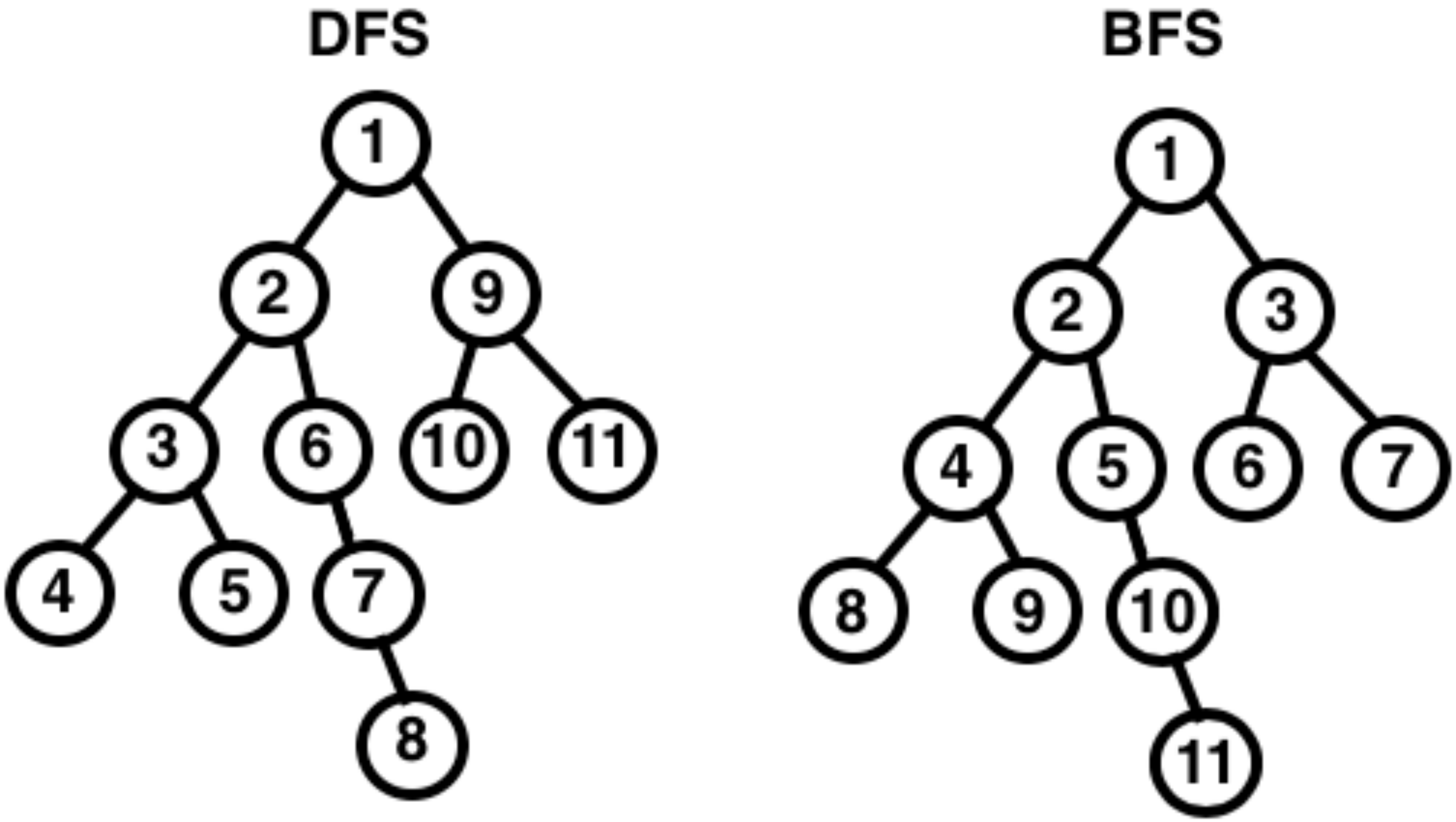
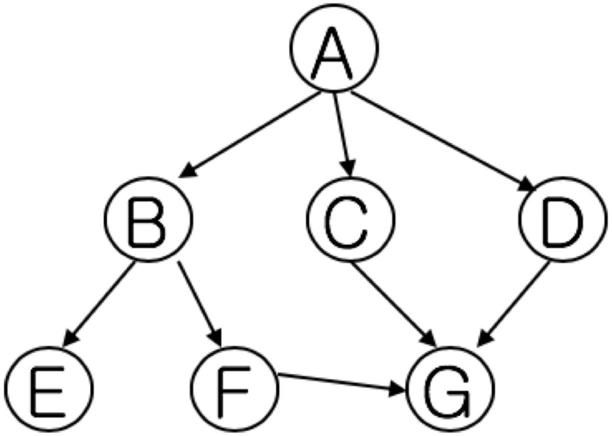
Search

Practice 2

DFS & BFS



Depth First Search



open = {START}, **closed** = Ø
while (open ≠ Ø)
    remove leftmost state from **open** → 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

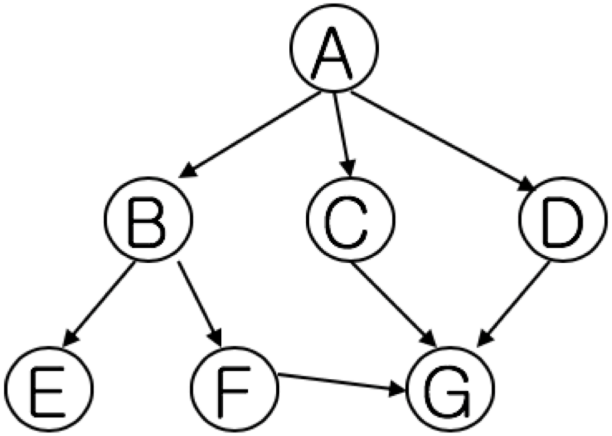
```
from copy import deepcopy

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** = Ø

while (open ≠ Ø)

  remove leftmost state from **open** → 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

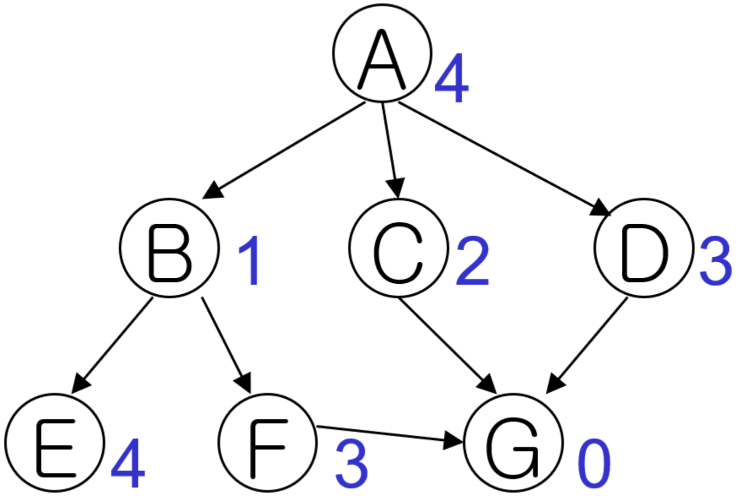
```python
from copy import deepcopy

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()

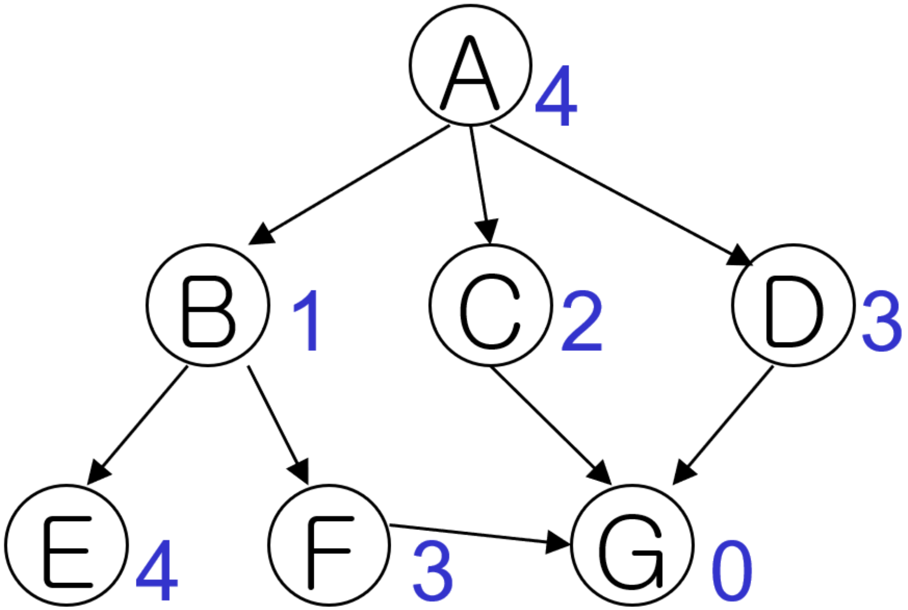
Best First Search



open = {START}, closed = Ø
while (open ≠ Ø)
    remove leftmost state from open → 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

```python
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

```python
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) open에 C를 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

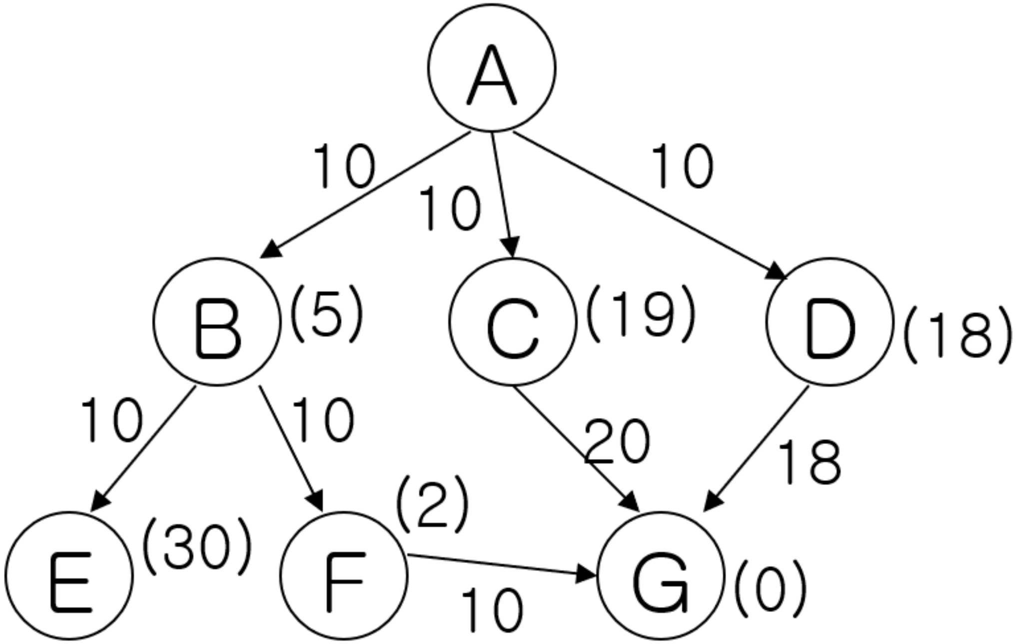
$$f(n) = g(n) + h(n)$$
$$\text{where } \mathbf{h(n) \leq h^*(n)}$$

⇒ *Admissible* !

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

A* Search 구현 실습

```python
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

```python
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

Best-First Search & A* Search

- 1. 아래의 표는 미국 동부의 저가 항공사인 Colgan Airline에서 운항하는 도시들과 각 도시들 사이의 직선 거리를 나타낸다. 숫자에 * 표가 있는 것은 두 도시 사이에 직항이 있다는 것을 의미한다. 직항이 있지 않은 경우 두 도시의 비행 거리는 두 도시를 연결하는 경로상의 직항 거리의 총 합이 된다. 다음의 각 평가 함수를 사용하여 **best-first search를 수행**하였을 경우 **Rockland, ME** 에서 **Keene, NH** 로 가는 경로를 찾는 탐색 과정을 **tree로 표현**하고 **그 결과로 얻어지는 경로를 구하시오.**
  - (g(n): 시작상태에서 n까지의 거리, h(n): n에서 목표상태까지의 추정거리(직선거리))

- a. $f(n) = g(n)$
- b. $f(n) = g(n) + h(n)$

Best-First Search & A* Search

	Aug	Bar	Bec	Blu	Bos	Ch NC	Ch VA	Hya	Kee	Lag	Man	Nan	New	Roc	Rut	DC
Augusta, ME	0	78	745	770	149*	858	626	185	187*	328	552	210	333	36	166	529
Bar-Harbor,ME	78	0	814	838	200	920	690	216	227	389	616	235	394	49*	243	592
Beckley,WV	745	814	0	35*	621	177	149	637	588	434	212	638	427	765	589	238*
Bluefield,WV	770	838	35*	0	643	142*	159	656	613	454	228	655	448	789	616	254*
Boston,MA	149*	200	621	643	0	720	492	63*	74	190	418	89	196	155*	130	394
Charlotte,NC	858	920	177	142*	720	0	234	722	702	531	306	716	525	873	714	329
Charlottesville,VA	626	690	149	159	492	234	0	501	469	302*	74*	498	296	642	481	98
Hyannis,MA	185	216	637	656	63*	722	501	0	135	204*	429	27*	212*	179	192	403
Keene,NH	187*	227	588	613	74	702	469	135	0	177	396	159	180*	178	58*	373
La-Guardia,NY	328	389	434	454	190	531	302*	204*	177	0	229	207*	9	342	207	204
Manassa,VA	552	616	212	228	418	306	74*	429	396	229	0	427	223	569	409	26*
Nantucket,MA	210	235	638	655	89	716	498	27*	159	207*	427	0	216*	201	217	402
Newyork,NJ	333	394	427	448	196	525	296	212*	180*	9	223	216*	0	347	208	198
Rockland,ME	36	49*	765	789	155*	873	642	179	178	342	569	201	347	0	195	545
Rutland,VT	166	243	589	616	130	714	481	192	58*	207	409	217	208	195	0	388
Washington,DC	529	592	238*	254*	394	329	98	403	373	204	26*	402	198	545	388	0