

Estruturas de Dados

Trabalho 2 – Árvore Vermelho e Preto

Fabício V. Matos

Draft 0

1. Correções

2. Regras do Jogo

- As regras aqui definidas têm precedência sobre eventuais regras apresentadas informalmente em sala de aula;
- O trabalho deve ser desenvolvido individualmente ou em duplas;
- Data/Hora limite de entrega: **11/07/2008 07:00:00** (segunda-feira às 7 da manhã)
- Estarão sujeitos a não serem corrigidos, recebendo nota zero, trabalhos:
 - Entregues fora do prazo;
 - Incompletos (faltando arquivos ou classes);
 - Com erro de compilação;
 - Com erro de execução grave (aborta ou não exibe resultado);
- Cada função deve ter uma indicação (comentário) de sua complexidade de tempo: $O(n)$, $O(\log n)$, etc.;
- Será avaliado:
 - 40% – Correção (todos os procedimentos se comportam da forma esperada);
 - 10% – Apresentação (identação, nomes de variáveis, etc.);
 - 20% – Clareza do código (códigos bem legíveis e comentados);
 - 30% – Elegância (algoritmos criativos e eficientes). Surpreendam-me!
- É importante destacar em sua documentação (arquivo doc.pdf ou doc.txt) questões relevantes relacionadas à elegância de suas soluções para evitar que ela possa vir a passar despercebida;
- Formato de entrega:
 - Um único arquivo “ed20051-trab1.tar.gz” anexado a um email:
 - Destino: `fabricao@qualidata.com.br`
 - Assunto: `ed20051-trab1`
 - Conteúdo: Na primeira linha os nomes com “.” no lugar do espaço e separados por “&” (caso o trabalho seja em grupo). Na segunda linha os emails separados por &. Exemplo:
`Trifosa.da.Silva&Trifena.Pereira`
`trifosa@uol.com.br&trifenapereira@hotmail.com`
 - O arquivo deve conter apenas os arquivos: *.cpp, *.h, makefile e, opcionalmente, doc.pdf (ou doc.txt) contendo a documentação do trabalho.
 - O arquivo makefile deve ter, pelo menos, as regras: “all” para compilar tudo e “clear” para excluir todos os executáveis e arquivos objeto (*.o).
 - Após enviar o email, você deverá receber uma mensagem (não automática) confirmando o recebimento do email. Só após receber tal confirmação você deve assumir que o trabalho foi entregue.
- Para todos efeitos, a data/hora efetiva de entrega do trabalho será a data fornecida pelo servidor `mail.qualidata.com.br` que constará no cabeçalho do e-mail (usuários Hotmail estão tendo problemas com o horário – tentem enviar com bastante antecedência).

3. O Programa Principal

- O nome do arquivo que conterá a função “main” será `run-rbtree.cpp`
- O arquivo `makefile` deve gerar um executável denominado `run-rbtree`.
- O dado a ser manipulado será:

```
typedef struct {  
    int Cod;  
    char Name[256];  
} Pessoa;
```

- A função para `getLabel(void *, char *)` irá apenas converter o inteiro “Cod” para string. Note que o `(void *)` apontará para um struct `Pessoa`.

Opções de Linha de Comando:

- a. **Linha de comando:** `[]# ./run-rbtree -s input.txt < searches.txt`
- input.txt : nome do arquivo que conterá os dados (código e nome) das Pessoas a serem indexadas pela Árvore Vermelho e Preto. Formato¹:

Definição	Exemplo (input.txt)
<Número de Elementos N>	1000
<código>TAB<nome>	1342 Fulano de Tal
<código>TAB<nome>	3442 Ciclano de Tal e Tal
...	...

- P
arâmetro “-s” (search) indica que serão esperadas entradas relativas às buscas que devem ser executadas. Tais entradas serão fornecidas pela entrada padrão (via `scanf` e redirecionamento de arquivos) obedecendo o seguinte formato:

Definição	Exemplo (searches.txt)
<Número de Buscas N>	4
<1º Código a ser buscado>	1342
<2º Código a ser buscado>	2654
...	9856
<Nº Código a ser buscado>	3442

- Saída: O sistema deverá imprimir stdout, na mesma ordem da entrada, os códigos fornecidos para busca, seguidos do nome da pessoa encontrada. Caso não encontre alguma pessoa, deve retornar “[Item not found]” no lugar de seu nome. Ao final deve imprimir a quantidade de elementos não encontrados conforme o exemplo abaixo:

Entrada (searches.txt)	Saída (stdout)
4	1342 - Fulano de Tal
1342	2654 - [Item not found]
2654	9856 - [Item not found]
9856	3442 - Ciclano de Tal e Tal
3442	=> 2 item(s) not found.

¹ Em cada linha deve haver uma pessoa: código seguido do caracter TAB (‘\t’) seguido do nome (string que se estende até o final da linha).

- b. Linha de comando:** []# ./run-rbtree -t -pre input.txt > preorder.txt
- o input.txt : mesmo formato;
 - o Parâmetro “-t” (Traversing) indica que não serão realizadas buscas, mas que após a árvore RBTree ser construída, todos os seus elementos serão exibidos no saída padrão (via printf) seqüencialmente na ordem definida pelo segundo parâmetro. Neste caso, -pre indica “Pré-Order”.
 - o Os valores possíveis para o segundo parâmetro são “-pre”, “-pos” e “-in” que significam, respectivamente, “Pré-Order”, “Pós-Order” e “In-Order”.
 - o Formato do output (na saída padrão): mesmo formato da entrada (código + nome), porém seguindo a ordenação indicada e sem a primeira linha (número de elementos).

4. A Classe TRBTree

Classe que implementa uma Arvore Binária de Busca Balanceada genérica – Árvore Vermelho e Preto genérica.

a. Tipos Enumerados

```
//Traversing Direction
enum TraversingDir {
    sdInOrder,
    sdPreOrder,
    sdPosOrder
};
```

b. Nó da árvore – Sugestão (faça como preferir)

```
//Colors of the nodes
enum RBColor {
    cRed,
    cBlack
};

typedef struct node {
    void *Data;
    struct node *pRight, *pLeft, *pFather;
    RBColor Color;
} RBNode;
```

c. Classe de Exceção

```
class TRBTreeError: public std::exception {
public:
    TRBTreeError (char *msg) { printf("[RBTreeError] %s\n", msg);
    };
};
```

d. Membros Públicos da Classe

```
TRBTree(int(*fnCompare)(void*, void*), void(*getLabel)(void*, char*));
```

- Método construtor.
- Se *fnCompare* for *NULL*, deve gerar uma exceção:
 throw TRBTreeError ("fnCompare is required");
- *FnCompare(a,b)* deve ter os seguintes retornos:
 - o *a < b => return -1*
 - o *a = b => return 0*
 - o *a > b => return 1*
- *getLabel(void*,char*)*, quando chamada, irá receber um o ponteiro para o campo *Data* de um nó e um *string* previamente alocado, com tamanho máximo de 256 caracteres.
- *getLabel(void *node, char *label)* deve retornar em "label" um *string* que descreve o dado contido no nó "node". Esta função será utilizada para o desenho da árvore em *PostScript*. Por exemplo, um árvore cuja chave seja inteira, deve retornar o *string* que representa o número da chave da árvore: "33", "147", "1", etc..
- *getLabel* pode ser *NULL*. Neste caso, os nós serão desenhados sem conteúdo interno.


```
~TRBTree ();
```

- Método destrutor.
- Deve liberar os nós da árvore - equivale a executar *DeleteAll()*


```
TraversingDir getTraversingDir();
```

- Retorna *FTraversingDir*


```
void setTraversingDir (TraversingDir value);
```

- Faz *FTraversingDir = value*
- Não altera *pCurrent*


```
void Insert(void *data);
```

- Insere um novo elemento na árvore
- Se já existir um elemento com mesma chave, deve gerar uma exceção:
 throw TRBTreeError ("Duplicated insertion isn't allowed");
- Atualiza *FCount* (*FCount++*), se necessário.
- Se *data == NULL* deve gerar uma exceção:
 throw TRBTreeError ("Insert(NULL) isn't allowed");
- Não atualiza *pCurrent*


```
bool Delete(void *data);
```

- Se existir um elemento com a mesma chave de "data", exclui tal elemento e retorna "true". Senão, retorna "false".
- Atualiza *pCurrent* para o sucessor do elemento excluído;
- Se o elemento excluído for o maior, atualiza para seu antecessor;
- Se *data = NULL*, deve gerar uma exceção:
 throw TRBTreeError ("Delete(NULL) isn't allowed");
- Atualiza *FCount* (*FCount--*), se necessário.


```
bool Destroy(void *data);
```

- Se existir um elemento com a mesma chave de "data", exclui tal elemento liberando o "Data" (*free(pCurrent->Data);*) e retorna "true". Senão, retorna "false".
- Atualiza *pCurrent* para o sucessor do elemento excluído;
- Se o elemento excluído for o maior, atualiza para seu antecessor;
- Se *data = NULL*, deve gerar uma exceção:
 throw TRBTreeError ("Destroy(NULL) isn't allowed");
- Atualiza *FCount* (*FCount--*), se necessário.

```

void *Search(void *data);
    ▪ Se existir um elemento com a mesma chave de "data", atualiza pCurrent e o retorna. Senão, retorna, não altera pCurrent e retorna NULL.
    ▪ Se data = NULL, deve gerar uma exceção:
      throw TRBTreeError ("Search(NULL) isn't allowed");

void Last();
    ▪ Atualiza pCurrent para o último elemento de uma travessia In-Order, Pré-Order ou Pós-Order (de acordo com FTraversingDir) da árvore.

void First();
    ▪ Atualiza pCurrent para o primeiro elemento de uma travessia In-Order, Pré-Order ou Pós-Order (de acordo com FTraversingDir) da árvore.

void Prior();
    ▪ Atualiza pCurrent para o elemento anterior de uma travessia In-Order, Pré-Order ou Pós-Order (de acordo com FTraversingDir) da árvore.
    ▪ Se pCurrent = NULL, equivale a "Last()".
    ▪ Se estiver no primeiro, faz pCurrent = NULL

void Next();
    ▪ Atualiza pCurrent para o elemento seguinte ao corrente considerando uma travessia In-Order, Pré-Order ou Pós-Order (de acordo com FTraversingDir) da árvore.
    ▪ Se pCurrent = NULL, equivale a "First()".
    ▪ Se estiver no último, faz pCurrent = NULL

void *getCurrent();
    ▪ Retorna pCurrent->Data

void DeleteAll();
    ▪ Exclui todos os nós da árvore. Faz pCurrent = NULL;

void DestroyAll();
    ▪ Semelhante a DeleteAll() mas libera o "Data" dos nós.

int getCount();
    ▪ Retorna FCount;

bool GoToFather();
    ▪ Se pCurrent ≠ NULL e pCurrent->pFather ≠ NULL, faz pCurrent = pCurrent->pFather e retorna "true"
    ▪ Se pCurrent = NULL ou pCurrent->pFather = NULL, não faz nada e retorna "false"

bool GoToLeft();
    ▪ Se pCurrent ≠ NULL e pCurrent->pLeft ≠ NULL, faz pCurrent = pCurrent->pLeft e retorna "true"
    ▪ Se pCurrent = NULL ou pCurrent->pLeft = NULL, não faz nada e retorna "false"

bool GoToRight();
    ▪ Se pCurrent ≠ NULL e pCurrent->pRight ≠ NULL, faz pCurrent = pCurrent->pRight e retorna "true"
    ▪ Se pCurrent = NULL ou pCurrent->pRight = NULL, não faz nada e retorna "false"

bool GoToRoot();
    ▪ Se pRoot ≠ NULL, faz pCurrent = pRoot e retorna "true"
    ▪ Se pRoot = NULL, não faz nada e retorna "false"

```

void Successor();

- Se *pCurrent* ≠ *NULL*, aponta para o sucessor. Se não existir, *pCurrent* recebe *NULL*. Se *pCurrent* = *NULL*, não faz nada.

void Antecessor();

- Se *pCurrent* ≠ *NULL*, aponta para o antecessor. Se não existir, *pCurrent* recebe *NULL*. Se *pCurrent* = *NULL*, não faz nada.

void WriteGraphvizSource(FILE *f, bool blackwhite = true);

- Gerar o código fonte no formato "DOT" da árvore, escrevendo-o (via *fprintf*) no arquivo "f". **Obs.:** Para escrever na saída padrão basta chamar a função passando *stdout*: *RBTree->WriteGraphvizSource(stdout)*
- Esta função assume que o descritor de arquivos "f" já está devidamente aberto (quem chama deve abri-lo).
- A árvore deve começar com a letra "T" apontando para o nó raiz.
- Se a árvore estiver vazia, deve aparecer apenas a letra "T"
- Se *blackwhite* = *true*, os nós vermelhos devem permanecer brancos e os nós pretos devem ser cinzas (sombreados). Caso contrário, os vermelhos ficam vermelhos e os pretos ficam pretos.
- O valor default (caso não seja informado na chamada da função) deve ser preto e branco (*blackwhite* = *true*).

e. Membros Protegidos da Classe

Nenhum.

f. Membros Privados da Classe

```
void LeftRotate(RBNode *node);
void RightRotate(RBNode *node);
//Obs.: Além destes métodos, quaisquer outros que sejam necessários.
```

```
//Atributos internos
RBNode *pRoot, *pCurrent;
int FCount;
int(*pFnComp)(void*, void*);
void(*pFnGetLabel)(void*, char*);
TraversingDir FTraversingDir;
```

- A direção *FTraversingDir* default deve ser *sdInOrder*

5. Sugestões de Implementação

- Implemente primeiro a *RBTree* como se fosse apenas um Árvore de Busca Binária (ignore o campo *Node.Color*). Depois que estiver funcionando, faça as alterações necessárias para que ela seja de fato um Árvore Vermelho e Preto.
- Implementar primeiro *WriteGraphvizSource()* pode facilitar a detecção de bugs (poderá ser detectado visualmente), economizando tempo de desenvolvimento. Uma possibilidade é implementar apenas a inserção (não balanceada) e em seguida a função *WriteGraphvizSource()*. Só então se implementaria as outras funções (rotação, exclusão, etc...)

Nota 1: Esta é versão ainda está sujeita a pequenas modificações;

Nota 2: Qualquer eventual erro ou discrepância encontrada nesta especificação deve ser notificado ao professor imediatamente para que seja corrigida na versão final desta especificação;

Nota 3: Algumas (poucas) definições propostas em sala foram propositalmente modificadas visando a melhoria das estruturas de dados;