Universidade Federal do Ceará - Campus Quixadá
QXD0115 – Estrutura de Dados Avançada – Turma 02A
Curso de Ciência da Computação
Prof. Atílio Gomes Luiz

Contagem de frequências usando Estruturas de Dados

1. Nesse trabalho, deverá ser desenvolvida uma aplicação que utilize as árvores binárias de busca balanceadas vistas em aula (AVL e Rubro-Negra) e também as tabelas hash.

   Faça um programa que leia um texto qualquer (arquivo no formato .txt) e imprima, em ordem alfabética, as palavras e a sua frequência no texto. Por exemplo, no texto "quatro mais quatro são oito" o seu programa deverá imprimir:

mais	1
oito	1
quatro	2
são	1

   A leitura do arquivo .txt deverá desprezar espaços em branco e sinais de pontuação, que serão considerados separadores de palavras. Além disso, a leitura deverá converter todas as letras maiúsculas em minúsculas. Se for necessário, por motivos de limitação da sua máquina, você pode considerar que existem no máximo 1024 palavras diferentes no texto (ou seja, se necessário, considere as primeiras 1024 diferentes palavras que aparecerem no texto e despreze as seguintes novas palavras).

   A pesquisa e a inserção das palavras do texto deverão ser implementadas com as seguintes **estruturas de dados**:

   (a) Dicionário (usando como base uma árvore AVL)

   (b) Dicionário (usando como base uma árvore Rubro-Negra)

   (c) Dicionário (usando como base uma tabela Hash com tratamento de colisão por encadeamento exterior)

   (d) Dicionário (usando como base uma tabela Hash com tratamento de colisão por endereçamento aberto)

   Coloque contadores no seu programa para determinar o número de comparações de chaves e atribuições de registros necessárias para montar a tabela de frequências em cada uma das estruturas acima (Apenas o número de comparações para montar a estrutura. Não é necessário considerar as comparações e atribuições para a impressão ordenada).

   Calcule também o tempo que cada estrutura leva apara montar a tabela.

   Você pode propor também outras métricas. Por exemplo, no tocante aos dicionários implementados por meio de árvores balanceadas, você poderia contar o número de rotações realizadas.

Analise através dos dados coletados o desempenho/eficiência de cada estrutura.

Use parâmetros de linha de comando para fazer as chamadas do seu programa. Esse tipo de execução é bastante comum em sistemas Linux. Por exemplo, se o seu programa executável chama-se `freq` e você quiser contar a frequência do arquivo `texto.txt` utilizando o dicionário com AVL, sua chamada pode ser feita assim na linha de comando do Linux (ou prompt de comando, no caso do Windows):

```
> freq dictionary_avl texto.txt
```

Descubra como fazer para o seu programa ler os parâmetros da linha de comando e defina (e documente) a sintaxe dos comandos e dos parâmetros a serem utilizados pelo usuário. Veja, nesse trabalho não estou pedindo uma aplicação com menu, mas sim uma aplicação de linha de comando.

Também faz parte do trabalho descobrir como fazer a leitura de um arquivo texto e a manipulação adequada de strings.

O que deve ser entregue

- Código fonte do programa em C++ (bem indentado e comentado).
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:
  1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  3. Listagem de testes executados e Resultados encontrados: os testes executados devem ser simplesmente apresentados. Devem ser apresentados os resultados com relação às métricas que foram pedidas acima na descrição do trabalho: número de comparação de chaves, tempo de montagem da tabela, análise do desempenho de cada ED, entre outras que você inventar.
  4. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  5. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da internet se for o caso.
  6. Formato: obrigatoriamente em PDF.
  7. Em Latex: a documentação do trabalho pode ser feita em formato de artigo. De preferência, prepare esse documento em Latex. Atualmente, existe o site **Overleaf** que facilita a produção de documentos em latex. Você pode Latex usar o modelo de artigos da Sociedade Brasileira de Computação (link aqui).

Como deve ser feita a entrega final

A entrega deve ser feita pelo Moodle na forma de um único arquivo zipado, contendo o código, os arquivos e a documentação.

Comentários Gerais:

- Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar;
- Clareza, identação e comentários no programa também vão valer pontos;
- **O trabalho é individual** (grupo de UM aluno);
- Trabalhos copiados terão nota zero;
- Trabalhos entregue em atraso não serão aceitos.

Dica 1: O autor de livros e professor Nívio Ziviani compôs algumas dicas de como deve ser feita uma boa implementação e documentação de um trabalho prático, que vale a pena ser consultada (link aqui).

Um olhar mais cuidadoso: O que são dicionários?

Um dicionário é uma estrutura de dados que armazena pares de chave-valor. Cada chave é única e é usada para acessar o valor associado a ela. Os dicionários são extremamente úteis para buscas rápidas, pois permitem acessar os valores diretamente através das chaves, em vez de iterar por todos os elementos.

Os dicionários são implementados de forma que a pesquisa, inserção e exclusão de pares de chave-valor sejam operações eficientes. Nos dicionários fornecidos pela maioria das linguagens de programação, como Python e C++, essas operações têm complexidade de tempo média $O(1)$ ou $O(\lg n)$, dependendo se foi implementado com um hash ou árvore balanceada, respectivamente.

Os dicionários são mutáveis, o que significa que você pode alterar suas entradas depois que eles são criados, adicionando, removendo ou modificando pares de chave-valor.

Aplicações de Dicionários:

- Mapeamento de dados: Dicionários são ideais para armazenar e manipular dados que podem ser mapeados de maneira clara, como configurações, dados de usuário, resultados de buscas, etc.
- Contagem de frequências: Podem ser usados para contar a frequência de elementos em uma lista.
- Armazenamento de dados em cache: Permitem armazenar resultados de operações caras para acesso rápido subsequente.
- Representação de grafos: Dicionários podem ser usados para representar grafos, onde as chaves são nós e os valores são listas de nós adjacentes.

Essa flexibilidade e eficiência tornam os dicionários uma estrutura de dados muito poderosa e amplamente utilizada em diversas áreas da programação.

As operações básicas que um dicionário deve fornecer para o usuário incluem:

- Criação: Permitir a criação de um dicionário vazio ou com pares de chave-valor iniciais.

- Inserção: Adicionar um novo par chave-valor ao dicionário.
- Atualização: Modificar o valor associado a uma chave existente.
- Acesso: Recuperar o valor associado a uma chave específica.
- Remoção: Remover um par chave-valor do dicionário usando a chave.
- Verificação de existência: Verificar se uma chave existe no dicionário.
- Iteração: Percorrer os pares de chave-valor do dicionário.
- Tamanho: Obter o número de pares chave-valor no dicionário.
- Limpeza: Remover todos os pares chave-valor do dicionário.

Dica 2: Esse trabalho pode ser dividido em duas grandes partes. A primeira delas lida com a parte de interface do programa, manipulação de arquivos e leitura de strings. A segunda parte lida com a implementação das estruturas de dados.

A fim de facilitar a interação entre essas duas partes, uma possibilidade de implementação seria implementar uma classe abstrata chamada `Dictionary` (em C++, por exemplo, essa classe poderia ser uma classe abstrata com funções virtuais puras ou apenas funções virtuais, depende de como você planeja fazer). Essa classe poderia ser utilizada para fazer a parte 1 se comunicar com a parte 2 de forma menos desacoplada. Assim, as 4 estruturas de dados seriam implementadas como classes-filha dessa classe abstrata e conteriam a implementação dos métodos abstratos da classe `Dictionary`.

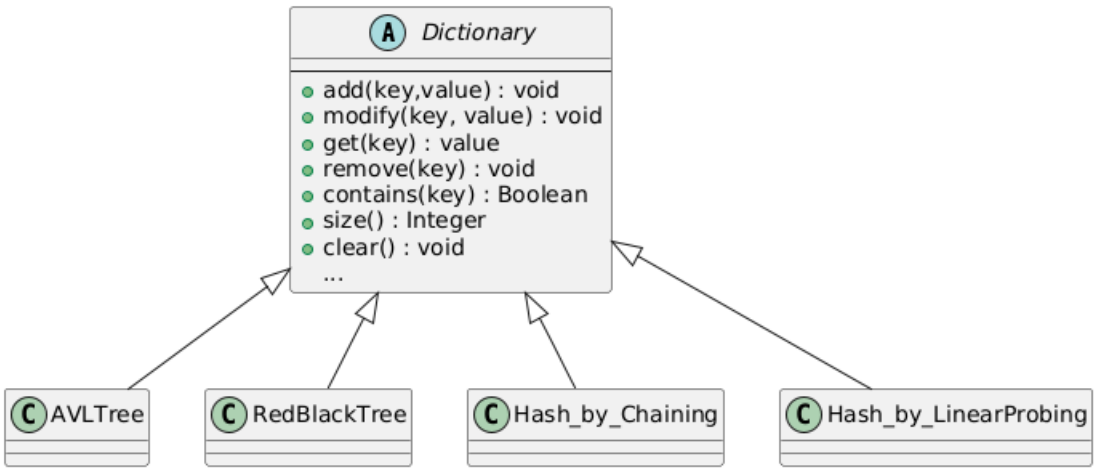

Figura 1: Uma possível organização das classes. Isso é apenas uma sugestão. Inclusive os nomes dos métodos abstratos que estão nessa figura é apenas uma sugestão. Alguns até estão faltando. Inclusive, usar essa organização é também apenas uma sugestão, nem sei se pode ser bom. Você deve avaliar.