Projeto de Inteligência Artificial

Rúben Santos, 41308
Rui Barata, 41872
2020/2021

Professor: Luís Filipe Barbosa de Almeida Alexandre

FACULDADE ENGENHARIA
Departamento de Informática

Questões

1. **Qual foi a penúltima pessoa que viste?**
   Rúben Santos

2. **Em que tipo de sala estás agora?**
   Rui Barata

3. **Qual o caminho para a sala de enfermeiros mais próxima?**
   Rui Barata

4. **Qual a distância até ao médico mais próximo?**
   Rúben Santos

5. **Quanto tempo achas que demoras a ir de onde estás até às escadas?**
   Rui Barata

6. **Quanto tempo achas que falta até ficares sem bateria?**
   Rui Barata

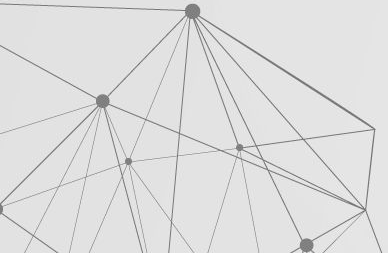
7. **Qual a probabilidade de encontrar um livro numa divisão, se já encontraste uma cadeira?**
   Rúben Santos

8. **Se encontrares um enfermeiro numa divisão, qual é a probabilidade de estar lá um doente?**
   Rúben Santos

1. Qual foi a penúltima pessoa que viste?

```
people = []
objects = []
```

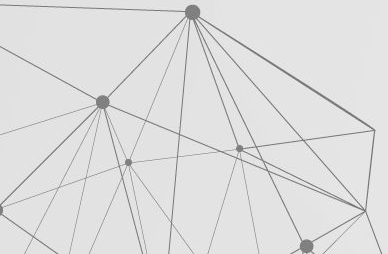
```
def resp1():
    if len(people) < 2: # quando ainda so viu 1 ou 0 pessoas
        print("Ainda não vi mais do que uma pessoa")
    else:
        print(people[1])
```



```
def update_Lists(obj, x, y): #funcao que adiciona objetos e pessoas as listas certas

    l = obj.split("_")[0]

    switcher2 = {
        'doente': True,
        'medico': True,
        'enfermeiro': True,
        'cama': False,
        'livro': False,
        'cadeira': False,
        'mesa': False,
    }

    #remover objectos das listas se ja estiverem presentes para a memoria não dar overflow
    if switcher2[l]:
        try:
            people.remove(obj)
        except:
            pass
    else:
        try:
            objects.remove(obj)
        except:
            pass

    switcher1 = {
        'doente': people.insert,
        'medico': people.insert,
        'enfermeiro': people.insert,
        'cama': objects.insert,
        'livro': objects.insert,
        'cadeira': objects.insert,
        'mesa': objects.insert
    }

    switcher1[l](0,obj)
    r = check_room(x,y)

    if(rooms[r][l].count(obj) == 0):
        rooms[r][l].append(obj)

    if(not (switcher2[l])):
        update_room_type(r)

    if (l == 'medico'):
        addMedico(obj)
```

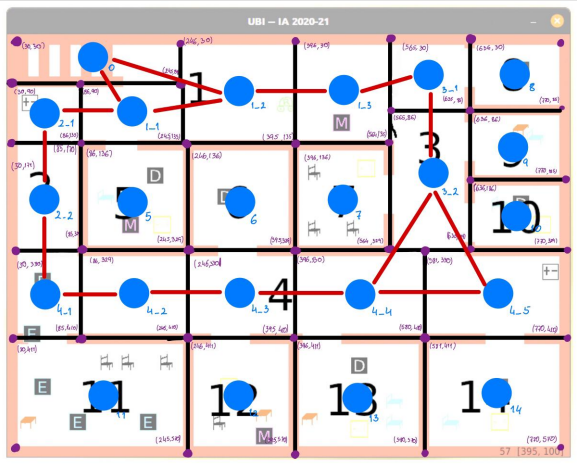
2. Em que tipo de sala estás agora?

```
rooms = {   # dicionario que contem os objetos presentes em cada quarto e outras informacoes sobre cada sala
    1: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': 0},
    2: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': 0},
    3: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': 0},
    4: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': 0},
    5: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    6: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    7: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    8: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    9: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    10: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    11: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    12: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    13: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
    14: {'doente': [], 'medico': [], 'enfermeiro': [], 'cama': [], 'livro': [], 'cadeira': [], 'mesa': [], 'type': (-1)},
} # tipo de divisão: 0 -> corredor || 1 -> quarto || 2 -> sala de enfermeiros || 3 -> sala de espera || -1 -> unknown
```



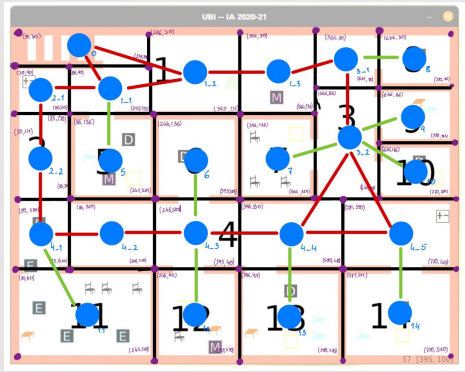
```
def resp2():

    r = check_room(posicaoX,posicaoY)

    # ve em que tipo de divisao esta
    if (0<=r<=4):                                                          #corredor
        print("Esta divisão é um corredor")
    elif (len(rooms[r]['cama']) != 0):                                     #quarto
        print("Esta divisão é um quarto")
    elif ((len(rooms[r]['mesa']) != 0) and (len(rooms[r]['cadeira']) != 0)):    #sala de enfermeiros
        print("Esta divisão é uma sala de enfermeiros")
    elif (len(rooms[r]['cadeira']) > 2):                                   #sala de espera
        print("Esta divisão é uma sala de espera")
    else:
        print("Ainda não tenho informação suficiente")
```

GRAFO




```python
# dicionarios
nodeBorders = { # bordas dos nodos
    "0": [(30,30), (245, 89)],
    "1_1": [(86,90), (245, 135)],
    "1_2": [(246,30), (395, 135)],
    "1_3": [(396,30), (564, 135)],
    "2_1": [(30,90), (85, 170)],
    "2_2": [(30,171), (85, 329)],
    "3_1": [(565,30), (635, 85)],
    "3_2": [(565,86), (635, 329)],
    "4_1": [(30,330), (85, 410)],
    "4_2": [(86,330), (245, 410)],
    "4_3": [(246,330), (395, 410)],
    "4_4": [(396,330), (580, 410)],
    "4_5": [(581,330), (770, 410)],
    "5": [(86,136), (245,329)],
    "6": [(246,136), (395,329)],
    "7": [(396,136), (564,329)],
    "8": [(636,30), (770,85)],
    "9": [(636,86), (770,185)],
    "10": [(636,186), (770,329)],
    "11": [(30,411), (245,570)],
    "12": [(246,411), (395,570)],
    "13": [(396,411), (580,570)],
    "14": [(581,411), (770,570)]
}
```

```python
def initGraph():    # funcao que inicializa o grafo
    global roomGraph
    global init_FLAG
    global nodo_atual

    init_FLAG = False

    for i in nodeBorders.keys(): # adiconar todos os nodos no dicionario
        roomGraph.add_node(i)

    # adicionar edges entre os corredores (pois estas conexoes existem sempre)
    roomGraph.add_edge("0", "1_1", weight=59.9416)      #1
    roomGraph.add_edge("1_1", "0", weight=59.9416)
    roomGraph.add_edge("0", "1_2", weight=184.4397)     #2
    roomGraph.add_edge("1_2", "0", weight=184.4397)
    roomGraph.add_edge("1_1", "1_2", weight=157.8765)   #3
    roomGraph.add_edge("1_2", "1_1", weight=157.8765)
    roomGraph.add_edge("1_2", "1_3", weight=159.5)      #4
    roomGraph.add_edge("1_3", "1_2", weight=159.5)
    roomGraph.add_edge("1_3", "3_1", weight=122.5765)   #5
    roomGraph.add_edge("3_1", "1_3", weight=122.5765)
    roomGraph.add_edge("3_1", "3_2", weight=150.0)      #6
    roomGraph.add_edge("3_2", "3_1", weight=150.0)
    roomGraph.add_edge("3_2", "4_5", weight=179.1829)   #7
    roomGraph.add_edge("4_5", "3_2", weight=179.1829)
    roomGraph.add_edge("4_5", "4_4", weight=187.5)      #8
    roomGraph.add_edge("4_4", "4_5", weight=187.5)
    roomGraph.add_edge("3_2", "4_4", weight=197.3582)   #9
    roomGraph.add_edge("4_4", "3_2", weight=197.3582)
    roomGraph.add_edge("4_3", "4_4", weight=167.5)      #10
    roomGraph.add_edge("4_4", "4_3", weight=167.5)
    roomGraph.add_edge("4_1", "4_2", weight=108.0)      #11
    roomGraph.add_edge("4_2", "4_1", weight=108.0)
    roomGraph.add_edge("4_2", "4_3", weight=155.0)      #12
    roomGraph.add_edge("4_3", "4_2", weight=155.0)
    roomGraph.add_edge("4_1", "2_2", weight=120.0)      #13
    roomGraph.add_edge("2_2", "4_1", weight=120.0)
    roomGraph.add_edge("2_1", "2_2", weight=120.0)      #14
    roomGraph.add_edge("2_2", "2_1", weight=120.0)
    roomGraph.add_edge("2_1", "1_1", weight=109.4086)   #15
    roomGraph.add_edge("1_1", "2_1", weight=109.4086)

    #inicializar nodo atual
    nodo_atual = findActualNode(posicaoX, posicaoY)
```



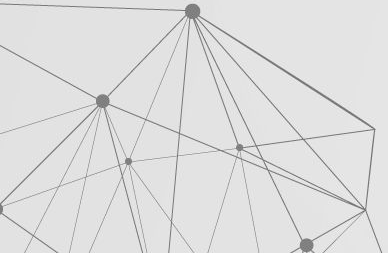

```python
if(aux != nodo_atual): # cria edges entre os corredores e divisoes
    l = list(roomGraph.edges())
    if(l.count((nodo_atual, aux)) == 0 and l.count((aux, nodo_atual)) == 0):

        (x1, y1) = getNodeCenter(nodo_atual)
        (x2, y2) = getNodeCenter(aux)

        roomGraph.add_edge(nodo_atual, aux, weight=check_distance(x1, y1, x2, y2))
        roomGraph.add_edge(aux, nodo_atual, weight=check_distance(x1, y1, x2, y2))

nodo_atual = aux    # atualiza nodo atual
```

3. Qual o caminho para a sala de enfermeiros mais próxima?

```
def resp3():

    if(rooms[check_room(posicaoX, posicaoY)]['type'] == 2): # ve se a sala atual ja e uma sala de enfermeiros
        print("Já me encontro numa sala de enfermeiros")
    else:

        minDistance = 100000
        minRoom = -1
        path = []

        n_atual = findActualNode(posicaoX, posicaoY)

        for (r, dic) in rooms.items(): # vai comparar todos os caminhos para todas as salas de enfermeiros e escolhe o mais pequeno
            if (dic['type'] == 2):
                l = nx.shortest_path(roomGraph, n_atual, str(r), "weight")
                aux = getPathCost(l)

                if (minDistance > aux):
                    minDistance = aux
                    minRoom = r
                    path = l


        if (minRoom == -1): # se nao encontrar nenhum caminho e porque ainda nao conhece nenhuma sala de enfermeiros
            print("Ainda não encontrei nenhuma sala de enfermeiros")
        else:   # se encontrar imprime o melhor
            printPath(path)
```

4. Qual a distância até ao médico mais próximo?

```
medicList = { # coordenadas dos medicos que estão em cada nodo
    "0": {},
    "1_1": {},
    "1_2": {},
    "1_3": {},
    "2_1": {},
    "2_2": {},
    "3_1": {},
    "3_2": {},
    "4_1": {},
    "4_2": {},
    "4_3": {},
    "4_4": {},
    "4_5": {},
    "5": {},
    "6": {},
    "7": {},
    "8": {},
    "9": {},
    "10": {},
    "11": {},
    "12": {},
    "13": {},
    "14": {},
}
```



```
def resp4():

    minDistance = 100000
    minRoom = -1
    path = []

    n_atual = findActualNode(posicaoX, posicaoY)

    for (r, med) in medicList.items():  # encontra caminho para o nodo mais proximo com medicos presentes
        if (len(med) > 0):

            path = nx.shortest_path(roomGraph, n_atual, r, "weight")
            aux = getPathCost(path)

            if (minDistance > aux):
                minDistance = aux
                minRoom = r

    minDis = 100000

    if(minRoom == -1):
        print("Ainda nao encontrei nenhum medico")
        return
```

4. Qual a distância até ao médico mais próximo? (Continuação)

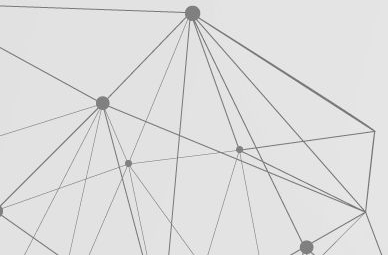
```
for (x2, y2) in medicList[minRoom].values():    # encontra o medico mais proximo nesse nodo

    if (n_atual == minRoom):    # se for no mesmo nodo em que o robo esta entao ve a distancia de todos os medicos presentes no nodo ate ao robo e escolhe o mais proximo

        aux = check_distance(posicaoX, posicaoY, x2, y2)

        if (minDis > aux):
            minDis = aux

    else:   # ve a distancia desde o penultimo nodo do caminho ate ao medico mais proximo desse nodo

        path.reverse()

        (cx, cy) = getNodeCenter(path[1])
        (minRoomx, minRoomy) = getNodeCenter(path[0])

        aux = check_distance(cx, cy, x2, y2) - check_distance(cx, cy, minRoomx, minRoomy)

        if (minDis > aux):
            minDis = aux

minDistance += minDis

print(minDistance)
```

5. Quanto tempo achas que demoras a ir de onde estás até às escadas?

```
if (vel == -1): # mede a velocidade
    if(velX == -1):
        velX = posicaoX
        velY = posicaoY

    if (velX == posicaoX and velY == posicaoY):
        velT = time.time()
    else:
        vel = 5/(time.time()-velT)
```

$$\bar{v} = \frac{\Delta s}{\Delta t}$$

```
def resp5():

    # calcula uma aproximacao do tempo que demora usando o custo do caminho calculado a dividir

    if (vel == -1):
        print("Ainda nao tenho informacao suficiente para responder")
        return

    n_atual = findActualNode(posicaoX, posicaoY)

    l = nx.shortest_path(roomGraph, n_atual, "0", "weight")

    t = getPathCost(l) / vel

    print("Devo chegar as escadas em {:.2f}s".format(t))
```



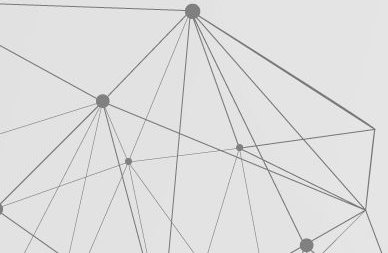
6. Quanto tempo achas que falta até ficares sem bateria?

```
batDataX = []
batDataY = []
Taux = -1
batAux = -1
```

```
def expo(x, a, b):
    return np.exp(a + b * x)
```

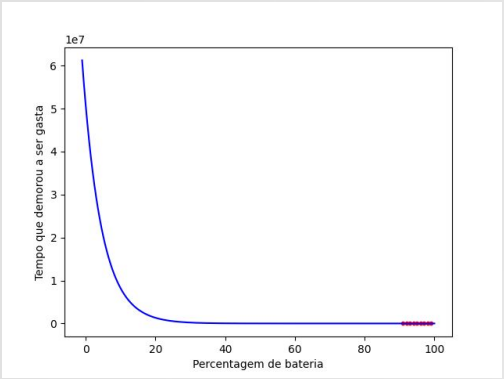
```
if (batAux == -1 or bat == 100):
    batAux = bat
    Taux = time.time()

if (batAux > int(bat)):

    if (batAux <= 99):
        batDataX.append(batAux)
        batDataY.append(float("{:.3f}".format(time.time()-Taux)))

    batAux = int(bat)
```

```
def resp6():

    try:
        (a, b),_ = sp.curve_fit(expo, batDataX, batDataY)
        print("{:.2f}s".format((expo(0, a, b) - expo(bat, a, b))))
    except:
        print("Ainda nao tenho informacao suficiente para responder")

    """ x = np.linspace(-1, 100, 202)
    y = expo(x, a, b)
    plt.plot(batDataX, batDataY, "r.")
    plt.plot(x, y, "b-")
    plt.xlabel("Percentagem de bateria")
    plt.ylabel("Tempo que demorou a ser gasta")
    plt.show() """
```



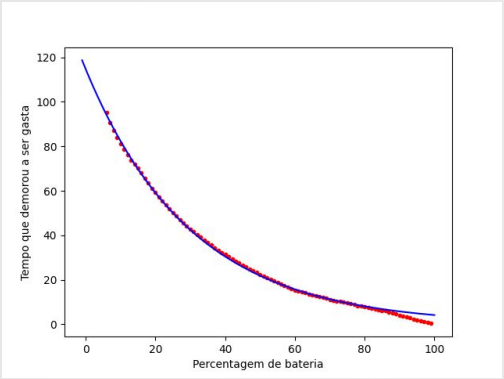
6. Quanto tempo achas que falta até ficares sem bateria? (Continuação)

51026948.92s



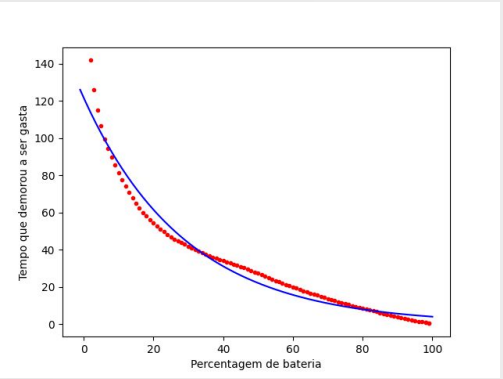
90%

265.77s



5%

117.77s



100%

7. Qual a probabilidade de encontrar um livro numa divisão, se já encontraste uma cadeira?

```
def resp7():

    # calcula a probabilidade condicionada de encontrar um livro
    #sabendo que encontrou uma cadeira

    #C: probabilidade de encontrar cadeira
    cfC = 0
    #P(L ^ C)
    cfLC = 0

    for (r, obj) in rooms.items():
        if(r>4):
            if (len(obj['cadeira']) > 0 and len(obj['livro']) > 0):
                cfLC += 1
            if (len(obj['cadeira']) > 0):
                cfC += 1

    if (cfC != 0):
        print("{:d}%".format(int((cfLC / cfC)*100)))
    else:
        print("{:d}%".format(cfC))
```

8. Se encontrares um enfermeiro numa divisão, qual é a probabilidade de estar lá um doente?

```
def resp8():
    # calcula a probabilidade condicionada de encontrar um doente
    #sabendo que encontrou um enfermeiro

    #C: probabilidade de encontrar cadeira
    cfE = 0
    #P(L ^ C)
    cfDE = 0

    for (r, obj) in rooms.items():
        if(r>4):
            if (len(obj['doente']) > 0 and len(obj['enfermeiro']) > 0):
                cfDE += 1
            if (len(obj['enfermeiro']) > 0):
                cfE += 1

    if (cfE != 0):
        print("{:d}%".format(int((cfDE / cfE)*100)))
    else:
        print("{:d}%".format(cfE))
```

OBRIGADO PELA SUA ATENÇÃO