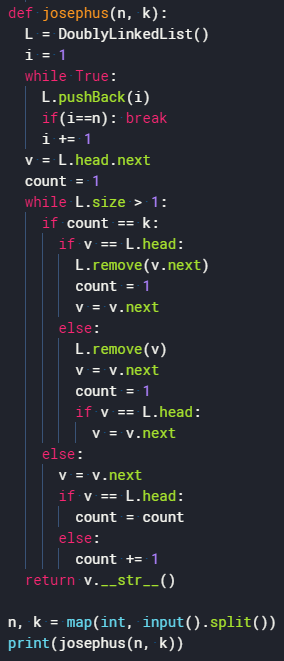
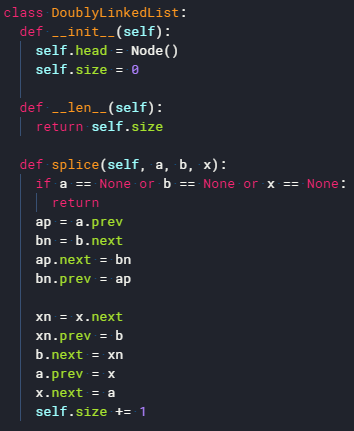
**자료구조 및 실습 레포트**

201903868 황가은

1. Josephus



Node 클래스 코드는 넣지 않았습니다.

양방향연결리스트 클래스에서는 기본적으로 splice 함수를 이용해서 여러 삽입 함수들을 만들고 하나의 삭제 함수를 만들었습니다. Josephus 문제에선 Josephus 함수만 보도록 하겠습니다.

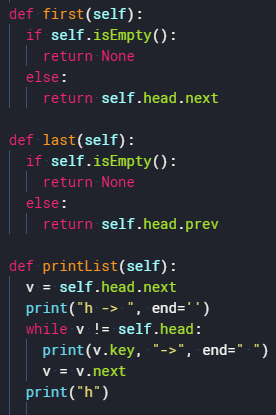
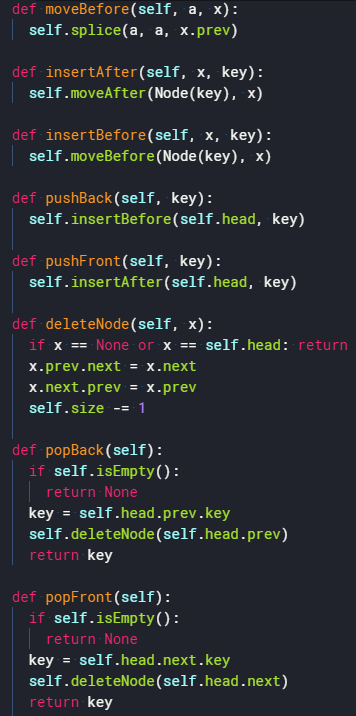
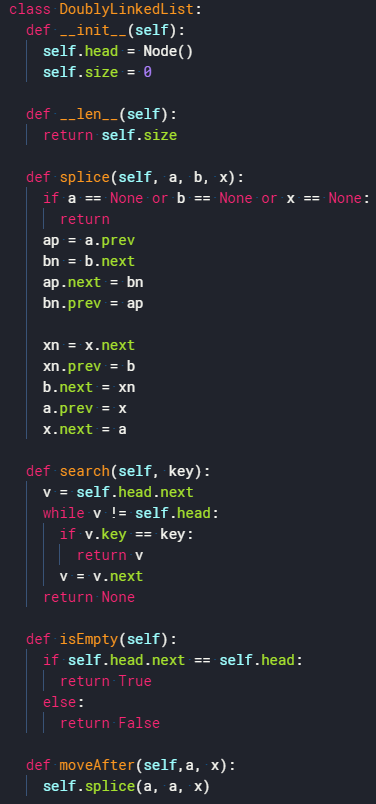
먼저 L이라는 양방향연결리스트를 선언해줍니다. 1부터 n까지의 수를 순서대로 넣기 위해서 반복문으로 pushback(i) 해줍니다. I는 반복문이 작동될때마다 1씩 증가하고 I 값과 k값이 같으면 반복문을 종료합니다.

V는 L의 dummy node의 바로 뒤 값을 가리킵니다. 이때 count 라는 변수를 1로 설정했습니다. Count는 몇 번 이동했는지 기록하기 위한 변수입니다. 그 다음에 반복문의 조건을 L의 크기가 1보다 클 동안 작동하게 만듭니다. 먼저 count 가 k와 같다는 조건문을 작성합니다. 조건이 성립할 때 v가 dummy node를 가리킨다면 v.next 를 삭제하고 count를 1로 초기화해주고 v는 그 다음 값으로 이동합니다. V가 dummy node값이 아닌 일반 값을 가리킨다면 v를 삭제하고 v는 다음 값으로 옮겨주며 count를 1로 초기화합니다. 이때, 옮긴 v의 값이 dummy node라면 그 다음 값으로 다시 옮겨줍니다.

Count와 k가 같지 않다면 v를 다음 값으로 옮겨줍니다. 옮기는 중 v가 dummy node를 지나가면 count 값은 변하지 않도록 합니다. 그 외엔 count가 1씩 증가합니다. 이렇게 해서 L의 크기가 1과 같거나 작게된다면 반복문을 빠져나와 v값을 반환합니다.

Josephus 함수의 시간복잡도는 O(n) 이라고 볼 수 있습니다. 빈 양방향연결리스트에 값을 1부터 n까지 넣는 것이므로 O(n) 시간이 걸립니다. 값을 다 넣은 양방향연결리스트의 크기는 n이 됩니다. 그러므로 n이 1보다 같거나 작을때까지 반복문이 작동합니다. 반복문 안에 있는 조건문들은 다 O(1) 시간에 작동합니다. Remove 함수의 경우 search를 사용하지 않고 v가 직접 이동하며 다니기 때문에 O(1) 시간입니다. 그러므로 양방향연결리스트를 돌면서 값을 삭제하는 것도 O(n) 시간이라고 볼 수 있습니다.

2. DoublyLinkedList



여기에서 양방향연결리스트에 사용된 함수들의 분석과 시간복잡도에 대해서 알아보겠습니다. 먼저 사용된 함수는 삽입, 이동, 검색, 삭제, 출력, 기타 연산들입니다. 가장 기본이 되는 splice 함수는 양방향연결리스트에서 a부터 b까지 사이의 값들을 빼내와서 x의 뒤에 붙입니다. 노드의 링크들만 수정하면 되기 때문에 시간복잡도는 O(1)입니다.

Search 함수는 양방향연결리스트를 돌면서 key 값과 같은 값을 가지는 node를 찾아 반환합니다. V는 dummy node의 다음 값을 가리킵니다. V가 dummy node가 아닌 동안 검사합니다. 조건에 부합하는 node를 찾기 위해서는 하나 하나씩 맞는지 검사를 해야 하기 때문에 O(n)시간이 걸립니다.

isEmpty 함수는 양방향연결리스트가 비었는지 즉, dummy node 밖에 없는지 확인하는 함수입니다. 그걸 확인하기 위해서 head node의 다음 값이 head node라면 빈 리스트이므로 True를 반환하고 그게 아니라면 False를 반환합니다. O(1) 시간이 걸립니다.

moveAfter 함수는 a를 x의 뒤로 옮겨주는 함수입니다. Splice 함수를 이용했기에 O(1)시간이 걸립니다. 그 외의 moveBefore, 새로운 key값을 가지는 node를 만들어 x의 앞과 뒤로 삽입하는 insertAfter, insertBefore, 리스트의 맨 뒤와 앞에 node를 삽입해주는 pushBack과 PushFront 또한 splice 함수를 이용하기에 O(1) 시간이 걸립니다.

deleteNode 함수는 x 노드를 삭제해주는 함수입니다. 밑에서 deleteNode를 입력하면 search 함수를 실행시켜줍니다. Search(x)를 한 결과가 None이라면 삭제하지 못하고, 그게 아니라면 x 노드의 앞뒤 링크를 수정하여 삭제합니다. 그리고 해당 양방향연결리스트의 크기를 1 줄입니다. 시간복잡도는 O(1)입니다.

popBack, popFront 함수는 양방향연결리스트의 dummy node를 제외한 맨 앞값과 뒷값을 제거하고 해당 key 값을 반환하는 함수입니다. 해당 리스트가 비었는지 확인하기 위해서 isEmpty 함수를 이용합니다. isEmpty 함수의 시간복잡도는 O(1) 이고 popBack, popFront의 시간복잡도도 O(1)입니다.

First, last 함수는 해당 양방향연결리스트의 dummy node를 제외한 맨 앞값과 뒷값을 반환합니다. popBack, popFront와 비슷하지만 해당 노드를 삭제하지는 않습니다. 기본 연산 차이만 있기에 이 함수들도 시간복잡도가 O(1)입니다.

printList 함수는 해당 양방향연결리스트가 가지고 있는 값들을 순서대로 출력해주는 함수입니다. V는 dummy node의 바로 다음 값을 가리킵니다. 그리고 v가 dummy node가 아닐 동안 v.key를 출력합니다. 시간복잡도는 O(1)입니다.

이렇게 DoublyLinkedList에 쓰인 함수들을 분석하고 시간복잡도를 알아봤습니다. Search 함수만 O(n) 시간이 걸리며 나머지 함수들은 O(1) 시간이 걸린다는 것을 알 수 있습니다. 여기에선 쓰이지 않았지만 join, split 함수도 splice 함수를 이용하기에 O(1)시간이 걸립니다.

양방향연결리스트와 배열의 수행시간을 비교해보면 결과적으로 양방향연결리스트가 빠릅니다. 배열은 양방향연결리스트와 달리 첫번째 인덱스부터 시작해야합니다. 그렇기에 특정 원소를 삭제하거나 특정 위치에 삽입하려면 search를 해야 합니다. 그러나 배열은 k번째 원소 값을 찾을 땐 O(1) 시간이 걸립니다. 배열은 저장된 값의 주소를 인덱스가 가리키기 때문에 바로 알 수 있는 장점이 있지만 그 외의 연산 수행 시간은 양방향연결리스트 보다 느립니다. 반면에 양방향연결리스트는 k번째 원소 값을 알기위해서 앞에서든 뒤에서든 링크를 타고 돌아다녀야만 알 수 있습니다. 그렇기에 O(k)시간이 걸립니다.

상황에 따라서 배열이 더 좋을수도 있으며 양방향연결리스트가 더 좋을수도 있습니다. 또한 각자의 장단점 또한 있기 때문에 양방향연결리스트가 연산 수행 속도가 빠르다고 하여 무조건적으로 사용하기 보단 상황에 올맞은 방법을 사용하는 게 좋다고 생각했습니다. 또한 언어 자체 내에서 제공하는 함수들도 좋지만 직접 class나 함수를 만들어 사용하는 게 더 효율적일 수도 있다는 생각이 들었습니다.