Universidade Federal do Espírito Santo

Departamento de Computação e Eletrônica - CEUNES
Estrutura de Dados I
Prof. Oberlan Romão

Exercício Programa 3

Verificador Ortográfico

O objetivo desse EP é comparar a utilização de *árvores binárias de busca* (balanceada vs. não balanceada). Para isso, você deve completar/implementar um programa que faça a leitura de um dicionário de palavras em português e um texto. Seu programa deve "marcar" as palavras escritas incorretamente, ou seja, que não estão no dicionário fornecido. Em seguida, ele deve listar essas palavras (sem repetí-las) e exibir algumas sugestões (como ocorre nos editores de texto modernos).

As palavras do dicionário devem ser carregadas e armazenas em uma árvore binária de busca (convencional ou AVL, dependendo do argumento passado ao programa). Já o conjunto de palavras escritas incorretamente, podem ficar em qualquer estrutura de sua escolha, mas a exibição delas deve obedecer a ordem de ocorrência, ou seja, se no texto a palavra ABC aparece antes da palavra XYZ, na exibição das palavras incorretas ABC deve vir antes de XYZ. Além disso, na listagem de palavras incorretas não deve haver repetição. A figura abaixo exibe um exemplo de execução do programa. Note que a ordem das palavras incorretas é a mesma do texto e que a palavra "dicionario" (sem acento) aparece apenas uma vez na listagem de palavras incorretas.

```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
❯ ./EP3 -d Dicionario2.txt -t EP3.txt
O objetivo desse EP é conparar a utilisação de arvores binarias de busca (balanceada vs. não balanceada). Para isso, voce deve completar/implement
ar um programa que faça a leitura de um dicionario de palavra em português e um texto. Seu programa deve "marcar" as palavras escritas incoretamen
te, ou seja, que não estão no dicionario fornecido. Em seguida, ele deve listar essas palavras (sem repetí-las) e ezibir algumas sugestões (como o
corre nos editores de texto modernos).


Palavra(s) incorreta(s) e sugestão(ões)
---------------------------------------
EP: AC, BC, EUA, Ema, Eva, IM, TV, a, ai, ah, ao, ar, as, da, d, de, do, e, em, eu, ex, ia, ih, ir, la, li, lo, ma, me, mo, na, no, o, oi, oh, os,
 ou, ri, se, si, te, ti, tu, uh, ui, um, vi, xi, à, é, ô,
conparar: compara, comparai, comparam, comparara, comparar, comparas, comprar, conjurar, consagrar, conspirar, contara, contaras, contaram, contat
ar, cooperar,
utilisação: utilização,
binarias: abanarias, afinarias, atinarias, babarias, bailarias, baixarias, bancarias, banharias, banirias, beijarias, beirarias, bipartias, bicari
a, bicaras, bicariam, bicarias, bigamias, binárias, bipartas, bizarras, blindarias, boiarias, bolarias, botarias, brigarias, brincarias, brindaria
s, bufarias, buzinarias, citarias, danarias, dignarias, ditarias, fiarias, ficarias, filarias, fincarias, findarias, fintarias, fitarias, fixarias
, gingarias, girarias, lidarias, ligarias, limarias, lixarias, miarias, mijarias, mimarias, minaras, minaria, minarias, minariam, minorias, mirari
as, mixarias, nanarias, ninaras, ninariam, ninaria, ninarias, ninharias, opinarias, ornarias, penarias, piarias, picarias, pifarias, pingarias, pi
ntarias, pisarias, reinarias, rifarias, rimarias, sanarias, tinirias, tirarias, uivarias, urinarias, vincarias, vingarias, virarias, visarias, xin
garias,
vs: AC, BC, Eva, IM, TV, a, ai, ah, ais, ao, aos, ar, as, asa, ave, aves, avos, bis, cus, da, d, das, de, do, dos, e, eis, els, em, eu, eva, ex, i
a, ias, ih, ir, jus, la, las, li, lo, los, ma, mas, me, mo, mos, na, nas, no, nos, o, oi, oh, ois, os, ou, ovo, ovos, pus, ri, ris, se, si, te, ti
, tu, uh, ui, um, uns, usa, use, uso, uvas, uva, vai, vais, vaso, vem, vens, ver, vez, vi, via, vias, viis, vii, vil, vim, vir, visa, vis, vise, v
iso, viu, voa, voas, voe, voes, vos, voz, vou, vá, vás, vãs, vã, vê, vês, vós, xi, à, às, é, ès, ô,
voce: Rose, adoce, alce, boca, bode, bole, bote, boxe, cace, coca, cocem, coce, cocei, coces, coco, coe, coice, cole, come, cone, core, cose, cote
, coze, doce, doca, doe, dome, dope, dose, dote, doze, evoca, evoco, face, foge, foca, foco, foice, fole, fome, fone, force, fuce, gole, gore, goz
e, hoje, ice, lace, loca, loco, lote, moe, mofe, moge, mole, more, mote, move, nome, note, nove, oca, oco, ode, orce, ore, pode, pose, pote, roe,
roca, roce, rocei, rocem, roces, rode, roge, role, sobe, soca, soco, soe, sole, some, sove, tece, toca, toco, tome, tope, torce, tose, troce, vaca
, vaie, vale, vare, vaze, vede, vele, vence, vete, vexe, vice, vicie, vide, viole, vire, vise, vive, voar, voam, voa, voai, voas, você, voei, voe,
 vodca, voem, voes, voga, volte, volve, vota, voou, vos, votei, vote, votem, votes, voz, voto, vou, voá, vozes,
completar/implementar:
dicionario: adicionaria, adicionarmo, dicionário,
incoretamente: incorretamente, indiretamente,
repetí-las:
ezibir: coibir, elixir, emitir, erigir, exibi, exibia, exibira, exibir, exibis, exibiu, exigir, eximir, inibir,
---------------------------------------
```

Sugestões de palavras

Como você deve ter observado na figura anterior, para cada palavra escrita errada, é exibida uma lista de sugestões (algo comum em muitos editores de textos). Para o EP, usaremos o algoritmo de Edição de Distância (Edit distance) para definir as sugestões. Esse algoritmo explora uma técnica chamada Programação Dinâmica (vista com detalhes na disciplina de Projeto e Análise de Algoritmos). Esse algoritmo tem complexidade $O(n \times m)$, onde n e m são os tamanhos dos strings. No arquivo `Util.c` você encontra uma implementação do algoritmo.

Em resumo, a função `int distanciaEdicao(char *a, char *b)` recebe duas `strings` e retorna a "distância" entre elas, ou seja, quantos mudanças (inserções, substituições e remoções) são necessárias para que uma `string` fique igual a outra. Por exemplo, `distanciaEdicao("uma", "uva")` retornaria 1 (basta trocarmos o caractere `"m"` por `"v"`). Dessa forma, quanto menor o valor retornado mais "próximo" duas `strings` estão. Para o EP, ao listar as sugestões, você deve definir um limite para que uma palavra seja considerada "sugestão" de uma palavra errada. Note que quanto maior esse limite, maior será o número de palavras sugeridas.

Perceba que para cada palavra errada, seu programa deve percorrer toda a árvore procurando por sugestões. Pense em como limitar o número de chamada da função `distanciaEdicao` (chamando-a apenas quando fizer sentido). Você também pode dar maior prioridade as palavras que tenham uma distância menor a palavra errada e limitar o número de palavras sugeridas (por exemplo, exibir, no máximo, as 5 melhores sugestões).

Análise de desempenho

Você deve fazer uma análise empírica de desempenho do seu programa, comparando a utilização de uma árvore binária de busca sem balanceamento e a árvore AVL. Utilize diferentes arquivos de textos com um número variado de palavras erradas. Faça um relatório técnico com a metodologia adotada para a realização dos experimentos e análise e discussão dos resultados. Faça tabelas e/ou gráficos para te ajudar na análise.
Dicionários. Você pode utilizar os dicionários (em português e inglês) disponibilizados no arquivo `EP3.zip` ou procurar por outros na internet. Sinta-se livre para manipular os dados dos dicionários, mas relate o que e como foi feito e o porquê dessa modificação no relatório. Caso utilize outros dicionários, envie-os na entrega do EP.
Textos. Teste seu programa com textos de tamanhos variados. Envie também os arquivos usados nos experimentos. Uma boa fonte de arquivos para testar o seu programa é o Projeto Gutenberg.

O que entregar

Você deve entregar, pelo **AVA**, um arquivo compactado contendo todos os códigos da sua implementação, uma arquivo `Makefile` (basta completar o que está no arquivo `EP3.zip`) e um relatório (no formato `.pdf`). Envie também os textos e dicionário utilizados.

Data de entrega: até às 6h do dia 10/05/2021.

Observações:

1. Sinta-se livre para usar outros dicionários ou modificar o disponibilizado, mas deixe claro o que foi feito/utilizado no relatório;
2. Você também pode (e deve) exibir outros dados no programa para te auxiliar no relatório. Por exemplo, o tempo para construir a árvore, o tempo para listar as palavras com as sugestões, a altura da árvore gerada, etc. Seja criativo;

3. Você deve adicionar outros arquivos ao projeto. Pense em como organizá-lo em diferentes TADs, cada um com seus arquivos `.h` e `.c`;
4. Nos nós das árvores ou de outras estruturas que contiverem `strings`, você deve utilizar alocação dinâmica (`char *`). Algo assim:

```
typedef struct no {
    char *<nomeCampo>;
    // Outros campos
} No;
```

   Códigos que não respeitem essa restrição (utilizando alocação estática) receberão 70% da nota da implementação;
5. Preferencialmente, use o Linux (ou o CS50 IDE) para a implementação. No Windows, você pode usar o Code::Blocks para auxiliar na compilação do projeto;
6. Seu código deve ser compilado com as flags:

```
-O0 -std=c11 -Wall -Werror -Wextra -Wno-sign-compare -Wno-unused-parameter -Wno-unused-variable -Wshadow
```

7. Códigos com erros de sintaxe (que não compilem) receberão nota 0;
8. **A compilação será feita usando o `Makefile`, se você não enviá-lo ou ele estiver incompleto, sua nota na implementação será 0;**
9. Código com vazamento de memória, valerá 70% da nota da implementação;
10. Código que não segue o Guia de Estilo, valerá 90% da nota da implementação;
11. Em caso de plágio, será atribuído 0 a todos os envolvidos.

Critérios de avaliação

A nota geral do EP será da seguinte forma:

- **Implementação**: 5.0 pontos;
  - Organização: 1.0 ponto (separação dos códigos em `.h` e `.c`);
  - Implementação da Árvore Binária de Busca (não balanceada): 1.5 pontos;
  - Implementação da Árvore AVL: 1.5 pontos;
  - Outras estruturas de dados: 1.0 ponto;
- **Relatório técnico**: 5.0 pontos (caprichem). O relatório deve conter, pelo menos, os seguintes pontos:
  - Metodologia adotada para os experimentos;
  - Análise e discussão dos experimentos feitos (faça a análise para diferentes textos com número variado de palavras corretas e incorretas);
  - Tabela e/ou gráficos (preferencialmente, feitos com o Matplotlib) comparando os resultados.