Equipe: Davi Sousa Soares
Victor Kauan da Silva Miranda
João Pedro Saleh de Sousa

Professor: Raimundo Santos Moura

NOVEMBRO
2024

Conteúdo

1 Introdução

Uma tabela de símbolos é uma estrutura de dados abstrata que se resume em uma lista de elementos cada um com uma chave e um valor associado a mesma. Essa lista de elementos pode ser organiza como uma lista encadeada, um array ordenado para que seja possível realizar uma busca binária na tabela ou usando uma função criptografia chamada hash para associar um elemento a um índice de um array de tamanho variado, caso dois elementos recebam o mesmo índice será organizado como uma lista encadeada

1.1 Objetivos

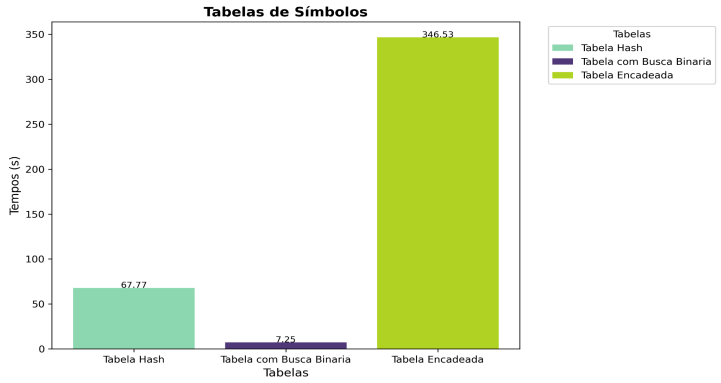
A primeira parte desse trabalho se concentra em comparar a velocidade de inserção de uma grande quantidade de elementos(cerca de 2 milhões de palavras) nas 3 variações da tabela de símbolos.
A segunda parte foca na tabela Hash e como ele se comporta com diferentes tamanhos do array.

1.2 Especificações

Código feito em python usando as bibliotecas “matplotlib” para a realização dos gráficos,“re” para o tratamento de palavras do arquivo,”os” para criação de pastas para guardar os gráficos e “json” para guardar os tempos
Testes realizados em um notebook com processador:12th Gen Intel(R) Core(TM) i7-12700H
e sistema operacional linux,usando em média 10% da cpu.

```
def clean_word(word:str) -> str:
    return
re.sub(r"(?<![a-zA-Z])'|'(?![a-zA-Z])|(?<![a-zA-Z0-9])[^\w']+|[^\w']+(?
![a-zA-Z0-9])", "", word)
```

Todas as palavras do arquivo passam por essa função para então serem inseridas



2 Parte 1

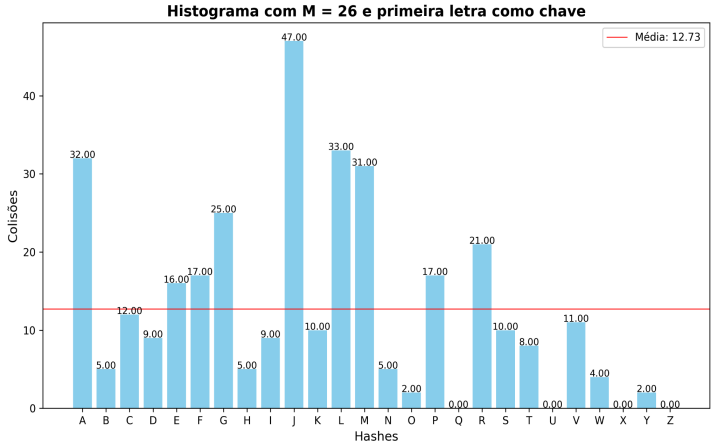
As estruturas foram separadas em 3 arquivos .py, TabelaDeSimbolosBuscaBinaria.py, TabelaDeSimbolosEncadeada.py e HashTable.py. A leitura e criação do gráfico do tempo é feito pelo arquivo menu.py e o arquivo leLeipzig junta todos essas peças para realizar a parte 1 do trabalho

2.1 Conclusões

- Tabela com busca Binária é a mais rápida na inserção: O(log n)
- Tabela encadeada é muito mais lenta em todas as operações: O(n)
- A tabela Hash tem uma velocidade boa mas lenta em comparação a tabela com busca binária: O(n/m),m sendo a capacidade

3 Parte 2

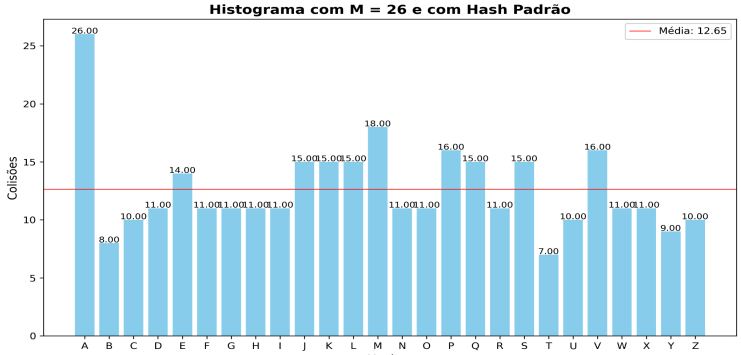
3.1 Questão 1




```
return (ord(key[0])-65) %self.M
```

função hash que usa a primeira letra do nome. O nome com maior colisão é o J e o de menor é Q,Z ,U e X

3.2 Questão 2

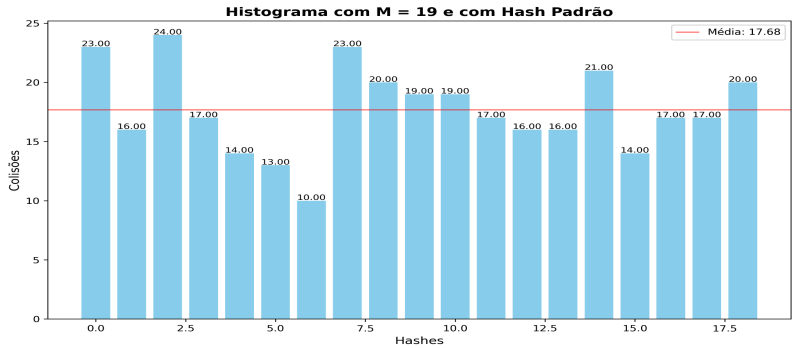


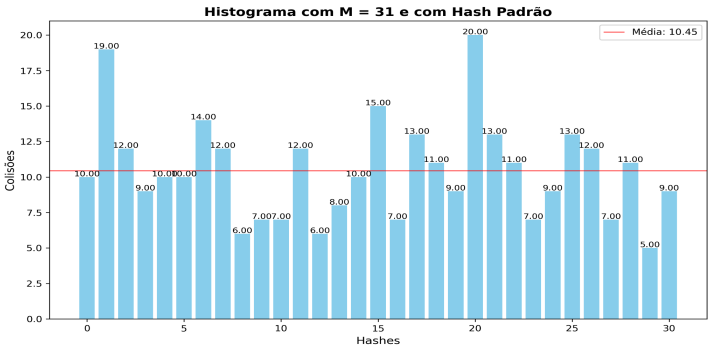

```
return hash(key) % self.M
```

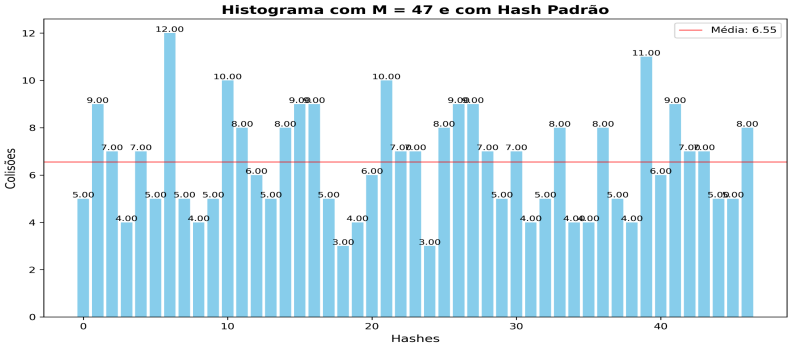
função hash usada.

Houve um espalhamento melhor que na questão 1 sendo as letra com mais colisões a letra K e a letra I e as com menos colisões as letras G e Q

3.3 Questão 3

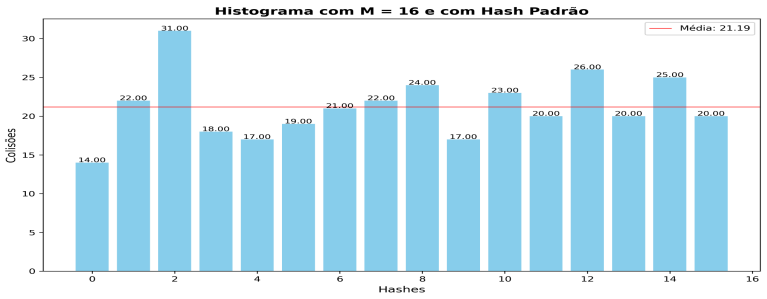


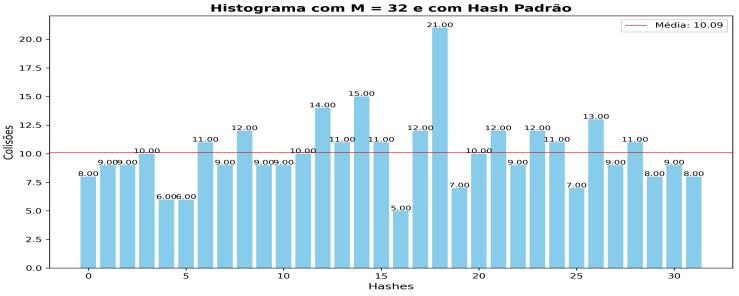


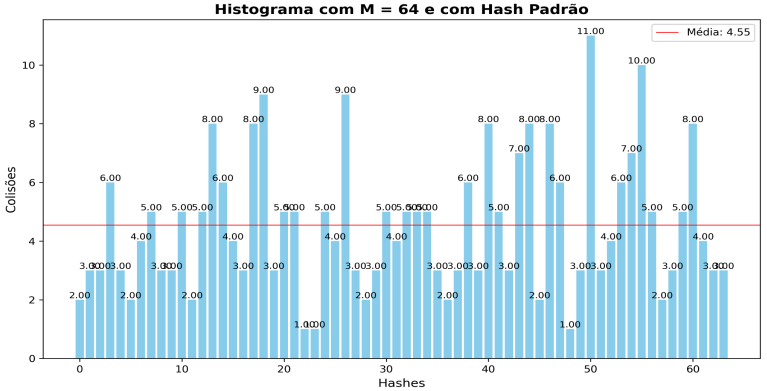


Podemos ver que nesse exemplo a quantidade de colisões se aproxima da linha N/M em M = 19, tornando esse M o que espalha melhor

3.4 Questão 4







Comparando os resultados com a questão 3 não é muito visível a diferença,além da sutil mudança em que muitos elementos estão passando muito da linha N/M e muitos estão longe de encostar nela