TÉCNICO LISBOA

Fundamentos da Programação
Ano letivo 2023-24
Primeiro Projeto
6 de Outubro de 2023

Montanhas e vales

Neste primeiro projeto de Fundamentos da Programação os alunos irão desenvolver as funções que permitam obter algumas informações sobre o estado de um território retangular formado por caminhos verticais e horizontais. As interseções dos caminhos de um território podem ou não estar ocupadas por montanhas e formar cadeias de montanhas e vales.

1 Descrição de um território

1.1 Territórios e interseções

Um **território** é uma estrutura retangular formado por N_v caminhos verticais e N_h caminhos horizontais. Os caminhos verticais são identificados por letras maiúsculas de `A` até `Z`, no máximo; enquanto que os caminhos horizontais são identificados por um número inteiro do 1 até 99, no máximo. Um ponto no território onde um caminho horizontal encontra um caminho vertical é chamado **interseção**. Cada interseção de um território é identificada pelos identificadores dos caminhos que a formam. Duas interseções são ditas **adjacentes** se forem conectadas por um caminho horizontal ou vertical sem outras interseções entre elas.

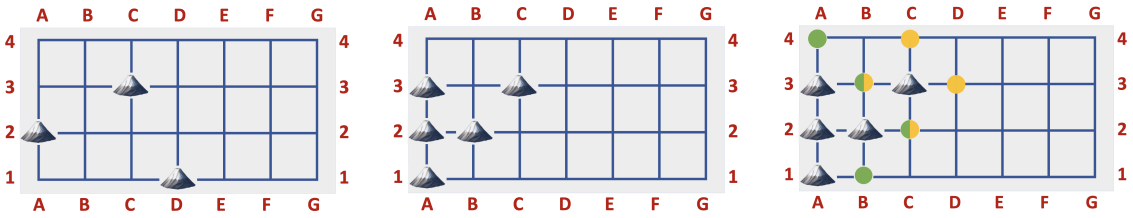
Num território, todas as interseções podem estar *livres* ou *ocupadas* por montanhas. O exemplo da Figura 1a) mostra um território formado por 7 caminhos verticais ($N_v = 7$) por 4 caminhos horizontais ($N_h = 4$), com montanhas situadas nas interseções `A2`, `C3` e `D1`.

A **ordem de leitura** das interseções do território é sempre feita da esquerda para a direita seguida de baixo para cima.

1.2 Conexões, cadeias de montanhas e vales

Duas interseções *ocupadas* (ou *livres*) estão **conetadas** se for possível traçar um percurso desde uma interseção para a outra passando sempre por interseções adjacentes *ocupadas* (ou *livres*). No exemplo da Figura 1b) a montanha em `A1` está conetada à montanha em `A3`, mas não está conetada à montanha em `C3`.

Chamamos **cadeia** de montanhas ao conjunto de uma ou mais interseções ocupadas por montanhas que estão todas conetadas entre si e que não estão conetadas a nenhuma outra montanha. Analogamente, definimos a cadeia de interseções livres como o conjunto de uma ou mais interseções livres que estão todas conetadas entre si e que não estão conetadas a nenhuma outra interseção livre. O **vale** de uma montanha é o conjunto de



a) b) c)

Figura 1: a) Território com três montanhas situadas nas interseções `A2`, `C3` e `D1`. b) Território com duas cadeias de montanhas: uma formada por quatro montanhas (`A1`, `A2`, `B2` e `A3`) e a outra formada por uma montanha (`C3`). c) Território com as interseções do vale da montanha `A1` (igual para `A2`, `B2` e `A3`) marcadas com um ponto verde e as interseções do vale da montanha `C3` marcadas em amarelo. As interseções `C2` e `B3` formam parte dos dois vales.

interseções livres adjacentes a essa montanha ou adjacente a uma montanha da mesma cadeia de montanhas. Os exemplos das Figuras 1b) e 1c) mostram duas cadeias de montanhas com os respetivos vales.

2 Trabalho a realizar

O objetivo do primeiro projeto é escrever um programa em Python que permita obter informações do estado de um território como o descrito anteriormente. Para isso, deverá definir o conjunto de funções solicitadas, assim como algumas funções auxiliares adicionais, caso seja necessário. Apenas as funções para as quais a verificação da correção dos argumentos é explicitamente pedida devem verificar a validade dos argumentos, para as outras assume-se que estão corretos.

2.1 Representação do território e das interseções

Considere que um *território* é representado internamente (ou seja, no seu programa) por um tuplo com N_v tuplos. Cada um dos N_v tuplos contêm N_h valores inteiros que representam as interseções de cada um dos caminhos verticais. Os valores inteiros podem tomar valores igual a 1 se a interseção corresponder a uma montanha ou 0 se estiver livre. Assim, o território da Figura 1a) é definido pelo tuplo `((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0),(0,0,0,0),(0,0,0,0))`.
Considere também que as interseções são representadas internamente como um tuplo de dois elementos, o primeiro a cadeia de carateres v correspondente ao caminho vertical onde se encontra e o segundo o valor inteiro h correspondente ao caminho horizontal. Assim, as montanhas presentes na Figura 1a) ocupam as interseções `('A',2)`, `('C',3)` e `('D',1)`.

2.1.1 eh_territorio: universal → booleano (1 valor)

eh_territorio(arg) recebe um argumento de qualquer tipo e devolve `True` se o seu argumento corresponde a um território e `False` caso contrário, sem nunca gerar erros. Nesta parte do projeto, considere que um território corresponde a um tuplo de tuplos como descrito, e que o território contêm no mínimo 1 caminho vertical e 1 caminho horizontal.

```
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0))
>>> eh_territorio(t)
True
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0))
>>> eh_territorio(t)
False
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,2,0,0),(0,0,0,0))
>>> eh_territorio(t)
False
```

2.1.2 obtem_ultima_intersecao: territorio → intersecao (0,5 valores)

obtem_ultima_intersecao(t) recebe um território e devolve a *intersecao* do extremo superior direito do território.

```
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0))
>>> obtem_ultima_intersecao(t)
('E', 4)
>>> t = ((0,1,0,0,0),(0,0,0,0,1),(0,0,1,0,1))
>>> obtem_ultima_intersecao(t)
('C', 5)
```

2.1.3 eh_intersecao: universal → booleano (1 valor)

eh_intersecao(arg) recebe um argumento de qualquer tipo e devolve `True` se o seu argumento corresponde a uma interseção e `False` caso contrário, sem nunca gerar erros. Nesta parte do projeto, considere que uma interseção corresponde a um tuplo como descrito.

```
>>> eh_intersecao(('B', 25))
True
>>> eh_intersecao((25, 'B'))
False
>>> eh_intersecao(('A', 200))
False
```

2.1.4 eh_intersecao_valida: territorio × intersecao → booleano (0,5 valores)

eh_intersecao_valida(t, i) recebe um território e uma interseção, e devolve `True` se a interseção corresponde a uma interseção do território, e `False` caso contrário.

```
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0))
>>> eh_intersecao_valida(t, ('A', 1))
True
>>> eh_intersecao_valida(t, ('A', 2))
True
>>> eh_intersecao_valida(t, ('A', 50))
False
```

2.1.5 eh_intersecao_livre: territorio × intersecao → booleano (0,5 valores)

eh_intersecao_livre(t, i) recebe um território e uma interseção do território, e devolve `True` se a interseção corresponde a uma interseção livre (não ocupada por montanhas) dentro do território e `False` caso contrário.

```
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0))
>>> eh_intersecao_livre(t, ('A', 1))
True
>>> eh_intersecao_livre(t, ('A', 2))
False
```

2.1.6 obtem_intersecoes_adjacentes: territorio × intersecao → tuplo (1,5 valores)

obtem_intersecoes_adjacentes(t, i) recebe um território e uma interseção do território, e devolve o tuplo formado pelas interseções válidas adjacentes da interseção em ordem de leitura de um território.

```
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0))
>>> obtem_intersecoes_adjacentes(t, ('C',3))
(('C', 2), ('B', 3), ('D', 3), ('C', 4))
>>> obtem_intersecoes_adjacentes(t, ('A',1))
(('B', 1), ('A', 2))
```

2.1.7 ordena_intersecoes: tuplo → tuplo (1 valor)

ordena_intersecoes(tup) recebe um tuplo de interseções (potencialmente vazio) e devolve um tuplo contendo as mesmas interseções ordenadas de acordo com a ordem de leitura do território.

```
>>> tup = (('A',1), ('A',2), ('A',3), ('B',1), ('B',2), ('B',3))
>>> ordena_intersecoes(tup)
(('A',1), ('B',1), ('A',2), ('B',2), ('A',3), ('B',3))
```

2.1.8 territorio_para_str: territorio → cad. carateres (2 valores)

territorio_para_str(t) recebe um território e devolve a cadeia de caracteres que o representa (a representação externa ou representação "para os nossos olhos"), de acordo com o exemplo na seguinte interação. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'territorio_para_str: argumento invalido'`.

```
>>> t = ((0,1,0,0),(0,0,0,0))
>>> territorio_para_str(t)
'   A B\n 4 . .  4\n 3 . .  3\n 2 X .  2\n 1 . .  1\n   A B'
>>> t=((1,1,1,0,0,0,0,0,1,1),)
>>> territorio_para_str(t)
('   A\n10 X 10\n 9 X  9\n 8 .  8\n 7 .  7\n 6 .  6\n 5 .  5\n'
 ' 4 .  4\n 3 X  3\n 2 X  2\n 1 X  1\n   A')
>>> t = ((0,1,0,0),(0,0,0,0),(0,0,1,0),(1,0,0,0),(0,0,0,0))
>>> print(territorio_para_str(t))
   A B C D E
 4 . . . . .  4
 3 . . X . .  3
 2 X . . . .  2
 1 . . . X .  1
   A B C D E
```

2.2 Funções das cadeias de montanhas e dos vales

2.2.1 obtem_cadeia: territorio × intersecao → tuplo (2 valores)

obtem_cadeia(t,i) recebe um território e uma interseção do território (ocupada por uma montanha ou livre), e devolve o tuplo formado por todas as interseções que estão conetadas a essa interseção ordenadas (incluida si própria) de acordo com a ordem de leitura de um território. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'obtem_cadeia: argumentos invalidos'`.

```
>>> t = ((1,1,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> print(territorio_para_str(t))
   A B C D E
 4 . . . . .  4
 3 X . X . .  3
```

```
 2 X X . . .  2
 1 X . . . .  1
   A B C D E
>>> obtem_cadeia(t,('A',2))
(('A', 1), ('A', 2), ('B', 2), ('A', 3))
>>> obtem_cadeia(t, ('C', 3))
(('C', 3),)
>>> obtem_cadeia(t,('A',4))
(('B', 1), ('C', 1), ('D', 1), ('E', 1), ('C', 2), ('D', 2),
 ('E', 2), ('B', 3), ('D', 3), ('E', 3), ('A', 4), ('B', 4),
 ('C', 4), ('D', 4), ('E', 4))
```

2.2.2 obtem_vale: territorio × intersecao → tuplo (1,5 valores)

obtem_vale(t,i) recebe um território e uma interseção do território ocupada por uma montanha, e devolve o tuplo (potencialmente vazio) formado por todas as interseções que formam parte do vale da montanha da interseção fornecida como argumento ordenadas de acordo à ordem de leitura de um território. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'obtem_vale: argumentos invalidos'`.

```
>>> t = ((1,1,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> obtem_vale(t,('A',1))
(('B', 1), ('C', 2), ('B', 3), ('A', 4))
>>> obtem_vale(t,('C',3))
(('C', 2), ('B', 3), ('D', 3), ('C', 4))
>>> obtem_vale(t,('B',1))
Traceback (most recent call last): <...>
ValueError: obtem_vale: argumentos invalidos
```

2.3 Funções de informação de um território

2.3.1 verifica_conexao: territorio × intersecao ×-intersecao → booleano (0,5 valores)

verifica_conexao(t,i1,i2) recebe um território e duas interseções do território e devolve `True` se as duas interseções estão conetadas e `False` caso contrário. Se algum dos argumentos dado for inválido, a função deve gerar um erro com a mensagem `'verifica_conexao: argumentos invalidos'`.

```
>>> t = ((1,1,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> verifica_conexao(t, ('A',1), ('A',3))
True
```

```
>>> verifica_conexao(t, ('A',1), ('C',3))
False
>>> verifica_conexao(t, ('A',4), ('B',1))
True
```

2.3.2 calcula_numero_montanhas: territorio → int (0,5 valores)

calcula_numero_montanhas(t) recebe um território e devolve o número de interseções ocupadas por montanhas no território. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'calcula_numero_montanhas: argumento invalido'`.

```
>>> t = ((1,1,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> calcula_numero_montanhas(t)
5
>>> t = ((1,0,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> calcula_numero_montanhas(t)
4
```

2.3.3 calcula_numero_cadeias_montanhas: territorio → int (1,5 valores)

calcula_numero_cadeias_montanhas(t) recebe um território e devolve o número de cadeias de montanhas contidas no território. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'calcula_numero_cadeias_montanhas: argumento invalido'`.

```
>>> t = ((1,1,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> calcula_numero_cadeias_montanhas(t)
2
>>> t = ((1,0,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> calcula_numero_cadeias_montanhas(t)
4
```

2.3.4 calcula_tamanho_vales: territorio → int (2 valores)

calcula_tamanho_vales(t) recebe um território e devolve o número total de interseções diferentes que formam todos os vales do território. Por exemplo, na Figura 1c) o tamanho dos vales é de 6 interseções, marcadas com pontos amarelos, verdes e verde-amarelos. Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'calcula_tamanho_vales: argumento invalido'`.

```
>>> t = ((1,1,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> calcula_tamanho_vales(t)
```

```
6
>>> t = ((1,0,1,0),(0,1,0,0),(0,0,1,0),(0,0,0,0),(0,0,0,0))
>>> calcula_tamanho_vales(t)
7
```

3 Condições de Realização e Prazos

- A entrega do 1º projeto será efetuada exclusivamente por via eletrónica. Para submeter o seu projeto deverá realizar pelo menos uma atualização do repositório remoto GitLab fornecido pelo corpo docente, até às **17:00 do dia 23 de Outubro de 2023**. Depois desta hora, qualquer atualização do repositório será ignorada. Não serão aceites submissões de projetos por outras vias sob pretexto algum.

- A solução do projeto deverá consistir apenas num único ficheiro com extensão *.py* contendo todo o código do seu projeto.

- Cada aluno tem direito a **15 submissões sem penalização**. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.

- Será considerada para avaliação a **última** submissão (mesmo que tenha pontuação inferior a submissões anteriores). Deverá, portanto, verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada.

- Submissões que não corram nenhum dos testes automáticos por causa de pequenos erros de sintaxe ou de codificação, poderão ser corrigidos pelo corpo docente, incorrendo numa penalização de três valores.

- Não é permitida a utilização de qualquer módulo ou função não disponível built-in no Python 3.

- Pode, ou não, haver uma discussão oral do trabalho e/ou uma demonstração do funcionamento do programa (será decidido caso a caso).

- Lembre-se que no Técnico, a fraude académica é levada muito a sério e que a cópia numa prova (projetos incluídos) leva à reprovação na disciplina e eventualmente a um processo disciplinar. Os projetos serão submetidos a um sistema automático de deteção de cópias[1], o corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.

[1] `https://theory.stanford.edu/~aiken/moss`

4 Submissão

A submissão do projeto de FP é realizada atualizando o repositório remoto GitLab privado fornecido pelo corpo docente para cada aluno (ou grupo de projeto). O endereço web do repositório do projeto dos alunos é `https://gitlab.rnl.tecnico.ulisboa.pt/ist-fp/fp23/prj1/(curso)/(grupo)`, onde:

- `(curso)` pode ser `leic-a`, `leic-t`, `leti` ou `leme`;
- `(grupo)` pode ser:
  - o ist-id para os alunos da LEIC-A, LEIC-T e LETI (ex. `ist190000`);
  - `g` seguido dos dois dígitos que identificam o número de grupo de projeto dos alunos da LEME (ex. `g00`).

Sempre que é realizada uma nova atualização do repositório remoto é desencadeado o processo de avaliação automática do projeto e é contabilizada uma nova submissão. Quando a submissão tiver sido processada, poderá visualizar um relatório de execução com os detalhes da avaliação automática do seu projeto em `http://fp.rnl.tecnico.ulisboa.pt/reports/(grupo)/`. Adicionalmente, receberá no seu email o mesmo relatório. Se não receber o email ou o relatório web aparentar não ter sido atualizado, contacte com o corpo docente. Note que o sistema de submissão e avaliação não limita o número de submissões simultâneas. Um número elevado de submissões num determinado momento, poderá ocasionar a rejeição de alguns pedidos de avaliação. Para evitar problemas de último momento, **recomenda-se que submeta o seu projeto atempadamente**.

Detalhes sobre como aceder ao GitLab, configurar o par de chaves SSH, executar os comandos de Git e recomendações sobre ferramentas, encontram-se na página da disciplina na seção "Material de Apoio - Ambiente de Desenvolvimento"[2].

5 Classificação

A nota do projeto será baseada nos seguintes aspetos:

1. **Avaliação automática (80%).** A avaliação da correta execução será feita com um conjunto de testes unitários utilizando o módulo de Python pytest[3].
   Serão usados um conjunto de testes públicos (disponibilizados na página da disciplina) e um conjunto de testes privados. Como a avaliação automática vale 80% (equivalente a 16 valores) da nota do projeto, uma submissão obtém a nota máxima de 1600 pontos.
   O facto de um projeto completar com sucesso os testes públicos fornecidos (naturalmente, só devem submeter depois do programa passar todos estes testes) não

[2] `https://fenix.tecnico.ulisboa.pt/disciplinas/FProg3/2023-2024/1-semestre/ambiente-de-desenvolvimento`

[3] `https://docs.pytest.org/en/7.4.x/`

implica que esse projeto esteja totalmente correto, pois estes não são exaustivos. É da responsabilidade de cada aluno garantir que o código produzido está de acordo com a especificação do enunciado usando testes próprios adicionais, de forma a completar com sucesso os testes privados.

2. **Avaliação manual (20%).** Estilo de programação e facilidade de leitura. Em particular, serão consideradas as seguintes componentes:

   - Boas práticas (1,5 valores): serão considerados entre outros a clareza do código, a integração de conhecimento adquirido durante a UC e a criatividade das soluções propostas.
   - Comentários (1 valor): deverão incluir a assinatura das funções definidas, comentários para o utilizador (*docstring*) e comentários para o programador.
   - Tamanho de funções, duplicação de código e abstração procedimental (1 valor).
   - Escolha de nomes (0,5 valores).

6 Recomendações e aspetos a evitar

As seguintes recomendações e aspetos correspondem a sugestões para evitar maus hábitos de trabalho (e, consequentemente, más notas no projeto):

- Leia todo o enunciado, procurando perceber o objetivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer as suas questões.
- No processo de desenvolvimento do projeto, comece por implementar as várias funções pela ordem apresentada no enunciado, seguindo as metodologias estudadas na disciplina.
- Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes *outputs*, conforme ilustrado nos vários exemplos.
- Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá sentir a Lei de Murphy em funcionamento (todos os problemas são mais difíceis do que parecem; tudo demora mais tempo do que nós pensamos; e se alguma coisa puder correr mal, ela vai correr mal, na pior das alturas possíveis).
- Não duplique código. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos.
- Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação.

- A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada.
- Quando o programa gerar um erro, preocupe-se em descobrir qual a causa do erro. As "marteladas" no código têm o efeito de distorcer cada vez mais o código.