Binary-Search-Tree

1115 이정우

목차

1 자료구조 개요

AVL 트리는 이진 탐색 트리의 한 종류로, 노드 삽입이나 삭제 시 자동으로 트리 균형을 유지하는 자가 균형 이진 탐색트리이다. 모든 노드는 왼쪽 서브트리와 오른쪽 서브트리의 높이 차가 절댓값 1을 넘지 않아야 하며 이를 균형 인수라고 한다. 균형이 깨지면 Rotation 연산을 사용해 트리를 균형 상태로 만든다.

2 자료구조 상세

2.1 변수와 ADT

2.1.1 변수

- `root : int`
  - 트리 전체의 시작 노드이다.
- `Node : struct`
  - data, left, right를 저장하는 변수이다.

2.1.2 ADT

- `height(struct Node *n) -> int`
  - 재귀적인 방법을통해 높이를 찾는 함수이다.
- `avl_insert(struct Node *node, int key) -> struct Node*`
  - 새 노드를 삽입하며 bf에 따라, RR, RL, LL, LR 연산을 하는 함수이다.
- `insert() -> void`
  - 노드에 삽입할 데이터 값을 입력받아 avl_insert에 값을 넘기는 함수이다.

2.1.3 Exception

- 특별한 예외 상황은 존재하지 않는다.

2.2 구현 상세 및 핵심 코드

2.2.1 `height(struct Node *n)`

struct Node 형태의 주소를 n으로 받아 재귀적인 방법으로 높이를 알아내는 함수이다.

```
int height(struct Node *n){
    if(!n) return 0;
    int hl = height(n->left);
    int hr = height(n->right);
    return (hl > hr ? hl : hr) + 1;
}
```

2.2.2 `avl_insert(struct Node *node, int key)`

struct Node 형태의 주소를 node로 받고 삽입할 데이터를 key 매개변수로 받는 함수로, 삽입할 위치를 찾아 삽입 연산을 한뒤 스택에서 함수 연산이 빠져나가며 그 위치에서 bf값에 따른 회전 연산을 하게된다. struct Node*를 반환형태로 가지며 이는 루트를 의미한다.

```
struct Node* avl_insert(struct Node *node, int key){

    if(!node){
        struct Node *cur = (struct Node*)malloc(sizeof(struct Node));
        cur->data = key;
        cur->left = cur->right = NULL;
        return cur;
    }
    if(key < node->data) node->left = avl_insert(node->left, key);
    else if(key > node->data) node->right = avl_insert(node->right, key);
    else return node;

    int bf = height(node->left) - height(node->right);

    if(bf > 1){
        if(key < node->left->data){
            struct Node *z = node;
            struct Node *y = z->left;
            struct Node *T2 = y->right;
            y->right = z;
            z->left = T2;
            return y;
```

```
        }else{
            {
                struct Node *z = node->left;
                struct Node *y = z->right;
                struct Node *T2 = y->left;
                y->left = z;
                z->right = T2;
                node->left = y;
            }
            {
                struct Node *z = node;
                struct Node *y = z->left;
                struct Node *T2 = y->right;
                y->right = z;
                z->left = T2;
                return y;
            }
        }
    }
    if(bf < -1){
        if(key > node->right->data){
            struct Node *z = node;
            struct Node *y = z->right;
            struct Node *T2 = y->left;
            y->left = z;
            z->right = T2;
            return y;
        }else{
            {
                struct Node *z = node->right;
                struct Node *y = z->left;
                struct Node *T2 = y->right;
                y->right = z;
                z->left = T2;
                node->right = y;
            }
            {
                struct Node *z = node;
                struct Node *y = z->right;
                struct Node *T2 = y->left;
                y->left = z;
                z->right = T2;
```

```
                return y;
            }
        }
    }
    return node;
}
```

2.2.3 insert()

삽입할 데이터를 입력받아 avl_insert에 struct Node 자료형의 root와 데이터를 넘긴다.

```
void insert(){
    int N;
    while(scanf("%d", &N) == 1){
        root = avl_insert(root, N);
    }
}
```

2.3 복잡도 분석

- AVL 트리는 모든 연산에서 트리의 높이를 log N 수준으로 유지하므로 탐색, 삽입, 삭제 모두 평균과 최악의 시간복잡도가 O(log N)이다. 또한 삽입 중에 회전을 하게되는 연산의 시간복잡도는 O(1)이다.

2.4 전체 코드

```
#include <stdio.h>
#include <stdlib.h>

struct Node{
    int data;
    struct Node *left;
    struct Node *right;
};

struct Node *root = NULL;

int height(struct Node *n){
    if(!n) return 0;
```

```
    int hl = height(n->left);
    int hr = height(n->right);
    return (hl > hr ? hl : hr) + 1;
}

struct Node* avl_insert(struct Node *node, int key){

    if(!node){
        struct Node *cur = (struct Node*)malloc(sizeof(struct Node));
        cur->data = key;
        cur->left = cur->right = NULL;
        return cur;
    }
    if(key < node->data) node->left = avl_insert(node->left, key);
    else if(key > node->data) node->right = avl_insert(node->right, key);
    else return node;

    int bf = height(node->left) - height(node->right);

    if(bf > 1){
        if(key < node->left->data){
            struct Node *z = node;
            struct Node *y = z->left;
            struct Node *T2 = y->right;
            y->right = z;
            z->left = T2;
            return y;
        }else{
            {
                struct Node *z = node->left;
                struct Node *y = z->right;
                struct Node *T2 = y->left;
                y->left = z;
                z->right = T2;
                node->left = y;
            }
            {
                struct Node *z = node;
                struct Node *y = z->left;
                struct Node *T2 = y->right;
                y->right = z;
                z->left = T2;
```

```
                return y;
            }
        }
    }
    if(bf < -1){
        if(key > node->right->data){
            struct Node *z = node;
            struct Node *y = z->right;
            struct Node *T2 = y->left;
            y->left = z;
            z->right = T2;
            return y;
        }else{
            {
                struct Node *z = node->right;
                struct Node *y = z->left;
                struct Node *T2 = y->right;
                y->right = z;
                z->left = T2;
                node->right = y;
            }
            {
                struct Node *z = node;
                struct Node *y = z->right;
                struct Node *T2 = y->left;
                y->left = z;
                z->right = T2;
                return y;
            }
        }
    }
    return node;
}

void insert(){
    int N;
    while(scanf("%d", &N) == 1){
        root = avl_insert(root, N);
    }
}

int main(){
```

```
    insert();
    return 0;
}
```

3 활용 방안

1. DB의 데이터가 편향적일때
    - AVL 트리를 이용하면 효율적으로 데이터를 사용할수있다.

4 인상깊었던 점

이 방법을 찾아낸 사람이 정말 대단하다 생각했다. 그리고 약간 이진탐색트리를 배울때 뭔가 더 효율적일수 있을것같은데라는 생각이 들었는데 여기서 그걸 깨달아서 재밌었다.

5 참고문헌

C AVL 트리(AVL Tree) 구현 - 서리 개인 개발 블로그