

Linked List

Jin Hyun Kim
Fall, 2019

In this Topic

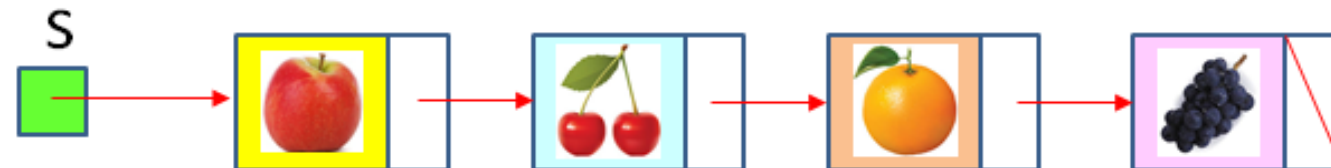
- List
- Single Linked List
- Doubled Linked List
- Circular Linked List

List

- 일반적인 리스트(List)는 일련의 동일한 타입의 항목(item)들
 - 실생활의 예: 학생 명단, 시험 성적, 서점의 신간 서적, 상점의 판매 품목, 실시간 급상승 검색어, 버킷 리스트 등
- 일반적인 리스트의 구현:
 - 1차원 파이썬 리스트(list)
 - 단순연결리스트
 - 이중연결리스트
 - 원형연결리스트

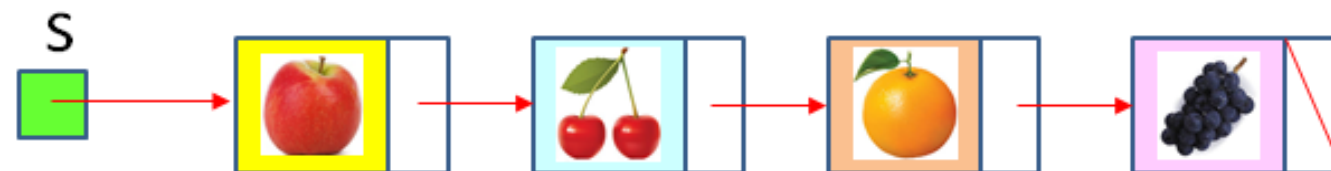
단순연결리스트

- 단순연결리스트(Singly Linked List)는 동적 메모리 할당을 이용해 리스트를 구현하는 가장 간단한 형태의 자료구조
- 동적 메모리 할당을 받아 노드(node)를 저장하고, 노드는 레퍼런스를 이용하여 다음 노드를 가리키도록 만들어 노드들을 한 줄로 연결 시킴



단순연결리스트

- 연결리스트에서는 삽입이나 삭제 시 항목들의 이동이 필요 없음
- 연결리스트에서는 항목을 탐색하려면 항상 첫 노드부터 원하는 노드를 찾을 때까지 차례로 방문하는 순차탐색(Sequential Search)해야 함



단순연결리스트

- size()
- is_empty()
- insert_front(item)
- insert_after(item, p)
- delete_front()
- delete_after(p)
- search(target)
- print_list()

SList Usage

SList

SList

SList

SList

SList

수행복잡도분석

- `search()`는 탐색을 위해 연결리스트의 노드들을 첫 노드부터 순차적으로 방문해야 하므로 $O(N)$ 시간 소요
- 삽입이나 삭제 연산은 각각 $O(1)$ 개의 레퍼런스만을 갱신하므로 $O(1)$ 시간 소요
- 단, `insert_after()`나 `delete_after()`의 경우에 특정 노드 `p`의 레퍼런스가 주어지지 않으면 `head`로부터 `p`를 찾기 위해 `search()`를 수행해야 하므로 $O(N)$ 시간 소요

이중연결리스트

- 이중연결리스트(Doubly Linked List)는 각 노드가 두 개의 레퍼런스를 가지고 각각 이전 노드와 다음 노드를 가리키는 연결리스트



- 단순연결리스트는 삽입이나 삭제할 때 반드시 이전 노드를 가리키는 레퍼런스를 추가로 알아내야 하고, 역방향으로 노드들을 탐색할 수 없음
- 이중연결리스트는 단순연결리스트의 이러한 단점을 보완하나, 각 노드마다 추가로 한 개의 레퍼런스를 추가로 저장해야 한다는 단점을 가짐

Exercise

- $A = [1, 2, 4, 5, 7, 9, 19, 100]$ 같은 리스트를 입력으로 리스트를 구현하라.
- 특정 요소가 리스트에 있는지 확인하는 function을 작성하라.

```
def isin(self, n):
```

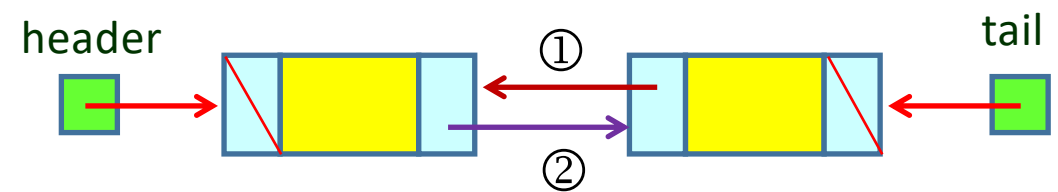
```
....
```

DList

- `is_empty()`
- `insert_before(p, item)`
- `insert_after(p, item)`
- `delete(x)`
- `print_list()`

DList Usage

DList



[그림 2-8] DList 객체 생성

DList

DList: insert_before()

DList: insert_after

DList: delete()

- 1 32 f = x.prev
- 2 33 r = x.next
- 3 34 f.next = r
- 4 35 r.prev = f

DList

```
39 def print_list(self):
40     if self.is_empty():
41         print('리스트 비어있음')
42     else:
43         p = self.head.next
44         while p != self.tail:
45             if p.next != self.tail:
46                 print(p.item, ' <=> ', end='')
47             else:
48                 print(p.item)
49             p = p.next
50
51 class EmptyError(Exception):
52     pass
```

노드들을 차례로 방문하기 위해

underflow 시 에러 처리

수행복잡도분석

- `search()`는 탐색을 위해 연결리스트의 노드들을 첫 노드부터 순차적으로 방문해야 하므로 $O(N)$ 시간 소요
- 삽입이나 삭제 연산은 각각 $O(1)$ 개의 레퍼런스만을 갱신하므로 $O(1)$ 시간 소요
- 단, `insert_after()`나 `delete_after()`의 경우에 특정 노드 `p`의 레퍼런스가 주어지지 않으면 `head`로부터 `p`를 찾기 위해 `search()`를 수행해야 하므로 $O(N)$ 시간 소요

원형연결리스트

- 원형연결리스트(Circular Linked List)는 마지막 노드가 첫 노드와 연결된 단순연결리스트
- 원형연결리스트에서는 마지막 노드의 레퍼런스가 저장된 last가 단순연결리스트의 head와 같은 역할

원형연결리스트

- 마지막 노드와 첫 노드를 $O(1)$ 시간에 방문할 수 있는 장점
- 리스트가 empty가 아니면 어떤 노드도 None 레퍼런스를 가지고 있지 않으므로 프로그램에서 None 조건을 검사하지 않아도 되는 장점
- 원형연결리스트에서는 반대 방향으로 노드들을 방문하기 쉽지 않으며, 무한 루프가 발생할 수 있음에 유의할 필요


CList

- insert(item)
- first()
- delet()
- print_list()

CList Usage

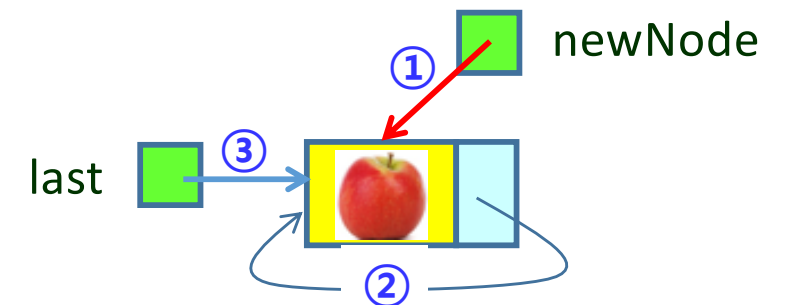
CList

CList:: insert(item)



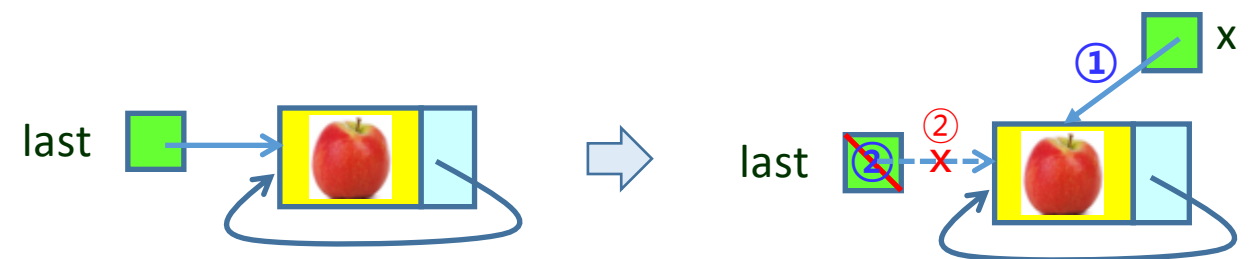
```
15 n = self.Node(item, None)
16 if self.is_empty():
17     n.next = n
18     self.last = n
19 else:
20     n.next = self.last.next
21     self.last.next = n
```

last 



CList:: delete()

```
33 x = self.last.next  
34 if self.size == 1:  
35     self.last = None  
36 else:  
37     self.last.next = x.next
```



응용

- 여러 사람이 차례로 돌아가며 하는 게임을 구현하는데 적합한 자료 구조
- 많은 사용자들이 동시에 사용하는 컴퓨터에서 CPU 시간을 분할하여 작업들에 할당하는 운영체제에 사용
- 이항힙(Binomial Heap)이나 피보나치힙(Fibonacci Heap)과 같은 우선순위큐를 구현하는 데에도 원형연결리스트가 부분적으로 사용

수행복잡도분석

- 원형연결리스트에서 삽입이나 삭제 연산 각각 상수 개의 레퍼런스를 갱신하므로 $O(1)$ 시간에 수행
- 탐색 연산: last로부터 노드들을 순차적으로 탐색해야 하므로 $O(N)$ 소요

List

Next Topic

- Stack and Queue