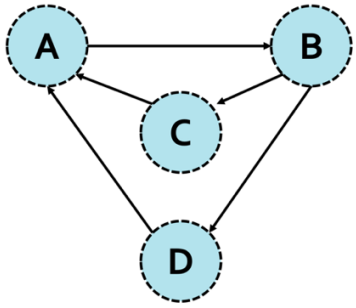
# Assignment 4

*Advanced Programming (INFO135)*

**Published at:** 13:00, Friday, 11.03.2022
**Deadline:** 13:00, Friday, 18.03.2022

---

1. Given the following Graph, which set represents the Edge set of the Graph?




a) { (A,B), (B,C), (B,D), (C,A), (C,D) }
b) { (A,B), (B,C), (B,D), (C,A), (D,A) }
c) { (A,B), (B,C), (C,B), (C,A), (D,A) }
d) { (A,B), (B,C), (A,C), (C,A), (D,B) }
e) None of the above

---

2. Extend the implementation of the solver for the N-queen problem by adding a new function called **`is_solution()`** that receives as a parameter a candidate solution and checks if the solution is valid (correct) or not. The candidate solution is in the form of a **list of string**, indicating the position of the queens in the chessboard.

   **Note 1:** You can find an example implementation of solver for N-queen problem in lecture 5.
   **Note 2:** You can assume N=5.

   You can change other functions if needed. Here is an example output for two candidate solutions.

```
COLUMNS = "abcde"
NUM_QUEENS = len(COLUMNS)
ACCEPT = 1
CONTINUE = 2
ABANDON = 3
all_solutions = []
```

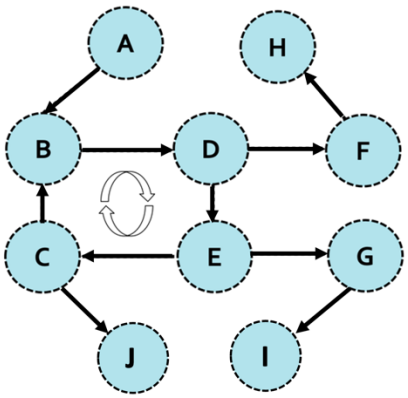
```
def solve(partial_sol):...

def examine(partial_sol):...

def attacks(p1, p2):...

def extend(partial_sol):...

def is_solution(candidate_solution):
      [your code here]


candidate_solution1 = ['d3', 'c1', 'e5', 'b4', 'a2']
candidate_solution2 = ['e4', 'a1', 'c5', 'd2', 'b1']

result1 = is_solution(candidate_solution1)
result2 = is_solution(candidate_solution2)

print("Candidate Solution 1:", result1)
print("Candidate Solution 2:", result2)
```

**[Output]:**

```
Candidate Solution 1: Valid!
Candidate Solution 2: Invalid!
```

---

3. Cycle in a Graph is a path that starts from a vertex (node) and ends at the same vertex. The Graph in the following figure shows a cycle.




Use the **`Graph`** class in Lecture notes 7, write a new method called **`find_cycle(node)`** that receives a node as a parameter and traverses the Graph starting from the given node.

The method should print "Cycle found!" if it detects a cycle in the Graph and "Cycle not found!", otherwise. You can use either BFS or DFS algorithm.

```
my_graph = Graph()
my_graph.add_edge('A', 'B')
my_graph.add_edge('B', 'D')
my_graph.add_edge('C', 'B')
my_graph.add_edge('C', 'J')
my_graph.add_edge('D', 'E')
my_graph.add_edge('D', 'F')
my_graph.add_edge('E', 'C')
my_graph.add_edge('E', 'G')
my_graph.add_edge('F', 'H')
my_graph.add_edge('G', 'I')


result = my_graph.find_cycle('A')
print(result)
```

**[Output]**

```
Cycle found!
```