자료구조 과제 5 보고서

Jumagul Alua, 20231632

1. Insertion and deletion for a max heap

Input1.txt 입력 파일을 읽들어 자식보다 크거나 같은 최대 힙을 찾아낼 프로그램을 완성했다.

```
typedef struct node *treePointer;
typedef struct node {
    int key;
    treePointer parent;
    treePointer leftChild, rightChild;
} node;
```

일단, 각 노드가 두개 자식을 가지게 되는 구조체로 선언해주었다.

```
treePointer queue[1000]
```

그리고 BFS 탐색을 사용하기 위해서 큐를 선언해주었다.

```
treePointer create(int key) {
    treePointer newNode = (treePointer)malloc(sizeof(node));
    newNode->key = key;
    newNode->parent = NULL;
    newNode->leftChild = NULL;
    newNode->rightChild = NULL;
    return newNode;
}
```

주어진 키 값을 가지고 새로운 노드를 동적으로 할당하고 초기화하는 create 함수이다.

```
treePointer find(treePointer node) {
    if (!node) {
        return NULL;
    }

    treePointer last = NULL;
    //treePointer queue[1000];
    int front = 0;
    int rear = 0;

    queue[rear++] = node;

    while (front < rear) {
        treePointer current = queue[front++];
        last = current;

        if (current->leftChild){
            queue[rear++] = current->leftChild;
        }
        if (current->rightChild) {
            queue[rear++] = current->rightChild;
        }
    }
    return last;
}
```

Find 하무수에는 level-order traversal 를 사용하여 마지막에 추가된 노드를 찾아준다.

```
void up(treePointer node) {
    while ((node->parent) && (node->key)>(node->parent->key)) {
        swap(node, node->parent);
        node = node->parent;
    }
}
```

Up 함수는 새로운 노드가 힙의 조건을 만족하도록 이동하는 작업을 한다. 노드를 부모와 비교하여 부모보다 큰 경우에는 조건을 만족할 때까지 반복하면서 부모와 위치를 교환한다.

```
void down(treePointer node) {
    while (node) {
        treePointer maxNode = node;

        if ((node->leftChild) && (node->leftChild->key > maxNode->key)) {
            maxNode = node->leftChild;
        }
        if ((node->rightChild) && (node->rightChild->key > maxNode->key)) {
            maxNode = node->rightChild;
        }
        if (maxNode == node) {
            break;
        }
        swap(node, maxNode);
        node = maxNode;
    }
}
```

Down 함수는 노드의 자식 중에서 큰 값이 있는 경우에는 조건을 만족할 때까지 반복하면서 해당 자식과 위치를 교환한다.

```
int exists(int key) {
    //treePointer queue[1000];
    int front = 0;
    int rear = 0;
    if (root) {
        queue[rear++] = root;
    }
    while (front<rear) {
        treePointer current = queue[front++];
        if (current->key == key) {
            return 1;
        }
        if (current->leftChild) {
            queue[rear++] = current->leftChild;
        }
        if (current->rightChild) {
            queue[rear++] = current->rightChild;
        }
    }
    return 0;
}
```

BFS 을 사용하여 트리를 탐색하며, 키 값이 발견되면 1 을 반환하고 그렇지 않으면 0 을 반환하여 키 값이 힙에 존재하는지 확인하는 함수이다.

```
void insert(int key) {
    if (exists(key)) {
        printf("Exist number\n");
        return;
    }
    treePointer newNode = create(key);
    if (!root) {
        root = newNode;
    } else {
        treePointer last = find(root);
        if (!last->leftChild) {
            last->leftChild = newNode;
            newNode->parent = last;
        } else {
            last->rightChild = newNode;
            newNode->parent = last;
        }
        up(newNode);
    }
    printf("Insert %d\n", key);
}
```

키 값이 이미 존재하는지 exists 함수를 사용하여 확인하고, 없다면 create 함수를 사용하여 새로운 노드를 생성하고 힙의 조건을 만족하도록 up 함수를 호출하여 상향 이동시키는 삽입 insert 함수이다.

```
void delete() {
    if (!root) {
        printf("The heap is empty\n");
        return;
    }

    int max = root->key;

    treePointer last = find(root);
    if (last == root) {
        free(root);
        root = NULL;
    } else {
        root->key = last->key;
        if (last->parent->leftChild == last) {
            last->parent->leftChild = NULL;
        } else {
            last->parent->rightChild = NULL;
        }
        free(last);
        down(root);
    }
    printf("Delete %d\n", max);
}
```

힙이 비어 있는지 확인하고 루트 노드를 찾아서 삭제한다. 만약 루트 노드가 아니라면 삭제된 위치에 마지막 노드의 값을 복사하고 마지막 노드를 삭제한 후에 down 함수를 호출하여 하향 이동시키는 함수이다.

```c
int main() {
    char command;
    int key;
    FILE *input = fopen("input1.txt", "r");
    FILE *output = freopen("output1.txt", "w", stdout);
    while (fscanf(input, " %c", &command) != EOF) {
        switch (command) {
            case 'i':
                fscanf(input, "%d", &key);
                insert(key);
                break;
            case 'd':
                delete();
                break;
            case 'q':
                fclose(input);
                fclose(output);
                return 0;
            default:
                break;
        }
    }
    fclose(input);
    fclose(output);
    return 0;
}
```

미지막으로 main 함수에는 입력 파일에서 명령어를 읽어들여 해당 명령어에 따라 삽입, 삭제 등의 작업을 한다. 그리고 결과적으로 각 작업의 결과는 출력 파일인 'output1.txt'에 쓰인다.

2. Traversal a binary search tree

   Preorder 순서에서 inoreder 과 postorder traversal 순서로 결과를 출력하는 이진 탐색 트리를 구성하는 프로그램이다.

   문제 1 과 유사하게 두 개의 구조체를 선언해준다.

```
TreeNode* construct(int preorder[], int* indx, int key, int min, int max, int size) {
    if (*indx >= size) {
        return NULL;
    }

    TreeNode* root = NULL;

    if (key>min && key<max) {
        root = create(key);
        *indx += 1;

        if (*indx<size) {
            root->left = construct(preorder, indx, preorder[*indx], min, key, size);
        }

        if (*indx<size) {
            root->right = construct(preorder, indx, preorder[*indx], key, max, size);
        }
    }
    return root;
}
```

그 다음, 주어진 배열을 이용하여 binary search tree 를 제귀적으로 구성해주는 create 함수를 만든다.

```
void inorder(TreeNode* root) {
    if (root == NULL) {
        return;
    }
    inorder(root->left);
    printf("%d ", root->key);
    inorder(root->right);
}

void postorder(TreeNode* root) {
    if (root == NULL) {
        return;
    }
    postorder(root->left);
    postorder(root->right);
    printf("%d ", root->key);
}
```

수업 시간에 배운 inorder 하고 postorder traversal 함수를 이요하여 키 값을 출력한다.

```
void freeTree(TreeNode* root) {
    if (root == NULL) {
        return;
    }
    freeTree(root->left);
    freeTree(root->right);
    free(root);
}

int ifhas(int arr[], int size) {
    for (int i=0; i<size-1; i++) {
        for (int j=i+1; j<size; j++) {
            if (arr[i] == arr[j]) {
                return 1;
            }
        }
    }
    return 0;
}
```

트리의 모든 노드를 해제하는 free 함수와 중복된 값이 있는지 확인해주는 ifhas 함수도 넣어준다.

```c
int main() {
    FILE *input = fopen("input2.txt", "r");
    FILE *output = freopen("output2.txt", "w", stdout);
    if (input == NULL) {
        fprintf(stderr, "Error opening input file.\n");
        return 1;
    }
    int n;
    fscanf(input, "%d", &n);
    int preorder[n];
    for (int i = 0; i < n; i++) {
        fscanf(input, "%d", &preorder[i]);
    }
    fclose(input);
    if (ifhas(preorder, n)) {
        printf("cannot construct BST\n");
        fclose(output);
        return 0;
    }
    int indx = 0;
    TreeNode* root = construct(preorder, &indx, preorder[0], INT_MIN, INT_MAX, n);

    if (root == NULL) {
        printf("cannot construct BST\n");
        fclose(output);
        return 0;
    }
    printf("Inorder: ");
    inorder(root);
    printf("\n");

    printf("Postorder: ");
    postorder(root);
    printf("\n");

    freeTree(root);
    fclose(output);
```

그리고, 마지막으로 main 함수에는, 'input2.txt' 파일을 입력으로 받고 만약에 중복된 값이 있거나 root 널이라면, cannot construct BST 라는 출력을 준다. 아닌 경우 배열의 inorder 과 postorder 함수를 불러서 출력 파일인 'output2.txt' 파일에 넣어준다.

3. Binary search tree 를 사용해서 우선순위 큐를 나타내는 프로그램이다.
   앞 문제들처럼 트리 구저체로 선언해준다.

```c
TreeNode* insert(TreeNode* root, int key, int* success) {
    if (root == NULL) {
        *success = 1;
        return createNode(key);
    }
    if (key<root->key) {
        root->left = insert(root->left, key, success);
    } else if (key>root->key) {
        root->right = insert(root->right, key, success);
    } else {
        *success = 0;
    }
    return root;
}
```

주어진 키 값을 트리에 산입하고 중복된 키가 있다면 산입이 실패로 표시되게 만든 insert 함수이다.

```c
TreeNode* find(TreeNode* root) {
    while (root && root->right != NULL) {
        root = root->right;
    }
    return root;
}

TreeNode* delete(TreeNode* root, int* max) {
    if (root == NULL) {
        return NULL;
    }
    if (root->right == NULL) {
        *max = root->key;
        TreeNode* leftChild = root->left;
        free(root);
        return leftChild;
    }
    root->right = delete(root->right, max);
    return root;
}

void freeTree(TreeNode* root) {
    if (root == NULL) {
        return;
    }
    freeTree(root->left);
    freeTree(root->right);
    free(root);
}
```

트리에서 최대 키 값을 찾아내는 find 함수와, 그 노드를 삭제하는 delete 함수와, 트리의 모든 노드를 해제하는 free 함수를 넣어준다.

```c
int main() {
    TreeNode* root = NULL;
    char buffer[100];
    char command[10];
    int key;
    FILE *input = fopen("input3.txt", "r");
    FILE *output = freopen("output3.txt", "w", stdout);
    if (input == NULL) {
        fprintf(stderr, "Error opening input file.\n");
        return 1;
    }

    while (fgets(buffer, sizeof(buffer), input)) {
        if (sscanf(buffer, "%s %d", command, &key) == 2) {
            if (strcmp(command, "push") == 0) {
                int success;
                root = insert(root, key, &success);
                if (success) {
                    printf("Push %d\n", key);
                } else {
                    printf("Exist number\n");
                }
            }
        }
        else if (sscanf(buffer, "%s", command) == 1) {
            if (strcmp(command, "top") == 0) {
                TreeNode* maxNode = find(root);
                if (maxNode) {
                    printf("The top is %d\n", maxNode->key);
                } else {
                    printf("The queue is empty\n");
                }
            }
            else if (strcmp(command, "pop") == 0) {
                if (root) {
                    int max;
                    root = delete(root, &max);
                    printf("Pop %d\n", max);
                } else {
                    printf("The queue is empty\n");
                }
            }
            else if (strcmp(command, "q") == 0) {
                fclose(input);
                fclose(output);
                freeTree(root);
                return 0;
            }
        }
    }

    fclose(input);
    fclose(output);
    freeTree(root);
    return 0;
}
```

입력 파일에서 명령어를 읽어들여 BST 에 삽입, 삭제 등의 작업을 수행하는 main 함수이다.

- push key: 키 값을 트리에 삽입한다. 이미 존재하는 경우 "Exist number"를 출력한다.
- top: 트리에서 최대 키 값을 출력한다. 트리가 비어 있으면 "The queue is empty"를 출력한다.
- pop: 트리에서 최대 키 값을 가진 노드를 삭제하고, 그 값을 출력한다. 트리가 비어 있으면 "The queue is empty"를 출력한다.

- q: 프로그램을 종료하고, 트리의 메모리를 해제한다.
- 입력 파일과 출력 파일을 닫고, 트리의 메모리를 해제한.