

# Search Algorithms

---

Practice 2

# DFS & BFS

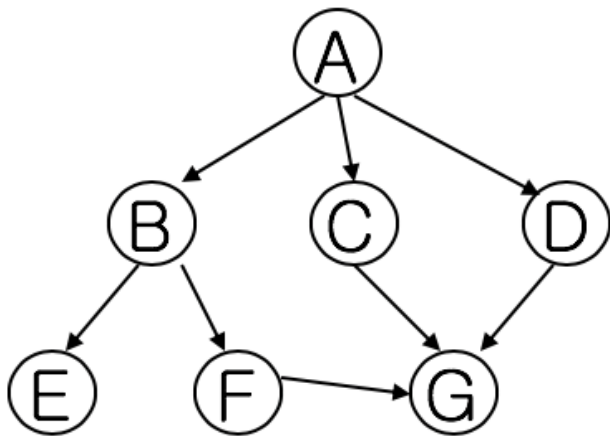
DFS



BFS



# Depth First Search



**open** = {START}, **closed** =  $\emptyset$

while (open  $\neq \emptyset$ )

    remove leftmost state from **open**  $\rightarrow X$

    if (X is GOAL) return (success)

    else

        generate children of X

        put X into **closed**

        eliminate child if it is in open or close

        put remaining children into left of **open**(stack)

return (fail)

# DFS 구현 실습

```
from copy import deepcopy

graph = {'A': ['B', 'C', 'D'],
        'B': ['E', 'F'],
        'C': ['G'],
        'D': ['G'],
        'E': [],
        'F': ['G'],
        'G': []}

# print fuction
def print_list(X, open_list, closed):
    print("-----")
    print("X =", X)
    print('open :', open_list)
    print('closed :', closed)

def DFS(graph, start, goal):
    open_list = []
    closed_list = []
    #open = {START}, closed = {}
    open_list.extend(start)
    print_list(None, open_list, closed_list)
    while(open_list):
        #DFS Algorithm

    return "*** Fail ***"

start_state = input("Start State: ")
goal_state = input("Goal State: ")
print(DFS(deepcopy(graph), start_state, goal_state))
```

# DFS 구현 실습

```
while(open_list):
    #remove leftmost state from open -> X
    (1)

    #if (X is GOAL) return (success)
    if (2):
        print_list(X, open_list, closed_list)
        return "*** Success ***"

    else:
        #generate children of X
        (3)

        #put X into closed
        (4)

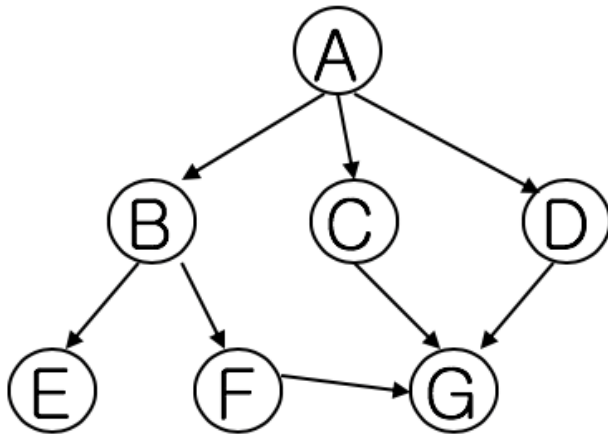
        # eliminate child if it is in open or close
        (5)

        #put remaining children into left of open(stack)
        (6)

    #print list
    print_list(X, open_list, closed_list)
```

- (1) 가장 왼쪽에 있는 State를 X에 할당
  - hint: pop()
- (2) X와 GOAL이 같을 때를 표현
- (3) children에 X의 자식노드들을 할당
  - graph의 구조를 참고하세요.
- (4) closed에 X를 추가
  - hint: extend()
- (5) open과 closed 리스트를 확인하여 중복된 child를 제거
  - for문과 if문을 사용
- (6) 남은 children을 open 리스트 왼쪽에 추가
  - hint: + 연산자

# Breadth First Search



**open** = {START}, **closed** =  $\emptyset$

while (open  $\neq \emptyset$ )

    remove leftmost state from **open**  $\rightarrow X$

    if (X is GOAL) return (success)

    else

        generate children of X

        put X into **closed**

        eliminate child if it is in open or close

        put remaining children into right of **open**(queue)

return (fail)

# BFS 구현 실습

```
from copy import deepcopy

graph = {'A': ['B', 'C', 'D'],
        'B': ['E', 'F'],
        'C': ['G'],
        'D': ['G'],
        'E': [],
        'F': ['G'],
        'G': []}

# print fuction
def print_list(X, open_list, closed):
    print("-----")
    print("X =", X)
    print('open :', open_list)
    print('closed :', closed)

def BFS(graph, start, goal):
    open_list = []
    closed_list = []
    #open = {START}, closed = {}
    open_list.extend(start)
    print_list(None, open_list, closed_list)
    while(open_list):
        #BFS algorithm

    return "*** Fail ***"

start_state = input("Start State: ")
goal_state = input("Goal State: ")
print(BFS(deepcopy(graph), start_state, goal_state))
```

# BFS 구현 실습

```
while(open_list):
    #remove leftmost state from open -> X
    (1)

    if (2):
        print_list(X, open_list, closed_list)
        return "*** Success ***"
    else:
        #generate children of X
        (3)

        #put X into closed
        (4)

        # eliminate child if is in open or close
        (5)

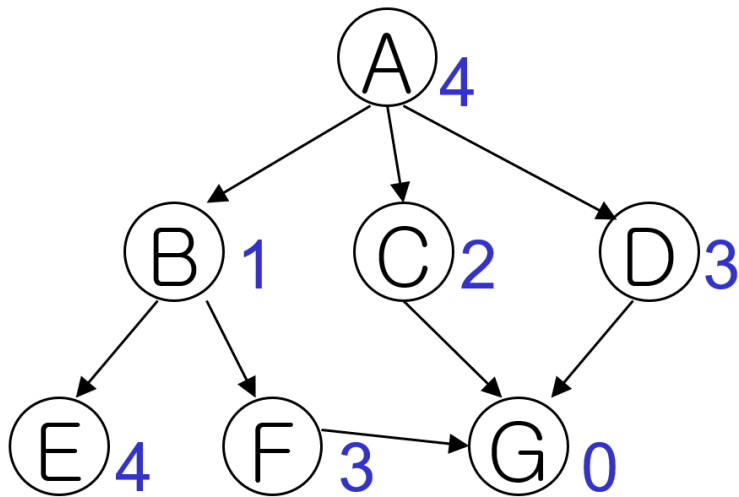
        # put remaining children into right of open(queue)
        (6)

# print list
print_list(X, open_list, closed_list)
```

- (1) 가장 왼쪽에 있는 State를 X에 할당
  - hint: pop()
- (2) X와 GOAL이 같을 때를 표현
- (3) children에 X의 자식노드들을 할당
  - graph의 구조를 참고하세요.
- (4) closed에 X를 추가
  - hint: extend()
- (5) open과 closed 리스트를 확인하여 중복된 child를 제거
  - for문과 if문을 사용
- (6) 남은 children을 open 리스트 오른쪽에 추가
  - hint: extend()



# Best First Search



**open** = {START}, **closed** =  $\emptyset$

while (**open**  $\neq \emptyset$ )

remove leftmost state from **open**  $\rightarrow X$

if ( $X$  is GOAL) return (success)

else

generate and evaluate children of  $X$

put  $X$  into **closed**

for each child  $C$

if  $C$  is in **open** update path

if  $C$  is in **closed** and reached by shorter path

remove  $C$  from **closed**, put into **open**

else

put  $C$  into **open**

reorder **open**(priority queue)

return (fail)

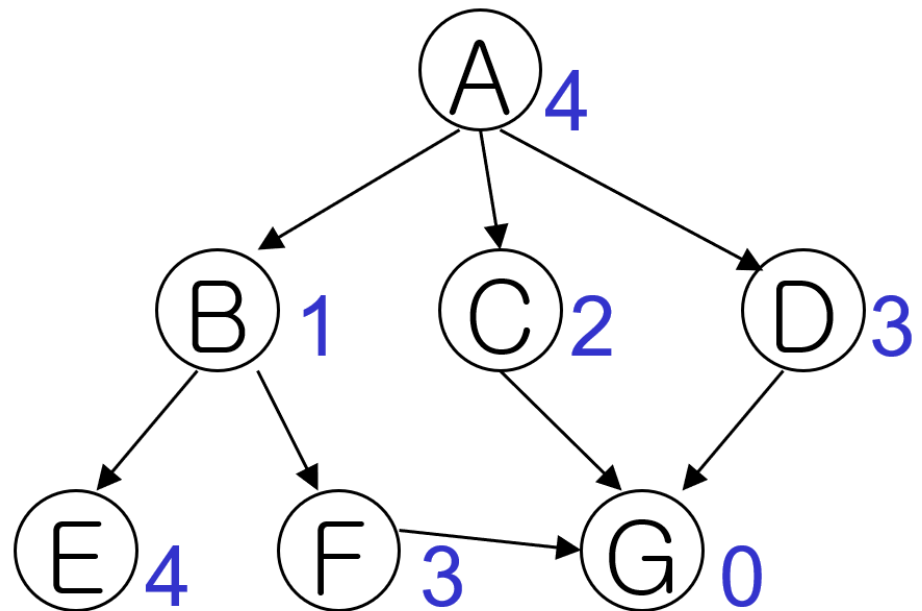
# Best First Search 구현 실습

```
from copy import deepcopy

graph = {'A': ['B', 'C', 'D'],
        'B': ['E', 'F'],
        'C': ['G'],
        'D': ['G'],
        'E': [],
        'F': ['G'],
        'G': []}

f = {'A': 4,
     'B': 1,
     'C': 2,
     'D': 3,
     'E': 4,
     'F': 3,
     'G': 0}

def print_list(X, open_list, closed, f):
    f_list = []
    print("-----")
    print("X =", X)
    print('open :', open_list)
    if len(open_list) != 0:
        for state in open_list:
            f_list.append(f[state])
    print('f_value :', f_list)
    print('closed :', closed)
```



# Best First Search 구현 실습

```
while(open_list):
    #remove leftmost state from open -> X
    (1)

    if (2):
        print_list(X, open_list, closed_list, f)
        return "*** Success ***"
    else:
        #generate and evaluate children of X
        (3)

        #put X into closed
        closed_list.extend(X)

        sorted_children = sorted(children, key=f.get)
        for (4):
            #if C is in open    update path
            if child in open_list:
                # In this problem, we don't need to use new value
                # because, all nodes have only one value
                print("update child from open list: ", child)

            #if C is in closed and reached by shorter path
            #remove C from closed, put into open
            if child in closed_list:
                # In this problem, we don't need to use new value
                # because, all nodes have only one value
                print("update child from closed list", child)

            #put C into open
            else:
                (5)

        #reorder open(priority queue)
        open_list = (6)

    print_list(X, open_list, closed_list, f)
```

- (1) 가장 왼쪽에 있는 State를 X에 할당
  - hint: pop()
- (2) X와 GOAL이 같을 때를 표현
- (3) children에 X의 자식노드들을 할당
  - graph의 구조를 참고하세요.
- (4) sorted\_children의 각 child마다 반복문 실행
  - for-in문 사용
- (5) closed에 X를 추가
  - hint: extend()
- (6) open\_list를 재정렬
  - hint: sorted\_children을 참조

# A\* Search

- A search algorithm is admissible
  - If it is **guaranteed to find a minimal path** to a solution whenever such a path exist
- A\* algorithm
  - A best-first search with

$$f(n) = g(n) + h(n)$$

where  **$h(n) \leq h^*(n)$**

➡ *Admissible !*

( $h^*(n)$ ) : actual cost (distance) from n to G)

# A\* Search 구현 실습

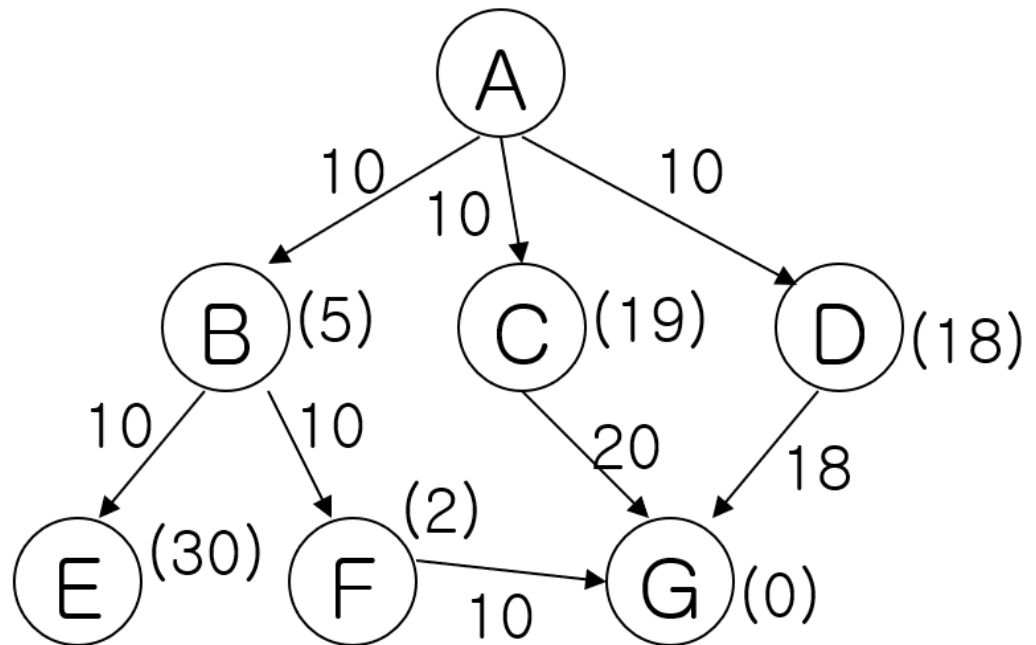
```
from copy import deepcopy

graph = {'A': ['B', 'C', 'D'],
        'B': ['E', 'F'],
        'C': ['G'],
        'D': ['G'],
        'E': [],
        'F': ['G'],
        'G': []}

g = {'A': {'A':0},
     'B': {'A':10},
     'C': {'A':10},
     'D': {'A':10},
     'E': {'B':20},
     'F': {'B':20},
     'G': {'F':30, 'C':30, 'D':28}}

h = {'A': 15,
     'B': 5,
     'C': 19,
     'D': 18,
     'E': 30,
     'F': 2,
     'G': 0}

f = {}
```



# A\* Search 구현 실습

```
while(open_list):
    #remove leftmost state from open -> X
    (1)
    if (2):
        print_list(X, open_list, closed_list, f)
        return "*** Success ***"
    else:
        #generate and evaluate children of X
        (3)
        #put X into closed
        (4)
        #update f by g, h
        for child in children:
            f[child] = (5)
        sorted_children = sorted(children, key=f.get)
        for child in sorted_children:
            #if C is in open update path
            if child in open_list:
                print("update child from open list: ", child)
                f[child] = (5)
            #if C is in closed and reached by shorter path
            #remove C from closed, put into open
            if child in closed_list:
                print("update child from closed list", child)
                f[child] = (5)
            #put C into open
            else:
                if child not in open_list:
                    (6)
            #reorder open(priority queue)
            (7)
        print_list(X, open_list, closed_list, f)
```

- (1) 가장 왼쪽에 있는 State를 X에 할당
  - hint: pop()
- (2) X와 GOAL이 같을 때를 표현
- (3) children에 X의 자식노드들을 할당
  - graph의 구조를 참고하세요.
- (4) closed에 X를 추가
- (5) f를 update
  - $f = g + h$
- (6) open\_list에 포함되지 않은 각 child를 추가
- (7) open\_list를 재정렬
  - hint: sorted\_children을 참조