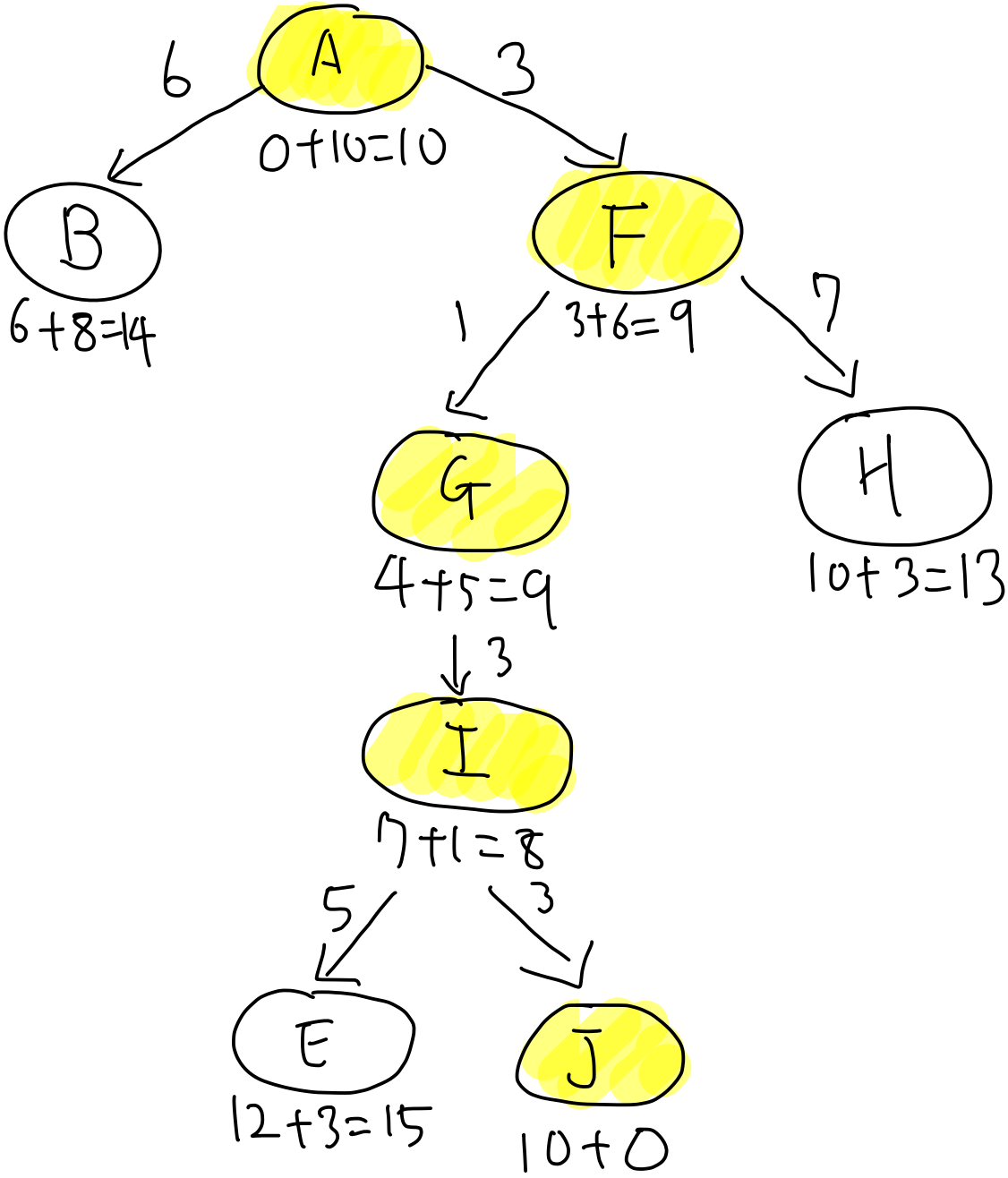
(1)



(2) A → F → G → I → J

(3) 출발지점은 A노드이고 도착 지점이 J노드일때 A지점에서 갈 수 있는 노드는 B노드와 F노드뿐이며 B노드는 6+8=14, F노드는 3+6=9의 가중치를 갖는다.
가중치가 작은 F노드를 선택한다. F노드는 G노드와 H노드를 갈 수 있다. G노드는 4+5=9, H노드는 10+3=13의 가중치를 갖는다.
선택되지 않는 노드 B, G, H 중 가중치가 가장 낮은 G노드를 선택한다. G노드는 I노드만 갈 수 있다. I노드의 가중치는 7+1=8이다.
선택되지 않는 노드 B, I, H 중 가중치가 가장 낮은 I노드를 선택한다. I노드가 갈 수 있는 노드는 E, J뿐이고 E노드의 가중치는 12+3=15, J노드의 가중치는 10+0=10 갖인다.
선택되지 않는 노드 B, E, J, H 중 가중치가 가장 낮은 J노드를 선택되고 J노드는 도착 지점이서 A->F->G->I->J가 최소 경로이다.

(4)

[37]
```
#가중치 리스트
weight = {
    'A':{'B':6, 'F':3},
    'B':{'A':6, 'C':3, 'D':2},
    'C':{'B':3, 'D':1, 'E':5},
    'D':{'B':2, 'C':1, 'E':8},
    'E':{'C':5, 'D':8, 'I':5, 'J':5},
    'F':{'A':3, 'G':1, 'H':7},
    'G':{'F':1, 'I':3},
    'H':{'F':7, 'I':2},
    'I':{'E':5, 'G':3, 'H':2, 'J':3},
    'J':{'E':5, 'I':3}
}

#현재노드에서 목적지까지 추정거리 리스트
dis = {
    'A':10, 'B':8, 'C':5, 'D':7, 'E':3, 'F':6, 'G':5, 'H':3, 'I':1, 'J':0
}
```

[43]
```
class Node:
    def __init__(self, parent=None, position=None):
        self.parent = parent #이전 노드 주소
        self.position = position # 현재위치

        self.g = 0 #실제비용
        self.h = 0 #목표까지 예측비용
        self.f = 0 #총 비용

    def print(self):
        print("parent = ", self.parent)
        print("position = ", self.position)
```

[54]
```
start = 'A'
end = 'J'
path = aStar(start, end)
print(path)
```

```
['A', 'F', 'G', 'I', 'J']
```

```
def aStar(start, end):
  #startNode와 endNode 초기화
  startNode = Node(None, start);
  endNode = Node(None, end)

  #openList, closeList 초기화
  openList = []
  closeList = []

  #openList에 시작 노드 추가
  openList.append(startNode)

  #endNode를 찾을 때까지 실행
  while openList:
    #현재 노드 지정
    currentNode = openList[0]
    currentIdx = 0

  #이미 같은 노드가 opnList에 있고, f 값이 더크면 currentNode를 openList안에 있는 값으로 교체
    for index, item in enumerate(openList):
      if item.f < currentNode.f:
        currentNode = item
        currentIdx = index

  # openList에서 제거하고 closeList에 추가
    openList.pop(currentIdx)
    closeList.append(currentNode)

  #현재 노드가 목적지면 current.position 추가하고 current의 부모로 이동
    if currentNode.position == endNode.position:
      path = []
      current = currentNode
      while current is not None:
        path.append(current.position)
        current = current.parent
      return path[::-1] #reverse

    children = []
    for key, value in weight[currentNode.position].items():
      new_node = Node(currentNode, key)
      children.append(new_node)

    # 자식들 모두 loop
    for child in children:
      #자식 closeList에 있으면 continue
      if child in closeList:
        continue

      #f,g,h값 업데이트
      child.g = currentNode.g + weight[currentNode.position][child.position]
      child.h = dis[child.position]
      child.f = child.g + child.h

      if len([openNode for openNode in openList if child == openNode and child.g > openNode.g]) > 0:
        continue

      openList.append(child)
```

참고자료: https://choiseokwon.tistory.com/210