Projecto de Biocomputação 2020-2021

Departamento de Matemática, IST

Dezembro de 2020

Ficheiros

Os ficheiros disponibilizados na página da cadeira são os seguintes:

- `graph_adt.py` - módulo que disponibiliza o tipo de dados *grafo*;
- `path_adt.py` - módulo que disponibiliza o tipo de dados *caminho*;
- `ind_adt.py` - módulo que disponibiliza o tipo de dados *indivíduo*;
- `event_adt.py` - módulo que disponibiliza o tipo de dados *evento*;
- `cap_adt.py` - módulo que disponibiliza o tipo de dados *cadeia de acontecimentos pendentes*;
- `tube_adt.py` - módulo que disponibiliza o tipo de dados *solução biomolecular*.

graph_adt.py

Este módulo disponibiliza o tipo de dados *grafo*, através das operações a seguir descritas:

- `newG(V)` - recebe uma lista `V` de elementos e devolve o grafo vazio (sem arestas) cujos vértices são os elementos da lista `V`;
- `add_edgesG(g,E)` - recebe um grafo `g` e uma listas de pares `E` e devolve o grafo com os pares de `E` como arestas;
- `get_nodesG(g)` - recebe um grafo `g` e devolve a lista dos vértice do grafo;
- `get_edgesG(g)` - recebe um grafo `g` e devolve a lista das arestas do grafo;
- `emptyG(g)` - recebe um grafo `g` e devolve `True` se o grafo for vazio e `False` em caso contrário.

Seguem-se alguns exemplos de utilização, em que se constrói o primeiro grafo do enunciado do projecto:

```
In [1]: import graph_adt as graph
In [2]: g=graph.newG([1,2,3,4])
In [3]: g=graph.add_edgesG(g,[(1,2),(1,3),(2,3),(3,2),(2,4),(3,4)])
In [4]: graph.get_nodesG(g)
Out[4]: [1, 2, 3, 4]
In [5]: graph.get_edgesG(g)
Out[5]: [(1, 2), (1, 3), (2, 3), (3, 2), (2, 4), (3, 4)]
In [6]: graph.emptyG(g)
Out[6]: False
```

path_adt.py

Este módulo disponibiliza o tipo de dados *caminho*, através das seguintes operações:

- `newP(v1,v2)` - recebe dois vértices `v1` e `v2` e devolve o caminho de `v1` para `v2`;
- `firstP(p)` - recebe um caminho `p` e devolve o primeiro vértice desse caminho;
- `lastP(p)` - recebe um caminho `p` e devolve o último vértice desse caminho;
- `lenP(p)` - recebe um caminho `p` e devolve o seu comprimento;
- `compP(p1,p2)` - recebe dois caminhos `p1` e `p2` e devolve `True` se o caminho `p1` for compatível com o caminho `p2`, isto é, se o último vértice de `p1` for igual ao primeiro vértice de `p2`, e devolve `False` em caso contrário;
- `glueP(p1,p2)` - recebe dois caminhos `p1` e `p2` e se `p1` for compatível com `p2` devolve o caminho resultante de juntar `p1` com `p2` (por esta ordem), e devolve apenas `p1` se este não for compatível com `p2`;
- `breakP(p,i)` - recebe um caminho `p` e um inteiro `i` e devolve um par constituído pelos dois caminhos resultantes de partir `p` na posição `i`;
- `crossesP(p,v)` - recebe um caminho `p` e um vértice `v` e devolve `True` se o caminho `p` passa no vértice `v`, e devolve `False` em caso contrário;
- `copyP(p)` - recebe um caminho `p` e devolve uma cópia desse caminho;
- `showP(p)` - recebe um caminho `p` e imprime-o.

Seguem-se alguns exemplos de utilização.

```
 In [1]: import map_adt as map
 In [2]: a1=path.newP(1,2)
         a2=path.newP(2,4)
         a3=path.newP(1,3)
         a4=path.newP(3,4)
         a5=path.newP(2,3)
         a6=path.newP(3,2)
 In [3]: path.firstP(a1)
 Out[3]: 1
 In [4]: path.lastP(a1)
 Out[4]: 2
 In [5]: path.lenP(a1)
 Out[5]: 2
 In [6]: path.compP(a1,a2)
 Out[6]: True
 In [7]: path.compP(a1,a4)
 Out[7]: False
 In [8]: p1=path.glueP(a1,a2)
 In [9]: path.showP(p1)
 Out[9]: [1, 2, 4]
In [10]: path.lenP(p1)
Out[10]: 3
In [11]: path.crossesP(p1,1)
Out[11]: True
In [12]: path.crossesP(p1,2)
Out[12]: True
In [13]: path.crossesP(p1,3)
Out[13]: False
In [14]: p2,p3=path.breakP(p1,1)
In [15]: path.showP(p2)
Out[15]: [1, 2]
In [16]: path.showP(p3)
Out[16]: [2, 4]
In [17]: p4=path.copyP(p1)
In [18]: path.showP(p4)
Out[18]: [1, 2, 4]
```

ind_adt.py

Este módulo disponibiliza o tipo de dados *indivíduo*, através das seguintes operações:

- `newI(p,x,y,z,i)` - recebe um caminho `p`, três números reais `x`, `y`, `z` e um identificador de indivíduo `i` e devolve um indivíduo com essas características, ou seja, com caminho `p`, coordenadas `(x,y,z)` e identificador `i`;
- `pathI(i)` - recebe um indivíduo `i` e devolve o seu caminho;
- `xposI(i)` - recebe um indivíduo `i` e devolve a sua coordenada x;
- `yposI(i)` - recebe um indivíduo `i` e devolve a sua coordenada y;
- `zposI(i)` - recebe um indivíduo `i` e devolve a sua coordenada z;
- `idI(i)` - recebe um indivíduo `i` e devolve o seu identificador;
- `distI(i1,i2)` - recebe dois indivíduos `i1` e `i2` e devolve a distância entre eles;
- `eqI(i1,i2)` - recebe dois indivíduos `i1` e `i2` e devolve `True` se estes forem o mesmo indivíduo, e devolve `False` em caso contrário.

Seguem-se alguns exemplos de utilização.

```
In [1]: import ind_adt as ind
In [2]: i1=ind.newI(a3,1,2,3,1)
        i2=ind.newI(a1,2,3,4,2)
        i3=ind.newI(a2,1,1,1,3)
        i4=ind.newI(a4,1,2,1,4)
        i5=ind.newI(a5,0,0,0,5)
        i6=ind.newI(a6,4,3,2,6)
In [3]: path.showP(ind.pathI(i1))
Out[3]: [1, 3]
In [4]: ind.xposI(i1),ind.yposI(i1),ind.zposI(i1)
Out[4]: (1, 2, 3)
In [5]: ind.distI(i1,i2)
Out[5]: 1.7320508075688772
In [6]: ind.eqI(i1,i1)
Out[6]: True
In [7]: ind.eqI(i1,i2)
Out[7]: False
```

event_adt.py

Este módulo disponibiliza o tipo de dados *evento*, através das seguintes operações:

- `newE(t,k,i)` - recebe um tempo `t`, um tipo de evento `k` e um identificador de indivíduo `i` e devolve um evento com essas características;
- `timeE(e)` - recebe um evento `e` e devolve o seu tempo;
- `kindE(e)` - recebe um evento `e` e devolve o seu tipo;
- `idE(e)` - recebe um evento `e` e devolve o identificador do indivíduo que lhe está associado.

Seguem-se alguns exemplos de utilização.

```
In [1]: import event_adt as event
In [2]: e1=event.newE(1.5,"conc",1)
        e2=event.newE(7.5,"conc",2)
        e3=event.newE(3.5,"desl",1)
        e4=event.newE(25.5,"desl",3)
        e5=event.newE(10.5,"cis",0)
        e6=event.newE(12.5,"conc",7)
        e7=event.newE(2.5,"conc",6)
In [3]: event.timeE(e1)
Out[3]: 1.5
In [4]: event.kindE(e1)
Out[4]: 'conc'
In [5]: event.idE(e1)
Out[5]: 1
```

cap_adt.py

Este módulo disponibiliza o tipo de dados *cadeia de acontecimentos pendentes*, através das seguintes operações:

- `newS()` - cap vazia;
- `addS(s,ev)` - acrescenta o evento `ev` à cap `s`;
- `delS(s)` - apaga o evento com menor instante da cap `s`, caso esta não esteja vazia;

- `nextS(s)` - devolve o evento com menor instante da cap `s`, caso esta não esteja vazia;
- `lenS(s)` - devolve o comprimento da cap `s`;
- `show(s)` - mostra o conteúdo da cap `s`.

Seguem-se alguns exemplos de utilização.

```
In [1]: import cap_adt as cap
In [2]: c=cap.newS()
        c=cap.addS(c,e1)
        c=cap.addS(c,e2)
        c=cap.addS(c,e3)
        c=cap.addS(c,e4)
        c=cap.addS(c,e5)
        c=cap.addS(c,e6)
        c=cap.addS(c,e7)
In [3]: cap.showS(c)
         1.5 conc 1
         2.5 conc 6
         3.5 desl 1
         7.5 conc 2
         10.5 cis 0
         12.5 conc 7
         25.5 desl 3
In [4]: e=cap.nextS(c)
In [5]: event.timeE(e),event.kindE(e),event.idE(e)
Out[5]: (1.5, 'conc', 1)
In [6]: c=cap.delS(c)
In [7]: e=cap.nextS(c)
In [8]: event.timeE(e),event.kindE(e),event.idE(e)
Out[8]: (2.5, 'conc', 6)
In [9]: cap.lenS(c)
Out[9]: 6
```

tube_adt.py

Este módulo disponibiliza o tipo de dados *solução biomolecular*, através das seguintes operações:

- `newT()` - solução vazia;
- `existsT(t,x)` - devolve `True` se o indivíduo com identificador `x` pertence à solução `t`, e devolve `False` em caso contrário;
- `addT(t,i)` - acrescenta o indivíduo `i` à solução `t`, se já existir em `t` um indivíduo com o mesmo identificador é substituído por `i`;
- `delT(t,x)` - apaga o indivíduo com identificador `x` da solução `t`;
- `get_indT(t,x)` - devolve o indivíduo com identificador `x` na solução `t`;
- `get_idsT(t)` - devolve a lista de identificadores de todos os indivíduos que se encontram na solução `t`;
- `neigT(t,x)` - devolve a lista com até 5 indivíduos compatíveis com o indivíduo com identificador `x` que se encontram mais próximos, na solução `t`;

Seguem-se alguns exemplos de utilização:

```
In [1]: import tube_adt as tube
 In [2]: t=tube.addT(t,i1)
         t=tube.addT(t,i2)
         t=tube.addT(t,i3)
         t=tube.addT(t,i4)
         t=tube.addT(t,i5)
         t=tube.addT(t,i6)
 In [3]: tube.existsT(t,6)
 Out[3]: True
 In [4]: tube.existsT(t,10)
 Out[4]: False
 In [5]: t=tube.delT(t,6)
 In [6]: tube.existsT(t,6)
 Out[6]: False
 In [7]: ii1=tube.get_indT(t,1)
 In [8]: ind.idI(ii1),ind.xposI(ii1),ind.yposI(ii1),ind.zposI(ii1)
 Out[8]: (1, 1, 2, 3)
 In [9]: tube.get_idsT(t)
 Out[9]: [1, 2, 3, 4, 5]
In [10]: w=tube.neigT(t,2)
In [11]: [ind.idI(i) for i in w]
Out[11]: [3, 5]
```

Recorde-se que o indivíduo com identificador 2 corresponde à aresta `a1=(1,2)`. Os únicos indivíduos na solução compatíveis com este são o indivíduo com identificador 3, correspondente à aresta `a2=(2,4)` e o indivíduo com identificador 5 correspondente à aresta `a5=(2,3)`. Portanto, no caso de não existirem 5 indivíduos compatíveis, devolvem-se os que existirem.