

2 번

리스트는 item 을 차례대로 저장한다. item 은 위치와 순서를 가지고 함수를 이용하여 값을 추가하거나 삭제 할 수 있다. 구현이 복잡하지만 배열처럼 size 가 제한되어 있지 않고 삽입과 삭제를 할 때 효과적이다..


문제:

2. (Programming)

Suppose two linked lists $a = \{a_1, a_2 \dots, a_m\}$ and $b = \{b_1, b_2 \dots, b_n\}$ are sorted in an ascending order. Implement a merge function using a linked list that generates a new linked list c by merging a and b .

Here, elements should be sorted in an ascending order. Show the result using the following example. Also, analyze the time complexity (upper bound O or tight bound θ) of this function.

$$\begin{array}{l} a = \{1, 2, 5, 10, 15, 20, 25\} \\ b = \{3, 7, 8, 15, 18, 30\} \end{array} \Rightarrow c = \{1, 2, 3, 5, 7, 8, 10, 15, 15, 18, 20, 25, 30\}$$

 이화여자대학

해결방법:

집합 a,b 는 둘 다 오름차순으로 정렬되어 있다. 만약 오름차순이 아닌 무작위로 정렬되어 있다면 두 집합을 합치고 선택 정렬을 하겠지만 오름차순으로 정렬되어 있어서 다음과 같은 방법을 사용하였다.

ListNode c 를 만들고 두 집합의 data 를 하나씩 비교 후 더 큰 값을 c 에 넣는다. $C=c \rightarrow \text{link}$ 와 c 에 넣은 값의 집합을 $a=a \rightarrow \text{link}$ or $b=b \rightarrow \text{link}$ 하면서 다음 리스트로 넘겨준다. C 에 넣지 않은 집합은 리스트를 넘기지 않고 다음 값과 비교한다. 비교 후 넣는 일을 계속 반복하다가 집합 a,b 중 하나가 NULL 이면 반복을 멈춘다. 그 후 아직 NULL 이 아닌 집합을 리스트를 넘기며 node c 에 넣어준다.

시간 복잡도 분석:

Main()에서 대입 연산 16 번, insert_node 함수를 13 번 호출하고 있는데 insert_node 함수는 시간 복잡도가 $O(3)$ 이다. ($3 \times 13 = 39$ 번) printList 함수는 3 번 호출되고 그 함수는 대입연산이 n 번 실행된다. ($3 \times n = 3n$) 1 번 호출되는 Merge 함수는 비교연산이 $n+3$ 번, 대입연산이 n 번 일어나기 때문에 시간 복잡도는 $O(2n+3)$ 이다. 분석한 값들을 모두 더해보면 $16+39+3+3+2n+3n=5n+61$ 이고 총 시간 복잡도는 $O(5n+61)=O(n)$ 이다.

시간 복잡도: $O(n)$

코드실행결과:

```
> make -s
> ./main
a = {1, 2, 5, 10, 15, 20, 25, }
b = {3, 7, 8, 15, 18, 30, }
c = {1, 2, 3, 5, 7, 8, 10, 15, 15, 18, 20, 25, 30, }
> █
```

3 번

리스트 ADT 는 리스트를 추상 데이터를 타입으로 정의한 것이다. List ADT 는 연결리스트를 이용하여 구현한다. 각 연산들을 함수로 만들고 필요할 때마다 호출하여 연산을 한다.

문제:

3. (Programming)

When a linked list is defined as below, re-implement all functions in 'List ADT' (p55-p61) as well as add_first, add_last, delete_first, delete_last. Then, run the following example code.

```
int main()
{
    ListType list1;
    init(&list1);
    add_first(&list1, 20);
    add_last(&list1, 30);
    add_first(&list1, 10);
    add_last(&list1, 40);
    add(&list1, 2, 70);
    display(&list1);

    delete(&list1, 2);
    delete_first(&list1);
    delete_last(&list1);
    display(&list1);

    printf("%s\n", is_in_list(&list1, 20) == TRUE ? "TRUE": "FALSE");
    printf("%d\n", get_entry(&list1, 0));
}
```

```
typedef int element;
typedef struct ListNode {
    element data;
    struct ListNode *link;
} ListNode;

typedef struct {
    ListNode *head; // Head pointer
    ListNode *tail; // Tail pointer
    int length; // # of nodes
} ListType;
ListType list1;
```

해결방법:

강의자료에 나와있는 함수들의 구현법을 이해하고 주어진 코드의 맞춰서 재설계해야 한다. 강의

자료에 나와있지 않은 함수 add_first, add_last, delete_first, delete_last 의 구현법은 다음과 같다.

add_first: 전달받은 data 값을 넣기 위한 node p 를 만들고 data 값을 넣는다. 전달받은 list 의 head 값이 NULL 이면 p 의 다음 리스트를 NULL 로 넣고 list 의 head, tail 에 p 를 넣는다. Head 가 null 이 아니면 node p 의 list 값에 head 를 넣어주고 head 에 p 를 넣어준다. Node 의 개수가 늘어났기 때문에 list->lengh 를 +1 해준다.

:add_last: 전달받은 data 값을 넣기 위한 node p 를 만들고 data 값을 넣는다. 이때 마지막 node 를 찾아서 list->tail 에 넣어야 한다. node last 를 만들어주고 head 값을 넣어준다. Node last 의 link 값이 null 이 될 때까지 반복문을 돌린다. 반복을 다 끝냈을 때 node last 값을 tail 에 넣어준다. Tail==NULL 이면 p 를 head 와 tail 에 넣어주고 p 의 link 는 NULL 해준다. . Tail!=NULL 이면 tail 의 link 의 p 를 넣고 p->link 는 NULL 해준다. Tail 에 p 를 넣고 Node 의 개수가 늘어났기 때문에 list->lengh 를 +1 해준다.

delete_first: head==NULL 이면 삭제할 node 가 없기 때문에 error message 를 출력한다. head!=NULL 이면 삭제할 node 를 넣을 node 를 만들어주고 head 값을 넣어준다. List->head 는 그 다음 node 인 list->hrea->link 를 넣어주고 삭제할 노드를 삭제해준다. Node 의 개수가 줄었기 때문에 list->lengh 를 -1 해준다.

delete_last:: 마지막 노드의 이전 노드를 찾기 위해 node last_before 를 만들어준다. 만든 node 에 head 값을 넣어주고 반복문을 돌려서 마지막 노드의 이전 노드를 찾는다 last_before->link-link 값이 null 이 될 때까지 돌리고 반복문이 끝난 후 last_before->link 값을 tail 에 넣어준다.

Tail==NULL 이면 삭제할 node 가 없기 때문에 error message 를 출력한다. Tail 이 null 이 아니면 tail 값을 삭제하고 tail 에 last_befor 을 넣어준다. Tail ->link 는 null 을 해준다. Node 의 개수가 줄었기 때문에 list->lengh 를 -1 해준다.

코드 실행 결과:

```
C:\Users\ssh12\CLionProjects\untitled16\cmake-build-debug\untitled16.exe
( 10 20 70 30 40 )
( 20 30 )|
TRUE
20
```