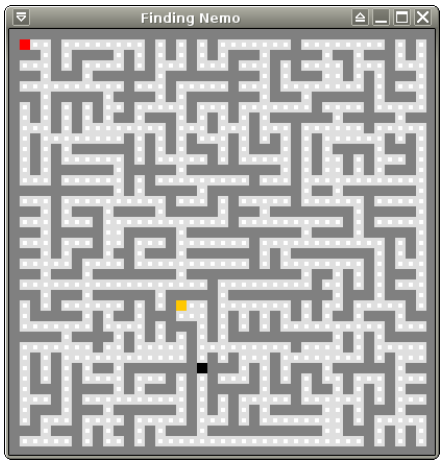
1. Introdução

O mundo aquático está em pânico hoje: o pequeno Nemo foi sequestrado! Marlin, seu pai, deve partir a sua procura. Mas atenção! O oceano é um verdadeiro labirinto, povoado de tubarões famintos! Você conseguirá ajudar Marlin a encontrar Nemo o mais rapidamente possível, evitando os tubarões tanto quanto possível?

O oceano se apresenta sob a forma de um labirinto quadrado, como na imagem abaixo. Marlin está inicialmente na célula vermelha, Nemo na célula laranja, e os tubarões nas células pretas. As células de cor cinza escuro são paredes intransponíveis.



Na sua implementação, o oceano é modelado por uma matriz completa para cada célula do labirinto. As relações de adjacência são implícitas. Cada célula tem por convenção 4 vizinhos: um à esquerda (oeste), um embaixo (sul), um à direita (leste) e um acima (norte). A classe `Cell.java` (disponível no SIGAA) implementa este modelo. Note que esta classe fornece os métodos `west,south,east,north` que permitem acessar facilmente às células adjacentes a uma célula dada. Ela também fornece um método `neighbors` que retorna a lista de células adjacentes na mesma ordem (oeste, sul, leste e norte). Enfim, a construção de uma nova

célula se faz por meio do construtor `Cell(int x, int y)` que recebe as coordenadas `x` e `y` da célula.

A estrutura e a exibição do oceano estão codificados na classe `Ocean`, fornecida no arquivo `Ocean.java`. Você pode executar esta classe se quiser: ela exibirá simplesmente a imagem acima em uma janela. Esta classe contem as constantes `final Cell nemo, marlin` que codificam as posições de Nemo e de Marlin, respectivamente no começo do programa. A classe `Ocean` fornece além disso os métodos a seguir:

- `Ocean(int n)` : um construtor. O índice `n` designa aqui o índice do labirinto escolhido a partir da base de dados de labirintos disponíveis (veja a classe `Data` no fim do arquivo `Ocean.java`
- `boolean isWall(Cell c)` : retorna `true` se uma parede se encontra na célula `c`
- `boolean isNemo(Cell c)` : retorna `true` se Nemo se encontra na célula `c`
- `boolean isMarlin(Cell c)` : retorna `true` se Marlin se encontra na célula `c`
- `boolean isShark(Cell c)` : retorna `true` se um tubarão se encontra na célula `c`
- `void setMark(Cell c)` : marca a célula `c` atribuindo-lhe valor 0
- `void setMark(Cell c, int val)`: marca a célula `c` atribuindo-lhe valor `val`
- `void unMark(Cell c)`: retira a marca da célula `c` atribuindo-lhe valor -1
- `boolean isMarked(Cell c)`: retorna `true` se a célula `c` está marcada, i.e., se o seu valor é ≥ 0
- `int getMark(Cell c)`: retorna a marca da célula `c`

As marcas das células são geradas da seguinte maneira. Inicialmente, elas valem todas -1, e dizemos neste caso que as células estão desmarcadas. Durante a execução do programa certas células serão marcadas, o que significa que suas marcas terão valores ≥ 0. Lembrem-se da função α dos slides dados em sala de aula.

A fim de ajudar Marlin a encontrar Nemo, nós vamos implementar nesta e na próxima prática vários algoritmos para percurso de grafos. Mais uma vez, os nós do grafo são as células do labirinto, e as arestas são representadas implicitamente pelas relações de adjacência oeste, sul, leste e norte. Todo o código desta prática será implementado na classe `Traversal` cujo esqueleto encontra-se no SIGAA.

2. Busca em Largura

Boa notícia! Graça à intervenção de Doris, Marlin vai receber a ajuda de um bando de peixes prateados. Ele vai portanto poder lançar a busca em todas as direções de uma vez, e a primeira dentre elas que encontrar Nemo fará um sinal imediato. Este processo será modelado por um percurso em largura.

2.1. Percurso simples

Complete o método `q21` da classe `Traversal`. Este método implementa um percurso em largura e para assim que Nemo é encontrado. Por enquanto, você não vai precisar reconstituir o caminho de Marlin até Nemo. Para a implementação da fila, utilize a classe `LinkedList` do Java (`http://www.tutorialspoint.com/java/util/java_util_linkedlist.htm`).

A fim de melhor visualizar o caminho percorrido, faça uma chamada ao método `slow` da classe `Traversal` em cada iteração da sua busca em largura.

2.2. Distância ao ponto de partida

Complete o método `q22` da classe `Traversal`. Este método implementa o mesmo percurso em largura do item anterior, mas aumenta o valor das marcas à medida que o percurso é realizado. Mais precisamente, a marca de uma célula dada deve corresponder a menor distância da célula até a célula inicial de Marlin no labirinto.

2.3. Caminho mais curto

Complete o método `q23` da classe `Traversal`. Este método implementa o mesmo percurso em largura dos itens anteriores, mas atribui uma marca a cada célula em função da célula que a precede no percurso. Mais precisamente, se alcançamos uma célula pela esquerda, atribuimos o valor 1. Por baixo, o valor 2. Pela direita, o valor 3 e pelo alto, o valor 4. A fim de facilitar a leitura do código, estes valores estão armazenados respectivamente nas constantes WEST, SOUTH, EAST e NORTH da classe `Traversal`.

Complete agora o método `backTrack`, que parte da célula de Nemo e segue as marcas no sentido inverso até encontrar a célula inicial de Marlin. Cada célula atravessada durante o backtracking será colorida de azul, o que significa que sua marca mudará para o valor armazenado no campo `pathColor` da classe `Traversal`. A condição de parada do método `backTrack` deve utilizar o método `isMarlin` da classe `Ocean`.

Na próxima prática nos preocuparemos com os tubarões...

*Este trabalho prático é de autoria de Jean-Christophe Filliâtre (Poly, France)