Elementos de Programação

Relatório da Primeira Parte do Projeto

Departamento de Matemática, IST

2018/2019

Os ficheiros enviados juntamente com este relatório que contém os módulos são os seguintes:

- evento_c.py - módulo que disponibiliza o tipo de dados *evento*
- cap_c.py - módulo que disponibiliza o tipo de dados *cadeia de acontecimentos pendentes*
- individuo_c.py - módulo que disponibiliza o tipo de dados *individuo*
- grelha_c.py - módulo que disponibiliza o tipo de dados *grelha*

evento_c.py

Este módulo disponibiliza o tipo de dados *evento*. O tipo de dados foi implementado através da definição da classe *evento* que disponibiliza os seguintes métodos:

- **tempo ():** devolve o instante do evento
- **tipo ():** devolve o tipo de evento
- **ID ():** devolve o identificador do evento

A criação destes objetos é feita fornecendo três parâmetros: o tempo do evento, o tipo de evento (reprodução, deslocamento, avaliação ou morte) e o identificador do objeto (nomeadamente, o identificador do indivíduo ao qual este evento está associado). De seguida, apresentamos alguns exemplos da utilização desta classe:

```
In [1]: import evento_c as event
In [1]: a = event.evento(1,'Tipo 1',20)
In [2]: b = event.evento(2,'Tipo 2',55)
In [3]: e = event.evento(3,'Tipo 1',83)
In [4]: a.tempo()
Out [4]: 20
In [5]: b.tipo()
Out [5]: 'Tipo 2'
In [6]: e.ID ()
Out [6]: 3
```

cap_c.py

Este módulo disponibiliza o tipo de dados *cadeia de acontecimentos pendentes*. O tipo de dados foi implementado através da definição da classe *cap* que disponibiliza os seguintes métodos:

- **adicionarE (e):** adiciona o evento *e* à cap
- **proxE ():** devolve o evento com menor tempo da cap, caso esta não esteja vazia
- **retirarE ():** elimina o evento com menor tempo da cap, caso esta não esteja vazia
- **eliminarID ()**: remove todos os eventos associados a um certo identificador
- **tamC ():** devolve o tamanho da cap
- **mostraC ():** mostra o conteúdo da cap

Internamente a CAP funciona como uma lista à qual se adicionam itens da classe *evento*. Os eventos são ordenados por ordem crescente de tempo. Desta forma, o próximo evento é sempre o primeiro da lista. Para manter a ordem quando se adiciona um evento à CAP, primeiro procura-se o elemento imediatamente anterior ao que se quer introduzir e introduz-se o evento a seguir a este. De forma a aumentar a eficiência, para encontrar este evento usa-se um algoritmo de busca parecido com o exercício feito na aula para a procura de zeros de uma função usando o Teorema de Bolzano-Weierstrass. Este módulo importa o módulo evento_c.py. Cada objeto é criado inicialmente sem eventos (cap vazia). De seguida, apresentamos alguns exemplos da utilização desta classe:

```
In [1]: import cap_c as cap
In [2]: c = cap.CAP ()
In [3]: c.adicionarE(a)
In [4]: c.adicionarE(b)
In [5]: c.adicionarE(e)
In [6]: c.proxE()
Out [6]: <evento_c.evento at 0x10e9e4ef0>
In [7]: c.proxE().time()
Out [7]: 20
In [7]: c.mostraC()
ID: 2    Tipo: tipo2    Tempo: 20
ID: 3    Tipo: tipo3    Tempo: 50
ID: 1    Tipo: tipo1    Tempo: 56

In [8]: c.retirarE()
In [9]: c.proxE().time()
Out [9]: 50
In [10]: c.eliminarID(1)
In [11]: c.mostraC()
ID: 3    Tipo: tipo3    Tempo: 50

In [12]: d.tamC()
Out [12]: 1
```

individuo_c.py

Este módulo disponibiliza o tipo de dados *individuo*. O tipo de dados foi implementado através da definição da classe *individuo* que disponibiliza os seguintes métodos:

- **estado ():** devolve o estado do indivíduo (pode ser S, E, I, R)
- **ID ():** devolve o identificador do indivíduo
- **mudae (e)**: permite alterar o estado de um indivíduo

A criação destes objetos é feita fornecendo dois parâmetros: o estado do indivíduo (suscetível, exposto, infetado ou recuperado) e o identificador do indivíduo. De seguida, apresentamos alguns exemplos da utilização desta classe:

```
In [1]: import individuo_c as individuo
In [2]: a = individuo.individuo(1,'S')
In [3]: b = individuo.individuo(2,'E')
In [4]: a.ID ()
Out [4]: 1
In [5]: a.estado()
Out [5]: 'S'
In [6]: b.estado()
Out [6]: 'E'
In [7]: a.mudae('I')
In [8]: a.estado()
Out [8]: 'I'
```

grelha_c.py

Este módulo disponibiliza o tipo de dados *grelha*. O tipo de dados foi implementado através da definição da classe *grelha* que disponibiliza os seguintes métodos:

- **encontrar (ID)**: devolve a posição de um indivíduo com um dado identificador
- **obj (p)**: devolve o objeto na posição p (seja este um indivíduo, obstáculo ou livre)
- **insI (ind,p)**: adiciona um indivíduo (ind) a uma dada posição (p)
- **delI (p)**: retira um indivíduo de uma dada posição
- **usaQ (p)**: devolve True se uma posição não tem obstáculos
- **livQ (p)**: devolve True se uma posição está livre (=None), e False caso contrário
- **ocuQ (p)**: devolve True se a posição p se encontra ocupada por um indivíduo, e False caso contrário
- **infQ (p)**: devolve True se uma posição tiver um indivíduo infetado, e False caso contrário
- **susQ (p):** devolve True se uma posição tiver um indivíduo suscetível, e False caso contrário
- **expQ (p):** devolve True se uma posição tiver um indivíduo exposto, e False caso contrário
- **recQ (p):** devolve True se uma posição tiver um indivíduo recuperado, e False caso contrário
- **lis_Q (c,w)**: devolve a lista dos elementos de w que verificam o predicado c
- **dentrogrelha(p)**: assegura que uma coordenada está contida na grelha. Caso não esteja, coloca-a na grelha

- **viz1 (ID)**: devolve as posições da vizinhança 1 de um indivíduo
- **viz2 (ID)**: devolve as posições da vizinhança 2 de um indivíduo
- **contactoQ (aID,bp)**: recebendo o ID de um indivíduo a e a posição de um indivíduo b, devolve True se estes estiverem em contacto, e False caso contrário
- **nInf ()**: devolve o número total de indivíduos infetados na grelha
- **c_estados ()**: devolve uma lista de listas com as coordenadas de cada indivíduo em cada estado

Este módulo importa o módulo individuo_c.py. A criação destes objetos é feita fornecendo dois parâmetros: a dimensão N da grelha e a lista de obstáculos. Internamente a grelha funciona como uma matriz: na criação de uma grelha ocupam-se todas as posições com "O" se for um obstáculo ou *None* caso contrário. De seguida, apresentamos alguns exemplos da utilização desta classe:

```
In [1]: import individuo_c as ind
In [2]: import grelha_c as gr

# Criar a grelha
In[3]:g=gr.Grelha(3,[(2,3),(2,2),(2,1),(2,0),(2,-1),(2,-2),(2,-3)])

# Cria indivíduos para pôr na grelha
In [4]: a = ind.individuo(1,'S')
In [5]: b = ind.individuo(2,'S')
In [6]:c = ind.individuo(3,'S')
In [7]: d = ind.individuo(4,'S')
In [8]: e = ind.individuo(5,'S')
In [9]: f = ind.individuo(6,'I')
In [10]: h = ind.individuo(7,'I')
In [11]: i = ind.individuo(8,'I')
In [12]: j = ind.individuo(9,'I')
In [13]: k = ind.individuo(10,'I')

# Adiciona os indivíduos à grelha
In [14]: g.insI(a,[1,2])
In [15]: g.insI(b,[1,-2])
In [16]: g.insI(c,[-1,-2])
In [17]: g.insI(d,[-1,-3])
In [18]: g.insI(e,[-1,3])
In [19]: g.insI(f,[1,3])
In [20]: g.insI(h,[1,1])
In [21]: g.insI(i,[1,-1])
In [22]: g.insI(j,[-1,-1])
In [23]: g.insI(k,[-1,0])

In [24]: g.obj ([1,1])
Out [24]: <individuo_c.individuo at 0x18153ea5c0>
In [25]: g.delI([1,1])
```

```
In [26]: g.obj([1,1])  # Como a posição está vazia, não devolve nada
In [27]: g.obj([2,1])
Out [27]: 'O'
In [28]: g.encontrar(3)
Out [28]: [-1, -2]
In [29]: g.encontrar(6)
Out [29]: [1, 3]

In [30]: g.usaQ([-3,3])       # para posição vazia (=None)
Out [30]: True
In [31]: g.usaQ([1,2])        # posição com indivíduo
Out [31]: True
In [32]: g.usaQ([2,2])        # posição com obstáculo
Out [32]: False

In [33]: g.livQ([-3,3])     # para posição vazia
Out [33]: True
In [34]: g.livQ([1,2])      # posição com indivíduo
Out [34]: False
In [35]: g.livQ([2,2])      # posição com obstáculo
Out [35]: False

In [36]: g.ocuQ([-3,3])       # para posição vazia
Out [36]: False
In [37]: g.ocuQ([1,2])        # posição com indivíduo
Out [37]: True
In [38]: g.ocuQ([2,2])        # posição com obstáculo
Out [38]: False

In [39]: g.susQ([1,-2])
Out [39]: True
In [40]: g.susQ([1,-1])
Out [40]: False
In [41]: g.infQ([1,-2])
Out [41]: False
In [42]: g.infQ([1,-1])
Out [42]: True
In [43]: g.expQ([1,-1])
Out [43]: False
In [44]: g.recQ([1,-1])
Out [44]: False

In [45]: g.dentrogrelha([4,0])
Out [45]: [-3, 0]
In [46]: g.dentrogrelha([2,0])
Out [46]: [2, 0]
```

```
In [47]: g.viz1(a.ID())
Out [47]: [[0, 1], [0, 2], [0, 3], [1, 1], [1, 3], [2, 1], [2, 2],
[2, 3]]
In [48]: g.viz1(f.ID())
Out [48]: [[0, 2], [0, 3], [0, -3], [1, 2], [1, -3], [2, 2], [2, 3],
[2, -3]]
In [49]: g.viz2(a.ID())
Out [49]:
[[-1, 0],
 [-1, 1],
 [-1, 2],
 [-1, 3],
 [-1, -3],
 [0, 0],
 [0, -3],
 [1, 0],
 [1, -3],
 [2, 0],
 [2, -3],
 [3, 0],
 [3, 1],
 [3, 2],
 [3, 3],
 [3, -3]]
In [50]: g.viz2(f.ID())
Out [50]:
[[-1, 1],
 [-1, 2],
 [-1, 3],
 [-1, -3],
 [-1, -2],
 [0, 1],
 [0, -2],
 [1, 1],
 [1, -2],
 [2, 1],
 [2, -2],
 [3, 1],
 [3, 2],
 [3, 3],
 [3, -3],
 [3, -2]]

In [51]: g.lis_Q(g.livQ,g.viz1(a.ID()))
Out [51]: [[0, 1], [0, 2], [0, 3], [1, 1]]
```

```
In [52]: g.lis_Q(g.livQ,g.viz1(f.ID()))
Out [52]: [[0, 2], [0, 3], [0, -3], [1, -3]]

In [53]: g.contactoQ(a.ID(),g.encontrar(b.ID()))
Out [53]: False
In [54]: g.contactoQ(a.ID(),g.encontrar(d.ID()))
Out [54]: True

In [55]: g.nInf()
Out [55]: 4

In [56]: g.c_estados()
Out [56]:
[[(-1, 3), (1, 2), (-1, -2), (1, -2), (-1, -3)],
 [],
 [(1, 3), (-1, 0), (-1, -1), (1, -1)],
 []]
```