Algorithms Lab Assignment 5

Pranav Gade

April 10, 2022

Batch: CS&AI
Roll no.: LCI2020010

1 Order Statistics for finding smallest, largest and ith element

Implementation of order Statistics for finding smallest, largest and ith element using binary tree.

1.1 Code

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

typedef struct Node {
    int value;
    struct Node* left;
    struct Node* right;
} Node;

void insert_into_tree(Node* root, Node* val) {
    if (val->value < root->value) {
        // insert in left
        if (root->left == NULL) root->left = val;
        else insert_into_tree(root->left, val);
    } else {
        // insert in right
        if (root->right == NULL) root->right = val;
        else insert_into_tree(root->right, val);
    }
}

int find_kth_element(Node* root, int* k) {
    int ret;
    if (root->left != NULL) {
        ret = find_kth_element(root->left, k);
    }
    if (*k == 0) return ret;
    (*k)--;
    if (*k == 0) return root->value;
```

```
    if (root->right != NULL) {
        ret = find_kth_element(root->right, k);
    }
    if (*k == 0) return ret;

    return -1;
}

int main(int argc, char** argv) {
    FILE* file;
    file = fopen("../kth_order_stat_input.txt", "r");
    int k;
    fscanf(file, "%d\n", &k);
    int n;
    fscanf(file, "%d\n", &n);
    int* nums = malloc(sizeof(int) * n); // not free'd
    for (int i = 0; i < n; ++i) {
        fscanf(file, "%d\n", &nums[i]);
    }

    int min = nums[0];
    int max = nums[0];
    Node* root = malloc(sizeof(Node));
    root->value = nums[0];
    root->left = NULL;
    root->right = NULL;
    for (int i = 1; i < n; ++i) {
        Node* val = malloc(sizeof(Node));
        val->value = nums[i];
        val->left = NULL;
        val->right = NULL;
        insert_into_tree(root, val);
        if (min > nums[i]) min = nums[i];
        if (max < nums[i]) max = nums[i];
    }

    int kth = find_kth_element(root, &k);

    FILE* out;
    out = fopen("../kth_order_stat_output.txt", "w");

    fprintf(out, "min: %d\n", min);
    fprintf(out, "max: %d\n", max);
    fprintf(out, "kth: %d\n", kth);

    return 0;
}
```

1.2 Input

```
5
10
5
1
3
2
4
```

```
6
8
9
0
7
```

1.3 Output

```
min: 0
max: 9
kth: 4
```

2 Order Statistics for finding smallest, largest and ith element using AVL tree

Implementation of order Statistics for finding smallest, largest and ith element using binary tree using AVL tree.

2.1 Code

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

typedef struct Node {
    int value;
    struct Node* left;
    struct Node* right;
    int height;
} Node;

int get_height(Node* node) {
    if (node == NULL) return 0;
    return node->height;
}

Node* right_rotate(Node* node) {
    Node* x = node->left;
    Node* t = x->right;

    x->right = node;
    node->left = t;

    int a = get_height(node->left);
    int b = get_height(node->right);
    node->height = ((a > b) ? a : b) + 1;
    int a1 = get_height(x->left);
    int b1 = get_height(x->right);
    x->height = ((a1 > b1) ? a1 : b1) + 1;

    return x;
}

Node* left_rotate(Node* node) {
    Node* r = node->right;
```

```
    Node* l = r->left;

    r->left = node;
    node->right = l;

    int a = get_height(node->left);
    int b = get_height(node->right);
    node->height = ((a > b) ? a : b) + 1;
    int a1 = get_height(r->left);
    int b1 = get_height(r->right);
    r->height = ((a1 > b1) ? a1 : b1) + 1;

    return r;
}

int get_balance(Node* node) {
    if (node == NULL)
        return 0;
    return get_height(node->left) - get_height(node->right);
}

Node* insert_into_tree(Node* root, Node* val) {
    if (root == NULL)
        return val;

    if (val->value < root->value)
        root->left = insert_into_tree(root->left, val);
    else if (val->value > root->value)
        root->right = insert_into_tree(root->right, val);
    else
        return root;

    int a = get_height(root->left);
    int b = get_height(root->right);
    root->height = 1 + ((a > b) ? a : b);

    int balance = get_balance(root);
    if (balance > 1 && val->value < root->left->value)
        return right_rotate(root);

    if (balance < -1 && val->value > root->right->value)
        return left_rotate(root);

    if (balance > 1 && val->value > root->left->value) {
        root->left = left_rotate(root->left);
        return right_rotate(root);
    }

    if (balance < -1 && val->value < root->right->value) {
        root->right = right_rotate(root->right);
        return left_rotate(root);
    }

    return root;
}

int find_kth_element(Node* root, int* k) {
```

```
92      int ret;
93      if (root->left != NULL) {
94          ret = find_kth_element(root->left, k);
95      }
96      if (*k == 0) return ret;
97      (*k)--;
98      if (*k == 0) return root->value;
99      if (root->right != NULL) {
100         ret = find_kth_element(root->right, k);
101     }
102     if (*k == 0) return ret;
103
104     return -1;
105 }
106
107 int main(int argc, char* *argv) {
108     FILE* file;
109     file = fopen("../kth_order_stat_avl_input.txt", "r");
110     int k;
111     fscanf(file, "%d\n", &k);
112     int n;
113     fscanf(file, "%d\n", &n);
114     int* nums = malloc(sizeof(int)*  n); // not free'd
115     for (int i = 0; i < n; ++i) {
116         fscanf(file, "%d\n", &nums[i]);
117     }
118
119     int min = nums[0];
120     int max = nums[0];
121     Node* root = malloc(sizeof(Node));
122     root->value = nums[0];
123     root->left = NULL;
124     root->height = 1;
125     for (int i = 1; i < n; ++i) {
126         Node* val = malloc(sizeof(Node));
127         val->value = nums[i];
128         val->left = NULL;
129         val->right = NULL;
130         val->height = 1;
131         root = insert_into_tree(root, val);
132         if (min > nums[i]) min = nums[i];
133         if (max < nums[i]) max = nums[i];
134     }
135
136     int kth = find_kth_element(root, &k);
137
138     FILE* out;
139     out = fopen("../kth_order_stat_avl_output.txt", "w");
140
141     fprintf(out, "min: %d\n", min);
142     fprintf(out, "max: %d\n", max);
143     fprintf(out, "kth: %d\n", kth);
144
145     return 0;
146 }
```

2.2 Input

```
5
10
5
1
3
2
4
6
8
9
0
7
```

2.3 Output

```
min: 0
max: 9
kth: 4
```

3 Footnotes

Code to generate graphs and this file is on github