

Step 7: Recursion and Iteration - Searching

Instructor: Eunil Park (eunilpark@skku.edu)



Last Class

1. 순서열
2. 선택정렬
3. 삽입정렬
4. 합병정렬
5. 퀵정렬
6. 버블정렬

Today's Schedule

1. Searching (검색)
2. 리스트 검색: OX문제
3. 리스트 검색: 찾은 위치 알려주기
4. 문자열 검색: 파일 입출력
5. 문자열 검색: 텍스트 파일에서 문자열 검색

Searching

- 순서열과 같은 데이터구조에 모여 있는 데이터 중에서 특정 데이터(key라고 함)를 찾는 문제
- Python에서는 표준 라이브러리에서 제공하는 순서열 공통 메소드인 “index”를 사용할 수 있어서 편리함

연산	의미
<code>s.index(x)</code>	s에서 가장 앞에 있는 키 x의 위치번호
<code>s.index(x,i)</code>	s의 i 위치에서 시작하여 가장 앞에 있는 키 x의 위치번호
<code>s.index(x,i,j)</code>	s의 i 위치와 j 위치 범위 내에서 가장 앞에 있는 키 x의 위치번호 (i 위치는 검색범위에 포함하고, j 위치는 검색범위에 포함하지 않음)

Random 샘플 생성

- Python에서는 표준 라이브러리에서 제공하는 “random” 모듈 사용

연산	의미
<code>random.sample(population, k)</code>	population 시퀀스에서 중복없이 k개를 무작위로 골라 리스트로 모아서 내준다.

- 사용 예: `random.sample(range(10000), 1000)`
 - 0 ~ 9,999사이 10,000개 정수 중에서 무작위로 1,000개를 중복없이 샘플링하여 리스트로 생성

Random 샘플 생성

- Random 샘플링 예:

```
import random
db = random.sample(range(10000), 1000)
key = db[109]
print(db.index(key))
```

- print(db.index(9999))? ➔ 리스트에 없는 데이터를 검색하면?

Traceback (most recent call last):

File "C:/Users/sw/AppData/Local/Programs/Python/Python36-32/test.py", line 5, in
<module>

```
    print(db.index(9999))
```

ValueError: 9999 is not in list

- 안전코딩 스타일 -> 값이 있는 경우에만 index 메소드를 호출!

```
if key in data:
    data.index(key)
```

검색: OX문제

- OX 검색 문제
 - “key가 list s에 있는가?”에 대답할 수 있는 함수
 - 입력: 리스트 s와 key
 - 출력: key가 s에 있으면 True, 없으면 False
- 검색 방법
 - 순차검색(sequential search)
 - 이분검색(binary search)

순차검색

- 순차검색(sequential search): 앞에서부터 차례로 하나씩 검색

s에서 key를 찾으려면

(반복조건)

s != []

- s의 선두원소 s[0]가 key와 같으면, 찾았으므로 True를 내줌
- 그렇지 않으면, s의 후미리스트 s[1:]에서 key를 재귀로 찾음

(종료조건)

s == []

- 검색 대상이 없으므로 False

```
def seq_search_ox(s, key):  
    if s != []:  
        if s[0] == key:  
            return True  
        else:  
            return seq_search_ox(s[1:], key)  
    else:  
        return False
```

- seq_search_ox([3, 5, 4, 2], 4)
→ seq_search_ox([5, 4, 2], 4)
→ seq_search_ox([4, 2], 4)
→ True
- seq_search_ox([3, 5, 4], 6)
→ seq_search_ox([5, 4], 6)
→ seq_search_ox([4], 6)
→ seq_search_ox([], 6)
→ False

순차검색

- 반복문 기반 코드

```
def seq_search_ox(s, key):  
    while s != []:  
        if s[0] == key:  
            return True  
        else:  
            s = s[1:]  
    return False
```

```
def seq_search_ox(s, key):  
    for x in s:  
        if x == key:  
            return True  
    return False
```

이분검색

- 순차검색 알고리즘의 단점
 - 키가 맨 뒤에 있거나 없는 경우 리스트 전체를 다 검색해야 함
- 이분검색(binary search)
 - 정렬된 리스트를 반으로 나누어 검색

ss에서 key를 찾으려면

(반복조건)

ss != []

- ss의 정가운데 원소의 위치번호를 mid라고 함
- key가 ss[mid]와 같으면, 찾았으므로 True를 내줌
- key가 ss[mid]보다 작으면, ss[:mid]에서 key를 재귀로 찾음
- 그렇지 않으면, ss[mid+1:]에서 key를 재귀로 찾음

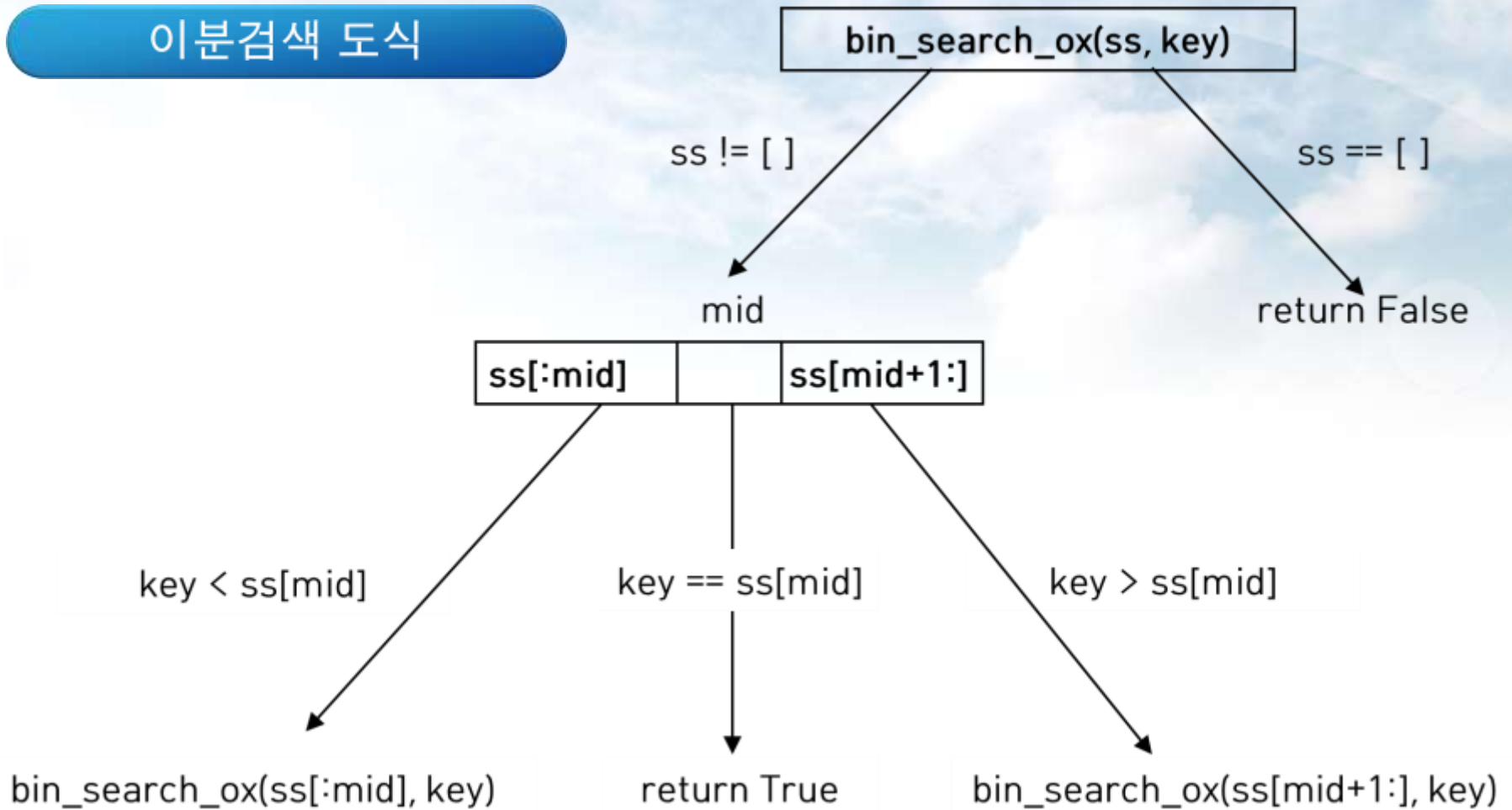
(종료조건)

ss == []

- 검색 대상이 없으므로 False

02. 리스트 검색: OX문제

이분검색 도식



이분검색 코드

```
def bin_search_ox(ss, key):  
    if ss != []:  
        mid = len(ss) // 2  
        if key == ss[mid]:  
            return True  
        elif key < ss[mid]:  
            return bin_search_ox(ss[:mid], key)  
        else:  
            return bin_search_ox(ss[mid+1:], key)  
    else:  
        return False
```

[재귀]

```
def bin_search_ox(ss, key):  
    while ss != []:  
        mid = len(ss) // 2  
        if key == ss[mid]:  
            return True  
        elif key < ss[mid]:  
            ss = ss[:mid]  
        else:  
            ss = ss[mid+1:]  
    return False
```

[while 반복문]

성능비교

- 대상 리스트의 크기가 커짐에 따라 비교회수의 차이가 커짐
- 이분 검색의 경우 리스트의 크기가 커져도 검색의 비교회수가 그리 많아지지 않음
 - 데이터를 정렬해 두면 좋은 이유!

리스트 길이	선형검색의 비교횟수	이분검색의 비교횟수
128	128	8
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

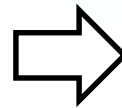
찾은 위치 알려주기 문제

- “list s에 key가 처음 나타나는 위치 번호는?”에 대답할 수 있는 함수
 - 입력: 리스트 s와 key
 - 출력: key가 s가 처음 나타나는 위치번호, key가 s에 없으면 None
- 검색 방법
 - 순차검색(sequential search)
 - 이분검색(binary search)

순차검색

- 코드

```
def seq_search_ox(s, key):  
    for x in s:  
        if x == key:  
            return True  
    return False
```



```
def seq_search(s, key):  
    i = 0  
    for x in s:  
        if x == key:  
            return i  
        i += 1  
    return None
```

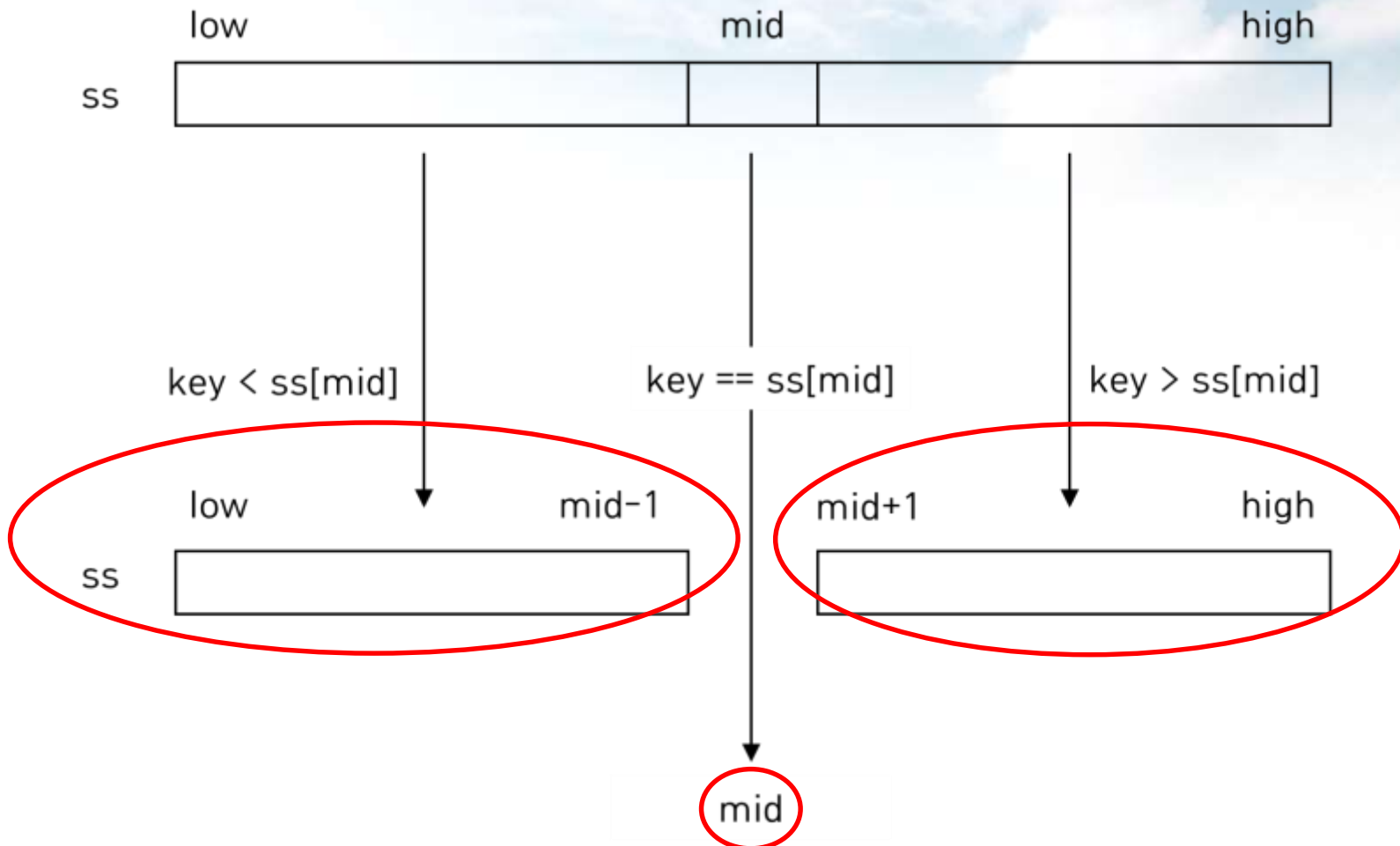

이분검색

- 위치번호를 내어줄 때
bin_search_ox 함수처럼
리스트를 자르면 오리지널
리스트의 위치번호를 잃어버림
- 리스트를 자르는 대신,
검색범위의 시작과 끝을
알려주는 위치번호는 각각 low와
high 변수에 기억하여
검색범위를 좁혀가며 검색

```
1 def bin_search(ss, key):  
2     low = 0  
3     high = len(ss) - 1  
4     while low <= high:  
5         mid = (high + low) // 2  
6         if key == ss[mid]:  
7             return mid  
8         elif key < ss[mid]:  
9             high = mid - 1  
10        else:  
11            low = mid + 1  
12        return None
```

이분검색

- 도식화



이분검색

- 코드

```
1 def bin_search(ss, key):
2     low = 0
3     high = len(ss) - 1
4     while low <= high:
5         mid = (high + low) // 2
6         if key == ss[mid]:
7             return mid
8         elif key < ss[mid]:
9             high = mid - 1
10        else:
11            low = mid + 1
12    return None
```

초기값

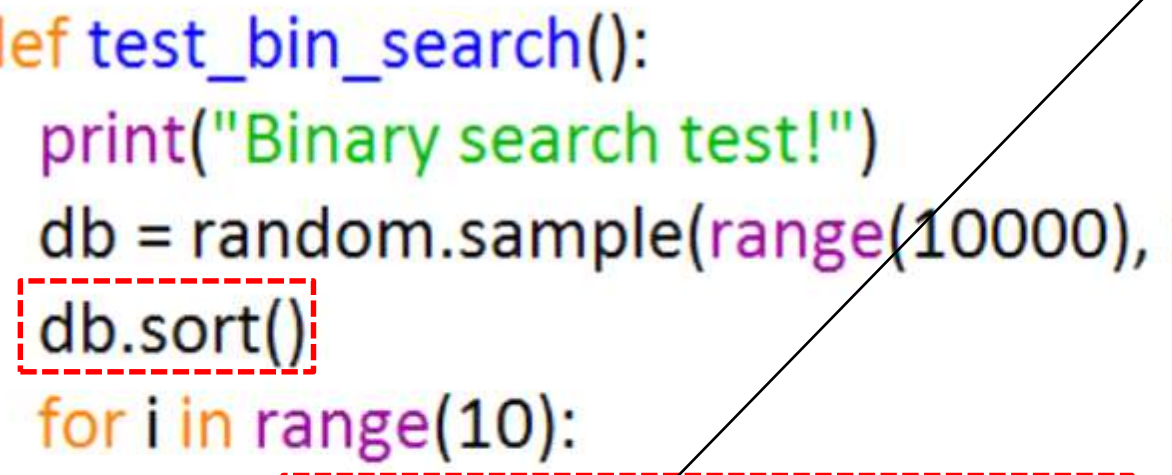
반복조건

이분검색

- 테스트 코드

연산	의미
<code>random.randrange(n)</code>	<code>range(n)</code> 에 있는 정수범위 내에서 무작위로 정수 하나 골라서 내준다.

```
def test_bin_search():  
    print("Binary search test!")  
    db = random.sample(range(10000), 1000)  
    db.sort()  
    for i in range(10):  
        key = random.randrange(10000)  
        index = bin_search(db, key)  
        print(key, "found at", index)
```



파일 입출력

- 표준입출력
 - 키보드와 실행창을 통한 상호작용
 - 오래 보존해야 하는 정보는 “파일”을 매개체로 저장/재사용
- 텍스트파일
 - 키보드로 입력 가능한 ASCII 문자로만 구성된 파일
 - 특정 플랫폼을 가리지 않고, 어디서나 사용하기 쉽고, 편집기로 쉽게 작성할 수 있음
 - 정보를 오래 저장하고 재사용할 때 요긴함

파일 열기와 닫기

- 파일을 읽거나 쓰려면 먼저 파일을 열어야 함
- 파일 열기: open 함수

```
t = open("input.txt", "r")
```

파일 변수

파일 이름

접근 모드

- 파일 닫기: close 함수

```
t.close()
```

← 파일 꼭 닫는 습관 필요!

파일 단위로 문자열읽기

- input.txt

새 나라의 대학생은 일찍 일어납니다
잠꾸러기 없는 나라
우리나라 좋은 나라

- 파일에서 n개 문자 읽기 함수: read(n)

```
t = open("input.txt", "r")  
print(t.read(1))  
print(t.read(9))  
t.close()
```

“새”

“ 나라의 대학생은”

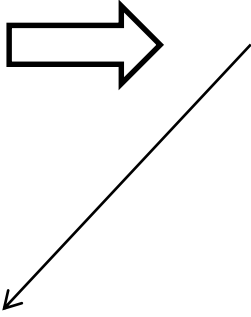
```
t = open("input.txt", "r")  
print(t.read())  
t.close()
```

➔ 파일 전체를 끝까지 다 읽음

줄 단위로 문자열읽기

- `readline(n)`: 현재 줄에서 `n`개 문자 읽어옴
- `readline()`: 한줄을 다 읽음 → 많이 사용되는 함수

```
t = open("input.txt", "r")  
print(t.readline())  
print(t.readline())  
print(t.readline())  
t.close()
```



새 나라의 대학생은 일찍 일어납니다
잠꾸러기 없는 나라
우리나라 좋은 나라

빈칸: 파일의 각 줄 끝부분에 줄바꿈문자(“\n”)가 있기 때문

문자열 쓰기

- 텍스트 파일에 쓸 때도 먼저 파일을 열어야 함

```
t = open("output.txt", "w")
```

- 문자열 쓰기 위해서 write() 함수를 호출

```
t.write("David Beckham is an English former professional footballer.\n")
```

- 줄을 바꾸기 위해서는 "\n" 명시해야 함

```
t = open("output.txt", "w")  
t.write("David Beckham is an English former professional footballer.\n")  
t.write("He played for six pro soccer clubs.\n")  
t.write("He also played the England national team.")  
t.close()
```

```
t = open("output.txt", "w")  
t.write("David Beckham is an English former professional footballer.\nHe played for six pro soccer clubs.\nHe also played the England national team.")  
t.close()
```

파일 메소드 요약

메소드	의미
close()	파일을 닫는다. 일단 닫힌 파일은 다시 열기 전에는 읽거나 쓸 수 없다.
read(n)	파일에서 문자 n개를 읽어서 문자열로 내준다.
read()	파일의 현재 위치에서 그 파일의 맨끝까지 문자를 모두 내준다.
readline(n)	파일의 현재 위치에서 그 줄의 문자 n개를 읽어서 문자열로 내준다.
readline	파일의 현재 위치에서 그 줄의 맨끝까지 문자를 모두 내준다.
readlines()	파일을 줄 별로 모두 읽어서 줄의 리스트로 내준다.
write(s)	문자열 s를 파일에 쓴다.
writelines(ss)	문자열 리스트 ss에 있는 문자열을 모두 파일에 쓴다.

문자열 메소드

연산	의미
<code>str.find(sub)</code>	str에서 맨 앞에 나오는 sub의 위치번호를 내줌, 없으면 -1을 내줌
<code>str.index(sub)</code>	str에서 맨 앞에 나오는 sub의 위치번호를 내줌, 없으면 ValueError 오류
<code>str.rfind(sub)</code>	str에서 맨 뒤에 나오는 sub의 위치번호를 내줌, 없으면 -1을 내줌
<code>str.startswith(prefix)</code>	str이 prefix로 시작하면 True 를, 그렇지 않으면 False 를 내줌
<code>str.endswith(suffix)</code>	str이 suffix로 끝나면 True 를, 그렇지 않으면 False 를 내줌

```
>>> sentence = "David Beckham is one of the English famous soccer player."
>>> print(sentence.find("is"))
>>> print(sentence.index("two"))
>>> print(sentence.rfind("Beckham"))
>>> print(str.startswith("david"))
>>> print(sentence.endswith(".reyalp"))
```

문자열 검색범위 지정

```
sentence = "David Beckham is one of the English famous soccer player."  
1 print(sentence.find("a"))           #sentence에서 "a"가 처음 나오는 위치번호  
2 print(sentence.find("a",6))         #sentence[6: ]에서 "a"가 처음 나오는 위치번호  
3 print(sentence.find("e",10, 50))    #sentence[10: 50]에서 "e"가 처음 나오는 위치번호
```

1: 1
2: 11
3: 19

문제 1

- 첫째로 나타나는 문자열 하나만 찾기
- 문제 상세:
 - 파일이름 filename과 찾을 문자열 key를 받아서 처음 나타나는 위치번호를 “result.txt” 파일에 적는 프로시저 find_first 제작. 찾으려는 key가 없으면 not found를 적는다.
 - 예: 파일이름이 “article.txt”, 찾으려는 문자열이 “컴퓨터” 이면,
find_first(“article.txt”, “컴퓨터”)를 호출하고, 결과를 “result.txt”에 적는다.
- 작업 절차
 1. 읽고 쓰는 파일을 각각 연다.
 2. 읽을 파일 전체를 문자열로 읽어온다.
 3. find 메소드를 사용, key가 있는 위치를 찾는다.
 4. 쓰는 파일에 위치번호를 쓴다.
 5. 연 파일을 모두 닫는다.

문제 1: 코드

```
1 def find_first(filename, key):
2     infile = open(filename, "r")
3     outfile = open("result.txt", "w")
4     text = infile.read()
5     pos = text.find(key)
6     if pos == -1:
7         outfile.write(key + "is not found.\n")
8     else:
9         outfile.write(key + "is at " + str(pos) + ".\n")
10    outfile.close()
11    infile.close()
12    print("done!")
```


문제 2

- 둘째로 나타나는 문자열 하나만 찾기
- 문제 상세:
 - 파일이름 filename과 찾을 문자열 key를 받아서 **두번째** 나타나는 위치번호를 “result.txt” 파일에 적는 프로시저 find_second 제작. 찾으려는 key가 없거나 한번만 나타나면 not found를 적는다.
 - 예: 파일이름이 “article.txt”, 찾으려는 문자열이 “컴퓨터” 이면,
find_second(“article.txt”, “컴퓨터”)를 호출하고, 결과를 “result.txt”에 적는다.
- 작업 절차
 1. 읽고 쓰는 파일을 각각 연다.
 2. 읽을 파일 전체를 문자열로 읽어온다.
 3. find 메소드를 사용, key가 있는 위치를 찾는다.
 4. 찾은 위치 바로 다음 번호부터 시작, **find** 메소드로 **key**의 위치를 한번 더 찾는다.
 5. 쓰는 파일에 위치번호를 쓴다.
 6. 연 파일을 모두 닫는다.

문제 2: 코드

```
def find_second(filename, key):  
    infile = open(filename, "r")  
    outfile = open("result.txt", "w")  
    text = infile.read()  
    pos = text.find(key)  
  
    if pos == -1:  
        outfile.write(key + "is not found.\n")  
    else:  
        outfile.write(key + "is at " + str(pos) + " the 2nd time.\n")  
    outfile.close()  
    infile.close()  
    print("done!")
```

Summary

1. Searching (검색)
2. 리스트 검색: OX문제
3. 리스트 검색: 찾은 위치 알려주기
4. 문자열 검색: 파일 입출력
5. 문자열 검색: 텍스트 파일에서 문자열 검색

Thanks

Step 7: Recursion and Iteration - Searching
Instructor: Eunil Park (eunilpark@skku.edu)

