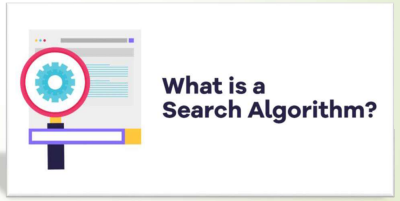
성균관대학교

문제해결과 알고리즘

탐색 알고리즘

JEE, Jung Eun
rosaliejee@skku.edu

Contents



1. 순차 탐색 (1/5)

- 순차 탐색 알고리즘
  - 초기 데이터 : 5 4 2 1 3
  - datalist

0	1	2	3	4
5	4	2	1	3

  - Ex1. 탐색 키(search key) : 2

0	1	2	3	4
5	4	2	1	3

    - 1단계 : datalist[0] vs Key => 5 != 2
    - 2단계 : datalist[1] vs Key => 4 != 2
    - 3단계 : datalist[2] vs Key => 2 == 2 → 탐색 성공
  - Ex2. 탐색 키(search key) : 7

0	1	2	3	4
5	4	2	1	3

    - 1단계 : datalist[0] vs Key => 5 != 7
    - 2단계 : datalist[1] vs Key => 4 != 7
    - 3단계 : datalist[2] vs Key => 2 != 7
    - 4단계 : datalist[3] vs Key => 1 != 7
    - 5단계 : datalist[4] vs Key => 3 != 7 → 탐색 실패

1. 순차 탐색 (2/5)

- *[w8_Ex1]* 순차 탐색 프로그램
  - 요구사항
    - 1. **순차 탐색 알고리즘**으로 입력 받는 **탐색 키(search key)**를 찾는 프로그램을 작성하시오.
      - 초기 데이터a : **5, 4, 2, 1, 3**
      - 초기 데이터b : **42, 63, 19, 25, 7, 36, 49**
    - 2. 순차 탐색을 수행하는 **사용자 정의 함수**를 작성하시오.
  - 문제해결 및 알고리즘
    - 순차 탐색 사용자 함수 정의
    - 메인코드
      - 순차 탐색 함수 호출
      - 함수 반환값으로결과출력
        - 탐색 성공
        - 탐색 실패

[실행 예시]

```
[순차 탐색]
리스트 데이터 : [5, 4, 2, 1, 3]
탐색 키(search key) 입력 : 2
=> 탐색 성공!!!
=> 탐색 위치(index) = 2
```

```
[순차 탐색]
리스트 데이터 : [42, 63, 19, 25, 7, 36, 49]
탐색 키(search key) 입력 : 25
=> 탐색 성공!!!
=> 탐색 위치(index) = 3
```

```
[순차 탐색]
리스트 데이터 : [5, 4, 2, 1, 3]
탐색 키(search key) 입력 : 7
=> 탐색 실패
```

```
[순차 탐색]
리스트 데이터 : [42, 63, 19, 25, 7, 36, 49]
탐색 키(search key) 입력 : 15
=> 탐색 실패
```

1. 순차 탐색 (3/5)

- *[w8_Ex2]* 순차 탐색 과정 출력 프로그램
  - 요구사항
    - 1. **순차 탐색 알고리즘**으로 입력 받는 **탐색 키(search key)**를 찾는 프로그램을 작성하시오.
      - 초기 데이터a : **5, 4, 2, 1, 3**
      - 초기 데이터b : **42, 63, 19, 25, 7, 36, 49**
    - 2. 순차 탐색을 수행하는 **사용자 정의 함수**를 작성하시오.
      - 단계별 탐색 과정에 대해 자세히 출력 하시오.
    - 3. 탐색을 성공하였을 경우 몇 회에 찾았는지 출력 하시오.

- **문제해결 및 알고리즘**
  - 순차 탐색 사용자 함수 정의 : 단계별 탐색 과정 출력
  - 메인코드
    - 순차 탐색 함수 호출
    - 함수 반환값으로결과출력:탐색 성공→ 횟수 출력 , 탐색 실패

1. 순차 탐색 (4/5)

- *[w8_Ex2] 실행 예시*

```
[순차 탐색]

리스트 데이터 : [5, 4, 2, 1, 3, 7, 6]

탐색 키(search key) 입력:3
1단계: 3와(과) 인덱스 0번째의 값 5 비교 => 불일치
2단계: 3와(과) 인덱스 1번째의 값 4 비교 => 불일치
3단계: 3와(과) 인덱스 2번째의 값 2 비교 => 불일치
4단계: 3와(과) 인덱스 3번째의 값 1 비교 => 불일치
5단계: 3와(과) 인덱스 4번째의 값 3 비교 => 일치

=> 5회에 탐색 성공!!!
=> 탐색 위치(index) = 4
```

```
[순차 탐색]

리스트 데이터 : [42, 63, 19, 25, 7, 36, 13, 49]

탐색 키(search key) 입력:19
1단계: 19와(과) 인덱스 0번째의 값 42 비교 => 불일치
2단계: 19와(과) 인덱스 1번째의 값 63 비교 => 불일치
3단계: 19와(과) 인덱스 2번째의 값 19 비교 => 일치

=> 3회에 탐색 성공!!!
=> 탐색 위치(index) = 2
```

```
[순차 탐색]

리스트 데이터 : [5, 4, 2, 1, 3, 7, 6]

탐색 키(search key) 입력:11
1단계: 11와(과) 인덱스 0번째의 값 5 비교 => 불일치
2단계: 11와(과) 인덱스 1번째의 값 4 비교 => 불일치
3단계: 11와(과) 인덱스 2번째의 값 2 비교 => 불일치
4단계: 11와(과) 인덱스 3번째의 값 1 비교 => 불일치
5단계: 11와(과) 인덱스 4번째의 값 3 비교 => 불일치
6단계: 11와(과) 인덱스 5번째의 값 7 비교 => 불일치
7단계: 11와(과) 인덱스 6번째의 값 6 비교 => 불일치

=> 탐색 실패
```

```
[순차 탐색]

리스트 데이터 : [42, 63, 19, 25, 7, 36, 13, 49]

탐색 키(search key) 입력:11
1단계: 11와(과) 인덱스 0번째의 값 42 비교 => 불일치
2단계: 11와(과) 인덱스 1번째의 값 63 비교 => 불일치
3단계: 11와(과) 인덱스 2번째의 값 19 비교 => 불일치
4단계: 11와(과) 인덱스 3번째의 값 25 비교 => 불일치
5단계: 11와(과) 인덱스 4번째의 값 7 비교 => 불일치
6단계: 11와(과) 인덱스 5번째의 값 36 비교 => 불일치
7단계: 11와(과) 인덱스 6번째의 값 13 비교 => 불일치
8단계: 11와(과) 인덱스 7번째의 값 49 비교 => 불일치

=> 탐색 실패
```


1. 순차 탐색 (5/5)

- *[w8_Ex3]* 과일 탐색 프로그램
  - 요구사항
    - 1. **순차 탐색**으로 입력 받는 과일을 리스트에서 찾는 프로그램을 작성하시오.
    - 2. 10개의 과일로 리스트를 생성 하시오.
    - 3. 입력한 과일을 찾았다면 저장 위치(index)를 출력 하시오.
    - 4. 무한 반복으로 과일을 찾을 수 있도록 하고, '끝' 입력 시 프로그램을 종료 하시오.

- **문제해결 및 알고리즘**
  - 순차 탐색 사용자 함수 정의
  - 리스트 생성 → list=[ ]
  - 무한 반복문 → while True

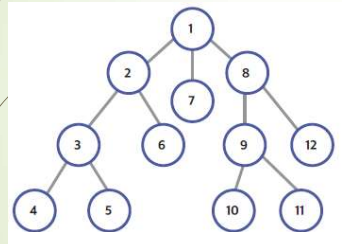
[실행 예시]

```
[과일 탐색 프로그램]
찾을 과일 입력 : 사과
사과의 저장 위치(index)는 0입니다.
찾을 과일 입력 : 딸기
딸기의 저장 위치(index)는 5입니다.
찾을 과일 입력 : 수박
수박의 저장 위치(index)는 3입니다.
```

```
찾을 과일 입력 : 자몽
자몽은(는) 과일 리스트에 없습니다.
찾을 과일 입력 : 바나나
바나나의 저장 위치(index)는 7입니다.
찾을 과일 입력 : 자두
자두은(는) 과일 리스트에 없습니다.
찾을 과일 입력 : 끝
프로그램을 종료합니다.
```


2. 그래프 순회 (1/5)

- 깊이 우선 탐색 (depth first search : DFS) 알고리즘
  - 탐색 과정 : **스택(Stack)**



- **그래프의 정점(vertex) 저장**
  - **딕셔너리(dictionary)** 자료형으로 선언

```
Dfs_graph ={'1':['8','7','2'],
            '2':['1','6','3'],
            '3':['2','5','4'],
            '4':['3'],
            '5':['3'],
            '6':['2'],
            '7':['1'],
            '8':['1','12','9'],
            '9':['8','11','10'],
            '10':['9'],
            '11':['9'],
            '12':['8']}
```

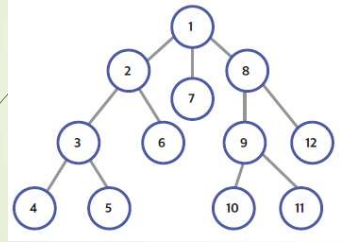
2. 그래프 순회 (2/5)

- **스택(Stack)을 이용하는 깊이 우선 탐색 과정**

1) 시작 정점 A를 결정하여 탐색한다.
2) 시작 정점 A에 인접한 정점 중에서
   a. 탐색하지 않은 정점 B가 있으면, 시작 정점 A를 스택에 push하고 B를 탐색한다. 그리고 B를 A로 설정하여 다시 2) 를 수행한다.
   b. 탐색하지 않은 정점이 없으면, 탐색의 방향을 바꾸기 위해서 스택을 pop하고 난 후, 반환 받은 가장 마지막 탐색 정점을 A로 설정하여 다시 2) 를 수행한다.
3) 스택이 공백이 될 때까지 2) 를 반복한다.

2. 그래프 순회 (3/5)

- 너비 우선 탐색 (breadth first search : BFS) 알고리즘
  - **탐색 과정 : 큐(Queue)**



- **그래프의 정점(vertex) 저장**
  - **딕셔너리(dictionary) 자료형으로 선언**

```
Bfs_graph ={'1':['2','7','8'],
            '2':['1','3','6'],
            '3':['2','4','5'],
            '4':['3'],
            '5':['3'],
            '6':['2'],
            '7':['1'],
            '8':['1','9','12'],
            '9':['8','10','11'],
            '10':['9'],
            '11':['9'],
            '12':['8']}
```


2. 그래프 순회 (4/5)

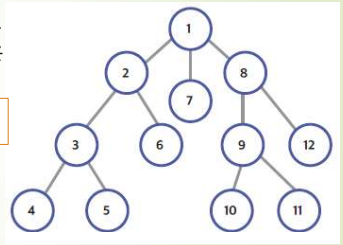
- **큐(Queue)를 이용한 너비 우선 탐색 방법**

1) 시작 정점 A를 결정하여 탐색한다.
2) 시작 정점 A에 인접한 정점들 중에서 탐색하지 않은 정점을 차례로 탐색하면서 큐에 넣는다.
3) 탐색하지 않은 인접한 정점이 없으면, 탐색했던 정점에서 인접한 정점들을 다시 차례로 탐색하기 위해서 큐에서 꺼내서(pop) 구한 정점에서 2)를 반복한다.
4) 큐가 공백이 될 때까지 2) ~ 3) 을 반복한다.

2. 그래프 순회 (5/5)

- *[w8_Ex4]* 깊이 우선 탐색 프로그램
  - 요구사항
    - 1. 그래프의 정점을 딕셔너리 자료형으로 선언 하시오.
    - 2. **깊이 우선 탐색 알고리즘(DFS)**을 사용하여 **입력 받은 정점**을 탐색하는 과정을 출력 하시오.

```
[깊이 우선 탐색(DFS)]
탐색 정점 : 10
탐색 순서 => ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
```



- *[w8_Ex5]* 너비 우선 탐색 프로그램
  - 요구사항
    - 1. 그래프의 정점을 딕셔너리 자료형으로 선언 하시오.
    - 2. **너비 우선 탐색 알고리즘(BFS)**을 사용하여 입력 받은 정점을 탐색하는 과정을 출력 하시오.

```
[너비 우선 탐색(BFS)]
탐색 정점 : 10
탐색 순서 => ['1', '2', '7', '8', '3', '6', '9', '12', '4', '5', '10']
```