Problem 1

Constraint satisfaction problems(CSPs) are defined with a set of variables or objects whose state must satisfy a number of constraints. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which is solved by constraint satisfaction algorithms.

What is Constraint satisfaction problems?

- 제약 충족 문제는 복수의 제약 조건을 충족하는 상태를 찾아내는 수학 문제를 가리킨다. CSP는 특히 인공지능이나 운용 과학 분야에서 심도 있게 연구되고 잇다. 보통 CSP는 고도의 복잡성을 보이기 때문에 적정한 시간 내에 문제를 풀기 위해서는 휴리스틱과 조합 최적화 기법을 조합할 필요가 있다

(a) Give two or three realistic examples around your environment which implement CSPs.

i) 스도쿠 : 숫자 퍼즐로, 가로 9칸, 세로 9칸으로 이루어져 있는 표에 1부터 9까지의 숫자를 채워 넣는 퍼즐이다.

ii) 가로세로퍼즐 : 십자가 모양으로 배치된 네모 칸에 단어를 맞히는 퍼즐 게임이다.

(b) Formulate the above mentioned applications in terms of goals(or objective function) and constraints.

i) 스도쿠 :

규칙.
1. 각 가로줄에는 1~9까지의 숫자가 겹치지 않도록 한번 들어간다.
2. 각 세로줄에는 1~9까지의 숫자가 겹치지 않도록 한번 들어간다.
3. 각 3×3 box 안에는 1~9까지의 숫자가 겹치지 않도록 한번 들어간다.

2	4	5	6		1		3	8
1		3	4		6		8	9
	7							

풀이.

1. 3행에서 노란색 cell 중 하나에 7이 들어가야 한다.
2. 6행에서 노란색 cell 중 하나에 7이 들어가야 한다.
3. 만약 (3, 5)가 7이면, (6, 7)이 7이 되고, (3, 7)이 7이면 (6, 5)가 7이 된다.
4. 결국 5열과 7열의 7은 (3, 5), (6, 7), (3, 7), (6, 5) 중에 두 개가 전부 포함 되게 하고, 따라서 나머지 칸들인 파란색 칸에는 숫자 7이 들어갈 수 없다.

- 위에 같은 방법을 X-Wing 이라고 한다. 이외에도 Single, Naked Pair, Hidden Pair, Sword Fish, Jelly Fish, Finned X-Wing, Sashimi X-Wing, Chain 등을 이용한 풀이가 있다.

ii) 십자말풀이 : 십자말풀이는 딱히 공식이라는 것이 없다. 문제를 보고 빈칸에 답을 채우는 게임이다. 또한 다른 답과도 조화를 이루도록 문제를 풀어야 한다. 따라서 겹치는 단어들은 같은 글자를 포함한다.

Problem 2

Graph coloring problems and applications were taught, in the previous lecture. As we discussed, graph coloring is widely used. But, there is no efficient algorithm available for coloring a graph with minimum number of colors as the problem is a known NP-complete problem. There are approximate algorithms to solve the graph coloring problem.

(a) Google NP-complete problems and describe it on your word.

NP-complete는 NP 집합에 속하는 결정 문제 중에서 가장 어려운 문제의 부분집합으로, 모

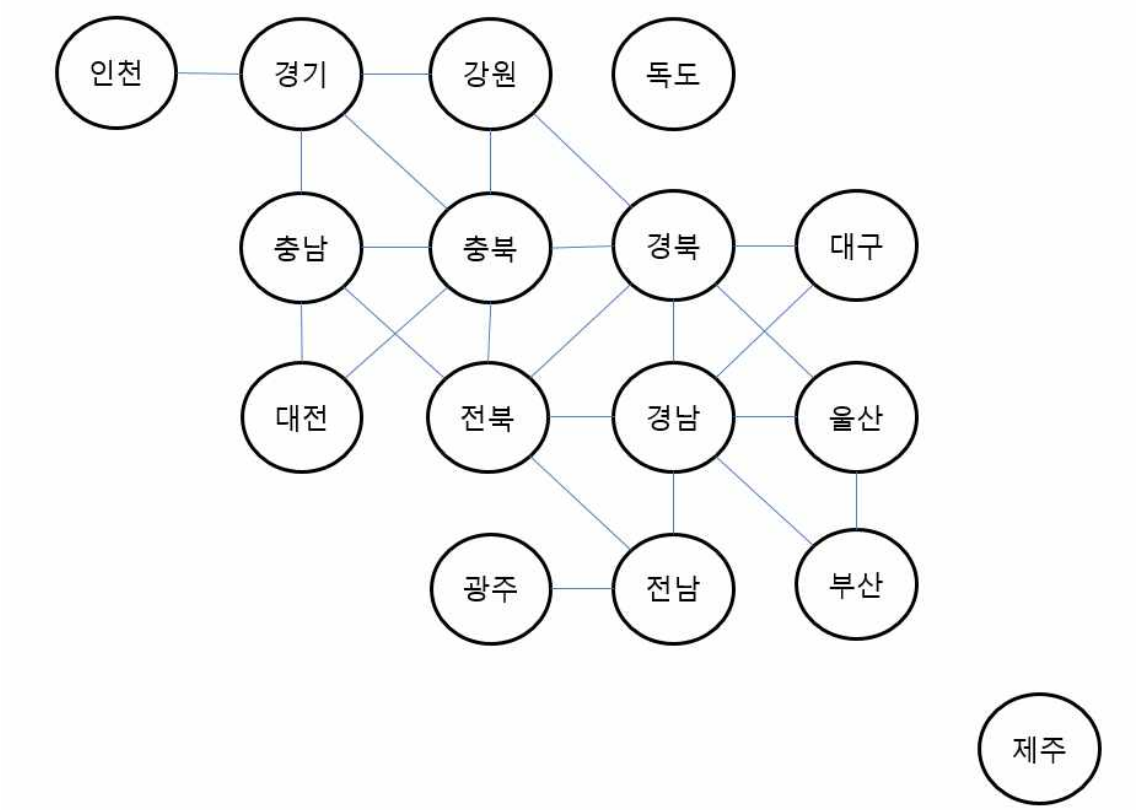
든 NP 문제를 다항 시간 내에 NP-complete problem로 환산할 수 있다. NP-complete problem중 하나라도 P에 속한다는 것을 증명한다면 모든 NP 문제가 P에 속하기 때문에, P-NP problem이 P-NP의 형태로 풀리게 된다. 반대로 NP-complete problem중 하나가 P에 속하지 않는다는 것이 증명 된다면 P=NP에 대한 반례가 되어 P-NP problem은 P≠NP의 형태로 풀리게 된다.

NP-complete는 다음의 조건을 만족하는 결정 문제 C의 집합이다.

1. C가 NP에 속한다.
2. NP에 속하는 모든 문제를 다항 시간 안에 C로 변환할 수 있다. 즉 NP-complete는 NP-Hard 중 NP인 문제들의 집합이다. NP-complete인 C를 다항시간 안에 풀 수 있다면 모든 NP-complete problem을 다항시간 안에 풀 수 있다.

NP-complete problem example로는 해밀턴 경로 문제, 충족 가능성 문제(SAT), 외판원 문제, 그래프 색칠 문제, 시간표 짜기 등이 있다.

(b) Given the Korea administrative region map(Figure 1), convert the map into a graph.



(c) Find the minimum number of colors for the Korea map.

- **Design your algorithm in pseudo code.**

```
colors = []
count=1
color_count=1
states = ['서울', '인천', '경기', '강원', '경북', '울산', '충북', '전북', '대구',
          '충남', '대전', '경남', '부산', '광주', '전남', '독도', '울릉도', '제주도']

neighbors = {}
neighbors['서울'] = ['인천', '경기']
neighbors['인천'] = ['서울', '경기']
neighbors['경기'] = ['서울', '인천', '강원', '충북', '충남']
neighbors['강원'] = ['경기', '충북', '경북']
neighbors['경북'] = ['강원', '충북', '전북', '대구', '울산']
neighbors['울산'] = ['경북', '경남', '부산']
neighbors['충북'] = ['경기', '충남', '대전', '전북', '경북', '강원']
neighbors['부산'] = ['울산', '경남']
neighbors['경남'] = ['대구', '전북', '전남']
neighbors['전남'] = ['광주', '전북']
neighbors['전북'] = ['경북', '충북', '충남', '경남','전남']
neighbors['광주'] = ['전남']
neighbors['대구'] = ['경북', '경남']
neighbors['충남'] = ['경기', '충북', '대전', '전북']
neighbors['대전'] = ['충남', '충북']
neighbors['독도'] = [' ']
neighbors['제주도'] = [' ']
neighbors['울릉도'] = [' ']

colors_of_states = {}

def findcolors(state, color):
    global count
    global color_count
    for neighbor in neighbors.get(state):
        color_of_neighbor = colors_of_states.get(neighbor)
        print(neighbor,"과(와)",state,"의 색:",color_of_neighbor, color)
        if color_of_neighbor == color:
            color_count += 1
            if color_count == count:
                    color_count += 1
                    count = color_count
                    print("나는 카운트야 ~", count)
```

```
                    colors.append(count)
                    color_count = 0
                    return False
            else :
                print("else 문 안이야~")
                return False
    color_count = 0
    return True

def get_color_for_state(state):
    for color in colors:
        print("input :",state)
        if findcolors(state, color):
            print(colors)
            print(color)
            print(state)
            return color

if __name__ == '__main__':
    colors.append(1)

    for state in states:
        colors_of_states[state] = get_color_for_state(state)

    print(colors_of_states)
    print("Minimum color Number :", count)
    print(colors)
```

(d) Implement your algorithm to find the minimum number of colors.

- **Submit your code and results.**
- **Discuss about how to analyze the space and time complexity of your own algorithm.**

Figure 1: South Korea administrative region map

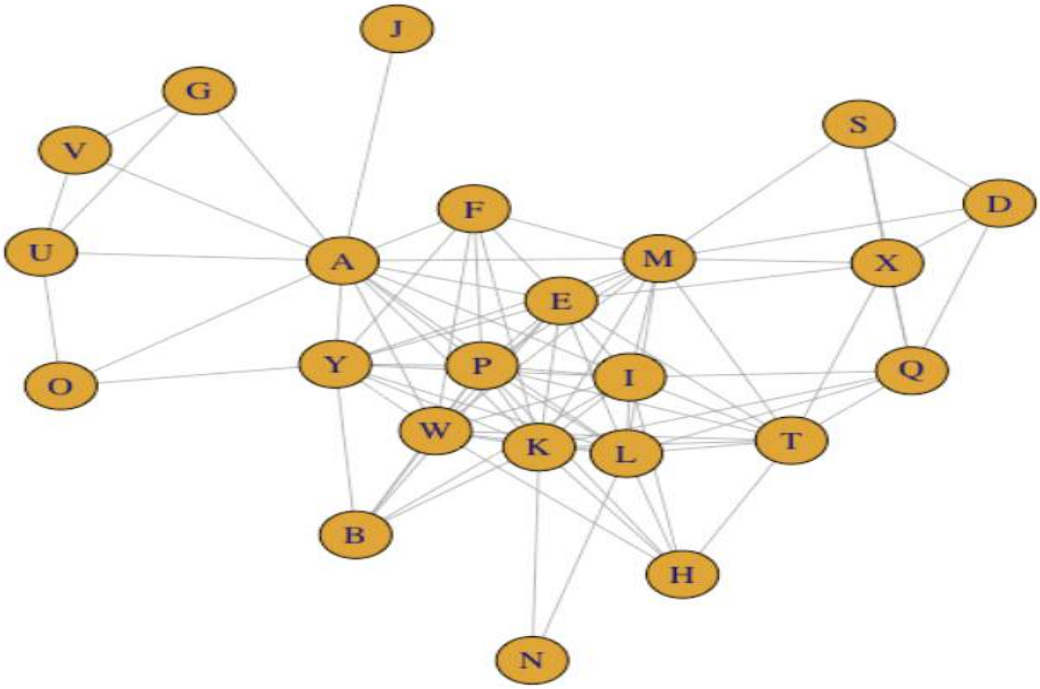
Figure 2: A undirected graph

Problem 3

For the following graph in Figure 2, answer the questions:

(a) Report the order of the vertices encountered on a breadth-first search starting from vertex A. Choose the vertices in alphabetical order.

(b) Report the order of the vertices encountered on a depth-frist search starting from vertex A. Choose the vertices in alphabetical order.



Problem 4

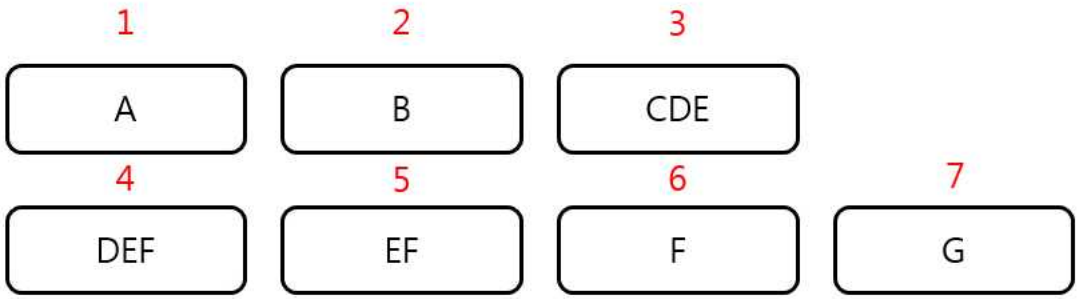
Propose a PEAS description of the task of designing an automated taxi driver. PEAS is an abbreviation of Performance measure, Environment, Actuators, and Sensors.

- Performance measure : Safe, Fast, Legal, Comfortable trip, Maximize profits.
- Environment : Roads, other traffic, pedestrians, customers.
- Actuators : Steering wheel, Accelerator, Brake, Signal, Horn.
- Sensors : Cameras, Sonar, Speedometer, GPS, Odometer, Engine sensors, Keyboard.

Problem 5

Find the solution of the graph in Figure 3. The edge weight is distance between two nodes and the weight below a node is the cost to node G.

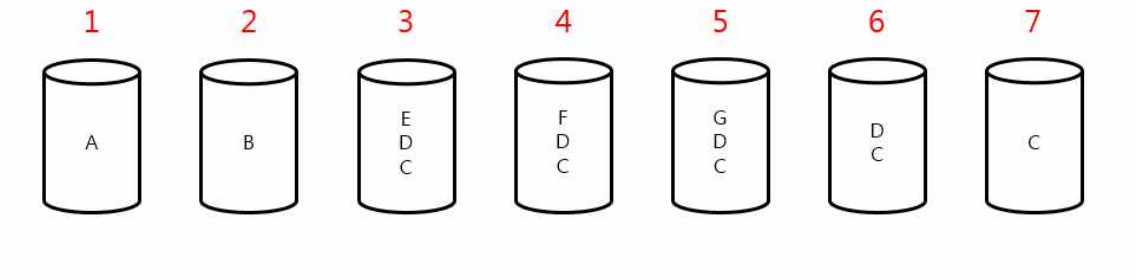
(a) When we start at node A and ignore the given cost values, find the breadth-first search of the graph.



-Queue-

A-B-C-D-E-F-G

(b) When we start at node A and ignore the given cost values, give the depth-first search of the graph.



-Stack-

A-B-E-F-G-D-C

(c) Find the optimal solution from A to G by applying the A* algorithm.

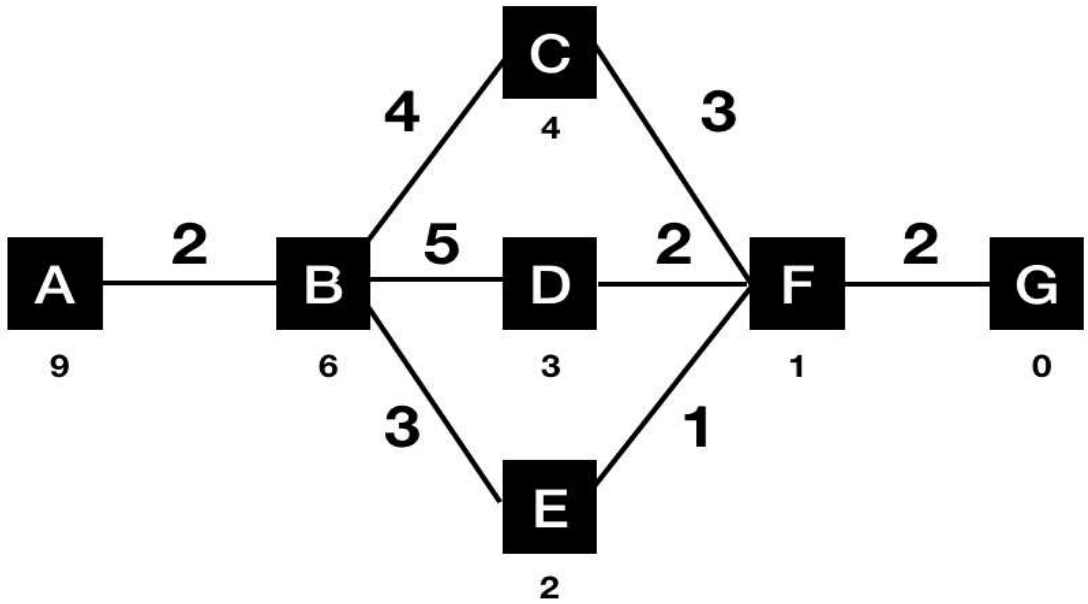


Figure 3: A graph example

Problem 6

For any event x and y in a sample space, prove

$$P(x|y)P(y) = P(y|x)P(x).$$

$P(x|y)P(y) = P(y|x)P(x)$
$P(x|y) = \frac{P(x \quad y)}{P(y)}\, P(y|x) = \frac{P(y \cap x)}{P(x)}$ 다.
x와 y가 똑같은 공간에서 $P(x \cap y) = P(y \cap x)$이다.
$\therefore P(x|y)P(y) = P(x \cap y) = P(y \cap x) = P(y|x)P(x)$

Problem 7

Given the discrete probability distribution in Table 1, answer the questions.

(a) Compute P(W).

P(W)	
W	P
SUN	0.6
RAIN	0.4

(b) Compute P(W|S=winter).

P(WIS=WINTER)		
S	W	P
WINTER	RAIN	0.625
	SUN	0.375

(c) Compute P(W|S=winter, T=cold).

P(WIS=WINTER T=COLD)			
S	T	W	P
WINTER	COLD	RAIN	0.67
		SUN	0.33

Table 1: A discrete probability distribution

S	T	W	P
summer	hot	sun	0.3
summer	hot	rain	0.05
summer	cold	sun	0.15
summer	cold	rain	0.1
winter	hot	sun	0.05
winter	hot	rain	0.05
winter	cold	sun	0.1
winter	cold	rain	0.2