6ªaula prática

Estruturas de dados lineares: pilhas, filas.
Documentação usando Doxygen

- Faça download do ficheiro ***aed2223_p06.zip*** da página e descomprima-o (contém a pasta ***lib*** ; a pasta ***docs*** com o ficheiro ***Doxyfile*** ; a pasta ***Tests*** com os ficheiros ***stackExt.h, funStackQueueProblem.h***, ***funStackQueueProblem.cpp***, ***cell.h, cell.cpp*** e ***tests.cpp*** ; e os ficheiros ***CMakeLists*** e ***main.cpp***)
- No CLion, abra um ***projeto***, selecionando a pasta que contém os ficheiros do ponto anterior.

1. A classe **StackExt** implementa uma estrutura do tipo pilha (stack) com um novo método: findMin.

```
class StackExt {
   //….
public:
   bool empty() const;           // verifica se a pilha está vazia
   T& top();                     // retorna o elemento no topo da pilha
   void pop();                   // remove elemento
   void push(const T& val);      // insere elemento
   T& findMin();                 // retorna valor do menor elemento
};
```

Além dos métodos já conhecidos da classe pilha, a classe **StackExt** inclui o método ***findMin.*** Este método retorna o valor do menor elemento da pilha.

- Deve implementar todos os métodos da classe **StackExt** em:

  Complexidade temporal esperada: $O(1)$, para todos os métodos

  Sugestão: Use a classe **stack** da biblioteca STL. Use duas stacks: uma para guardar todos os valores, e outra para guardar os valores mínimos à medida que estes vão surgindo.

- Efetue a documentação do código implementado, usando Doxygen** (ver nota no final da ficha)

2. Implemente a função:

vector<string> FunStackQueueProblem::binaryNumbers(int n)

Dado um número *n*, a tarefa é gerar todos os números binários com valores decimais de 1 a *n*.
Sugestão: Use uma fila para auxiliar na geração/construção dos números binários.

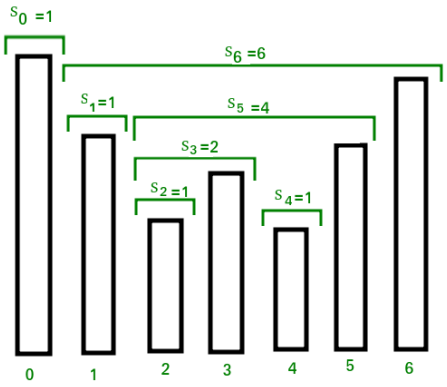
Complexidade temporal esperada: $O(n)$

Exemplo de execução:
input*:* *n=5*
output*:* *result* = {“1”, “10”, “11”, “100”, “101”}

3. Implemente a função:

vector<int> FunStackQueueProblem::calculateSpan(vector<int> prices)

O *span* do preço de uma ação pode ser definido como o número máximo de dias consecutivos antes do dia atual em que o preço da ação era igual ou menor que o preço atual.
Dado um vetor de preços de uma determinada ação num conjunto de dias consecutivos, a função calcula o *span* do preço dessa ação nesses dias, retornando essa informação num vetor.



Sugestão: Note que $span_i$ pode ser facilmente calculado se conhecermos o dia mais próximo anterior a i, tal que o preço daquele dia seja maior do que o preço do dia i. Use uma pilha neste passo.

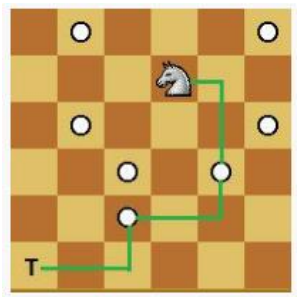
Complexidade temporal esperada: $O(n)$

Exemplo de execução:
input: *prices* = {100, 80, 60, 70, 60, 75, 85}
output: *result* = {1, 1, 1, 2, 1, 4, 6}

4. Implemente a função:

int FunStackQueueProblem::knightJumps(int initialPosx, int initialPosy, int targetPosx, int targetPosy, int n)

Dado um tabuleiro de xadrez quadrado de tamanho $n \times n$, a posição do cavalo (*initialPosx*, *initialPosy*) e a posição destino (*targetPosx*, *targetPosy*) , a função deve determinar o número mínimo de saltos que o cavalo dá para alcançar a posição destino.



na figura:
- os círculos ilustram a posição seguinte que o cavalo pode atingir com um salto.
- o cavalo atinge a posição destino com 3 saltos: (4,5) -> (5,3) -> (3,2) -> (1,1)

Sugestão: Use uma fila para ir guardando as posições possíveis que o cavalo pode alcançar. Construa a fila de modo a que na frente da fila estão as posições possíveis de alcançar em menor número de saltos.

Complexidade temporal esperada: $O(n^2)$

Exemplo de execução:

input: *initialPosx=4, initialPosy=5, targetPosx=1, targetPosy=1, n=6*

output: *result* = 3

****Notas sobre Doxygen**

Doxygen gera documentação a partir de código fonte. Passos a seguir (consultar também página moodle e Doxygen):

1. Instalar Doxygen.
2. Incluir no projeto a referência ao Doxygen, colocando essa informação em *"CMakeLists.txt"*.
   O texto seguinte serve como exemplo. Note a indicação (deve alterar para o que pretender) de:
   - ficheiro de configuração *Doxyfile* em "CMAKE_CURRENT_SOURCE_DIR}/docs/Doxyfile"

```
# Doxygen Build
find_package(Doxygen)
if(DOXYGEN_FOUND)
    set(BUILD_DOC_DIR "${CMAKE_SOURCE_DIR}/docs/output")
    if(NOT EXISTS "${BUILD_DOC_DIR}")
        file(MAKE_DIRECTORY "${BUILD_DOC_DIR}")
    endif()

    set(DOXYGEN_IN "${CMAKE_CURRENT_SOURCE_DIR}/docs/Doxyfile")
    set(DOXYGEN_OUT "${CMAKE_CURRENT_BINARY_DIR}/Doxyfile")
    configure_file("${DOXYGEN_IN}" "${DOXYGEN_OUT}" @ONLY)

    message("Doxygen build started")
    add_custom_target(Doxygen ALL
            COMMAND "${DOXYGEN_EXECUTABLE}" "${DOXYGEN_OUT}"
            WORKING_DIRECTORY "${CMAKE_CURRENT_BINARY_DIR}"
            COMMENT "Generating API documentation with Doxygen"
            VERBATIM)
else(DOXYGEN_FOUND)
    message("Doxygen needs to be installed to generate the documentation.")
endif(DOXYGEN_FOUND)
```

3. O ficheiro de configuração *Doxyfile* deve ser colocado na pasta indicada em "*CMakeLists.txt*". Um exemplo deste ficheiro está na página moodle da UC e já foi incluído no conjunto de ficheiros fornecidos para esta aula (***aed2223_p06.zip)***
   Note a indicação (deve alterar para o que pretender) do local onde se encontra o código fonte:
   - o ficheiro *Doxyfile* indica em "INPUT =" o local onde se encontra o código fonte

4. Documentar o código fonte C++; lista de comandos pode ser consultada aqui.