ĐẠI HỌC QUỐC GIA TPHCM

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

---

# Báo cáo đồ án

## Đề tài: TREAP - Cây tìm kiếm nhị phân ngẫu nhiên

---

**Môn học: Cấu trúc giải thuật và lập trình**

Sinh viên thực hiện:
Đặng Quốc Cường (23520192)

Giáo viên hướng dẫn:
ThS. Nguyễn Thanh Sơn

Ngày 10 tháng 5 năm 2024

# Mục lục

# Lời cảm ơn

Trong thời gian làm đồ án, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô và bạn bè.

Em xin gửi lời cảm ơn chân thành đến những thầy, cô bộ môn Cấu trúc dữ liệu và giải thuật là những người đã tận tình hướng dẫn, chỉ bảo em trong suốt quá trình làm đồ án.

Em cũng xin gửi lời cảm ơn đến các bạn cùng lớp đã đóng góp ý kiến và đưa ra những lời khuyên để em thực hiện đồ án lần này.

Với điều kiện thời gian cũng như kinh nghiệm còn hạn chế của sinh viên năm nhất, bài báo cáo này không thể tránh được những thiếu sót. Em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy, cô để có điều kiện bổ sung, nâng cao kỹ năng, ý thức của mình, để phục vụ tốt hơn công tác thực tế sau này.

Em xin chân thành cảm ơn,

**Đặng Quốc Cường**

# 1 Giới thiệu

## 1.1 Thông tin đồ án

| | |
|---|---|
| Tên chủ đề | Cây tìm kiếm nhị phân ngẫu nhiên - TREAP |
| Người hướng dẫn | Th.S Nguyễn Thanh Sơn |
| Người thực hiện | Đặng Quốc Cường |

Bảng 1: Thông tin đồ án

.
Sau khi đã học qua cây tìm kiếm nhị phân BST, ta cũng biết được cây BST còn một vài hạn chế. Nên từ đó, ta có cây tìm kiếm nhị phân ngẫu nhiên - TREAP là phiên bản nâng cấp dựa trên cây BST và có thể tối ưu những hạn chế của cây BST trước đó gặp phải.

## 1.2 Thông tin sinh viên thực hiện

| Tên sinh viên | MSSV | Gmail | SĐT |
|---|---|---|---|
| Đặng Quốc Cường | 23520192 | 23520192@gm.uit.edu.vn | 0347607267 |

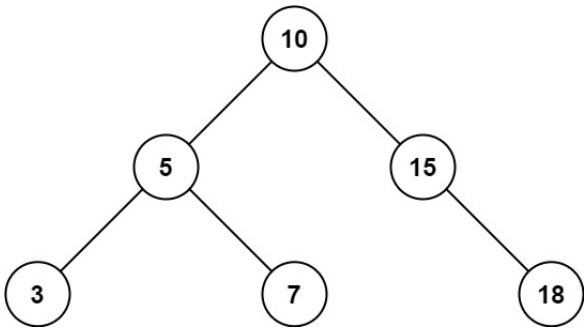
Bảng 2: Thông tin sinh viên thực hiện

.

# 2 Nhắc lại về cây tìm kiếm nhị phân - BST

## 2.1 Sơ lược về cây tìm kiếm nhị phân

- Cây nhị phân gồm nhiều nút, mỗi nút có nhiều nhất là 2 nút con, gọi là **nút con trái** và **nút con phải**. Các nút không có nút con nào gọi là **nút lá**.
- Ngoài ra, mỗi nút đều có duy nhất một nút cha, trừ **nút gốc** của cây không có cha.
- Mỗi nút có thể chứa thêm thông tin, gọi là **khóa**. Khóa của nút thường là một số nguyên, tuy nhiên có thể là bất cứ kiểu dữ liệu nào.




Hình 1: Cây tìm kiếm nhị phân

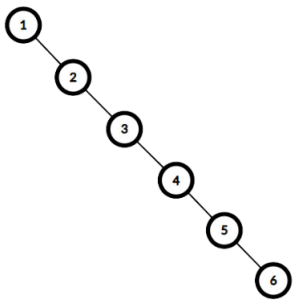
## 2.2 Tính chất của cây tìm kiếm nhị phân

Với cây tìm kiếm nhị phân BST, khóa của các nút sẽ thỏa mãn các tính chất sau :

- Khóa của các nút không trùng lặp, mỗi nút có một khóa khác với tất cả các nút còn lại
- Với mỗi nút, khóa của tất cả các nút trong cây con bên trái **nhỏ hơn** khóa của nút đó.
- Với mỗi nút, khóa của tất cả các nút trong cây con bên phải **lớn hơn** khóa của nút đó.

## 2.3 Độ phức tạp và vấn đề của cây tìm kiếm nhị phân

- Độ phức tạp của các thao tác trên tùy thuộc vào độ cao của cây khi truy vấn. Với dữ liệu ngẫu nhiên, cách cài đặt cây nhị phân trong bài này có độ phức tạp trung bình cho mỗi thao tác là $\mathcal{O}(\log n)$ với $N$ là số nút trên cây.
- Tuy nhiên, độ phức tạp trong trường hợp **xấu nhất** (hình bên dưới) là $\mathcal{O}(n)$, vì vậy cây này không được sử dụng trong thực tế.




Hình 2: Cây tìm kiếm nhị phân dưới dạng đường thẳng

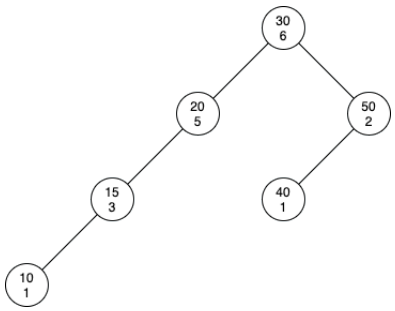
## 2.4 Kết luận

Ta cần tìm kiếm một cấu trúc dữ liệu cây có độ phức tạp tối ưu hơn so với cây tìm kiếm nhị phân BST.

# 3 Cây tìm kiếm nhị phân ngẫu nhiên - TREAP

## 3.1 Sơ lược về cây tìm kiếm nhị phân ngẫu nhiên

- Treap là một cấu trúc kết hợp giữa cây tìm kiếm nhị phân và heap.
- Treap đã được **Raimund Siedel** và **Cecilia Aragon** đề xuất vào năm 1989 [1].
- Treap đơn giản là một cây nhị phân có 2 khóa, trong đó một khóa thỏa mãn tính chất của Heap, còn một khóa thỏa mãn tính chất của cây tìm kiếm nhị phân. Khóa biểu diễn cho Heap sẽ được **sinh ngẫu nhiên** để đảm bảo rằng cây nhị phân của chúng ta **không quá cao**.




Hình 3: Hình ví dụ về Treap

Cài đặt mỗi nút, C++ :

```
mt19937 rd(chrono::steady_clock::now().time_since_epoch().count());
#define rand rd

// Ham sinh random trong doan [l, r]
int Rand() {
    int l = 1, r = 1000000000;
    return l + rd() * 1LL * rd() % (r - l + 1);
}

class Node {
    public:
        int data, priority;
        Node *left, *right;

        Node(int d) {
            data = d;
            priority = Rand();
            left = NULL;
            right = NULL;
        }
};
```

Mỗi nút gồm khóa tìm kiếm là `key`, khóa `heap` là `priority`. Khi tạo mới một nút, ta tự động lấy ngẫu nhiên giá trị của khóa `heap`.

## 3.2 Tính chất của cây tìm kiếm nhị phân ngẫu nhiên

Như đã nói, Treap là sự kết hợp của Heap và cây tìm kiếm nhị phân nên các nút thỏa mãn các tính chất sau :

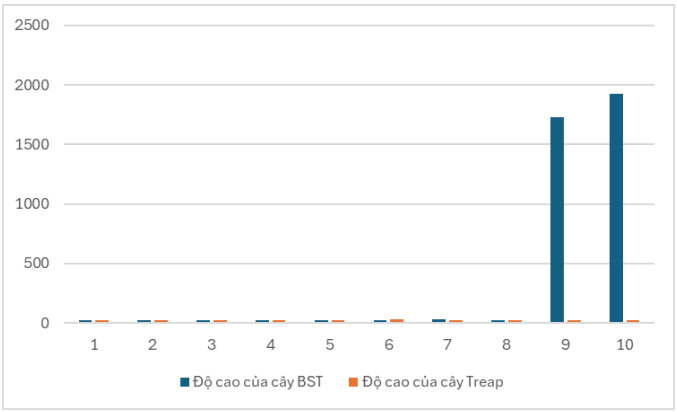
- Với mỗi nút, khóa tìm kiếm của tất cả các nút trong cây con bên trái **nhỏ hơn** khóa tìm kiếm của nút đó.
- Với mỗi nút, khóa tìm kiếm của tất cả các nút trong cây con bên phải **lớn hơn** khóa tìm kiếm của nút đó.
- Với mỗi nút, khóa `heap` của các nút trong cây con sẽ **nhỏ hơ**n (`max-heap`) hoặc **lớn hơn** (`min-heap`) khóa `heap` của nút đó.

Giống như cây Red-Black và AVL, Treap là cây tìm kiếm nhị phân cân bằng, nhưng **không đảm bảo** có độ cao là $\log(n)$, với $N$ là số nút của cây. Ý tưởng là sử dụng thuộc tính **ngẫu nhiên hóa** và Heap để duy trì sự cân bằng với **xác suất cao** [2].

Tham khảo, phân tích giá trị kỳ vọng của độ cao cây Treap tại đây [3].

Ta có kết quả thực nghiệm về độ cao của cây tìm kiếm nhị phân BST và cây tìm kiếm nhị phân ngẫu nhiên Treap qua 10 bộ dữ liệu, mỗi bộ tạo nên cây nhị phân khoảng 2000 nút :

- 8 bộ dữ liệu đầu tiên được sinh ngẫu nhiên.
- 1 bộ dữ liệu được sinh theo giá trị các nút tăng dần.
- 1 bộ dữ liệu được sinh theo giá trị các nút giảm dần.




Hình 4: Biểu đồ độ cao của cây BST và Treap qua kết quả thực nghiệm

**Nhận xét** : Với việc có được những tính chất trên như đã nói, cây tìm kiếm nhị phân ngẫu nhiên Treap có **sự tối ưu về độ cao** hơn so với cây tìm kiếm nhị phân BST.

## 3.3 Một số thao tác cơ bản trên cây tìm kiếm nhị phân ngẫu nhiên

### 3.3.1 Thao tác tìm kiếm

Thao tác tìm kiếm được thực hiện giống như trên cây tìm kiếm nhị phân, không cần quan tâm đến khóa `heap`.
Thao tác tìm kiếm, `C++`:

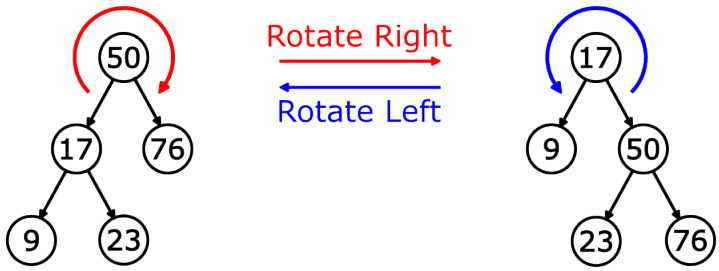
```
Node *find(Node *root, int key) {
    if (root == NULL) return nullptr;
    if (key > root->data) return find(root->right, key);
    // neu key lon hon gia tri dinh hien tai thi di theo nhanh phai
    if (key < root->data) return find(root->left, key);
    // neu key nho hon gia tri dinh hien tai thi di theo nhanh trai
    return root;
}
```

Độ phức tạp : $\mathcal{O}(\log n)$

### 3.3.2 Thao tác xoay cây

Sau khi thêm hoặc xóa nút, ta sẽ thực hiện một vài thao tác để cây không vi phạm tính chất `heap`. Các thao tác này chính là thao tác xoay cây, gồm 2 loại là:

- Xoay trái
- Xoay phải




Hình 5: Mô tả thao tác xoay cây

Thao tác xoay trái, xoay phải, `C++`:

```
Node *rotate_right(Node *node) {
    Node *x = node->left, *t = x->right;
    x->right = node;
    node->left = t;
    return x;
}

Node *rotate_left(Node *node) {
    Node *x = node->right, *t = x->left;
    x->left = node;
    node->right = t;
```

```
    return x;
}
```

Độ phức tạp : $\mathcal{O}(1)$

### 3.3.3 Thao tác thêm phần tử

Khi ta thêm một phần tử $x$ vào cây tìm kiếm nhị phân ngẫu nhiên:

- Tạo thêm một nút có khóa nhị phân bằng $x$ và khóa `heap` được sinh ngẫu nhiên.
- Thực hiện tương tự như việc thêm phần tử ở cây tìm kiếm nhị phân.
- Thực hiện thêm các thao tác xoay cây, để cây không bị vi phạm, vẫn thỏa mãn tính chất của các khóa `heap` [4].

Thao tác thêm vào cây một giá trị $x$, với các khóa `heap` thỏa mãn `min-heap`, `C++`:

```
Node* insert(Node* root, int x) {
    // root la dinh goc cua cay Treap
    // x la gia tri duoc them vao
    if(root == NULL) return new Node(x);

    if(x < root->data) {
        root->left = insert(root->left, x);

        if (root->priority > root->left->priority)
            root = rotate_right(root);

        // neu cay bi vi pham, xoay phai
    } else {
        root->right = insert(root->right, x);

        if (root->priority > root->right->priority)
            root = rotate_left(root);

        // neu cay bi vi pham, xoay trai
    }
    return root;
}
```

Độ phức tạp : $\mathcal{O}(\log n)$

### 3.3.4 Thao tác xóa phần tử

Khi ta xóa một phần tử $x$ trên cây tìm kiếm nhị phân ngẫu nhiên [5]:

- Trường hợp 1 : Nếu nút cần xóa là **nút lá**, thì ta chỉ việc xóa nó đi một cách nhanh chóng.
- Trường hợp 2 : Nếu nút cần xóa có **một nút con**, thì ta thay thế nút đó bằng **nút con không rỗng**.
- Trường hợp 3 : Nếu nút cần xóa có **hai nút con**. Để thỏa mãn `min-heap` thì :
  - Nếu khóa `heap` của nút con trái **nhỏ hơn** khóa `heap` của nút con thì ta thực hiện xoay phải cây.

– Ngược lại thì ta thực hiện xoay trái cây.

Ý tưởng của trường hợp 3 là đưa nút hiện tại về trường hợp 1 hoặc trường hợp 2.

Thao tác xóa trong cây một giá trị $x$, với các khóa `heap` thỏa mãn `min-heap`, `C++`:

```
Node* deleteNode(Node* root, int x) {
    // root la dinh goc cua cay Treap
    // x la gia tri can xoa

    if (root == NULL) return root;

    // Neu x khong phai o root
    if (x < root->data)
        root->left = deleteNode(root->left, x);
    else if (x > root->data)
        root->right = deleteNode(root->right, x);

    // Neu x o root
    // Truong hop 1 : root chi co nut phai
    else if (root->left == NULL) {
        Node *temp = root->right;
        delete(root);
        root = temp; // Xet nut phai lam root
    }

    // Truong hop 2 : root chi co nut trai
    else if (root->right == NULL) {
        Node *temp = root->left;
        delete(root);
        root = temp; // Xet nut trai lam root
    }

    // Truong hop 3 : Neu x o root va root co ca hai nut con
    else if (root->left->priority < root->right->priority) {
        root = rotate_right(root);
        root->right = deleteNode(root->right, x);
    }
    else {
        root = rotate_left(root);
        root->left = deleteNode(root->left, x);
    }

    return root;
}
```

Độ phức tạp : $\mathcal{O}(\log n)$

## 3.4 Ứng dụng

Một số bài tập có thể dùng Treap để giải :

- Median Updates
- C11SEQ - Dãy số

- Codeforces - Radio Stations
- CodeChef - The Prestige

### 3.5 Kết luận

- Treap là một cấu trúc dữ liệu cây nhị phân tìm kiếm ngẫu nhiên dựa trên nền tảng của cây tìm kiếm nhị phân BST và Heap.
- Treap có hiệu suất ở các thao tác tìm kiếm, thêm và xóa tối ưu hơn so với cây tìm kiếm nhị phân BST truyền thống.
- Lựa chọn giữa BST và Treap phụ thuộc vào nhu cầu cụ thể của ứng dụng. Vì cần lưu ý rằng Treap phức tạp hơn BST để triển khai và có thể có chi phí hoạt động cao hơn.

## 4 Tài liệu tham thảo

[1] C. R. Aragon and R. G. Seidel, "Randomized search trees," in 30th Annual Symposium on Foundations of Computer Science, Oct. 1989, pp. 540–545. doi: 10.1109/SFCS.1989.63531.

[2] "Treap (A Randomized Binary Search Tree)," GeeksforGeeks. Accessed: May 09, 2024. https://www.geeksforgeeks.org/treap-a-randomized-binary-search-tree/

[3] J. Ong and S. Shrestha, "1 Analysis of the Height of a Treap", https://www2.hawaii.edu/~nodari/teaching/f19/scribes/notes09.pdf

[4] "Cây tìm kiếm nhị phân - VietCodes." Accessed: May 07, 2024. https://vietcodes.github.io/algo/bst

[5] "Implementation of Search, Insert and Delete in Treap," GeeksforGeeks. Accessed: May 09, 2024. https://www.geeksforgeeks.org/implementation-of-search-insert-and-delete-in-treap/

## 5 Phụ lục

- Link github của đồ án : tại đây.
- Tham khảo hàm random số : tại đây.