7ªaula prática – Árvores binárias de pesquisa. Árvores binárias

Faça download do ficheiro ***aeda2021_p07.zip*** e descomprima-o (contém a pasta ***lib***, a pasta ***Tests*** com os ficheiros ***dictionary.h, dictionary.cpp***, ***bst.h, dic.txt, binaryTree.h, game.h e game.cpp*** e ***tests.cpp,*** e os ficheiros ***CMakeLists*** e ***main.cpp***)

Deverá realizar a ficha respeitando a ordem das alíneas.

Enunciado

1) Dicionários eletrónicos são ferramentas muito úteis. Mas se não forem implementados numa estrutura adequada, a sua manipulação pode ser bastante morosa. Pretende-se implementar um dicionário utilizando uma árvore de pesquisa binária (**BST**) onde as palavras se encontram ordenadas alfabeticamente. Considere que a árvore contém objetos da classe **WordMeaning:**

```
class WordMeaning
{
    string word;       // the word
    string meaning;  // meaning of the word (explanation)
public:
    WordMeaning(string w, string m): word(w), meaning(m){}
    string getWord() const { return word; }
    string getMeaning() const { return meaning; }
    void setMeaning(string m) { meaning=m; }
};
```

Pretende-se implementar a classe **Dictionary**, com a seguinte estrutura:

```
class Dictionary {
    BST<WordMeaning> words;
public:
    BST<WordMeaning> getWords() const;
};
```

a) Complete a classe **Dictionary**, declarando convenientemente o membro-dado *words*. Implemente também o membro-função:

```
void Dictionary::readDictionary (ifstream &f)
```

que lê, a partir de um ficheiro, as palavras e o seu significado e guarda essa informação na árvore. O ficheiro contém um número par de linhas, em que a primeira linha contém a palavra e a linha seguinte o seu significado.

```
cat
feline mammal
banana
fruit
...
```

Nota:

Para aceder a ficheiros no seu programa (necessário ler o ficheiro *dic.txt*), pode:

a) especificar o caminho absoluto, ou

b) alterar "Working directory" no CLion (Run -> Edit Configurations… -> Working directory) para a pasta onde os ficheiros se encontram, ou

c) adicionar ao ficheiro *CMakeLists.txt* a diretiva

set(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}/Tests")

neste caso, os ficheiros compilados e a ler/escrever são colocados na pasta *projeto*/Tests

---

b) Implemente na classe **Dictionary** o membro-função:

```
void Dictionary::print() const
```

que imprime no monitor o conteúdo do dicionário ordenado alfabeticamente por palavras, no formato:

```
word1
meaning of word1
word2
meaning of word2
....
```

c) Implemente na classe **Dictionary** o membro-função:

```
string Dictionary::searchFor(string word) const
```

que retorna o significado da palavra indicada como argumento. Se a palavra não existir, lança a exceção **WordInexistent**. Esta classe possui os seguintes métodos que deve implementar:

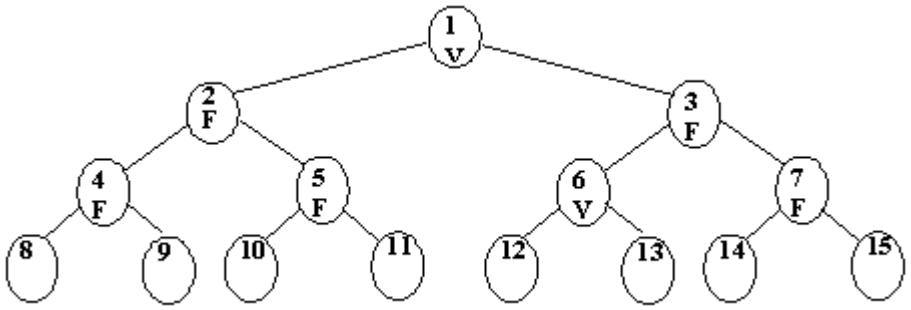
```
string WordInexistent::getWordBefore() const
// returns the word immediately before

string WordInexistent::getMeaningBefore() const
// returns the meaning of the word immediately before

string WordInexistent::getWordAfter() const
// returns the word immediately after

string WordInexistent::getMeaningAfter() const
// returns the meaning of the word immediately after
```

d) Implemente na classe **Dictionary** o membro-função:

```
bool Dictionary::correct(string word, string newMeaning)
```

que modifica o significado da palavra `word` para um novo significado `newMeaning` indicado no argumento. Se a palavra para a qual se pretende alterar o significado existir, o método retorna true, senão esta nova palavra é adicionada ao dicionário e o método retorna false.

2) Uma bola é lançada sobre um conjunto de círculos dispostos sob a forma de uma árvore binária completa (ver a figura).



(*nota*: uma árvore binária completa tem todos os seus níveis completamente preenchidos)

Cada círculo tem uma pontuação e um estado representado por um valor booleano que indica qual o caminho que a bola percorre quando chega a esse círculo: se o estado é igual a falso, a bola vai para a esquerda; se é igual a verdadeiro, a bola vai para a direita. Quando a bola passa por um qualquer círculo, este muda o seu estado: se era verdadeiro passa a falso; se era falso passa a verdadeiro. Sempre que a bola passa num círculo, é também incrementado o número de visitas a esse círculo (inicialmente, igual a 0). Quando a bola atinge um círculo na base (nó da árvore), o jogador que lançou a bola ganha o número de pontos inscritos nesse círculo. Ganha o jogo o jogador que conseguir a maior soma de pontos em uma série de *n* lançamentos.
Utilize uma árvore binária (**BinaryTree**) para representar o conjunto de círculos que constituem o tabuleiro de jogo (classe **Game**). A informação contida em cada nó da árvore está representada na classe **Circle**:

```
class Circle {
    int points;
    bool state;
    int nVisits;
public:
    Circle(int p=0, bool s=false): points(p), state(s), nVisits(0) {}
    //…
};

 class Game {
   BinaryTree<Circle> game;
public:
   BinaryTree<Circle> &getGame () { return game; }
};
```

a) Implemente o construtor da classe Jogo, que cria um tabuleiro de jogo:

```
Game::Game(int h, vector<int> &points, vector<bool> &states )
```

Esta função cria uma árvore binária completa, de altura *h*. Os vetores *points* e *states* representam a pontuação e o estado dos círculos (nós da árvore) quando se efetua uma visita por nível.

Nota: Se numerar a posição dos nós de uma árvore visitada por nível de 0 a n-1 (n= nº de nós da árvore), o nó na posição *p* possui o filho esquerdo e o filho direito nas posições *2*p+1* e *2*p+2*, respetivamente.

b) Implemente a função que descreve o estado do jogo em um determinado instante:

```
string Game::writeGame()
```

Esta função retorna a informação sobre todo o tabuleiro de jogo no formato *"points-state-nVisits\nl"* para cada um dos círculos, onde *points* é um inteiro, *state* é "true" ou "false" e *nVisits* é o nº de visitas já efetuadas a esse círculo, quando a árvore de jogo é percorrida por nível.

c) Implemente a função que realiza uma jogada:

```
int Game::move()
```

Esta função realiza uma jogada, segundo as regras já descritas, devendo alterar o estado e incrementar o número de visitas de todos os círculos por onde a bola passa. A função retorna a pontuação do círculo base (folha da árvore) onde a bola lançada termina o seu percurso.

Sugestão: use um iterador por nível para percorrer a árvore.

d) Implemente a função que determina qual o círculo mais visitado:

```
int Game::mostVisited()
```

Esta função retorna o número de visitas realizadas ao círculo mais visitado até ao momento, de todas as jogadas já realizadas (com exceção da raiz da árvore, claro).