TÉCNICO LISBOA

Fundamentos de Programação

Ano letivo 2018-19

Primeiro Projeto

20 de Outubro de 2018

Hello Quantum Técnico

1 A computação quântica[1]

Os computadores tradicionais armazenam e processam informação usando bits, entidades que tomam um de dois valores possíveis, os quais designamos por 0 e 1.

Nos computadores quânticos a informação é representada por bits quânticos ou *qubits* (de "quantum bits"). Representaremos cada qubit por um retângulo que contém um par de *células* com círculos coloridos, como se mostra na Figura 1. Cada círculo pode apresentar uma de duas cores, branco ou preto, ou não estar colorido. Os estados possíveis das células do qubit são as seguintes: se o círculo for branco, a célula está ativa; se o círculo for preto, a célula está inativa; se o círculo não estiver colorido, a célula está simultaneamente ativa e não ativa (estado incerto).

Figura 1: Qubit, em que a célula inferior está ativa (círculo branco) e a superior está no estado incerto (círculo não colorido).

1.1 Questionando os qubits

O processo de obter um bit a partir de um qubit é essencialmente semelhante a colocar uma questão de resposta binária, ou seja, uma pergunta com apenas duas respostas possíveis, 0 ou 1.

As questões mais frequentes dizem respeito às cores de cada um dos círculos de um qubit, por exemplo: "qual a cor do círculo inferior?". A resposta a esta questão é 0 se o círculo for preto e 1 se o círculo for branco. No entanto, se o círculo não estiver colorido (também dito que está *vazio*), a resposta

[1] Adaptado de *Hello Quantum: Taking your first steps into quantum computation* by James Wootton https://medium.com/qiskit/hello-quantum-2c1c00fe830c

não é determinística, mas terá forçosamente de ser 0 ou 1. Neste caso, a resposta é determinada aleatoriamente. O mesmo acontece se a pergunta abordar o círculo superior.

Contudo, se forem colocadas simultaneamente as duas perguntas, o qubit apenas está confiante numa das respostas, ou seja, um dos círculos está sempre vazio. Isto significa que não é possível conhecer o estado de ambas as células do qubit ao mesmo tempo. Isto acontece porque cada qubit só consegue dar uma resposta de cada vez, devido à incerteza inerente ao seu estado.

Uma forma de resolver esta situação é considerarmos pares de qubits e registarmos a concordância entre células dos dois qubits adicionando mais células à nossa representação: uma por cada par de questões que possam ser colocadas. Denominamos estas células adicionais como *células de observação*. Na Figura 2 ilustra-se um par de qubits (à esquerda) e a representação da concordância dos seus círculos inferiores (à direita). A nova célula de observação tem um círculo preto porque ambos os qubits estão de acordo sobre a cor dos seus círculos inferiores.

Figura 2: Um par de qubits (à esquerda); um par de qubits, aliados à representação da concordância das células inferiores (à direita).

Por outro lado, se quisermos representar a discordância entre duas células utilizamos um círculo branco. Na Figura 3, representamos a discordância entre as células superiores dos dois qubits com um círculo branco na célula de observação superior[2].

Figura 3: Um par de qubits (à esquerda); um par de qubits, aliados à representação da concordância das células inferiores e da discordância das células superiores (à direita).

Quando a questão envolve a cor do círculo inferior de um qubit e do círculo superior do outro, a concordância e discordância registam-se nas células de observação da esquerda ou da direita. A Figura 4 (à direita), representa a incerteza da concordância das células questionadas.

[2] Note que em lógica quântica a relação entre duas células em estado incerto pode ser de concordância, discordância ou desconhecida. Ou seja, na Figura 3, o círculo de observação superior poderia também ser preto ou não ter cor.

Figura 4: Um par de qubits (à esquerda); um par de qubits, aliados à representação da concordância e discordância das células inferiores e superiores, conjuntamente com a incerteza aliada às respostas cruzadas de ambos as células (à direita).

Em conjunto, os quatro novos círculos dão uma descrição completa das respostas expectáveis para quaisquer questões de concordância que coloquemos a um par de qubits.

1.2 Operações sobre qubits

Todos os qubits iniciam a sua vida num estado semelhante – quando questionados sobre a sua célula inferior, todos responderão 0 (círculo preto). Portanto, dois qubits no seu estado inicial podem ser representados como na Figura 5, onde se inclui a concordância das células inferiores, e a incerteza associada a quaisquer outras questões. Os qubits são alterados através de portas (do inglês, *gates*), existindo três portas para o fazer, que designaremos por porta X, porta Y e porta Z.

Figura 5: Dois qubits no estado inicial.

A operação mais simples que podemos aplicar a um qubit é alterar o estado da sua célula inferior, de preto para branco e vice-versa. Esta operação é executada através da porta X. Repare que os estados de incerteza (círculos vazios) não são alterados por esta operação, mantendo-se inalterados. A alteração da célula inferior de um qubit afeta naturalmente a forma como o qubit se relaciona com outros qubits, e, portanto, as concordâncias tornam-se discordâncias e vice-versa. A consequência da operação da porta X é a alteração do estado de uma linha completa de células, como se ilustra na Figura 6 e na Figura 7 quando aplicada ao qubit da esquerda e da direita, respetivamente.

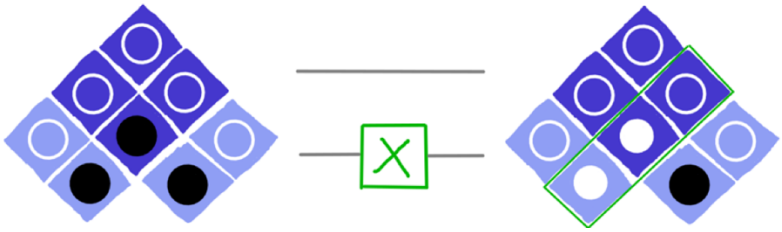

Figura 6: Aplicação da porta X ao qubit da esquerda.

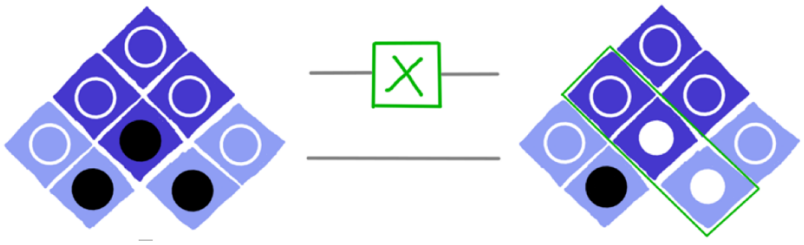

Figura 7: Aplicação da porta X ao qubit da direita.

De forma semelhante, a porta Z altera o estado da célula superior de um qubit. A Figura 8 ilustra a aplicação da porta Z ao qubit da esquerda.

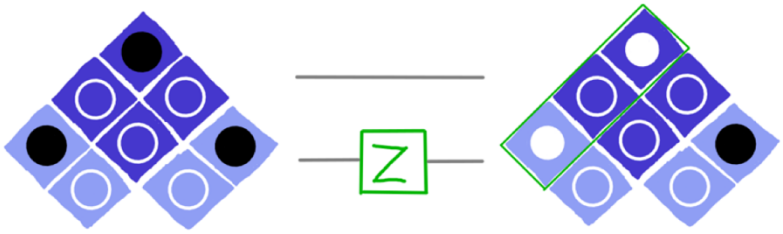

Figura 8: Aplicação da porta Z ao qubit da esquerda.

A porta H, por sua vez, troca os estados das células superiores e inferiores de um qubit entre si, alterando consequentemente as células indicadoras de concordância e discordância com o outro qubit.

Figura 9: Aplicação da porta H ao qubit da esquerda.

2 O jogo Hello Quantum

O jogo *Hello Quantum* é um puzzle desenhado para ensinar os princípios introdutórios da computação quântica. O jogo foi desenhado pela IBM Research em Yorktown Heights, New York em colaboração com o Professor James Wootton da Universidade de Basileia, Suíça.[3]

[3] https://helloquantum.mybluemix.net/#

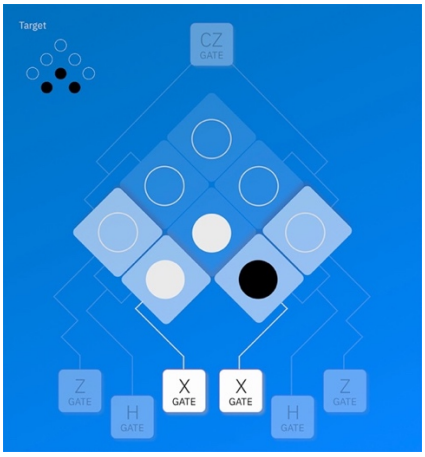

Figura 10: Exemplo de um jogo de *Hello Quantum*.

O objetivo do jogo é descobrir a sequência mínima de operações (através do uso de portas) que transformam um tabuleiro inicial, ilustrado no centro da Figura 10, num tabuleiro objetivo apresentado no canto superior esquerdo da mesma figura (*target*).

Como a Figura 10 ilustra, cada tabuleiro do jogo é constituído por *8* (3^2-1) células, dispostas em *2* linhas de *3* células seguidas de uma última linha com apenas *2* células. O tabuleiro é visualizado com uma rotação de 45º à esquerda.

No jogo existem 3 operadores, ou *portas*, que atuam nos qubits, modificando assim as células do tabuleiro: a porta X, a porta Z e a porta H. Estes operadores atuam num único ou em dois conjuntos de células:

- *Porta X*: o operador correspondente a esta porta, aplicado a um qubit, tem como resultado a inversão do valor da célula inferior desse qubit, e consequentemente, de todas as restantes células que estejam na mesma linha ou coluna, caso seja o qubit da esquerda ou da direita, respetivamente.

  A Figura 11 ilustra a aplicação da porta X ao qubit da esquerda do tabuleiro (à esquerda da figura) e o da direita (à direita da figura).

Figura 11: Aplicação da porta X ao qubit da esquerda do tabuleiro (i), resultando no tabuleiro (ii); aplicação da porta X ao qubit da direita do tabuleiro (i), resultando no tabuleiro (iii).

- *Porta Z*: o operador correspondente a esta porta, aplicado a um qubit, apresenta resultados semelhantes aos da *porta X*, excetuando que opera sobre a célula superior.

A Figura 12 ilustra a aplicação da porta Z ao qubit da esquerda do tabuleiro (à esquerda da figura) e da direita (à direita da figura).

(i) (ii) (iii)

Figura 12: Aplicação da porta Z ao qubit da esquerda do tabuleiro (i), resultando no tabuleiro (ii); aplicação da porta Z ao qubit da direita do tabuleiro (ii), resultando no tabuleiro (iii).

- *Porta H*: o operador correspondente a esta porta, aplicado a um qubit, tem como resultado a troca de estado de ambas as suas células entre si, com o resultado de afetar, consequentemente, as linhas ou colunas onde se encontra, quando aplicado ao qubit da esquerda ou direita, respetivamente.

  A Figura 13 ilustra a aplicação da *porta H* ao qubit da esquerda do tabuleiro (à esquerda da figura) e da direita (à direita da figura).

(i) (ii) (iii)

Figura 13: Aplicação da porta H ao qubit da esquerda do tabuleiro (i), resultando no tabuleiro (ii); aplicação da porta H ao qubit da direita do tabuleiro (ii), resultando no tabuleiro (iii).

3 Trabalho a realizar

O objetivo do primeiro projeto é escrever um programa em Python, correspondendo às funções descritas nesta secção, que permita aplicar os diferentes operadores ou portas a um tabuleiro com 8 células (3^2-1), como o ilustrado na Figura 10. Para isso, deverá definir um conjunto de funções que correspondam à aplicação de cada um dos operadores apresentados e algumas funções auxiliares para manipulação do tabuleiro.

3.1 Representação do tabuleiro

Considere que um tabuleiro é representado internamente (ou seja, no seu programa) por um tuplo de 3 tuplos, os dois primeiros com 3 células e o último com 2 células (ver Figura 14). Cada célula é representada por “0” se está inativa, por “1” se está ativa e por “-1” se está simultaneamente ativa e não ativa (ou seja, se está no estado incerto).

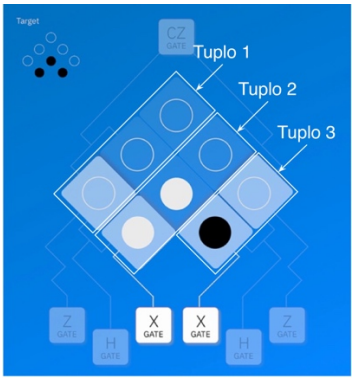

Figura 14: Tuplos e sua correspondência com as células do tabuleiro.

Assim, o tabuleiro da Figura 14 será dado pelo tuplo `((-1, -1, -1), (1, 1, -1), (0, -1))`.

As funções a escrever durante o projeto são as seguintes:

- *eh_tabuleiro*: universal → *booleano*[4] **(1 valor)**

Esta função recebe um argumento de qualquer tipo e devolve *True* se o seu argumento corresponde a um tabuleiro e *False* caso contrário. Nesta parte do projeto, considere que um tabuleiro corresponde a um tuplo contendo três tuplos, os dois primeiros com três elementos e o último com dois elementos e que os elementos destes tuplos são 0, 1, ou -1. A sua função nunca pode dar erro, seja qual for o valor que é fornecido. Apenas responde *True* ou *False*. Não verifique se existe consistência entre os valores contidos nas células do tabuleiro.

```
>>> t = ((-1, -1, -1), (0, 0, -1), (1, -1))
>>> eh_tabuleiro(t)
True
>>> t = ((-1, -1, -1), (0, 0, -1), (1, -1, 0))
>>> eh_tabuleiro(t)
False
>>> eh_tabuleiro(3)
False
```

- *tabuleiro_str*: tabuleiro → *cad. caracteres* **(3 valores)**

Esta função recebe um tabuleiro 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 (note que o tabuleiro é representado com os qubits fazendo um ângulo de 45º com a horizontal). Se o argumento dado for inválido, a função deve gerar um erro com a mensagem `'tabuleiro_str: argumento invalido'` caso um dos argumentos introduzidos não seja válido.

[4] A notação eh_*tabuleiro*: univeral → *booleano* indica que a função com o nome *eh_tabuleiro* tem um argumento do tipo universal e o seu valor é do tipo *booleano*. Esta notação tem o nome de *assinatura da função eh_tabuleiro*.

```
>>> t = ((-1, -1, -1), (1, 1, -1), (0, -1))
>>> tabuleiro_str(t)
'+-------+\n|...x...|\n|..x.x..|\n|.x.1.x.|\n|..1.0..|\n+-------+'
>>> target = ((-1, -1, -1), (0, 0, -1), (0, -1))
>>> print(tabuleiro_str(target))
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..0.0..|
+-------+
>>> print(tabuleiro_str(((-1, -1, -1), (0, 0, -1), (1, -1, 0))))
Traceback (most recent call last): <...>
builtins.ValueError: tabuleiro_str: argumento invalido
```

- *tabuleiros_iguais*: tabuleiro x tabuleiro → *booleano* **(1 valor)**

Esta função recebe dois tabuleiros e devolve *True* se os tabuleiros são iguais e *False* caso contrário. Se um dos argumentos não for um tabuleiro, a função origina um erro. Dois tabuleiros são iguais se e só se o conteúdo de cada célula de um tabuleiro é o mesmo que a correspondente célula no outro tabuleiro.

```
>>> p1 = ((-1, -1, -1), (1, 1, -1), (0, -1))
>>> print(tabuleiro_str(p1))
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..1.0..|
+-------+
>>> tabuleiros_iguais(p1, p1)
True
>>> tabuleiros_iguais(p1, target)
False
>>> p2 = ((-1, -1, -1), (0, 0, -1), (0, -1))
>>> tabuleiros_iguais(p2, target)
True
>>> tabuleiros_iguais((-1, -1, -1), target)
Traceback (most recent call last): <...>
builtins.ValueError: tabuleiros_iguais: um dos argumentos nao e tabuleiro
>>> tabuleiros_iguais([-1, -1, -1], target)
Traceback (most recent call last): <...>
builtins.ValueError: tabuleiros_iguais: um dos argumentos nao e tabuleiro
>>> tabuleiros_iguais(tuple(), target)
Traceback (most recent call last): <...>
builtins.ValueError: tabuleiros_iguais: um dos argumentos nao e tabuleiro
```

3.2 Operadores

• *porta_x*: tabuleiro x {“E”, “D”} → *tabuleiro* **(3 valores)**

Esta função recebe um tabuleiro e um carácter (“E” ou “D”) e devolve um novo tabuleiro resultante de aplicar a porta X ao qubit esquerdo ou direito, conforme o carácter seja “E” ou “D”, respetivamente. A

função deve verificar a validade dos seus argumentos, gerando um erro com a mensagem `'porta_x: um dos argumentos e invalido'` caso um dos argumentos introduzidos não seja válido.

```
>>> p2 = porta_x(p1,"E")
>>> print(tabuleiro_str(p2))
+-------+
|...x...|
|..x.x..|
|.x.0.x.|
|..0.0..|
+-------+
>>> p3 = porta_x(p2,"D")
>>> print(tabuleiro_str(p3))
+-------+
|...x...|
|..x.x..|
|.x.1.x.|
|..0.1..|
+-------+
>>> p = porta_x(p2, "X")
Traceback (most recent call last): <...>
builtins.ValueError: porta_x: um dos argumentos e invalido
```

• *porta_z*: tabuleiro x {"E", "D"} → *tabuleiro* **(3 valores)**

Esta função recebe um tabuleiro e um carácter ("E" ou "D") e devolve um novo tabuleiro resultante de aplicar a porta Z ao qubit da esquerda ou da direita, conforme o carácter seja "E" ou "D", respetivamente. A função deve verificar a validade dos seus argumentos, gerando um erro com a mensagem `'porta_z: um dos argumentos e invalido'` caso um dos argumentos introduzidos não seja válido.

```
>>> p7 = ((1, -1, 1), (-1, -1, -1), (-1, 0))
>>> print(tabuleiro_str(p7))
+-------+
|...1...|
|..x.x..|
|.1.x.0.|
|..x.x..|
+-------+
>>> p8 = porta_z(p7, "E")
>>> print(tabuleiro_str(p8))
+-------+
|...0...|
|..x.x..|
|.0.x.0.|
|..x.x..|
+-------+
>>> p9 = porta_z(p8, "D")
>>> print(tabuleiro_str(p9))
+-------+
|...1...|
|..x.x..|
|.0.x.1.|
|..x.x..|
+-------+
```

```
>>> p = porta_z(p9, "X")
Traceback (most recent call last): <...>
builtins.ValueError: porta_z: um dos argumentos e invalido
```

• *porta_h*: tabuleiro x {"E", "D"} → *tabuleiro* **(3 valores)**

Esta função recebe um tabuleiro e um carácter ("E" ou "D") e devolve um novo tabuleiro resultante de aplicar a porta H ao qubit da esquerda ou da direita, conforme o carácter seja "E" ou "D", respetivamente. A função deve verificar a validade dos seus argumentos, gerando um erro com a mensagem `'porta_h: um dos argumentos e invalido'` caso um dos argumentos introduzidos não seja válido.

```
>>> p4 = ((-1, -1, -1), (0, -1, 0), (-1, 0))
>>> print(tabuleiro_str(p4))
+-------+
|...x...|
|..x.0..|
|.x.x.0.|
|..0.x..|
+-------+
>>> p5 = porta_h(p4, "E")
>>> print(tabuleiro_str(p5))
+-------+
|...0...|
|..x.x..|
|.0.x.0.|
|..x.x..|
+-------+
>>> p6 = porta_h(p5, "D")
>>> print(tabuleiro_str(p6))
+-------+
|...x...|
|..0.x..|
|.0.x.x.|
|..x.0..|
+-------+
>>> p = porta_h(p6, "X")
Traceback (most recent call last):
builtins.ValueError: porta_h: um dos argumentos e invalido
```

4 Sugestões

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. Ao desenvolver cada uma das funções pedidas, comece por perceber se pode usar alguma das anteriores.

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

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.

5 Aspetos a evitar

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

1. Não pense que o projeto se pode fazer nos últimos dias. Se apenas iniciar o seu trabalho neste período irá ver 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);
2. *Não duplique código*. Se duas funções são muito semelhantes é natural que estas possam ser fundidas numa única, eventualmente com mais argumentos;
3. Não se esqueça que as funções excessivamente grandes são penalizadas no que respeita ao estilo de programação;
4. A atitude "vou pôr agora o programa a correr de qualquer maneira e depois preocupo-me com o estilo" é totalmente errada;
5. 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.

6 Condições de Realização e Prazos

A entrega do 1º projeto será efetuada exclusivamente por via eletrónica. Deverá submeter o seu projeto através do sistema *Mooshak*, até às **23:59 do dia 3 de Novembro de 2018**. Depois desta hora, não serão aceites projetos sob pretexto algum.

Deverá submeter um único ficheiro com extensão `.py` contendo todo o código do seu projeto. O ficheiro de código deve conter em comentário, na primeira linha, o número e o nome do aluno.

No seu ficheiro de código não devem ser utilizados caracteres acentuados ou qualquer carácter que não pertença à tabela ASCII. Isto inclui comentários e cadeias de caracteres. Programas que não cumpram este requisito serão penalizados em três valores.

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. O corpo docente da cadeira será o único juiz do que se considera ou não copiar num projeto.

7 Sistema *Mooshak*

A submissão e avaliação da execução do projeto de FP é feita utilizando o sistema Mooshak[5].

7.1 Submissão

A submissão do projeto será feita através do sistema *Mooshak*. Para obter as necessárias credenciais de acesso e poder usar o sistema deverá:

- **Obter uma password** para acesso ao sistema, seguindo as instruções na página: http://acp.tecnico.ulisboa.pt/~fpshak/cgi-bin/getpass. A password ser-lhe-á enviada para o email que tem configurado no *Fenix*. Se a password não lhe chegar de imediato, aguarde.
- Após ter recebido a sua password por email, deve **efetuar o login** no sistema através da página: http://acp.tecnico.ulisboa.pt/~fpshak/. Preencha os campos com a informação fornecida no email.
- Utilize o botão "Browse...", selecione o ficheiro com extensão .py contendo **todo o código** do seu projeto. O seu ficheiro .py deve conter a implementação das funções pedidas no enunciado. De seguida clique no botão "Submit" para **efetuar a submissão**.
  Aguarde (20-30 seg) para que o sistema processe a sua submissão!!!
- Quando a submissão tiver sido processada, poderá visualizar na tabela o resultado correspondente. Receberá no seu email um **relatório de execução** com os detalhes da avaliação automática do seu projeto podendo ver o número de testes passados/falhados.
- Para sair do sistema utilize o botão "Logout".

Submeta o seu projeto atempadamente, dado que as restrições seguintes podem não lhe permitir fazê-lo no último momento:

- Só poderá efetuar uma nova submissão pelo menos 15 minutos depois da submissão anterior.
- Só são permitidas 10 submissões em simultâneo no sistema, pelo que uma submissão poderá ser recusada se este limite for excedido[6].
- Não pode ter submissões duplicadas, ou seja submissão igual à anterior.
- 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. Não serão abertas exceções.

[5] A versão de Python utilizada nos testes automáticos é Python 3.5.3.
[6] Note que o limite de 10 submissões simultâneas no sistema *Mooshak* implica que, caso haja um número elevado de tentativas de submissão sobre o prazo de entrega, alguns grupos poderão não conseguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

7.2 Testes automáticos

A avaliação da execução será feita através do sistema *Mooshak.* O tempo de execução de cada teste está limitado, bem como a memória utilizada. O sistema não deverá ser utilizado para *debug* e como tal, relembra-se que o tempo entre submissões e o número de submissões é limitado.

Cada aluno tem direito a 10 submissões sem penalização no Mooshak. Por cada submissão adicional serão descontados 0,1 valores na componente de avaliação automática.

Os testes considerados para efeitos de avaliação podem incluir ou não os exemplos disponibilizados, além de um conjunto de testes adicionais. O facto de um projeto completar com sucesso os exemplos fornecidos não implica, pois, que esse projeto esteja totalmente correto, pois o conjunto de exemplos fornecido não é exaustivo. É da responsabilidade de cada aluno garantir que o código produzido está correto.

Não será disponibilizado qualquer tipo de informação sobre os casos de teste utilizados pelo sistema de avaliação automática. Os ficheiros de teste usados na avaliação do projeto serão disponibilizados na página da disciplina após a data de entrega.

Pontuação: Existem 206 casos de teste configurados no sistema. 12 testes públicos valendo 0 pontos cada e 194 testes privados. Como a avaliação automática vale 70% (equivalente a 14 valores) da nota, uma submissão obtém a nota máxima de 1400 pontos.

8 Classificação

A nota do projeto será baseada nos seguintes aspetos:

1. Execução correta (70%). Esta parte da avaliação é feita recorrendo ao sistema *Mooshak* que sugere uma nota face aos vários aspetos considerados.
2. Estilo de programação e facilidade de leitura, nomeadamente a abstração procedimental, nomes bem escolhidos, qualidade (e não quantidade) dos comentários e tamanho das funções (30%). Os seus comentários deverão incluir, entre outros, a assinatura de cada função definida.