13

CHAPTER

검색

학습목표

- 검색의 개념을 파악한다.
- 검색을 위한 코드 형식을 이해한다.
- 검색의 다양한 알고리즘을 학습한다.
- 검색을 활용한 응용 프로그래밍을 작성한다.

- **검색**
  - '검색'은 정렬된 상태에서 빠르게 원하는 것을 찾을 수 있음

뒤죽박죽 섞여 있는 단어 퍼즐에서는 단어 찾는 데
오랜 시간이 걸림

알파벳 순서로 되어 있는 퍼즐에서는 빠르고 쉽게
단어를 찾을 수 있음

- **검색의 개념**
  - 어떤 집합에서 원하는 것을 찾는 것으로, 탐색이라고도 함

그림 13-1 검색의 다양한 예

  - 검색에는 순차 검색, 이진 검색, 트리 검색 등이 있음
  - 검색에 실패하면 -1을 반환하는 것이 일반적임

- **검색 알고리즘의 종류**
  - 순차 검색
    - 검색할 집합이 정렬되어 있지 않은 상태일 때
    - 처음부터 차례대로 찾아보는 것으로, 쉽지만 비효율적임
    - 집합의 데이터가 정렬되어 있지 않다면 이 검색 외에 특별한 방법 없음
  - 이진 검색
    - 데이터가 정렬되어 있다면 이진 검색도 사용 가능
    - 순차 검색에 비해 월등히 효율적이라 데이터가 몇 천만 개 이상이어도 빠르게 찾아낼 수 있음
  - 트리 검색
    - 데이터 검색에는 상당히 효율적이지만 트리의 삽입, 삭제 등에는 부담

- **순차 검색**
  - 정렬되지 않은 집합의 순차 검색 원리와 구현

그림 13-2 키 순으로 정렬되지 않은 집합

  - 검색에 성공하는 경우

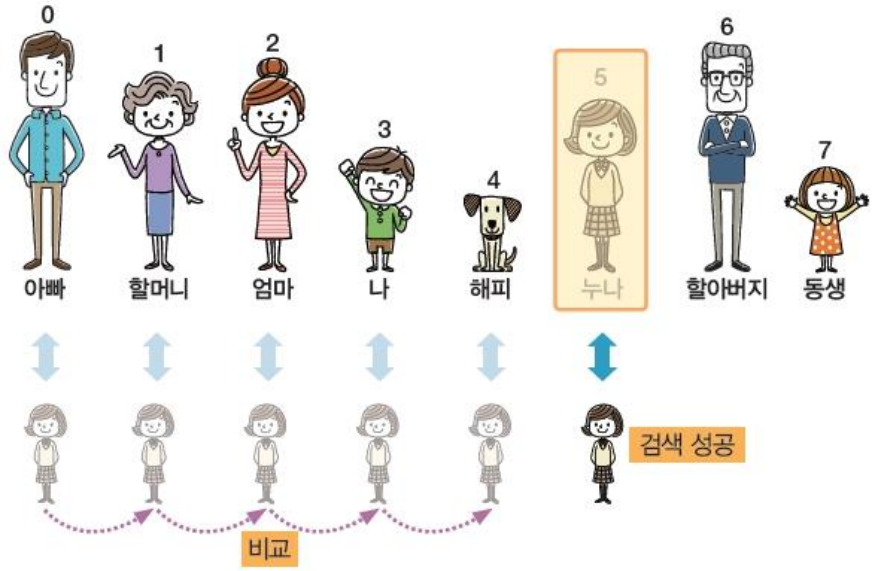


그림 13-3 검색 성공 예

Section 02 순차 검색과 이진 검색 알고리즘의 원리와 구현

- 검색에 실패하는 경우

그림 13-4 검색 실패 예

Code13-01.py 정렬되지 않은 데이터의 순차 검색

```
1  ## 클래스와 함수 선언 부분 ##
2  def seqSearch(ary, fData) :
3      pos = -1
4      size = len(ary)
5      print('## 비교한 데이터 ==> ', end = ' ')
6      for i in range(size) :
7          print(ary[i], end = ' ')
8          if ary[i] == fData :
9              pos = i
10             break
11     print()
12     return pos
13
```

Section 02 순차 검색과 이진 검색 알고리즘의 원리와 구현

```
14 ## 전역 변수 선언 부분 ##
15 dataAry = [188, 150, 168, 162, 105, 120, 177, 50]
16 findData = 0
17
18 ## 메인 코드 부분 ##
19 findData = int(input('* 찾을 값을 입력하세요. --> '))
20 print('배열 -->', dataAry)
21 position = seqSearch(dataAry, findData)
22 if position == -1 :
23     print(findData, '(이)가 없네요.')
24 else :
25     print(findData,' (은)는 ', position, '위치에 있음')
```

실행 결과

```
* 찾을 값을 입력하세요. --> 105
배열 --> [188, 150, 168, 162, 105, 120, 177, 50]
## 비교한 데이터 ==> 188 150 168 162 105
105 (은)는  4 위치에 있음

* 찾을 값을 입력하세요. --> 100
배열 --> [188, 150, 168, 162, 105, 120, 177, 50]
## 비교한 데이터 ==> 188 150 168 162 105 120 177 50
100 (이)가 없네요.
```

SELF STUDY 13-1

Code13-01.py를 수정해서 중복된 데이터가 여러 개 있을 때, 위치 여러 개를 리스트로 만들어서 반환하도록 코드를 변경하자. 코드 중 dataAry와 findData는 다음과 같이 변경한다.

dataAry = [188, 50, 150, 168, 50, 162, 105, 120, 177, 50]

findData = 50

실행 결과

```
배열 --> [188, 50, 150, 168, 50, 162, 105, 120, 177, 50]
50 (은)는 [1, 4, 9] 위치에 있음
```

- 정렬된 집합의 순차 검색 원리와 구현

그림 13-5 키 순으로 정렬된 집합

- 검색에 성공하는 경우

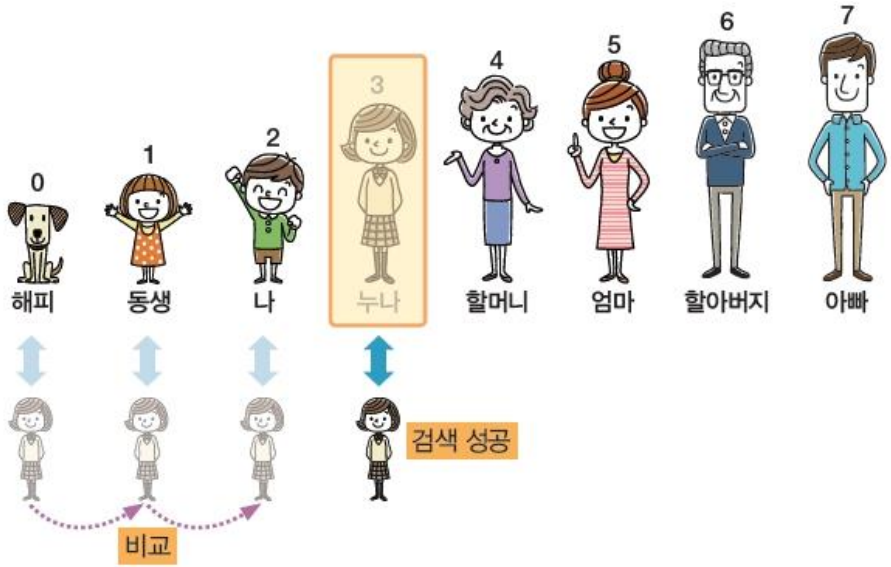


그림 13-6 검색 성공

Section 02 순차 검색과 이진 검색 알고리즘의 원리와 구현

- 검색에 실패하는 경우

그림 13-7 검색 실패

Code13-02.py 정렬된 데이터의 순차 검색

```
1  ## 클래스와 함수 선언 부분 ##
2  def seqSearch(ary, fData) :
3      pos = -1
4      size = len(ary)
5      print('## 비교한 데이터 ==> ', end = ' ')
6      for i in range(size) :
7          print(ary[i], end = ' ')
8          if ary[i] == fData :
9              pos = i
10             break
11         elif ary[i] > fData :
12             break
13     print()
14     return pos
```

```
15
16 ## 전역 변수 선언 부분 ##
17 dataAry = [188, 150, 168, 162, 105, 120, 177, 50]
18 findData = 0
19
20 ## 메인 코드 부분 ##
21 dataAry.sort()
22 findData = int(input('* 찾을 값을 입력하세요. --> '))
23 print('배열 -->', dataAry)
24 position = seqSearch(dataAry, findData)
25 if position == -1 :
26     print(findData, '(이)가 없네요.')
27 else :
28     print(findData, '(은)는 ', position, '위치에 있음')
```

실행 결과

```
* 찾을 값을 입력하세요. --> 150
배열 --> [50, 105, 120, 150, 162, 168, 177, 188]
## 비교한 데이터 ==> 50 105 120 150
150 (은)는  3 위치에 있음

* 찾을 값을 입력하세요. --> 70
배열 --> [50, 105, 120, 150, 162, 168, 177, 188]
## 비교한 데이터 ==> 50 105
70 (이)가 없네요.
```

Section 02 순차 검색과 이진 검색 알고리즘의 원리와 구현

- 순차 검색의 시간 복잡도
  - 정렬되지 않은 집합의 순차 검색은 데이터 개수가 n개라면 시간 복잡도는 O(n)으로 표현됨
  - 정렬된 집합의 순차 검색은 검색할 데이터가 없는 경우에도 데이터의 크기가 작다면 앞쪽만 검색한 후 검색 실패를 효율적으로 확인할 수 있음(시간 복잡도는 O(n)으로 표현)

- **이진 검색**
  - 이진 검색의 원리와 시간 복잡도
    - 이진 검색은 전체를 반씩 잘라 내서 한쪽을 버리는 방식을 사용

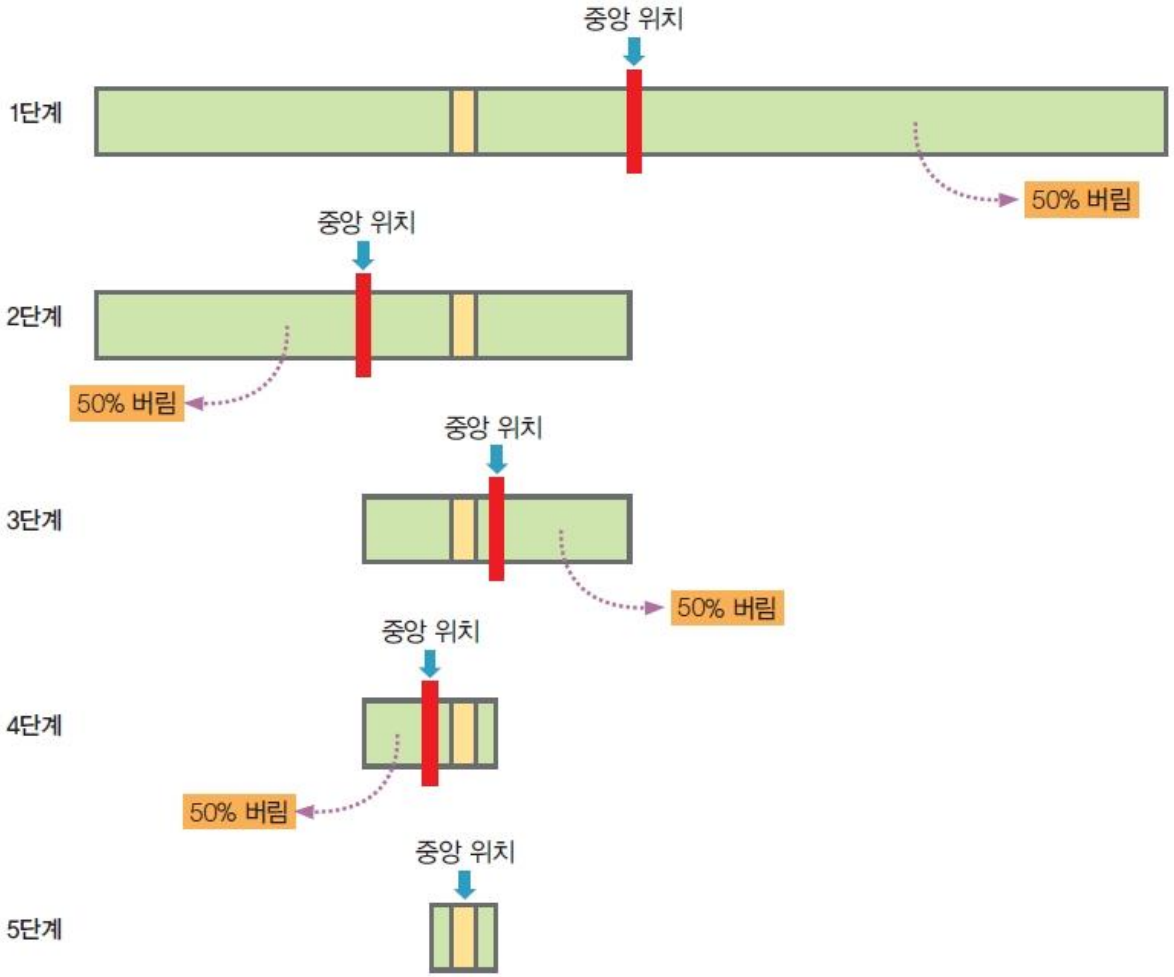
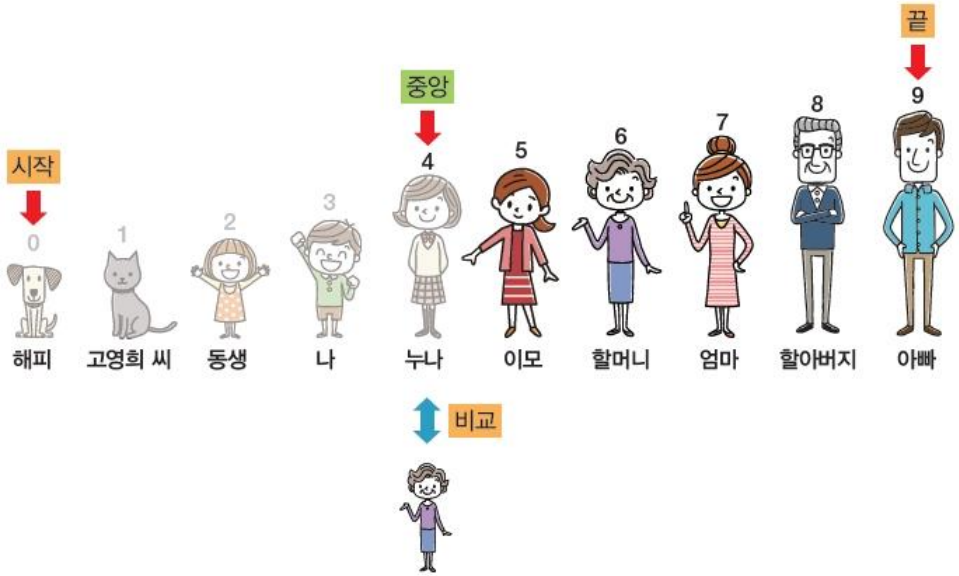


그림 13-8 이진 탐색의 간단한 그림

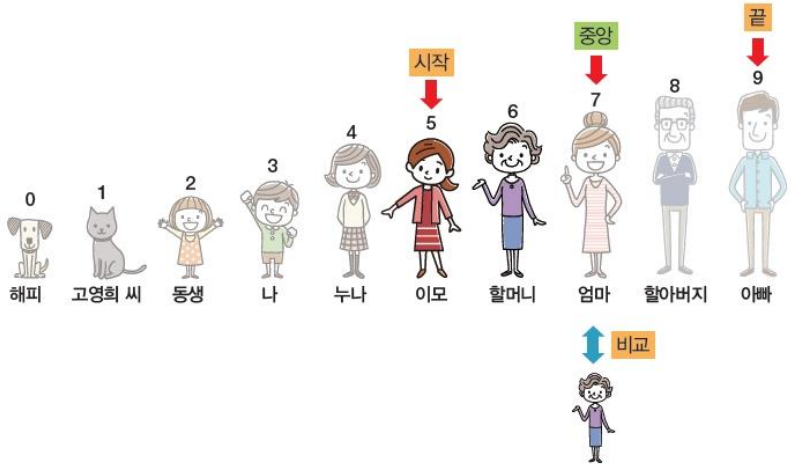
- 이진 검색 구현
  - 정렬된 가족 데이터에서 '할머니'와 키가 같은 사람을 찾는 과정을 이진 탐색으로 구현하는 예

그림 13-9 이진 탐색할 정렬된 가족 데이터

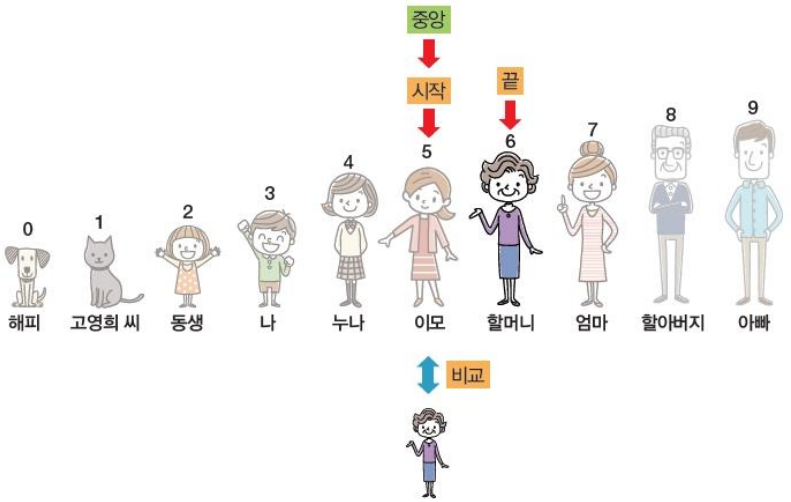
1 전체의 첫 데이터를 '시작'으로 지정하고, 마지막 데이터를 '끝'으로 지정한 후 시작과 끝의 중앙인 누나를 할머니와 비교한다.



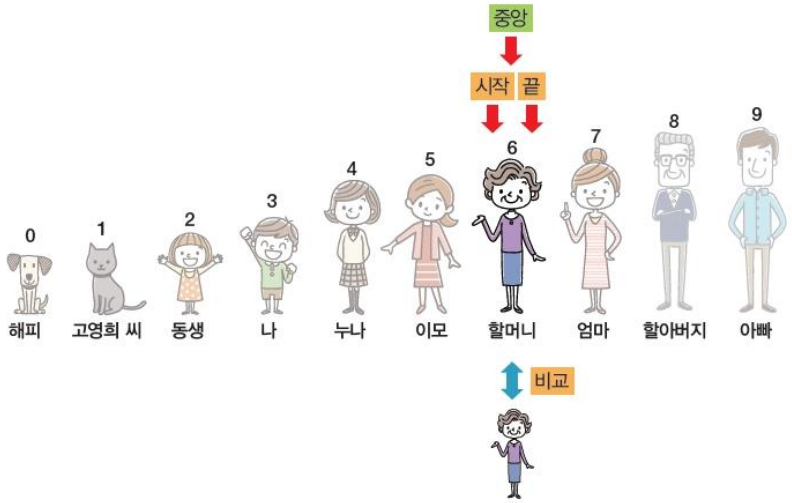
2 끝은 그대로 두고 시작을 중앙(누나)의 바로 오른쪽 이모로 옮긴다. 중앙(누나)의 오른쪽 그룹에서 다시 시작과 끝의 ½ 위치인 새 중앙(엄마)을 할머니와 비교한다.



3 시작은 그대로 두고 끝을 중앙(엄마)의 바로 왼쪽인 할머니로 옮긴다. 중앙(엄마)의 왼쪽 그룹에서 다시 시작과 끝의 ½ 위치인 새 중앙(이모)을 할머니와 비교한다.



4 끝은 그대로 두고 시작을 중앙(이모)의 바로 오른쪽으로 옮긴다. 중앙(이모)의 오른쪽 그룹에서 다시 시작과 끝의 ½ 위치인 새 중앙(할머니)을 할머니와 비교한다.



이진 탐색에서 검색 실패

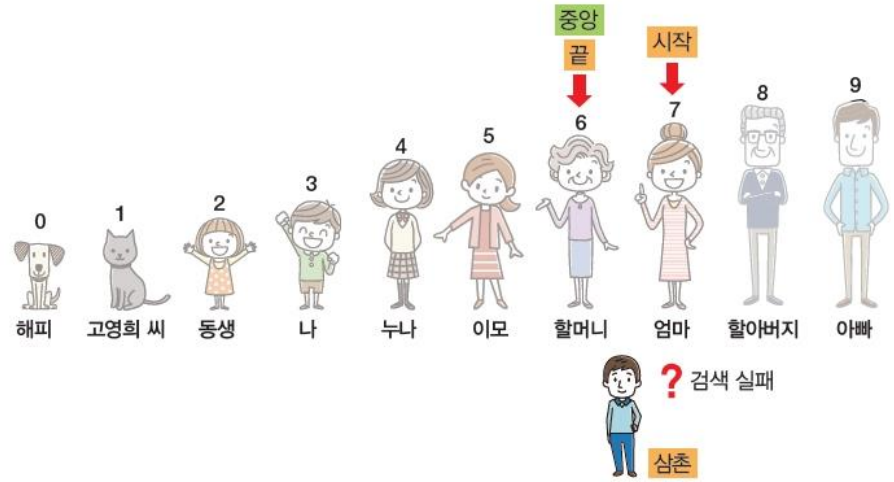


그림 13-10 이진 탐색에서 실패할 경우

Section 02 순차 검색과 이진 검색 알고리즘의 원리와 구현

- 이진 검색처럼 검색할 범위를 ½씩 반복해서 분할하는 기법을 분할 정복이라고 함

```
시작 = 0
끝 = 배열길이 - 1
while (시작 <= 끝)
    중앙 = (시작+끝) // 2
    if 찾는 값 == 중앙 :
        검색 성공. 중앙 위치 반환
    elif 찾는값 > 중앙 :
        시작 = 중앙 + 1
    else :
        끝 = 중앙 - 1

while 문에서 중앙 위치를 반환하지 못하고, 여기까지 오면 검색 실패
```

Code13-03.py 정렬된 데이터의 이진 검색

```
1  ## 클래스와 함수 선언 부분 ##
2  def binSearch(ary, fData) :
3      pos = -1
4      start = 0
5      end = len(ary) - 1
6
```

```
7       while (start <= end) :
8           mid = (start + end ) // 2
9           if fData == ary[mid] :
10              return mid
11          elif fData > ary[mid] :
12              start = mid + 1
13          else :
14              end = mid - 1
15
16      return pos
17
18  ## 전역 변수 선언 부분 ##
19  dataAry = [50, 60, 105, 120, 150, 160, 162, 168, 177, 188]
20  findData = 162    # 할머니 키
21
22  ## 메인 코드 부분 ##
23  print('배열 -->', dataAry)
24  position = binSearch(dataAry, findData)
25  if position == -1 :
26      print(findData, '(이)가 없네요.')
27  else :
28      print(findData, '(은)는 ', position, '위치에 있음')
```

실행 결과

```
배열 --> [50, 60, 105, 120, 150, 160, 162, 168, 177, 188]
162 (은)는  6 위치에 있음
```

Section 02 순차 검색과 이진 검색 알고리즘의 원리와 구현

SELF STUDY 13-2

Code13-03.py를 수정해서 랜덤하게 0과 100000 사이의 숫자 100000개를 생성한 후 0과 100000 사이의 숫자 하나를 랜덤하게 뽑아서 찾도록 코드를 작성하자. 또 몇 번 반복해서 검색에 성공하거나 실패했는지도 확인하자.

실행 결과

```
배열 일부 --> [0, 2, 2, 3, 3] ~~~ [99995, 99997, 99997, 99999, 100000]
35495 (은)는  35852 위치에 있음
## 17회 검색함

배열 일부 --> [0, 1, 4, 4, 7] ~~~ [99991, 99992, 99993, 99995, 99998]
36779 (이)가 없네요.
## 16회 검색함
```

- **색인으로 도서관 책 찾기**
  - 색인 또는 인덱스(Index)는 순서가 없는 데이터에서 특정 열(Column)만 추출하여 순서대로 정렬한 것
  - 책장의 책을 도서명으로 찾는 경우

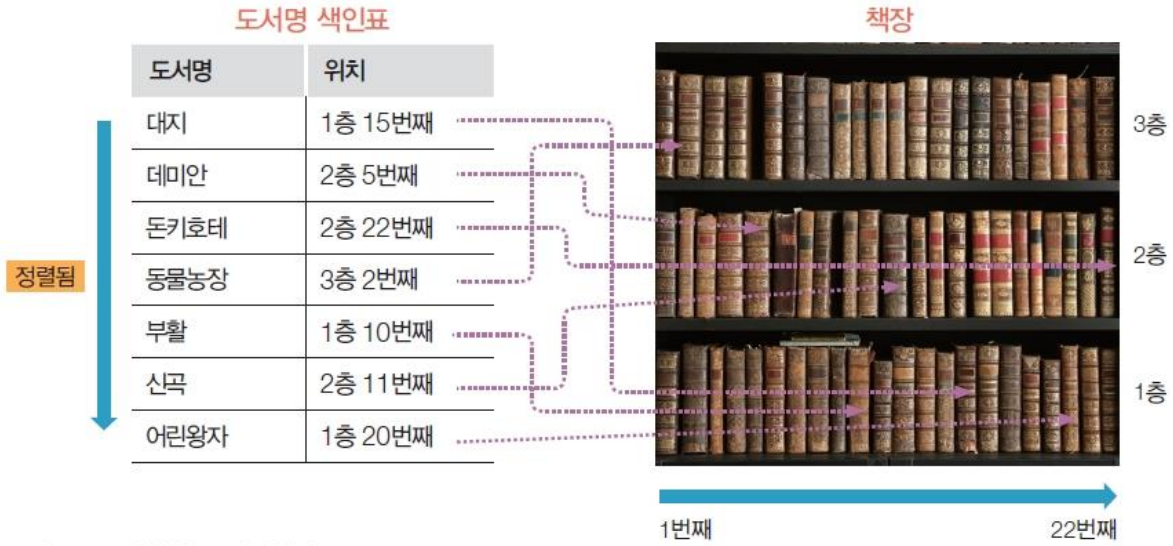


그림 13-11 책장과 도서명 색인표

  - 순서 없이 꽂혀 있는 책들을 왼쪽 [도서명 색인표]를 만들어 사용하면 필요한 책을 빠르게 찾을 수 있음

Section 03 이진 검색 알고리즘의 응용

- 도서명과 작가가 있는 책장에서 도서명으로 색인을 만드는 예

작가명 색인표

작가명	위치
단테	2층 11번째
세르반테스	2층 22번째
쌩떽쥐베리	1층 20번째
조지오웰	3층 2번째
톨스토이	1층 10번째
펄벅	1층 15번째
헤르만헤세	2층 5번째

정렬됨

그림 13-12 작가명 색인표

순번(책 위치)	도서명	작가명
0	어린왕자	쌩떽쥐베리
1	이방인	까뮈
2	부활	톨스토이
3	신곡	단테
4	돈키호테	세르반테스
5	동물농장	조지오웰
6	데미안	헤르만헤세
7	파우스트	괴테
8	대지	펄벅

그림 13-13 색인을 만들 책장 예

1 책장 배열을 정의한다.

```
책장배열 = [['어린왕자', '쌩떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'],
          ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'],
          ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]
```

Section 03 이진 검색 알고리즘의 응용

2 도서명만 추출하고, 도서명 옆에 책의 순번(=책장 위치)을 기록한다.

```
정렬전_색인표 = []
순번 = 0
for 책한권 in 책장배열 :
    정렬전_색인표.append((책한권[도서명], 순번))
    순번 += 1
```

도서명	순번
어린왕자	0
이방인	1
부활	2
신곡	3
돈키호테	4
동물농장	5
데미안	6
파우스트	7
대지	8

그림 13-14 정렬 전 색인표

3 도서명을 기준으로 색인표 배열을 정렬한다. 파이썬의 sorted( ) 함수를 사용해서 0번째 열인 도서명 열을 기준으로 정렬하면 된다.

```
정렬후_색인표 = sorted(정렬전_색인표, key=0번째 열)
```

도서명	순번
대지	8
데미안	6
돈키호테	4
동물농장	5
부활	2
신곡	3
어린왕자	0
이방인	1
파우스트	7

그림 13-15 정렬 후 색인표

4 같은 방식으로 2~3의 과정을 거쳐 [작가명 색인표]를 만든다.

Code13-04.py 정렬된 데이터의 이진 검색

```
1  from  operator import itemgetter
2
3  ## 클래스와 함수 선언 부분 ##
4  def makeIndex(ary, pos) :
5      beforeAry = []
6      index = 0
7      for data in ary :
8          beforeAry.append((data[pos], index))
9          index += 1
10
11     sortedAry = sorted(beforeAry, key=itemgetter(0))
12     return sortedAry
13
14 def bookSearch(ary, fData) :
15     pos = -1
16     start = 0
17     end = len(ary) - 1
18
19     while (start <= end) :
20         mid = (start + end ) // 2
21         if fData == ary[mid][0] :
22             return ary[mid][1]
```

```
23          elif fData > ary[mid][0] :
24              start = mid + 1
25          else :
26              end = mid - 1
27
28      return pos
29
30  ## 전역 변수 선언 부분 ##
31  bookAry = [['어린왕자', '쌩떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'],
32             ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'],
33             ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]
34  nameIndex = []
35  authIndex = []
36
37  ## 메인 코드 부분 ##
38  print('# 책장의 도서 ==>', bookAry)
39  nameIndex = makeIndex(bookAry, 0)
40  print('# 도서명 색인표 ==>', nameIndex)
41  authIndex = makeIndex(bookAry, 1)
42  print('# 작가명 색인표 ==>', authIndex)
43
44  findName = '신곡'
45  findPos = bookSearch(nameIndex, findName)
46  if findPos != -1 :
47      print(findName, '의 작가는', bookAry[findPos][1], '입니다.')
48  else :
49      print(findName, ' 책은 없습니다.')
```

```
50
51 findName = '괴테'
52 findPos = bookSearch(authIndex, findName)
53 if findPos != -1 :
54     print(findName, '의 도서는', bookAry[findPos][0], '입니다.')
55 else :
56     print(findName, ' 작가는 없습니다.')
```

실행 결과

```
# 책장의 도서 ==> [['어린왕자', '쌩떽쥐베리'], ['이방인', '까뮈'], ['부활', '톨스토이'], ['신곡', '단테'], ['돈키호테', '세르반테스'], ['동물농장', '조지오웰'], ['데미안', '헤르만헤세'], ['파우스트', '괴테'], ['대지', '펄벅']]

# 도서명 색인표 ==> [('대지', 8), ('데미안', 6), ('돈키호테', 4), ('동물농장', 5), ('부활', 2), ('신곡', 3), ('어린왕자', 0), ('이방인', 1), ('파우스트', 7)]

# 작가명 색인표 ==> [('괴테', 7), ('까뮈', 1), ('단테', 3), ('세르반테스', 4), ('쌩떽쥐베리', 0), ('조지오웰', 5), ('톨스토이', 2), ('펄벅', 8), ('헤르만헤세', 6)]

신곡 의 작가는 단테 입니다.
괴테 의 도서는 파우스트 입니다.
```

응용예제 01 편의점에서 판매된 물건 목록과 개수 세기

난이도★★☆☆☆

예제 설명

편의점에서는 매일 다양한 물건들이 판매된다. 하루에 판매되는 물건은 당연히 중복되어 여러 개가 판매된다. 마감 시간에 오늘 판매된 물건의 종류와 개수를 살펴볼 때 중복된 것은 하나만 남기도록 한다. 정렬과 이진 검색을 활용해서 처리한다.

실행 결과

```
Python
File  Edit  Shell  Debug  Options  Window  Help
========== RESTART: C:₩CookData₩Ex13-01.py ==========
#오늘 판매된 전체 물건(중복O, 정렬X) --> ['코카콜라', '츄파춥스', '도시락', '삼각김밥', '레쓰비캔커피', '츄파춥스', '도시락', '츄파춥스', '레쓰비캔커피', '코카콜라', '코카콜라', '도시락', '코카콜라', '삼각김밥', '삼각김밥', '바나나맛우유', '삼각김밥', '츄파춥스', '바나나맛우유', '코카콜라']
#오늘 판매된 전체 물건(중복O, 정렬O) --> ['도시락', '도시락', '도시락', '레쓰비캔커피', '레쓰비캔커피', '바나나맛우유', '바나나맛우유', '삼각김밥', '삼각김밥', '삼각김밥', '삼각김밥', '츄파춥스', '츄파춥스', '츄파춥스', '츄파춥스', '코카콜라', '코카콜라', '코카콜라', '코카콜라', '코카콜라']
#오늘 판매된 물품 종류(중복x) --> ['레쓰비캔커피', '츄파춥스', '바나나맛우유', '도시락', '삼각김밥', '코카콜라']

결산 결과 ==> [('레쓰비캔커피', 2), ('츄파춥스', 4), ('바나나맛우유', 2), ('도시락', 3), ('삼각김밥', 4), ('코카콜라', 5)]
>>>
Ln: 37  Col: 4
```

응용예제 02 순차 검색과 이진 검색의 성능 비교하기

난이도★★☆☆☆

예제 설명

데이터 100만 개를 랜덤하게 생성해서 정렬되지 않은 배열과 정렬된 배열을 각각 준비한다. 정렬 여부만 다르지 두 배열에는 동일한 값이 들어 있다. 정렬되지 않은 배열에는 순차 검색으로 몇 번 만에 데이터를 찾는지, 정렬된 배열에서는 이진 검색으로 몇 번 만에 데이터를 찾는지 비교한다.

실행 결과