TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA Công Nghệ Thông Tin

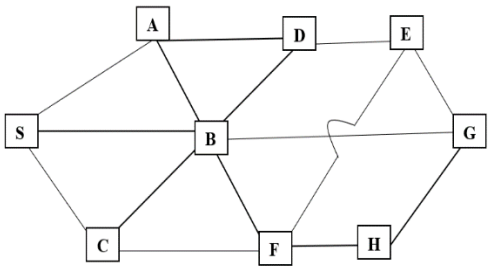
BỘ MÔN: Công Nghệ Phần Mềm
ĐỀ THI VÀ BÀI LÀM

Tên học phần: Trí tuệ nhân tạo
Mã học phần: Hình thức thi: *Tự luận có giám sát*
Đề số: **01** Thời gian làm bài: 70 phút *(không kể thời gian chép/phát đề)*
Được sử dụng tài liệu khi làm bài.

Họ tên: …Lê Hữu Long……**Lớp**:…18TCLC_ĐT3 ……**MSSV**:……102180213…

Sinh viên làm bài trực tiếp trên tệp này, lưu tệp với định dạng MSSV_HọTên.pdf và nộp bài thông qua MSTeam:

Câu 1 (*5 điểm*): Cho đồ thị vô hướng G = (V,E) như hình vẽ với V là tập đỉnh và E là tập cạnh.



a) *(2 điểm)* Hãy viết đoạn code biểu diễn đồ thị trên bằng cách khởi tạo tập đỉnh V và tập cạnh E. (Ví dụ: V = [“S”, “A”, “B”], E =[(“S”, “A”), (“S”, “B”)])

 # Trả lời: Dán code vào bên dưới

```
class Node:
    def __init__(self, data):
        self.data = data
        self.parents = []
        self.children = []

    def get_data(self):
        return self.data

    def get_children(self):
        return [node.get_data() for node in self.children]

    def get_parents(self):
        return [node.get_data() for node in self.parents]

class Tree:
    def __init__(self):
        self.nodes = []
        self.edges = []

    def clear(self):
```

```
        self.nodes = []
        self.edges = []

    def number_of_nodes(self):
        return len(self.nodes)

    def number_of_edges(self):
        return len(self.edges)

    def get_index(self, node):
        for idx, n in enumerate(self.nodes):
            if n.get_data() == node.get_data():
                return idx
        return -1

    def add_node(self, node_name):
        node = Node(node_name)
        if not self.is_contains(node):
            self.nodes.append(node)

    def add_node_from(self, array_of_nodes_name):
        for el in array_of_nodes_name:
            node = Node(el)
            if not self.is_contains(node):
                self.nodes.append(node)

    def is_contains(self, node):
        for el in self.nodes:
            if el.get_data() == node.get_data():
                return True
        return False

    def add_edge(self, start_name, end_name):
        start_node = Node(start_name)
        end_node = Node(end_name)
        if not self.is_contains(start_node):
            self.add_node(start_name)
        if not self.is_contains(end_node):
            self.add_node(end_name)
        start_index = self.get_index(start_node)
        end_index = self.get_index(end_node)
        self.nodes[start_index].children.append(end_node)
        self.nodes[end_index].parents.append(start_node)
        self.edges.append((self.nodes[start_index], self.nodes[end_index]))

    def add_edges_from(self, array_of_tuple_node):
        for tup in array_of_tuple_node:
            start = tup[0]
            end = tup[1]
            self.add_edge(start, end)

    def show_nodes(self):
```

```
        return [node.get_data() for node in self.nodes]

    def show_edges(self):
        return [(edge[0].get_data(), edge[1].get_data()) for edge in self.edges]
```

```
if __name__ == "__main__":
    tree = Tree()
    # G = nx.Graph()
    tree.add_node("S")
    tree.add_node_from(["S", "A", "B", "C", "D", "E", "F", "G", "H"])
    print(tree.show_nodes())
    tree.add_edges_from(
        [
            ("S", "A", 1),
            ("S", "B", 1),
            ("S", "C", 1),
            ("A", "D", 1),
            ("A", "B", 1),
            ("B", "C", 1),
            ("B", "D", 1),
            ("B", "F", 1),
            ("B", "G", 1),
            ("C", "F", 1),
            ("D", "E", 1),
            ("E", "F", 1),
            ("E", "G", 1),
            ("F", "H", 1),
            ("H", "G", 1),
        ]
    )
    result = Breadth_First_Search(tree,"S", "G")
    print(result)
```

b) *(3 điểm)* Hãy viết chương trình sử dụng thuật toán **tìm kiếm theo chiều rộng (BFS)** để tìm đường đi từ đỉnh "S" đến đỉnh "G" trong đồ thị được biểu diễn ở câu a). Trong chương trình, hãy in ra thứ tự đỉnh khám phá trong quá trình tìm kiếm. Nếu không tìm thấy thì in "*Khong tim thay duong di*"

 # Trả lời: Dán code vào bên dưới

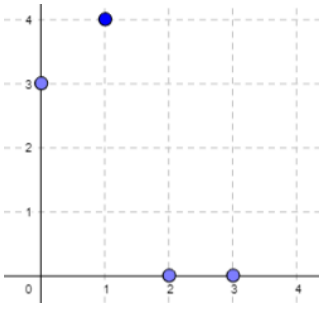
```
def Breadth_First_Search(tree, initialState, goalTest):
    # start_node = Node(initialState)
    # end_node = Node(destinationState)
    frontier = []
    frontier.append(initialState)
    explored = []
    while len(frontier) > 0:
        print("Frontier >> ", frontier)
```

```
        state = frontier.pop(0)
        state_node = Node(state)
        explored.append(state)
        # print("Explored >> ", explored)
        if goalTest == state:
            print("Da tim thay")
            return True
        index_state = tree.get_index(state_node)
        for neighbor in tree.nodes[index_state].get_children():
            if neighbor not in list(set(frontier + explored)):
                frontier.append(neighbor)
    print("Khong tim thay duong di")
    return False
```

 # Trả lời: Dán kết quả thực thi vào bên dưới:

```
['S', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
Frontier >>  ['S']
Frontier >>  ['A', 'B', 'C']
Frontier >>  ['B', 'C', 'D']
Frontier >>  ['C', 'D', 'F', 'G']
Frontier >>  ['D', 'F', 'G']
Frontier >>  ['F', 'G', 'E']
Frontier >>  ['G', 'E', 'H']
Da tim thay
True
```

Câu 2 (2 *điểm*): Cho 4 tọa độ như hình và trả lời các câu hỏi sau:



a) *(1 điểm)* Mô tả thuật toán hoặc hàm thực thi thuật toán *k*-means

 # Trả lời: viết mô tả thuật toán hoặc dán code vào bên dưới

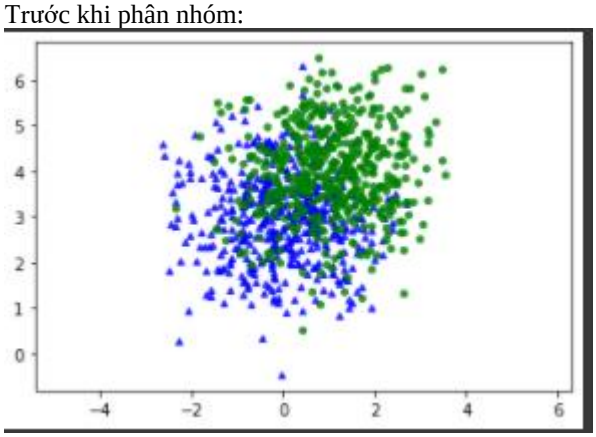
Đầu vào: Dữ liệu X và số lượng cluster cần tìm K.
Đầu ra: Các center M và label vector cho từng điểm dữ liệu Y.

1. Chọn K điểm bất kỳ làm các center ban đầu.
2. Phân mỗi điểm dữ liệu vào cluster có center gần nó nhất.
3. Nếu việc gán dữ liệu vào từng cluster ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.

4. Cập nhật center cho từng cluster bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào cluster đó sau bước 2.
5. Quay lại bước 2.

b) *(1 điểm)* Nếu sử dụng thuật toán *k*-means với k = 2 thì kết quả phân nhóm sẽ như thế nào? (các điểm thuộc mỗi nhóm, trọng tâm của mỗi nhóm).

 # Trả lời: viết câu trả lời vào bên dưới
Với K = 2
Trước khi phân nhóm:



Sau khi phân nhóm:
Ta có 2 điểm hội tu của 2 loại và phân bố các điểm như sau
(0.5, 3.5) ; (2.5, 0)

.

Câu 3(3 *điểm*): Cho hàm f(*x*) = $\left(1 - \frac{2}{e^x}\right)^2$, hãy viết chương trình tìm giá trị nhỏ nhất nhỏ nhất của f(x) sử dụng thuật toán Gradient Descent Method

Trả lời: Dán code vào bên dưới

```
import math
def f(x):
  return (1-2/math.exp(x))**2

def df(x):
  return 4*math.exp(-x) - 8*math.exp(-2*x)

cur_x = 20
rate = 0.01
precision = 0.000001
previous_step_size = 1
```

```
max_iters = 10000
iters = 0

while previous_step_size > precision and iters < max_iters:
    prev_x = cur_x
    cur_x = cur_x - rate * df(prev_x)
    previous_step_size = abs(cur_x - prev_x)
    iters = iters+1

print("The local minimum occurs at:", cur_x)
print("The minimum value is: ", f(cur_x))
```

Trả lời: Dán kết quả thực thi vào bên dưới:

```
The local minimum occurs at: 19.999999999917552
The minimum value is:  0.9999999917553855
```

Đà Nẵng, ngày 22 tháng 08 năm 2021

GIẢNG VIÊN BIÊN SOẠN ĐỀ THI

TRƯỞNG BỘ MÔN
(đã duyệt)