UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

CURSO DE BACHARELADO EM CIÊNCIAS DA COMPUTAÇÃO

Jessé Pires Barbato Rocha

Jhonatan Guilherme de Oliveira Cunha

CORRETOR ORTOGRÁFICO

CAMPO MOURÃO

2020

1. Relatório

Assim como no trabalho anterior, no intuito de nivelar o conhecimento a respeito do problema proposto, discutimos a ideia geral do trabalho. Após isso, começamos a pensar no que seria necessário para construir o corretor e, então, dividir as tarefas.

Além da própria *Trie*, que foi usada para procurar as palavras no dicionário e também para garantir que não haveriam palavras repetidas retornadas pelas regras, também notamos a necessidade de implementar uma Lista. Seu objetivo era armazenar as palavras sugeridas tanto na aplicação das regras quanto para exibi-las para o usuário.

No intuito facilitar a busca das palavras pelas funções de busca aproximada, também notamos a necessidade de implementar uma TAD Pilha.

Também deixamos organizados em uma TAD, os dados que deveriam ser retornados pelo programa para análise. Isso auxiliou muito para a organização do código e também facilitou no entendimento deste.

Vale ressaltar que muitos detalhes da implementação como, por exemplo, alterações nas TAD's e, até mesmo a implementação da TAD para análise, só foram notados durante o desenvolvimento do programa.

Feito isso, começamos a dividir o trabalho. A implementação da *Trie* usada foi a feita durante as ADNP's, sendo necessária apenas a mudança da quantidade de filhos que cada nó poderia ter. Jhonatan ficou responsável pela implementação da Pilha e pelas funções descritas nos itens 2 e 4 e 7 da especificação do trabalho.

Já Jessé, ficou responsável pela implementação da Lista e pelas funções descritas nos itens 3 e 5. Também foi o responsável pela execução dos testes e coleta dos dados.

A função descrita no item 6 foi feita em conjunto, uma vez que precisávamos inventar nossas próprias regras. A primeira que notamos ser uma regra que retornaria boas sugestões foi de aplicar sobre a regra 2, a regra 1. Assim, caso uma palavra tenha mais do que 5 caracteres, aplicamos a regra 2 nos prefixos com $n-2$ e $n-3$, onde $n > 5$. Assim, caso o prefixo ainda contenha algum erro, conseguimos contorná-lo, usando a regra 1.

Outra regra utilizada foi uma alteração da regra 1. Nesta regra, caso a palavra tenha mais do que 4 caracteres, usamos utilizamos um padrão com dois asteriscos que vão cobrindo todas as posições da palavra. Assim é possível cobrir praticamente todas as possibilidades de palavras que seriam corretas para um determinado caso.

Por fim, desenvolvemos mais uma regra que tinha por objetivo tratar de letras repetidas/incorretas numa dada palavra. A ideia desta regra é ir retirando uma letra por vez da

palavra e realizar uma busca no dicionário com este novo prefixo. Assim, caso o erro da palavra seja uma letra repetida ou incorreta, é possível eliminá-la e encontrar uma chave válida no dicionário.

Refatorações foram feitas durante todo o desenvolvimento de acordo com a necessidade.

Vale também ressaltar a importância de duas ferramentas usadas durante o desenvolvimento do programa: o Valgrind, que auxiliou na correção de erros do código, prevenindo, principalmente, o vazamento de memória alocada (o *memory leak*); e o Git e Github que auxiliou no desenvolvimento de maneira geral.

2. Análise

Como solicitado na especificação do trabalho, foram coletados, (da execução do programa com cada arquivo de teste), o total de palavras, as palavras incorretas detectadas, o número médio de sugestões por palavra incorreta e, por fim, o tempo de execução em segundos. Tais dados estão dispostos na Tabela 1 abaixo.

Pode-se notar que, dado o volume de dados com que o programa precisava lidar, o tempo de execução foi extremamente baixo. Outrossim, a quantidade média de sugestões proporcionou boas opções de palavras corretas.

	Total de Palavras	**Palavras Incorretas**	**Número Médio de Sugestões por Palavra Incorreta**	**Tempo de Execução (s)**
casmurro1.txt	826	53	35.83	0.26
casmurro2.txt	530	43	17.63	0.25
casmurro3.txt	452	34	23.06	0.25
casmurro4.txt	797	44	22.68	0.25
memposthumas1.txt	697	58	25.57	0.25
memposthumas2.txt	529	42	27.29	0.25
memposthumas3.txt	563	43	51.30	0.26
casmurro_all.txt	2605	176	25.24	0.29
memposthumas_all.txt	1789	143	33.81	0.28
all.txt	4394	316	29.34	0.32

Tabela 1: Alguns resultados da execução do corretor ortográfico sobre os testes propostos

3. Conclusão

Pudemos, com esse trabalho, notar mais uma vez a importância que o conhecimento das estruturas de dados tem para a computação. Com a utilização adequada destas, a implementação do programa proposto foi muito mais eficiente, organizada e facilitada. No caso específico da *Trie*, foi possível notar a eficiência desta no armazenamento e busca por palavras.