Trabalho Prático 0: Frequência de Palavras

Algoritmos e Estruturas de Dados III – 2016/2
Entrega: 05/09/2016

1 Introdução

O objetivo desse trabalho é trabalhar estruturas vistas em *AEDS2* para relembrar conceitos de programação em *C*, conceitos de estruturas de dados e ordem de complexidade. A estrutura que cobraremos nesse trabalho é a implementação da árvore trie para contagem da frequência que palavras de um dado dicionário aparece em um texto. O nome *trie* vem da palavra "re**trie**val", pois essa estrutura pode encontrar uma palavra de um dicionário apenas com seu prefixo.

O trabalho consiste então das seguintes tarefas:

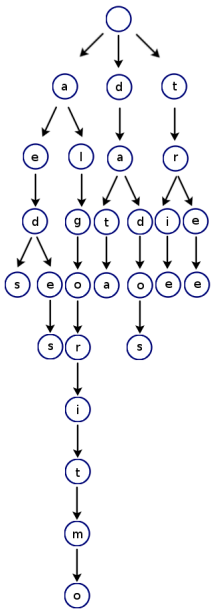
- Dado um conjunto de palavras que compõe o *dicionário*, construir a árvore *trie*;
- Dando um outro conjunto de palavras que compões o texto verificar se elas estão presentes na árvore;
- Contar a frequência que cada palavra que pertence ao dicionário está presente no texto, é necessário que a contagem dessa frequência seja interna à árvore, isto é, essa informação deve ser implementada na mesma estrutura da árvore.

2 Trie

A ideia principal da arvore trie é:

- Cada vértice representa uma letra da palavra;
- A raiz da árvore representa a palavra vazia;
- Cada vértice representa uma letra, onde o vértice de distância k da raiz da árvore representa um prefixo de tamanho k

Figura 1: exemplo de arvore trie



- Se um vértice v é pai de um outro vértice w, então o vértice w tem o prefixo associado ao v.

Na figura Figura 1 temos um exemplo de arvore formada pelas palavras *Aeds, Aedes, Algo, Algoritmo, dados, data, trie* e *tree*.

No nosso trabalho em especial, teremos um campo adicional informando a frequência que cada palavra do dicionário é encontrado no texto.

3 Entrada e saída

Entrada A entrada de cada execução do programa será composta por 4 linhas.

1. A primeira temos um valor inteiro N que informa o número de palavras do dicionário;

2. A segunda linha teremos N palavras que são todas as palavras do dicionário, pode assumir que todas as palavras fazem parte do alfabeto latino do padrão ISO que consiste em 26 letras a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, todas minusculas;

3. A terceira linhas teremos um número inteiro M que informa o número de palavras do texto;

4. A quarta linha teremos M palavras, cada uma pertencente ao texto, assim como nas palavras do dicionário, todas são pertencentes ao alfabeto latino do padrão ISO e todas minusculas

Exemplo de entrada

```
8
aeds aedes algo algoritmo dados data trie tree
24
a turma de aeds deve desenvolver um algoritmo
baseado na estrutura arvore trie mas como eles
sao alunos da ufmg implementar uma trie tree e
facil
```

A árvore trie representada por essa entrada é a mesma apresentada na figura Figura 1

Devemos notar que:

- O texto pode conter palavras que não fazem parte do dicionário;
- No texto podemos ter palavras que são prefixo de palavras do dicionário, mas não são iguais, por exemplo, o dicionário contém a palavra *carroceria* e no texto aparecer a palavra *carro*. vemos nesse caso que a palavra *carro* não pertence ao dicionário, logo ela não deve ser contabilizada;
- Nem todas as palavras do dicionário devem aparecer no texto.

Saída A saída do programa é uma única linha contendo o número de vezes que cada palavra do dicionário aparece no texto, na ordem apresentada pela entrada.

Exemplo de saída

```
1 0 0 1 0 0 2 1
```

4 O que deve ser entregue

Deverá ser submetido um arquivo *.zip* contendo somente uma pasta chamada `tp0` e dentro desta deverá ter: (i) Documentação e (ii) Implementação.

Documentação Poderá ter no máximo 10 páginas e deverá seguir tanto os critérios de avaliação discutidos na Seção 5.1, bem como as diretrizes sobre a elaboração de documentações disponibilizadas no *moodle*. Além desses requisitos básicos, a documentação do tp0 deverá ter:

1. Experimentos mostrando o tempo de execução e a memória alocada quando o número de palavras no dicionário e do texto aumentam.

2. Experimentos mostrando a reação do algoritmo quando a frequência das palavras do dicionário presentes no texto aumenta.d

Implementação Código fonte do seu TP (*.c* e *.h*)

Makefile Inclua um *makefile* na submissão que permita compilar o trabalho

5 Avaliação

Eis uma lista **não exaustiva** dos critérios de avaliação que serão utilizados.

5.1 Documentação

Introdução Inclua uma breve explicação do problema que está sendo resolvido no seu trabalho e um resumo da sua solução.

Solução do Problema Você deve descrever a solução do problema de maneira clara e precisa, detalhando e justificando os algoritmos e estruturas de dados utilizados. Para tal, artifícios como pseudo-códigos, exemplos ou diagramas podem ser úteis. Note que documentar uma solução não é o mesmo que documentar seu código. **Não** é necessário incluir trechos de código em sua documentação nem mostrar detalhes de sua implementação, exceto quando estes influenciem o seu algoritmo principal, o que se torna interessante.

Análise de Complexidade Inclua uma análise de complexidade de tempo e espaço dos principais algoritmos e estrutura de dados utilizados. Cada complexidade apresentada deverá ser devidamente **justificada** para que seja aceita.

Avaliação Experimental Sua documentação deve incluir os resultados de experimentos que avaliem o tempo de execução de seu código em função de características da entrada. Cabe a você gerar entradas para esses experimentos. Por exemplo: se esse trabalho fosse sobre ordenação, seria interessante mostrar como o tempo de execução de cada algoritmo varia quando o número de items a serem ordenados aumenta. Para tal, um gráfico mostrando o tempo de execução em função do tamanho da entrada pode ser interessante. Você também deve interpretar os resultados obtidos. Comente sobre cada gráfico ou tabela que você apresentar mostrando o que é possível concluir a partir dele.

5.2 Implementação

Linguagem & Ambiente O seu programa deverá ser implementado na linguagem **C** e poderá fazer uso de funções da biblioteca padrão da linguagem. Trabalhos que utilizem qualquer outra linguagem de programação e/ou que façam uso de outras bibliotecas que não a padrão serão zerados. Além disso, certifique-se que seu código compile e funcione corretamente nas máquinas **LINUX** dos laboratórios do DCC.

Casos de teste A sua implementação passará por um processo de correção automatizado, portanto, o formato da saída do seu programa deve ser idêntico aquele descrito nessa especificação. Saídas com qualquer divergência serão consideradas erradas, mesmo que as divergências sejam *whitespaces*. e.g. espaços, *tabs*, quebras de linha, etc. Para auxilia-lo na depuração do seu código, será fornecido um pequeno, **não-exaustivo**, conjunto de entradas e suas respectivas saídas. É seu dever certificar-se que seu código funciona corretamente para qualquer entrada válida.

Alocação Dinâmica Algoritmos e estruturas de dados deverão fazer uso de memória alocada dinamicamente (`malloc()` ou `calloc()`). Certifique-se que seu programa utiliza essas regiões de memória corretamente, pois os monitores penalizarão implementações que realizam *out-of-bounds access* e que tenham vazamento de memória (não desalocar memoria dinâmica).
A alocação dinâmica deverá fazer uso das funções malloc() ou calloc() da biblioteca padrão C, bem como liberar tudo o que for alocado utilizando free(), para gerenciar o uso da memória. **DICA:** Utilize `valgrind` antes de submeter o seu TP.

Qualidade do código Seu código também será avaliado no quesito de legibilidade, dando atenção, porém não limitando-se, aos seguintes items: (i) **INDENTAÇÃO**; (ii) nomes de variável e função descritivos e claros; (iii) Modularização adequada; (iv) Comentários dentro de funções, explicando o que certos trechos mais complicados fazem; (v) Comentários fora de funções, explicando, em alto-nível, o que as funções mais importantes fazem; (vi) funções concisas que desempenham somente uma tarefa; (vii) **Proibido uso de variáveis globais**.

Atrasos Trabalhos poderão ser entregues após o prazo estabelecido, porém sujeitos a uma penalização regida pela seguinte fórmula:

$$\Delta_p = \frac{2^{d-1}}{0.32}\,\%$$

Por exemplo, se a nota dada pelo corretor for 70 e você entregou o TP com 4 dias corridos de atraso, sua penalização será de $\Delta_p = 25\%$ e, portanto, a sua nota final será: $N_f = 70 \cdot (1 - \Delta_p) = 52.2$. Note que a penalização é exponencial e 6 dias de atraso resultam em uma penalização de 100%.

6 Consideração Final

A implementação da estrutura de árvore *trie* é bem comum e simples. Assim como em todos os trabalhos dessa disciplina é estritamente proibida a copia parcial ou integra de códigos, seja da internet ou de colegas. Utilizaremos o algoritmo *MOSS* para detecção de plágio em trabalhos, seja honesto. Você não aprende nada copiando código de terceiros nem pedindo a outra pessoa que faça o trabalho por você. Se a cópia for detectada, sua nota será zerada e os professores serão informados para que as devidas providências sejam tomadas.

HAVE FUN!!!