기본적인 자료구조

Algorithms

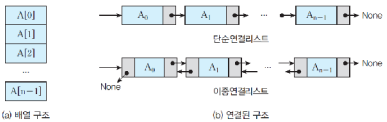
Division of Artificial Intelligence Engineering
(Deptartment of IT Engineering)
Prof. Kang, Jiwoo

목차

리스트의 종류

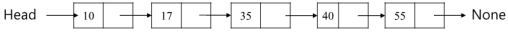
- **ArrayList**

- **LinkedList**
  - Singly Linked List
  - Doubly Linked List (이중연결리스트)
  - Circular Linked List (원형연결리스트)
  - Circular Doubly Linked List

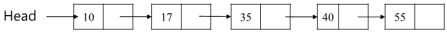


리스트의 종류

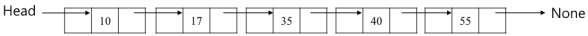
- Singly Linked List



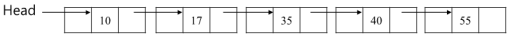
- **Circular** Linked List



- Doubly Linked List

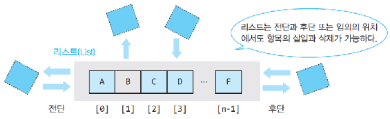


- **Circular** Doubly Linked List



리스트

- 항목들이 순서대로 나열, 각 항목들은 위치를 갖음



- 파이썬에서 자료구조 “리스트”가 필요하면
  - 파이썬의 리스트를 사용
  - 파이썬 리스트
    - 자료구조 (ArrayList / Linkedlist) 를 구현한 클래스

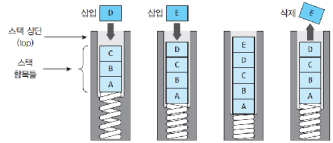
파이썬 리스트

- 파이썬에서 ArrayList가 필요하면

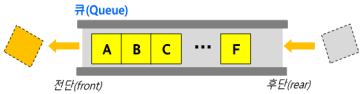
```
from collections import
    queue =
    queue.append(new_element)
    value = queue.pop()
    queue.appendleft(new_element)
    value = queue.popleft()
```

스택과 큐

- 후입선출(LIFO: Last-In First-Out) (FILO: First-In Last-Out)



- 선입선출(FIFO: First-In First-Out)



파이썬 리스트를 이용한 스택 큐

- Python Simple Usage Example (보통 구현된 리스트를 활용)

```
list = []
Stack
    list =
Queue
    list =
value = list.pop()
```

```
list = []
list.append(new_element)
Stack
    value =
Queue
    value =
```

ArrayList vs LinkedList

(파이썬 List가 각각 ArrayList와 LinkedList라고 가정)

1. list = []

2. list.append(10)

3. for i range(10):
       list.append(i)

4. list = [11] + list

ArrayList vs LinkedList

(파이썬 List가 각각 ArrayList와 LinkedList라고 가정)

5. list.pop()

6. list.pop(0)

7. for i range(len(list)):
       print(list[i])

파이썬 리스트

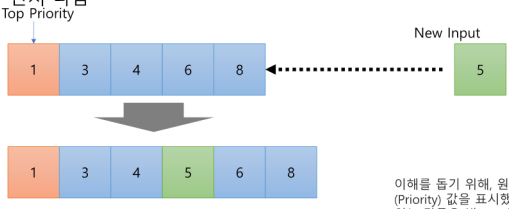
- **파이썬에서 스택이 필요하면**
  - 방법 1: 파이썬 리스트 이용
  - 방법 2: 큐 모듈(queue)의 클래스 사용
  - 방법 3: 직접 클래스로 구현해서 사용

- **파이썬에서 큐가 필요하면**
  - 방법 1: 큐 모듈(queue)의 클래스 사용
  - 방법 2: 직접 클래스로 구현해서 사용

- `import queue`
  - `q = queue.` 또는 `q = queue.`
  - `q.put(new_element)`
  - `value = q.get()`
  - `size = q.qsize()`

우선순위 큐

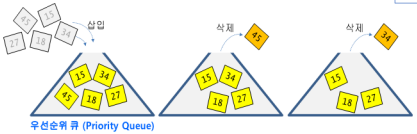
- **우선순위 큐 (Priority Queue)**
  - 일반적인 큐(Queue)는 먼저 집어넣은 데이터가 먼저 나오는 **FIFO (First In First Out) 구조**이지만, **우선순위 큐 (Priority Queue)**는 들어온 순서와 상관없이 **우선 순위가 높은 데이터** (보통 더 높은 우선순위를 더 낮은 값을 표현)가 먼저 나옴



이해를 돕기 위해, 원소의 값에 우선순위(Priority) 값을 표시했으며, 원소가 저장하고 있는 값들은 별도로 존재할 수 있음

우선순위 큐

- **우선순위의 개념을 큐에 도입한 자료구조**
  - 선형 자료구조가 아님
  - 힙(heap)이 가장 효율적인 구현 방법



- **파이썬에서 우선순위 큐가 필요하면**
  - heapq 모듈을 사용
  - 큐 모듈(queue)의

- **import queue**
  - q = queue.
  - q.put((priority, element))
  - value = q.get()
  - size = q.qsize()

```
>>> import queue
>>> q = queue.PriorityQueue()
>>> q.put((10, "ABC"))
>>> q.put((5, "DEF"))
>>> q.put((8, "GH"))
>>> print(q.get())
(5, 'DEF')
>>> print(q.get())
(8, 'GH')
>>> print(q.get())
(10, 'ABC')
```

- Python구현된 Priority Queue는 **priority가 낮은 순서가 우선**

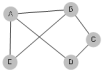
그래프 (Graph)란?

- **그래프 (Graph) 구조**
  - 버스 정류장과 여러 노선이 함께 포함된 형태 또는, 링크드인(Linked in)과 같은 사회 관계망 서비스의 연결 등의 형태

그래프 (Graph)란?

- **그래프 (Graph)**
  - 현상이나 사물을 **정점 vertex (노드 node)**과 **간선 edge**으로 표현한 것
  - Graph ***G = (V, E)***
    V: 정점 집합
    E: 간선 집합

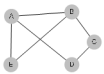
  - 두 정점이 간선으로 연결되어 있으면 하다고 한다
    - **"이웃 관계"** 에 있다고 표현하기도 한다.

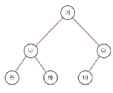


정점 (A, B), (A, D), (A, E), (B, C), (B, E), (C, D)가 서로 **인접** 하다.

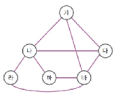
그래프 (Graph)란?

- **경로 (Path)**
  - 정점 A에서 C까지의 경로는 A-B-C, A-E-B-C, A-D-C 등이 있다.





(a) 트리 의
그림 9-1 트리와 그래프의 차이



(b) 그래프 예

지하철 노선도(부산)

K I X 노선도

도시 노선망

전기 회로도

인맥 관계도

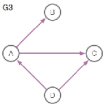
그림 9-2 그래프를 활용한 다양한 예

그래프 (Graph)란?

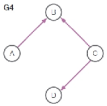
그래프의 종류

- 무방향성 그래프 (Undirected Graph)
  - 간선에 방향이 없는 그래프

- 방향성 그래프 (Directed Graph)
  - G3



G4



- 정점 표현

```
V(G1) = { A, B, C, D }
V(G2)  { A, B, C, D }
```

- 간선 표현

```
E(G1) = { (A, B), (A, C), (A, D), (B, C), (C, D) }
E(G2) = { (A, B), (B, D), (D, C) }
```

- 정점 표현

```
V(G3) = { A, B, C, D }
V(G4) = { A, B, C, D }
```

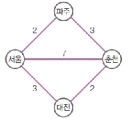
- 간선 표현

```
E(G3)  { <A, B>, <A, C>, <B, A>, <B, D>, <D, C> }
E(G4) = { <A, B>, <C, B>, <C, D> }
```


그래프 (Graph)란?

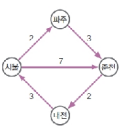
그래프의 종류

- 가중치 그래프 (Weighted Graph)
  - 간선마다 가중치가 다르게 부여된 그래프
  - 간선에 방향이 없는 그래프 (Weighted Undirected Graph)



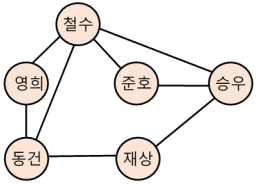
(a) 무방향 가중치 그래프

  - 간선에 방향이 있는 그래프 (Weighted Directed Graph)



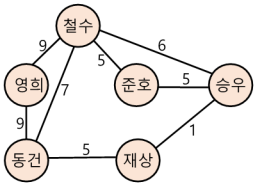
(b) 방향 가중치 그래프

그래프의 예



사람들간의 친분 관계를 나타낸 그래프
(두 도시 간 도로 존재 여부, 두 집안 간의 혼인 여부 등)

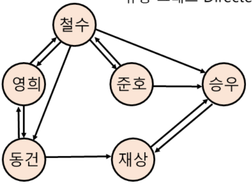
그래프의 예



친밀도를 가중치로 나타낸 친분관계 그래프
(도시들 간의 거리, 두 지점 사이에 묻힌 가스 파이프의 용량,
두 공항 사이의 비행 시간 등)

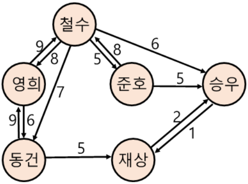
그래프의 예

유향 그래프 Directed graph=Digraph



방향을 고려한 친분관계 그래프
(각 사람이 다른 사람에게 애정을 가지고 있는가, 기업들 간의 제품 공급 관계, 제품 생산 공정에서의 선후 관계 등)

그래프의 예



가중치를 가진 유향 그래프
(누가 누구를 얼마만큼 좋아하는지, 서울과 LA간 비행기 노선에서 가는 데 걸리는 시간과 오는 데 걸리는 시간이 다른 경우)

SOOKMYUNG 3D AI UNIVERSE LABORATORY

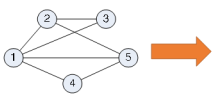
그래프 표현법

- **그래프는 정점의 집합과 간선의 집합의 결합**
  - → 그래프를 표현하는 문제는 "정점의 집합"과 "간선의 집합"의 표현 문제
  - → 간선은 정점과 정점이 "인접" 관계에 있음을 나타내는 존재
  - → 그래프의 표현 문제는
    "**간선, 즉 정점과 정점의 인접 관계를 어떻게 나타내는가?**"의 문제로 귀결

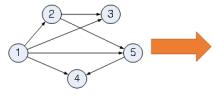
- **행렬**을 이용하는 방식은
- **리스트**를 이용하는 방식은

그래프 표현법

- **인접 행렬 : 정점끼리의 인접 관계를 나타내는 행렬**
  - 그래프의 **정점의 수를 *N*** 이라고 한다면, ***N x N*** 크기의 행렬
  - 행렬의 각 원소를 한 정점과 또 다른 정점이 **인접**해 있는 경우(즉, 정점 사이에 **간선이 존재**하는 경우)에는 **1**로 표시하고, 인접해 있지 않은 경우에는 0으로 표시



	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	0	1
3	1	1	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0



	1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	0	0
3	1	1	0	0	0
4	1	0	0	0	1
5	1	1	0	0	0

그래프 표현법

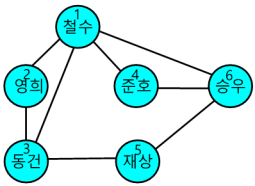
- **인접 행렬 : 정점끼리의 인접 관계를 나타내는 행렬**
  - ***N x N*** 행렬로 표현
    - **원소 (i, j) == 1** : 정점 i 와 정점 j 사이에 간선이 있음
    - **원소 (i, j) == 0** : 정점 i 와 정점 j 사이에 간선이 없음

N : 총 정점의 수

  - **유향 그래프**의 경우
    - 원소 (i, j)는 정점 i 로부터 정점 j 로 연결되는 **간선이 있는지**, 두 정점이 **인접**한지를 나타냄
    - 원소 (i, j)와 원소 (j, i)가 대칭적이지 않음,
      무향 그래프 인 경우에는 항상 원소 (i, j) == 원소 (j, i) 대칭
  - **가중치가 있는 그래프**의 경우
    - 원소 (i, j)는 **1 대신에 가중치 값**을 가짐

그래프 표현법

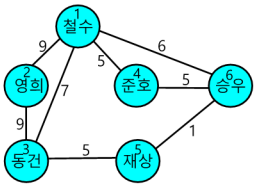
- **인접 행렬 (무향 그래프의 예)**



	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	1	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	1	0	0	1
6	1	0	0	1	1	0

그래프 표현법

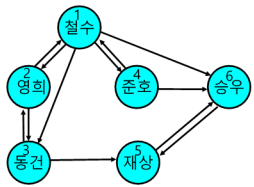
- 인접 행렬 (가중치 무향 그래프의 예)



	1	2	3	4	5	6
1	0	9	7	5	0	6
2	9	0	9	0	0	0
3	7	9	0	0	5	0
4	5	0	0	0	0	5
5	0	0	5	0	0	1
6	6	0	0	5	1	0

그래프 표현법

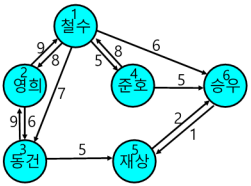
- 인접 행렬 (유향 그래프의 예)



	1	2	3	4	5	6
1	0	1	1	1	0	1
2	1	0	1	0	0	0
3	0	1	0	0	1	0
4	1	0	0	0	0	1
5	0	0	0	0	0	1
6	0	0	0	0	1	0

그래프 표현법

- **인접 행렬 (가중치 유향 그래프의 예)**



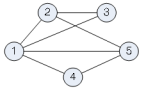
	1	2	3	4	5	6
1	0	8	7	5	0	6
2	9	0	6	0	0	0
3	0	9	0	0	5	0
4	8	0	0	0	0	5
5	0	0	0	0	0	2
6	0	0	0	0	1	0

그래프 표현법

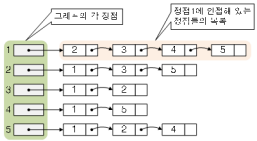
- **인접 행렬 (Adjacency Matrix) 표현법**
  - 장점
    - 이해하기 쉽고, 간선 존재 여부를 즉각 알 수 있음
  - 단점
    - **N^2에 비례하는 공간** 필요 (N은 정점 수)
    - 행렬의 모든 원소를 채우는 데에만 N^2에 비례하는 시간 필요
  - 간선의 밀도가 높은 그래프에서는 유리
    - 정점이 100만개인데 간선이 200만개 밖에 없다면?

그래프 표현법

- **인접 리스트 : 그래프 내의 각 정점의 인접 관계를 표현하는 리스트**
  - N **개의 리스트** (연결리스트 또는 배열)로 표현
  - i번째 리스트는 정점 i에 인접한 정점들을 리스트로 연결
  - 가중치 (Weighted) 있는 그래프의 경우: 리스트에 가중치 (Weight)도 보관



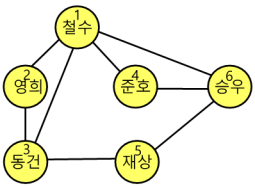
정점	인접 정점
1	2, 3, 4, 5
2	1, 3, 5
3	1, 2
4	1, 5
5	1, 2, 4

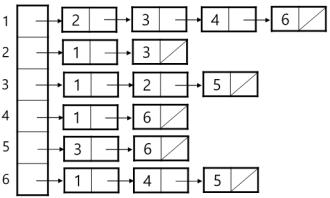


그래프 표현법

- **인접 리스트 (무향 그래프)**

❖ **무향 그래프**에서 총 간선 수의 **두 배의 항목**으로 표현
❖ **유향 그래프**에서는 간선 하나당 **항목 하나씩** 필요

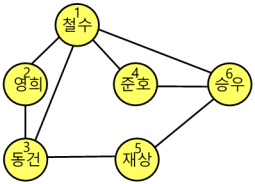




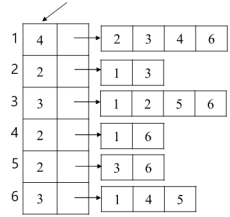
그래프 표현법

▪ 인접 리스트 (무향 그래프)

❖ **무향 그래프**에서 총 간선 수의 **두 배의 항목**으로 표현
❖ **유향 그래프**에서는 간선 하나당 **항목 하나씩** 필요



각 정점에 인접한 정점 수

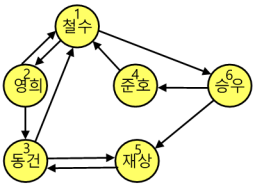


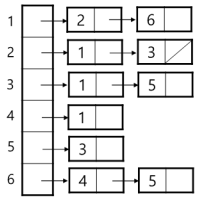
- 정해진 건 없다. (표준은 없다) LinkedList, ArrayList, 위 예처럼 배열 표현을 해도 문제 없다.

그래프 표현법

▪ 인접 리스트 (유향 그래프)

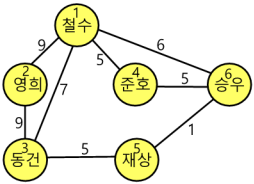
❖ **무향 그래프**에서 총 간선 수의 **두 배의 항목**으로 표현
❖ **유향 그래프**에서는 간선 하나당 **항목 하나씩** 필요

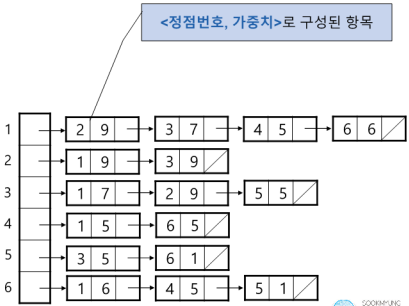




그래프 표현법

- **인접 리스트 (가중치 무향 그래프)**





그래프 표현법

- **인접 리스트 (Adjacency List) 표현법**
  - 장점
    - 간선 총 수에 비례하는 양만큼만 필요
    - 모든 가능한 정점 쌍에 비해 간선 수가 적을 때 유리
  - 단점
    - 거의 모든 정점에 간선이 있는 경우 구조적 접근 (Indexing) 비 효율성 (LinkedList 구현일 경우 메모리에서도 손해, 리스트 연산량 오버헤드)
    - 정점 i와 j 간에 간선이 있는지 알아볼 때 시간이 많이 걸림

THANK YOU
Q&A

- **Name:** Prof. Kang, Jiwoo
- **Office:** Saehim Hall # 605 (02-2077-7445)
- **E-Mail:** jwkang@sm.ac.kr
- **Webpage**: http://ai.sookmyung.ac.kr