RELATÓRIO ESTRUTURA DE DADOS

Luiz Eduardo Rodrigues Correia, Matheus de Araújo Correia Lima Melo,
Thiago Fernandes Bonfim Sousa.

1- Introdução

O spell checker - que em português significa “verificador de palavras” - tem como objetivo alertar o usuário sobre a presença de palavras com erros gramaticais em um texto qualquer. Ele realiza testes com as palavras de um texto, comparando-as com um dicionário utilizado na sua implementação. Presume-se que o dicionário esteja correto e atualizado, o que possibilita a boa execução do corretor ortográfico.

Para a implementação do spell checker, neste trabalho, será utilizado a tabela hash. Essa estrutura de dados torna a busca, comparação das palavras do texto e a construção do dicionário mais eficiente, uma vez que para uma quantidade limitada de entradas com um universo muito maior de possíveis chaves - como é o caso de strings em um texto - operações comuns para dicionários tais como busca e inserção tem um custo constante O(1) para um caso médio. O método consiste em utilizar funções de hash que calculam o índice para palavra a ser verificada ou inserida no vetor que armazena o dicionário.

No entanto, a tabela hash possibilita o acontecimento de colisões, eventos em que entradas distintas produzem o mesmo valor de hash, ou seja, a mesma posição no dicionário, de forma que o uso de boas técnicas de tratamento de colisões determinam a eficiência da ferramenta implementada. Assim, este trabalho busca apresentar soluções e decisões que garantam um bom funcionamento para ferramenta proposta, bem como mostrar os resultados obtidos.

2- Metodologia

Após escolhida uma função de hash apropriada para strings, bem como o método para tratamento de colisões - chaining que consiste em transformar cada elemento do array em um apontador para elementos com a mesma chave assim fazendo de cada elemento do array uma espécie de lista simplesmente encadeada, que ocasiona vantagens como o número de chaves armazenadas poder ser maior que dimensão do array e também se houver várias colisões não afetará a tabela só tornará mais lento o acesso a lista associada, a única desvantagem percebida foi a memória adicional para os apontadores nas listas ligadas.

O desempenho do Spell Checker foi analisado, tomando como parâmetros determinísticos o tempo de execução do programa e o número de buckets vazios no dicionário. Assim, submetido a variações do número de buckets disponíveis na tabela e da quantidade de palavras fornecidas para análise foi verificada melhora no desempenho mediante alguns parâmetros.

Para a ferramenta de análise de texto desenvolvida, as operações de inserção e procura no dicionário pesam na qualidade do produto, uma vez que o dicionário precisa ser criado e as palavras testadas contra ele. Em listas duplamente encadeadas, operações de inserção e remoção podem ocorrer com custo constante ou linear no pior caso, no entanto, uma vez que o dicionário, para esse sistema, não precisa ser alterado depois de criado, uma operação de remoção é desnecessária. Assim, as colisões foram tratadas de forma que cada posição na tabela(bucket) é uma lista simplesmente encadeada(economiza espaço de armazenamento),e elementos que coincidam no seu valor de hash são adicionados nessa lista.

```
unsigned int hash(char * blau)
{
  unsigned int hashinho = 0;
  int i;

  for(i=0; blau[i]!='\0'; i++){
    hashinho += ((hashinho*251) + blau[i]);
  }

  return hashinho%TOTAL;
}
```

Figura 1 - Tabela Hash.

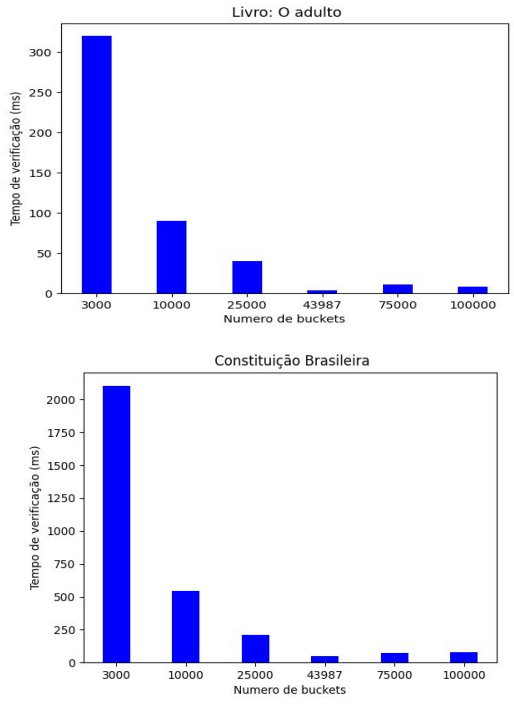
Como forma de garantir uma análise correta do programa, foram utilizados para a verificação a Constituição brasileira e o conto em formato de livro O Adulto, de Gillian Flynn,

com 215590 e 13461 palavras(o número de palavras foi calculado através do Spell Checker), respectivamente, vale ressaltar que devido o limitação do dicionário houve um número de dos textos que não foram encontradas 6735 do conto O adulto e 114563 da Constituição. Antes da análise, os arquivos foram devidamente tratados através de outra aplicação para garantir a boa execução do verificador gramatical, garantindo compatibilidade com o formato de texto do dicionário.

3- Resultados

É possível observar no gráfico a lentidão na execução do corretor ao utilizar apenas 3.000 buckets, um dos motivos para essa lentidão, foi a estratégia utilizada ao calcular o hash. Após Multiplicarmos o caractere(em ascii) pelo número 251, somamos ele com as letras que o sucedem e depois calcularmos o mod deste valor com o total de buckets(como mostrado na figura 1). Essa função de Hash não é ideal, caso o número de buckets colocado não seja um número primo e afastado de uma potência de 2. Caso essa função seja utilizada e o número de buckets não atenda os requisitos anteriores, o desempenho do programa tem uma otimização aparente, devido à disponibilização de mais buckets e não por boa distribuição dos mesmos.

Figura 2: tempo x número de buckets.

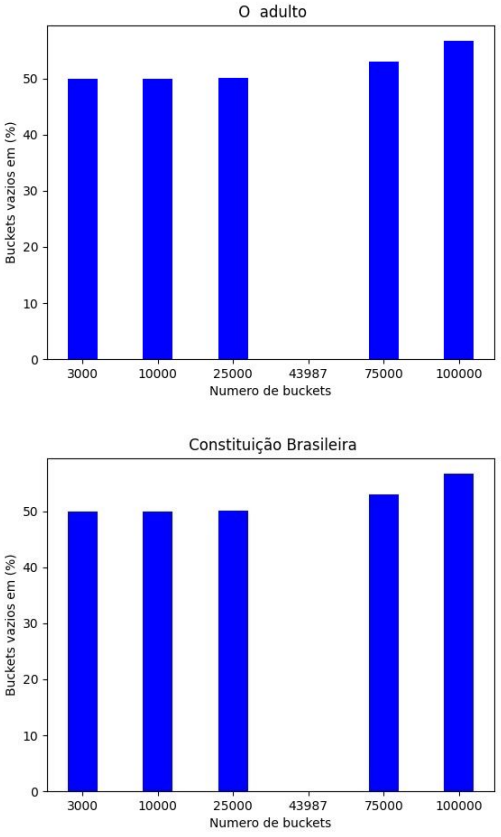


É possível observar a ineficiência na utilização do número de buckets, nos dois arquivos, independente do número de palavras. Além disso, pode-se observar que mesmo não tendo uma boa distribuição de buckets, o programa continua executando toda a verificação em um tempo relativamente baixo(Figura 2).

4- Conclusão

A melhora que acontece de forma abrupta na distribuição, se dá ao fato de termos encontrado um número primo “ideal” de buckets e consequente a melhor distribuição.

Figura 3: Porcentagem de buckets vazios.



Observa-se que embora o número de palavras do livro O adulto seja muito inferior em relação a quantidade de palavras do texto da Constituição, o desempenho e utilização de buckets é praticamente o mesmo ao utilizar o número 43987, que usamos como “número ideal”, por atender as características que o número de buckets precisa ter para a boa utilização do método da divisão(função de hash).

REFERÊNCIAS

Slides disponibilizados pelo professor.

CORMEN. Thomas. Algoritmos: Teoria e prática.PDF