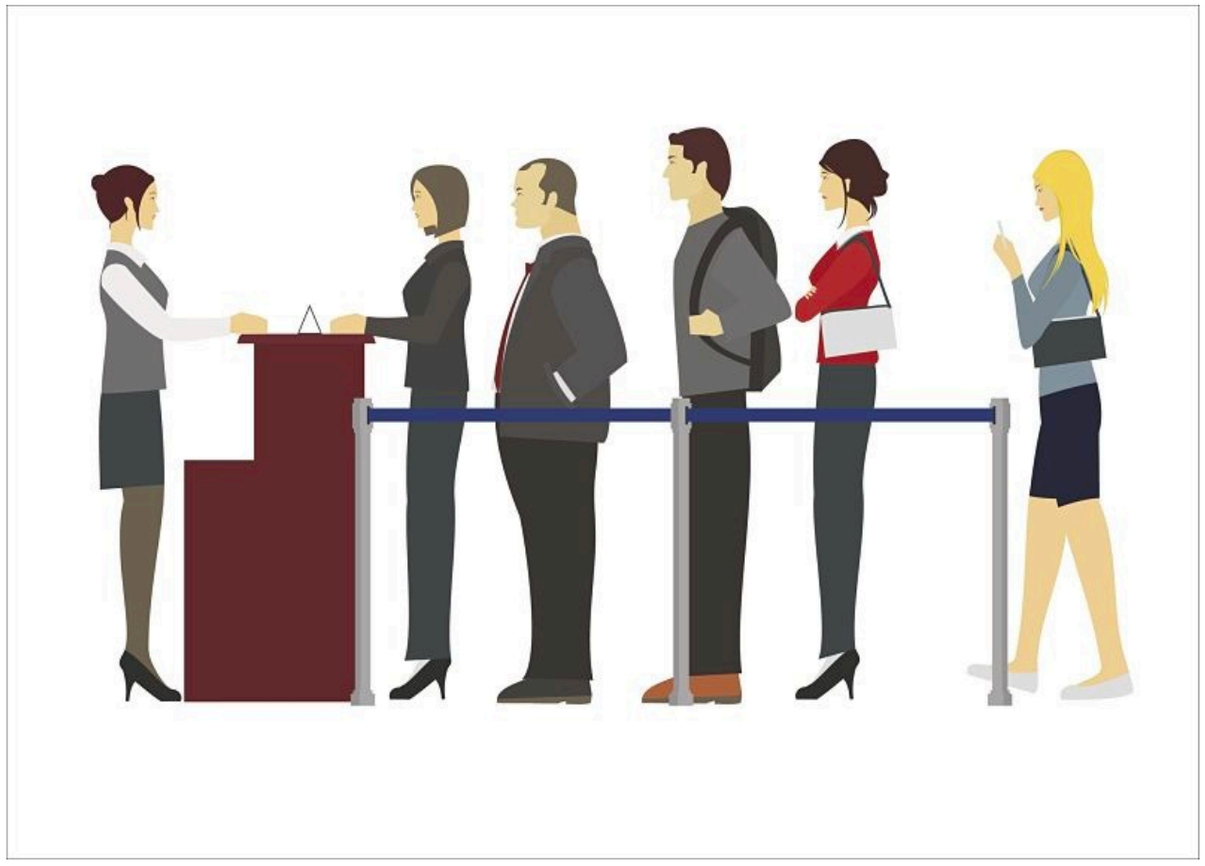
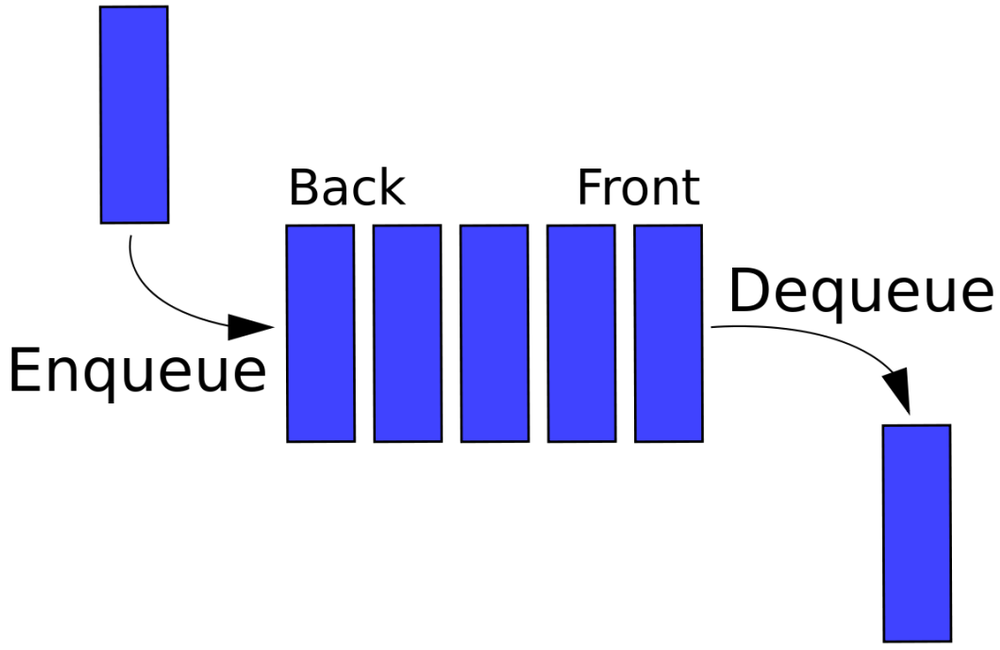




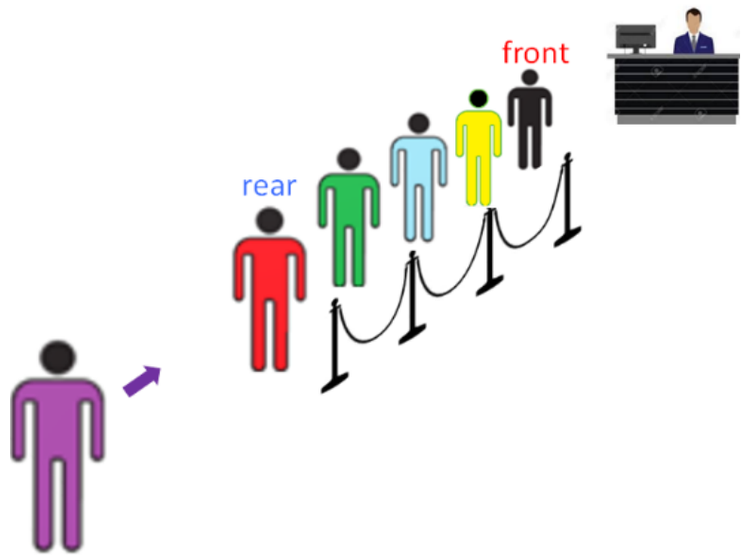
Queue

Jin Hyun Kim

Fall, 2019

큐

- 큐(Queue): 삽입과 삭제가 양 끝에서 각각 수행되는 자료구조



- 일상생활의 관공서, 은행, 우체국, 병원 등에서 번호표를 이용한 줄서기가 대표적인 큐

- 선입 선출(First-In First-Out, FIFO) 원칙 하에 item의 삽입과 삭제 수행

파이썬 리스트로 구현

```
01 def add(item): # 삽입 연산
02     q.append(item)
03
04 def remove(): # 삭제 연산
05     if len(q) != 0:
06         item = q.pop(0)
07         return item
08
09 def print_q(): # 큐 출력
10     print('front ->  ', end='')
11     for i in range(len(q)):
12         print('{!s:<8}'.format(q[i]), end='')
13     print('  <- rear')
```

맨 뒤에 새 항목 삽입

맨 앞의 항목 삭제

맨 앞부터 항목들을 차례로 출력

파이썬 큐 실험

```
14 q = []
15 add('apple')
16 add('orange')
17 add('cherry')
18 add('pear')
19 print('사과, 오렌지, 체리, 배 삽입 후: \t', end='')
20 print_q()
21 remove()
22 print('remove한 후:\t\t', end='')
23 print_q()
24 remove()
25 print('remove한 후:\t\t', end='')
26 print_q()
27 add('grape')
28 print('포도 삽입 후:\t\t', end='')
29 print_q()
```

리스트 선언

일련의 큐 연산과 출력

단순연결리스로 큐 구현

```
01 class Node:
02     def __init__(self, item, n):
03         self.item = item
04         self.next = n
05 def add(item): # 삽입 연산
06     global size
07     global front
08     global rear
09     new_node = Node(item, None)
10     if size == 0:
11         front = new_node
12     else:
13         rear.next = new_node
14     rear = new_node
15     size += 1
```

노드 생성자
항목과 다음 노드 레퍼런스

전역 변수

새 노드 객체를 생성

연결리스트의 맨 뒤에 삽입

단순연결리스로 큐 구현

```
16 def remove(): # 삭제 연산
17     global size
18     global front
19     global rear
20     if size != 0:
21         fitem = front.item
22         front = front.next
23         size -= 1
24         if size == 0:
25             rear = None
26     return fitem
```

전역 변수

연결리스트에서 front가 참조하던 노드 분리시킴

제거된 맨 앞의 항목 리턴

```
27 def print_q(): # 큐 출력
28     p = front
29     print('front: ', end='')
30     while p:
31         if p.next != None:
32             print(p.item, '->    ', end='')
33         else:
34             print(p.item, end = '')
35         p = p.next
36     print('   : rear')
37 front = None
38 rear = None
39 size = 0
40 :
54
```

단순연결리스트(스택)의 항목을 차례로 출력

초기화

[프로그램 3-3]의 line 15~29와 동일

큐 응용

- CPU의 태스크 스케줄링(Task Scheduling)
- 네트워크 프린터
- 실시간(Real-time) 시스템의 인터럽트(Interrupt) 처리
- 다양한 이벤트 구동 방식(Event-driven) 컴퓨터 시뮬레이션
- 콜 센터의 전화 서비스 처리 등
- 이진트리의 레벨순회(Level-order Traversal)
- 그래프에서 너비우선탐색(Breath-First Search) 등

수행복잡도

- 리스트로 구현한 큐의 add와 remove 연산은 각각 O(1) 시간이 소요

- 하지만 리스트 크기를 확대 또는 축소시키는 경우에 큐의 모든 항목들을 새 리스트로 복사해야 하므로 O(N) 시간이 소요

- 단순연결리스트로 구현한 큐의 add와 remove 연산은 각각 O(1) 시간

- 삽입 또는 삭제 연산이 rear 와 front로 인해 연결리스트의 다른 노드들을 일일이 방문할 필요 없이 각 연산이 수행되기 때문

데크 (DeQue)

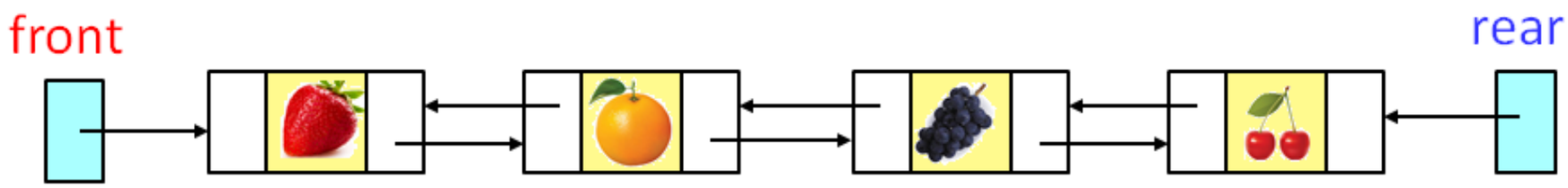
- 데크(Double-ended Queue, Deque): 양쪽 끝에서 삽입과 삭제를 허용하는 자료구조

- 데크는 스택과 큐 자료구조를 혼합한 자료구조

- 따라서 데크는 스택과 큐를 동시에 구현하는데 사용

데크의 구현

- 이중연결리스트로 구현하는 것이 가장 이상적
  - 단순연결리스트는 rear가 가리키는 노드의 이전 노드의 레퍼런스를 알아야 삭제가 가능하기 때문



파이선의 데크

- Collections 패키지에 정의되어 있음

```
01 from collections import deque
02 dq = deque('data')
03 for elem in dq:
04     print(elem.upper(), end='')
05 print()
06 dq.append('r')
07 dq.appendleft('k')
08 print(dq)
09 dq.pop()
10 dq.popleft()
11 print(dq[-1])
12 print('x' in dq)
13 dq.extend('structure')
14 dq.extendleft(reversed('python'))
15 print(dq)
```

새 데크 객체를 생성

맨 뒤와 맨 앞에 항목 삽입

맨 뒤와 맨 앞의 항목 삭제

맨 뒤의 항목 출력

맨 뒤와 맨 앞에 여러 항목 삽입

수행복잡도

- 데크를 배열이나 이중연결리스트로 구현한 경우, 스택과 큐의 수행 시간과 동일
- 양 끝에서 삽입과 삭제가 가능하므로 프로그램이 다소 복잡
- 이중연결리스트로 구현한 경우는 더 복잡함

데크의 응용

- 스크롤(Scroll)
- 문서 편집기 등의 undo 연산
- 웹 브라우저의 방문 기록 등
    - 웹 브라우저 방문 기록의 경우, 최근 방문한 웹 페이지 주소는 앞에 삽입하고, 일정 수의 새 주소들이 앞쪽에서 삽입되면 뒤에서 삭제가 수행

요약

- 큐는 삽입과 삭제가 양 끝에서 각각 수행되는 선입선출(FIFO) 자료구조

- 큐는 CPU의 태스크 스케줄링, 네트워크 프린터, 실시간 시스템의 인터럽트 처리, 다양한 이벤트 구동 방식 컴퓨터 시뮬레이션, 콜 센터의 전화 서비스 처리 등에 사용되며, 이진트리의 레벨순회와 그래프의 너비우선 탐색에 사용

- 데크는 양쪽 끝에서 삽입과 삭제를 허용하는 자료구조로서 스택과 큐 자료구조를 혼합한 자료구조

- 데크는 스크롤, 문서 편집기의 undo 연산, 웹 브라우저의 방문 기록 등에 사용

스택, 큐, 데크의 수행시간 비교

자료구조	구현	삽입	삭제	비고
스택 큐 데크	파이썬 리스트	O(1)	O(1)	*타 언어의 배열
	연결리스트†	O(1)	O(1)	†데크는 이중연결리스트로 구현