[CSE1017]
프로그래밍기초

Recursion and Iteration : Searching

#06. 재귀와 반복 : 검색

김현하

한양대학교 ERICA 소프트웨어학부

2024년도 1학기

python™

목차

순차검색

순차검색

- "키 x가 리스트 s에 있는가?"
  - 리스트 s 에서 검색할 키 x의 존재 유무를 확인하는 문제
  - x가 있을 경우 `True`, 없으면 `False` 로 대답
- 순차검색
  - 리스트의 맨 앞에서 시작하여 순서대로 검사

존재 유무 확인

- 문제의 입출력 요구사항
  - 입력(파라미터) : 키의 리스트 s, 검색할 키 x
  - 출력(리턴) : x가 s에 있으면 `True`, 없으면 `False` 로 대답
- 순차검색sequential search : 리스트의 맨 앞에서 시작하여 순서대로 검사

리스트 s에서 키 x를 검색하려면,

(반복 조건) `s != []`

  s의 선두원소 `s[0]`이 x와 같으면, 찾았으므로 `True`를 리턴 한다.

  그렇지 않으면, s의 후미 리스트 `s[1:]`에서 키 x를 재귀로 검색한다.

(종료 조건) `s == []`

  검색 대상이 없으므로 `False`를 리턴한다.

존재 유무 확인

리스트 s에서 키 x를 검색하려면,

(반복 조건) s != []

s의 선두원소 s[0]이 x와 같으면, 찾았으므로 True를 리턴 한다.

그렇지 않으면, s의 후미 리스트 s[1:]에서 키 x를 재귀로 검색한다.

(종료 조건) s == []

검색 대상이 없으므로 False를 리턴한다.

```
1  def seq_search_OX(s, x):
2      if s != []:
3          if s[0] == x:
4              return True
5          else:
6              return seq_search_OX(s[1:], x)
7      else:
8          return False
```

존재 유무 확인

```
1  def seq_search_OX(s, x):
2      if s != []:
3          if s[0] == x:
4              return True
5          else:
6              return seq_search_OX(s[1:], x)
7      else:
8          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([3,5,4,2], 6)
⇒ seq_search_OX([5,4,2], 6)
⇒ seq_search_OX([4,2], 6)
⇒ seq_search_OX([2], 6)
⇒ seq_search_OX([], 6)
⇒ False
```

- while 루프 버전
- for 루프 버전

존재 유무 확인

while 루프 버전

```
1  def seq_search_OX(s, x):
2      while s != []:
3          if s[0] == x:
4              return True
5          else:
6              s = s[1:]
7      return False
```

for 루프 버전

```
1  def seq_search_OX(s, x):
2      for key in s:
3          if key == x:
4              return True
5      return False
```

존재 유무 확인

```
1  def seq_search_OX(s, x):
2      if s != []:
3          if s[0] == x:
4              return True
5          else:
6              return seq_search_OX(s[1:], x)
7      else:
8          return False
```

논리 연산자로 흐름 제어

```
1  def seq_search_OX(s, x):
2      return s != [] and (s[0] == x or seq_search_OX(s[1:], x))
```

위치 인덱스 찾기

- 문제의 입출력 요구사항
  - 입력(파라미터) : 키의 리스트 s, 검색할 키 x
  - 출력(리턴) : x가 s에 있으면 x의 위치 인덱스, 없으면 None
- 순차검색sequential search : 리스트의 맨 앞에서 시작하여 순서대로 검사

위치 인덱스 찾기

```
1  def seq_search_OX(s, x):
2      if s != []:
3          if s[0] == x:
4              return True
5          else:
6              return seq_search_OX(s[1:], x)
7      else:
8          return False
```

리스트를 잘라내기 때문에
인덱스 정보를 잃어버림

```
1  def seq_search_OX(s, x):
2      def loop(s, i):
3          if s != []:
4              if s[0] == x:
5                  return i
6              else:
7                  return loop(s[1:], i + 1)
8          else:
9              return None
10     return loop(s, 0)
```

인덱스 정보를 추가

위치 인덱스 찾기

```
1  def seq_search_OX(s,x):
2      def loop(s, i):
3          if s != []:
4              if s[0] == x:
5                  return i
6              else:
7                  return loop(s[1:], i + 1)
8          else:
9              return None
10     return loop(s, 0)
```

```
1  def seq_search(s,x):
2      def loop(i):
3          if i < len(s):
4              if s[i] == x:
5                  return i
6              else:
7                  return loop(i + 1)
8          else:
9              return None
10     return loop(0)
```

```
seq_search([3,5,4,2], 4)
⇒ loop(0)
⇒ loop(1)
⇒ loop(2)
⇒ 2
```

```
seq_search([3,5,4,2], 6)
⇒ loop(0)
⇒ loop(1)
⇒ loop(2)
⇒ loop(3)
⇒ loop(4)
⇒ None
```

- while 루프 버전
- for 루프 버전

이분검색

이분검색

- "키 x가 리스트 s에 있는가?"
  - 리스트 s 에서 검색할 키 x의 존재 유무를 확인하는 문제
  - x가 있을 경우 `True`, 없으면 `False` 로 대답
- 이분검색
  - 순차검색 알고리즘은 키의 위치에 따라 검색성능이 갈림
  - 리스트가 정렬되어 있으면 리스트 전체를 볼 필요가 없음

존재 유무 확인

- 이분검색binary search : 정렬된 리스트를 반으로 나누어 재귀로 검색하는 알고리즘

리스트 ss에서 키 x를 검색하려면,

(반복 조건) ss != []

  ss의 가운데 원소의 인덱스를 mid로 지정하고,

  x가 ss[mid]와 같으면, 찾았으므로 True를 리턴한다.

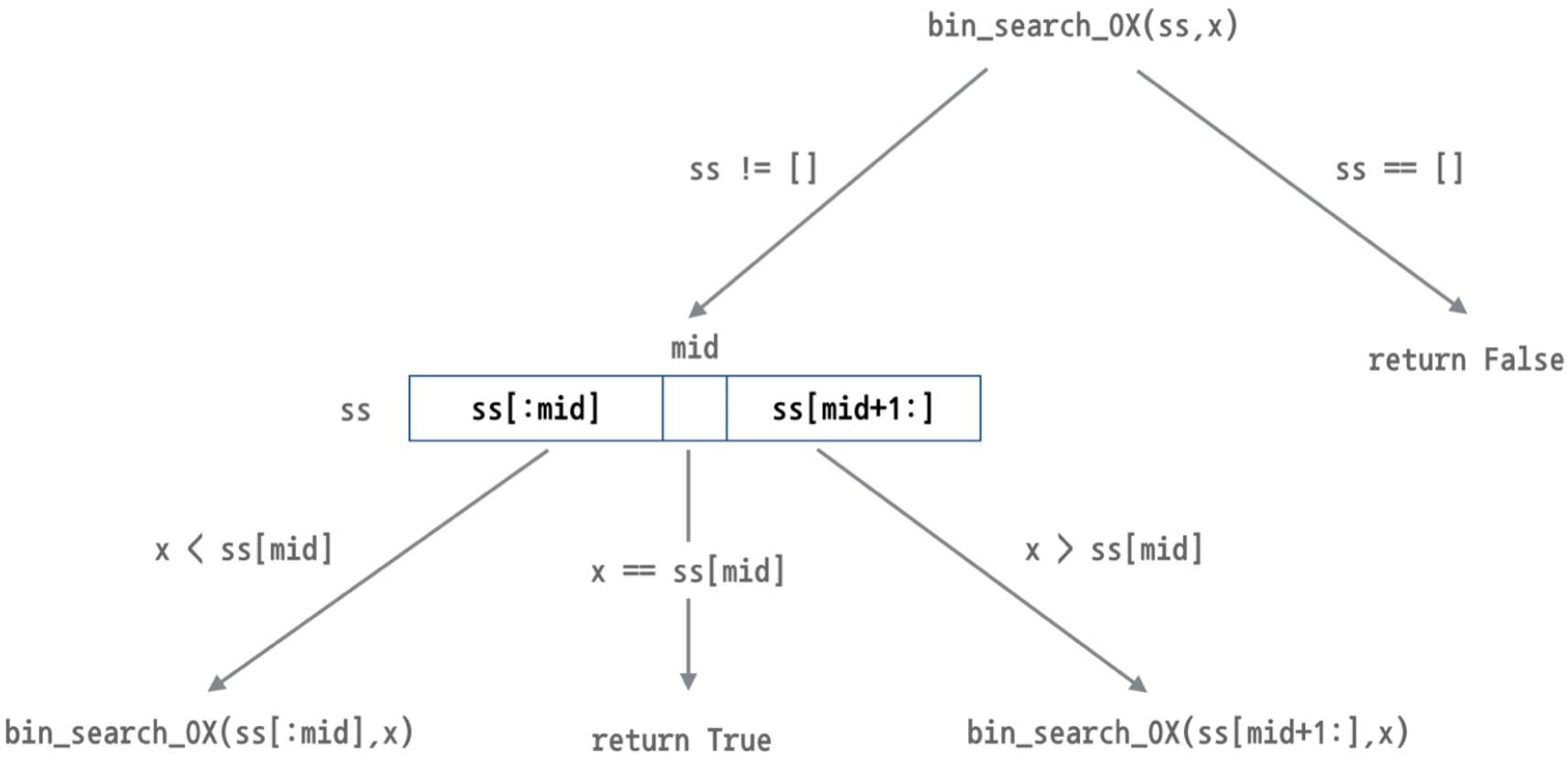
  x가 ss[mid]보다 작으면, ss[:mid]에서 x를 재귀로 검색한다.

  x가 ss[mid]보다 크면, ss[mid+1:]에서 x를 재귀로 검색한다.

(종료 조건) ss == []

  검색 대상이 없으므로 False를 리턴한다.

존재 유무 확인


bin_search_OX(ss,x)
ss != []
ss == []
mid
return False
ss
ss[:mid]
ss[mid+1:]
x < ss[mid]
x == ss[mid]
x > ss[mid]
bin_search_OX(ss[:mid],x)
return True
bin_search_OX(ss[mid+1:],x)

존재 유무 확인

리스트 ss에서 키 x를 검색하려면,

(반복 조건) ss != []

ss의 가운데 원소의 인덱스를 mid로 지정하고,

x가 ss[mid]와 같으면, 찾았으므로 True를 리턴한다.

x가 ss[mid]보다 작으면, ss[:mid]에서 x를 재귀로 검색한다.

x가 ss[mid]보다 크면, ss[mid+1:]에서 x를 재귀로 검색한다.

(종료 조건) ss == []

검색 대상이 없으므로 False를 리턴한다.

```
1   def bin_search_OX(ss, x):
2       if ss != []:
3           mid = len(ss) // 2
4           if x == ss[mid]:
5               return True
6           elif x < ss[mid]:
7               return bin_search_OX(ss[:mid], x)
8           else:
9               return bin_search_OX(ss[mid+1:], x)
10      else:
11          return False
```

존재 유무 확인

```
1   def bin_search_OX(ss, x):
2       if ss != []:
3           mid = len(ss) // 2
4           if x == ss[mid]:
5               return True
6           elif x < ss[mid]:
7               return bin_search_OX(ss[:mid], x)
8           else:
9               return bin_search_OX(ss[mid+1:], x)
10      else:
11          return False
```

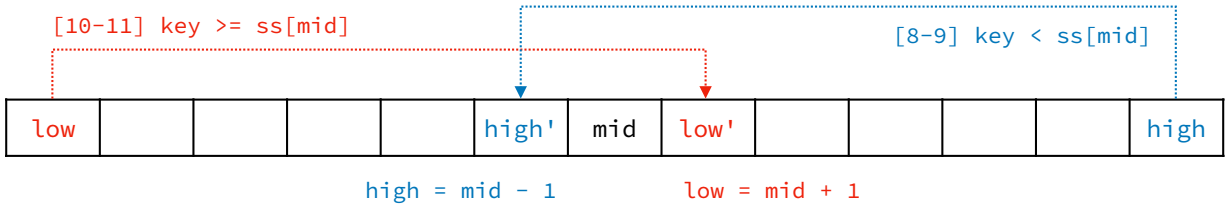
```
>>> s = [3,5,8,7,4,6,1,9,2]
>>> s.sort()
>>> print(bin_search_OX(s,5))
>>> print(bin_search_OX(s,8))
>>> print(bin_search_OX(s,1))
>>> print(bin_search_OX(s,11))
```

자습:
- 함수흐름추적
- while 루프 버전
- for 루프 버전??

위치 인덱스 찾기

- 앞에서 작성한 bin_search_OX함수는 리스트를 반으로 자르면서 오리지널 인덱스 정보를 잃어버림
- 현재 검색중인 리스트의 최소low/최대high값을 들고 다니도록 알고리즘을 수정

```
1   def bin_search(ss, key):
2       low = 0
3       high = len(ss) - 1
4       while low <= high:  # 최소 하나의 원소가 포함
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
```



성능 비교

계산 비용 분석

- 검색 알고리즘에서 최악의 경우, 검색 완료시까지 비교 횟수는?

리스트 길이	순차 검색의 비교횟수	이분검색의 비교횟수
28	28	5
1,024	1,024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

실행 시간 테스트

- 표준 라이브러리 random 모듈의 sample 함수로 검색 대상 리스트 생성

연산	의미
random.sample(population, k)	시퀀스 population에서 중복 없이 k개를 무작위로 골라 리스트로 모아서 리턴한다.

- 표준 라이브러리 random 모듈의 randrange 함수로 임의의 검색 키 생성

연산	의미
random.randrange(n)	range(n) 정수범위에서 무작위로 정수 하나를 골라서 내준다.

- 표준 라이브러리 time 모듈의 perf_counter 함수로 시간 값 획득

연산	의미
time.perf_counter()	호출 시점의 절대 시각을 가능한 최대 정밀도의 초 단위로 내준다.

- code : 6-13.py, 6-14.py

텍스트파일 처리

텍스트파일 처리

- `input` 함수로 받은 키보드 입력이나 `print`로 출력한 결과는 프로그램을 종료하면 모두 사라짐
- 정보를 영구 보존하려면 파일file이라는 단위로 하드디스크와 같은 외부기억장치에 저장 필요
- 텍스트파일textfile
  - 키보드로 입력가능한 문자로만 구성한 문자열의 형태로 저장한 파일
  - 플랫폼을 가리지 않고 어디에서나 기본편집기로 읽기/쓰기 가능

파일 열기와 닫기

- 파일 열기 `t = open("input.txt", "r", encoding="UTF-8")`
  - `"input.txt"` : (경로를 포함한) 파일 이름
  - `"r"` : 접근 모드access mode, 기본값은 `"r"`

접근 모드	의미	
	파일이 있으면	파일이 없으면
`"r"`	파일에서 읽기	오류
`"w"`	기존 파일의 내용을 지우고 쓰기	새로 생성한 파일에 쓰기
`"a"`	기존 파일의 내용 뒤에 이어서 쓰기	새로 생성한 파일에 쓰기
`"r+"`	파일에서 읽고 쓰기	오류
`"w+"`	기존 파일의 내용을 지우고 읽고 쓰기	새로 생성한 파일에 읽고 쓰기
`"a+"`	기존 파일의 내용 뒤에 이어서 읽고 쓰기	새로 생성한 파일에 읽고 쓰기

  - `encoding` : 텍스트를 읽어들이는 인코딩/디코딩 코드 (기본값은 `locale.getlocale()` 혹은 `locale.getencoding()`) * 3.11 이상
- 파일 닫기 `t.close()`
  - 파일을 닫지 않으면? : 보안 문제 / 리소스 점유 등의 문제 발생

파일 읽기와 쓰기

- 파일에서 문자열 읽기
  - `t.read(n)` : 파일 t(의 현재위치)에서 첫 n개의 문자를 읽어서 문자열로 리턴
  - `t.read()` : 파일 t(의 현재위치)에서 (남은) 문자열 전체를 읽어서 문자열로 리턴
  - `t.readline()` : 파일 t(의 현재위치)에서 한줄 단위로 읽어서 문자열로 리턴
  - `t.readlines()` : 파일 t에서 (남은) 문자열 전체를 줄 단위로 읽은 후 문자열의 리스트로 리턴
- 파일에 문자열 쓰기
  - 파일을 열 때 쓰기권한이 있어야 함
    - `t = open("output.txt", "w")`
  - `t.write(s)` : 파일 t(의 현재위치)에 문자열 s를 출력
  - `t.writelines(ss)` : 파일 t(의 현재위치)에 문자열의 리스트 ss의 내용을 줄 단위로 구분해서 출력

문자열 검색

문자열 메소드

- 텍스트 파일 = 문자열 덩어리
- 문자열 메소드

메소드 호출	의미
str.find(*sub*)	str에서 맨 앞에 나타나는 sub의 시작 인덱스를 리턴, 없으면 -1을 리턴
str.index(*sub*)	str에서 맨 앞에 나타나는 sub의 시작 인덱스를 리턴, 없으면 ValueError 오류 발생
str.rfind(*sub*)	str에서 맨 뒤에 나타나는 sub의 시작 인덱스를 리턴, 없으면 -1을 리턴
str.startswith(*prefix*)	str이 prefix로 시작하면 True리턴, 그렇지 않으면 False를 리턴
str.endswith(*suffix*)	str이 suffix로 끝나면 True리턴, 그렇지 않으면 False를 리턴

처음 나타나는 문자열 하나만 찾기

- 파일이름이 filename과 찾으려는 문자열 x를 인수로 받음
  - 텍스트파일 filename에서 문자열 x가 처음 나타나는 위치 인덱스 정보를 "result.txt" 파일에 씀
  - 문자열 x가 존재하지 않을 경우, 찾지 못했다는 정보를 "result.txt" 파일에 씀
- article.txt 파일을 다운받아 아래 함수를 테스트

```
1   def find_1st(filename, x):
2       infile = open(filename, "r")
3       outfile = open("result.txt", "w")
4       text = infile.read()
5       position = text.find(x)
6       if position == -1:
7           outfile.write(x + " is not found.\n")
8       else:
9           outfile.write(x + " is at " + str(position) + ".\n")
10      outfile.close()
11      infile.close()
12      print("Done")
```

둘째로 나타나는 문자열 하나만 찾기

- 파일이름이 filename과 찾으려는 문자열 x를 인수로 받음
  - 텍스트파일 filename에서 문자열 x가 둘째로 나타나는 위치 인덱스 정보를 "result.txt" 파일에 씀
  - 문자열 x가 존재하지 않을 경우, 찾지 못했다는 정보를 "result.txt" 파일에 씀
- article.txt 파일을 다운받아 아래 함수를 테스트

```
1   def find_2nd(filename, x):
2       infile = open(filename, "r")
3       outfile = open("result.txt", "w")
4       text = infile.read()
5       position = text.find(x)
6       position = text.find(x, position + 1)
7       if position == -1:
8           outfile.write(x + " is not found.\n")
9       else:
10          outfile.write(x + " is at " + str(position) + " the 2nd time.\n")
11      outfile.close()
12      infile.close()
13      print("Done")
```