

Data Structure

자료구조



정렬 기초



한국기술교육대학교
온라인평생교육원

학습내용

- 정렬
- 선택 정렬
- 삽입 정렬
- 버블 정렬
- 정렬의 응용: 집합

학습목표

- 정렬의 개념을 설명할 수 있다.
- 선택 정렬의 동작원리를 설명할 수 있다.
- 삽입 정렬의 동작원리를 설명할 수 있다.
- 버블 정렬의 동작원리를 설명할 수 있다.
- 알고리즘의 개발에 정렬을 활용할 수 있다.

정렬



정렬 기초

정렬

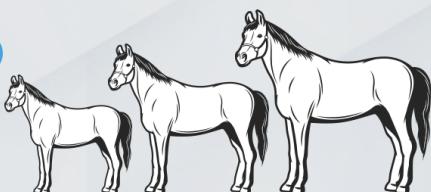
1 정렬의 개념

◉ 정렬이란?

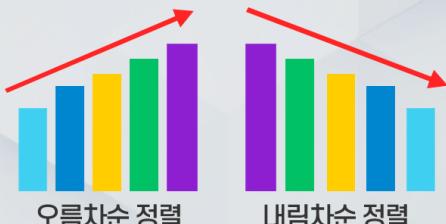
정렬

- 데이터를 순서대로 재배열한 것
- 가장 기본적이고, 중요한 알고리즘임
- 비교할 수 있는 모든 속성들은 정렬의 기준이 될 수 있음
- 예 오름차순 정렬, 내림차순 정렬, 키 순

예



경주마의 정렬(키 순)



정렬 기초

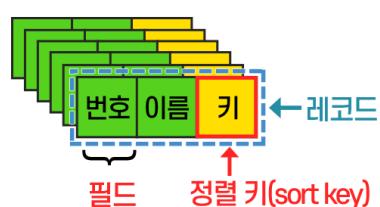
정렬

1 정렬의 개념

◉ 용어들

레코드 (Record)

- 정렬시켜야 될 대상
- 여러 개의 필드로 이뤄짐
- 필드들 중 정렬의 기준이 되는 필드인 정렬 키가 있음



경주마 레코드



정렬

정렬 기초

정렬

2 정렬 알고리즘의 종류

● 효율성에 따른 분류

단순하지만 비효율적인 방법

- ▶ 삽입 정렬, 선택 정렬, 버블 정렬 등

복잡하지만 효율적인 방법

- ▶ 쿵 정렬, 힙 정렬, 병합 정렬, 기수 정렬, 팀 정렬 등

정렬 기초

정렬

2 정렬 알고리즘의 종류

● 정렬 장소에 따른 분류

내부 정렬

- ▶ 모든 데이터가 메인 메모리에 있음

외부 정렬

- ▶ 외부 기억 장치에 대부분의 레코드가 있음



정렬

정렬 기초

정렬

2 정렬 알고리즘의 종류

- 키값의 비교 여부에 따른 분류

키값의 “비교”를 이용한 정렬

- ▶ 대부분의 정렬이 포함

“배분”을 이용한 정렬

- ▶ 기수 정렬, 카운팅 정렬이 포함

정렬 기초

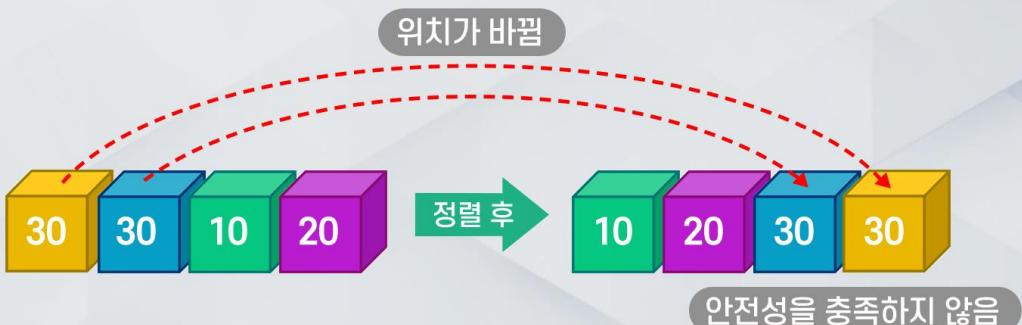
정렬

2 정렬 알고리즘의 종류

- 정렬 알고리즘의 특징

안정성(Stability)

- ▶ “동일한 키값의 레코드가 정렬 후 상대적인 위치를 유지하는가?”





정렬

정렬 기초

정렬

2 정렬 알고리즘의 종류

● 정렬 알고리즘의 특징

제자리 정렬

- ▶ “추가적인 메모리 공간을 필요로 하는가?”

입력의 종류에 따른 성능 차이

- ▶ “입력의 종류와 상관 없이 동일한 시간이 걸리는가?”



선택 정렬

정렬 기초

선택 정렬

1 선택 정렬의 원리

● 선택 정렬의 아이디어

선택 정렬

- 오른쪽 리스트에서 가장 작은 숫자를 선택해 왼쪽 리스트의 맨 뒤로 이동하는 작업을 반복

정렬 기초

선택 정렬

1 선택 정렬의 원리

● 선택 정렬의 아이디어

정렬 된(왼쪽) 리스트	정렬 안 된(오른쪽) 리스트	설명
[]	[5,3,8,4,9,1,6,2,7]	초기 상태
[1]	[5,3,8,4,9,6,2,7]	1선택 및 이동
[1,2]	[5,3,8,4,9,1,6,7]	2선택 및 이동
[1,2,3]	[5,8,4,9,6,7]	3선택 및 이동
...	...	4~8선택 및 이동
[1,2,3,4,5,6,7,8,9]	[]	9선택 및 이동

▶ 제자리 정렬이 되도록 하려면?



선택 정렬

정렬 기초

선택 정렬

1 선택 정렬의 원리

● 선택 정렬 전략

제자리 정렬 전략

- 입력 리스트를 두 부분으로 나누어 사용
 - 정렬된 부분과 정렬 안 된 부분
- 정렬 안 된 부분에서 가장 작은 숫자를 찾아 첫 번째 원소와 교환
 - 정렬된 부분이 점점 커지고, 정렬 안 된 부분이 줄어듦

정렬 기초

선택 정렬

1 선택 정렬의 원리

● 선택 정렬 과정 예





선택 정렬

정렬 기초

선택 정렬

2 선택 정렬 알고리즘

```

def selection_sort(A) :
    n = len(A)
    for i in range(n-1) :
        least = i;
        for j in range(i+1, n) :
            if (A[j]<A[least]) :
                least = j
        A[i], A[least] = A[least], A[i]      # 배열 항목 교환
        printStep(A, i + 1);                 # 중간 과정 출력용 문장
    
```

정렬 기초

선택 정렬

2 선택 정렬 알고리즘

● 선택 정렬의 시간 복잡도

시간 복잡도

- $(n-1) + (n-2) + \dots + 1 = n(n-1)/2 = O(n^2)$
- 리스트의 크기가 정해지면 자료 이동 횟수가 미리 결정
- 최선, 최악의 복잡도 동일

선택 정렬



정렬 기초

선택 정렬

2 선택 정렬 알고리즘

○ 선택 정렬의 특징

선택 정렬의 장점	선택 정렬의 단점
<ul style="list-style-type: none"> 알고리즘이 간단 제자리 정렬 자료 이동 횟수가 미리 결정 	<ul style="list-style-type: none"> 속도가 느림 안전성을 만족하지 않음 <p>예 [3, 2, 3, 1] $[1, 2, 3, 3] \rightarrow [1, 2, 3, 3] \rightarrow [1, 2, 3, 3]$ $\rightarrow [1, 2, 3, 3]$</p>

정렬 기초

선택 정렬

2 선택 정렬 알고리즘

○ 테스트 프로그램

```
data = [ 5, 3, 8, 4, 9, 1, 6, 2, 7]
print( "Original : " , data)
selection_sort(data)
print( "Selection : " , data)
```

```
C:\WINDOWS\system32\cmd.exe
Original : [5, 3, 8, 4, 9, 1, 6, 2, 7] 정렬 안 된 리스트
Step 1 = [1, 3, 8, 4, 9, 5, 6, 2, 7]
Step 2 = [1, 2, 8, 4, 9, 5, 6, 3, 7]
Step 3 = [1, 2, 3, 4, 9, 5, 6, 8, 7]
Step 4 = [1, 2, 3, 4, 9, 5, 6, 8, 7]
Step 5 = [1, 2, 3, 4, 5, 9, 6, 8, 7]
Step 6 = [1, 2, 3, 4, 5, 6, 9, 8, 7]
Step 7 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Step 8 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Selection : [1, 2, 3, 4, 5, 6, 7, 8, 9] 정렬된 리스트
```

선택 정렬

```

simpleSort.py  x testAllSort
1 import random
2
3 def printStep(arr, val) :
4     print(" Step %2d = " % val, end='')
5     print(arr)
6
7 # 선택 정렬 함수
8 def selection_sort(A) :
9     n = len(A)
10    for i in range(n-1) :
11        least = i;
12        for j in range(i+1, n) :
13            if (A[j]<A[least]) :
14                least = j
15        A[i], A[least] = A[least], A[i]
16        printStep(A, i + 1);
17
18
19
20
21
22
23
24
25
26

```

Original : [5, 3, 8, 4, 9, 1, 6, 2, 7]
Step 1 = [1, 3, 8, 4, 9, 5, 6, 2, 7]
Step 2 = [1, 2, 8, 4, 9, 5, 6, 3, 7]
Step 3 = [1, 2, 3, 4, 9, 5, 6, 8, 7]
Step 4 = [1, 2, 3, 4, 9, 5, 6, 8, 7]
Step 5 = [1, 2, 3, 4, 5, 9, 6, 8, 7]
Step 6 = [1, 2, 3, 4, 5, 6, 9, 8, 7]
Step 7 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Step 8 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Selection : [1, 2, 3, 4, 5, 6, 7, 8, 9]
계속하려면 아무 키나 누르십시오 . . .

배열 항목 교환
중간 과정 출력용 문장

(I) 실습하기

↳ 선택 정렬 알고리즘의 실행결과

실습 단계

선택 정렬 알고리즘의 실행결과

삽입 정렬



정렬 기초

삽입 정렬

1 삽입 정렬의 원리

● 삽입 정렬의 아이디어

삽입 정렬

- 정렬되어 있는 부분에 새로운 레코드를 올바른 위치에 삽입하는 과정 반복

예 카드 게임



정렬 기초

삽입 정렬

1 삽입 정렬의 원리

● 삽입 정렬 전략

제자리 정렬 전략

- 입력 리스트를 두 부분으로 나누어 사용
 - 정렬된(왼쪽) 부분과 정렬 안된(오른쪽) 부분
- 오른쪽 리스트의 첫 항목을 순서대로 왼쪽에 끼워 넣음



삽입 정렬

정렬 기초

삽입 정렬

1 삽입 정렬의 원리

● 삽입 정렬 전략

끼워 넣기

- 많은 항목들의 이동이 발생
- 끼워 넣을 위치에 따라 이동할 항목의 수가 다름
 - 최악의 경우? 최선의 경우?

정렬 기초

삽입 정렬

1 삽입 정렬의 원리

● 삽입 정렬 과정 예



삽입 정렬



정렬 기초

삽입 정렬

2 삽입 정렬 알고리즘

```
def insertion_sort(A) :
    n = len(A)
    for i in range(1, n) :
        key = A[i]
        j = i-1
        while j>=0 and A[j] > key :    # 두 번째 항목부터 모든 항목들을
            A[j + 1] = A[j]
            j -= 1
        A[j + 1] = key
    printStep(A, i)
```

정렬 기초

삽입 정렬

2 삽입 정렬 알고리즘

◎ 테스트 프로그램

C:\WINDOWS\system32\cmd.exe

```
Original : [5, 3, 8, 4, 9, 1, 6, 2, 7]
Step 1 = [3, 5, 8, 4, 9, 1, 6, 2, 7]
Step 2 = [3, 5, 8, 4, 9, 1, 6, 2, 7]
Step 3 = [3, 4, 5, 8, 9, 1, 6, 2, 7]
Step 4 = [3, 4, 5, 8, 9, 1, 6, 2, 7]
Step 5 = [1, 3, 4, 5, 8, 9, 6, 2, 7]
Step 6 = [1, 3, 4, 5, 6, 8, 9, 2, 7]
Step 7 = [1, 2, 3, 4, 5, 6, 8, 9, 7]
Step 8 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Insertion : [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

정렬 안 된 부분

정렬된 부분



삽입 정렬

정렬 기초

삽입 정렬

2 삽입 정렬 알고리즘

◉ 복잡도 분석

최선의 경우 $O(n)$ 이미 정렬되어 있는 경우 (비교 $n-1$ 번)최악의 경우 $O(n^2)$

역순으로 정렬되어 있는 경우

▶ 역순으로 정렬되어 있어 모든 단계에서 앞에 놓인 자료 전부 이동

평균의 경우

 $O(n^2)$

정렬 기초

삽입 정렬

2 삽입 정렬 알고리즘

◉ 삽입 정렬의 특징

삽입 정렬의 장점

- 알고리즘이 간단
- 제자리 정렬
- 안전성을 만족

삽입 정렬의 단점

- 속도가 느림
- 레코드가 큰 경우, 많은 이동이 필요
- 대부분 정렬되어 있다면 효율적



버블 정렬

정렬 기초

버블 정렬

1 버블 정렬의 원리

● 버블 정렬의 아이디어

버블 정렬(bubble sort)

- 인접한 2개의 레코드를 비교해 순서대로 서로 교환
- 비교-교환 과정을 리스트 전체에 수행(스캔)
- 1번의 스캔이 완료되면 리스트의 오른쪽 끝에 가장 큰 레코드가 들어감

정렬 기초

버블 정렬

1 버블 정렬의 원리

● 버블 정렬의 아이디어



버블 정렬



정렬 기초

버블 정렬

1 버블 정렬의 원리

● 버블 정렬 전략

버블 정렬 전략

- 교환이 발생하지 않을 때까지 끝으로 이동한 레코드를 제외하고 다시 스캔 반복

정렬 기초

버블 정렬

1 버블 정렬의 원리

● 버블 정렬 전략



버블 정렬



정렬 기초

버블 정렬

2 버블 정렬 알고리즘

```

def bubble_sort(A) :
    n = len(A)
    for i in range(n-1, 0, -1) :
        bChanged = False
        for j in range(i) :
            if (A[j]>A[j+1]) :
                A[j], A[j+1] = A[j+1], A[j]
                bChanged = True

        if not bChanged: break      # 교환이 없으면 종료
    printStep(A, n - i)          # 중간 과정 출력용 문장

```

정렬 기초

버블 정렬

2 버블 정렬 알고리즘

● 테스트 프로그램

선택 C:\WINDOWS\system32\cmd.exe

```

Original   : [5, 3, 8, 4, 9, 1, 6, 2, 7] ← 항목을 교환할 범위
Step 1 = [3, 5, 4, 8, 1, 6, 2, 7, 9]
Step 2 = [3, 4, 5, 1, 6, 2, 7, 8, 9]
Step 3 = [3, 4, 1, 5, 2, 6, 7, 8, 9] ← 정렬된 부분
Step 4 = [3, 1, 4, 2, 5, 6, 7, 8, 9]
Step 5 = [1, 3, 2, 4, 5, 6, 7, 8, 9]
Step 6 = [1, 2, 3, 4, 5, 6, 7, 8, 9] ← 더 이상 교환이 일어나지 않음. 정렬 종료
Bubble    : [1, 2, 3, 4, 5, 6, 7, 8, 9]

```



버블 정렬

정렬 기초

버블 정렬

2 버블 정렬 알고리즘

● 버블 정렬 시간 복잡도

비교 횟수	이동 횟수
$\sum_{i=1}^{n-1} i = \frac{n(n - 1)}{2} = O(n^2)$	<ul style="list-style-type: none"> 역순으로 연결된 경우(최악): 이동 횟수 = 3* 비교 횟수 이미 정렬된 경우(최선): 이동 횟수 = 0 평균의 경우: $O(n^2)$

정렬 기초

버블 정렬

2 버블 정렬 알고리즘

● 버블 정렬의 특징

버블 정렬의 장점	버블 정렬의 단점
<ul style="list-style-type: none"> 알고리즘이 간단 제자리 정렬 안전성을 만족 	<ul style="list-style-type: none"> 최악 혹은 평균적인 경우의 시간 복잡도가 $O(n^2)$로, 속도가 느림 레코드의 이동이 과다하지만, 최선의 경우와 같이 어느 정도 정렬되어 있다면 효과적



정렬의 응용: 집합

정렬 기초

정렬의 응용: 집합

1 정렬된 리스트로 구현한 집합

정렬되지 않은 리스트로 집합 구현

자료구조 “집합”을 구현하는 하나의 방법

→ 집합의 원소들을 항상 정렬된 순으로 저장한다면?

정렬된 리스트로 집합 구현

- 삽입 연산은 좀 더 복잡해짐
 - 추가할 원소의 위치를 찾아 끼워 넣기 필요
- 집합의 비교나 합집합, 차집합, 교집합 연산
 - 더 효율적인 알고리즘이 가능해짐

정렬 기초

정렬의 응용: 집합

1 정렬된 리스트로 구현한 집합

● 집합의 비교 연산 ‘==’

기본 조건

두 집합의 원소의 개수가 같아야 함

원소들이 정렬되어 있지 않는 경우

- 각 원소를 다른 집합의 모든 원소와 비교
- 시간 복잡도 $O(n^2)$

원소들이 정렬되어 있는 경우

- 원소들이 순서대로 같은 원소를 가져야 함
- 시간 복잡도 $O(n)$

정렬의 응용: 집합



정렬 기초

정렬의 응용: 집합

2 합집합 연산의 성능 개선 예

● 합집합 연산

합집합 연산 알고리즘

- 가장 작은 원소부터 비교해 더 작은 원소를 새로운 집합에 넣고 그 집합의 인덱스를 증가시킴
- 두 집합의 현재 원소가 같으면 하나만 넣음. 인덱스는 모두 증가시킴
- 한쪽 집합이 모두 처리되면 나머지 집합의 남은 원소를 순서대로 새 집합에 넣음
- 시간 복잡도가 $O(n^2)$ 에서 $O(n)$ 으로 개선됨
- 교집합과 차집합도 동일한 방법 적용 가능

정렬 기초

정렬의 응용: 집합

2 합집합 연산의 성능 개선 예

● 합집합(정렬되지 않은 리스트)





정렬의 응용: 집합

2 합집합 연산의 성능 개선 예

◎ 합집합(정렬된 리스트)



2 합집합 연산의 성능 개선 예

◎ 합집합 연산의 시간 복잡도

시간 복잡도

- 두 집합을 한 번씩만 스캔하면 됨
- $O(n^2) \rightarrow O(n)$ 으로 개선됨
- 교집합과 차집합도 동일한 방법 적용 가능함



정렬의 응용: 집합

정렬 기초

정렬의 응용: 집합

2 합집합 연산의 성능 개선 예

◎ 복잡도 비교

집합의 연산	정렬되지 않은 리스트	정렬된 리스트
insert(e)	$O(n)$	$O(n)$
eq(setB)	$O(n^2)$	$O(n)$
union(setB)	$O(n^2)$	$O(n)$
intersect(setB)	$O(n^2)$	$O(n)$
difference(setB)	$O(n^2)$	$O(n)$