Cấu Trúc Dữ Liệu Và Giải Thuật

Bài tập lớn 2

JJK RESTAURANT OPERATIONS (Phần 2 - Hồi tưởng)

nhóm thảo luận Code
https://www.facebook.com/groups/211867931379013

Tp. Hồ Chí Minh, Tháng 11/2023

Mục lục

1 Nhà hàng Điều Hành cả hai

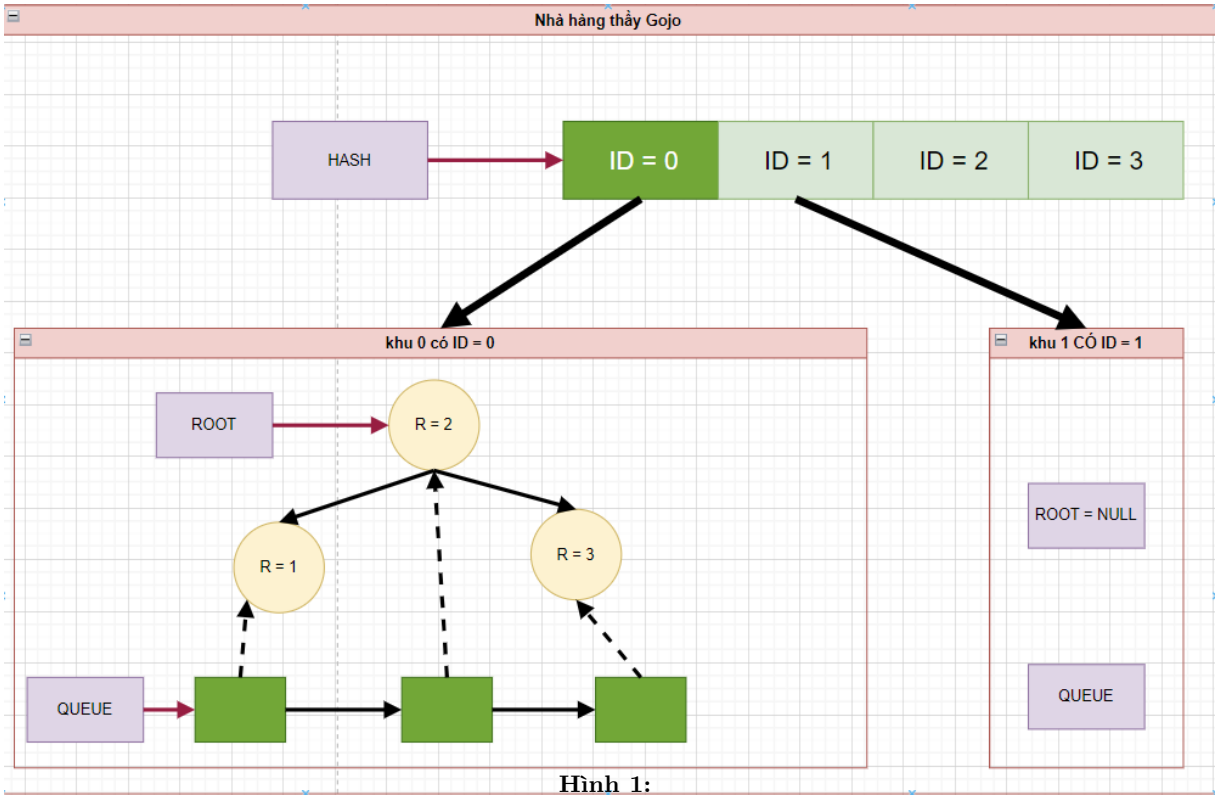
1.1 class JJK RESTAURANT OPERATIONS

1.2 class HuffTree AVL

2 Nhà hàng Gojo

2.1 Class RESTAURANT Gojo

- **Cấu trúc:** Nhà hàng của thầy chia 2 có dạng là nhiều khu được đưa vào *hashTable* để quản lý với trong một khu lại lại quản lí bằng *TreeBST* ý tưởng như hình bên dưới



Hình 1:
Cấu trúc lưu trữ nhà hàng thầy Gojo

1. với bảng *hash* sẽ lưu trữ dưới dạng *vector(array)* (cũng là mảng mà thôi) với kích thước được cấp phát cố định là *MAXSIZE* với màu xanh dương đậm biểu thị trong khu đó có người đang ăn, còn xanh nhạt biểu thị khu đó đang trống
2. Về Khu không có khách thì *ROOT* thì sẽ là *NULL* và *QUEUE* đang rỗng
3. Đối với Khu có khách thì *ROOT* sẽ lưu trữ cây nhị phân tìm kiếm *BST* còn *QUEUE* sẽ lưu trữ thời gian mà khách hàng đó dùng chưa lưu trữ *FIFO* nào đến trước thì ưu tiên đuổi trước.
4. **Thư viện vector** được dùng như một *array* được cấp phát động.
5. **Thư viện queue** được dùng với cơ chế *FIFO*.

- **Code:**

```
class RESTAURANT_Gojo{
        class Tree_BST;
private:
        vector<Tree_BST> areaTable;
public:
        RESTAURANT_Gojo():areaTable(MAXSIZE){}
        void insertAreaTable(int result);
        void remove_KOKUSEN();
        void print_LIMITLESS(int number);
private:
        class Tree_BST{
                class Node;
        private:
                Node* root;
                queue<int> queueTime;
        public:
            Tree_BST():root(nullptr){}
            int size(){return queueTime.size();}
            void insert(int result);
            void remove_recursive(Node* nodeDelete);
            void remove();
            void print_recursive(Node* node);
            void print()
        private:
                class Node{
                private:
                        int result;
                        Node* left;
                        Node* right;
                        friend class Tree_BST;
                public:
                        Node(int result) : result(result), left(NULL),
                         → right(NULL) {}
                };
        };
};
```

1. **class Tree_BST :** lưu trữ một cây tìm kiếm nhị phân với các thành phần được mô tả dưới
   - **class Node :** lưu trữ một node của cây với *key* sẽ tương đương *result* này chắc dễ rồi
   - textcolorred**root:** là node gốc của cây đó
   - **QueueTime:** được lưu trữ cơ chế *FIFO* nào vô trước thì ra trước, trong trường hợp này nào ăn lâu nhất là đuổi về sớm nhất. với mỗi phần tử trong *queue* là *result* của từng giá trị đó
   - **Các hàm:** được mô tả kĩ hơn phần dưới, với *size* là lấy kích thước tương đương với kích thước *queue*, sẽ không xung đột với thứ tự khi xóa nha vì min nhỏ nhất bên phải luôn là số thứ 2 giống nó.
2. **Các hàm:** nói kĩ hơn ở phần sau.

2.2 LAPSE

Hàm này sẽ thêm khách hàng vào nhà hàng của Gojo.

2.2.1 insertAreaTable trong class RESTAURANT_Gojo

- **Mô tả:** Với đầu vào là *result* xử lí ở bước phần trước, với một bảng *hash* đã cho trước thì chũng ta chỉ cần tìm ra *ID* thông qua công thức thầy cho **ID = result % MAXSIZE + 1;** sau khi tìm được thì ta chỉ cần gọi hàm *insert* của class *Tree_BST*.
- **Code:**

```
void insertAreaTable(int result)
{
    //! thêm khách hàng vào khu có ID = result % MAXSIZE + 1
    int ID = result % MAXSIZE + 1;
    areaTable[ID].insert(result);
}
```

2.2.2 insert trong class Tree_BST

- **Mô tả:** Hàm này thêm một node và cây *BST* chắc ổn thôi này làm không được thì kì sau gặp anh tiếp nha  .
- **Chú ý:** Nếu giá trị thêm vào sau bằng giá trị đã có thì thêm vào bên phải của cây
- **Code:**

```
void insert(int result){
    //TODO: tự code đi các bạn: khi new nhớ push vào queueTime
    //TODO: giá trị Result của từng khách sẽ là khóa dùng để xây dựng cây BST
    //TODO Nếu giá trị thêm vào sau bằng giá trị đã có thì thêm vào bên phải
     ↪ của cây
}
```

2.3 KOKUSEN

Hàm này sẽ đuổi khách hàng cho là gián điệp vào nhà hàng Gojo.

2.3.1 remove_KOKUSEN trong class RESTAURANT_Gojo

- **Mô tả:** xóa các khách hàng trong tất cả các bàn trong nhà ăn
- **Code:**

```
void remove_KOKUSEN()
{
    for(int i = 0; i < MAXSIZE; i++) areaTable[i].remove();
}
```

2.3.2 remove trong class Tree_BST

- **Mô tả:** Hàm này xóa một số node và cây *BST*.
- **Chú ý:** Node có giá trị nhỏ nhất của cây con bên phải sẽ được dùng để thay thế với node cần xóa khi thực hiện tác vụ xóa một node trong cây BST
- **Hướng giải quyết theo các bước sau:** yêu cầu làm **Bước 1**

Bước 1: Đếm số lượng node cần xóa nhân viên sẽ chuyển đổi cây BST (gọi là cây BST gốc) sang một mảng các giá trị theo hậu thứ tự post-order. Sau đó, nhân viên sẽ tính toán xem có bao nhiêu hoán vị của mảng vừa tạo mà nếu thêm lần lượt các phần tử đó theo thứ tự chỉ mục từ nhỏ đến lớn, có thể sinh ra cùng một cây BST như cây BST gốc

Bước 2: khách hàng tại khu vực đó theo thứ tự FIFO. với số lượng là kết quả trên

Bước 3: *remove_recursive* này xóa node trong cây tìm kiếm nhị phân thôi.

- **Code:**

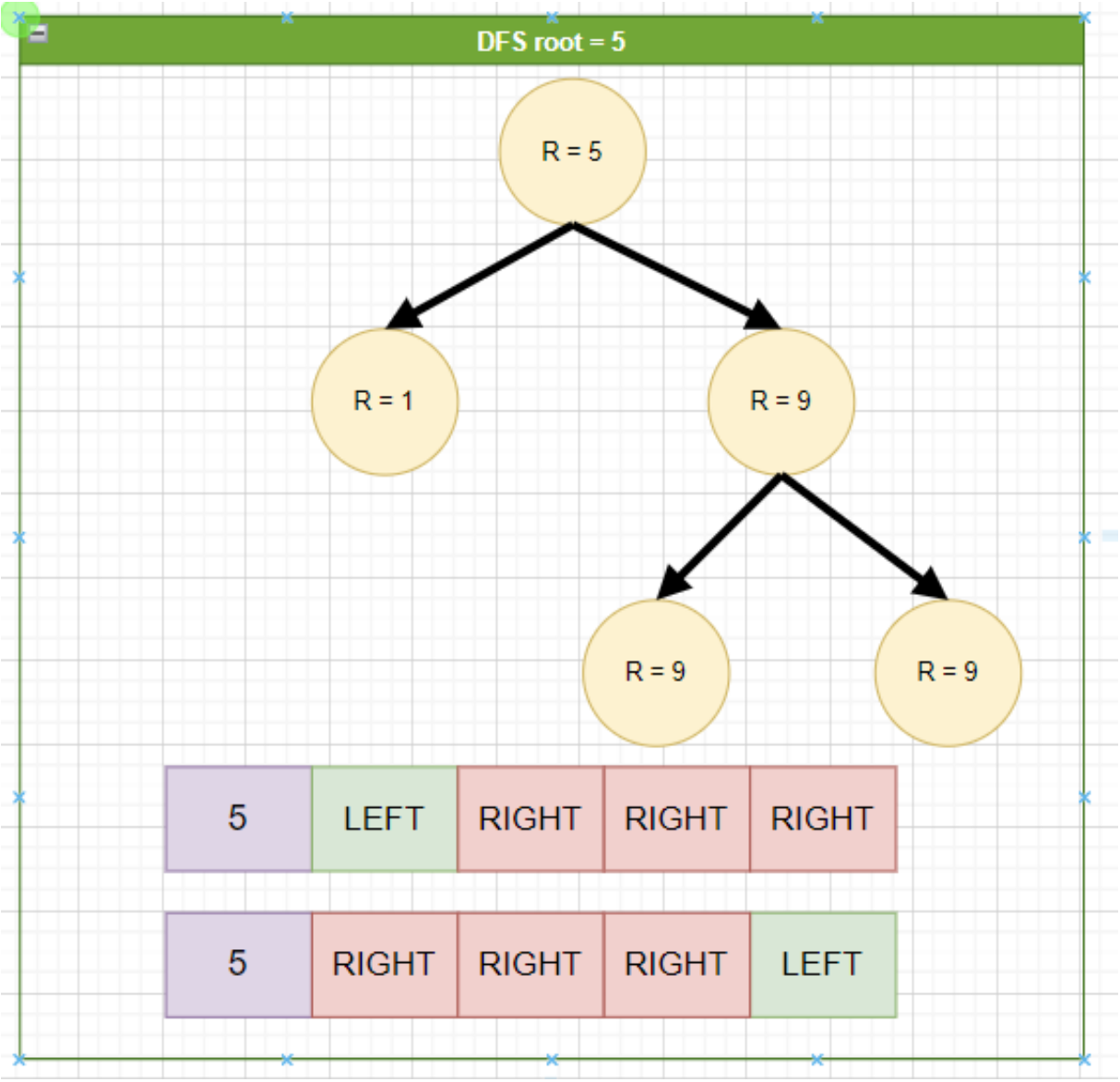
```
void remove_recursive(Node* nodeDelete)
{
    if(nodeDelete == nullptr) return;
    //TODO: xóa thôi
}
void remove(){
    if(this->size() == 0) return; //! trường hợp rỗng bỏ qua
    //* bước 1: đếm số lượng node cần xóa
    unsigned long long number = DFS(root);
    if(this->size() == 1) return;

    //* bước 2: xóa node trong cây với số lượng đã tính trên
    while(number == 0 && !queueTime.empty())
    {
        Node* temp = queueTime.front();
        queueTime.pop();
        remove_recursive(temp);
        delete temp;
    }
}
```

2.4 ý tưởng của hàm DFS

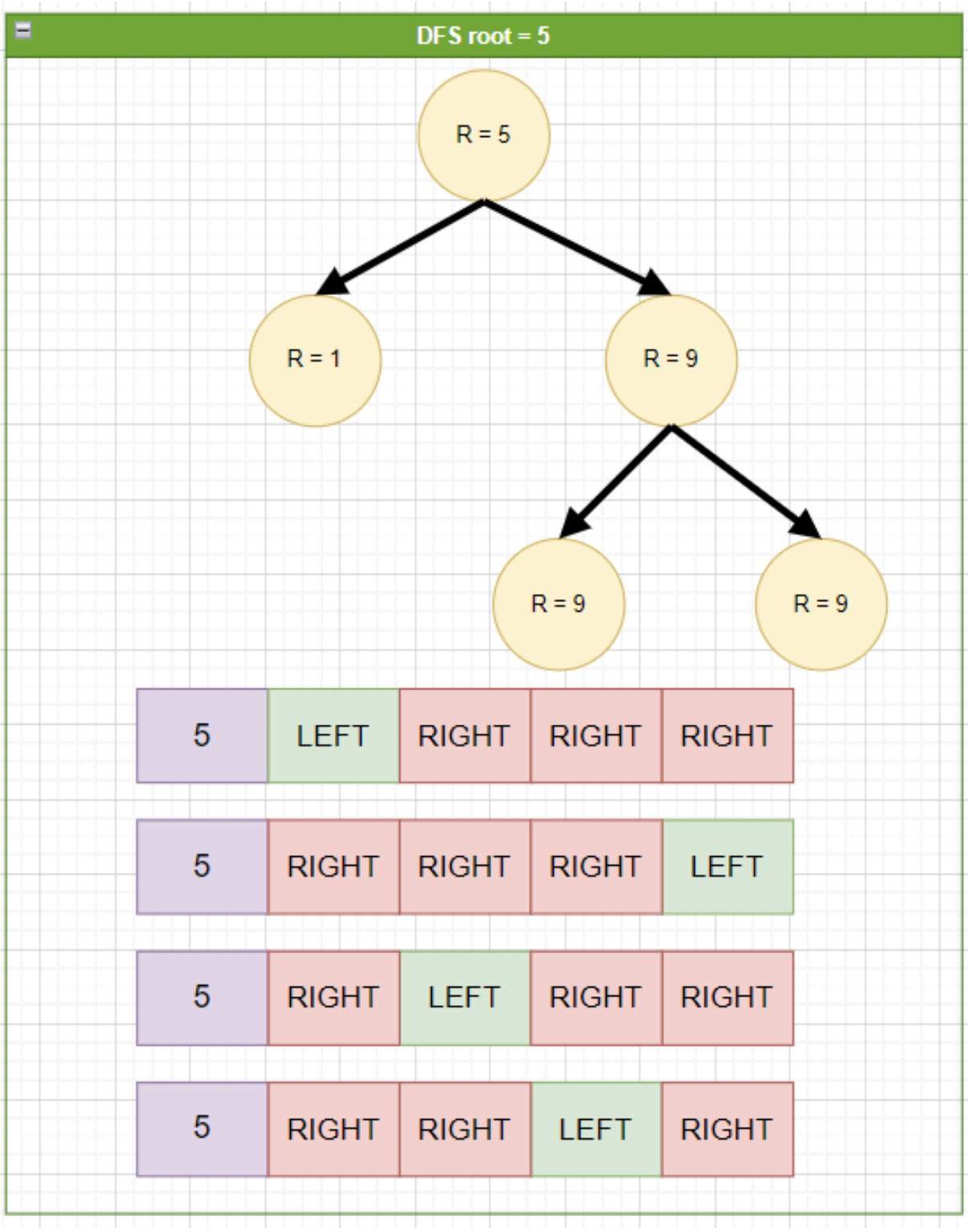
Hàm này có chắc năng tính số hoán vị có thể tạo ra một cây *BST* tương tự cây ban đầu, ý tưởng ban dưới

Bước 1: ta thấy *root* sẽ là cố định không thể duy chuyển luôn insert ở đầu cây, nhưng *left* và *right* chúng ta có thể thêm 1 trong 2 trước đều được không ảnh hưởng đến cấu trúc của cây, hình dưới có thể thấy *left right* sẽ có thể bên trái hay bên phải đều được



Bước 2: Ta suy ra công thức đầu tiên $result = DFS(left) * DFS(right)$ nếu cây trong bên trái và phải chỉ có 1 node

Bước 3: nhưng mà code không bao giờ mơ như thế đối với số lượng node *left* và *right* có thể lớn hơn một, thì ta xếp sao cho vị trí của các node trong *left* không đổi thứ tự của nó ví dụ 1,3,5 nếu chuyển thành 3,5,1 thì sai vì thứ tự là $1 < 3 < 5$, *right* cũng tương tự, VD left(1), left(2), right(3), right(4) các trường hợp hoán đổi được là 1,2,3,4 or 1,3,2,4 or 1,2,4,3 or 2,1,3,4 or 2,1,4,3 or 2,4,1,3 -> dễ dàng thấy là chỉnh hợp $C(4, 2)$



Bước 4: Ta suy ra công thức đầu tiên $result = C(size(left) + size(right), size(left)) * DFS(left) * DFS(right)$

2.5 LIMITLESS

- **Mô tả:** hàm này gọi theo thứ tự *print_LIMITLESS* trong class *RESTAURANT_Gojo* gọi tới *print* trong class *Tree_BST* sau đó gọi *print_recursive* trong class *Tree_BST*, này khá dễ nha in theo thứ thự *inorder* thôi đáp án luôn là tăng dần nha do này là cây *BST* không hiểu là xác định !!  .
- **Code:** *print* **trong** *Tree_BST*

```
//* hàm này theo trung thứ tự (in-order) thôi không gì khó hết
void print_recursive(Node* node)
{
    if(node == nullptr) return; //! trường hợp dừng print
    print_recursive(node->left);
    solution << node->result << "\n";
    print_recursive(node->right);
}
void print(){
    if(this->size() == 0) return; //! trường hợp rỗng bỏ qua
    print_recursive(root);
}
```

2.6 Test case

Test	Mô tả	Lỗi
1	insert một node và print rỗng và print 1 node	in 1 phần tử trả về rỗng và insert cập nhật *queue*
2	insert nhiều phần tử bên phải right	lỗi gọi đệ quy root->right
3	insert nhiều phần tử bên phải right có bằng nhau	lỗi gọi đệ quy root->right chưa xét bằng nhau
4	insert nhiều phần tử bên trái left	lỗi gọi đệ quy root->left
5	insert left và right	lỗi gọi đệ quy trái và phải
6	insert left và right có bằng nhau	lỗi gọi đệ quy trái và phải
7	tất cả đều bằng nhau	lỗi đệ quy phải bằng nhau
8	có 2 khu	lỗi phân chia ID
9	có 3 khu	lỗi phân chia ID
10	có 3 khu	print ra number không nằm trong khoảng
11	Xóa rỗng	chưa xét điều kiện xóa rỗng
12	chỉ có 1 node	1 node thì không xóa vì không có hoán vị
13	chỉ có 2 node tính được number = 1	xem thử tính number, xóa đúng không
14	chỉ có 3 node tính được number = 1	xem thử tính number, xóa đúng không
15	chỉ có 3 node tính được number = 2	xem thử tính number, xóa đúng không
16	nhiều node tính được number = 8	xem thử tính number, xóa cây rỗng
17	nhiều node tính được number = 8	xem thử tính number, xóa cây rỗng
18	có 2 khu và number = 2 cả 2 TH	xem phân chia ID
19-100	random MAXSIZE bé hơn 10	random
19-100	random MAXSIZE lớn hơn 10	random

Bảng 1: *test phần nhà hàng Gojo*

3 Nhà hàng Sukuna

3.1 Class RESTAURANT Sukuna

3.2 LAPSE

3.3 KEITEIKEN

3.4 CLEAVE

nhóm thảo luận Code

https://www.facebook.com/groups/211867931379013

CHÚC CÁC EM HỌC TỐT