6ªaula prática - Árvores binárias de pesquisa. Árvores binárias.

Instruções

- Faça download do ficheiro ***aed2122_p06.zip*** da página da disciplina e descomprima-o (contém a pasta ***lib,*** a pasta ***Tests*** com os ficheiros ***dictionary.cpp, dictionary.h, game.cpp, game.h, bst.h, binaryTree.h*** e ***tests.cpp***, e os ficheiros ***CMakeLists*** e ***main.cpp***)
- Deverá realizar esta ficha respeitando a ordem das alíneas.

1. Dicionários eletrónicos são ferramentas muito úteis. 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 **WordMean**, e o dicionário está representado pela classe **Dictionary**.

```
class WordMean {
   string word;
   string meaning;
public:
   WordMean(string w, string m);
   string getWord() const;
   string getMeaning() const;
   void setMeaning(string m);
};
```

```
class Dictionary {
   BST<WordMean> words;
public:
   void readFile(ifstream& fich);
   void print() const;
   bool update(string word1, string mean1);

};
```

a) Analise a classe **Dictionary**, verificando a declaração do membro-dado *words*. Implemente o membro-função:

void Dictionary::readFile(ifstream& fich)

Esta função lê, a partir de um ficheiro (*), as palavras e o seu significado, e guarda essa informação na árvore *words*. O ficheiro é composto por um número par de linhas, em que a primeira linha contém a palavra e a linha seguinte o seu significado.

```
gato
mamifero felino
morango
fruto
…
```

b) Implemente o membro-função:

void Dictionary::print() const

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

```
palavra1
significado da palavra1
palavra2
significado da palavra2
…
```

c) Implemente o membro-função:

string Dictionary::consult(string word1, WordMean& previous, WordMean& next) const

Esta função retorna o significado da palavra *word1*. Se a palavra *word1* não existir no dicionário, a função retorna a string "*word not found*" e deve colocar em *next* e *previous* os objetos **WordMean** existentes no dicionário relativos às palavras imediatamente antes e imediatamente depois (ordem alfabética) da palavra *word1*, respetivamente.

d) Implemente o membro-função:

bool Dictionary::update(string word1, string mean1)

Esta função modifica o significado da palavra *word1* para um novo significado *mean1*. Se a palavra *word1* existir no dicionário, o método retorna *true*, senão esta nova palavra com significado *mean1* é adicionada ao dicionário e o método retorna *false*.

e) Implemente o membro-função:

int BST<Comparable>::size(const Comparable& el) const

Esta função retorna o tamanho do nó da BST que contém o elemento (*element*) *el*, ou -1 se o elemento não existir.

Na resolução, pode ser útil a utilização do membro-função privado da BST:

BinaryNode<Comparable> find(const Comparable& x, BinaryNode<Comparable>* t) const*

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

a) especificar o caminho absoluto, ou

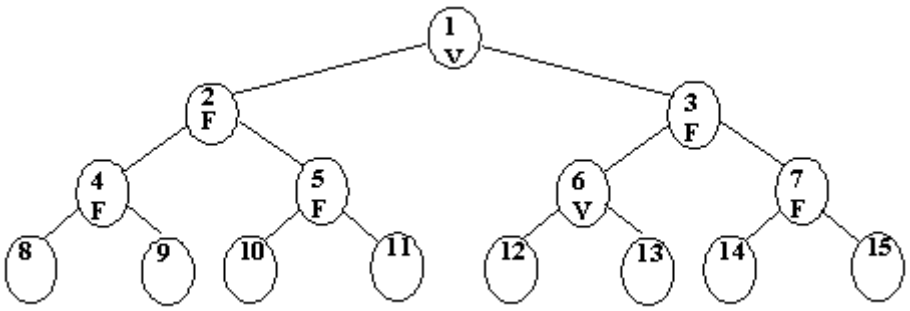
b) alterar "*Working directory*" no CLion (*Run -> Edit Configurations… -> Working directory*) para a pasta onde os ficheiros se encontram, ou

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

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, com possível exceção do último nível de um certo ponto para a direita)

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);
    int getPoints() const;
    bool getState() const;
    void changeState();
    int getNVisits() const;
    int incNVisits();
};


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

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

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

Esta função cria uma árvore binária completa, de altura *height*. 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 realiza uma jogada:

```
int Game::play()
```

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.

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

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

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).