BCC202 – Estruturas de Dados I (2023-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)
Estagiário docente: **Natália Fernanda de Castro Meira**

Trabalho Prático III (TP III) - 10 pontos, peso 1.

- Data de entrega: 04/02/2024 até 23:55. O que vale é o horário do *Moodle*, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação;

- O trabalho é em grupo de até 2 (duas) pessoas.

- Entregar um relatório.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

Salvando dados na Árvore Rubro-Negra

Uma árvore rubro-negra é um tipo de árvore binária de busca balanceada. Diferente das árvores AVL que utilizam a altura das sub-árvores para balanceamento, a RBT (*Red-black tree*), ou Árvore Rubro-Negra, utiliza um esquema de coloração dos nós para manter o balanceamento da árvore. Cada nó da árvore possui um atributo de cor, que pode ser vermelho ou preto. Além da cor, a árvore deve satisfazer o seguinte conjunto de propriedades:

- Todo nó da árvore é vermelho ou preto.
- A raiz é sempre preta.
- Todo nó folha (`NULL`) é preto.
- Se um nó é vermelho, então os seus filhos são pretos.
- Não existem nós vermelhos consecutivos.
- Para cada nó, todos os caminhos desse nó para os nós folhas descendentes contém o mesmo número de nós pretos.

O balanceamento da árvore é feito através de rotação e ajuste das cores a cada inserção ou remoção. A Figura 1 apresenta um exemplo de uma árvore rubro-negra.

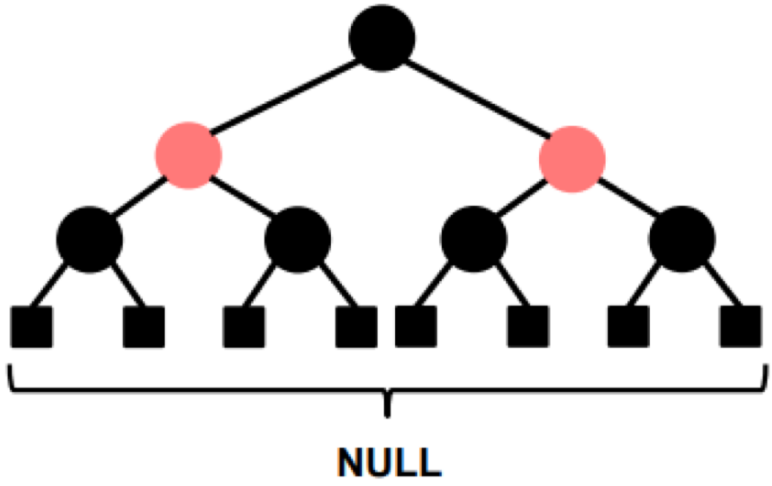

Figura 1: Exemplo da RBT.

Dada a RBT, considere uma biblioteca que deseja visualizar os dados de várias pessoas por idade para ver melhor qual as idades das pessoas que fazem uso dela. Para isso, você utilizará a árvore rubro-negra para inserção desses dados. Por se tratar de uma árvore balanceada, ficará fácil a visualização.

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada).
- Um grande número de *Warnings* ocasionará a redução na nota final.

O que deve ser entregue

- Código fonte do programa em C (**bem identado e comentado**).
- Documentação do trabalho (relatório[1]). A documentação deve conter:

[1]Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

1. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
2. **Implementação:** descrição sobre a implementação do programa. **Não faça** "*print screens*" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
3. **Testes:** descrição dos testes realizados e listagem da saída (não edite os resultados).
4. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho. Por exemplo, avaliar o tempo gasto de acordo com o tamanho do problema.
5. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
6. **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.
7. **Formato:** PDF ou HTML.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *Moodle* até 04/02/2024 até 23:55, um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF**.

Detalhes da implementação

Para atingir o seu objetivo, você deverá construir um Tipo Abstrato de Dados (TAD) `RBTree` como representação da árvore que você irá implementar. Ele possui os seguintes atributos: dado, cor, nó pai, esquerda e direita. O TAD deverá implementar, pelo menos, as seguintes operações:

- `alocarArvore`: aloca um (ou mais) TAD `RBTree`.
- `desalocarArvore`: desaloca um TAD `RBTree`.
- `leArvore`: inicializa o TAD `RBTree` a partir de dados do terminal.
- `rotacaoDireita`: função que realiza a rotação da árvore para a direita.
- `rotacaoEsquerda`: função que realiza a rotação da árvore para a esquerda.
- `insercao`: função para inserir um nó na árvore.
- `balanceamento`: função que realiza o balanceamento da árvore.
- `printInOrder`: função para imprimir a árvore *inOrder*.

Outro TAD a ser implementado é o TAD `Pessoa` que tem os atributos nome, data de nascimento e idade.

O TAD deve ser implementado utilizando a separação interface no `.h` e implementação `.c` bem como as convenções de tradução.

Considerações

O código-fonte deve ser modularizado corretamente em três arquivos: `tp.c`, `arvore.h` e `arvore.c`. O arquivo `tp.c` deve apenas invocar e tratar as respostas das funções e procedimentos definidos no arquivo `grafo.h`. A separação das operações em funções e procedimentos está a cargo do aluno, porém, não deve haver acúmulo de operações dentro de uma mesma função/procedimento.

O limite de tempo para solução de cada caso de teste é de apenas **um segundo**. Além disso, o seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada). *Warnings* ocasionará a redução pela metade da nota final. Assim sendo, utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

Entrada

A entrada é dada por meio do terminal. Para facilitar, a entrada será fornecida por meio de arquivos. **Use eles como entrada via terminal (`./executavel < arquivo_teste.txt`).**

A primeira linha especifica a operação a ser realizada: 1 - Inserção, 2 - Impressão *InOrder* da árvore, 0 - para sair do *loop*. A segunda linha depende da opção selecionada na primeira, caso seja 1, então o próximo dado é a quantidade de dados a serem inseridos. Após a quantidade de dados, são inseridos os dados na seguinte ordem: nome, data de nascimento e idade, lembrando que é apenas o primeiro nome. Após finalizar a inserção dos dados é solicitado novamente a opção 1, 2 ou 0. Caso a opção escolhida seja 2, a árvore é impressa na tela *InOrder*. E caso seja 0, finaliza o *loop*. Após finalizar o *loop*, os dados são exibidos *inOrder*.

Saída

A saída consiste nos dados na árvore impressos *inOrder*.

Exemplo de caso de teste

Exemplos de saídas esperadas dada uma entrada:

Entrada	Saída
1 // primeira opção 3 // inserir 3 dados Barbara 26/10/1999 23 Luis 18/09/2013 9 Lucas 14/01/1997 26 2 // imprimir a árvore 0 // sair do loop	Dados inOrder: Nome: Luis Data de Nascimento: 18/09/2013 Idade: 9 Nome: Barbara Data de Nascimento: 26/10/1999 Idade: 23 Nome: Lucas Data de Nascimento: 14/01/1997 Idade: 26

A SAÍDA DA SUA IMPLEMENTAÇÃO DEVE SEGUIR EXATAMENTE A SAÍDA PROPOSTA.

Diretivas de Compilação

As seguintes diretivas de compilação devem ser usadas (essas são as mesmas usadas no *Moodle*).

```
$ gcc -c arvore -Wall
$ gcc -c tp.c -Wall
$ gcc arvore.o tp.o -o exe -lm
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta *valgrind*. O *valgrind* é um *framework* de instrumentação para análise dinâmica de um código e é muito útil para resolver dois problemas em seus programas: **vazamento de memória e acesso a posições inválidas de memória** (o que pode levar a *segmentation fault*). Um exemplo de uso é:

```
1 gcc -g -o exe arquivo1.c arquivo2.c -Wall
2 valgrind --leak-check=full -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
1 ==xxxxx== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.

O SEU CÓDIGO SERÁ TESTADO NOS COMPUTADORES DO LABORATÓRIO (AMBIENTE LINUX)