Estruturas de Dados

Trabalho Prático 2

Informações Gerais

- O trabalho deve ser realizado individualmente.
- A entrega será feita via sistema de correção automática Testr.
  Link: http://200.137.66.73:8000/ .
- Data limite para entrega: 21/03/2025 (23:59:59).
- Trabalhos entregues após a data limite não serão corrigidos e receberão nota zero.
- Alunos cujos trabalhos tenham alto índice de interseção de conteúdo (indício de cola) serão convidados a conversar sobre a situação e poderão receber nota zero.

Contexto

O trabalho consiste em implementar uma tabela hash genérica e uma árvore binária de busca genérica e utilizar as estruturas para resolver dois problemas. As estruturas devem ser utilizadas sem modificação em ambos os problemas, isto é, informações específicas de cada problema não devem existir nos arquivos de implementação das estruturas de dados.

Problema 1: Considere um sistema que indexa ações de empresas da bolsa de valores. Cada empresa possui nome (máximo de 100 caracteres, sem espaços), sigla (máximo de 32 caracteres, sem espaço), valor unitário da ação, número total de ações e número de ações vendidas. O programa principal irá receber duas entradas. Primeiro será digitado na entrada padrão o nome de um arquivo. Este arquivo irá conter um número inteiro **n** seguido de **n** linhas, cada uma com os dados de uma empresa separados por espaço. Os dados devem ser armazenados em uma tabela hash usando a sigla como chave para recuperar os dados da empresa, e em uma árvore binária de busca usando o preço unitário como critério primário de ordenação (em caso de empate, considerar ordem alfabética da sigla). Ponteiros para os dados das empresas existirão **em ambas** as estruturas de dados e cada uma será utiliza para propósitos específicos detalhados a seguir.

A segunda entrada do programa será um número inteiro **m** seguido de **m** linhas contendo operações a serem feitas usando os dados existentes do arquivo. As operações possíveis são:

- UPDATE <sigla> <valor>: atualiza o preço da ação da empresa dada pela sigla. O valor deve ser atualizado tanto na tabela hash quanto na árvore binária. Na árvore binária o preço anterior deve ser utilizado para encontrar os dados da empresa na árvore e remover estes dados. A seguir, uma nova entrada contendo o valor atualizado deve ser inserida.
- GET <sigla>: Mostra na tela os dados da empresa que possui a sigla.
- RM <sigla>: Remove de ambas as estruturas os dados da empresa que contém a sigla. Para remover os dados da árvore, o valor das ações deverá ser utilizado para buscar o registro.
- INTERVAL <min> <max>: Mostrar na tela as siglas das empresas cujos valores das ações estão dentro do intervalo dado. A busca deve ser feita utilizando a árvore binária.
- MIN: Mostrar na tela a sigla da empresa com ações mais baratas.
- MAX: Mostrar na tela a sigla da empresa com ações mais caras.
- SORTED: Mostrar na tela as siglas das empresas e os valores das ações de forma ordenada (ascendente) pelo valor das ações. Esta operação deve ser feita usando uma função de percurso em árvores binárias. Adicionar todas as empresas em um array e ordenar o array é uma solução inválida.

Caso de Teste 1	
lst1.txt	5 Petrobras PETRO 125.63 500 433 ValeSA VALE3 98.3 8000 79822 MetaPlatformsInc META 716.37 60 10 NvidiaCorp NVDC34 16.40 125002 11023 CocaColaDRN COCA34 65.05 5081889 4172339
Entrada	lst1.txt 5 GET PETRO GET COCA34 GET NVDC34 MAX MIN
Saída	Petrobras PETRO 125.63 500 433 CocaColaDRN COCA34 65.05 5081889 4172339 NvidiaCorp NVDC34 16.40 125002 11023 META NVDC34

Problema 2: Uma empresa de jogos gostaria de implantar um sistema para criar partidas automaticamente entre jogadores que têm níveis similares. Cada jogador possui nickname (máximo de 100 caracteres, sem espaço), nome (máximo de 100 caracteres sem espaço), número de partidas disputadas e número de partidas vencidas. O programa principal irá receber duas entradas. Primeiro será digitado na entrada padrão o nome de um arquivo contendo um número inteiro **n** seguido de **n** linhas, cada uma com os dados de um jogador separados por espaço. Os dados devem ser armazenados em uma tabela hash usando o nickname como chave para recuperar os dados do jogador, e em uma árvore binária de busca usando o percentual de vitórias como critério primário de ordenação (em caso de empate, considerar ordem alfabética do nickname). Ponteiros para os dados dos jogadores existirão **em ambas** as estruturas de dados e cada uma será utiliza para propósitos específicos detalhados a seguir.

A segunda entrada do programa será um número inteiro **m** seguido de **m** linhas contendo operações a serem feitas usando os dados existentes do arquivo. As operações possíveis são:

- GET <nickname>: Mostra na tela os dados do jogador que possui o nickname.
- RM <nickname>: Remove de ambas as estruturas os dados do jogador empresa que contém o nickname. Para remover os dados da árvore, o percentual de vitórias deverá ser utilizado para buscar o registro.
- VICTORIES <nickname> <qtd>: o número de vitórias e de partidas deve ser incrementado de **qtd** para o jogador em ambas as estruturas. Na árvore, o percentual de vitórias anterior deve ser usado para remover os dados do jogador e um novo registro deve ser inserido com as quantidades atualizadas.
- DEFEATS <nickname> <qtd>: o número de partidas (mas não de vitórias) deve ser incrementado de **qtd** para o jogador em ambas as estruturas. Na árvore, o percentual de vitórias anterior deve ser usado para remover os dados do jogador e um novo registro deve ser inserido com as quantidades atualizadas.
- MATCH <nickname>: Mostra na tela o jogador com o percentual de vitórias mais parecido com o do jogador dado como entrada. Esta busca é chamada de busca por vizinho mais próximo (nearest neighbor search).
- INTERVAL <min> <max>: Mostrar na tela os nicknames dos jogadores com percentual de vitórias no intervalo. A busca deve ser feita utilizando a árvore binária.
- MIN: Mostrar na tela o nickname do jogador com o menor percentual de vitórias.
- MAX: Mostrar na tela o nickname do jogador com o maior percentual de vitórias.
- SORTED: Mostrar na tela as nicknames dos jogadores e os percentuais de vitórias de forma ordenada (descendente) pelo percentual de vitórias. Esta operação deve

ser feita usando uma função de percurso em árvores binárias. Adicionar todos os jogadores em um array e ordenar o array é uma solução inválida.

Caso de Teste 1	
lst1.txt	5 Skye MarianaRibeiro 10 10 Barto LucasBartolomeu 5 10 Morpheus123 AndreSouto 1 11 GueriGueri PauloAndreSilva 8 9 Trixter LuanaCarvalho 3 5
Entrada	lst1.txt 5 GET Skye GET Barto GET Trixter MIN MAX
Saída	Skye MarianaRibeiro 10 10 Barto LucasBartolomeu 5 10 Trixter LuanaCarvalho 1 5 Morpheus123 Skye

Regras

- A tabela hash deve realizar hash de strings usando a abordagem de multiplicar os caracteres por potências de uma base prima (31).
- As potências deveriam começar em 0 no último caractere e serem incrementas em direção à posição zero. Contudo, para evitar a geração de números excessivamente grandes e a necessidade de usar tipos com muitos bytes para armazenar os valores de hash, aplique o operador de módulo a cada novo caractere adicionado à soma começando pelo primeiro.
- O tamanho da tabela deve ser 23.
- O tratamento de colisões deve ser feito usando uma lista encadeada com novos elementos adicionados à esquerda (push_front).
- Números reais devem ser exibidos com duas casas depois da vírgula.
- Todas as estruturas de dados para armazenar coleções (lista encadeada, tabela hash e árvore binária de busca) devem ser opacas e genéricas. Tipos internos (como nós da lista encadeada ou da árvore binária de busca) devem ser ocultados do usuário das

estruturas de dados. Para isto, estes tipos internos e suas funções de manipulação devem ser definidos no arquivo de implementação (.c) e não devem ser mencionados no arquivo de cabeçalho (.h).

- Erros no valgrind levarão a punições graves nas notas.