TÉCNICO LISBOA

Fundamentos da Programação

Ano lectivo 2017-18

Segundo Projeto

Palavra GURU Multi Jogador

O jogo *Palavra Guru* é um quebra-cabeças que requer a formação de palavras a partir de um conjunto de letras. Uma *palavra* é uma cadeia de caracteres composta exclusivamente por letras, e que satisfaz uma dada gramática. O conjunto de letras pode ter repetições, uma vez que as palavras possíveis são as geradas pela permutação sem repetições das diferentes letras. Deste modo as palavras podem ter no máximo tantos caracteres quantas as letras presentes no conjunto. Por exemplo, a partir do conjunto de letras *{A, A, L}* pode gerar-se a palavra *ALA*.

Na versão multi-jogador, o objetivo do jogo é descobrir o conjunto de palavras válidas que se pode gerar a partir desse conjunto de letras, de acordo com uma gramática. O jogo termina quando todas as palavras válidas são descobertas, e o vencedor é o jogador com maior pontuação.

O jogo desenrola-se por jogadas. Em cada jogada, um jogador propõe uma e uma só palavra, e recebe a pontuação correspondente à sua palavra: **se a palavra é válida, o jogador ganha tantos pontos quantas as letras usadas; se a palavra não é válida, o jogador é penalizado na mesma proporção**. Os jogadores jogam por ordem da sua inscrição no jogo.

A validade de cada palavra depende da gramática que define a linguagem.

Por exemplo, a partir do conjunto de letras *{A, E, L}* podem gerar-se todas as palavras do conjunto *{A, E, L, AE, AL, EA, EL, LA, LE, AEL, ALE, EAL, ELA, LAE, LEA}*, mas apenas *A*, *E*, *LA*, *LE*, *AEL*, *ALE*, *ELA* e *LAE* são palavras válidas na gramática apresentada.

No caso de haver repetições de letras no conjunto, por exemplo *{A, M, A, L}*, podem gerar-se todas as palavras do conjunto *{A, L, M, AA, AL, AM, LA, LM, MA, ML, AAL, AAM, ALA, ALM, AMA, AML, LAA, LAM, LMA, MAA, MAL, MLA, AALM, AAML, ALAM, ALMA, AMAL, AMLA, LAAM, LAMA, LMAA, MAAL, MALA, MLAA}*, em que apenas *{A, LA, MA, ALA, AMA, LAM, MAL, ALAM, ALMA, AMAL, AMLA, LAMA, MALA}* são palavras válidas.

1 Trabalho a Realizar

O objetivo do segundo projeto é escrever um programa em Python que permita jogar um jogo de Palavra Guru na versão multi-jogador.

Para tal, deverá definir um conjunto de tipos abstratos de dados que deverão ser utilizados para manipular a informação necessária ao decorrer do jogo, bem como um conjunto de funções adicionais que permitirão jogar o jogo propriamente dito.

Deverá ainda considerar a gramática definida no primeiro projeto, e que se transcreve em baixo.

1.1 Gramática Guru

```
<palavra>::=<monossilabo>|<silaba>*<silaba_final>
<silaba>::=<vogal>|<silaba_2>|<silaba_3>|<silaba_4>|<silaba_5>
<silaba_final>::=<monossilabo_2>|<monossilabo_3>|<silaba_4>|<silaba_5>
<silaba_5>::=<par_consoantes><vogal>NS
<silaba_4>::=<par_vogais>NS|<consoante><vogal>NS|<consoante><vogal>IS
              |<par_consoantes><par_vogais>
              |<consoante><par_vogais><consoante_final>
<silaba_3>::=QUA|QUE|QUI|GUE|GUI|<vogal>NS|<consoante><par_vogais>
              |<consoante><vogal><consoante_final>
              |<par_vogais><consoante_final>
              |<par_consoantes><vogal>
<silaba_2>::=<par_vogais>|<consoante><vogal>|<vogal><consoante_final>
<monossilabo>::=<vogal_palavra>|<monossilabo_2>|<monossilabo_3>
<monossilabo_3>::=<consoante><vogal><consoante_terminal>
              |<consoante><ditongo>
              |<par_vogais><consoante_terminal>
<monossilabo_2>::=AR|IR|EM|UM|<vogal_palavra>S|<ditongo_palavra>
              |<consoante_freq><vogal>
<par_consoantes>::=BR|CR|FR|GR|PR|TR|VR|BL|CL|FL|GL|PL
<consoante>::=B|C|D|F|G|H|J|L|M|N|P|Q|R|S|T|V|X|Z
<consoante_final>::=N|P|<consoante_terminal>
<consoante_terminal>::=L|M|R|S|X|Z
<consoante_freq>::=D|L|M|N|P|R|S|T|V
<par_vogais>::=<ditongo>|IA|IO
<ditongo>::=AE|AU|EI|OE|OI|IU|<ditongo_palavra>
<ditongo_palavra>::=AI|AO|EU|OU
<vogal>::=I|U|<vogal_palavra>
<vogal_palavra>::=<artigo_def>|E
<artigo_def>::=A|O
```

1.2 Tipos Abstratos de Dados (TAD)

TAD *palavra_potencial* (4 val.)

O TAD *palavra_potencial* será usado para representar as palavras propostas pelos jogadores, que são potencialmente válidas de acordo com a gramática e para o conjunto de letras em jogo.

Uma *palavra potencial* é uma cadeia de caracteres que apenas contenha todas ou algumas das letras existentes no conjunto de letras em jogo, independentemente de ser válida ou não de acordo com a gramática.

As operações básicas associadas a este TAD são:

- *cria_palavra_potencial*: *cad. caracteres* × *tuplo de letras* → *palavra_potencial*

  Esta função corresponde ao construtor do tipo *palavra_potencial*. Recebe como argumentos uma cadeia de caracteres e um conjunto de letras (potencialmente com repetições) e devolve uma *palavra_potencial*. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem:

  - `'cria_palavra_potencial:argumentos invalidos.'` caso algum dos seus argumentos não seja válido;
  - `'cria_palavra_potencial:a palavra nao e valida.'` se a cadeia de caracteres contenha mais do que as letras existentes no conjunto de letras em jogo, ou letras diferentes daquelas.

- *palavra_tamanho*: *palavra_potencial* → *inteiro*

  Este selector recebe como argumento um elemento do tipo *palavra_potencial* e devolve o número de letras da palavra.

- *e_palavra_potencial*: *universal* → *lógico*

  Este reconhecedor recebe um único argumento e devolve `True` caso esse argumento seja do tipo *palavra_potencial*, e `False` em caso contrário.

- *palavras_potenciais_iguais*: *palavra_potencial* × *palavra_potencial* → *lógico*

  Este teste recebe como argumentos dois elementos do tipo *palavra_potencial* e devolve `True` caso esses argumentos representem a mesma palavra, e `False` em caso contrário.

- *palavra_potencial_menor*: *palavra_potencial* × *palavra_potencial* → *lógico*

  Este teste recebe como argumentos dois elementos do tipo *palavra_potencial* e devolve `True` caso o primeiro argumento represente uma palavra alfabeticamente anterior à palavra representada pelo segundo argumento, e `False` em caso contrário.

- *palavra_potencial_para_cadeia*: *palavra_potencial* → *cad. caracteres*

Esta função recebe como argumento um elemento do tipo *palavra_potencial* e devolve uma cadeia de caracteres que a represente de acordo com o exemplo em baixo.

Exemplo de interação:

```
>>> conjunto = ("a", "E", "L", "M", "T")
>>> p = cria_palavra_potencial("METa", conjunto)
[...]
builtins.ValueError: cria_palavra_potencial:argumentos invalidos.
>>> conjunto = ("A", "E", "L", "M", "T")
>>> p = cria_palavra_potencial("1META", conjunto)
[...]
builtins.ValueError: cria_palavra_potencial:argumentos invalidos.
>>> p = cria_palavra_potencial("AMETA", conjunto)
[...]
builtins.ValueError: cria_palavra_potencial:a palavra nao e valida.
>>> p = cria_palavra_potencial("META", conjunto)
>>> palavra_tamanho(p)
4
>>> e_palavra_potencial(p)
True
>>> e_palavra_potencial("1META")
False
>>> e_palavra_potencial("mETA")
False
>>> palavras_potenciais_iguais(p, p)
True
>>> p1 = cria_palavra_potencial("META", conjunto)
>>> palavras_potenciais_iguais(p, p1)
True
>>> p2 = cria_palavra_potencial("TELA", conjunto)
>>> palavras_potenciais_iguais(p, p2)
False
>>> palavra_potencial_menor(p1, p2)
True
>>> palavra_potencial_menor(p2, p1)
False
>>> palavra_potencial_para_cadeia(p)
'META'
```

TAD *conjunto_palavras* (5 val.)

O TAD *conjunto_palavras* será utilizado para guardar conjuntos de *palavras_potenciais*.

As operações básicas associadas a este TAD são:

- *cria_conjunto_palavras*: $\rightarrow$ *conjunto_palavras*

  Esta função corresponde ao construtor do tipo *conjunto_palavras*. Não recebe argumentos e devolve um conjunto de palavras vazio.

- *numero_palavras*: *conjunto_palavras* → *inteiro*

  Este seletor recebe como argumento um elemento do tipo *conjunto_palavras* e devolve um inteiro correspondente ao número de palavras guardadas.

- *subconjunto_por_tamanho*: *conjunto_palavras* × *inteiro* → *lista*

  Este seletor recebe como argumentos um elemento do tipo *conjunto_palavras* e um *inteiro* `n`, e devolve uma lista com as palavras_potenciais de tamanho `n` contidas no conjunto de palavras.

- *acrescenta_palavra*: *conjunto_palavras* × *palavra_potencial* →

  Este modificador recebe como argumentos um elemento do tipo *conjunto_palavras* e uma *palavra_potencial*, e tem como efeito juntar a palavra ao conjunto de palavras, caso esta ainda não pertença ao conjunto. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem

  `'acrescenta_palavra:argumentos invalidos.'`

  caso os argumentos não sejam válidos. Se a palavra já existir no conjunto, o conjunto fica inalterado.

- *e_conjunto_palavras*: *universal* → *lógico*

  Este reconhecedor recebe um único argumento e devolve `True` caso esse argumento seja do tipo *conjunto_palavras*, e `False` em caso contrário.

- *conjuntos_palavras_iguais*: *conjunto_palavras* × *conjunto_palavras* → *lógico*

  Este teste recebe como argumentos dois elementos do tipo *conjunto_palavras* e devolve `True` caso os dois argumentos contenham as mesmas palavras, e `False` caso contrário.

- *conjunto_palavras_para_cadeia*: *conjunto_palavras* → *cad. caracteres*

  Esta função recebe como argumento um elemento do tipo *conjunto_palavras* e devolve uma cadeia de caracteres que o represente. As palavras são enumeradas por ordem crescente do seu tamanho, e para cada tamanho são ordenadas alfabeticamente, como se ilustra no exemplo de interação

Exemplo de interação:

```
>>> letras = ("A", "E", "L")
>>> p1 = cria_palavra_potencial("A", letras)
>>> p2 = cria_palavra_potencial("E", letras)
>>> p3 = cria_palavra_potencial("LA", letras)
>>> p4 = cria_palavra_potencial("ELA", letras)
>>> c = cria_conjunto_palavras()
>>> acrescenta_palavra(c, p3)
>>> acrescenta_palavra(c, p4)
>>> acrescenta_palavra(c, p1)
>>> acrescenta_palavra(c, p2)
```

```
>>> conjunto_palavras_para_cadeia(c)
'[1->[A, E];2->[LA];3->[ELA]]'
>>> subconjunto_por_tamanho(c, 1)
[A, E]
```

TAD *Jogador* (5 val.)

O TAD *jogador* será utilizado para representar cada jogador. Este TAD deverá permitir guardar a pontuação obtida por cada jogador assim como as listas de palavras válidas e inválidas propostas por ele.

- *cria_jogador*: *cad. caracteres* → *jogador*

  Esta função corresponde ao construtor do tipo *jogador*. Recebe como argumento uma cadeia de caracteres correspondente ao nome do jogador, e devolve um *jogador*. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem `'cria_jogador:argumento invalido.'` caso o argumento não seja válido.

- *jogador_nome*: *jogador* → *cad. caracteres*

  Este selector recebe como argumento um elemento do tipo *jogador*, e devolve o nome do jogador.

- *jogador_pontuacao*: *jogador* → *inteiro*

  Este selector recebe como argumento um elemento do tipo *jogador*, e devolve a pontuação obtida pelo jogador até ao momento.

- *jogador_palavras_validas*: *jogador* → *conjunto_palavras*

  Este selector recebe como argumento um elemento do tipo *jogador*, e devolve o conjunto de palavras válidas propostas pelo jogador até ao momento.

- *jogador_palavras_invalidas*: *jogador* → *conjunto_palavras*

  Este selector recebe como argumento um elemento do tipo *jogador*, e devolve o conjunto de palavras inválidas propostas pelo jogador até ao momento.

- *adiciona_palavra_valida*: *jogador* × *palavra_potencial* →

  Este modificador recebe como argumentos um elemento do tipo *jogador* e uma *palavra_potencial* `p`, e tem como efeito adicionar a palavra p ao conjunto de palavras válidas propostas pelo jogador, e atualizar a pontuação do jogador convenientemente. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem

  `'adiciona_palavra_valida:argumentos invalidos.'`

  caso os argumentos não sejam válidos.

- *adiciona_palavra_invalida*: *jogador* × *palavra_potencial* →

  Este modificador recebe como argumentos um elemento do tipo *jogador* e uma *palavra_potencial* `p`, e tem como efeito adicionar a palavra p ao conjunto de palavras inválidas propostas pelo jogador, e atualizar a pontuação do jogador convenientemente. A função deve verificar a validade dos seus argumentos, gerando um `ValueError` com a mensagem

  `'adiciona_palavra_invalida:argumentos invalidos.'`

  caso os argumentos não sejam válidos.

- *e_jogador*: *universal* → *lógico*

  Este reconhecedor recebe um único argumento e devolve `True` caso esse argumento seja do tipo *jogador*, e `False` em caso contrário.

- *jogador_para_cadeia*: *jogador* → *cad. caracteres*

  Esta função recebe como argumento um elemento do tipo *jogador* e devolve uma cadeia de caracteres que o represente. Cada jogador é descrito pelo seu nome, seguido pela pontação obtida e pelos conjuntos de palavras válidas e inválidas, de acordo com o exemplo em baixo.

Exemplo de interação:

```
>>>letras = ("A", "E", "L")
>>> p1 = cria_palavra_potencial("ELA", letras)
>>> p2 = cria_palavra_potencial("AL", letras)
>>> jog = cria_jogador("joao")
>>> adiciona_palavra_valida(jog, p1)
>>> adiciona_palavra_invalida(jog, p2)
>>> jogador_para_cadeia(jog)
'JOGADOR joao PONTOS=1 VALIDAS=[3->[ELA]] INVALIDAS=[2->[AL]]'
```

1.3 Funções Adicionais

- *gera_todas_palavras_validas*: *tuplo de letras* → *conjunto_palavras* **(3 val.)**

  Esta função recebe como argumento um tuplo de letras (possivelmente com repetições) para formar palavras e devolve um conjunto de palavras, contendo todas as palavras geradas a partir das letras dadas ou de subconjuntos das mesmas, que são válidas de acordo com a gramática.

  Por exemplo, para uma qualquer lista de três letras existirão no máximo 15 palavras válidas (3 palavras de uma letra, 6 palavras de duas letras e 6 palavras de três letras), mas apenas algumas delas serão válidas de acordo com a gramática.

  Exemplo:

```
>>> letras = ("A", "E", "L")
>>> conjunto_palavras_para_cadeia(gera_todas_palavras(letras))
[1->[A, E];2->[LA, LE];3->[AEL, ALE, ELA, LAE]]
```

- *guru_mj*: *tuplo de letras* → **(3 val.)**

  Esta função corresponde à função principal do jogo e permite jogar um jogo completo de $Palavra\,Guru\,MultiJogador$. Recebe como argumento um *tuplo de letras* correspondente ao conjunto de letras a usar na formação das palavras, e apresenta o jogador vencedor, ou a indicação de empate, caso exista mais do que um jogador com a melhor pontuação.

  A função deve começar por pedir aos utilizadores os nomes dos vários jogadores. Para tal deve apresentar a mensagem

  `Introduza o nome dos jogadores (-1 para terminar)...`

  e esperar que o nome de cada jogador seja introduzido. Cada nome deve ser introduzido a seguir à mensagem

  `JOGADOR` *`n`* `->`

  em que *n* representa o número do jogador. Depois de recolher os nomes dos jogadores, a função deve proceder à sua criação e prosseguir com o pedido de jogadas até que o jogo termine.

  O pedido de cada jogada é feito depois de apresentar os números da jogada e de palavras a descobrir, através da mensagem

  `JOGADA` *`k`* `- Falta descobrir` *`n`* `palavras`

  em que *k* representa o número da jogada e *n* o número de palavras ainda por descobrir. A jogada é introduzida depois de apresentado o nome do jogador que deve propor a próxima palavra, através da mensagem

  `JOGADOR` *`nome`* `->`

  em que *nome* corresponde ao nome do jogador que deve propor a palavra. Depois de introduzida a jogada, os jogadores devem ser informados sobre a validade da palavra introduzida através da mensagem

  `proposta` `- palavra VALIDA`

  caso a palavra proposta (*proposta*) seja válida, e

  `proposta` `- palavra INVALIDA`

  caso seja inválida.

  O jogo termina quando não há mais palavras para descobrir para o conjunto de letras dado, e deve ser apresentado o vencedor do jogo através da mensagem

`FIM DE JOGO! O jogo terminou com a vitoria do jogador` *nome* `com` *n* `pontos.`

em caso de vitória do jogador *nome* tendo obtido *n* pontos, e a mensagem

`FIM DE JOGO! O jogo terminou em empate.`

Depois desta mensagem deve seguir-se a apresentação dos resultados dos diferentes jogadores, pela mesma ordem de introdução, através da representação externa do TAD jogador.

Exemplo 1:

```
>>> guru_mj(("A", "E", "L"))
Descubra todas as palavras geradas a partir das letras:
('A', 'E', 'L')
Introduza o nome dos jogadores (-1 para terminar)...
JOGADOR 1 -> joao
JOGADOR 2 -> maria
JOGADOR 3 -> -1
JOGADA  1 - Falta descobrir  8  palavras
JOGADOR joao -> ELA
ELA  - palavra VALIDA
JOGADA  2 - Falta descobrir  7  palavras
JOGADOR maria -> AEL
AEL  - palavra VALIDA
JOGADA  3 - Falta descobrir  6  palavras
JOGADOR joao -> ALE
ALE  - palavra VALIDA
JOGADA  4 - Falta descobrir  5  palavras
JOGADOR maria -> LAE
LAE  - palavra VALIDA
JOGADA  5 - Falta descobrir  4  palavras
JOGADOR joao -> LA
LA  - palavra VALIDA
JOGADA  6 - Falta descobrir  3  palavras
JOGADOR maria -> LE
LE  - palavra VALIDA
JOGADA  7 - Falta descobrir  2  palavras
JOGADOR joao -> A
A  - palavra VALIDA
JOGADA  8 - Falta descobrir  1  palavras
JOGADOR maria -> E
E  - palavra VALIDA
FIM DE JOGO! O jogo terminou em empate.
JOGADOR joao PONTOS=9 VALIDAS=[1->[A];2->[LA];3->[ALE, ELA]] INVALIDA
S=[]
JOGADOR maria PONTOS=9 VALIDAS=[1->[E];2->[LE];3->[AEL, LAE]] INVALID
AS=[]
```

Exemplo 2:

```
>>> guru_mj(("A", "E", "L"))
Descubra todas as palavras geradas a partir das letras:
('A', 'E', 'L')
Introduza o nome dos jogadores (-1 para terminar)...
JOGADOR 1 -> diogo
JOGADOR 2 -> tiago
JOGADOR 3 -> -1
JOGADA  1 - Falta descobrir  8  palavras
JOGADOR diogo -> AEL
AEL  - palavra VALIDA
JOGADA  2 - Falta descobrir  7  palavras
JOGADOR tiago -> LEA
LEA  - palavra INVALIDA
JOGADA  3 - Falta descobrir  7  palavras
JOGADOR diogo -> ALE
ALE  - palavra VALIDA
JOGADA  4 - Falta descobrir  6  palavras
JOGADOR tiago -> ELA
ELA  - palavra VALIDA
JOGADA  5 - Falta descobrir  5  palavras
JOGADOR diogo -> LAE
LAE  - palavra VALIDA
JOGADA  6 - Falta descobrir  4  palavras
JOGADOR tiago -> LA
LA  - palavra VALIDA
JOGADA  7 - Falta descobrir  3  palavras
JOGADOR diogo -> LE
LE  - palavra VALIDA
JOGADA  8 - Falta descobrir  2  palavras
JOGADOR tiago -> EL
EL  - palavra INVALIDA
JOGADA  9 - Falta descobrir  2  palavras
JOGADOR diogo -> A
A  - palavra VALIDA
JOGADA  10 - Falta descobrir  1  palavras
JOGADOR tiago -> E
E  - palavra VALIDA
FIM DE JOGO! O jogo terminou com a vitoria do jogador diogo com 12 po
ntos.
JOGADOR diogo PONTOS=12 VALIDAS=[1->[A];2->[LE];3->[AEL, ALE, LAE]] I
NVALIDAS=[]
JOGADOR tiago PONTOS=1 VALIDAS=[1->[E];2->[LA];3->[ELA]] INVALIDAS=[2
->[EL];3->[LEA]]
```

1.4 Sugestões

1. Leia o enunciado completo, procurando perceber o objetivo das várias funções pedidas. Em caso de dúvida de interpretação, utilize o horário de dúvidas para esclarecer as suas questões.

2. No processo de desenvolvimento do projeto, comece por implementar os vários TADs e as várias funções pela ordem apresentada no enunciado, seguindo as metodologias estudadas na disciplina. Ao desenvolver cada um dos TADs e funções pedidos, comece por perceber se pode usar algum(a) do(a)s anteriores.
3. Para verificar a funcionalidade das suas funções, utilize os exemplos fornecidos como casos de teste.
4. Tenha o cuidado de reproduzir fielmente as mensagens de erro e restantes *outputs*, conforme ilustrado nos vários exemplos.

2 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.

3 Classificação

A avaliação da execução será feita através do sistema *Mooshak*, onde, tal como no primeiro projeto, existem vários testes configurados no sistema. 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, 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.[1] Nesse caso tente mais tarde.

[1]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 con-

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.

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, a abstração de dados, 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 assim como uma descrição da representação interna adotada em cada um dos TADs definidos.

4 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 6 de Dezembro de 2017**. 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.

Duas semanas antes do prazo, serão publicadas na página da cadeira as instruções necessárias para a submissão do código no *Mooshak*. Apenas a partir dessa altura será possível a submissão por via eletrónica. Nessa altura serão também fornecidas a cada um as necessárias credenciais de acesso. Até ao prazo de entrega poderá efetuar o número de entregas que desejar, sendo utilizada para efeitos de avaliação a última entrega efetuada. Deverá portanto verificar cuidadosamente que a última entrega realizada corresponde à versão do projeto que pretende que seja avaliada. Não serão abertas exceções.

---

seguir submeter nessa altura e verem-se, por isso, impossibilitados de submeter o código dentro do prazo.

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.