của trang web và quyết định cấu trúc trang web sẽ được hiển thị như thế nào.

HTML không phải là ngôn ngữ lập trình, nó chỉ giúp người dùng tạo và cấu trúc các thành phần trong trang web hoặc ứng dụng, phân chia các đoạn văn, heading, links, blockquotes... trong trang web.

5. CSS

CSS là viết tắt của Cascading Style Sheets, nó là một ngôn ngữ được sử dụng để tìm và định dạng lại các phần tử được tạo ra bởi các ngôn ngữ đánh dấu (HTML). Nói ngắn gọn hơn là ngôn ngữ tạo phong cách cho trang web. Nếu HTML đóng vai trò định dạng các phần tử trên website như việc tạo ra các đoạn văn bản, các tiêu đề, bảng,... thì CSS sẽ giúp chúng ta có thể thêm style vào các

phần tử HTML đó như đổi bố cục, màu sắc trang, đổi màu chữ, font chữ, thay đổi cấu trúc.

PHÁT TRIỂN WEB TRỰC QUAN HÓA CÂY ĐỎ ĐEN

Phần tài liệu kĩ thuật của đồ án được chia làm ba phần tương đương ba giai đoạn tương ứng với quá trình phát triển đồ án: Thiết kế các phần tử cơ bản của web với HTML, Lập trình các luồng xử lý của chương trình bằng Javascript, Tạo script trang trí cho trang web bằng CSS, Hiện thực hàm và cấu trúc dữ liệu cây đỏ đen cho trang web, Tiến hành chạy trên online site và localhost.

1. Thiết kế các phần tử cơ bản của web với HTML

Đầu tiên tác giả tiến hành thực hiện viết các lớp đối tượng tượng trưng cho các phần tử của trang web thực thi. Các phần tử này bao gồm:

- Tên trang web: Red Black Tree Visualization.
- Các ô lấy dữ liệu input người dùng.

Các phần tử cơ bản của trang web dựng bằng HTML.

2. Lập trình các luồng xử lý của chương trình bằng Javascript

Luồng xử lý back-end của chương trình được dựa trên việc cài đặt cấu trúc dữ liệu cây đỏ đen và các thuật toán liên quan bằng Javascript nhúng cùng với các thành phần của trang web như lấy dữ liệu nhập vào của người dùng.

```
class Node {
  constructor(options) {
    Object.assign(this, options);
    this.node = document.createElement('div');
    this.node.classList.add('node');
    this.node.textContent = this.value;
    this.node.style.backgroundColor = this.color;
    if (this.value === 'none') this.node.style.visibility = 'hidden';
    this.field.appendChild(this.node);
  }
}
export { Node };
```

Tạo đối tượng node kết nối với các thuộc tính đã tạo trong HTML.

```
class Tree {
  constructor(list) {
    this.list = list;
    this.leaves = [];
    this.nodes = [];

    this.leaves.push(this.list);
    let current = this.list;
    let idx = 0;

    while (idx < this.leaves.length) {
      if (!current) {
        this.leaves[idx] = null;
        current = this.leaves[++idx];
        continue;
      }
      if (current && current.left) {
        this.leaves[idx * 2 + 1] = current.left;
      } else {
        this.leaves[idx * 2 + 1] = null;
      }

      if (current && current.right) {
        this.leaves[idx * 2 + 2] = current.right;
      } else {
        this.leaves[idx * 2 + 2] = null;
      }
      current = this.leaves[++idx];
    }
  }
```

```
display() {
  for (const cnv of document.querySelectorAll('canvas')) {
    cnv.remove();
  }

  this.leaves.forEach((leaf, idx) => {
    let field;
    if (idx === 0) field = document.querySelector('.row.root');
    if (idx === 1 || idx === 2) field = document.querySelector('.row.one');
    if (idx > 2 && idx < 7) field = document.querySelector('.row.two');
    if (idx > 6 && idx < 15) field = document.querySelector('.row.three');
    if (idx > 14 && idx < 31) field = document.querySelector('.row.four');
    if (idx > 30 && idx < 63) field = document.querySelector('.row.five');
    if (idx > 62 && idx < 127) field = document.querySelector('.row.six');
    if (idx > 126 && idx < 255) field = document.querySelector('.row.seven');
    const { value } = leaf ? leaf : { value: 'none' };
    const color = !leaf || leaf.color === 1 ? 'red' : 'black';
    this.nodes[idx] = new Node({ value, color, field });
  });
  this.drawLines();
}
```

```
drawLines() {
  this.nodes.forEach((node, idx) => {
    if (node.value === 'none' || idx === 0) return;

    const rectNode = this.nodes[idx].node.getBoundingClientRect();
    const rectParent =
      this.nodes[Math.floor((idx - 1) / 2)].node.getBoundingClientRect();
    const sideLeft = rectNode.left < rectParent.left;

    const canvas = document.createElement('canvas');
    canvas.width = Math.abs(rectParent.left - rectNode.left);
    canvas.height = Math.abs(rectNode.top - rectParent.top);

    if (sideLeft) {
      canvas.style.left = `${rectNode.right - 20}px`;
    } else {
      canvas.style.left = `${rectParent.right - 20}px`;
    }

    canvas.style.top = `${rectParent.top + 20}px`;
    const ctx = canvas.getContext('2d');

    if (sideLeft) {
      ctx.moveTo(0, canvas.height);
      ctx.lineTo(canvas.width, 0);
    } else {
      ctx.moveTo(0, 0);
      ctx.lineTo(canvas.width, canvas.height);
    }

    ctx.lineWidth = 2;
    ctx.stroke();
    document.body.appendChild(canvas);
  });
}
```

Phác họa đối tượng Tree cho cây đỏ đen.

2.1. Cài đặt các chức năng của cây đỏ đen vào chương trình

Sau khi phát họa các đối tượng trong folder class, tác giả tiếp tục cài đặt các giải thuật bản chất của cây đỏ đen vào trong chương trình để tạo luồng xử lý cho các tác vụ được người dùng thực hiện trên web.

Sau khi phát họa các đối tượng trong folder class, tác giả tiếp tục cài đặt các giải thuật bản chất của cây đỏ đen vào trong chương trình để tạo luồng xử lý cho các tác vụ được người dùng thực hiện trên web.

helper > JS constants.js > ...

```js
const RED = 1;
const BLACK = 0;
const LEFT = 'left';
const RIGHT = 'right';

export { RED, BLACK, LEFT, RIGHT };
```

File constants trong folder helper lưu các thuộc tính cơ bản về màu sắc và quy ước của chúng trong chương trình.

helper > JS functions.js > cloneTree > forEach() callback

```js
import { LEFT, RIGHT } from './constants';

// Create a new deep copy
// Create a copy of the orginal object to prevent errors while editing

const cloneTree = (tree) => {
  const newTree = {};
  Object.keys(tree).forEach((key) => {
    if (tree[key] && typeof tree[key] === 'object') {
      if (tree === tree[key].parent) {
        newTree[key] = tree[key];
        newTree[key].parent = newTree;
      } else newTree[key] = cloneTree(tree[key]);
    } else {
      newTree[key] = tree[key];
    }
  });
  return newTree;
};

const getGrandFather = (node) => {
  const { parent } = node;
  const { parent: grandFather } = parent;
  return grandFather;
};

const getUncle = (node) => {
  const grandFather = getGrandFather(node);
  const { parent } = node;
  if (!grandFather) return null;
  if (parent === grandFather.left) {
    return grandFather.right;
  } else {
    return grandFather.left;
  }
};
```

```
const getSibling = (node) => {
  const { parent } = node;
  if (parent.left) {
    return parent.left;
  } else {
    return parent.right;
  }
};

const rotate = (node, direction) => {
  let anotherDir;
  if (direction === LEFT) {
    anotherDir = RIGHT;
  } else {
    anotherDir = LEFT;
  }
  const { parent } = node;
  const grandFather = getGrandFather(node);
  const temp = node[direction];

  node.parent = grandFather;

  if (grandFather) {
    let sideGrandfather;
    if (grandFather.left === parent) {
      sideGrandfather = LEFT;
    } else {
      sideGrandfather = RIGHT;
    }
    grandFather[sideGrandfather] = node;
  }
  node[direction] = parent;
  parent.parent = node;
  parent[anotherDir] = temp;
  if (temp) temp.parent = parent;
};
```

```
const buildTree = (node) => {
  while (node.parent) {
    node = node.parent;
  }
  return node;
};

export { cloneTree, getUncle, getGrandFather, getSibling, rotate, buildTree };
```

File functions trong folder helper cài đặt các thao tác cơ bản được thực hiện trên cây như xoay cây, tạo bảng copy (clone) của cây để tránh lỗi khi chỉnh sửa, lấy cái nút cha, con và các thao tác liên quan không bao gồm hai thao tác chính của người dùng thực hiện trên trang web là thêm và xóa node.

Đối với các thao tác chính trên trang web là thêm vào và xóa nút, hai thao tác này sẽ được tổ chức ở một file riêng và kết hợp với các thao tác đã lập trình trước đó.

helper > JS rbtree.js > add

```
1   import {
2     cloneTree,
3     getUncle,
4     getGrandFather,
5     getSibling,
6     rotate,
7     buildTree,
8   } from './functions';
9   import { RED, BLACK, LEFT, RIGHT } from './constants';
10
11  function add(tree, value) {
12    const newNode = {
13      value,
14      left: null,
15      right: null,
16      parent: null,
17      color: RED,
18    };
19
20    if (!tree) {
21      newNode.color = BLACK;
22      return newNode;
23    }
24
25    const resultTree = cloneTree(tree);
26
27    let parentNode;
28    let currentNode = resultTree;
29
30    // Insert node
31    while (currentNode) {
32      parentNode = currentNode;
33      if (currentNode.value === newNode.value) return resultTree;
34      if (newNode.value > currentNode.value) {
35        currentNode = currentNode.right;
36      } else {
37        currentNode = currentNode.left;
38      }
39    }
40
```

```
currentNode = newNode;
currentNode.parent = parentNode;
if (currentNode.value > parentNode.value) {
  parentNode.right = currentNode;
} else {
  parentNode.left = currentNode;
}

while (
  currentNode.color === RED &&
  currentNode.parent &&
  currentNode.parent.color === RED
) {
  const uncle = getUncle(currentNode);
  const grandFather = getGrandFather(currentNode);
  const { parent } = currentNode;

  if (uncle && uncle.color === RED) {
    parent.color = BLACK;
    uncle.color = BLACK;
    grandFather.color = RED;
    currentNode = grandFather;
    if (!currentNode.parent) currentNode.color = BLACK;
    continue;
  }

  if (!uncle || uncle.color === BLACK) {
    let sideOfTree, anotherDir;
    if (parent === grandFather.left) {
      sideOfTree = LEFT;
      anotherDir = RIGHT;
    } else {
      sideOfTree = RIGHT;
      anotherDir = LEFT;
    }
```

```
      if (parent[sideOfTree] === currentNode) {
        currentNode = parent;
        rotate(currentNode, anotherDir);
        currentNode.color = BLACK;
        currentNode[anotherDir].color = RED;
      } else {
        rotate(currentNode, sideOfTree);
        rotate(currentNode, anotherDir);
        currentNode.color = BLACK;
        currentNode[anotherDir].color = RED;
      }
    }
  }
  return buildTree(currentNode);
}

function remove(tree, value) {
  const resultTree = cloneTree(tree);
  let deletedNode = resultTree;

  while (value !== deletedNode.value) {
    if (value > deletedNode.value) {
      deletedNode = deletedNode.right;
    } else {
      deletedNode = deletedNode.left;
    }
    if (!deletedNode) return resultTree;
  }

  let maxInLeft = deletedNode;
  if (deletedNode.left) {
    maxInLeft = deletedNode.left;
    while (maxInLeft.right) {
      maxInLeft = maxInLeft.right;
    }
  }
```

```
deletedNode.value = maxInLeft.value;

const currentNode = maxInLeft;
const { parent } = currentNode;
let direction, anotherDir;
if (parent.left === currentNode) {
  direction = LEFT;
  anotherDir = RIGHT;
} else {
  direction = RIGHT;
  anotherDir = LEFT;
}

parent[direction] = null;

while (currentNode.color === 0 && currentNode.parent) {
  const { parent: currentParent } = currentNode;
  const sibling = getSibling(currentNode);
  const nephewDirection = sibling[direction];
  const nephewAnother = sibling[anotherDirection];

  if (!sibling) break;

  if (sibling.color === RED) {
    sibling.color = BLACK;
    currentParent.color = RED;
    rotate(sibling, direction);
    continue;
  }

  if (
    (!nephewDirection || nephewDirection.color === BLACK) &&
    (!nephewAnother || nephewAnother.color === BLACK)
  ) {
    sibling.color = RED;
    currentParent.color = BLACK;
    break;
  }
```

```
    if (
      (!nephewDirection || nephewDirection.color === BLACK) &&
      (!nephewAnother || nephewAnother.color === BLACK)
    ) {
      sibling.color = RED;
      currentParent.color = BLACK;
      break;
    }

    if (
      nephewDirection &&
      nephewDirection.color === RED &&
      (!nephewAnother || nephewAnother.color === BLACK)
    ) {
      sibling.color = RED;
      nephewDirection.color = BLACK;
      rotate(nephewDirection, anotherDirection);
      continue;
    }

    if (nephewAnother && nephewAnother.color === RED) {
      sibling.color = currentParent.color;
      currentParent.color = BLACK;
      nephewAnother.color = BLACK;
      rotate(sibling, direction);
      break;
    }
  }
  return buildTree(currentNode);
}

export { add, remove };
```

File rbtree trong folder helper cài đặt hai thao tác chính là thêm và xóa nút.

2.2. Tổ chức dữ liệu trong chương trình)

Đối với các đối tượng nút, cây và các giá trị nút được người dùng nhập vào sẽ được lưu trữ và xử lý khác nhau trong chương trình để thuận tiện với mục đích sử dụng, truy vấn.

- Đối với các giá trị nút người dùng nhập vào sẽ được lưu trữ trong một liên kết đơn (linked list)
- Đối với cây và nút sẽ được tổ chức bằng cách lưu vào hai mảng riêng biệt.

Lưu ý, đối với ngôn ngữ dùng cho luồng backend là Javascript, trong Javascript

không phân ra từng kiểu lưu trữ dữ liệu như trên mà tác giả chỉ thực hiện khai báo để tường minh hơn.

3. Tạo script trang trí cho trang web bằng CSS

Để trang trí và làm đẹp hơn cho trang web cũng như "tô màu" cho các thành phần đã được gọi trong HTML, tác giả viết một file script bằng CSS và nhúng các font chữ và màu sắc vào trang web. File script này cũng được cài đặt vào luồng chạy của chương trình trong file HTML.:

```css
:root {
  /* dark shades of primary color*/
  --clr-primary-1: hsl(22, 28%, 21%);
  --clr-primary-2: hsl(22, 28%, 29%);
  --clr-primary-3: hsl(22, 28%, 37%);
  --clr-primary-4: hsl(22, 28%, 45%);
  /* primary/main color */
  --clr-primary-5: hsl(22, 31%, 52%);
  /* lighter shades of primary color */
  --clr-primary-6: hsl(22, 31%, 60%);
  --clr-primary-7: hsl(22, 31%, 67%);
  --clr-primary-8: hsl(20, 31%, 74%);
  --clr-primary-9: hsl(22, 31%, 81%);
  --clr-primary-10: hsl(22, 31%, 88%);
  /* darkest grey - used for headings */
  --clr-grey-1: hsl(209, 61%, 16%);
  --clr-grey-2: hsl(211, 39%, 23%);
  --clr-grey-3: hsl(209, 34%, 30%);
  --clr-grey-4: hsl(209, 28%, 39%);
  /* grey used for paragraphs */
  --clr-grey-5: hsl(210, 22%, 49%);
  --clr-grey-6: hsl(209, 23%, 60%);
  --clr-grey-7: hsl(211, 27%, 70%);
  --clr-grey-8: hsl(210, 31%, 80%);
  --clr-grey-9: hsl(212, 33%, 89%);
  --clr-grey-10: hsl(210, 36%, 96%);
  --clr-white: #fff;
  --clr-red-dark: hsl(360, 67%, 44%);
  --clr-red-light: hsl(360, 71%, 66%);
  --clr-green-dark: hsl(125, 67%, 44%);
  --clr-green-light: hsl(125, 71%, 66%);
  --clr-black: #222;
  --transition: all 0.3s linear;
  --spacing: 0.1rem;
  --radius: 0.25rem;
  --light-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
  --dark-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
  --max-width: 1170px;
  --fixed-width: 620px;
}
```

Tạo các biến tên cho các màu để dễ dàng truy cập hơn khi thiết kế.

```css
body {
  /* font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, Oxygen,
    Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif; */
    font-family: 'Lucida Sans', 'Lucida Sans Regular', 'Lucida Grande', 'Lucida Sans Unicode', Geneva, Verdana, sans-serif  ;
  background: var(--clr-white);
  color: var(--clr-grey-1);
  line-height: 1.5;
  font-size: 0.875rem;
}

h1 {
  font-size: 2.5rem;
}

.header {
  height: 5rem;
  width: 90vw;
  margin: 0 auto;
  max-width: var(--max-width);
  display: flex;
  align-items: center;
  justify-content: space-between;
}

.logo {
  font-family: 'Courgette', cursive;
  color: var(--clr-primary-7);
}

.section-1 {
  background-color: var(--clr-primary-1);
}

.form_control {
  display: grid;
  grid-template-columns: 400px 400px;
  column-gap: 30px;
  margin-bottom: 1.25rem;
  padding: 1rem;
}
```

```css
.section-1 {
  background-color: var(--clr-primary-1);
}

.form_control {
  display: grid;
  grid-template-columns: 400px 400px;
  column-gap: 30px;
  margin-bottom: 1.25rem;
  padding: 1rem;
}

.search_input {
  padding: 0.5rem;
  background: var(--clr-grey-10);
  border-radius: var(--radius);
  border-color: var(--clr-grey-5);
  letter-spacing: var(--spacing);
  margin-right: 10px;
}

.btn {
  text-transform: uppercase;
  background: var(--clr-primary-5);
  color: var(--clr-primary-10);
  padding: 0.375rem 0.75rem;
  letter-spacing: var(--spacing);
  display: inline-block;
  font-weight: 400;
  transition: var(--transition);
  font-size: 0.875rem;
  cursor: pointer;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.2);
  border-radius: var(--radius);
  border-color: transparent;
}
.btn:hover {
  color: var(--clr-primary-1);
  background: var(--clr-primary-7);
}

.field {
  margin-top: 1.25rem;
  background-color: var(--clr-grey-8);
  padding: 1rem;
}
```

```css
.field {
  margin-top: 1.25rem;
  background-color: var(--clr-grey-8);
  padding: 1rem;
}

.message {
  height: 20px;
  padding: 5px 20px;
  color: red;
}

.node {
  width: 40px;
  height: 40px;
  box-sizing: border-box;
  border-radius: 50%;
  background-color: black;
  padding-top: 0.55rem;
  padding-left: 0.65rem;
  color: var(--clr-white);
  font-size: 1rem;
  z-index: 10;
}

.row {
  height: 5rem;
  display: flex;
  justify-content: space-around;
}

.root {
  justify-content: center;
}

canvas {
  position: absolute;
}
```

Thiết kế hình dạng cho các thành phần của trang web, phần còn lại của file style.css

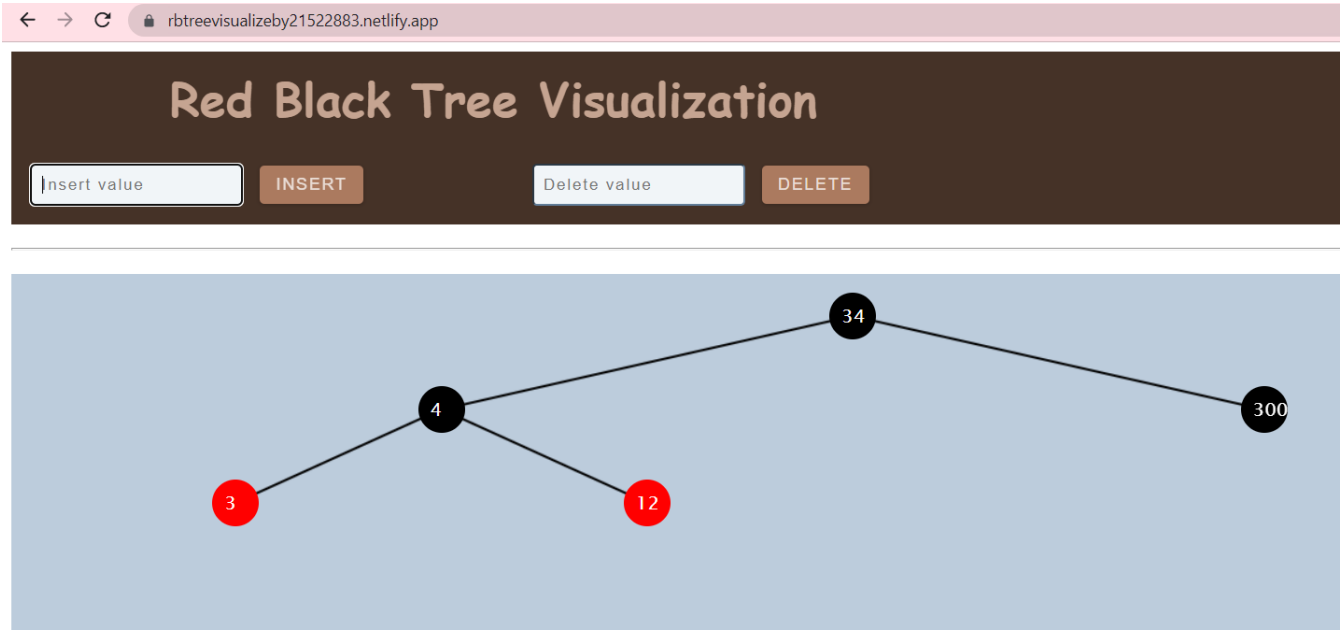
4. Deploy trang web trên localhost và web host với tên miền miễn phí

Sau khi xây dựng xong tất cả các phần của trang web, tiến hành deploy trang web trên hai nền tảng là localhost và webhost với tên miền được cung cấp miễn phí

4.1. Deploy trên webhost)

Đối với webhost, các bước thực hiện như sau:

- Bước 1: Đăng tải tất cả code lên Github.
- Bước 2: Truy cập vào trang web https://www.netlify.com/ để thực hiện deploy từ Github, trang web cũng cung cấp tên miền miễn phí là .netlify.app, đối với tên site người dùng được thay đổi tùy ý.
- Kết quả: Deploy thành công với link website là: https://rbtreevisualizeby21522883.netli

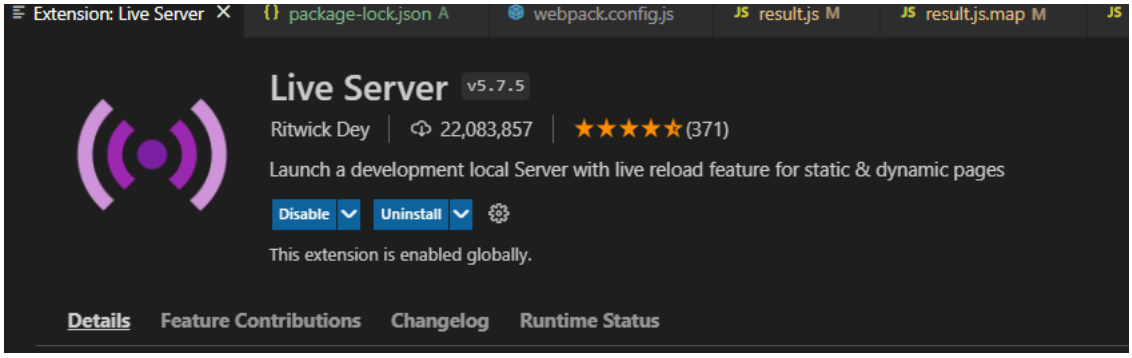


Giao diện web khi mở bằng trình duyệt.

4.2. Deploy trên localhost

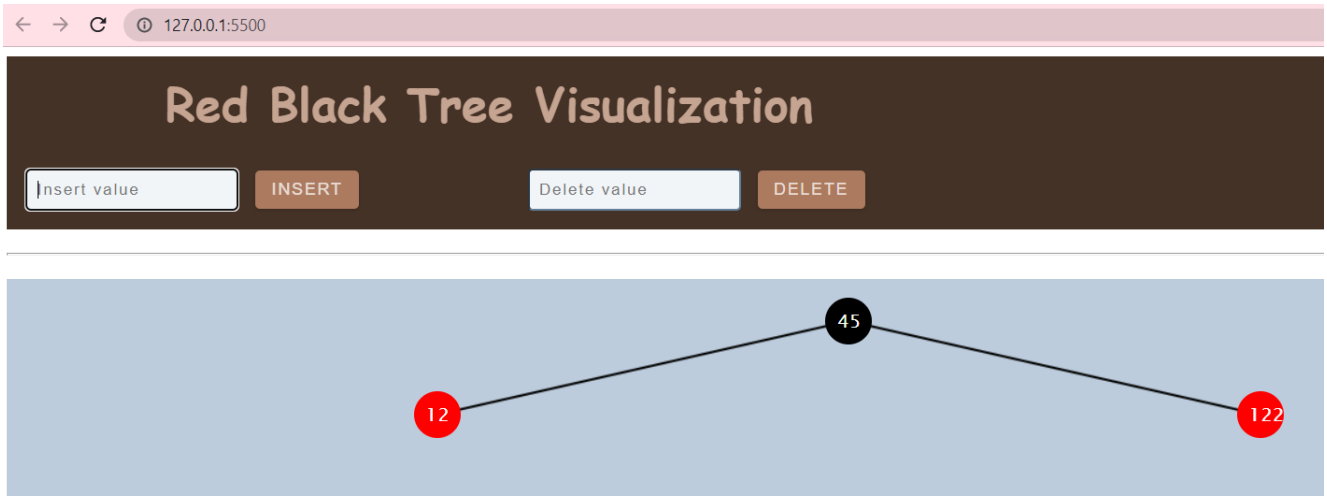
Đối với localhost, các bước thực hiện như sau:

- Bước 1: Sử dụng Visual Studio Code để chạy toàn bộ chương trình.
- Bước 2: Cài đặt một extension có tên là "Live Server" để cấp phát trên localhost từ code trong máy.



Live Server.

- Chạy lệnh npx webpack trên Terminal để gom tất cả các code.
- Nhấn Go Live để tiến hành chạy trên localhost.
- Kết quả: Deploy thành công trên localhost với cùng kết quả với link website là: https://rbtreevisualizeby21522883.netlify.app/.



Giao diện web trên localhost.

TỔNG KẾT

1. Tổng quát kết quả đạt được

Ưu điểm:

- Xây dựng được trang web trực quan hóa thuật toán cây đỏ đen với các thao tác cơ bản, giao diện người dùng thân thiện.
- Tổ chức và quản lý được cấu trúc dữ liệu trong chương trình khá ổn định.
- Bước đầu tiếp cận được với lập trình web bằng CSS, JS ,HTML.

Nhược điểm:

- Chưa hoàn thành đồ án đúng thời hạn.
- Các dependency của trang web còn khá cũ.
- Còn lỗi ở một số công đoạn (tác giả sẽ hoàn thiện trong thời gian ngắn nhất).

TÀI LIỆU THAM KHẢO

[1] An Introduction to JavaScript: Here Is All You Need to Know. Retrieved 20 May 2022, from https://www.simplilearn.com/tutorials/javascript-tutorial/introduction-to-javascript

[2] Binary Search Tree. Retrieved 30 May 2022, from https://www.programiz.com/dsa/bi search-tree

[3] CSS là gì? | TopDev. Retrieved 19 May 2022, from https://topdev.vn/blog/css-la-gi/

[4] Dương, S. (2021). Thuật toán Level Order Traversal (LOT) hay Breadth First Traversal (BFT) để duyệt tree - VNTALKING. Retrieved 29 May 2022, from https://vntalking.com/thuat-toan-level-order-traversal-lot-hay-breadth-first-traversal-bft-de-duyet-tree.html

[5] Parvez, F. (2020). Introduction to HTML: What is HTML and How it Works?. Retrieved 17 May 2022, from https://www.mygreatlearning.com/blog/html-tutorial/

[6] Red Black Tree - javatpoint. Retrieved 30 May 2022, from https://www.javatpoint.con black-tree

[7] Red-Black Tree | Set 1 (Introduction) - GeeksforGeeks. Retrieved 30 May 2022, from https://www.geeksforgeeks.org/red-black-tree-set-1-introduction

[8] Red/Black Tree Visualization. Retrieved 30 May 2022, from https://www.cs.usfca.edu

[9] Sedgewick, R., Wayne, K. (2011). Algorithms, 4th Edition.. Addison-Wesley. ISBN: 978-0-321-57351-3

[10] Yadav, P. (2022). Red Black Tree - Scaler Topics. Retrieved 30 May 2022, from https://www.scaler.com/topics/data-structures/red-black-tree/