ĐỀ GIỮA KỲ THI TRÍ TUỆ NHÂN TẠO – 03/2022

(Thời gian 70 phút)

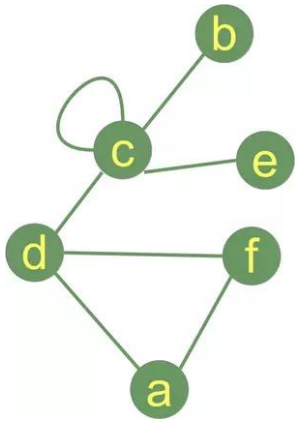
Họ tên:…Lý Thanh Hải……

Lớp:……19TCLC_DT2……

MSSV:…102190061……...

Làm bài trên file này, lưu bài với định dạng MSSV_HọTên.pdf và nộp bài thông qua MS Teams.

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 = [“a”, “b”, “c”], E =[(“a”, “d”), (“a”, “f”)])

Dán code vào bên dưới

```
def initialTop(numberTop):
    Tops = []
    for i in range(0, numberTop):
        Tops.append(input())
    return Tops

def initialEdge(numberEdge):
    Edges = []
    for i in range(0, numberEdge):
        edge1, edge2 = map(str, input().split(", "))
        Edges.append((edge1, edge2))
    return Edges

# bắt đầu khởi tạo tập đỉnh và tập cạnh để biểu diễn đồ thị
numberTop = int(input()) # số đỉnh V
numberEdge = int(input()) # số cạnh E
print("Các đỉnh là: ", initialTop(numberTop))
print("Các cạnh là: ", initialEdge(numberEdge))
```

b. (**3 điểm**) Hãy viết chương trình sử dụng thuật toán **tìm kiếm theo chiều sâu** để tìm đường đi từ đỉnh "a" đến đỉnh "b" trong đồ thị được biểu diễn ở câu A. Trong chương trình, hãy in ra cách di chuyển từ đỉnh "a" đến đỉnh "b" nếu tìm thấy. Nếu không tìm thấy thì in "*Khong tim thay duong di*"

Dán code vào bên dưới

```
def addEdge(u, v, graph):
    graph[u].append(v)

def printAllPathsUtil(u, d, visited, path):
    visited[u]= True
    path.append(u)

    if u == d:
        print (path)
    else:
        for i in graph[u]:
            if visited[i]== False:
                printAllPathsUtil(i, d, visited, path)

        path.pop()
        visited[u]= False


def printAllPaths(s, d, numberEdge):
    visited = [False] * (numberEdge)
    path = []
    printAllPathsUtil(s, d, visited, path)

#graph = ["a", "b", "c", "d", "e", "f"]
graph = {
    'a': [],
    'b': [],
    'c': [],
    'd': [],
    'e': [],
    'f': []
}
numberEdge = 7
for i in range(0, numberEdge):
    a, b = map(str, input().split(", "))
    addEdge(a, b, graph)

print("Nhập nguồn: ")
s = input()
print("Nhập đích: ")
d = input()
print(f"Tất cả các đường từ ${s} đến ${d}: ")
printAllPaths(s, d, numberEdge)
```

Dán kết quả thực thi vào bên dưới:

```
a, d
a, f
b, c
c, c
c, d
c, e
d, f
{'a': ['d', 'f'], 'b': ['c'], 'c': ['c', 'd', 'e'], 'd': ['f'], 'e': [], 'f': []}
Nhập nguồn:
a
Nhập đích:
b
Tất cả các đường từ $a đến $b:
```

2. (**5 điểm**) Hãy viết chương trình cờ Ca-Ro người chơi với máy. Biết máy được thực thi theo thuật toán Mini-max. Bàn cờ tối thiểu 10x10 ô, khi bên nào đánh liên tục 5 ô liên tiếp theo đường thẳng (ngang, dọc, chéo) thì bên đó sẽ thắng, kết thúc cuộc chơi.

Dán code vào bên dưới

```python
# Tic-Tac-Toe
import numpy as np
import random
from time import sleep

# Tạo bàn cờ rỗng
def create_board(n):
    return(np.zeros([n, n]))

# Kiểm tra danh sách còn rỗng
def possibilities(board):
    l = []

    for i in range(len(board)):
        for j in range(len(board)):

            if board[i][j] == 0:
                l.append((i, j))
    return(l)

# chọn ngẫu nhiên
def random_place(board, player):
    selection = possibilities(board)
    current_loc = random.choice(selection)
    board[current_loc] = player
```

```
    return(board)

# kiểm tra thắng theo cột
def row_win(board, player):
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[x, y] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)

# kiểm tra thắng theo dòng
def col_win(board, player):
    for x in range(len(board)):
        win = True

        for y in range(len(board)):
            if board[y][x] != player:
                win = False
                continue

        if win == True:
            return(win)
    return(win)

# kiểm tra thắng theo đường chéo
def diag_win(board, player):
    win = True
    y = 0
    for x in range(len(board)):
        if board[x, x] != player:
            win = False
    if win:
        return win
    win = True
    if win:
        for x in range(len(board)):
            y = len(board) - 1 - x
            if board[x, y] != player:
                win = False
    return win

# Đánh giá thắng thua
def evaluate(board):
    winner = 0
```

```
    for player in [1, 2]:
        if (row_win(board, player) or
            col_win(board,player) or
            diag_win(board,player)):

            winner = player

    if np.all(board != 0) and winner == 0:
        winner = -1
    return winner

pc = 2

# Đánh giá cục diện trận đấu
def value(board):
  v = evaluate(board)
  if v == pc:
    return 1
  elif v == 3 - pc:
    return -1
  else:
    return 0

# thuật toán minimax
def minimax(board, d, player):
  if d==0 or evaluate(board)!=0:
    return board, value(board)
  if player == pc:
    max,bmax = -10, 1
    for l in possibilities(board):
      child = np.copy(board)
      child[l] = player
      b,v = minimax(child, d-1, 3-player)
      if max<=v:
        max,bmax = v,child
    return bmax,max
  else:
    min,bmin = 10, 1
    for l in possibilities(board):
      child = np.copy(board)
      child[l] = player
      b,v = minimax(child, d-1, 3-player)
      if min>=v:
        min,bmin = v,child
    return bmin,min

# lựa chọn nước đi sử dụng minimax
def minimax_place(board):
  b, v = minimax(board,2,pc)
```

```
  return b

# chọn thủ công
def hand_place(board, player):
    selection = possibilities(board)
    i,j = map(int,input().split())
    if (i,j) in selection:
      board[i,j] = player
    return(board)

# Main function to start the game
def play_game():
    board, winner, counter = create_board(3), 0, 1
    print(board)

    while winner == 0:
        for player in [1, 2]:
            if player == pc:
              print("PC move")
              board = minimax_place(board)
            else:
              print("you move")
              board = hand_place(board, player)


            print(board)
            counter += 1
            winner = evaluate(board)
            if winner != 0:
                break
    return(winner)

# Driver Code
print("Winner is: " + str(play_game()))
```

Dán hình ảnh kết quả thực thi vào bên dưới:

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 2. 2. 2. 2.]]
PC move
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 2. 2. 2. 2. 2.]]
you move
6 0
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 2. 2. 2. 2. 2.]]
PC move
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 2. 2. 2. 2. 2. 2.]]
you move
```

Nêu ý tưởng cụ thể để giảm số nhánh trong quá trình tìm kiếm ở trong code của mình.

- Nếu như đạt đến giới hạn tìm kiếm (đến tầng dưới cùng của cây tìm kiếm), tính giá trị tĩnh của thế cờ hiện tại ứng với người chơi ở đó. Ghi nhớ kết quả.
- Nếu như mức đang xét là của người chơi cực tiểu, áp dụng thủ tục Minimax này cho các con của nó. Ghi nhớ kết quả nhỏ nhất.
- Nếu như mức đang xét là của người chơi cực đại, áp dụng thủ tục Minimax này cho các con của nó. - Ghi nhớ kết quả lớn nhất.