

Data Structure

자료구조



탐색과 맵 구조



한국기술교육대학교
온라인평생교육원

학습내용

- 탐색과 맵 구조의 개념
- 순차 탐색
- 이진 탐색과 보간 탐색
- 맵의 응용: 영어 단어장

학습목표

- 탐색의 개념과 맵 구조를 설명할 수 있다.
- 순차 탐색의 동작 원리에 대해 설명할 수 있다.
- 이진 탐색과 보간 탐색에 대해 설명할 수 있다.
- 맵 구조를 구현할 수 있다.

탐색과 맵 구조의 개념



탐색과 맵 구조

탐색과 맵 구조의 개념

1 탐색의 개념

탐색

- 테이블에서 원하는 **탐색키**를 가진 레코드를 찾는 작업
- 테이블: 레코드의 집합
- 탐색키: 레코드를 서로 구별하여 인식할 수 있는 키
- 테이블을 구성하는 방법에 따라 탐색 효율이 달라짐
- ◎ 탐색을 위한 다양한 자료구조가 있음

탐색과 맵 구조

탐색과 맵 구조의 개념

1 탐색의 개념

◎ 맵(Map), 딕셔너리(Dictionary)

- ▶ 탐색을 위한 자료구조
- ▶ **엔트리(entry)**, 또는 키를 가진 레코드(Keyed record)의 집합

키(Key)

영어 단어와 같은 레코드를
구분할 수 있는 탐색키

값(Value)

단어의 의미와 같이
탐색키와 관련된 값





탐색과 맵 구조의 개념

탐색과 맵 구조

탐색과 맵 구조의 개념

1 탐색의 개념

◎ 파이썬의 딕셔너리

딕셔너리(Dictionary)

- 파이썬에서 제공하는 자료구조 맵

(예) 색 이름에 대한 RGB 값

```

colorTable = { 'red' : (255,0,0), 'green' : (0,255,0),
               'blue' : (0,0,255), 'yellow' : (255,255,0), ...}
    
```

keys
↓
colorTable = { 'red' : (255,0,0), 'green' : (0,255,0),
↑
values
↑
'blue' : (0,0,255), 'yellow' : (255,255,0), ... }
↑
엔트리(keyed record)

탐색과 맵 구조

탐색과 맵 구조의 개념

1 탐색의 개념

◎ 파이썬의 딕셔너리

딕셔너리(Dictionary)

- {}: 집합과 혼동하지 말아야 함.

- colorTable = {} → 공백 딕셔너리
- colorName = set() → 공백 집합

탐색과 맵 구조의 개념



탐색과 맵 구조

탐색과 맵 구조의 개념

2 맵의 추상 자료형

데이터	연산
키를 가진 레코드(엔트리)의 집합	<ul style="list-style-type: none"> ▪ Map (): 비어 있는 새로운 맵을 만듦 ▪ search(key): 탐색키 key를 가진 레코드를 찾아 반환 ▪ insert(entry): 주어진 entry를 맵에 삽입 ▪ delete(key): 탐색키 key를 가진 레코드를 찾아 삭제 ▪ size(): 모든 엔트리의 개수를 반환 ▪ clear(): 맵을 공백상태로 만듦 ▪ display(): 맵을 화면에 보기 좋게 출력

탐색과 맵 구조

탐색과 맵 구조의 개념

3 순환과 반복

● 맵을 구현하는 방법들

▶ 테이블 구성 방법에 따라 알고리즘과 성능이 달라짐.

엔트리를 리스트에 저장하는 방법	엔트리를 이진 트리에 저장하는 방법	해싱을 이용하는 방법
<ul style="list-style-type: none"> ▪ 정렬되지 않은 리스트: 순차 탐색 맵 ▪ 정렬된 리스트: 이진 탐색 맵 	<ul style="list-style-type: none"> ▪ 이진 탐색 트리를 이용한 맵 	<ul style="list-style-type: none"> ▪ 해시 맵

순차 탐색



탐색과 맵 구조

순차 탐색

1 순차 탐색이란?

순차 탐색(Sequential Search)
<ul style="list-style-type: none"> 가장 단순하고 직접적인 탐색 방법 정렬되지 않은 배열을 처음부터 마지막까지 하나씩 검사

조건
<ul style="list-style-type: none"> 테이블에 대한 특별한 조건 없음 어떤 테이블에 대해서도 탐색 가능



탐색과 맵 구조

순차 탐색

1 순차 탐색이란?

◎ 순차 탐색의 예

8을 찾는 경우

9	5	8	3	7
---	---	---	---	---

$9 \neq 8$ 탐색 계속

9	5	8	3	7
---	---	---	---	---

$5 \neq 8$ 탐색 계속

9	5	8	3	7
---	---	---	---	---

$8 = 8$ 탐색 성공

탐색성공 → 종료

2를 찾는 경우

9	5	8	3	7
---	---	---	---	---

$9 \neq 2$ 탐색 계속

9	5	8	3	7
---	---	---	---	---

$5 \neq 2$ 탐색 계속

9	5	8	3	7
---	---	---	---	---

$8 \neq 2$ 탐색 계속

9	5	8	3	7
---	---	---	---	---

$3 \neq 2$ 탐색 계속

9	5	8	3	7
---	---	---	---	---

$7 \neq 2$ 탐색 계속

더 이상 항목이 없음 → 탐색실패



순차 탐색

탐색과 맵 구조

▶ 순차 탐색

2 순차 탐색 알고리즘

순차 탐색 알고리즘

- 탐색 성공 → 인덱스 반환
- 탐색 실패 → -1 반환
- 탐색을 위한 시작과 끝 인덱스를 제공

```
def sequential_search(A, key, low, high) :
    for i in range(low, high+1) :
        if A[i] == key : return i    # 탐색에 성공하면 키 값의 인덱스 반환
    return -1                      # 탐색에 실패하면 -1 반환
```

탐색과 맵 구조

▶ 순차 탐색

2 순차 탐색 알고리즘

◎ 순차 탐색 시간 복잡도

시간 복잡도

- 최선의 경우: 1번 비교
- 최악의 경우: n번 비교
- 평균 비교 횟수: $(n+1)/2$ 번 비교
- $O(n)$

이진 탐색과 보간 탐색



탐색과 맵 구조

이진 탐색과 보간 탐색

1 이진 탐색이란?

이진 탐색(Binary Search)

▪ 배열의 **중앙에 있는 값**을 조사하여 찾고자 하는 항목이 왼쪽 또는 오른쪽 부분 배열에 있는지를 알아내어 **탐색의 범위를 반으로 줄여가며** 탐색 진행

예 사전에서 단어 찾기

예 10억 명 중에서 특정한 이름 탐색

◎ 순차 탐색: 평균 5억 번의 비교 필요

◎ 이진 탐색: 단지 30번의 비교 필요

조건

테이블에서 엔트리들이 키값에 따라 정렬되어 있어야 함



탐색과 맵 구조

이진 탐색과 보간 탐색

1 이진 탐색이란?

○ 이진 탐색의 예



이진 탐색과 보간 탐색



탐색과 맵 구조

이진 탐색과 보간 탐색

2 이진 탐색 알고리즘

● 순환구조를 이용한 이진 탐색 알고리즘

```
def binary_search(A, key, low, high) :
    if (low <= high) :          # 하나 이상의 항목이 있어야 탐색
        middle = (low + high) // 2
        if (key == A[middle]) :   # 탐색 성공
            return middle
        elif (key < A[middle]) :  # 왼쪽 부분리스트 탐색
            return binary_search(A, key, low, middle - 1)
        else :                   # 오른쪽 부분리스트 탐색
            return binary_search(A, key, middle + 1, high)
    return -1                  # 탐색 실패
```

탐색과 맵 구조

이진 탐색과 보간 탐색

2 이진 탐색 알고리즘

● 반복을 이용한 이진 탐색 알고리즘

```
def binary_search_iter(A, key, low, high) :
    while (low <= high) :          # 항목들이 남아 있으면(종료 조건)
        middle = (low + high) // 2
        if key == A[middle]:       # 탐색 성공
            return middle
        elif (key > A[middle]):   # key가 middle의 값보다 크면
            low = middle + 1      # middle+1 ~ high 사이 검색
        else:                      # key가 middle의 값보다 작으면
            high = middle - 1     # low ~ middle-1 사이 검색
    return -1                  # 탐색 실패
```

이진 탐색과 보간 탐색



탐색과 맵 구조

이진 탐색과 보간 탐색

2 이진 탐색 알고리즘

● 이진 탐색 복잡도 분석

시간 복잡도

- 최선의 경우: 1번 비교
예) $n/2$ 위치에 찾는 항목이 있을 때
- 최악의 경우: $O(\log n)$
예) 찾는 항목이 리스트에 없을 때



탐색과 맵 구조

이진 탐색과 보간 탐색

2 이진 탐색 알고리즘

● 이진 탐색의 특징

- ▶ 매우 효율적인 탐색 가능
- ▶ 리스트가 반드시 정렬되어 있어야 함
- ▶ 데이터의 삽입과 삭제가 빈번한 응용에는 적합하지 않음

- 항목 하나의 삽입/삭제 연산이 $O(n)$ 이 걸림
- 이진 탐색 트리 등이 더 유리



이진 탐색과 보간 탐색



탐색과 맵 구조

이진 탐색과 보간 탐색

3 보간 탐색

보간 탐색(Interpolation Search)

- 이진 탐색의 개선 알고리즘
 - 탐색키가 존재 할 위치를 예측하여 탐색
 - 리스트를 불균등하게 분할하여 탐색
 - 탐색 값과 위치는 비례한다는 가정
- 예) 사전이나 전화번호부를 탐색할 때
- ‘ㅎ’으로 시작하는 단어는 사전의 뒷부분에서 찾음
 - ‘ㄱ’으로 시작하는 단어는 앞부분에서 찾음

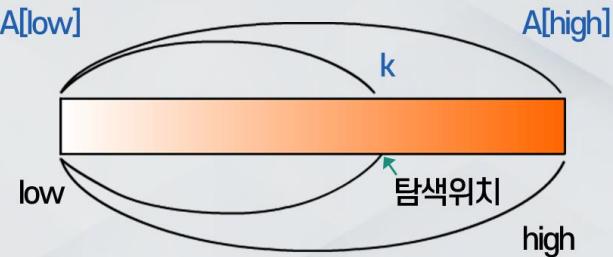
탐색과 맵 구조

이진 탐색과 보간 탐색

3 보간 탐색

◎ 탐색 값과 위치가 비례한다는 가정

▶ 가정



▶ 탐색 위치 계산

$$\text{탐색위치} = \text{low} + (\text{high} - \text{low}) * \frac{k - A[\text{low}]}{A[\text{high}] - A[\text{low}]}$$

이진 탐색과 보간 탐색



탐색과 맵 구조

이진 탐색과 보간 탐색

3 보간 탐색

● 보간 탐색 알고리즘

▶ 반복을 이용한 보간 탐색 알고리즘

```
def interpolation_search(A, key, low, high) :  
    if (low <= high) : # 하나 이상의 항목이 있어야 탐색  
        middle = int(low + (high-low)*(key-A[low])/A[high]-A[low])  
        if (key == A[middle]) : # 탐색 성공  
            return middle  
        elif (key < A[middle]) : # 왼쪽 부분리스트 탐색  
            return interpolation_search(A, key, low, middle - 1)  
        else : # 오른쪽 부분리스트 탐색  
            return interpolation_search(A, key, middle + 1, high)  
    return -1 # 탐색 실패
```



맵의 응용: 영어 단어장

탐색과 맵 구조

맵의 응용: 영어 단어장

1 맵의 응용: 나의 단어장

◎ 단어장 설계

맵 엔트리

- Key: 영어 단어
- Value: 단어의 의미

맵 구현: 이진 탐색 이용

- 엔트리는 리스트(테이블)에 저장
- 테이블은 반드시 key값에 따라 정렬되어 있어야 함

탐색과 맵 구조

맵의 응용: 영어 단어장

1 맵의 응용: 나의 단어장

◎ 엔트리 클래스

```
class Entry:
    def __init__( self, key, value ):
        self.key = key
        self.value = value

    def __str__( self ):
        return str("\t%s:%s"%(self.key, self.value))
```



맵의 응용: 영어 단어장

탐색과 맵 구조

맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

● 맵의 추상 자료형

데이터	연산
키를 가진 레코드(엔트리)의 집합	<ul style="list-style-type: none"> ▪ Map (): 비어 있는 새로운 맵을 만듦 ▪ search(key): 탐색키 key를 가진 레코드를 찾아 반환 ▪ insert(entry): 주어진 entry를 맵에 삽입 ▪ delete(key): 탐색키 key를 가진 레코드를 찾아 삭제 ▪ size(): 모든 엔트리의 개수를 반환 ▪ clear(): 맵을 공백상태로 만듦 ▪ display(): 맵을 화면에 보기 좋게 출력

탐색과 맵 구조

맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

● 맵 클래스

맵 클래스

```
class BinaryMap:
    def __init__( self ):
        self.table = [] # 엔트리 저장을 위한 테이블(리스트)
```

- 테이블은 반드시 정렬 상태를 유지하고 있어야 함.



맵의 응용: 영어 단어장

탐색과 맵 구조

맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

● 기본 연산

▶ 테이블 크기

```
def size( self ):
    return len(self.table)
```

▶ 화면 출력

```
def display(self, msg):
    print(msg)
    for entry in self.table :
        print(entry )
```

탐색과 맵 구조

맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

● 탐색 연산

▶ 이진 탐색 알고리즘 사용 가능

- 리스트가 정렬 상태를 유지
- 탐색을 위한 시작과 종료 인덱스 제공

```
def search(self, key) :          # 이진 탐색
    pos = binary_search(self.table, key, 0, self.size()-1)
    if pos >= 0 :                # 탐색 성공: 인덱스
        return self.table[pos]
    else :                       # 탐색 실패: -1
        return None
```

▶ 복잡도: $O(\log n)$



맵의 응용: 영어 단어장

탐색과 맵 구조

맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

● 삽입 연산

▶ 리스트가 정렬 상태를 유지하도록 삽입

- 삽입 위치를 찾은 후 그 곳에 삽입

```
def insert(self, key, value) :          # 삽입 연산
    pos = 0
    while pos < self.size() :           # 삽입할 위치를 찾아야 함
        if self.table[pos].key >= key:
            break
        pos += 1

    self.table.insert(pos, Entry(key, value)) # 엔트리 삽입
```

▶ 복잡도: $O(n)$

탐색과 맵 구조

맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

● 삭제 연산

▶ 삭제할 엔트리를 찾은 후 삭제

- 이진 탐색 사용 가능

```
def delete(self, key) :
    # 삭제할 키값을 가진 엔트리의 인덱스 찾기: 이진 탐색
    pos = binary_search(self.table, key, 0, self.size()-1)
    if pos >= 0 :                      # 엔트리가 있으면
        self.table.pop(pos)             # 리스트에서 삭제
```

▶ 복잡도: $O(n)$

맵의 응용: 영어 단어장



Code editor showing Python code for a binary map implementation.

```

# SequentialMap.
class Entry:
    def __init__( self, key, value ):
        self.key = key
        self.value = value

    def __str__( self ):
        return str("\t%s:%s"%(self.key, self.value) )

class BinaryMap:
    def __init__( self ):
        self.table = []

    def size( self ): return len(self.table)

    def insert(self, key, value) :           # 삽입 연산
        pos = 0
        while pos < self.size() :
            if self.table[pos].key >= key:
                break
            pos += 1

```

실습 단계

str(self) 함수는 연산자 중복 함수

모든 함수 인자로 self를 주고, 멤버 함수 안에서도 사용

삽입 연산: 위치를 찾아 엔트리 삽입

탐색 연산: 키값에 해당하는 엔트리 반환

삭제 연산: 엔트리를 먼저 찾고, 엔트리 삭제

binary_search는 멤버 함수가 아니고 하나의 함수

공백 상태의 map 생성

알파벳 순으로 정렬

엔트리 출력 시 탭을 띄움



맵의 응용: 영어 단어장

2 이진 탐색 맵 구현

파이썬의 딕셔너리를 이용한 나의 단어장

```
print("<파이썬의 딕셔너리를 이용한 나의 단어장>")
d = {}
d['data'] = '자료'
d['structure'] = '구조'
d['sequential search'] = '선형 탐색'
d['game'] = '게임'
d['binary search'] = '이진 탐색'
print("나의 단어장:")
print(d)
if d.get('game') : print("탐색:game --> ", d['game'])
if d.get('over') : print("탐색:over --> ", d['over'])
if d.get('data') : print("탐색:data --> ", d['data'])
d.pop('game')
```

```
print("삭제:game ")
print("나의 단어장:")
print(d)
```

2 이진 탐색 맵 구현

파이썬의 딕셔너리를 이용한 나의 단어장

```
C:\WINDOWS\system32\cmd.exe
<파이썬의 딕셔너리를 이용한 나의 단어장>
나의 단어장:
{'data': '자료', 'structure': '구조', 'sequential search': '선형 탐색', 'game': '게임', 'binary search': '이진 탐색'}
탐색:game --> 게임
탐색:data --> 자료
삭제:game
나의 단어장:
{'data': '자료', 'structure': '구조', 'sequential search': '선형 탐색', 'binary search': '이진 탐색'}
```