Informática para Ciências e Engenharias

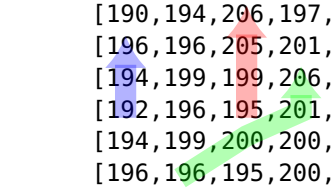
Trabalho Prático Nº 1 — 2018/19

1 Introdução

Um problema clássico que se coloca em várias situações é o de procurar um ponto óptimo por sucessivas deslocações a partir dum ponto de partida. No exemplo que vamos considerar neste trabalho, é dado um mapa de uma região (montanhosa) organizado numa grelha cujas células contém a sua altitude média (em metros).

No exemplo simples ilustrado na matriz `teste`, um mapa é dividido numa matriz de 7 linhas e 5 colunas, cada uma representando um quadrado com 20m de lado e organizada como uma lista de listas. O problema da subida é o de atingir a célula mais "alta" a partir de uma célula de partida, visitando todas as células do caminho. Eis algumas possibilidades.

```
teste = [
   [206,205,204,190,208],
   [190,194,206,197,203],
   [196,196,205,201,193],
   [194,199,199,206,205],
   [192,196,195,201,193],
   [194,199,200,200,205],
   [196,196,195,200,193] ]
```



1. A partir da célula `teste[4][2]`, com altitude 195, e seguindo sempre em direção Norte (isto é, para células na mesma coluna e decrementando a linha), visitam-se as células `teste[4][2]`, `teste[3][2]`, `teste[2][2]` e `teste[1][2]`, com altitudes de, respectivamente, 199, 205 e 206, como ilustrado pela seta vermelha. A célula `teste[0][2]` já não fará parte do caminho porque a sua altitude, 204, é menor que 206.

2. Igualmente restringindo o caminho à direcção Norte, mas partindo da célula `teste[4][0]`, obtemos `teste[4][0]`, `teste[3][0]` e `teste[2][0]`, com altitudes 192, 194 e 196, ilustrado pela seta azul. Como a célula a seguir nesta direcção tem uma altitude inferior, o caminho termina em `teste[2][0]`.

3. Se simplesmente subirmos para a célula mais alta na vizinhança, até não haver nenhuma mais alta, e sem restringir a direcção, então partindo de `teste[6][1]`, o caminho seguirá por `teste[5][2]`, `teste[4][3]` e terminará em `teste[3][3]`, como ilustrado pela seta verde, seguindo sempre para a célula mais alta das 8 vizinhas em cada passo do caminho.

Outro aspecto a considerar é a distância percorrida. Esta será a distância Euclideana tendo em conta a dimensão dos quadrados representados pelas células da matriz e a diferença de altitude. Assim, entre a célula na linha r_1 e coluna c_1 e a célula na linha r_2 e coluna c_2, com altitudes respectivamente a_1 e a_2, e sendo w a largura do quadrado representado por cada célula, a distância será, para cada trecho:

$$d = \sqrt{(w(r_1 - r_2))^2 + (w(c_1 - c_2))^2 + (a_1 - a_2)^2}$$

Para o terceiro exemplo acima, da subida pela célula vizinha mais alta, a distância total será a soma dos três trechos:

$$\sqrt{(20(6-5))^2 + (20(1-2))^2 + (196-200)^2} = 28.5657137141714$$

$$\sqrt{(20(5-4))^2 + (20(2-3))^2 + (200-201)^2} = 28.30194339616981$$

$$\sqrt{(20(4-3))^2 + (20(3-3))^2 + (201-206)^2} = 20.615528128088304$$

$$total = 77.48318523842951$$

2 Objectivo do Trabalho

Neste trabalho, pretende-se que implemente as funções indicadas abaixo. Para as testar vai necessitar das matrizes `teste` e `estrela` fornecidas no módulo `tp1_data.py`. Deverá importar estas variáveis, que representam as matrizes como listas de listas. A variável `teste` tem a matriz dos exemplos anteriores. Para esta, assuma que cada célula representa um quadrado de 20m de lado. A variável `estrela` tem uma matriz de 100 por 100 células representando a geografia da parte da Serra da Estrela indicada nas imagens *mapa.png* e *estrela.png* e representadas na Figura 1. Nesta matriz cada célula representa um quadrado de 180m de lado. As altitudes, em ambos os casos, estão em metros.

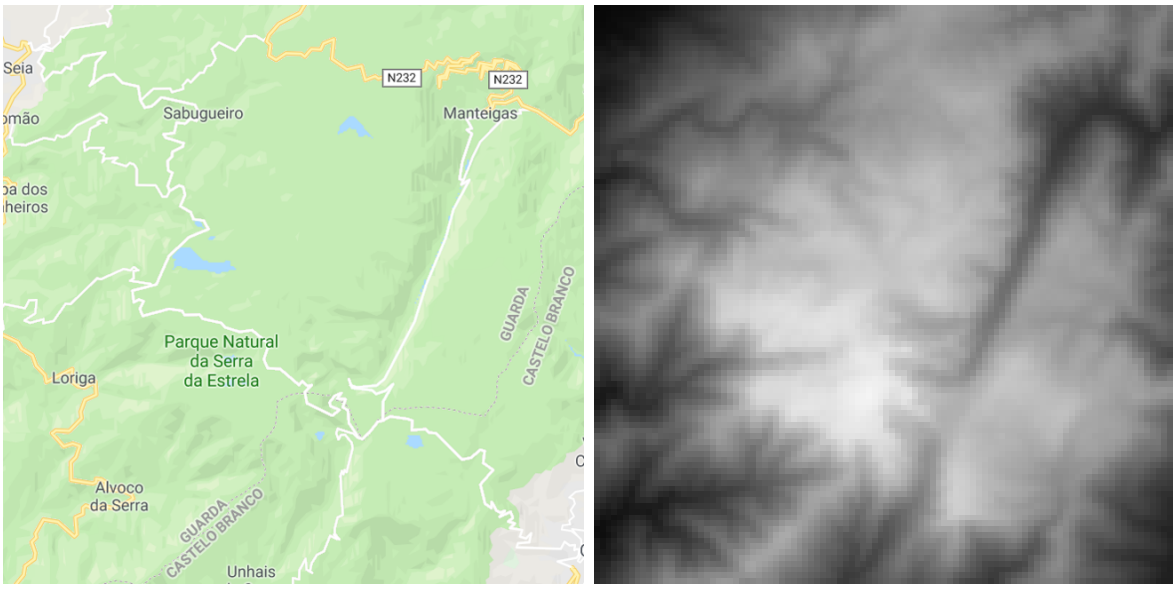


Figura 1: Região representada na variável `estrela` e respectivo mapa de altitude, correspondente aos valores da matriz (mais claro representa pontos mais altos).

2.1 Distância

Implemente uma função com esta assinatura:

```
def distancia(altitudes, lado, p1, p2):
```

que retorna a distância entre as células indicadas por `p1` e `p2` num mapa definido na matriz `altitudes` como uma lista de listas com células quadradas de largura `lado`. Os parâmetros `p1` e `p2` são tuplos com dois valores inteiros cada um, indicando respectivamente a linha e coluna de cada uma das duas células entre as quais se quer medir a distância. A distância deve ser medida pela fórmula dada na Introdução.

Por exemplo, usando a matriz `teste`, estas chamadas à função devem dar estes resultados:

```
In : distancia(teste, 20, (2, 0), (3, 1))
Out: 28.442925306655784

In : distancia(teste, 20, (2, 1), (6, 4))
Out: 100.04498987955368

In : distancia(teste, 20, (2, 3), (2, 4))
Out: 21.540659228538015
```

2.2 Distância Percorrida

Implemente uma função com assinatura

```
def distancia_percorrida(altitudes, lado, caminho):
```

que retorna a distância percorrida no caminho indicado, que é uma lista de tuplos com a linha e coluna de cada célula visitada nesse caminho.

Por exemplo:

```
In : caminho =  [ (2,0),(3,1),(3,2),(4,3) ]
In : distancia_percorrida(teste, 20, caminho)
Out: 76.79781906417143
```

2.3 Subida para Norte

Implemente uma função com assinatura

```
def subir_norte(altitudes, inicio):
```

que, a partir da célula indicada em `inicio`, que é um tuplo com a linha e a coluna, retorna um tuplo com as coordenadas da célula atingida numa subida para Norte (decrementando a linha) até atingir que a célula vizinha a Norte não tenha uma altitude maior ou tenha atingido o limite da matriz. Por exemplo:

```
In : subir_norte(teste, (3, 0) )
Out: (2, 0)

In : subir_norte(teste, (1, 0) )
Out: (0, 0)
```

2.4 Vizinhas

Implemente uma função com assinatura

```
def vizinhas(alturas, celula):
```

que retorna uma lista de tuplos com as coordenadas (linha e coluna) das células vizinhas da célula indicada em `celula`, que é também um tuplo com linha e coluna. As células vizinhas são as células adjacentes nos quatro pontos cardeais (N, S, E, O) e quatro pontos colaterais (NE, NO, SO, SE). Assim, no máximo cada célula poderá ter 8 vizinhas. No entanto, se estiver na margem da matriz terá menos vizinhas, uma vez que se considerará apenas células dentro da matriz.

```
In : vizinhas(teste, (3,3) )
Out: [(2, 2), (2, 3), (2, 4), (3, 2), (3, 4), (4, 2), (4, 3), (4, 4)]

In : vizinhas(teste, (5,4) )
Out: [(4, 3), (4, 4), (5, 3), (6, 3), (6, 4)]

In : vizinhas(teste, (0,0) )
Out: [(0, 1), (1, 0), (1, 1)]
```

2.5 Vizinha Mais Alta

Implemente uma função com assinatura

```
def vizinha_mais_alta(altitudes, celula):
```

que retorna um tuplo com as coordenadas (linha e coluna) da célula vizinha daquela indicada em `celula` com a maior altitude se for mais alta que a célula de referência. Se nenhuma vizinha for mais alta que a célula indicada em `celula`, a função deve retornar as coordenadas de `celula`. Se existirem várias células vizinhas com a maior altitude a função pode retornar qualquer uma delas. Por exemplo:

```
In : vizinha_mais_alta(teste, (5,3) )
Out: (5, 4)

In : vizinha_mais_alta(teste, (1,1) )
Out: (0, 0)

In : vizinha_mais_alta(teste, (1,2) )
Out: (1, 2)
```

2.6 Subir Mais Alto

Implemente uma função com assinatura

```
def subir_mais_alto(altitudes, celula):
```

que dada uma célula do mapa retorna o caminho, uma lista de tuplos com a linha e coluna de cada célula visitada, tal que cada célula passe para a célula vizinha de maior altitude até atingir um pico (*i.e.* uma célula sem células vizinhas mais altas). Por exemplo:

```
In : subir_mais_alto(teste, (1, 0))
Out: [(1, 0), (0, 0)]

In : subir_mais_alto(teste, (6, 1))
Out: [(6, 1), (5, 2), (4, 3), (3, 3)]
```

Note que o caminho inclui a célula inicial.

2.7 Extra: Escolher o Caminho

Nota: esta alínea exige a implementação de funções auxiliares e não é tão simples quanto as restantes. Além disso, vale apenas 1 valor da cotação total de 20 valores para este trabalho, por isso devem considerar esta alínea apenas depois de concluir o restante trabalho.

Para escolher o melhor caminho entre duas células da matriz, implemente uma função com a assinatura

```
def escolhe_caminho(altitudes, lado, inicio, destino):
```

que recebe o mapa, o lado do quadrado de terreno representado por cada célula e duas células da matriz, cada uma representada por um tuplo com a linha e coluna. Esta função deve devolver o caminho da célula de início à célula de destino (inclusive) como uma lista de tuplos. Para escolher a célula vizinha a cada passo do caminho vamos considerar um valor de «custo» associado a essa

célula que representa o nosso desejo de não subir desnecessariamente e de nos aproximarmos do destino. Assim, o «custo» associado a cada célula será a soma da altitude dessa célula com a raiz quadrada da distância entre essa célula e o destino multiplicada por 50:

$$custo = altitude + 50\sqrt{distancia}$$

Calculado este valor para todas as células vizinhas vamos escolher aquela com o menor «custo». Ou seja, a que nos aproxima mais do destino sem nos obrigar a subir desnecessariamente. O factor de 50 foi determinado empiricamente para este mapa para calibrar o peso de cada factor na decisão e a raiz quadrada da distância faz com que diferenças na distância entre células vizinhas se vão tornando mais importantes conforme nos aproximamos do destino.

Para criar esta função podem basear-se no resto do trabalho, porque o problema é semelhante ao da subida, mas provavelmente terão de planear e implementar funções auxiliares antes de conseguir implementar esta.

No módulo `tp1_data.py` há uma função auxiliar para desenhar o caminho. Requer os ficheiros *estrela.png* e *mapa.png* na pasta de trabalho e gera os ficheiros *caminho.png* e *caminho_mapa.png* com o caminho desenhado a vermelho. Por exemplo, para escolher o caminho de Manteigas, na célula `estrela[21][81]`, ao ponto mais alto da Serra da Estrela, na célula `estrela[67][47]`, com esta matriz e a heurística indicada, este deverá dar o resultado indicado para o caminho e o desenho do caminho está na Figura 2:

```
In : caminho = escolhe_caminho(estrela,180,(21,81),(67,47))

In : print(caminho)
[(21, 81), (22, 81), (23, 80), (24, 79), (25, 79), (26, 79), (27, 79),
 (28, 78), (29, 78), (30, 77), (31, 77), (32, 77), (33, 76), (34, 76),
 (35, 75), (36, 75), (37, 75), (38, 75), (39, 75), (40, 74), (41, 73),
 (42, 73), (43, 73), (44, 72), (45, 72), (46, 71), (47, 71), (48, 70),
 (49, 70), (50, 69), (51, 69), (52, 69), (53, 68), (54, 67), (55, 67),
 (56, 66), (57, 65), (58, 66), (59, 66), (60, 65), (61, 64), (62, 63),
 (63, 62), (64, 61), (64, 60), (64, 59), (64, 58), (64, 57), (64, 56),
 (64, 55), (64, 54), (64, 53), (63, 52), (64, 51), (65, 50), (66, 49),
 (67, 48), (67, 47)]

In : desenha_caminho(caminho)
```

3 Critérios de Avaliação do Trabalho

De acordo com o Regulamento de Avaliação de Conhecimentos da FCT/UNL,[1] os estudantes directamente envolvidos numa fraude são liminarmente reprovados na disciplina. Em ICE, considera-se que um aluno que **dá** ou que **recebe** código num trabalho comete fraude. Os alunos que cometerem fraude num trabalho não obterão frequência.

Os trabalhos serão avaliados de acordo com os seguintes critérios.

- Utilização correta dos elementos básicos de Python.
- Implementação correcta das funções pedidas.
- Código legível, com nomes adequados e funções documentadas.

[1] Em `http://www.fct.unl.pt/sites/default/files/documentos/estudante/informacao_academica/Reg_Aval.pdf`

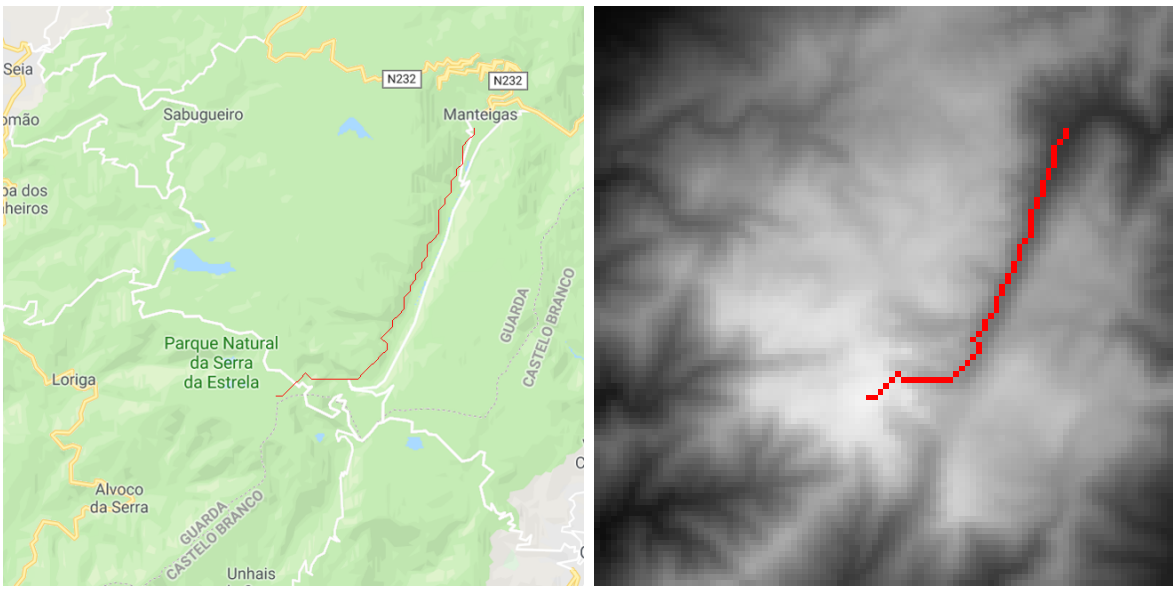


Figura 2: Caminho entre Manteigas e o alto da Serra com a heurística indicada.

A maior parte da cotação total corresponde à implementação das funções pedidas nos pontos 2.1 a 2.6. O estilo do código (nomes, organização e documentação) terá um peso pequeno, mas ainda significativo.

Não implementar função pedida no ponto 2.7 (encontrar o caminho) desconta apenas um valor na nota final do trabalho. Dado o grau de dificuldade dessa função, recomendo que só tentem implementá-la depois de concluído tudo o resto e apenas se acharem o restante trabalho aborrecido por ser demasiado fácil.

Qualquer omissão ou incorrecção na implementação contribui para reduzir a nota, mas é sempre preferível entregar um trabalho incompleto ou com erros do que não entregar nada.

A nota final do trabalho prático estará sujeita a ajustes conforme a defesa de nota que cada aluno, individualmente, fará no dia do primeiro teste. **Se não puder vir ao teste deve combinar a defesa de nota com o docente das práticas**.

4 Testes, entrega e enunciado

Serão disponibilizados na página da disciplina os módulos com dados e um script para testes locais que poderão usar para testar as vossas funções antes de submeter o trabalho. Será também publicado um guião com as instruções detalhadas para a entrega.

Esse material, bem como este enunciado, poderão sofrer pequenas alterações caso for necessário esclarecer algumas dúvidas recorrentes. Por favor prestem atenção aos avisos caso haja actualizações e mantenham as vossas versões actualizadas.