Data Structure HW5

20231515
컴퓨터공학과
김다은

<목차>

1. 문제 1
2. 문제 2
3. 문제 3

문제 1.

max heap binary tree 를 구성하는 것이 핵심인 문제였다. max heap binary tree 의 경우, parent node 의 key 값이 leftchild, rightchild node 의 key 값보다 커야 한다. 이 점에 유의하여 함수들을 작성하였다.

문제 조건으로 주어진 treePointer, node 자료구조를 사용하였다.

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

프로그램 실행 단계:

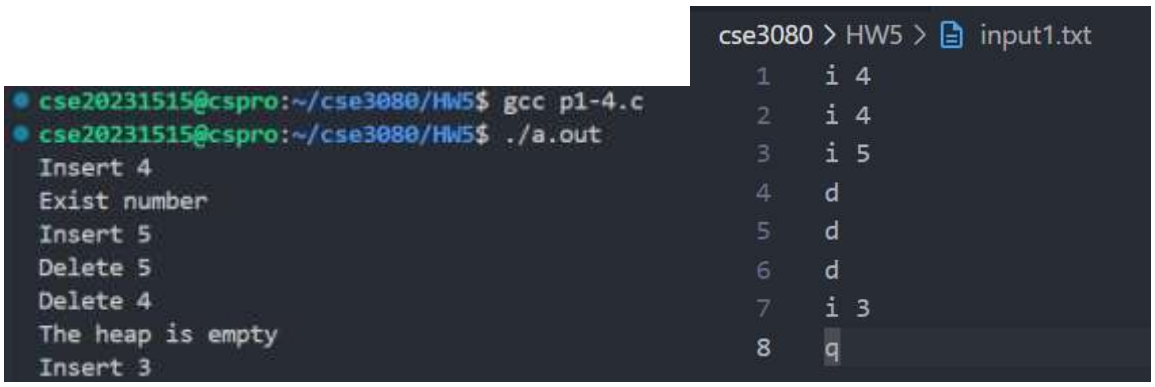
1. main 함수에서 input1.txt 파일의 내용을 읽어온다.
2. q 가 입력되었다면 프로그램을 종료한다.
3. i 가 입력되었다면 key 값을 추가로 입력 받고, insert 함수를 호출한다.
    A. 해당 key 를 가진 노드가 이미 존재하는지 확인한다.
    B. 추가할 맨 마지막 노드의 위치에서부터 root 로 이동하면서 max heap 조건을 만족하도록 swap 한다.
4. d 가 입력되었다면, deleteroot 함수를 호출한다.
    A. 트리가 비어 있다면 에러 메시지를 출력한다.
    B. root 노드의 key 값과 맨 마지막 노드의 key 값을 swap 한다.
    C. root 노드에서부터 child 노드로 내려가면서 max heap 의 조건을 만족하도록 swap 하며 tree 를 수정한다.
5. 1~4 과정을 반복한다.

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

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

// 같은 key 를 가지는 노드가 존재하는지 search
```

```
int search(treePointer head, int target) {
    if(head) {
        if(head->key == target) return 1;
        else {
            return search(head->leftChild, target) || search(head->rightChild,
target);
        }
    }
    return 0;
}

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

void insert(treePointer** root, int key, int *n) {
    treePointer *heap = *root;
    // 해당 key 를 가진 노드가 이미 존재하는지 확인
    if(search(heap[1], key)) {
        printf("Exist number\n");
        return;
    }

    treePointer newnode = createNode(key);
    int i = ++(*n);
    heap[i] = newnode;
    if(i != 1) {
        newnode->parent = heap[i/2];
        if(heap[i/2]->leftChild == NULL) {
            heap[i/2]->leftChild = newnode;
        }
        else {
            heap[i/2]->rightChild = newnode;
        }
    }

    // root 로 올라가면서 swap
    while((i!=1) && (key > heap[i/2]->key)) {
        heap[i]->key = heap[i/2]->key;
        i /= 2;
    }
    heap[i]->key = key;
    printf("Insert %d\n", key);
}

void delete(treePointer** root, int *n) {
    treePointer *heap = *root;
    if(*n == 0) {
        printf("The heap is empty\n");
        return;
    }

    int parent = 1, child = 2;
```

```
    int delkey = heap[1]->key;
    int tarkey = heap[(*n)]->key;
    // swap
    heap[1]->key = heap[(*n)]->key;

    if(*n != 1) {
        if((*n)%2 == 0) {
            heap[(*n)/2]->leftChild = NULL;
        }
        else heap[(*n)/2]->rightChild = NULL;
    }

    free(heap[(*n)]);
    heap[(*n)--] = NULL;

    if(*n == 0) {
        printf("Delete %d\n", delkey);
        return;
    }

    // root 에서부터 내려가면서 swap
    while(child <= *n) {
        // left, right 중 키 값이 큰 것으로 child 선택
        if(child < *n && heap[child]->key < heap[child+1]->key) {
            child++;
        }
        // child 보다 key 가 큰 경우
        if(tarkey >= heap[child]->key) break;
        // child 보다 key 가 작은 경우
        heap[parent]->key = heap[child]->key;
        parent = child;
        child *=2;
    }
    heap[parent]->key = tarkey;
    printf("Delete %d\n", delkey);
}

int main() {
    char c;
    int key, n = 0;
    treePointer *heap = (treePointer*)malloc(sizeof(treePointer)*1000);
    FILE* fin = fopen("input1.txt", "r");
    fscanf(fin, "%c", &c);
    while(c != 'q') {
        if(c == 'i') {
            fscanf(fin, "%d", &key);
            insert(&heap, key, &n);
        }
        else if(c == 'd') {
            delete(&heap, &n);
        }
        fscanf(fin, "%c", &c);
    }
    free(heap);
    fclose(fin);
}
```

실제 입력 결과:


cse20231515@cspro:~/cse3080/HW5$ gcc p1-4.c
cse20231515@cspro:~/cse3080/HW5$ ./a.out
Insert 4
Exist number
Insert 5
Delete 5
Delete 4
The heap is empty
Insert 3
cse3080 > HW5 > input1.txt
1 i 4
2 i 4
3 i 5
4 d
5 d
6 d
7 i 3
8 q

문제 2.

input 파일을 읽어 Binary Search Tree 를 구성하고, Inorder 순환 방식과 postorder 순환 방식을 이용해 BST 의 노드를 순환하는 프로그램을 만드는 문제였다.

BST 의 경우 parent node 의 key 값이 rightchild 의 key 값 보다는 작다. 또한 parent node 의 key 값이 leftchild 의 key 값 보다는 크다는 것이 특징이다. 이 점을 유의하여 함수들을 작성하였다.

BST 를 구성하기 위해 다음과 같은 자료구조를 사용하였다.

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

프로그램 실행 단계:

1. main 함수에서 input2.txt 의 내용을 읽어온다.
2. input2.txt 에 주어진 key 의 수만큼 insert 함수를 호출한다.
    A. makenode 함수를 호출하여 입력된 key 값을 가지는 새로운 노드를 생성한다.
    B. 이미 tree 에 같은 key 를 가지는 노드가 존재하는 경우 에러 메시지를 출력한 뒤 프로그램을 종료한다.
    C. MST 의 조건을 만족하는 위치에 노드를 삽입한다.
3. inorder 함수를 호출한다.
4. postorder 함수를 호출한다.

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

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

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

int insert(treePointer* head, int key) {
    treePointer newnode = makenode(key);
    treePointer ptr = *head;
    // 맨 첫 노드로 삽입하는 경우
    if(*head == NULL) {
        *head = newnode;
        return 1;
    }
    treePointer left = ptr->leftChild, right = ptr->rightChild;
    while(ptr) {
        if(ptr->key == key) {
            printf("cannot construct BST\n");
            return 0;
        }
        // ptr 의 오른쪽으로
        else if(ptr->key < key) {
            if(right == NULL) {
                // 삽입
                ptr->rightChild = newnode;
                newnode->parent = ptr;
                break;
            }
            ptr = ptr->rightChild;
        }
        // ptr 의 왼쪽으로
        else {
            if(left == NULL) {
                // 삽입
                ptr->leftChild = newnode;
                newnode->parent = ptr;
                break;
            }
```

```
            ptr = ptr->leftChild;
        }
        left = ptr->leftChild;
        right = ptr->rightChild;
    }
    return 1;
}

void inorder(treePointer head) {
    if(head) {
        // int i = head->leftChild ? head->leftChild->key : 0;
        // int j = head->rightChild ? head->rightChild->key : 0;
        // printf("\n| %d: %d %d\n", head->key, i, j);
        inorder(head->leftChild);
        printf("%d ", head->key);
        inorder(head->rightChild);
    }
}

void postorder(treePointer head) {
    if(head) {
        postorder(head->leftChild);
        postorder(head->rightChild);
        printf("%d ", head->key);
    }
}

int main() {
    FILE* fin = fopen("input2.txt", "r");
    int n, key;
    treePointer head = NULL;

    fscanf(fin, "%d", &n);
    for(int i=0; i<n; i++) {
        fscanf(fin, "%d", &key);
        if(insert(&head, key) == 0) {
            return 0;
        }
    }

    // Inorder traversal
```

```
    printf("Inorder: ");
    inorder(head);
    printf("\n");

    // Postorder traversal
    printf("Postorder: ");
    postorder(head);
    printf("\n");

    fclose(fin);
}
```

실제 입력 결과

```
cse20231515@cspro:~/cse3080/HW5$ ./a.out
Inorder: 2 5 30 35 40 80
Postorder: 2 5 35 80 40 30
```

```
1   6
2   30 5 2 40 35 80
```

문제 3.

Binary Search Tree 를 관리하는 것이 핵심인 문제였다. BST 의 경우 parent node 의 key 값이 rightchild 의 key 값 보다는 작다. 또한 parent node 의 key 값이 leftchild 의 key 값 보다는 크다는 것이 특징이다. 이 점을 유의하여 함수들을 작성하였다.

구현을 위해 다음과 같은 자료구조를 사용하였다.

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

프로그램의 진행 단계는 다음과 같다.

1. main 함수에서 input3.txt 파일로부터 입력 값을 읽어온다.
2. q 가 입력된 경우 프로그램을 종료한다.
3. push 가 입력된 경우 추가로 key 값을 읽어온 뒤, insert 함수를 호출한다.
   A. makenode 함수를 호출하여 key 값을 가지는 새로운 노드를 생성한다.
   B. 이미 tree 에 동일한 key 를 가지는 노드가 존재한다면, 에러메세지를 출력한 뒤 함수를 종료시킨다.
   C. BST 의 조건을 만족하는 위치에 노드를 삽입한다.
4. top 이 입력된 경우 printtop 함수를 호출한다.
   A. tree 가 비어 있는 경우 에러메세지를 출력한 뒤 함수가 종료된다.
   B. 가장 큰 key 값을 가지는 노드의 key 값을 형식에 맞추어 출력한다.
5. pop 이 입력된 경우 popnode 함수를 호출한다.
   A. tree 가 비어 있는 경우 에러메세지를 출력한 뒤 함수를 종료시킨다.
   B. 가장 큰 key 를 가지는 노드를 제거한다.
6. 1~6 과정을 반복한다.

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

typedef struct node *treePointer;
typedef struct node {
```

```
    int key;
    treePointer parent;
    treePointer leftChild, rightChild;
} node;

treePointer makenode(int key) {
    treePointer newnode = (treePointer)malloc(sizeof(node));
    newnode->key = key;
    newnode->parent = NULL;
    newnode->leftChild = NULL;
    newnode->rightChild = NULL;
    return newnode;
}

void insert(treePointer* head, int key) {
    treePointer newnode = makenode(key);
    treePointer ptr = *head;
    // 맨 첫 노드로 삽입하는 경우
    if(*head == NULL) {
        *head = newnode;
    }
    else {
        treePointer left = ptr->leftChild, right = ptr->rightChild;
        while(ptr) {
            if(ptr->key == key) {
                printf("Exist number\n");
                return;
            }
            // ptr 의 오른쪽으로
            else if(ptr->key < key) {
                if(right == NULL) {
                    ptr->rightChild = newnode;
                    newnode->parent = ptr;
                    break;
                }
                ptr = ptr->rightChild;
            }
            // ptr 의 왼쪽으로
            else {
                if(left == NULL) {
                    ptr->leftChild = newnode;
```

```
                    newnode->parent = ptr;
                    break;
                }
                ptr = ptr->leftChild;
            }
            left = ptr->leftChild;
            right = ptr->rightChild;
        }
    }
    printf("Push %d\n", key);
}

void printtop(treePointer head) {
    // tree 가 비어 있는 경우
    if(head == NULL) {
        printf("The queue is empty\n");
        return;
    }
    while(head->rightChild) {
        head = head->rightChild;
    }
    printf("The top is %d\n", head->key);
}

void popnode(treePointer* head) {
    treePointer pre = NULL;
    treePointer tmp = *head;

    if(*head == NULL) {
        printf("The queue is empty\n");
        return;
    }

    while(tmp->rightChild) {
        pre = tmp;
        tmp = tmp->rightChild;
    }
    // head node 를 pop 해야 하는경우
    if(pre == NULL) {
        *head = tmp->leftChild;
    }
```

```
    else pre->rightChild = NULL;
    printf("Pop %d\n", tmp->key);
    free(tmp);
}

int main() {
    char input[10]; int key;
    treePointer head = NULL;
    FILE* fin = fopen("input3.txt", "r");

    fscanf(fin, "%s", input);
    while(strcmp(input, "q") != 0) {
        if(strcmp(input, "push") == 0) {
            fscanf(fin, "%d", &key);
            insert(&head, key);
        }
        else if(strcmp(input, "top") == 0) {
            printtop(head);
        }
        else if(strcmp(input, "pop") == 0) {
            popnode(&head);
        }
        fscanf(fin, "%s", input);
    }

    fclose(fin);
}
```

실제 입력 결과

```
cse20231515@cspro:~/cse3080/HW5$ ./a.out
Push 3
Exist number
The top is 3
Push 5
The top is 5
Pop 5
Exist number
Pop 3
The queue is empty
```

cse3080 > HW5 > input3.txt

```
1   push 3
2   push 3
3   top
4   push 5
5   top
6   pop
7   push 3
8   pop
9   pop
10  q
```