**Unrolled Linked List 구현 및 실험**

**1. 서론**

Unrolled linked list를 C로 구현하고 기본적인 linked list와의 비교 실험을 진행하였다.

**2. 구현**

**2.1. 목표**

단방향 Unrolled linked list를 C로 구현하고 단방향 linked list와의 비교 실험을 진행한다. (Unrolled) linked list는 리스트의 가장 앞/뒤에 데이터를 삽입/삭제하는 연산 insert/remove를 지원한다. 또한 원하는 위치(pos)의 데이터를 가져오는 연산 get을 지원한다.

**2.1. 설계**

Unrolled linked list는 기존 linked list 노드에 데이터를 하나 저장하는 대신 배열을 이용하여 여러 데이터를 저장하는 자료구조이다. 배열을 이용하기 때문에 데이터 삽입/삭제 정책을 다양하게 선택할 수 있다. 이번 실험에서는 어떤 정책으로 구현했는지 설명한다. 다음 구조체를 이용하여 구현했다.

typedef struct ULNode {

int front;

int back;

int max;

struct ULNode\* next;

Item items[1];

// items [\_ \_ a b d e \_ \_ \_]

// ^ ^ ^

// front back max

} ULNode;

**2.1.1. 특정 위치에 데이터 읽기/쓰기**

마치 배열처럼 인덱스(pos)를 이용하여 리스트에 저장된 데이터를 읽고 쓸 수 있게 하였다. a,b,c,d,e가 저장된 노드 크기 5인 unrolled linked list에서 각 데이터에 할당된 pos는 다음과 같다.

**proot -> [\_ \_ a b c] -> [\_ \_ d e \_]**

**0 1 2 3 4**

pos를 이용하여 리스트에서 원하는 데이터를 읽을 수 있다. pos는 -1 이상의 정수를 허용한다. -1은 리스트의 마지막 위치를 의미한다. 읽거나 삭제하려는 pos에 데이터가 없는 경우 아무 일도 일어나지 않는다.

**2.1.2. insert 정책**

데이터 삽입은 Unrolled linked list의 노드의 중간(⌊노드 크기 / 2⌋)에서부터 일어난다. front와 back 인덱스 변수를 이용하여 삽입/삭제를 구현하였다. 노드의 앞쪽에 데이터가 삽입될 경우 front를 한 칸 앞으로 옮긴다.노드의 뒤쪽에 데이터가 삽입될 경우 back을 한 칸 뒤로 옮긴다. 노드의 배열에 더이상 데이터를 삽입할 수 없으면 새로운 노드를 생성한다.

**2.1.3. insert 예시**

노드의 크기가 5인 경우, 삽입은 2(= ⌊5/2⌋)에서부터 일어난다. 다음은 노드 크기 5인 빈 리스트이다.

**v------max**

**proot -> [\_ \_ \_ \_ \_]**

**front----^ ^----back**

빈 노드에서 back은 2(= ⌊5/2⌋), front는 그보다 1 작은 1에서 시작한다. max는 노드의 크기인 5이다.

여기서 pos=-1(뒤쪽)에 데이터 a를 저장하면 다음과 같다.

**proot -> [\_ \_ a \_ \_]**

**front----^ ^----back**

여기서 pos=-1(뒤쪽)에 데이터 b,c를 저장하면 다음과 같다.

**proot -> [\_ \_ a b \_]**

**front----^ ^----back**

**v----max**

**proot -> [\_ \_ a b c]**

**front----^ ^----back**

back과 max가 같아지면 다음과 같이 새로운 노드를 생성한다.

**proot -> [\_ \_ a b c] -> [\_ \_ d \_ \_]**

**f^ b^ f^ ^b**

**proot -> [\_ \_ a b c] -> [\_ \_ d e \_]**

**f^ b^ f^ ^b**

pos=3에 데이터 X를 삽입하면 다음과 같다.

**proot -> [\_ \_ a b c] -> [\_ X d e \_]**

**f^ b^ f^ ^b**

pos=3에 Y,Z를 삽입하면 다음과 같이 새로운 노드를 생성한다.

**proot -> [\_ \_ a b c] -> [Y X d e \_]**

**f^ b^ f^ ^b**

**proot -> [\_ \_ a b c] -> [\_ \_ Z \_ \_] -> [Y X d e \_]**

**f^ b^ f^ b^ f^ ^b**

**2.1.4. remove 정책**

노드의 앞쪽 데이터가 삭제될 경우 front를 한칸 뒤로 옮긴다. 노드의 뒤쪽 데이터가 삭제될 경우 back을 한 칸 앞으로 옮긴다. 노드의 배열에 데이터가 모두 삭제된 경우 노드를 제거한다. 리스트에 노드가 하나만 존재하는 경우에는 제거하지 않는다. 빈 리스트는 데이터가 없는 노드 하나로 구성된 리스트로 정의하였다.

**2.1.5. remove 예시**

**proot -> [\_ \_ a b c] -> [\_ \_ Z \_ \_] -> [Y X d e \_]**

**f^ b^ f^ b^ f^ ^b**

위 리스트에서 pos=0의 데이터를 제거하면 다음과 같다.

**proot -> [\_ \_ \_ b c] -> [\_ \_ Z \_ \_] -> [Y X d e \_]**

**f^ b^ f^ b^ f^ ^b**

만일 노드의 데이터를 모두 제거하면 노드를 삭제하고 삭제한 노드의 전 후 노드를 연결한다. 위 리스트에서 Z를 제거(pos=2)하면 다음과 같다.

**proot -> [\_ \_ \_ b c] -> [\_ \_ \_ \_ \_] -> [Y X d e \_]**

**f^ b^ f^b^ f^ ^b**

**proot -> [\_ \_ \_ b c] -> [Y X d e \_]**

**f^ b^ f^ ^b**

예외적으로 리스트에 노드가 하나만 존재할 경우 노드를 삭제하지 않는다. 이는 빈 리스트를 빈 노드 하나로 정의했기 때문이다.

**2.1.6. linked list 구현**

이번 실험에서 linked list는 노드 크기가 1인 특수한 unrolled linked list로 간주하여 구현하였다.

**3. 구현 상세**

**3.1. ulist.{c,h}**

**#define TRUE (1==1)**

**#define FALSE (!TRUE)**

**#define SUCCESS (0)**

**#define FAILURE (1)**

에러 처리와 예외 값 처리를 위한 상수.

**typedef int Item;**

원하는 데이터를 Item으로 설정하여 사용할 수있다. 이번 실험에서 Item은 int로 설정하였다.

**typedef struct ULNode {**

**int front;**

**int back;**

**int max;**

**struct ULNode\* next;**

**Item items[1];**

**// items [\_ \_ a b d e \_ \_ \_]**

**// ^ ^ ^**

**// front back max**

**} ULNode;**

Unrolled linked list의 노드를 정의하는 구조체다. items는 데이터를 저장하는 배열이다. items는 노드 생성 시에 다양한 크기로 설정될 수 있다. 이를 구현하기 위해 C에서 흔히 사용되는 이디엄인 trailing array hack을 적용하였다.

front는 items의 첫번째 데이터 바로 앞을 가리키는 인덱스 변수다. back은 items의 마지막 데이터 바로 뒤를 가리키는 인덱스 변수다. max는 노드가 담을 수 있는 최데 대이터 수를 의미하며, back 인덱스 변수의 최대값이라고 해석할 수도 있다.

next는 리스트에서 다음 노드를 가리킨다. 다음 노드가 없는 경우 NULL로 설정된다.

**int init\_list(ULNode\*\* lst, int node\_size);**

빈 리스트를 생성한다. 빈 리스트는 items에 데이터가 없는 노드 하나로 이루어진 리스트이다. node\_size가 max이며, back은 ⌊max / 2⌋이다. front는 back - 1 이다. 상술한 노드를 동적할당한다. 생성에 성공한 경우 SUCCESS를 반환한다.

**int insert(ULNode\*\* lst, int pos, Item item);**

pos에 item을 삽입한다. pos = 0일 경우 리스트의 가장 앞에 삽입한다. 삽입에 성공한 경우 SUCCESS를 반환한다. pos = -1일 경우 리스트의 가장 뒤에 삽입한다. pos가 너무 크거나 음수일 경우 리스트는 변경되지 않으며 FAILURE를 반환한다. 기본적으로 노드의 front, back을 움직여 노드의 items에 삽입하나, 더 이상 삽입이 불가능할 경우 새로운 노드를 동적할당하여 데이터를 삽입한다.

**int remove(ULNode\*\* lst, int pos, Item\* removed);**

pos의 데이터를 삭제하고 removed에 삭제되는 데이터를 할당한다. 삭제에 성공한 경우 SUCCESS를 반환한다. pos = 0일 경우 리스트의 가장 앞에 있는 데이터를 삭제한다. pos = -1일 경우 리스트의 가장 뒤에 있는 데이터를 삭제한다. pos가 너무 크거나 -1보다 작을 경우 리스트는 변경되지 않으며 FAILURE를 반환한다. 기본적으로 노드의 front, back을 움직여 items에서 데이터를 삭제하며, 만일 노드에서 모든 데이터가 삭제된 경우 노드의 메모리 할당을 해제한다. 예외적으로 리스트에 노드가 하나가 남았을 때는 노드에서 데이터가 모두 삭제되어도 노드를 유지한다.

**int is\_empty(ULNode\* lst);**

리스트가 비어있다면 TRUE, 그렇지 않으면 FALSE를 반환한다. 빈 리스트는 데이터 없는 노드 하나로 이루어진 리스트이다.

**int get(ULNode\* lst, int pos, Item\* accessed);**

pos에 있는 데이터를 accessed에 할당한다. pos = 0일 경우 리스트의 가장 앞에 있는 데이터를 읽다. pos = -1일 경우 리스트의 가장 뒤에 있는 데이터를 읽는다. pos가 너무 크거나 -1보다 작을 경우 데이터를 읽는데 실패한다. 데이터를 가져오는데 성공한 경우 SUCCESS를, 실패한 경우 FAILURE를 반환한다. 리스트는 변경되지 않는다.

**int decode\_pos(ULNode\* lst, int pos,**

**ULNode\*\* ret\_prev, ULNode\*\* ret\_node, int\* ret\_idx);**

입력된 lst와 pos를 해석하여 ret\_prev, ret\_node, ret\_idx를 할당한다. ret\_node는 pos에 해당하는 데이터가 속하는 노드, ret\_prev는 ret\_node 바로 전 노드이다. 없으면 NULL로 할당한다. ret\_idx는 ret\_node에서 pos에 해당하는 데이터의 items에서의 인덱스다. 즉 (\*ret\_node)->items[\*ret\_idx]로 lst의 pos에 존재하는 데이터를 가져올 수 있다. 만일 pos가 리스트에 존재하지 않는 데이터를 가리키면, FAILURE를 반환하며 pos를 성공적으로 해석했다면 SUCCESS를 반환한다.

**int print\_node(ULNode\* node, int all);**

노드의 데이터와 값을 다음과 같이 출력한다.

노드 주소: [front값] [back값] [max값] [items의 데이터] 다음 노드 주소

0x55b18bfcb3b0: front[-1] back[3] max[7] arr[ 5 4 3 \_ \_ \_ \_ ] next:0x55b18bfcb360

**int print\_list(ULNode\* lst, int all);**

print\_node를 이용하여 lst의 모든 노드를 출력한다.

**int get\_tail(ULNode\* lst, ULNode\* tail);**

마지막 노드를 tail에 할당하고 리스트에 존재하는 노드의 개수를 반환한다.

**static int new\_node(ULNode\*\* pnode, int max, ULNode\* next);**

max와 next를 이용하여 새로운 노드를 동적할당하여 pnode에 할당한다. 실패할 경우 FAILURE, 성공할 경우 SUCCESS를 반환한다.

**static int add\_back(ULNode\* node, Item item);**

노드의 back을 옮겨 item을 삽입한다. 성공하면 SUCCESS, 실패하면 FAILURE를 반환한다.

**static int add\_front(ULNode\* node, Item item);**

노드의 front를 옮겨 item을 삽입한다. 성공하면 SUCCESS, 실패하면 FAILURE를 반환한다. 노드의 max가 1인 경우, 함수 내부에서 예외적으로 add\_back을 실행한다. 삽입/삭제 정책에 따르면 max=1인 경우 모든 노드의 front가 항상 -1이기 때문이다.

**static int num\_items(ULNode\* node);**

현재 노드에 저장된 item의 수를 반환한다.

**static int full\_front(ULNode\* node);**

현재 노드에서 add\_front가 불가능하면 TRUE, 가능하면 FALSE를 반환한다.

#endif //\_ULIST\_H\_