[0902] Queue live
목차
1. 큐
큐의 특성
큐의 구조 및 기본연산
큐의 주요 연산
큐의 연산 과정
큐의 구현
<연습문제1>
선형 큐 이용시의 문제점
원형 큐의 구조
원형 큐의 연산 과정
원형 큐의 구현
2. 우선순위 큐 (Priority Queue)
배열을 이용한 우선순위 큐
3. 큐의 활용: 버퍼 (Buffer)
버퍼
키보드 버퍼
예제) 마이쮸
<연습문제2>
4. BFS(Breadth First Search) 🌟
BFS 알고리즘
BFS 예제
<연습문제3>

[0902] Queue live

목차

- 큐
- 우선순위 큐
- ☆**BFS**☆
- 큐의 활용 : 버퍼
- 최단경로
- 실습 1, 2

1. 큐

큐의 특성

- 스택과 마찬가지로 삽입과 삭제의 위치가 제한적인 자료구조
  - 큐의 뒤에서는 삽입만 하고, 큐의 앞에서는 삭제만 이루어지는 구조
- 선입선출구조(FIFO: first in first out)
  - 큐에 삽입한 순서대로 원소가 저장되어, 가장 먼저 삽입(first in)된 원소는 가장 먼저 삭제(first out) 된다

큐의 구조 및 기본연산

- 큐의 선입선출 구조
    - 머리(front): 저장된 원소 중 첫번째 원소
    - 꼬리(rear): 저장된 원소 중 마지막 원소
- 큐의 기본연산
    - 삽입: enQueue
    - 삭제:deQueue

큐의 주요 연산

- 큐의 사용을 위해 필요한 주요 연산은 다음과 같음

연산	기능
enQueue(item)	큐의 뒤쪽(rear 다음)에 원소를 삽입하는 연산
deQueue()	큐의 앞쪽(front)에서 원소를 삭제하고 반환하는 연산
createQueue()	공백 상태의 큐를 생성하는 연산
isEmpty()	큐가 공백상태인지를 확인하는 연산
isFull()	큐가 포화상태인지를 확인하는 연산
Qpeek()	큐의 앞쪽(front)에서 원소를 삭제 없이 반환하는 연산

큐의 연산 과정

ppt 참고 267p

큐의 구현

ppt 참고

- 선형 큐
    - 1차원 배열을 이용한 큐
        - 큐의 크기 = 배열의 크기
        - front: 저장된 첫번째 원소의 인덱스
        - rear: 저장된 마지막 원소의 인덱스
    - 상태 표현
        - 초기 상태: front = rear = -1
        - 공백 상태: front = rear
        - 포화 상태: rear = n-1 (n: 배열의 크기, n-1: 배열의 마지막 인덱스)
- 초기 공백 큐 생성
    - 크기 n인 1차원 배열 생성
    - front와 rear를 -1로 초기화

<연습문제1>

- 큐를 구현하여 다음 동작을 확인해 봅시다
  - 세 개의 데이터 1, 2, 3을 차례로 큐에 삽입하고
  - 큐에서 세 개의 데이터를 차례로 꺼내서 출력한다
    - 1, 2, 3이 출력되야 함

```
# Queue1 (C)
# front, rear 이용
Q = [0] * 100
front, rear = -1, -1  # 크기가 고정됨

def enQueue(item):
    global rear  # 함수 내부에서 전역변수를 수정하려면 global이 필요
    if rear == len(Q) -1 :
        print("Queue Full")
    else:
        rear = rear + 1  # rear += 1
        Q[rear] = item

def deQueue():
    global front
    if front == rear:
        print("Queue Empty")
    else:
        front += 1
        return Q[front]

def Qpeek():
    if front == rear:
        print("Queue Empty")
    else:
        return Q[front + 1]

enQueue(1)
enQueue(2)
enQueue(3)
print(Qpeek())
print(deQueue())  # 삭제되는 건 아님
print(deQueue())
print(deQueue())
print(deQueue())

print(Q)

>>
1
1
2
3
Queue Empty
None
```

```
[1, 2, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```python
# Queue2 (python)
Q = []

Q.append(1)  # >> enQ : push()
print(Q)
Q.append(2)
print(Q)
Q.append(3)
print(Q)

print(Q.pop(0))  # 리스트는 앞에꺼 땡겨줌
print(Q)
print(Q.pop(0))  # pop(0)은 조금 시간이 걸림
print(Q)
print(Q.pop(0))  # deQ : pop()
print(Q)

>>
[1]
[1, 2]
[1, 2, 3]
1
[2, 3]
2
[3]
3
[]
```

선형 큐 이용시의 문제점

- 잘못된 포화상태 인식
  - 선형 큐를 이용하여 원소의 삽입과 삭제를 계속할 경우, 배열의 앞부분에 활용할 수 있는 공간이 있음에도 불구하고, rear = n-1 인 상태, 즉, 포화상태로 인식하여 더 이상의 삽입을 수행하지 않게 됨
- 해결방법1
  - 매 연산이 이루어질 때마다 저장된 원소들을 배열의 앞부분으로 모두 이동시킴
  - 원소 이동에 많은 시간이 소요되어 큐의 효율성이 급격히 떨어짐
- 해결방법2
  - 1차원 배열을 사용하되, 논리적으로는 배열의 처음과 끝이 연결되어 원형 형태의 큐를 이룬다고 가정하고 사용
  - 원형 큐의 논리적 구조 (ppt참고)

원형 큐의 구조

- 초기 공백 상태
  - front = rear = 0
- Index의 순환
  - front와 rear의 위치가 배열의 마지막 인덱스인 n-1를 가리킨 후, 그 다음에는 논리적 순환을 이루어 배열의 처음 인덱스인 0으로 이동해야 함
  - 이를 위해 나머지 연산자 mod를 사용함
- front 변수
  - 공백 상태와 포화 상태 구분을 쉽게 하기 위해 front가 있는 자리는 사용하지 않고 항상 빈자리로 둠
- 삽입 위치 및 삭제 위치

	삽입 위치	삭제 위치
선형큐	rear = rear + 1	front = front + 1
원형큐	rear = (rear + 1) mod n	front = (front + 1) mod n

원형 큐의 연산 과정

ppt 그림보고 이해하기 (280p)

원형 큐의 구현

- 초기 공백 큐 생성
  - 크기 n인 1차원 배열 생성
  - front와 rear를 0으로 초기화
- 공백상태 및 포화상태 검사: isEmpty(), isFull()

  ....

  ppt 참고!!!

```
# 원형 Queue (C)
# 파이썬에서는 선형, 원형 차이 없음
# front, rear 이용
SIZE = 4
Q = [0] * SIZE
front, rear = 0, 0   # 크기가 고정됨

def enQueue(item):
    global rear  # 함수 내부에서 전역변수를 수정하려면 global이 필요
    if (rear+1) % SIZE == front:   # full 꽉찬상태
        print("Queue Full")
    else:
        rear = (rear + 1) % SIZE
        Q[rear] = item

def deQueue():
    global front
```

```
    if front == rear:
        print("Queue Empty")
    else:
        front = (front + 1) % SIZE
        return Q[front]

def Qpeek():
    if front == rear:
        print("Queue Empty")
    else:
        return Q[(front + 1) % SIZE]

enQueue(1)
enQueue(2)
enQueue(3)

print(deQueue())  # 삭제되는 건 아님
print(deQueue())
print(deQueue())
enQueue(4)
print(deQueue())

enQueue(5)
print(deQueue())

print(Q)

>>
1
2
3
4
5
[4, 5, 2, 3]
```

연결큐의 구조 (XX)

2. 우선순위 큐 (Priority Queue)

개념

- 우선순위 큐의 특성
  - 우선순위를 가진 항목들을 저장하는 큐
  - **FIFO 순서가 아니라 우선순위가 높은 순서대로 먼저 나가게 된다**
- 우선순위 큐의 적용 분야
  - 시뮬레이션 시스템
  - 네트워크 트래픽 제어
  - 운영체제의 테스크 스케줄링
- 우선순위 큐의 구현
  - 배열을 이용한 우선순위 큐
  - 리스트를 이용한 우선순위 큐
- 우선순위 큐의 기본연산

    - 삽입: enQueue
    - 삭제: deQueue

배열을 이용한 우선순위 큐

- 배열을 이용하여 우선순위 큐 구현
    - 배열을 이용하여 자료 저장
    - 원소를 삽입하는 과정에서 우선순위를 비교하여 적절한 위치에 삽입하는 구조
    - 가장 앞에 최고 우선순위의 원소가 위치하게 됨
- 문제점
    - 배열을 사용하므로, 삽입이나 삭제 연산이 일어날 때 원소의 재배치가 발생함
    - 이에 소요되는 시간이나 메모리 낭비가 큼

3. 큐의 활용: 버퍼 (Buffer)

버퍼

- 버퍼
    - 데이터를 한 곳에서 다른 한 곳으로 전송하는 동안 일시적으로 그 데이터를 보관하는 메모리의 영역
    - 버퍼링: 버퍼를 활용하는 방식 또는 버퍼를 채우는 동작을 의미
- 버퍼의 자료 구조
    - 버퍼는 일반적으로 입출력 및 네트워크와 관련된 기능에서 이용
    - 순서대로 입력/출력/전달되어야 하므로 FIFO방식의 자료구조인 큐가 활용됨

키보드 버퍼

ppt 303p

예제) 마이쮸

- Queue를 이용하여 마이쮸 나눠주기 시뮬레이션

```

```

<연습문제2>

```

```

---

4. BFS(Breadth First Search) 🌟

- 그래프: 비선형 구조
  - 표현 방법 : 인접행렬
  - 순회 : DFS(재귀), BFS
- 그래프를 탐색하는 방법에는 크게 두 가지가 있음
  - 깊이 우선 탐색 (DFS) - stack
  - 너비 우선 탐색 (BFS) - queue
- 너비우선탐색은 탐색 시작점의 인접한 정점들을 먼저 모두 차례로 방문한 후에, 방문했던 정점을 시작점으로 하여 다시 인접한 정점들을 차례로 방문하는 방식
- 인접한 정점들에 대해 탐색을 한 후, 차례로 다시 너비우선탐색을 진행해야 하므로, 선입선출 형태의 자료구조인 큐를 활용함

ppt 그림참고 311p

BFS 알고리즘

- 입력 파라미터: 그래프G와 탐색 시작점v **(deQ시 방문처리 -> 방문처리할때 해야할 일 하기 => 출발점에서 얼만큼 떨어져있는지 visited를 보고 확인가능)**

```
def BFS(G, v):  # 그래프G, 탐색 시작점v
    visited = [0] * (n+1)  # n: 정점의 개수
    q = []

    q.append(v)             # 시작정점 v를 enQueue
    visited[v] = 1          # 방문한 것으로 표시

    while len(q) != 0:      # 큐가 비어있지 않은 경우
        t = q.pop(0)        # deQueue(왼쪽 원소 반환)
        for w in G[t]:      # 정점t와 인접한 정점w에 대해
            if not visited[w]:  # 방문하지 않은 곳이라면
                q.append(w)     #enQueue
                visited[w] = visited[t] + 1    #방문한 것으로 표시
```

BFS 예제

ppt 보면서 이해!! 313p

<연습문제3>

다음은 연결되어 있는 두 개의 정점 사이에 간선을 순서대로 나열 해놓은 것이다. 모든 정점을 너비우선 탐색하여 경로를 출력하시오. 시작 정점을 1로 시작하시오.

- 1 2 1 3 2 4 2 5 4 6 5 6 6 7 3 7
- 출력 결과의 예) 1-2-3-4-5-7-6

```
# < 1. 인접행렬!! >
'''
7 8
1 2 1 3 2 4 2 5 4 6 5 6 6 7 3 7
```

```
'''

def bfs(v):
    # 큐, 방문처리
    Q = []
    visit = [0] * (V+1)

    # enQ(v), visit(v)
    Q.append(v)
    visit[v] = 1
    print(v, end=' ')
    # 큐가 비어있지 않은 동안
    while Q:
        # v = deQ()
        v = Q.pop(0)
        # v의 인접한 정점(w), 방문 안한 정점이면
        for w in range(1, V+1):
            if G[v][w] == 1 and visit[w] == 0:
                # enQ(w), 방문처리(w) 해야함
                Q.append(w)
                visit[w] = 1
                print(w, end=' ')

# 입력 -> 인접행렬
V, E = map(int, input().split())
temp = list(map(int, input().split()))
# 인접행렬 초기화
G = [[0] * (V+1) for _ in range(V+1)]
# 인접행렬 저장
for i in range(E):
    s, e = temp[2*i], temp[2*i+1]
    G[s][e] = G[e][s] = 1
# 인접행렬 출력
for i in range(1, V+1):
    print("{} {}".format(i, G[i]))

bfs(1)

>>
1 2 3 4 5 7 6
```

```
# < 2. 인접리스트!! >
'''
7 8
1 2 1 3 2 4 2 5 4 6 5 6 6 7 3 7
'''

def bfs(v):
    Q = []
    visit = [0] * (V+1)

    # enQ
    Q.append(v)
    visit[v] = 1
    print(v, end=' ')

    while Q:
```

```
        v = Q.pop(0)
        for w in G[v]:
            if not visit[w]:
                Q.append(w)
                visit[w] = 1
                print(w, end=' ')

# 입력 -> 인접리스트
V, E = map(int, input().split())
temp = list(map(int, input().split()))
# 인접리스트
G = [[] for _ in range(V+1)]
# print(G) # [[], [], [], [], [], [], [], []]
for i in range(E):
    s, e = temp[2*i], temp[2*i+1]
    G[s].append(e)
    G[e].append(s)
# print(G)  # [[], [2, 3], [1, 4, 5], [1, 7], [2, 6], [2, 6], [4, 5, 7], [6, 3]]

bfs(1)

>>
1 2 3 4 5 7 6
```

```
# < 3. 1에서 가장 멀리 있는 정점의 번호는 얼마이고 몇칸 떨어져 있을까? >
'''
7 8
1 2 1 3 2 4 2 5 4 6 5 6 6 7 3 7
'''

def bfs(v):
    Q = []

    # enQ
    Q.append(v)
    visit[v] = 1
    print(v, end=' ')

    while Q:
        v = Q.pop(0)
        for w in G[v]:
            if not visit[w]:
                Q.append(w)
                visit[w] = visit[v] + 1
                print(w, end=' ')

# 입력 -> 인접리스트
V, E = map(int, input().split())
temp = list(map(int, input().split()))
# 인접리스트
G = [[] for _ in range(V+1)]
# print(G)
visit = [0] * (V + 1)  # 여기서 선언돼야함
for i in range(E):
    s, e = temp[2*i], temp[2*i+1]
    G[s].append(e)
    G[e].append(s)
```

```
# print(G)

bfs(1)
print()

# 1에서 가장 멀리 있는 정점의 번호는 얼마이고 몇칸 떨어져 있을까?
maxI = 0
for i in range(1, V+1):
    if visit[maxI] < visit[i]:
        maxI = i
print(maxI, visit[maxI]-1)

>>
1 2 3 4 5 7 6
6 3
```

```
# < 4. 딕셔너리 >
'''
7 8
1 2 1 3 2 4 2 5 4 6 5 6 6 7 3 7
'''

# 1에서 가장 멀리 있는 정점의 번호는 얼마이고 몇칸 떨어져 있을까?

def bfs(v):
    Q = []

    # enQ
    Q.append(v)
    visit[v] = 1
    print(v, end=' ')

    while Q:
        v = Q.pop(0)
        for w in G[v]:
            if not visit[w]:
                Q.append(w)
                visit[w] = visit[v] + 1
                print(w, end=' ')

# 입력 -> 인접리스트
V, E = map(int, input().split())
temp = list(map(int, input().split()))
# 인접리스트
# G = [[] for _ in range(V+1)]
G = {i:[] for i in range(1, V+1)}
# print(G)
visit = [0] * (V + 1)  # 여기서 선언돼야함
for i in range(E):
    s, e = temp[2*i], temp[2*i+1]
    G[s].append(e)
    G[e].append(s)
# print(G)

bfs(1)
print()
```

```
# 1에서 가장 멀리 있는 정점의 번호는 얼마이고 몇칸 떨어져 있을까?
maxI = 0
for i in range(1, V+1):
    if visit[maxI] < visit[i]:
        maxI = i
print(maxI, visit[maxI]-1)

>>
1 2 3 4 5 7 6
6 3
```