DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
MATD04 – Estruturas de Dados
Prof. Roberto Freitas Parente
roberto.parente@ufba.br

Trabalho 04

1 Instruções:

- O trabalho é **individual**. O nome do integrante deve estar no cabeçalho comentado no arquivo.
- Será aplicado detector de plágio! Uma vez detectado plágio, a **nota da disciplina** será **zerada**.
- Data de entrega: **08 de dezembro de 2019 (domingo)** no sistema do moodle até as 23:55 (a cada hora (ou fração) atrasada será descontado 1 ponto).
- A entrega deve ser no formato ZIP com o seguinte conteúdo:

  + Codigo-fonte (bem como os arquivos `.c` e `.h` referentes aos TAD utilizados);

  + Arquivo Makefile[1] associado para compilar o código.

- O código deve estar escrito em `Linguagem C` e as únicas bibliotecas permitidas são: `stdio.h`, `stdlib.h` e `string.h` caso contrário o trabalho será invalidado.
- Para validação do código será utilizado um ambiente GNU/Linux com o compilador `GCC` (versão 7+) passando os seguintes parâmetros[2]:
  `gcc -Wall -Wextra -Werror -Wpedantic`
- Para verificação de vazamento de memória e gerência dos ponteiros, será utilizado o software `Valgrind`[3].
- Para verificação de correspondência com a saída especificada será utilizado o software `diff`[4].

2 Correção

Serão utilizados os seguintes critérios para correção:

2.1 Compilação

- Compilação com parâmetros requeridos;
- Makefile funcionando;
- **Se não compilar utilizando o Makefile enviado, o trabalho não será corrigido.**

2.2 Entrada/Saída

- Cada entrada gera uma saída correta.

2.3 TAD / Estrutura de dados

- Implementação correta das estruturas de dados.
- Implementação correta do conceito de TAD.

2.4 Organização do código

- Funções separadas para cada uma das operações.
- Comunicação por retorno de função.

2.5 Gerência de memória

- Teste de alocação.
- Remoção correta da memória alocada.
- Vazamentos de memória.
- Controle de ponteiros.

Observação: A correção será em boa parte automatizada. Faça o programa funcionar de acordo com as especificações!

[1] Veja mais sobre em `https://pt.wikibooks.org/wiki/Programar_em_C/Makefiles`.
[2] Leia mais sobre em: `https://gcc.gnu.org/onlinedocs/gcc-6.4.0/gcc/Warning-Options.html`.
[3] `http://valgrind.org`
[4] `https://www.geeksforgeeks.org/diff-command-linux-examples/`

Otimização de operações bancárias – Um protótipo

Um banco deseja otimização seus sistema de operações bancárias. Atualmente o sistema deles armazena todas as informações dos clientes em uma lista ordenada. A proposta do banco é que você desenvolva uma organização dos dados utilizando Árvores AVL, pois o gerente de T.I. do banco sabe que as operações em uma AVL têm custo de $O(\log n)$. Desta forma, você objetivo é elaborar uma TAD para uso de Árvore AVL e uma interface de teste que seja capaz de receber dados pela entrada padrão e computar a saída.

Simulação

A entrada constará de uma sequência de operações que seu programa deve conseguir computá-las e retornar uma saída determinada. Assim como na versão completa da simulação, será informada uma quantidade N de operações. Cada operação será identificada com uma letra seguida dos seus devidos parâmetros. Os dados de entrada são as informações dos cliente: Código_cliente: Inteiro; Operação: 1 para saque e 0 para depósito; e Valor: valor inteiro. A `chave de ordenação` deve ser o código do cliente.

A árvore deve guardar as informações e gerar um relatório descrito posteriormente que consta na quantidade de clientes distintos que foram inseridos bem como as informações referêntes ao código do cliente, quantidade de operações e saldo final.

3 Formato de entrada

A entrada constará de uma sequência de operações, terminada com uma **operação de término de sequência de operações**. As operações que podem ocorrer são:

- **insere nó:** esta operação consiste de uma linha contendo a letra 'i', seguida de um espaço e três números inteiros separados com um espaço que serão respectivamente código_cliente, operação e valor. Esta operação causa a inserção de um nó na árvore cuja chave será o número inteiro referente ao código_cliente. Caso já haja nó na árvore com valor de chave igual a este número, o nó com tal chave deve ter seus dados atualizados de acordo com a operação informada.

- **consulta nó:** esta operação consiste de uma linha contendo a letra 'c', seguida por um espaço, seguido por um valor inteiro. Esta operação verifica se há ou não nó na árvore com valor de chave igual ao número inteiro digitado.

- **remove nó:** esta operação consiste de uma linha contendo a letra 'r', seguida por um espaço, seguido por um valor inteiro. Esta operação retira o nó da árvore que tiver chave igual ao número inteiro indicado, se houver. Se não houver, esta operação não gera efeito.

- **lista chaves dos nós da árvore em ordem:** esta operação consiste de uma linha contendo a letra 'p', seguida de um espaço, seguido da letra 'c' ou 'd'. Se a segunda letra é 'c', esta operação lista as chaves dos nós da árvore em ordem crescente. Se a segunda letra for 'd', esta operação lista as chaves dos nós da árvore em ordem decrescente.

- **lista chaves de um determinado nível da árvore:** esta operação consiste de uma linha contendo a letra 'n', seguida de um espaço, seguido de um número inteiro maior ou igual a 1. Esta operação lista as chaves dos nós da árvore que tiverem nível igual ao número fornecido. Assuma que a raiz da árvore possui nível 1.

- **informa altura:** esta operação consiste de uma linha contendo a letra 'h'. Esta operação informa a altura da árvore.
- **término da entrada:** a sequência de operações será terminada por uma linha com a letra 'f'.

4 Formato da Saída

Cada operação deverá apresentar a saída da seguinte forma:

- **insere nó:** esta operação não gera saída.
- **consulta nó:** se houver um nó na árvore com a chave informada, esta operação gera, na saída, a sequência de caracteres "`existe no com chave: X`", onde X é a chave informada. Caso não haja, esta operação gera, na saída, a sequência de caracteres "`nao existe no com chave: X`", onde X é a chave informada.

  Observação: Não usamos nenhum tipo de acentuação.
- **remove chave:** esta operação não gera saída (independentemente se há ou não nó com o valor de chave indicado na operação).
- **lista chaves dos nós da árvore em ordem:** esta operação lista os dados contidos nos nós, um em cada linha. Os dados serão três inteiros que correspondem respectivamente ao código_cliente, quantidade_de_operação e saldo.
- **lista chaves de um determinado nível da árvore:** esta operação lista as chaves dos nós, uma em cada linha. A sequência das chaves dos nós listados deve seguir a ordem da representação gráfica da árvore, da esquerda para a direita.
- **informa altura:** esta operação deve retornar um inteiro que informa a altura da árvore corrente. Considere que a árvore vazia tem algura zero.
- **término da entrada:** esta operação irá gerar um relatório com todos os dados armazenados na árvore. Para tal, deve iniciar imprimindo a seguinte sequência de caractere "`-+- Inicio relatorio -+-`", depois imprimir o inteiro L que representa a quantidade distintas de clientes seguido de L linhas que devem conter três inteiros C_i, Q_i e S_i, para $1 \leq i \leq L$, separados por um espaço em branco e finalizar com a seguinte sequência de caractere "`-+- Fim relatorio -+-`". A ordem de impressão dos dados é dada pela operação de remoção da raiz da árvore.

Restrições

Algumas restrições para a implementação das estruturas de dados do sistema:

- Deve-se utilizar a entrada padrão para receber os dados e a saída padrão para imprimir os dados;
- Cada nó da árvore deve, no mínimo, ter três ponteiros onde se referenciam, respectivamente, ao seu filho esquerdo, filho direito e pai;
- O sistema tem que utilizar **fortemente** o conceito de tipo abstrato de dados (TAD) na implementação da árvore AVL, i.e., as definições dos nós e das estruturas de dados da árvore devem estar definidas no arquivo `avl.c` e as interfaces/assinaturas das funções e tipos no arquivo`avl.h`;

- Não deve utilizado em **nenhuma hipótese** variáveis globais tanto no arquivo principal quanto no TAD;
- O conjunto: TAD + arquivo principal deve gerenciar a memória de maneira ótima, i.e., tudo que for alocado deve ser desalocado[5];
- No arquivo principal, cada operação de entrada possível deve ter uma função para chamar as funções do TAD e formatar sua saída.

Observações

- As operaçõres de remoção, inserção, percurso e balanceamentos devem estar de acordo com o que foi visto em sala, pois caso contrário a sua saída será diferente e terá nota prejudicada.

[5]Será utilizado o software `Valgrind` para verificação de corretude.