

Cây đồ đen

TÀI LIỆU

MATHEMATICS AND STATISTICS

Thích 0

Chia sẻ 0

Tweet



0

Cây đồ đen (tiếng Anh: red-black tree) là một dạng cây tìm kiếm nhị phân tự cân bằng, một cấu trúc dữ liệu được sử dụng trong khoa học máy tính. Cấu trúc ban đầu của nó được đưa ra vào năm 1972 bởi Rudolf Bayer. Ông gọi chúng là các "B-cây cân bằng" còn tên hiện nay được đưa ra từ 1978 bởi Leo J. Guibas và Robert Sedgwick. Nó là cấu trúc phức tạp nhưng cho kết quả tốt về thời gian trong trường hợp xấu nhất. Các phép toán trên chúng như tìm kiếm (search), chèn (insert), và xóa (delete) trong thời gian $O(\log n)$, trong đó n là số các phần tử của cây.

Thuật ngữ

Một cây đồ-đen là một cây nhị phân, là một cấu trúc dữ liệu trong khoa học máy tính để tổ chức các thành phần dữ liệu có thể so sánh, chẳng hạn các số. Trong cây nhị phân các thành phần dữ liệu được đưa vào các nút (node). Trong các nút có một nút nằm ở vị trí khởi đầu không là con của nút nào được gọi là gốc. Các nút khác đều là con của một nút nào đó và có không quá hai con. Từ nút gốc có một đường đi duy nhất đến mỗi nút khác trên cây.

Các nút không có con được gọi là lá (leaf node). Trong cây đồ đen, các lá được gán giá là null; nghĩa là chúng không chứa bất kỳ dữ liệu nào.

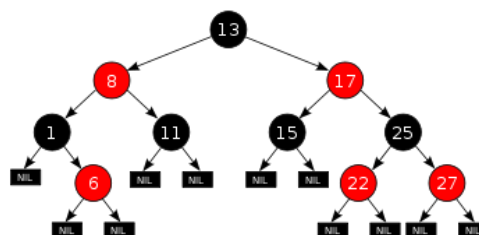
Các cây tìm kiếm nhị phân, bao gồm cả cây đồ-đen, thỏa mãn tính chất: mỗi nút được gán một giá trị sao cho giá trị trên mỗi nút nhỏ hơn hoặc bằng tất cả các giá trị trên các nút thuộc cây con phải và lớn hơn các giá trị nằm trên cây con trái. Điều đó làm cho quá trình tìm kiếm nhanh hơn.

Thuận lợi khi sử dụng

Các cây đồ đen cùng với các cây AVL, thường đảm bảo một thời gian tốt nhất trong trường hợp xấu nhất cho các phép toán chèn (insertion), xóa (deletion), và tìm kiếm (search).

Các cây đồ-đen là một đồng cấu của các cây 2-3-4. Ngược lại, cho một cây 2-3-4, có ít nhất một cây đồ-đen với các thành phần dữ liệu theo đúng thứ tự ấy. Các phép chèn và xóa trên cây 2-3-4 cũng tương đương với đổi màu và quay trong cây đồ đen. Điều này làm cho các cây 2-3-4 có một công cụ quan trọng tương đương logic với cây đồ-đen. Do đó các giải thuật trên các cây 2-3-4 tuy có trước cây đồ-đen nhưng thường ít được dùng hơn cây đồ đen.

Các tính chất



Ví dụ một cây đồ-đen

Mỗi nút của cây đồ-đen có thuộc tính "màu" nhận một trong hai giá trị "đỏ" hoặc "đen". Ngoài ra:

Một nút hoặc là đỏ hoặc đen.

Gốc là đen.

Tất cả các lá là đen.

Cả hai con của mọi nút đỏ là đen. (và suy ra mọi nút đỏ có nút cha là đen.)

Tất cả các đường đi từ một nút đã cho tới các lá chứa một số như nhau các nút đen.

Tính chất 5 còn được gọi là tính chất "cân bằng đen". Số các nút đen trên một đường đi từ gốc tới mỗi lá được gọi là độ dài đen của đường đi đó. Trong bài này chỉ xét các đường đi từ gốc tới các lá nên ta sẽ gọi tất cả các đường đi như vậy là đường đi. Sức mạnh của cây đồ đen nằm trong các tính chất trên. Từ các tính chất này suy ra trong các đường đi từ gốc tới các lá đường đi dài nhất không vượt quá hai lần đường đi ngắn nhất. Do đó cây đồ đen là gần cân bằng. Vì các thuật toán chèn, xóa, tìm kiếm trong trường hợp xấu nhất đều tỷ lệ với chiều cao của cây nên cây đồ đen rất hiệu quả trong các trường hợp xấu nhất, không giống như cây tìm kiếm nhị phân thông thường.

Để thấy rõ sức mạnh này, ta chú ý rằng không có đường đi nào từ gốc tới một lá chứa hai nút đỏ liên nhau (theo tính chất 4). Do đó trên mỗi đường số nút đỏ không nhiều hơn số nút đen. Đường đi ngắn nhất là đường đi chỉ có nút đen, đường đi dài nhất có thể là đường đi xen kẽ giữa các nút đỏ và đen. Theo tính chất 5, số các nút đen trên hai đường đi đó bằng nhau, và do đó đường đi dài nhất không vượt quá hai lần đường đi ngắn nhất.

Trong nhiều biểu diễn của dữ liệu cây, có thể có các nút chỉ có một con và có các lá có chứa dữ liệu. Tuy nhiên có thể biểu diễn cây đồ đen ta có một chút thay đổi mà không làm thay đổi tính chất cơ bản của cây và độ phức tạp của các thuật toán. Với mục đích này, ta đưa thêm các lá null vào làm con phải hoặc con trái hoặc cả hai của những nút không có chúng, các lá này không chứa dữ liệu mà chỉ làm nhiệm vụ thông báo rằng tại đây cây đã kết thúc, như hình vẽ ở trên. Việc thêm các nút này làm cho tất cả các nút trong của cây đều chứa dữ liệu và có hai con, hay khác đi cây đồ đen cùng với các lá null là cây nhị phân đầy đủ. Khi đó số các "lá null" nhiều hơn số các nút chứa dữ liệu của cây một lá.

Một số người định nghĩa cây đồ đen bằng cách gán màu đồ đen cho các cạnh chứ không phải các nút. Tuy nhiên điều đó không tạo nên sự khác biệt. Khi ấy màu của mỗi nút tương ứng với màu của cạnh nối nó với nút cha.

Các phép toán trên cây đồ đen

Có thể áp dụng ngay các phép chèn, xóa trong cây tìm kiếm nhị phân vào cây đồ đen mà không cần sửa chữa gì vì cây đồ đen là trường hợp riêng của cây tìm kiếm nhị phân. Tuy nhiên, khi đó có thể có một số tính chất trong định nghĩa của cây đồ đen sẽ bị vi phạm. Việc khôi phục các tính chất đồ đen sẽ cần một số nhỏ cỡ $O(\log n)$ hoặc trung bình chỉ $O(1)$ các phép đổi màu (tốn rất ít thời gian) và không quá ba phép quay cho phép xóa, hai cho phép chèn. Toàn bộ các giải thuật chèn và xóa có độ phức tạp thời gian cỡ $O(\log n)$.

Phép chèn

Phép chèn bắt đầu bằng việc bổ sung một nút như trong cây tìm kiếm nhị phân bình thường và gán cho nó màu đỏ. Ta xem xét để bảo toàn tính chất đồ đen từ các nút lân cận với nút mới bổ sung. Thuật ngữ nút chú bác sẽ dùng để chỉ nút anh (hoặc em) với nút cha của nút đó như trong cây phả hệ. Chú ý rằng:

Tính chất 3 (Tất cả các lá -là các nút null là đen) giữ nguyên.

Tính chất 4 (Cả hai con của nút đỏ là đen) nếu bị thay đổi chỉ bởi việc thêm một nút đỏ có thể sửa bằng cách gán màu đen cho một nút đỏ hoặc một phép quay.

Tính chất 5 (Tất cả các đường đi từ gốc tới các lá có cùng một số nút đen) nếu bị thay đổi chỉ bởi việc thêm một nút đỏ có thể sửa bằng cách gán màu đen cho một nút đỏ hoặc một phép quay.

Chú ý: Nhân N sẽ dùng để chỉ nút đang chèn vào, P chỉ nút cha của N', G chỉ ông của N', và U chỉ chú bác của N'. Nhớ rằng, giữa các trường hợp, vai trò và nhãn của các nút có thể thay đổi còn trong cùng một trường hợp thì không.

Mỗi trường hợp được giới thiệu bằng một đoạn mã C. Nút chú bác và nút ông dễ dàng xác định nhờ các hàm sau:

```
struct node *grandparent(struct node *n) {
    return n->parent->parent;
}
struct node *uncle(struct node *n) {
    if (n->parent == grandparent(n)->left)
        return grandparent(n)->right;
    else
        return grandparent(n)->left;
}
```

Trường hợp 1: Nút mới thêm N ở tại gốc. Trong trường hợp này, gán lại màu đen cho N, để bảo toàn tính chất 2 (Gốc là đen). Vì mới chỉ bổ sung một nút, Tính chất 5 được bảo đảm vì mọi đường đi chỉ có một nút.

```
void insert_case1(struct node *n) {
    if (n->parent == NULL)
        n->color = BLACK;
    else
        insert_case2(n);
}
```

Trường hợp 2: Nút cha P của nút mới thêm là đen, khi đó Tính chất 4 (Cả hai nút con của nút đỏ là đen) không bị vi phạm vì nút mới thêm có hai con là "null" là đen. Tính chất 5 cũng không vi phạm vì nút mới thêm là đỏ không ảnh hưởng tới số nút đen trên tất cả đường đi.

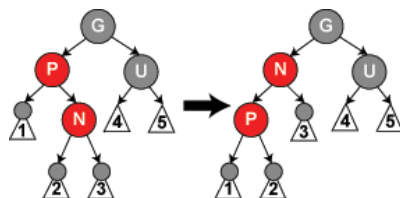
```
void insert_case2(struct node *n) {
    if (n->parent->color == BLACK)
        return; /* Tree is still valid */
    else
        insert_case3(n);
}
```

Chú ý: Trong trường hợp tiếp theo nếu N có ông là nút G, vì nếu cha P là đỏ và P không ở gốc thì G là đen. Như vậy, N cũng có chú bác là U, although it may be a leaf in cases 4 and 5.

Trường hợp 3: Cả cha P và bác U là đỏ, thì thể đổi cả hai thành đen còn G thành đỏ (để bảo toàn tính chất 5 .Khi đó nút mới N có cha đen. Vì đường đi bất kỳ đi qua cha và bác của "N" phải đi qua ông của N nên số các nút đen trên đường đi này không thay đổi. Tuy thế nút ông G có thể vi phạm tính chất 2 (Gốc là đen) hoặc 4 (Cả hai con của nút đỏ là nút đen) (tính chất 4 bị vi phạm khi cha của G là đỏ). Để sửa chữa trường hợp này gọi một thủ tục đệ quy trên G từ trường hợp 1. Note that this is the only recursive call, and it occurs prior to any rotations, which proves that a constant number of rotations occur.

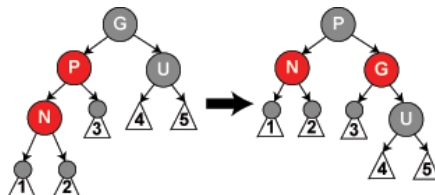
```
void insert_case3(struct node *n) {
    if (uncle(n) != NULL && uncle(n)->color == RED) {
        n->parent->color = BLACK;
        uncle(n)->color = BLACK;
        grandparent(n)->color = RED;
        insert_case1(grandparent(n));
    }
    else
        insert_case4(n);
}
```

Chú ý: Trong các trường hợp tiếp theo, giả sử rằng nút cha P là con trái của cha của nó. Nếu nó là con phải, left và right đổi chỗ cho nhau trong cases 4 and 5.



Trường hợp 4: Nút cha P là đỏ nhưng nút chú bác U là đen, nút mới N là con phải của nút P, và P là con trái của nút G. Trong trường hợp này, thực hiện quay trái chuyển đổi vai trò của nút mới N và nút cha P do đó định dạng lại nút P bằng Trường hợp 5 (đổi vai trò N và P) vì tính chất 4 bị vi phạm (Cả hai con của nút đỏ là đen). Phép quay cũng làm thay đổi một vài đường đi (các đường đi qua cây con nhãn "1") phải đi qua thêm nút mới N, nhưng vì N là đỏ nên không làm chúng vi phạm tính chất 5

```
void insert_case4(struct node *n) {
    if (n == n->parent->right && n->parent == grandparent(n)->left) {
        rotate_left(n->parent);
        n = n->left;
    } else if (n == n->parent->left && n->parent == grandparent(n)->right) {
        rotate_right(n->parent);
        n = n->right;
    }
    insert_case5(n);
}
```



Trường hợp 5: Nút cha P là đỏ nhưng nút bác U là đen, nút mới N là con trái của nút P, và P là con trái của nút ông G. Trong trường hợp này, một phép quay phải trên nút ông G được thực hiện; kết quả của phép quay là trong cây mới nút P trở thành cha của cả hai nút N và nút G. Đã biết G là đen, vì bây giờ nó là con của P. Đổi màu của P và G thì cây thỏa mãn tính chất 4. Tính chất 5 không bị vi phạm vì các đường đi qua G trước đây bây giờ đi qua P.

```
void insert_case5(struct node *n) {
    n->parent->color = BLACK;
    grandparent(n)->color = RED;
    if (n == n->parent->left && n->parent == grandparent(n)->left) {
        rotate_right(grandparent(n));
    } else {
        /* Here, n == n->parent->right && n->parent == grandparent(n)->right */
        rotate_left(grandparent(n));
    }
}
```

Phép xóa

Trong cây tìm kiếm nhị phân bình thường khi xóa một nút có cả hai con (không là lá null), ta tìm phần tử lớn nhất trong cây con trái hoặc phần tử nhỏ nhất trong cây con phải, chuyển giá trị của nó vào nút đang muốn xóa (xem Cây tìm kiếm nhị phân). Khi đó chúng ta xóa đi nút đã được copy giá trị, nút này có ít hơn hai con (không là lá null). Vì việc copy giá trị không làm mất tính chất đồ đen nên không cần phải sửa chữa gì cho thao tác này. Việc này chỉ đặt ra khi xóa các nút có nhiều nhất một con (không là lá null).

Chúng ta sẽ thảo luận về việc xóa một nút có nhiều nhất một con (không là lá null).

Nếu ta xóa một nút đỏ, ta có thể chắc chắn rằng con của nó là nút đen. Tất cả các đường đi đi qua nút bị xóa chỉ đơn giản bớt đi một nút đỏ do đó tính chất 5 không thay đổi. Ngoài ra, cả nút cha và nút con của nút bị xóa đều là nút đen, do đó tính chất 3 và 4 vẫn giữ nguyên. Một trường hợp đơn giản khác là khi xóa một nút đen chỉ có một con là nút đỏ. Khi xóa nút đó các tính chất 4 và 5 bị phá vỡ, nhưng nếu gán lại màu cho nút con là đen thì chúng lại được khôi phục.

Trường hợp phức tạp xảy ra khi cả nút bị xóa và nút con của nó đều là đen. Chúng ta sẽ bắt đầu bằng việc thay nút bị xóa bằng nút con của nó. Chúng ta sẽ gọi nút con này (trong vị trí mới của nó là N, và anh em với nó (con khác của nút cha mới) là S. Tiếp theo ta vẫn dùng P chỉ cha mới của N, SL chỉ con trái của S, và SR chỉ con phải của S (chúng tồn tại vì S không thể là lá).

Chú ý: Giữa các trường hợp khác nhau, vai trò và nhãn của các nút có thể thay đổi, nhưng trong một trường hợp mọi nhãn giữ vai trò không thay đổi. Trong hình vẽ các màu đồ đen được thể hiện khi màu của nút đã rõ ràng, màu trắng biểu thị một màu chưa rõ (hoặc đỏ hoặc đen).

Chúng ta sẽ sử dụng hàm sau tìm người anh em của N':

```

struct node *sibling(struct node *n) {
    if (n == n->parent->left)
        return n->parent->right;
    else
        return n->parent->left;
}

```

Chú ý: Tất nhiên, chúng ta cần hoàn chỉnh các lá null sau mọi phép thay đổi. Nếu nút bị xóa không có con N khác "lá null", dễ dàng thấy rằng các tính chất được thỏa mãn. Còn nếu N là một "lá null", có thể sửa chữa lược đồ (hoặc code) để trong tất cả các trường hợp các tính chất được thỏa mãn.

Trước mỗi bước ta có thể dùng hàm (function) `replace_node` thay thế nút con child vào vị trí của nút bị xóa trên cây. Để thuận tiện các đoạn code trong mục này quy ước rằng các "lá null" được biểu diễn bằng các đối tượng nút thực sự khác biệt một chút với NULL (code trong phép chèn có biểu diễn không như vậy).

```

void delete_one_child(struct node *n) {
    /* Giả thiết : n có ít nhất một nút con null */
    struct node *child = is_leaf(n->right) ? n->left : n->right;
    replace_node(n, child);
    if (n->color == BLACK) {
        if (child->color == RED)
            child->color = BLACK;
        else
            delete_case1(child);
    }
    free(n);
}

```

Ghi chú: Nếu N là "lá null" và ta không muốn biểu diễn các "lá null" bằng các đối tượng nút thực, ta có thể sửa giải thuật bằng cách trước hết gọi `delete_case1()` trên cha của nó (nghĩa là nút bị xóa n trong đoạn code trên) rồi sau đó mới xóa nó. Ta có thể làm như vậy vì cha của nó là đen, do đó có diễn biến như với "lá null" (một số người gọi là "lá ảo", "lá ma"). Ta cũng có thể xóa nó khỏi cuối của n và sẽ khôi phục lại sau tất cả các phép toán.

Nếu cả N và gốc ban đầu của nó là đen thì sau khi xóa các đường qua "N" giảm bớt một nút đen. Do đó vi phạm Tính chất 5, cây cần phải cân bằng lại. Có các trường hợp sau:

Trường hợp 1

Trường hợp 1: N là gốc mới. Trong trường hợp này chúng ta dừng lại. Ta đã giải phóng một nút đen khỏi mọi đường đi và gốc mới lại là đen. Không tính chất nào bị vi phạm.

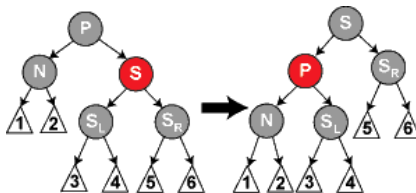
```

void delete_case1(struct node *n) {
    if (n->parent == NULL)
        return;
    else
        delete_case2(n);
}

```

Chú ý: Trong các trường hợp 2, 5, và 6, ta quy ước N là con trái của cha P. Nếu nó là con phải, left và right sẽ trao đổi cho nhau. Tuy nhiên code ví dụ làm cho cả hai trường hợp.

Trường hợp 2

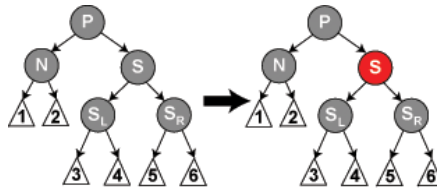


Trường hợp 2: S là đỏ. Trong trường hợp này trao đổi màu của P và S, và sau đó quay trái tại P, nó sẽ làm cho S trở thành nút ông của N. Chú ý rằng P có màu đen và có một con màu đỏ. Tất cả các đường đi có số các nút đen giống nhau, bây giờ N có một anh em màu đen và cha màu đỏ, chúng ta có thể tiếp tục với các trường hợp 4, 5, hoặc 6. (anh em mới của nó là đen vì chỉ có một con của nút đỏ S.) Trong các trường hợp sau ta sẽ gọi anh em mới của N' là S.

```

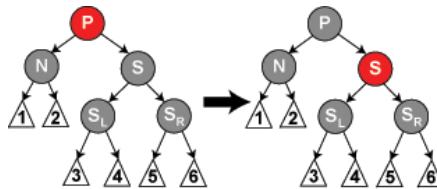
void delete_case2(struct node *n) {
    if (sibling(n)->color == RED) {
        n->parent->color = RED;
        sibling(n)->color = BLACK;
        if (n == n->parent->left)
            rotate_left(n->parent);
        else
            rotate_right(n->parent);
    }
    delete_case3(n);
}

```

Trường hợp 3

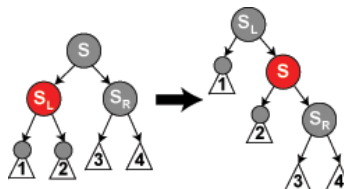
Trường hợp 3: P, S, và các con của S là đen. Trong trường hợp này, chúng ta gán lại cho S màu đỏ. Kết quả là mọi đường đi qua S, (tất nhiên chúng không qua N, có ít hơn một nút đen. Vì việc xóa đi cha trước đây của N làm tắt cả các đường đi qua N bớt đi một nút đen, nên chúng bằng nhau. Tuy nhiên tất cả các đường đi qua P bây giờ có ít hơn một nút đen so với các đường không qua P, do đó Tính chất 5 (Tất cả các đường đi từ gốc tới các nút lá có cùng số nút đen) sẽ bị vi phạm. Để sửa chữa nó chúng ta lại tái cân bằng tại P, bắt đầu từ trường hợp 1.

```
void delete_case3(struct node *n) {
    if (n->parent->color == BLACK &&
        sibling(n)->color == BLACK &&
        sibling(n)->left->color == BLACK &&
        sibling(n)->right->color == BLACK)
    {
        sibling(n)->color = RED;
        delete_case1(n->parent);
    }
    else
        delete_case4(n);
}
```

Trường hợp 4

Trường hợp 4: S và các con của S là đen nhưng P là đỏ. Trong trường hợp này, chúng ta đổi ngược màu của S và P. Điều này không ảnh hưởng tới số nút đen trên các đường đi không qua N, nhưng thêm một nút đen trên các đường đi qua N, thay cho nút đen đã bị xóa trên các đường này.

```
void delete_case4(struct node *n) {
    if (n->parent->color == RED &&
        sibling(n)->color == BLACK &&
        sibling(n)->left->color == BLACK &&
        sibling(n)->right->color == BLACK)
    {
        sibling(n)->color = RED;
        n->parent->color = BLACK;
    }
    else
        delete_case5(n);
}
```

Trường hợp 5

Trường hợp 5: S là đen, con trái của S là đỏ, con phải của S là đen, còn N là con trái của cha nó. Trong trường hợp này chúng ta quay phải tại S, khi đó con trái của S trở thành cha của S và N là anh em mới của nó. Sau đó ta trao đổi màu của S và cha mới của nó. Tất cả các đường đi sẽ có số nút đen như nhau, nhưng bây giờ N có một người anh em đen mà con phải của nó lại là đỏ, chúng ta chuyển sang Trường hợp 6. Hoặc N hoặc cha của nó bị tác động bởi việc dịch chuyển này.

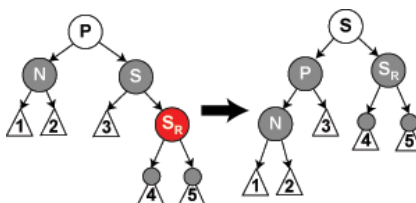
(Lưu ý trong trường hợp 6, ta đặt lại nút anh em mới của N là S.)

```

void delete_case5(struct node *n) {
    if (n == n->parent->left &&
        sibling(n)->color == BLACK &&
        sibling(n)->left->color == RED &&
        sibling(n)->right->color == BLACK)
    {
        sibling(n)->color = RED;
        sibling(n)->left->color = BLACK;
        rotate_right(sibling(n));
    }
    else if (n == n->parent->right &&
        sibling(n)->color == BLACK &&
        sibling(n)->right->color == RED &&
        sibling(n)->left->color == BLACK)
    {
        sibling(n)->color = RED;
        sibling(n)->right->color = BLACK;
        rotate_left(sibling(n));
    }
    delete_case6(n);
}

```

Trường hợp 6



Trường hợp 6: S là đen, con phải của S là đỏ và N là con trái của nút cha P. Trong trường hợp này chúng ta quay trái tại P, khi đó S trở thành cha của P và con phải của S. Chúng ta hoán đổi màu của P và S, và gán cho con phải của S màu đen. Cây con giữ nguyên màu của gốc do đó Tính chất 4 (Cả hai con của nút đỏ là đen) và Tính chất 5 không bị vi phạm trong cây con này. Tuy nhiên, N bây giờ có thêm một nút đen tiền nhiệm: hoặc P mới bị tô đen, nó đã là đen và S là nút ông của nó trở thành đen. Như vậy các đường đi qua N có thêm một nút đen.

Trong lúc đó, với một đường đi không đi qua N, có hai khả năng:

đi qua nút anh em của N. Khi đó cả trước và sau khi quay nó phải đi qua S và P, khi thay đổi màu sắc hai nút này đã trao đổi màu cho nhau. Như vậy đường đi này không bị thay đổi số nút đen.

đi qua nút bác của N', là con phải của S. Khi đó trước khi quay nó chỉ đi qua S, cha của S, và con phải của S, nhưng sau khi quay nó chỉ đi qua nút S và con phải của S, khi này S đã nhận màu cũ của cha P còn con phải của S's đã đổi màu từ đỏ thành đen. Kết quả là số các nút đen trên đường đi này không thay đổi.

Như vậy, số các nút đen trên các đường đi là không thay đổi. Do đó các tính chất 4 và 5 đã được khôi phục. Nút trắng trong hình vẽ có thể là đỏ hoặc đen, nhưng phải ghi lại trước và sau khi thay đổi.

```

void delete_case6(struct node *n) {
    sibling(n)->color = n->parent->color;
    n->parent->color = BLACK;
    if (n == n->parent->left) {
        /* Here, sibling(n)->right->color == RED */
        sibling(n)->right->color = BLACK;
        rotate_left(n->parent);
    }
    else
    {
        /* Here, sibling(n)->left->color == RED */
        sibling(n)->left->color = BLACK;
        rotate_right(n->parent);
    }
}


```

Nhắc lại rằng các hàm này có lời gọi đệ quy. Thêm nữa lời gọi không đệ quy sẽ được gọi sau một phép quay, do đó số lần thực hiện các phép quay là không đổi (không quá 3).

0 bình luận

Sắp xếp theo

Cũ nhất




Thêm bình luận...

Plugin bình luận của Facebook

TÀI VỆ

TÁI SỬ DỤNG(/user/reuse/m/6f96a98f/1)



Wikipedia (/profile/8)

0 GIÁO TRÌNH (/PROFILE/8?TYPES=2) | 5771 TÀI LIỆU (/PROFILE/8?TYPES=1)

(/profile/8)

ĐÁNH GIÁ:

0 dựa trên 0 đánh giá

NỘI DUNG CÙNG TÁC GIẢ

- Quảng ninh (/m/quang-ninh/990954be)
- Tim người (/m/tim-nguoi/c9dd5369)
- WTO (/m/wto/22570df1)
- Các kiểu bộ nhớ (/m/cac-kieu-bo-nho/f4953d18)
- phong trào thơ mới (/m/phong-trao-tho-moi/6a4f7b3d)
- Đá (/m/da/c01eb5f4)
- Alexandre Yersin (/m/alexandre-yersin/775eda83)
- Họ Dầu (/m/ho-dau/201a4385)
- Ngôi sao điện ảnh (/m/ngoi-sao-dien-anh/a4c2c341)
- Cá da phiến (/m/ca-da-phien/4c075293)

TRƯỚC

TIẾP

NỘI DUNG TƯƠNG TỰ

- Gà lôi lam đuôi trắng (/m/ga-loi-lam-duoi-trang/c323bb93)
- Chì Chà vá (/m/chi-cha-va/7e7c94b9)
- Lỗ đen không quá đen (/m/lo-den-khong-qua-den/44f3f749)
- Chà vá chân xám (/m/cha-va-chan-xam/8596966a)
- Nghiên cứu về lỗ đen (/m/nghien-cuu-ve-lo-den/2f305482)
- Tê giác đen (/m/te-giac-den/ce1edc3b)
- Đồ uống (/m/do-uong/89f2d40f)
- Lỗ đen (/m/lo-den/7381b999)
- Chim lặn mỏ đen (/m/chim-lan-mo-den/37ebfe6f)
- Báo sư tử (/m/bao-su-tu/fe59b34b)

TRƯỚC

TIẾP



Thư viện Học liệu Mở Việt Nam (VOER) được tài trợ bởi Vietnam Foundation (<http://www.vnfoundation.org>) và vận hành trên nền tảng Hanoi Spring (<http://www.hanoispring.com>). Các tài liệu đều tuân thủ giấy phép Creative Commons Attribution 3.0 trừ khi ghi chú rõ ngoại lệ.



Connect with Facebook

 (<https://www.facebook.com/voer.edu.vn>)