Computação 1, 2020.1
Trabalho Final

Para entrega até 3/3 às 10:00

Submeta suas soluções colocando os arquivos correspondentes na sua pasta do Google Drive

Lembre-se de escolher bons nomes para suas funções e variáveis, de documentar seu código com *docstrings* (documentação de função) e comentários onde for apropriado, e de fazer testes para suas funções (na documentação usando o módulo `doctest` ou em funções dedicadas).

O jogo Dots and Boxes

Neste trabalho, vamos implementar o jogo *Dots and Boxes*, criado em 1895 por Édouard Lucas.[1] [2]

O jogo é para dois jogadores e ocorre em uma malha quadrada de N^2 pontos. A cada rodada, o jogador da vez faz um traço ligando pontos adjacentes (horizontalmente ou verticalmente), desde que estes pontos não estivessem ligados por um traço anteriormente. Se este traço completar algum quadrado, dizemos que os quadrados completados (que podem ser um ou dois!) foram *capturados* pelo jogador da vez, que volta a jogar na próxima rodada. Se nenhum quadrado for capturado em uma rodada, a vez passa para o outro jogador. Vence o jogador que capturar mais do que a metade dos quadrados disponíveis na malha.

Para facilitar, vamos rotular as linhas de pontos com os números de N a 1 e as colunas de pontos com as N primeiras letras do alfabeto em ordem crescente (portanto, usaremos uma notação similar à usada no xadrez).

Um exemplo de jogo em uma malha de tamanho 3×3 entre jogadores denotados por $\circ$ e $\bullet$ pode ser visto na página 7. Abaixo de cada tabuleiro, o jogador com a *próxima* vez é indicado.

A representação do jogo em Python

Nossos jogos serão jogados apenas em malhas de tamanho $N \times N$ com $2 \leq N \leq 9$.

Cada *ponto* da malha será representado por uma `string` de comprimento 2, com primeiro caracter denotando o número da linha do ponto e segundo denotando a coluna do ponto. Portanto o ponto no canto inferior esquerdo da malha é `1a`, acima deste está o ponto `2a`, ao lado deste último está o ponto `2b`, etc.

[1] `https://pt.wikipedia.org/wiki/Timbiriche_(jogo)`
[2] `https://en.wikipedia.org/wiki/Dots_and_Boxes`

Cada *quadrado* será denotado pela string de comprimento 2 que denota seu vértice inferior direito.

Cada *jogada* será denotada por uma string de comprimento 3, sendo:

- 2 caracteres com conteúdo numérico denotando linhas, i e $i+1$, e 1 com letra denotando coluna, x, caso a jogada seja o traço vertical ligando o ponto na linha i, coluna x, ao ponto na linha $i+1$, coluna x;
- 1 caracter com conteúdo numérico denotando linha, i, e 2 com letras denotando colunas, x e y, caso a jogada seja o traço horizontal ligando o ponto na linha i, coluna x, ao ponto na linha i, coluna y. Note que, neste caso, necessariamente x e y devem ser letras consecutivas do alfabeto.

Para garantir que cada jogada seja denotada de uma única forma, vamos estipular que os símbolos nas strings descritas acima devem estar ordenados (números ordenados entre si, letras ordenadas entre si, e números antes de letras). Assim '1ab' é uma string válida para denotar uma jogada, mas '3c2' não é (a forma correta desta jogada seria '23c'). Note que esta é exatamente a ordem com que o Python compara strings formadas de números e letras minúsculas.

Com esta notação, as jogadas do jogo apresentado na página 7 são, em ordem: '1ab', '23a', '12a', '12c', '1bc', '12b', '2ab' (captura '1b'), '2bc' (captura '1c'), '3ab', '23c', '3bc', '23b' (captura '2b' e '2c').

O *estado* do jogo será representado por um dict (dicionário) de formato

```
jogo = {'tamanho': N,
        'jogadas': <set_jogadas>,
        nome_um: <set_quadrados_um>,
        nome_dois: <set_quadrados_dois}
```

onde:

- N é um int (o tamanho do lado do tabuleiro);
- <set_jogadas> é um set (conjunto) contendo as jogadas que já foram feitas até aquele momento no jogo
- nome_um é a string '\N{White Circle}';
- <set_quadrados_um> é um set contendo os quadrados capturados pelo jogador 1 até aquele momento no jogo;
- nome_dois é a string '\N{Black Circle}'; e
- <set_quadrados_dois> é um set contendo os quadrados capturados pelo jogador 2 até aquele momento no jogo.

Assim, ao final da oitava imagem da página 7 (i.e., última imagem da segunda fileira), o dicionário do jogo era:

```
{'tamanho': 3,
 'jogadas': {'1ab', '23a', '12a', '12c', '1bc', '12b', '2ab'},
 nome_um: {'1b'},
 nome_dois: set()}.
```

O trabalho

Baixe o arquivo

`https://www.hugonobrega.com/seu_nome_aqui.py`

e o renomeie com o seu nome (mantendo a terminação `.py`). Este arquivo contém as seguintes funções prontas para uso:

- `mostra`, que recebe como entrada um `jogo` e o apresenta na tela, como visto na página 7.
- `outro`, que recebe como entrada uma `string` com o nome de um dos jogadores (i.e., `nome_um` ou `nome_dois`) e retorna o nome do outro jogador.
- `jogar`, que recebe quatro entradas:
  - `primeiro`, uma função que faz as jogadas para o primeiro jogador no jogo (mais detalhes abaixo);
  - `segundo`, uma função que faz as jogadas para o segundo jogador no jogo;
  - `tamanho`, um `int`: o tamanho do lado da malha onde o jogo ocorrerá; e
  - `interativo`, um `bool`: caso `True`, a cada rodada a situação do jogo é mostrada na tela, e ao final o resultado é impresso na tela (sem retorno); caso `False`, o jogo corre sem nada ser impresso na tela, e o resultado é retornado (o nome do vencedor, caso haja, ou a `string` `'empate'`).

  E função `jogar` faz, então, a execução do jogo em si, recebendo as jogadas dos jogadores, validando-as, registrando-as, e controlando se o jogo deve continuar ou já acabou (mais detalhes abaixo).

- `torneio`, que recebe quatro entradas:
  - `jogador_um`, uma função de jogador (como acima);
  - `jogador_dois`, uma função de jogador (como acima);
  - `número_partidas`, um `int`: quantas partidas haverá no torneio, com cada jogador jogando como primeiro e como segundo;
  - `tamanho`, um `int` ou `None`: o tamanho da malha usada nos jogos do torneio, caso seja um `int`; caso seja `None`, tamanhos aleatórios serão usados para cada partida no torneio.

  A função `torneio` faz, então, `num_partidas` jogos com `jogador_um` começando e `num_partidas` com `jogador_dois` começando, e retorna um dicionário indicando as porcentagens de resultados obtidos (vitórias de cada jogador, ou empates)

Para funcionar, a função `jogar` precisa que diversas funções auxiliares sejam implementadas (as funções cujos corpos são apenas a palavra `pass`, no arquivo `seu_nome_aqui.py`), e esta é a primeira parte do seu trabalho (detalhes nas questões abaixo).

Em cada caso, lembre-se de fazer documentação da sua função, de adicionar comentários onde considerar relevante e de fazer funções de teste, quando apropriado.

É importante que você **não** altere os nomes das funções pedidas.

Questão 1. Implemente a função `cria_jogo`, que recebe um `int` e retorna um dicionário de jogo com malha do tamanho indicado e sem nenhuma jogada feita nem nenhum quadrado capturado (um "tabuleiro" pronto para o início de um jogo, em outras palavras).

Questão 2. Implemente a função `jogadas_válidas`, que recebe um dicionário de jogo e retorna o `set` contendo exatamente as jogadas válidas naquele jogo, i.e., as jogadas que podem efetivamente ser feitas naquele jogo.

Questão 3. Implemente a função `arestas_do_quadrado`, que recebe uma `string` representando um quadrado e retorna uma sequência (tupla, lista ou conjunto) contendo exatamente as `strings` que representam as 4 arestas que formam aquele quadrado.

Questão 4. Implemente a função `quadrados_contendo`, que recebe como entrada uma `string` representando uma aresta e um `int` que determina o tamanho do lado da malha, e retorna uma sequência (tupla, lista ou conjunto) contendo exatamente as `strings` representando os quadrados da malha que contêm aquela aresta.

Questão 5. Implemente a função `quadrados_capturados`, que recebe uma `string` representando uma jogada e um dicionário de jogo, e retorna uma sequência (tupla, lista ou conjunto) contendo exatamente as `strings` representando os quadrados daquele jogo que foram capturados por aquela jogada, **assumindo** que a jogada tenha sido a mais recente a ser realizada naquele jogo.

Questão 6. Implemente a função `jogo_acabou`, que recebe um dicionário de jogo e determina se o jogo acabou; o retorno deve ser `False`, caso o jogo não tenha acabado; a `string` `'empate'`, caso o jogo tenha acabado em empate; ou a `string` contendo o nome do vencedor, caso haja algum.

Criando jogadores

Cada jogador será uma função que recebe duas entradas: o dicionário de jogo e uma `string` que indica se o jogador é o primeiro ou o segundo no jogo (i.e., a `string` é `nome_um` ou `nome_dois`). A função retorna uma `string` indicando a jogada a ser realizada (idealmente, a jogada realizada deve ser válida, mas essa verificação será feita pela função `jogar`).

Ao criar cada jogador, você pode testá-lo usando as funções `jogar` e `torneio` (esta última sendo mais útil para jogadores que não dependam de interação com o usuário).

Questão 7. Implemente a função `jogador_humano`, que recebe entradas como explicado acima e retorna a jogada do ser humano sentado ao computador (o "usuário"). Para facilitar a vida do usuário, sua função deve aceitar jogadas mesmo que sejam digitadas fora da ordem que estabelecemos, i.e., caso o usuário digite `ab1` ou `b1a`, sua função deve interpretar como a jogada `1ab`.

Questão 8. Implemente a função `jogador_aleatório`, que escolhe uma jogada (válida) *aleatoriamente*. *Dica*: a função `sample` do módulo `random` recebe dois argumentos, o primeiro sendo uma sequência (tupla, lista ou conjunto) e o segundo um `int`, e retorna uma lista com o tamanho indicado de elementos da sequência dada , escolhidos aleatoriamente. Caso você receba um aviso `DeprecationWarning`, você pode ignorá-lo com segurança.

Questão 9. Implemente um jogador (com o nome que você quiser) que implementa a chamada "estratégia gulosa": caso haja quadrados a serem capturados no jogo, o jogador faz a jogada que captura o máximo possível de quadrados. Caso não seja possível capturar quadrados, não especificamos o que o jogador deve fazer (você pode decidir como preferir).

Questão 10. Implemente um jogador que faça jogadas seguindo alguma outra estratégia que você considerar interessante. Como guia, tente fazer seu jogador ter bom desempenho em torneios contra o `jogador_aleatório` e ser difícil de derrotar na função `jogar` contra o `jogador_humano` (jogando primeiro ou segundo).

Atenção! Seu jogador não deve demorar mais do que alguns poucos segundos para fazer cada jogada.

No dia 3/3, no horário normal de aula, faremos um torneio ao vivo "em sala" entre os jogadores criados pelos alunos como resposta à Questão 10! A participação neste torneio é completamente livre e não conta para a avaliação final de nenhuma forma.

Desafio—opcional

Uma ideia para fazer um jogador *realmente* bom é analisar todos os desdobramentos possíveis de cada jogada que ele possa fazer.

Vamos chamar uma jogada em um jogo de *vencedora* para um jogador se:

- com aquela jogada, aquele jogo acaba e é de fato vencido por aquele jogador; ou
- com aquela jogada naquele jogo, a próxima vez é do mesmo jogador, e existe alguma nova jogada na próxima rodada que é vencedora; ou
- com aquela jogada naquele jogo, a próxima vez é do outro jogador, e todas as jogadas possíveis na próxima rodada são *perdedoras* para o outro;

onde chamamos uma jogada em um jogo de *perdedora* para um jogador se:

- com aquela jogada, aquele jogo acaba e é de fato vencido pelo outro jogador; ou
- com aquela jogada naquele jogo, a próxima vez é do mesmo jogador, mas todas as novas jogadas possíveis na próxima rodada são perdedoras para este jogador; ou
- com aquela jogada naquele jogo, a próxima vez é do outro jogador, e existe alguma jogada possível para o outro jogador na próxima rodada que é vencedora para ele.

Note que as noções de jogada vencedora e perdedora são definidas por recursão *mútua*!

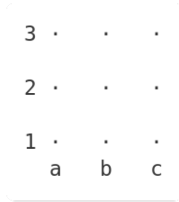
Questão 11. Implemente as funções `jogada_vencedora` e `jogada_perdedora`, que recebem três entradas:

- uma `string` contendo o nome do jogador;
- uma `string` denotando uma jogada;
- um dicionário de jogo,

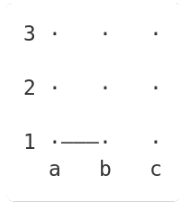
e retornam `True` caso a jogada seja vencedora/perdedora (de acordo com o caso) para aquele jogador naquele jogo, `False` contrário.

Questão 12. Crie um jogador "perfeito" que faça uma jogada vencedora, se possível, senão evite fazer uma jogada perdedora, se possível. Nos demais casos, não especificamos o que deve ser feito (você pode escolher como preferir).

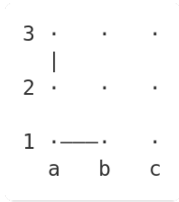
Questão 13 (Desafio dentro do Desafio!)**.** Provavelmente o seu jogador da Questão 12 demora *bastante* para fazer suas jogadas. Tente melhorar isso, implementando a ideia de *memoização* (i.e., uso de *cache*) de alguma forma.



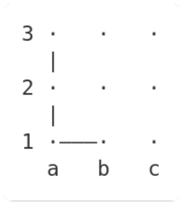
○



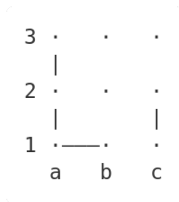
●



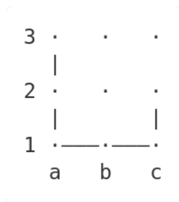
○



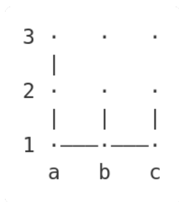
●



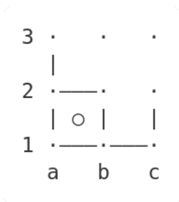
○



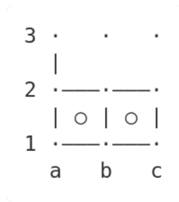
●



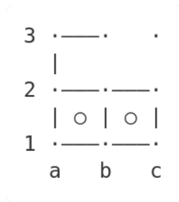
○



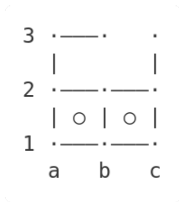
○



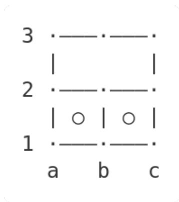
○



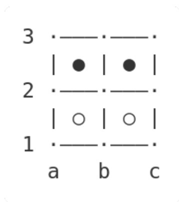
●



○



●



Empate!