문제해결과 알고리즘

Lecture *7*

2017.11

Contents

Search (탐색)

1. Sequence Search
2. Binary Search
3. DFS
4. BFS

일부 장표가 다음 site의 내용을 활용하여 작성되었음.
http://interactivepython.org/runestone/static/pythonds/index.html

Search (탐색)

- 여러개의 자료 중에서 특정 기준에 적합한 자료를 찾는 것
  - 도서관 책 중 원하는 책 찾기
  - 영어 사전에서 단어 찾기
  - 저렴한 가격에 좋은 물건 찾기
  - 네비게이션의 경로 찾기

- 탐색 키에 해당하는 인덱스 번호 반환
  - 인덱스 번호란
    - 자료 리스트 목록의 순차적 번호를 의미
    - 첫 번째 번호는 '0'부터 시작
- 예: [2, 4, 7, 8, 12]에서 7을 탐색하는 경우
  - 인덱스 번호 2가 반환
- 탐색하고자 하는 자료가 리스트 안에 포함되어 있지 않는 경우
  - 일반적으로 -1을 반환 (false)

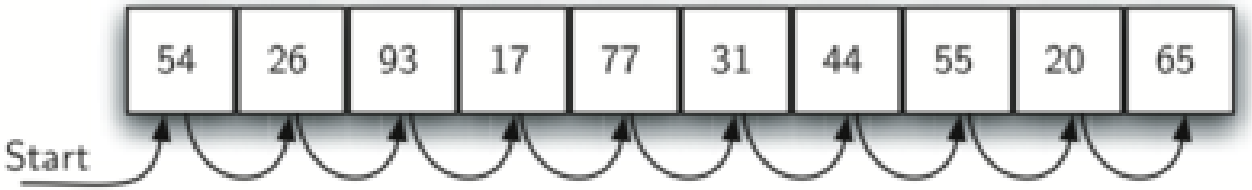
Example in python

```
aList = [123, 'xyz', 'zara', 'abc']
print("Index for xyz : ", aList.index('xyz'))
print("Index for zara : ", aList.index('zara'))
```

1. Sequence Search

순차탐색 (Sequential Search)

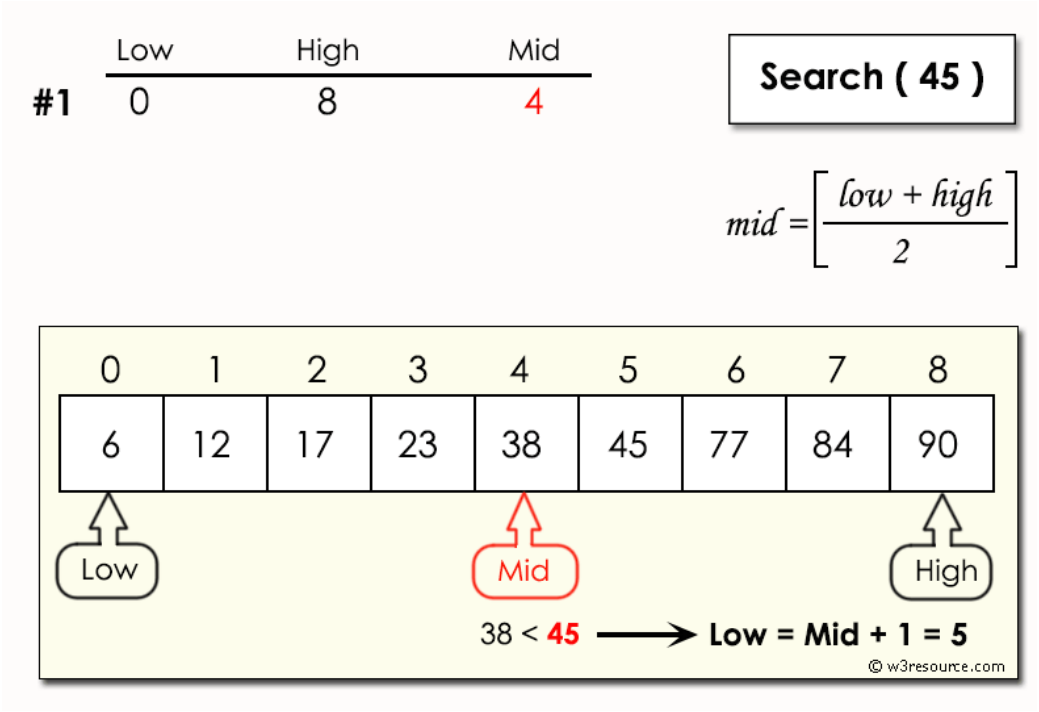
- 원하는 값을 찾을 때까지 자료를 처음부터 끝까지 차례로 하나씩 비교하면서 찾음
- 단순하지만 비효율적
- 시간복잡도 : O(n)



```
def Sequential_Search(dlist, item):
    pos = 0
    found = False

    while pos < len(dlist) and not found:
        if dlist[pos] == item:
            found = True
        else:
            pos = pos + 1

    return found, pos


print(Sequential_Search([11, 23, 58, 31, 56, 77, 43, 12,
65, 19], 31))
```

2. Binary Search

- 정렬이 되어 있다면, Sequence Search 보다 효율적 방법
- 중간 데이터부터 탐색
  - 오른쪽과 왼쪽으로 나누어서 탐색 가능
- Divide-and-Conquer

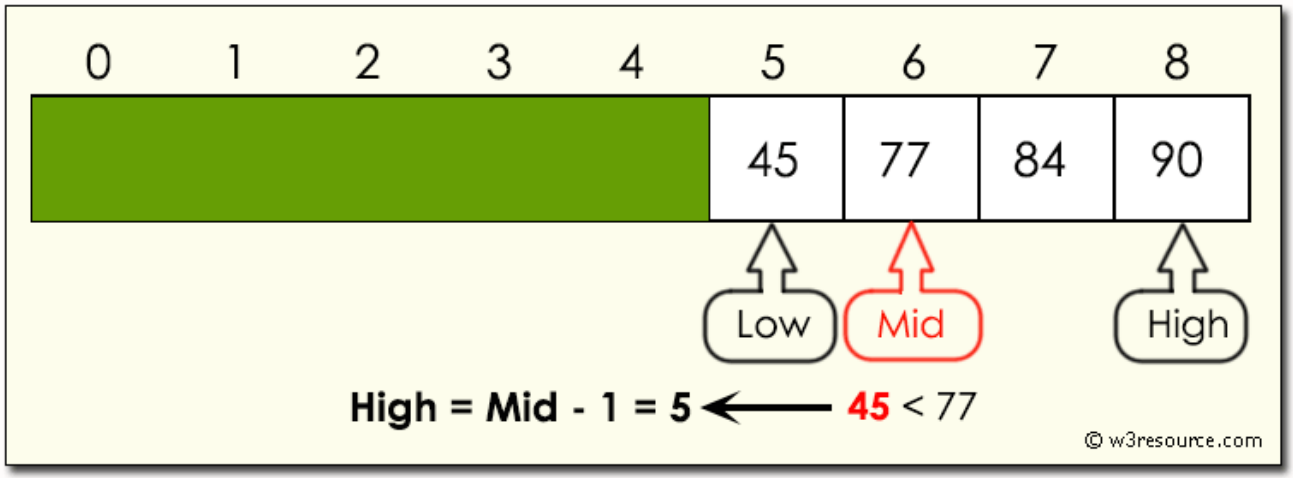


https://www.w3resource.com/w3r_images/binary-search-3.png

	Low	High	Mid
#1	0	8	4
#2	5	8	6

Search (45)

$$mid = \left[\frac{low + high}{2} \right]$$



https://www.w3resource.com/w3r_images/binary-search-3.png

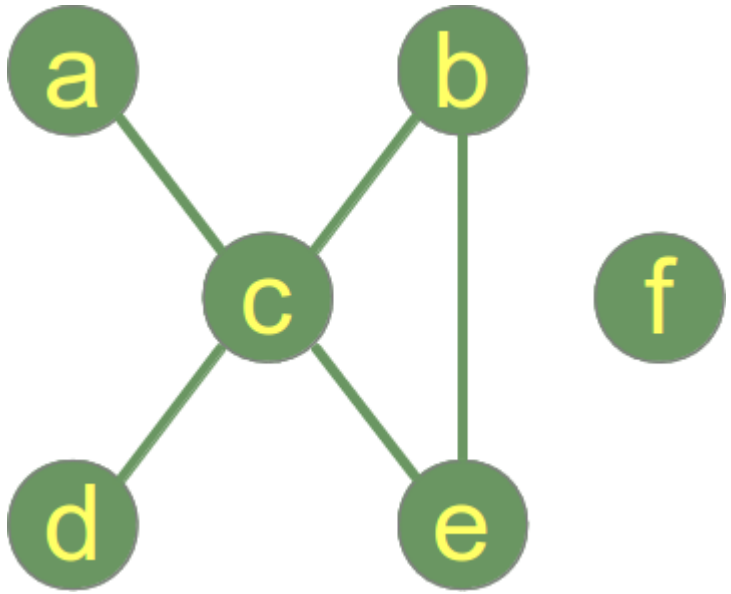
```
def binary_search(item_list, item):
    low = 0
    high = len(item_list) - 1
    found = False
    while (low <= high and not found):
        mid = (low + high) // 2
        if item_list[mid] == item:
            found = True
        else:
            if item < item_list[mid]:
                high = mid - 1
            else:
                low = mid + 1
    return found, mid


print(binary_search([1, 2, 3, 5, 8], 6))
print(binary_search([1, 2, 3, 5, 8], 5))
```

3. DFS (Depth First Search, 깊이 우선 탐색)

graph

- Nodes or Vertices
- Edges

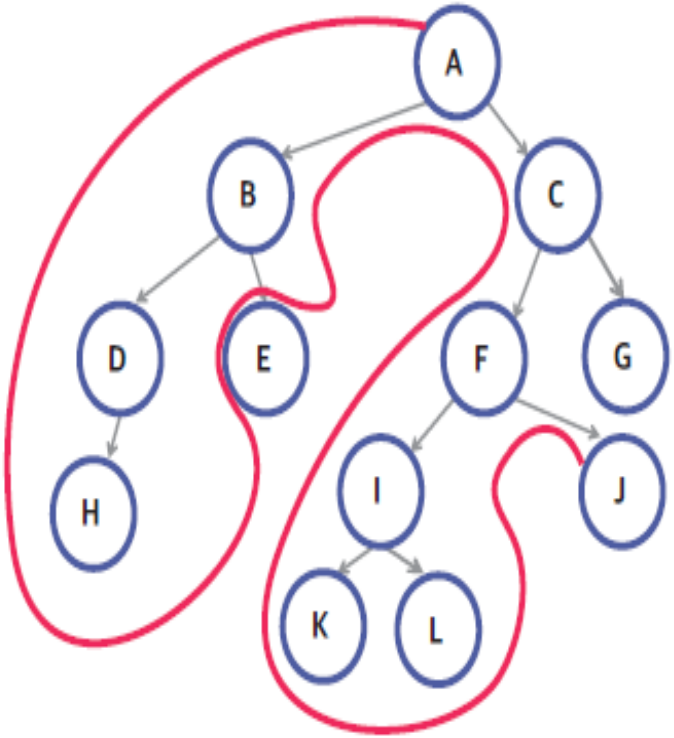
- Adjacent matrix




```
graph = { "a" : ["c"],
          "b" : ["c", "e"],
          "c" : ["a", "b", "d", "e"],
          "d" : ["c"],
          "e" : ["c", "b"],
          "f" : [] }
```

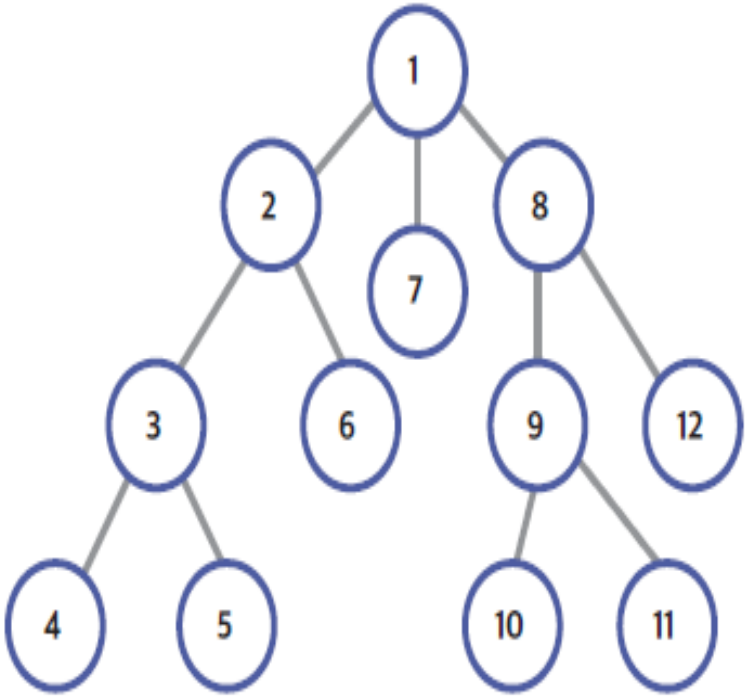
DFS

- 시작 node에서 연결된 간선(edge)이 존재하는 경우 계속하여 깊게 들어가며 탐색하는 알고리즘
- 깊게 들어가다 최하위 레벨에 해당하는 노드(leaf)에 도달하면 되돌아가기(backtracking)를 적용하여 다른 길이 연결된 가장 가까운 노드로 이동



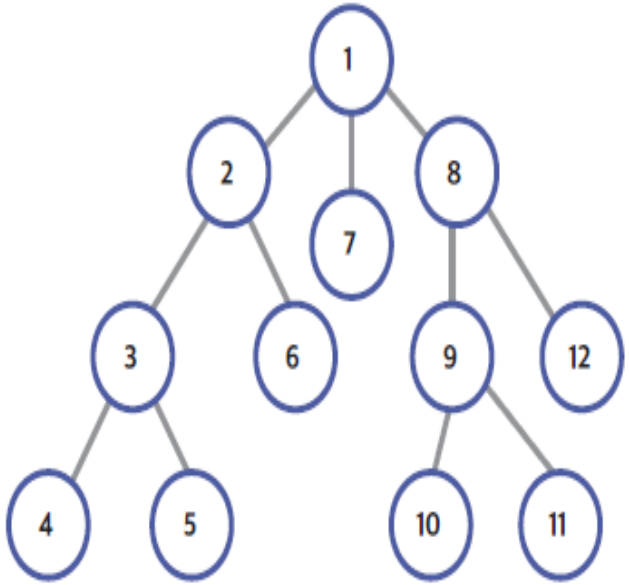
DFS 예시

- 주어진 트리에서 '9'를 탐색



DFS 예시

- 탐색 과정 : 스택 (Stack)



			PUSH					
		PUSH	4	POP				
	PUSH	3	5	5	POP			
	2	6	6	6	6	POP		PUSH
PUSH	7	7	7	7	7	7	POP	9
1	8	8	8	8	8	8	8	12

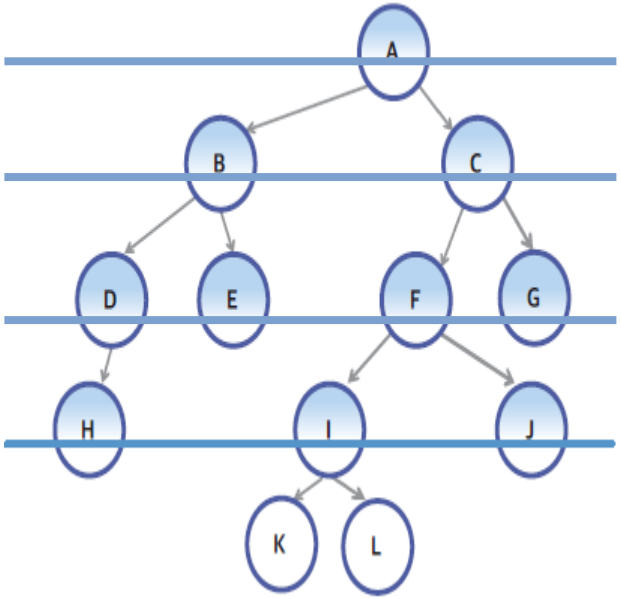
```
Stack stack // Stack 선언

stack.push(root)

while 스택에 원소가 있을 동안 do
  방문할 노드 V = stack.pop() // 스택 맨 위의 원소
  V.visit() // 방문
  stack.push(V의 자손 노드들 *이미 방문한 노드 제외)
        // 방문한 노드의 자손 노드들을 스택에 push
end
```

```
def dfs(graph, root, search):
    visited = []  # 각 node이 방문되었는지 기록
    stack = [root, ]

    while stack:  # stack 비워지면 탐색 종료
        node = stack.pop()  # 방문 노드
        if node not in visited:  # 새롭게 방문한 것 이라면
            visited.append(node)
            if node == search:
                break
            stack.extend([x for x in graph[node] if
x not in visited])

    return visited;
```

4. BFS (Breadth First Search, 너비 우선 탐색)

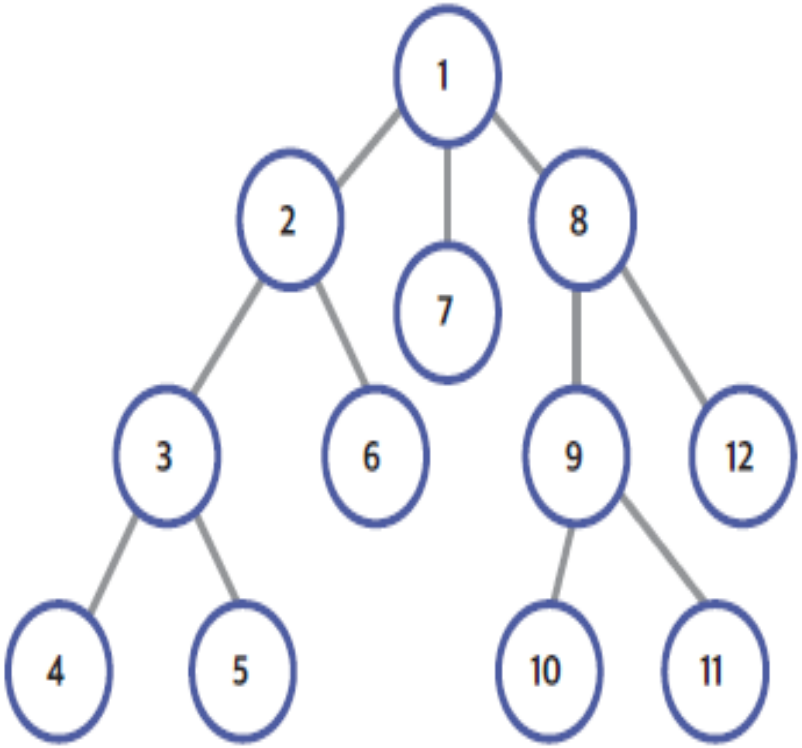
BFS

- 루트 노드로부터 시작하여 깊이가 1인 모든 노드들을 우선 비교 탐색
- 깊이가 2인 모든 노드를 비교하며 탐색
- 그 다음에 깊이가 3인 모든 노드들을 대상으로 비교 탐색
- 반복하여 깊이를 증가하며 탐색 범위를 확장하는 알고리즘



BFS 예시

- 주어진 트리에서 '9'를 탐색



BFS 예시

- 탐색 과정 : 큐 (Queue)