7주차. 탐색트리

iming03 · 방금 전 · 비공개

AVL트리 이진트리 탐색트리

알고리즘 스터디

목록 보기 4/4

7주차

백준 1620번 : 탐색트리 이용

입력

첫째 줄에는 도감에 수록되어 있는 포켓몬의 개수 N이랑 내가 맞춰야 하는 문제의 개수 M이 주어져. N과 M은 1보다 크거나 같고, 100,000보다 작거나 같은 자연수인데, 자연수가 뭔지는 알지? 모르면 물어봐도 괜찮아. 나는 언제든지 질문에 답해줄 준비가 되어있어.

둘째 줄부터 N개의 줄에 포켓몬의 번호가 1번인 포켓몬부터 N번에 해당하는 포켓몬까지 한 줄에 하나씩 입력으로 들어와. 포켓몬의 이름은 모두 영어로만 이루어져있고, 또, 음... 첫 글자만 대문자이고, 나머지 문자는 소문자로만 이루어져 있어. 아참! 일부 포켓몬은 마지막 문자만 대문자일 수도 있어. 포켓몬 이름의 최대 길이는 20, 최소 길이는 2야. 그 다음 줄부터 총 M개의 줄에 내가 맞춰야하는 문제가 입력으로 들어와. 문제가 알파벳으로만 들어오면 포켓몬 번호를 말해야 하고, 숫자로만 들어오면, 포켓몬 번호에 해당하는 문자를 출력해야해. 입력으로 들어오는 숫자는 반드시 1보다 크거나 같고, N보다 작거나 같고, 입력으로 들어오는 문자는 반드시 도감에 있는 포켓몬의 이름만 주어져. 그럼 화이팅!!!

출력

첫째 줄부터 차례대로 M개의 줄에 각각의 문제에 대한 답을 말해줬으면 좋겠어!!!. 입력으로 숫자가 들어왔다면 그 숫자에 해당하는 포켓몬의 이름을, 문자가 들어왔으면 그 포켓몬의 이름에 해당하는 번호를 출력하면 돼. 그럼 땡큐~

이게 오박사님이 나에게 새로 주시려고 하는 도감이야. 너무 가지고 싶다ㅠㅜ. 꼭 만점을 받아줬으면 좋겠어!! 파이팅!!!

문제 설명 : N개의 포켓몬이 주어지고 그 다음에 M개의 문제가 주어진다. M개의 문제에서 이름이 입력되면 번호를 출력하거나 번호가 입력되어있으면 이름을 출력해야한다.

탐색트리

이진 탐색트리: 내부노드에 (키, 원소) 쌍을 저장하며 다음의 성질을 만족하는 이진트리 : u, v, w는 모두 트리노드며 u와 w가 각각 v의 왼쪽과 오른쪽 부트리에 존재할 때 다음이 성립 (key(u) < key(v) < key(w))

AVL 트리: x트리 T의 모든 내부노드 v에 대해 v의 자식들의 좌우 높이 차이가 1을 넘지 않는 이진 탐색트리(높이 균형 속성)

문제 풀기 전:

일단 순대로 트리에 insert하고 search로 찾기

그런데 숫자로 찾는 건 가능하지만 문자로 찾을 때 정렬되어 있지 않으므로 힘듦 → 받는 순서를 key로 저장하고 문자순으로 정렬할까함 (대문자 소문자 유의)

2초 안에 풀어야하는데 노드로 하면 시간초과가 나는 듯 했음. 그래도 일단 탐색트리로 풀어봄

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#pragma warning(disable:4996)

struct TreeNode {
    int key;
    char name[21];
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* insert(struct TreeNode* root, int key, const char* name) {
    if (root == NULL) {
        struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        newNode->key = key;
        strncpy(newNode->name, name, sizeof(newNode->name));
        newNode->left = newNode->right = NULL;
        return newNode;
    }

    if (key < root->key) {
        root->left = insert(root->left, key, name);
    }
    else {
        root->right = insert(root->right, key, name);
    }

    return root;
}
```

```
int searchByName(struct TreeNode* root, const char* name) {
    if (root == NULL) {
        return -1;
    }

    int cmp = strcmp(name, root->name);
    if (cmp == 0) {
        return root->key;
    }
    else if (cmp < 0) {
        return searchByName(root->left, name);
    }
    else {
        return searchByName(root->right, name);
    }
}

char* searchByNumber(struct TreeNode* root, int key) {
    if (root == NULL) {
        return -1; // 찾지 못한 경우
    }

    if (key == root->key) {
        return root->name;
    }
    else if (key < root->key) {
        return searchByNumber(root->left, key);
    }
    else {
        return searchByNumber(root->right, key);
    }
}

int main() {
    int N, M;
    scanf("%d %d", &N, &M);
    struct TreeNode* root = NULL;

    for (int i = 1; i <= N; i++) {
        char name[21];
        scanf("%s", name);
        root = insert(root, i, name);
    }

    while (M--) {
        char question[21];
        scanf("%s", question);

        if (isdigit(question[0])) {
            int key = atoi(question);
            printf("%s\n", searchByNumber(root, key));
        }
        else {
            printf("%d\n", searchByName(root, question));
        }
    }

    return 0;
}
```

이렇게 했는데 문자로 찾으면 숫자 반환을 못함...자꾸 찾지 못했다고 -1을 반환함,,

근데 생각해보니까 문자열 순으로 정렬을 안해서였음 ㅎ

키 순으로 저장했던 것처럼 문자열 순으로 정렬해서 넣었더니 visual에서도 시간이 오래걸려서인지 출력이 암것도 안됨,,,

> 난관 봉착.... 어카노,,

블로그들을 참고하니까 애초에 구조체 배열에 qsort를 이용해서 정렬하고 번호로 찾을 때에는 그냥 리스트의 인덱스를 출력해버리고 문자로 찾을 때에만 이진탐색을 이용하더라,,,

퀵정렬을 통해서 푼 블로그를 참고해서 고쳐봤는데

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#pragma warning(disable:4996)

struct TreeNode {
    int key;
    char name[21];
    struct TreeNode* left;
    struct TreeNode* right;
};

struct TreeNode* insert(struct TreeNode* root, int key, const char* name) {
    if (root == NULL) {
        struct TreeNode* newNode = (struct TreeNode*)malloc(sizeof(struct TreeNode));
        newNode->key = key;
        strncpy(newNode->name, name, sizeof(newNode->name));
        newNode->left = newNode->right = NULL;
        return newNode;
    }

    if (key < root->key) {
        root->left = insert(root->left, key, name);
    }
    else {
        root->right = insert(root->right, key, name);
    }

    return root;
}

int searchByName(struct TreeNode* root, const char* name) {
    if (root == NULL) {
        return -1; // 찾지 못한 경우
    }

    int cmp = strcmp(name, root->name);
    if (cmp == 0) {
        return root->key;
```

```
        }
        else if (cmp < 0) {
            return searchByName(root->left, name);
        }
        else {
            return searchByName(root->right, name);
        }
    }

    char* searchByNumber(struct TreeNode* root, int key) {
        if (root == NULL) {
            return "X"; // 찾지 못한 경우
        }

        if (key == root->key) {
            return root->name;
        }
        else if (key < root->key) {
            return searchByNumber(root->left, key);
        }
        else {
            return searchByNumber(root->right, key);
        }
    }

    void quickSort(struct TreeNode* root, int l, int r) {
        int a, b;
        if (l >= r)
            return;
        a = b = inPlacePartition(root, l, r, r);
        quickSort(root, l, a - 1);
        quickSort(root, b + 1, r);
    }

    int inPlacePartition(struct TreeNode* root, int l, int r, int k) {
        struct TreeNode p;
        struct TreeNode tmp;
        int i, j;
        p = root[k];

        tmp = root[k];
        root[k] = root[r];
        root[r] = tmp;

        i = l; j = r - 1;

        while (i <= j) {
            while (i <= j && strcmp(root[i].name, p.name) <= 0) {
                i++;
            }
            while (i <= j && strcmp(root[i].name, p.name) >= 0) {
                j--;
            }
            if (i < j) {
                tmp = root[i];
                root[i] = root[j];
                root[j] = tmp;
            }
        }
        tmp = root[i];
        root[i] = root[r];
        root[r] = tmp;
```

```
        return i;
    }

    int main() {
        int N, M;
        scanf("%d %d", &N, &M);
        struct TreeNode* root = NULL;

        for (int i = 1; i <= N; i++) {
            char name[21];
            scanf("%s", name);
            root = insert(root, i, name);
        }

        while (M--) {
            char question[21];
            scanf("%s", question);

            if (isdigit(question[0])) {//숫자일 경우
                int key = atoi(question);
                printf("%s\n", searchByNumber(root, key));
            }
            else { //문자일 경우
                quickSort(root, 0, N - 1);
                printf("%d\n", searchByName(root, question));
            }
        }

        return 0;
    }
```

이번에도 문자 찾을 때 시간초과이다...

강민돌

민돌이의 공부

이전 포스트

4주차 퀵정렬, 합병정렬

0개의 댓글

댓글을 작성하세요

댓글 작성