

## 데이터로 하는 마케팅

데이터로 하는 마... 구독하기

[Python/알고리즘](#)

## DFS 완벽 구현하기 [Python]

BK\_Paul 2021. 1. 29. 08:32

## 비스포크웨딩&amp;혼수페어 10월

역대급 할인혜택 "비스포크" 웨딩박람회, 웨딩 혼수를  
지금 바로 신청!

샐리브라이드

## 1. DFS의 기본 개념

## 2. 스택/큐를 활용한 DFS 구현하기

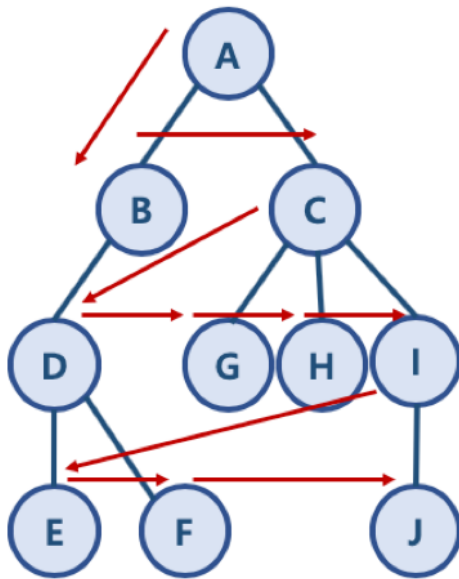
## 3. 재귀함수를 이용한 DFS 구현하기

## 1. DFS의 기본 개념

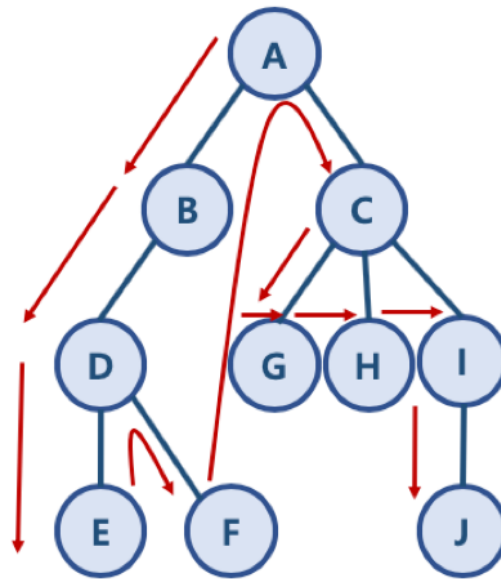
## (1) 기본개념

DFS란 Depth first search의 약자로서 **그래프 자료에서 데이터를 탐색하는 알고리즘**입니다.

한 번 예시를 들어 보겠습니다.



Breadth First Search



Depth First Search

오른쪽에 보시다 싶이 A부터 J까지 노드가 연결되어 있는 그래프 자료를 볼 수 있습니다. 그래프에서 특정 노드를 찾으려고 할 때, 위에서 부터 찾느냐 혹은 옆에서부터 찾느냐 그 차이가 있겠죠.

**위에서 아래로 찾는 방식을 바로 DFS(Depth First Search)라고 부르는 것입니다.**

BFS(Breadth First Search)는 차후 포스팅에서 다루도록 하겠습니다.

참고로 위에서 아래로 탐색할 때 왼쪽으로 가냐 오른쪽으로 가냐는 전혀 상관 없습니다~

**데이터 입력하기 귀찮으신 분들은 아래의 코드 복붙해주세요.**

```
1 graph = dict()
2
3 graph['A'] = ['B', 'C']
4 graph['B'] = ['A', 'D']
5 graph['C'] = ['A', 'G', 'H', 'I']
6 graph['D'] = ['B', 'E', 'F']
7 graph['E'] = ['D']
8 graph['F'] = ['D']
9 graph['G'] = ['C']
10 graph['H'] = ['C']
11 graph['I'] = ['C', 'J']
12 graph['J'] = ['I']
```

CS

## (2) DFS의 기본 원칙

DFS에서 데이터를 찾을 때는 항상 **"앞으로 찾아야 가야할 노드"**와 **"이미 방문한 노드"**를 기준으로 데이터를 탐색을 합니다.

이 원칙을 **반드시 기억**해주셔야 해요.

그래서 특정 노드가 **"앞으로 찾아야 가야할 노드"**라면 계속 검색을 하는 것이고, **"이미 방문한 노드"**면 무시하거나 따로 저장하면 됩니다.

### (3) DFS의 구현 방식

DFS를 구현할 때는 기본적으로 "스택/큐"를 활용할 수도 있고, "재귀함수를 통해 구현"할 수도 있습니다. 두 가지 방법 모두 아래의 부분에서 소개해드리도록 하겠습니다.

## 2. 스택/큐를 활용한 DFS 구현

### (1) 리스트를 활용한 DFS 구현

```

1  def dfs(graph, start_node):
2
3      ## 기본은 항상 두개의 리스트를 별도로 관리해주는 것
4      need_visited, visited = list(), list()
5
6      ## 시작 노드를 지정하기
7      need_visited.append(start_node)
8
9      ## 만약 아직도 방문이 필요한 노드가 있다면,
10     while need_visited:
11
12         ## 그 중에서 가장 마지막 데이터를 추출 (스택 구조의 활용)
13         node = need_visited.pop()
14
15         ## 만약 그 노드가 방문한 목록에 없다면
16         if node not in visited:
17
18             ## 방문한 목록에 추가하기
19             visited.append(node)
20
21             ## 그 노드에 연결된 노드를
22             need_visited.extend(graph[node])
23
24     return visited

```

CS

여기서는 need\_visited에서 추가된 Node들 중에서 가장 끝에 있는 것을 뽑아서 검색하는 방식입니다. 바로 이것이 "스택"을 활용한 방식이죠.

파이썬은 굉장히 편한 언어라서 리스트로도 쉽게 구현할 수 있습니다. 다만 list에서 pop을 활용하면 성능면에서 떨어지는 단점이 있어요.

```

7]: def dfs(graph, start_node):

    need_visited, visited = list(), list()
    need_visited.append(start_node)

    while need_visited:
        node = need_visited.pop()

        if node not in visited:
            visited.append(node)
            need_visited.extend(graph[node])

    return visited

8]: dfs(graph, 'A')

8]: ['A', 'C', 'I', 'J', 'H', 'G', 'B', 'D', 'F', 'E']

```

스택/큐를 활용한 DFS

## (2) deque를 활용한 DFS 구현

```

1 def dfs2(graph, start_node):
2     ## deque 패키지 불러오기
3     from collections import deque
4     visited = []
5     need_visited = deque()
6
7     ##시작 노드 설정해주기
8     need_visited.append(start_node)
9
10    ## 방문이 필요한 리스트가 아직 존재한다면
11    while need_visited:
12        ## 시작 노드를 지정하고
13        node = need_visited.pop()
14
15        ##만약 방문한 리스트에 없다면
16        if node not in visited:
17
18            ## 방문 리스트에 노드를 추가
19            visited.append(node)
20            ## 인접 노드들을 방문 예정 리스트에 추가
21            need_visited.extend(graph[node])
22
23    return visited
24

```

*Colored by Color Scripter*

CS

```
def dfs2(graph, start_node):
    ## deque 패키지 불러오기
    from collections import deque
    visited = []
    need_visited = deque()

    ##시작 노드 설정해주기
    need_visited.append(start_node)

    ## 방문이 필요한 리스트가 아직 존재한다면
    while need_visited:
        ## 시작 노드를 지정하고
        node = need_visited.pop()

        ## 만약 방문한 리스트에 없다면
        if node not in visited:

            ## 방문 리스트에 노드를 추가
            visited.append(node)
            ## 인접 노드들을 방문 예정 리스트에 추가
            need_visited.extend(graph[node])

    return visited
```

[데이터로 하는 마... 구독하기](#)

```
dfs2(graph, 'A')
```

```
['A', 'C', 'I', 'J', 'H', 'G', 'B', 'D', 'F', 'E']
```

스택/큐를 구현할 때는 collections라는 패키지에서 deque를 활용하시는 것을 추천드립니다.

논리는 거의 동일합니다만, **성능면에서 list() 형태보다 deque형태가 훨씬 좋아요!!**

### 3. 재귀함수를 통한 DFS 구현

```
1 def dfs_recursive(graph, start, visited = []):
2     ## 데이터를 추가하는 명령어 / 재귀가 이루어짐
3     visited.append(start)
4
5     for node in graph[start]:
6         if node not in visited:
7             dfs_recursive(graph, node, visited)
8     return visited
```

CS

```
[15]: def dfs_recursive(graph, start, visited = []):
      visited.append(start)

      for node in graph[start]:
          if node not in visited:
              dfs_recursive(graph, node, visited)
      return visited

[16]: dfs_recursive(graph, 'A')

[16]: ['A', 'B', 'D', 'E', 'F', 'C', 'G', 'H', 'I', 'J']
```

재귀 함수를 활용한 DFS

재귀 함수를 통해서 DFS를 구현해봤습니다.

특징 정인 것은 visited 자료형을 기본 함수 인자로 포함시키고, 방문한 리스트들을 재귀 함수를 통해서 계속 visited에 담는 방식입니다.

훨씬 단순한 구조이지만, 재귀 함수를 이해하지 못 하면 어려워 보일 수도 있죠. 저는 개인적으로 재귀 함수를 DFS 구현할 때 많이 활용합니다.

아직도 인적성 검사 안 해봤어?

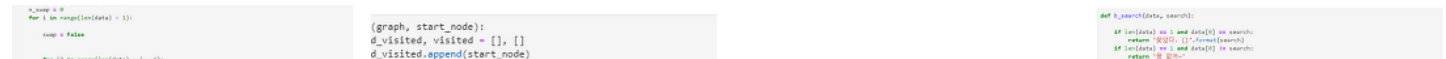
사람인 인성검사 아직도 안해본 분 찾습니다

사람인

'Python > 알고리즘' 카테고리의 다른 글		
[Python]버블정렬 알고리즘 코드 및 예시로 마스터 하기 (0)		2022.08.23
BFS 완벽 구현하기 - 파이썬 (0)		2021.02.01
그래프의 이해 - 고도 알고리즘을 위한 기초 개념 (0)		2021.01.17
탐색 알고리즘 - 이진 탐색 (0)		2021.01.11
탐색 알고리즘 - 순차 탐색(Sequential Search) (0)		2021.01.09

태그    #BFS,    #Depth first search,    #dfs,    #dfs구현,    #Python,    #데이터구조,    #알고리즘,    #재귀함수,    #코딩테스트,    #파이썬

'Python/알고리즘' Related Articles



```
def data[i] > data[i+1]:
    data[i], data[i+1] = data[i+1], data[i]
    swap = True
    continue
else:
    swap = False
    break
return data

def bubble_sort_descending(data):
    n_swap = 0
    for i in range(len(data)-1):
        swap = False
        for j in range(len(data)-1-i):
            if data[j] < data[j+1]:
                data[j], data[j+1] = data[j+1], data[j]
                swap = True
                n_swap += 1
        if swap == False:
            break
    return data
```

```
def need_visited:
    node = need_visited[0]
    del need_visited[0]

    if node not in visited:
        visited.append(node)
        need_visited.extend(graph[node])
    return visited

ph, 'A')
B', 'C', 'D', 'G', 'H', 'I', 'E', 'F', 'J']
```

NO IMAGE

```
if len(data) <= 1:
    return '통 한 것도 없어 ~'
median = (len(data)-1)//2

def search:
    return b_search(data[median], search)

import random
data = random.sample(range(10), 10)
data.sort()
data


[5, 6, 4, 11, 17, 34, 24, 29, 33, 38, 39, 40, 41, 45, 51, 54, 57, 61, 62]
b_search(data, 17)
'통없어..!'
```

[Python]버블정렬 알고리즘 코드 및 예시로 마스터 하기

BFS 완벽 구현하기 - 파이썬

그래프의 이해 - 고도 알고리즘을 위한 기초 개념

탐색 알고리즘 - 이진 탐색



익명2021.07.29 21:37

비밀댓글입니다

댓글주소수정/삭제댓글쓰기

광고



아직도 인적성 검사 안 해봤어?

사람인



지구깡냥2021.08.19 21:45신고

잘 읽었습니다!!


댓글주소수정/삭제댓글쓰기



k2021.08.27 22:35

dfs2의 함수 결과가 아닌거 같아요

댓글주소수정/삭제댓글쓰기



BK\_Paul2021.09.01 09:08신고

아 제가 스크린샷을 찍을 때 잘못 찍어서 올렸네요. 해당 내용 수정하도록 하겠습니다. 알려주셔서 정말 감사합니다~!

댓글주소수정/삭제

이름

암호

☐ Secret

여러분의 소중한 댓글을 입력해주세요.

댓글달기

