BÁO CÁO ĐỒ ÁN MÔN HỌC
(Đồ án tìm hiểu CTDL/Giải thuật)
Lớp: IT003.O21.CNTT

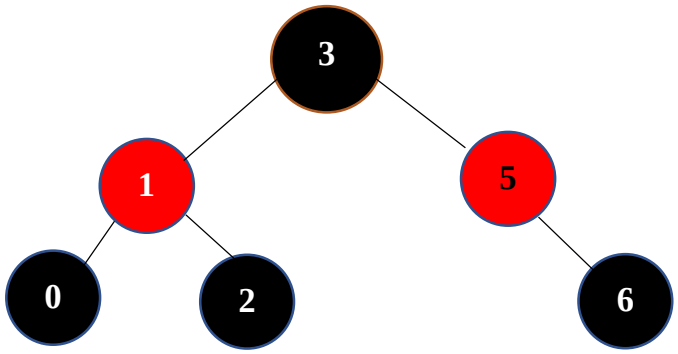
SINH VIÊN THỰC HIỆN

Mã sinh viên: 23520519 **Họ và tên: Lại Khánh Hoàng**

TÊN ĐỀ TÀI: CÂY ĐỎ ĐEN

CÁC NỘI DUNG CẦN BÁO CÁO: kết quả tìm hiểu được về CTDL/GT đã chọn

1. Giới thiệu đồ án: **Cây tìm kiếm nhị phân** (viết tắt tiếng Anh: BST - *Binary Search Tree*) là một cấu trúc dữ liệu rất thuận lợi cho bài toán tìm kiếm. Ta đã biết cây tìm kiếm nhị phân thông thường có những thuận lợi lớn về mặt lưu trữ và truy xuất dữ liệu trong phép toán tìm kiếm thêm vào hay loại bỏ một phần tử. Tuy nhiên nếu cây nhị phân không cân bằng, nó sẽ mất đi khả năng tìm kiếm nhanh (kể cả chèn và xóa) một phần tử. Trong đề tài này, em xin trình bày về một cách giải quyết vấn đề của cây không cân bằng, đó là cây đỏ đen – cây tìm kiếm nhị phân có thêm một vài đặc điểm



2. Quá trình thực hiện:
    a. Tuần 1: Tìm hiểu sơ qua về cây đỏ đen (định nghĩa, các tính chất nổi bật, các tính chất cơ sở)
    b. Tuần 2: Tìm hiểu các thư viện hỗ trợ cũng như ý nghĩa thực tế của cây đỏ đen, đồng thời trình bày đồ án
3. Kết quả đạt được
    a. Các định nghĩa/khái niệm cơ bản

    + **Cây tìm kiếm nhị phân (Binary Search Tree - BST)**: Là một cây trong đó mỗi nút có tối đa hai con và giá trị của nút con trái luôn nhỏ hơn hoặc bằng giá trị của nút cha, còn giá trị của nút con phải luôn lớn hơn hoặc bằng giá trị của nút cha.

    + **Cây tự cân bằng (Self-Balancing Tree)**: Là loại cây có các cơ chế tự điều chỉnh cấu trúc của nó sau mỗi thao tác (chèn, xóa,...) để duy trì thời gian tìm kiếm, chèn và xóa trong giới hạn logarit.

    + **Nút (Node)**: Mỗi phần tử trong cây gọi là một nút, bao gồm một giá trị và các con trỏ đến các nút con (trái và phải).

    + **Cây Đỏ-Đen:** là cây tìm kiếm nhị phân trong đó mỗi nút được tô màu đỏ hoặc đen. Nó là một loại cây tìm kiếm nhị có các thuộc tính sau:

        - Thuộc tính gốc: Node gốc có màu đen
        - Thuộc tính bên ngoài: Node lá có màu đen
        - Thuộc tính bên trong: Các node con của node đỏ có màu đen
        - Thuộc tính độ sâu: Các node lá đều có độ sâu màu đen như nhau
        - Thuộc tính đường dẫn : Mọi đường dẫn từ gốc đến một lá phải có cùng số lượng node đen

Do đó, từ các thuộc tính trên ta suy ra, cây đỏ đen là một cây nhị phân tìm kiếm tự cân bằng.

+ **Chiều cao đen (black-height)**: Chiều cao đen của một nút là số lượng nút đen từ nút đó đến một nút lá.

+ **Nút NIL:** có màu đen, được sử dụng để liên kết các nút cha với các nút con không tồn tại. Thay vì để liên kết này là null (trống), nó sẽ trỏ đến một nút NIL.

b. Các tính chất nổi bật

+ **Tính chất 1**: Nếu h là chiều cao của cây đỏ đen thì chiều cao đen $\geq \frac{h}{2}$.

Chứng minh: *Nếu chiều cao đen* $< \frac{h}{2}$*, khi đó số nút đỏ sẽ lớn hơn* $\frac{h}{2}$*, tuy nhiên do nút gốc và nút lá đều màu đen, số nút đỏ sẽ phải bé hơn* $h - 2$*. Theo nguyên lí Dirichlet, sẽ có hai nút đỏ nằm sát nhau, hay có một nút đỏ là cha của nút đỏ còn lại, vô lí với thuộc tính của cây đỏ đen*

+ **Tính chất 2**: Mỗi cây đỏ đen với n node sẽ có chiều cao không vượt quá $2\log_2(n+1)$.

Chứng minh: *Đối với mỗi cây đỏ đen, gọi h là chiều cao của cây. Khi đó,* $n \geq 2^h - 1$*, hay* $h \leq log_2(n+1)$*. Do đó với mỗi node lá, chiều cao đen của nó không vượt quá* $log_2(n+1)$*. Mà theo tính chất 1, ta dễ thấy số node đỏ sẽ bé hơn số node đen với mọi đường đi từ node gốc đến node lá. Từ đó ta có điều phải chứng minh.*

+ **Tính chất 3:** Cây đỏ đen là một cây nhị phân tìm kiếm tự cân bằng

Chứng minh: *Cây đỏ đen là một cây tìm kiếm nhị phân như định nghĩa.Ngoài ra theo 5 quy tắc mà cây đỏ đen tuân thủ, quy tắc 5 đảm bảo rằng mọi đường dẫn từ một nút đến các nút lá của nó đều chứa cùng số lượng nút đen. Điều này ngăn không cho một nhánh của cây trở nên quá sâu so với các nhánh khác và theo tính chất 2, chiều cao của cây bị giới hạn bởi chiều cao đen. Do đó cây đỏ đen cũng có tính chất tự cân bằng.*

+ **Tính chất 4:** Cấu trúc của một node trong cây đỏ đen bao gồm:

- Trường INFO (KEY): chứa thông tin của nút
- Trường Liên kết Trái (LEFT): Chứa liên kết con trỏ tới nút con trái

- Trường Liên kết phải (Right): Chứa liên kết con trỏ tới nút con phải
- Trường Liên kết cha (PARENT): Chứa liên kết con trỏ tới nút cha
- Trường màu (COLOR): Lưu trữ màu sắc của nút (đỏ, đen)

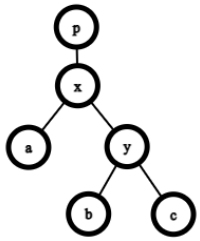
c. Các thao tác cơ sở (nếu là CTDL)

i) Thêm vào 1 node:

Trong quá trình thêm vào 1 node mới có thể vị phạm các thuộc tính của cây đỏ đen, để khắc phục, người ta thường dùng thao tác quay.

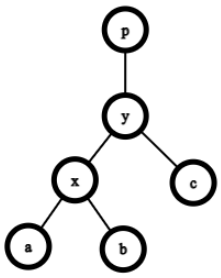
+ Xoay trái:

Giả sử ta có cây nhị phân tự cân bằng như sau và ta muốn xoay trái node *x*:



- Nếu node *y* có một nút con bên trái (gọi là *b*), ta gán b là node con bên phải của *x*.
- Nếu node cha *p* của *x* là NULL, cho *y* là node gốc của cây. Nếu không sẽ có hai trường hợp: nếu node *x* là node con bên trái của node cha, cho *y* là node con bên trái của *p*; ngược lại cho y là node con bên phải của *p*
- Cuối cùng, gán node *y* là node cha của node *x*.

Sau khi thực hiên các thao tác trên, ta sẽ được cây mới như sau:




```
void leftRotate(NodePtr x) {
  NodePtr y = x->right;
  x->right = y->left; // Gan node con ben trai b cua y la node con ben phai cua x
  if (y->left != TNULL) {
    y->left->parent = x; // Gan x la cha cua node con ben trai cua y
  }
  y->parent = x->parent; // Gan dia chi node cha cua y la node cha cua x
  if (x->parent == nullptr) { // Neu node cha cua x la NULL
    this->root = y; // Gan y la node goc cua cay
  } else if (x == x->parent->left) { // Neu x la node con ben trai cua node cha
    x->parent->left = y; // Gan y la node con ben trai cua p
  } else {
    x->parent->right = y; // Neu khong, gan y la node con ben phai cua p
  }
  y->left = x; // Gan node x la node con ben trai cua y
  x->parent = y; // Gan node y la node cha cua node x
}
```

+ *Xoay phải:* Thực hiện tương tự như thao tác xoay trái

Quay lại với việc chèn một node mới, để chèn một node mới (gọi là newNode), ta thực hiện 2 giai đoạn sau:

+) Giai đoạn 1: Chèn node mới

1. Gán địa chỉ node con bên phải và bên trái của newNode là NIL.
2. Gán màu đỏ cho newNode.
3. Gọi y là một node NIL và *x* là node gốc của cây.
4. Kiểm tra xem cây có rỗng không (nghĩa là x là node NIL). Nếu có, cho newNode là node gốc của cây và gán màu đen cho nó.
5. Nếu cây không rỗng, lặp lại các thao tác sau cho đến khi chạm tới node NIL:

    - So sánh newNode với node x.

- Nếu giá trị của newNode lớn hơn x, duyệt sang cây con bên phải. Nếu không, duyệt sang cây con bên trái

- Cho địa chỉ của node y trùng giá trị của node newNode.

Sau khi thực hiện vòng lặp trên, node y sẽ giữ địa chỉ của node cha của newNode, gán địa chỉ của node y là địa chỉ cha của newNode

6. So sánh newNode với node y. Nếu lớn hơn thì gán newNode là con bên phải của node y; nếu không, gán newNode là con bên trái.
7. Gọi hàm InsertFix để duy trì thuộc tính của cây đỏ đen nếu bị vi phạm.

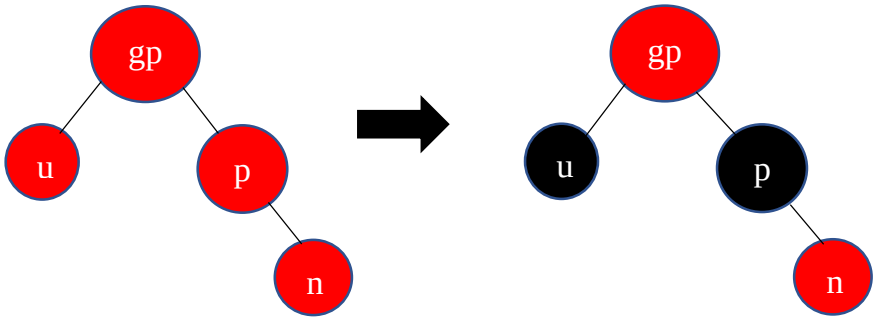
+) Giai đoạn 2: Duy trì thuộc tính của cây đỏ đen sau khi chèn

1. Lặp lại các thao tác sau đối với newNode nếu node cha của newNode (gọi là p) có màu đỏ:

   1.1. Nếu p là node con bên phải của node cha của nó (gọi là gp), gọi u là node con bên phải của gp, ta có các trường hợp sau:

   - Trường hợp 1: node u có màu đỏ

     Đổi màu node u và node p thành màu đen, node gp thành màu đỏ và gán địa chỉ của gp cho newNode.

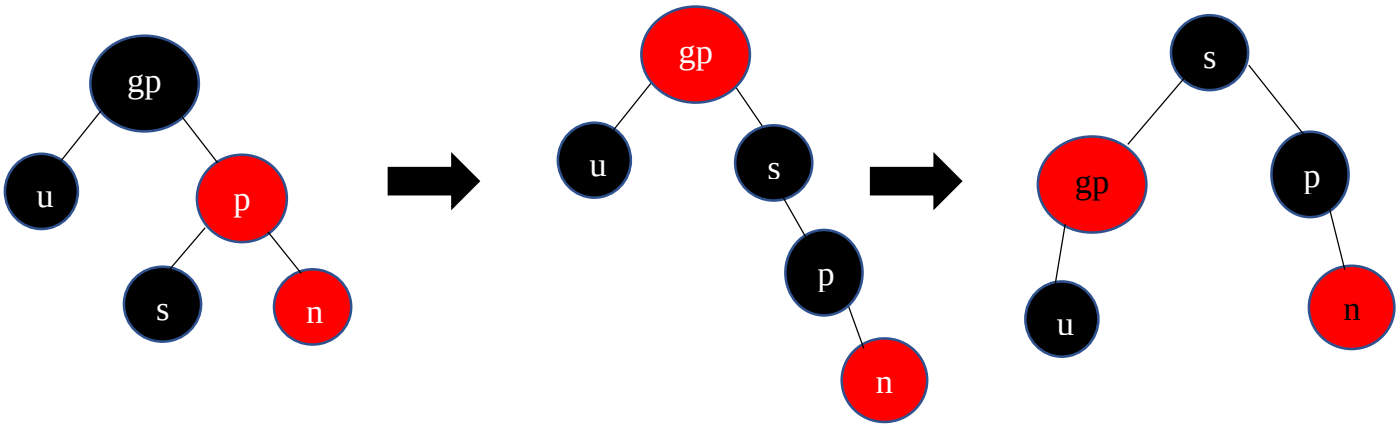


   - Trường hợp 2: node u có màu đen

     - Nếu newNode là node con bên phải của p, gán địa chỉ của node cha p của newNode vào

newNode và thực hiện xoay phải cho node newNode lúc này

- Đổi màu node cha của newNode thành màu đen và node ông của newNode thành màu đỏ
- Thực hiện xoay trái với node ông của node newNode




```
void insertFix(NodePtr k) {
    NodePtr u;
    while (k->parent->color == 1) {
      if (k->parent == k->parent->parent->right) { // Truong hop 1
        u = k->parent->parent->left;
        if (u->color == 1) { // Neu node u co mau do
          u->color = 0; // Doi mau node u
          k->parent->color = 0; // Doi mau node p
          k->parent->parent->color = 1; // Doi mau node gp
          k = k->parent->parent; // Gan dia chi cua gp cho newNode
        } else { // Neu node u co mau den
          if (k == k->parent->left) { // Neu newNode la node con ben phai cua p
            k = k->parent; // Gan dia chi cua p vao newNode
            rightRotate(k); // Thuc hien xoay phai newNode
          }
          k->parent->color = 0; // Doi mau newNode thanh mau den
          k->parent->parent->color = 1; // Doi mau node gp cua newNode thanh mau do
          leftRotate(k->parent->parent); // Thuc hien xoay trai node gp cua newNode
        }
      } else {
        u = k->parent->parent->right;
```

1.2. Nếu p là node con bên trái của gp và gọi u là node con bên phải của gp, tương tự như 1.1, ta có các trường hợp:

- Trường hợp 1: node u có màu đỏ

Đổi màu node u và node p thành màu đen, node gp thành màu đỏ và gán địa chỉ của gp cho newNode.

- Trường hợp 2: node u có màu đen
    - Nếu newNode là node con bên phải của p, thực hiện thao tác xoay trái đối với node p
    - Nếu không, thực hiện xoay phải node gp
    - Đổi màu node p thành màu đen và node gp thành màu đỏ

2. Nếu newNode lúc này trùng node gốc, hủy bỏ vòng lặp.
3. Cuối cùng, đặt màu của gốc là màu đen để đảm bảo tính chất của cây đỏ đen

ii) Xóa đi 1 node:

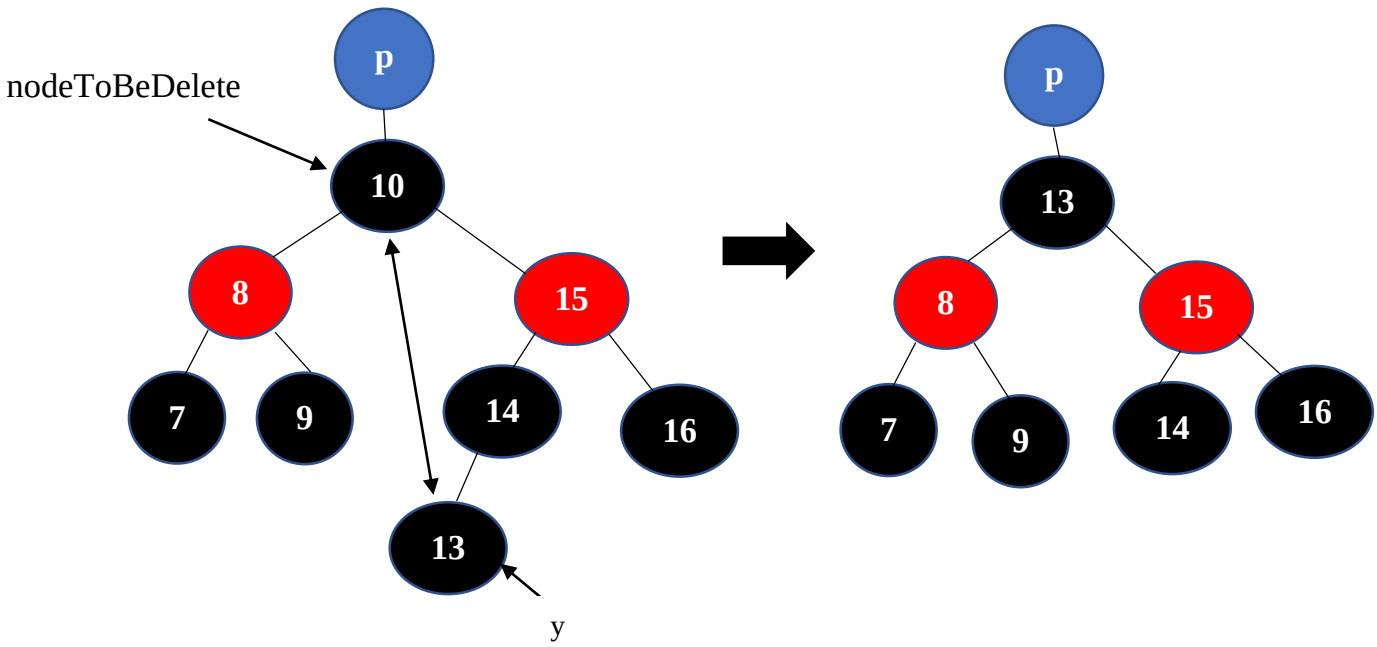
Để xóa một node mới (gọi là nodeToBeDeleted), ta thực hiện 2 giai đoạn sau:

+) Giai đoạn 1: Xóa node được yêu cầu

- Lưu màu của nodeToBeDeleted vào biến originalColor
- Nếu node con bên trái của node nodeToBeDeleted là NIL,
    - Gọi node con bên phải của node nodeToBeDeleted là node x
    - Thay thế node nodeToBeDeleted bằng node x
- Nếu node con bên phải của node nodeToBeDeleted là NIL,
    - Gọi node con bên trái của node nodeToBeDeleted là node x
    - Thay thế node nodeToBeDeleted bằng node x
- Nếu cả node con bên trái và bên phải node nodeToBeDeleted không phải là node NIL, ta thực hiện các thao tác sau:

- Gọi node có giá trị nhỏ nhất của cây con bên phải của node nodeToBeDeleted là y
- Lưu màu của y vào biến originalColor
- Gán x là node con bên phải của y
- Nếu y là node con của nodeToBeDeleted, gán y là node cha của node x
- Nếu không, thay thế y bằng node con bên phải của y
- Thay thế node nodeToBeDeleted bằng node y

- Nếu biến originalColor mang màu đen, gọi hàm DeleteFix(x) để đảm bảo tính chất của cây đỏ đen



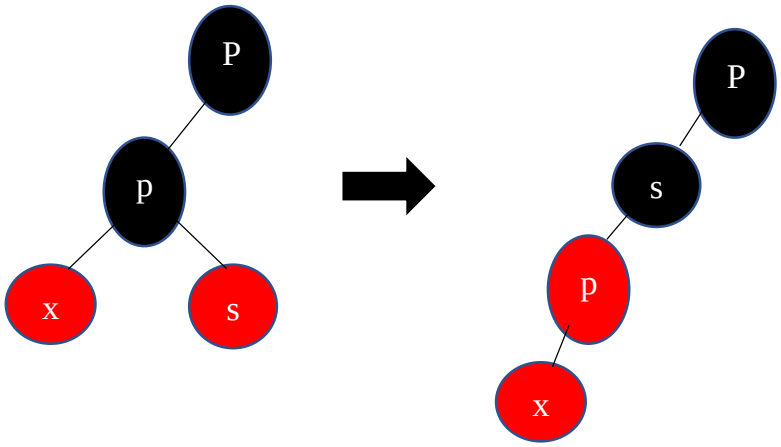
+) Giai đoạn 2: Duy trì thuộc tính của cây đỏ đen

Sau giai đoạn 1, vì node y luôn luôn là một node là có màu đen nên sau khi di chuyển y, một vài đường đi từ gốc đến node x (thực chất là node

NIL) sẽ bị giảm đi một đơn vị đối với chiều cao đen. Theo đó thuộc tính thứ 5 sẽ không được bảo đảm

Thực hiên các thao tác sau cho đến khi x không phải là gốc của cây và màu của x là ĐEN:

- Nếu x là node con trái của node cha của nó :
    - Gọi w là node con phải của node cha của node x.
    - Nếu node con phải của cha của node x có màu đỏ:
        - Đặt màu của node con bên phải của cha của node x là màu đen.
        - Đặt màu của cha của node x là màu đỏ.
        - Xoay trái node cha của node x.
        - Gán node con bên phải của node cha của node x cho node w.



- Nếu màu của cả node con bên phải và node con bên trái của w là màu đen:
    - Đặt màu của node w là màu đỏ.
    - Gán địa chỉ node cha của x vào node x.
- Nếu màu của node con bên phải của node w là màu đen:

- Đặt màu của node con bên trái của node w là màu đen.
- Đặt màu của w là ĐỎ.
- Thực hiên xoay phải node w.
- Gán node con bên phải của node cha của node x cho node w.

- Nếu không có trường hợp nào ở trên xảy ra, thì làm như sau:
  - Đặt màu của node w bằng màu của node cha của node x.
  - Đặt màu của node cha của x là màu đen.
  - Đặt màu của con phải của w là ĐEN.
  - Thực hiên xoay trái node cha của node x.
  - Gán địa chỉ của gốc của cây vào node x để kết thúc vòng lặp.

```
void deleteFix(NodePtr x) {
  NodePtr s;
  while (x != root && x->color == 0) {
    if (x == x->parent->left) {
      s = x->parent->right;
      if (s->color == 1) {
        s->color = 0;
        x->parent->color = 1;
        leftRotate(x->parent);
        s = x->parent->right;
      }

      if (s->left->color == 0 && s->right->color == 0) {
        s->color = 1;
        x = x->parent;
      } else {
        if (s->right->color == 0) {
          s->left->color = 0;
          s->color = 1;
          rightRotate(s);
          s = x->parent->right;
        }

        s->color = x->parent->color;
        x->parent->color = 0;
        s->right->color = 0;
        leftRotate(x->parent);
        x = root;
      }
```

- Nếu x là node con phải của node cha của nó, thực hiện tương tự như trên với phải thay đổi thành trái và ngược lại.
- Cuối cùng, đặt màu của x là ĐEN.

d. Các thư viện hỗ trợ (C++/Python)

+) Đối với C++, Trong C++, cây đỏ-đen được hỗ trợ thông qua thư viện chuẩn STL (Standard Template Library). Các cấu trúc dữ liệu như std::map, std::set, std::multimap, và std::multiset đều sử dụng cây đỏ-đen để quản lý dữ liệu một cách hiệu quả.

Các cấu trúc dữ liệu sử dụng cây đỏ đen:

1. **std::map**: Là một container lưu trữ các cặp khóa-giá trị, trong đó các khóa được sắp xếp theo thứ tự tăng dần. std::map không cho phép các khóa trùng lặp.
2. **std::set**: Là một container lưu trữ các giá trị không trùng lặp và được sắp xếp theo thứ tự tăng dần.
3. **std::multimap**: Giống std::map, nhưng cho phép các khóa trùng lặp.
4. **std::multiset**: Giống std::set, nhưng cho phép các giá trị trùng lặp

+) Thư viện **pyredblack** trong Python cung cấp một cách dễ dàng để sử dụng cấu trúc dữ liệu cây đỏ-đen (red-black tree). Một số câu lệnh cơ bản:

- **Tạo cây đỏ-đen**: pyredblack.rbdict(Germany='Berlin', ...) tạo một cây đỏ-đen và khởi tạo với các cặp khóa-giá trị.
- **Đếm số phần tử trong cây**: len(d) trả về số phần tử hiện có trong cây.
- **Lấy giá trị của một khóa**: d['Ireland'] truy xuất giá trị tương ứng với khóa 'Ireland'.
- **Lấy danh sách các khóa**: d.keys() trả về danh sách tất cả các khóa trong cây theo thứ tự tăng dần.
- **Lấy danh sách các giá trị**: d.values() trả về danh sách tất cả các giá trị trong cây theo thứ tự tăng dần của khóa.
- **Xóa phần tử cuối cùng**: d.popitem() xóa và trả về phần tử có khóa lớn nhất trong cây.

e. Vận dụng thực tế

Cây đỏ-đen (red-black tree) là một cấu trúc dữ liệu cây nhị phân tìm kiếm tự cân bằng, có nhiều ứng dụng thực tế quan trọng do khả năng cân bằng tự động và hiệu suất tốt trong các thao tác thêm, xóa và tìm kiếm. Dưới đây là một số ứng dụng thực tế của cây đỏ-đen:

1. **Cơ sở dữ liệu**: Trong hệ thống quản lý cơ sở dữ liệu (DBMS), cây đỏ-đen thường được sử dụng để triển khai các cấu trúc dữ liệu như cây B, cây B* và cây B+ để cải thiện hiệu suất tìm kiếm và truy xuất dữ liệu.

2. **Bộ điều khiển ổ đĩa**: Trong hệ thống tệp tin của hệ điều hành, cây đỏ-đen có thể được sử dụng để duy trì các vị trí của các khối dữ liệu trên ổ đĩa, giúp tối ưu hóa thời gian truy xuất dữ liệu.
3. **Ngôn ngữ lập trình**: Một số ngôn ngữ lập trình như C++ và Python cung cấp thư viện hoặc modules cho cây đỏ-đen. Các cấu trúc dữ liệu này có thể được sử dụng trong các ứng dụng phức tạp như tìm kiếm từ điển, quản lý tập hợp dữ liệu, và phát triển các thuật toán.
4. **Routing Tables trong mạng máy tính**: Trong các router và switch mạng, cây đỏ-đen có thể được sử dụng để lưu trữ và tìm kiếm các bản ghi trong bảng định tuyến, giúp cải thiện hiệu suất của hệ thống định tuyến.
5. **Trò chơi điện tử và đồ họa máy tính**: Trong các ứng dụng trò chơi điện tử và đồ họa máy tính, cây đỏ-đen có thể được sử dụng để quản lý các đối tượng hoặc tài nguyên trong trò chơi, như việc lưu trữ và tìm kiếm vị trí của các đối tượng trên màn hình.
6. **Quản lý tập tin và thư mục**: Trong các hệ thống tập tin và thư mục, cây đỏ-đen có thể được sử dụng để duy trì cấu trúc cây của thư mục và các tệp tin, giúp tối ưu hóa thời gian truy xuất và tìm kiếm tệp tin.
7. **Kiểm tra tính phân phối**: Trong các thuật toán phân phối dữ liệu như dịch vụ CDN (Content Delivery Network), cây đỏ-đen có thể được sử dụng để lưu trữ và truy xuất các vị trí của các nút mạng, giúp cải thiện hiệu suất và độ tin cậy của dịch vụ.

Những ứng dụng này chỉ là một phần nhỏ trong số nhiều ứng dụng của cây đỏ-đen trong thế giới thực. Sự linh hoạt và hiệu suất của cấu trúc dữ liệu này khiến nó trở thành một công cụ quan trọng trong nhiều lĩnh vực công nghiệp và khoa học máy tính.

4. Tài liệu tham khảo

- https://www.geeksforgeeks.org/introduction-to-red-black-tree/

- https://www.programiz.com/dsa/red-black-tree

- https://luanvan.net.vn/luan-van/bao-cao-cay-do-den-28255/

- Chat GPT

5. Phụ lục 1: Chương trình minh họa

https://github.com/laikhanhhoang/Red-Black-Tree

Cho dữ liệu nạp vào là một file text gồm 200 số (giá trị không vượt quá 5000), khi chạy chương trình sẽ xuất ra cây đỏ đen.

Ví dụ: Khi nhập vào các số từ 0 đến 10, chương trình sẽ xuất ra:

```
R----3(BLACK)
   L----1(BLACK)
   |  L----0(BLACK)
   |  R----2(BLACK)
   R----5(BLACK)
      L----4(BLACK)
      R----7(RED)
         L----6(BLACK)
         R----8(BLACK)
            R----9(RED)
```

Tương đương với cây đỏ đen sau: