

6-9 도수 정렬

- 분포수 세기 정렬
- 원소의 대소 관계 판단 X

6-9-1. 도수 정렬 알아보기

- 단계

- 1) 도수 분포표 만들기

• 배열 =

--	--	--	--	--	--

 $\rightarrow \min, \max$

→ 도수분포표

— 배열

• 크기: $\max - \min + 1$

• 도수분포표 [원소] + 1

2) 누적 도수분포표 만들기

• 누적 도수분포표 [i] = 누적 도수분포표 [i] + 도수분포표 [i-1]

3) 작업용 배열 만들기

• 배열 [i] = 작업용 배열 [누적 도수분포표 [배열[i]-1]]

4) 배열 복사하기

• 배열 [i] = 작업용 배열 [i]

PART 7 문자열 검색

- 어떤 문자열 안에 찾고자 하는 문자열이 포함되어 있는지 검사, 만약 포함되어 있다면 위치 반환

- 텍스트

— 검색되는 쪽의 문자열

피벗

— 찾아내는 문자열

7-1 브루트 포스법

7-1-1. 브루트 포스법 알아보기

- 단순법

— 선형검색을 단순하게 확장

- 효율이 좋지 않음
 - 이미 검사한 위치를 기억하지 못함

- 방법

- 텍스트 :

A	B	A	B	C	D	E
---	---	---	---	---	---	---

- 패턴 :

A	B	C
---	---	---

→ a 1.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 1번 째 문자 일치

2.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 2번 째 문자 일치

3.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 3번 째 문자 불일치

b 4.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 1번 째 문자 불일치

c 5.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 1번 째 문자 일치

6.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 2번 째 문자 일치

7.

A	B	A	B	C	D	E
---	---	---	---	---	---	---

A	B	C
---	---	---

 → 3번 째 문자 일치

⇒ return (4-2) = 모두 일치시, 텍스트 검색 꽂 인덱스 - 패턴 검색 꽂 인덱스

- 멤버십 연산자 - 파이썬

- 패턴 in 텍스트

- 존재 → TRUE, 없음 → FALSE

- 패턴 not in 텍스트

- in 기호 과 와 반대

II-2 KMP 법

- 이전에 검사한 결과를 효율적으로 사용

- 브루트 포스법 단점 개선

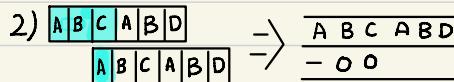
7-2-1. KMP법 알아보기

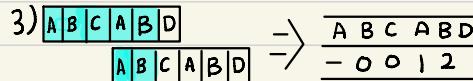
- 텍스트와 패턴안에서 겹치는 문자열을 찾아내 검사를 다시 시작할 위치를 구하여 패턴의 이동을 되도록이면 크거나하는 알고리즘
- '몇 번째 문자부터 다시 검색할지' 값을 표로 만들
- 사용할 표 만들기
 - 건너뛰기 표

• 패턴과 패턴을 서로 겹치도록 두고 검사를 시작할 곳을 계산

• ex) 1) 

- 파란색 부분이 일치하는지 검사

2) 

3) 

4) 

- 텍스트를 스캔하는 커서는 앞으로 갈 땐 뒤로 돌아오지 않음

- 브루트 포스법에 없는 특징

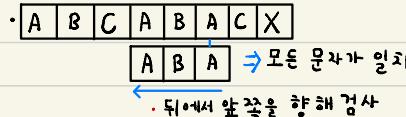
7-3 보이어·무어 법

- 실제 문자열 검색 알고리즘에서 널리 사용하는 알고리즘

7-3-1. 보이어·무어 법 알아보기

• 방법

- 패턴의 끝 문자에서 시작하여 앞쪽을 향해 검사 수행



- 일치하지 않는 문자 발견하면 미리 준비한 표를 바탕으로 패턴이 이동하는 값 결정

• 표 만들기

— 패턴에 포함되지 않는 문자를 만날 경우

• 패턴 이동량 = 패턴의 길이

— 패턴에 포함되는 문자를 만날 경우

• 패턴 이동량

= 패턴의 길이 - 마지막에 나오는 문자의 인덱스 - 1

. 패턴 안에 중복되지 않는 몇 글자 일경우

— 패턴 이동량 = 패턴의 길이

- ex) 패턴 → 'ABAC'

A	B	C	나머지 문자
1	2	4	4
↑	↑	↓	4-1-1
↑			4-2-1

• 보이어 우어법 예시

— 0 1 2 3 4 5 6 7 8
텍스트: A D E C C A B A C

패턴: A B A C → 일치

→ 불일치 = 패턴 이동량 = E → 4

· 현재 주목중인 문자 인덱스 2 $\xrightarrow{+4}$ 6 이동

0 1 2 3 4 5 6 7 8
A D E C C A B A C
A B A C → 불일치 : 패턴 이동량 = B → 2

0 1 2 3 4 5 6 7 8
A D E C C A B A C

A B A C →

return:

일치 \Rightarrow index = 5에서

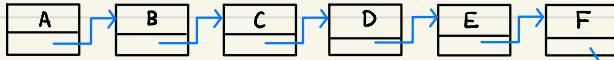
PART 8 리스트

- 데이터에 순서를 매겨 늘어놓은 자료구조

8-1 연결리스트

8-1-1. 연결리스트 알아보기

- 머리노드



— 데이터를 사슬처럼 연결

꼬리노드

뒤쪽노드를 가리키는 포인터

- 노드

— 연결리스트에서 각각의 원소

— 구성

- 데이터

- 포인터

— 뒤쪽 노드를 참조하는

— 종류

- 머리노드

— 맨 앞의 노드

- 꼬리노드

— 맨 끝의 노드

- 앞쪽 노드, 뒤쪽노드

— 특정 노드의 앞, 뒤

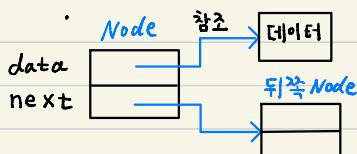
8-1-2. 배열로 연결리스트 만들기

- 단점

— 데이터를 삽입·삭제함에 따라 데이터를 옮겨야 하므로 효율적이지 않음

8-2 포인터를 이용한 연결리스트

8-2-1. 포인터로 연결리스트 만들기



- 자기참조형

— 자신과 같은 클래스 형의 인스턴스를 참조하기

위한 참조용 필드 next가 있는 구조

8-3 커서를 이용한 연결리스트

8-3-1. 커서로 연결리스트 만들기

· 커서

- int형 정수값인 인덱스로 나타낸 포인터
- 머리노드를 나타내는 head
- 머리노드의 인덱스값 소유
- 꼬리노드의 뒤쪽 커서
- - |

8-3-2. 배열안에 비어 있는 원소 처리하기

· 노드 삽입

· 저장장소

- 배열 안에서 가장 끝쪽에 있는 인덱스의 위치

· 노드 삭제

- 배열 안에 빈 공간이 생김

· 삭제 여러번 반복

→ 배열안에 빈 공간이 많이 생김 → 관리 필요

8-3-3. 프리 리스트

- 삭제된 레코드 그룹을 관리할 때 사용하는 자료구조
- 연결리스트 + 프리 리스트

8-4 원형 이중 연결 리스트

8-4-1. 원형리스트 알아보기

- 연결리스트의 꼬리노드가 다시 머리노드를 가리키는 모양

8-4-2. 이중 연결 리스트

· 연결리스트의 단점

- 앞쪽 노드를 찾기 어렵다는 것

- → 이중 연결리스트에서 단점 극복

· 노드

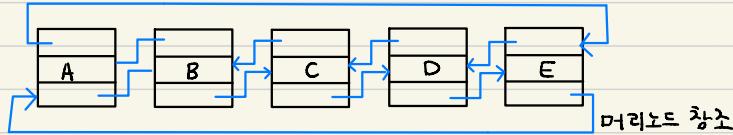
· data

- 뒤쪽노드, 앞쪽노드에 대한 각각의 포인터

8-4-3. 원형 이중 연결 리스트

- 원형리스트 + 이중 연결 리스트

head 꼬리노드참조



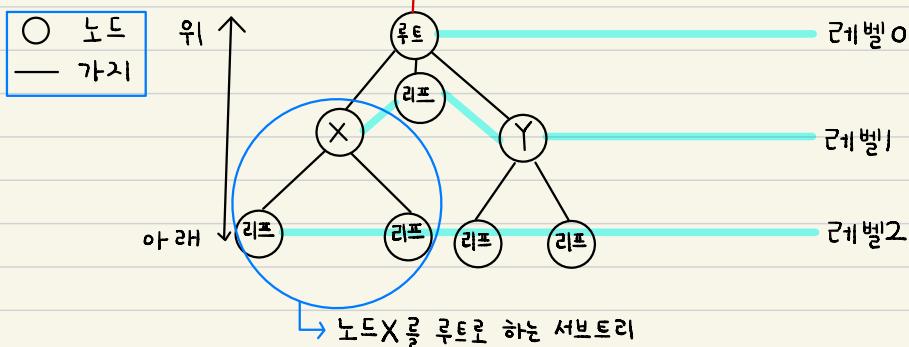
PART9 트리

9-1 트리구조

- 데이터 사이의 계층 관계를 표현하는 구조

9-1-1. 트리구조와 관련 용어

- 트리구조



- 루트

- 트리의 가장 위쪽에 있는 노드

- 루트는 트리에 1개만 존재

- 리프

- 트리의 가장 아래쪽에 있는 노드

- 단말노드, 외부노드

- 가지가 더 이상 뻗어나갈 수 없는 마지막노드

- 비단말노드

- 내부노드

- 리프를 제외한 모든 노드

- 자식
 - 어떤 노드와 가지로 연결되었을 때 아래쪽 노드

- 부모
 - 어떤 노드와 가지로 연결되었을 때 가장 위쪽 노드

- 형제
 - 부모가 같은 노드

- 조상
 - 어떤 노드에서 위쪽으로 가지를 따라가면 만나는 모든 노드

- 자손
 - 어떤 노드에서 아래쪽으로 가지를 따라가면 만나는 모든 노드

- 레벨
 - 루트에서 얼마나 멀리 떨어져 있는지를 나타내는 것

- 차수
 - 각 노드가 갖고 있는 자식의 수
 - n진 트리
 - 모든 노드의 차수가 n이하인 트리

- 높이
 - 루트에서 가장 멀리 있는 리프 까지의 거리
 - = 리프 레벨의 최댓값

- 서브트리
 - 어떤 노드를 루트로하고 그 자손으로 구성된 트리

- 빙트리
 - = 널트리
 - 노드와 가지가 전혀 없는 트리

9-1-2. 순서트리와 무순서트리

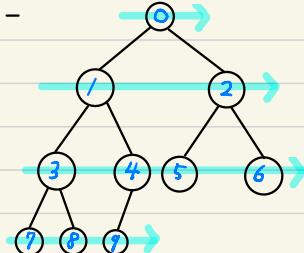
- 순서트리
 - 형제노드의 순서관계 있음

- 무순서트리
 - 형제노드의 순서관계 없음

9-1-3. 순서트리의 검색

• 너비 우선 검색

- 둘 꼭 우선 검색, 가로검색, 수평검색



• 깊이 우선 검색

- 세로검색, 수직검색

- 리프에 도달할 때까지 아래쪽으로 내려가면서 검색하는 것을 우선
- 리프에 도달 후, 다시 부모노드로 돌아가고 그 뒤 다시 자식노드로 돌아감
- 전위 순회

• 노드방문 → 왼쪽노드 → 오른쪽노드

– 중위순회

• 왼쪽자식 → 노드방문 → 오른쪽노드

– 후위순회

• 왼쪽자식 → 오른쪽자식 → 노드방문

9-2 이진트리와 이진검색트리

9-2-1. 이진트리 알아보기

- 노드가 왼쪽자식과 오른쪽자식만을 갖는 트리

• 특징

- 왼쪽자식과 오른쪽자식을 구분

• 왼쪽 서브트리, 오른쪽 서브트리

9-2-2. 완전이진트리 알아보기

- 마지막 레벨을 제외하고 모든 레벨에 노드가 가득 차 있음

- 마지막 레벨 → 왼쪽부터 오른쪽으로 노드를 채움

→ 반드시 끝까지 안 채워도 됨

- 높이 $K \rightarrow$ 노드의 최대수 = $2^{K+1} - 1$

- n 개의 노드 \rightarrow 높이 : $\log n$

9-2-3. 이진 검색 트리 알아보기

- 조건

- 왼쪽 서브트리 노드의 키값은 자신의 노드 키값보다 작아야 합니다.
- 오른쪽 서브트리 노드의 키값은 자신의 노드 키값보다 커야 합니다.
- 키값이 같은 노드는 복수로 존재하지 않습니다.

- 특징

- 구조가 단순
- 중위 순회의 깊이 우선 검색을 통하여 노드값을 오름차순으로 얻을 수 있다.
- 이진 검색과 비슷한 방식으로 아주 빠르게 검색할 수 있다.
- 노드를 삽입하기 쉬움