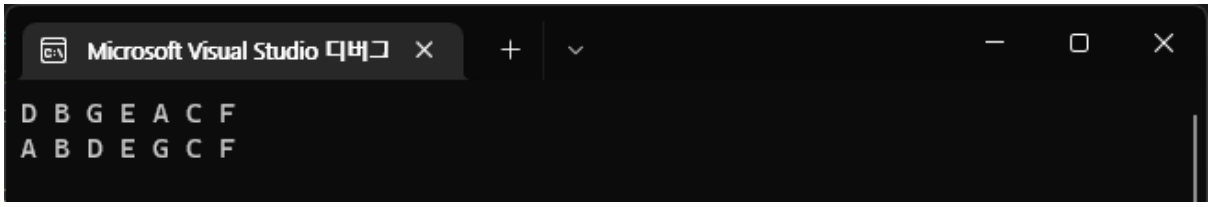
자료구조 과제 4

20211533 남정연

1. 실행 결과



2. 코드 설명

```
threadedPointer initTree(FILE* fp) {
    threadedPointer root = (threadedPointer) malloc(sizeof(threadedTree)); // root
    root->leftChild = root;
    root->leftThread = 1;
    root->rightChild = root;
    root->rightThread = 0;
    threadedPointer search = root;
    char m, c1, c2;
    while (fscanf(fp, " %c", &m) != EOF) {
        if (m == 'I') { // insert
            if (!fscanf(fp, " %c %c", &c1, &c2)) continue;
            threadedPointer child = (threadedPointer) malloc(sizeof(threadedTree));
            child->data = c1;
            if (c2 == 'L') insertLeft(search, child, c1); // left insert
            else if (c2 == 'R') insertRight(search, child, c1); // right insert
        }
        else if (m == 'S') {
            if (!fscanf(fp, " %c", &c1)) continue;
            search = tinorderSearch(root, c1); // perform search
        }
    }
    return root;
}

/*
* inorder의 경우 LVR 형태이므로 현재 노드가 V라면
* predec는 thread L혹은 non thread L의 last R이다.
* 트리의 높이를 h라고 하면 최악의 경우 h번 탐색하므로
* O(h)
*/
threadedPointer inpredec(threadedPointer tree) {
    /* find the inorder predecessor of tree in a threaded binary tree */
    threadedPointer temp;
    temp = tree->leftChild;
    if (!tree->leftThread) // 왼쪽 자식이 thread가 아니라면
        while (!temp->rightThread) // 오른쪽 자식이 thread가 아닌 한
            temp = temp->rightChild; // temp에 오른쪽 자식 대입
    return temp;
}
```

```
/*
* insucc의 시간 복잡도가 O(h) 이므로
* O(h)
*/
void insertLeft(threadedPointer parent, threadedPointer child, char data) {
    /* insert child as the left child of parent in a threaded binary tree */
    threadedPointer temp;
    child->leftChild = parent->leftChild; // left의 경우 부모와 같음
    child->leftThread = parent->leftThread;
    child->rightChild = parent; // right는 부모임
    child->rightThread = 1;
    parent->leftChild = child; // 부모와 자식을 연결
    parent->leftThread = 0;
    if (!child->leftThread) { // 노드를 중간에 삽입하는 경우
        temp = insucc(child); // inorder sucessor를 찾아
        temp->rightChild = child; // thread를 자식으로 설정
    }
}

/*
* inorder traverse를 할 때 최악의 경우 insucc을 V번 호출하므로
* O(Vh)
*/
threadedPointer tinorderSearch(threadedPointer tree, char data) {
    /* traverse the threaded binary tree inorder */
    threadedPointer temp = tree;
    for (; ; ) {
        temp = insucc(temp);
        if (temp == tree || temp->data == data) break;
    }
    return temp;
}

/*
* 함수가 V번 호출되므로
* O(V)
*/
void tpreorder(threadedPointer tree) {
    /* traverse the threaded binary tree preorder */
    printf("%c ", tree->data); // V 출력

    if (!tree->leftThread) { // L 처리
        tpreorder(tree->leftChild);
    }

    if (!tree->rightThread) { // R 처리
        tpreorder(tree->rightChild);
    }
}
```

initTree가 timeCost가 가장 높다. 삽입 횟수를 N, 검색 횟수를 M이라고 하면 시간 복잡도는 O(Nh+MVh)가 된다. V는 검색 시점에 따라 달라지는 함수이나 개략적으로 삽입 횟수와 동일하다 생각하면 최종 시간 복잡도는 O(N(M+1)h)가 된다.