Relatório de Análise de Algoritmos de Ordenação

Alunos: Luis Otavio, Rafael Chicovis

Este relatório apresenta uma análise detalhada do desempenho de duas funções hash distintas: LoseLose e Polinomial, avaliadas com base em três critérios principais: número de colisões, tempo de inserção e tempo de busca. O objetivo é comparar a eficiência de ambas as funções hash em termos de como elas lidam com a distribuição de chaves e o tempo gasto em operações de inserção e busca. Os resultados fornecem uma visão abrangente sobre o comportamento de cada função hash, destacando suas vantagens e limitações em diferentes cenários de uso.

Foram implementadas as funções:

LoseLose:

```
public int getHashCode(String value) {
    var hash = 0;
    for (int i = 0; i < value.length(); i++) {
        hash += value.charAt(i);
    }
    return hash;
}
```

E uma função de hash simples que calcula o valor de hash somando os valores ASCII dos caracteres de uma chave (string). O valor resultante é usado para determinar a posição da chave na tabela hash, aplicando o operador módulo com o tamanho da tabela. Colisões ocorrem quando duas chaves diferentes geram o mesmo valor de hash e precisam ser tratadas, geralmente com técnicas como encadeamento ou sondagem. Embora seja fácil de implementar, o LoseLose pode ser ineficiente devido à alta probabilidade de colisões.

Polinomial:

```java
public int getHashCode(String value) {
    int hash = 0;
    int prime = 31;

    for (int i = 0; i < value.length(); i++) {
        hash = prime * hash + value.charAt(i);
    }

    return Math.abs(hash);
}
```

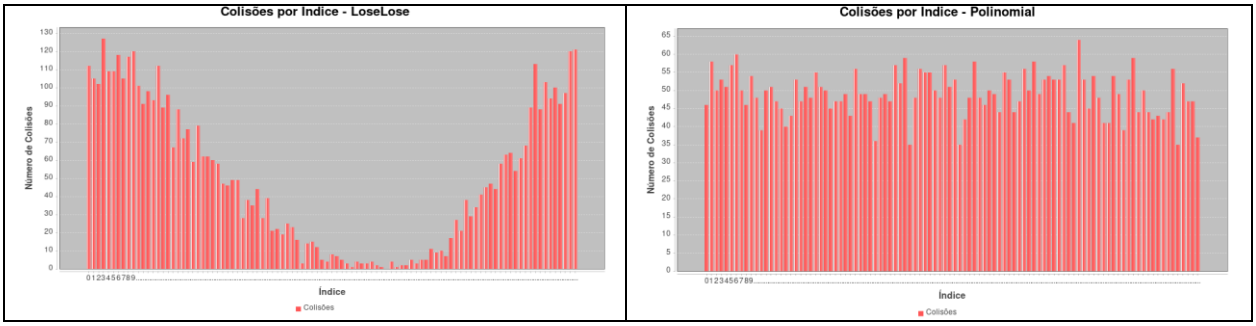
Essa função de hash usa um algoritmo polinomial, onde o valor do hash é calculado multiplicando o valor atual por 31 (um número primo) e somando o código ASCII de cada caractere da string. Isso melhora a distribuição das chaves e reduz colisões, sendo uma técnica eficiente e amplamente usada em funções de hash.

Alimentando tabelas hash com female_names.txt (5000 nomes)

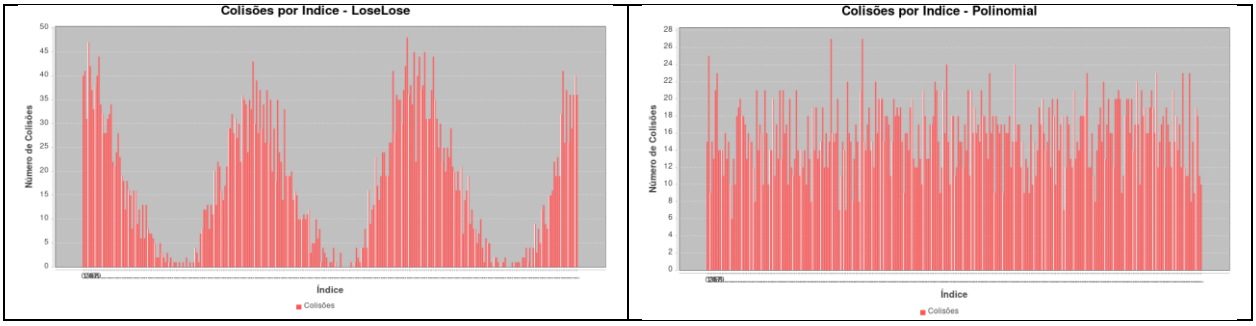
Número de índices	Polinomial	LoseLose
100	Colisões: 4901 Tempo total de inserção: 10.776 ms Tempo total de busca: 17.302 ms	Colisões: 4902 Tempo total de inserção: 11.271 ms Tempo total de busca: 21.920 ms
300	Colisões: 4701 Tempo total de inserção: 14.631 ms Tempo total de busca: 13.932 ms	Colisões: 4718 Tempo total de inserção: 21.662 ms Tempo total de busca: 16.982 ms
600	Colisões: 4401 Tempo total de inserção 11.925 ms Tempo total de busca 12.837 ms	Colisões: 4492 Tempo total de inserção: 12.600 ms Tempo total de busca: 15.696 ms

Gráfico de colisões por índice:

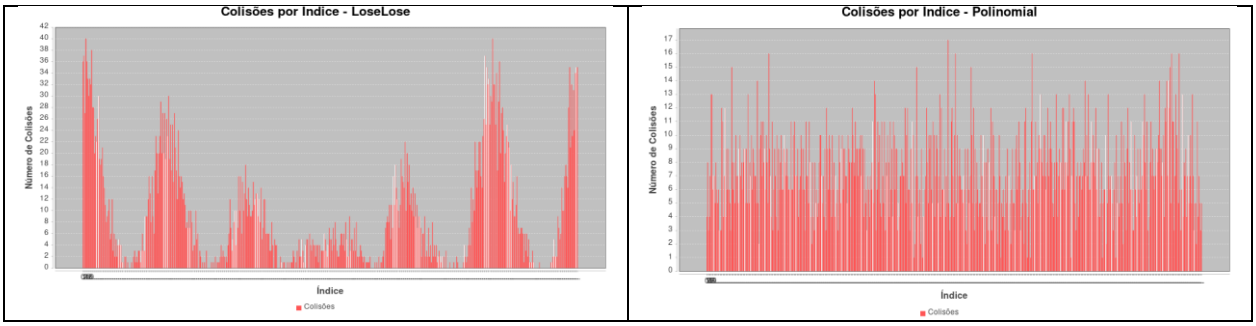
100 índices:



300 índices:



600 índices:



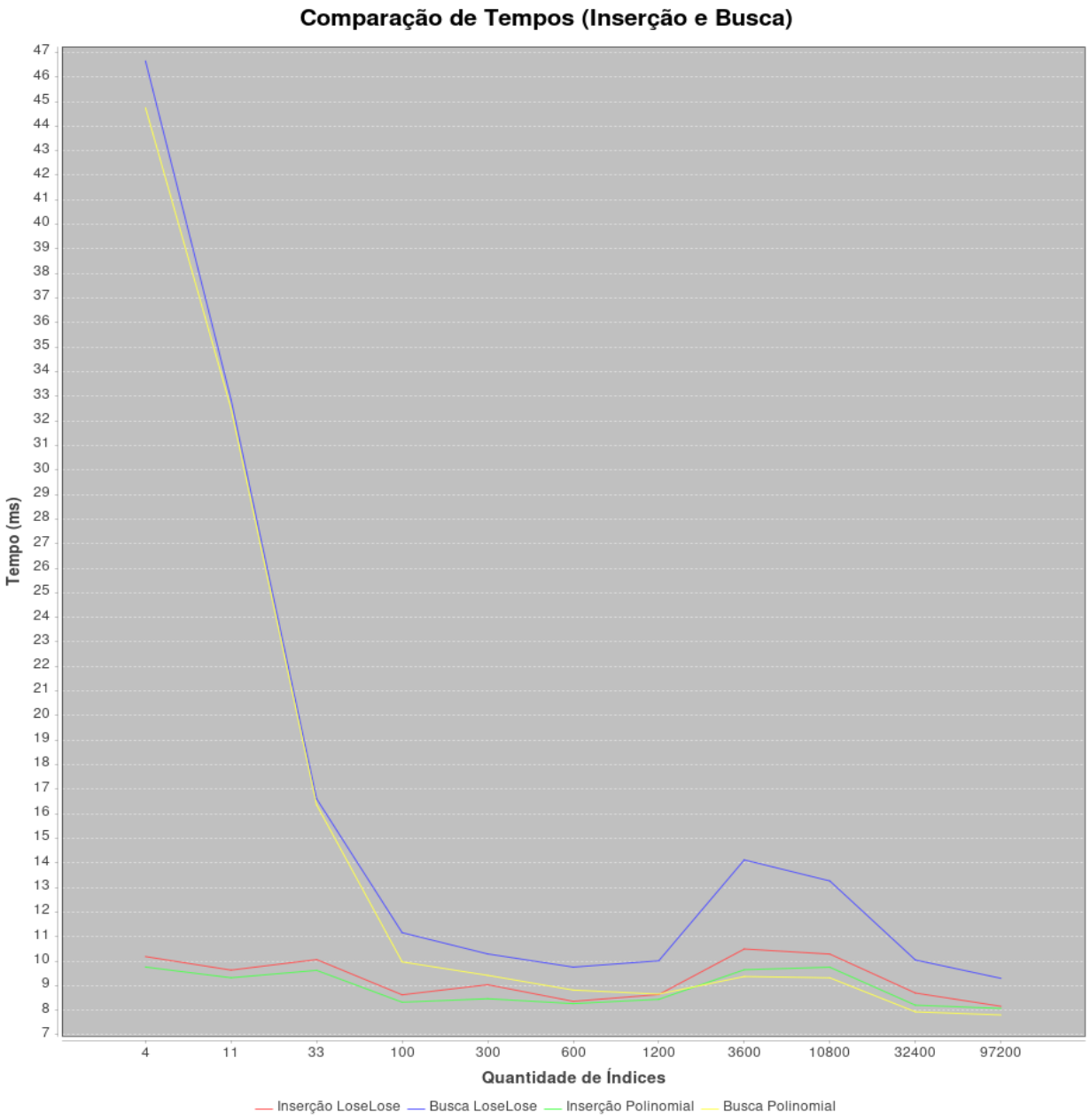
A função hash LoseLose exibe um comportamento irregular à medida que o número de índices aumenta, formando oscilações a cada 100 índices. Isso ocorre devido à distribuição ineficiente das chaves, resultando em mais índices vazios ou com poucas chaves. Já a função polinomial oferece uma distribuição mais uniforme das chaves, o que resulta em tempos de inserção e busca mais estáveis e eficientes, com menos índices vazios. Portanto, embora ambos os métodos sigam um padrão de aumento de tempo conforme os índices crescem, a função polinomial é mais eficiente e estável em comparação com a LoseLose.

Número de chaves por índice:

Obs.: foi reduzido a 50 índices para não deixar o documento muito extenso

Polinomial	LoseLose
Índice 0: 101 chave(s)	Índice 0: 121 chave(s)
Índice 1: 95 chave(s)	Índice 1: 112 chave(s)
Índice 2: 94 chave(s)	Índice 2: 107 chave(s)
Índice 3: 103 chave(s)	Índice 3: 130 chave(s)
Índice 4: 111 chave(s)	Índice 4: 115 chave(s)
Índice 5: 107 chave(s)	Índice 5: 114 chave(s)
Índice 6: 108 chave(s)	Índice 6: 123 chave(s)
Índice 7: 102 chave(s)	Índice 7: 111 chave(s)
Índice 8: 97 chave(s)	Índice 8: 121 chave(s)
Índice 9: 100 chave(s)	Índice 9: 123 chave(s)
Índice 10: 105 chave(s)	Índice 10: 102 chave(s)
Índice 11: 94 chave(s)	Índice 11: 93 chave(s)
Índice 12: 96 chave(s)	Índice 12: 104 chave(s)
Índice 13: 100 chave(s)	Índice 13: 96 chave(s)
Índice 14: 105 chave(s)	Índice 14: 116 chave(s)
Índice 15: 97 chave(s)	Índice 15: 93 chave(s)
Índice 16: 100 chave(s)	Índice 16: 103 chave(s)
Índice 17: 94 chave(s)	Índice 17: 72 chave(s)
Índice 18: 108 chave(s)	Índice 18: 95 chave(s)
Índice 19: 103 chave(s)	Índice 19: 79 chave(s)
Índice 20: 106 chave(s)	Índice 20: 90 chave(s)
Índice 21: 103 chave(s)	Índice 21: 70 chave(s)
Índice 22: 114 chave(s)	Índice 22: 91 chave(s)
Índice 23: 97 chave(s)	Índice 23: 71 chave(s)
Índice 24: 93 chave(s)	Índice 24: 81 chave(s)
Índice 25: 111 chave(s)	Índice 25: 89 chave(s)
Índice 26: 102 chave(s)	Índice 26: 81 chave(s)
Índice 27: 94 chave(s)	Índice 27: 87 chave(s)
Índice 28: 105 chave(s)	Índice 28: 77 chave(s)
Índice 29: 93 chave(s)	Índice 29: 85 chave(s)
Índice 30: 99 chave(s)	Índice 30: 92 chave(s)
Índice 31: 92 chave(s)	Índice 31: 75 chave(s)
Índice 32: 105 chave(s)	Índice 32: 87 chave(s)
Índice 33: 98 chave(s)	Índice 33: 81 chave(s)
Índice 34: 77 chave(s)	Índice 34: 104 chave(s)
Índice 35: 103 chave(s)	Índice 35: 93 chave(s)
Índice 36: 110 chave(s)	Índice 36: 105 chave(s)
Índice 37: 93 chave(s)	Índice 37: 77 chave(s)
Índice 38: 109 chave(s)	Índice 38: 85 chave(s)
Índice 39: 98 chave(s)	Índice 39: 89 chave(s)
Índice 40: 103 chave(s)	Índice 40: 116 chave(s)
Índice 41: 80 chave(s)	Índice 41: 138 chave(s)
Índice 42: 92 chave(s)	Índice 42: 106 chave(s)
Índice 43: 102 chave(s)	Índice 43: 108 chave(s)
Índice 44: 113 chave(s)	Índice 44: 110 chave(s)
Índice 45: 92 chave(s)	Índice 45: 117 chave(s)
Índice 46: 104 chave(s)	Índice 46: 105 chave(s)
Índice 47: 97 chave(s)	Índice 47: 104 chave(s)
Índice 48: 106 chave(s)	Índice 48: 126 chave(s)
Índice 49: 90 chave(s)	Índice 49: 131 chave(s)

Relação de tempo por quantidade de índices:



As funções hash LoseLose e Polinomial seguem um padrão semelhante em relação ao tempo de execução versus a quantidade de índices, formando linhas parecidas tanto na busca quanto na inserção. No entanto, apesar dessa semelhança, o método Polinomial se mostra mais eficiente, apresentando tempos menores para ambas as operações.

Comparação de Desempenho:

Colisões: A função Polinomial demonstrou um número significativamente menor de colisões em comparação à função LoseLose, o que leva a uma distribuição de chaves mais eficiente.

Tempo de Inserção e Busca: Embora ambas as funções sigam um padrão semelhante de aumento no tempo conforme o número de índices cresce, a função Polinomial é consistentemente mais rápida em termos de tempo de inserção e busca.

Distribuição das Chaves: A distribuição das chaves na função Polinomial é mais uniforme, o que resulta em uma menor probabilidade de colisões e maior eficiência na utilização da tabela hash.

Conclusão:

A função Polinomial se mostrou mais eficiente que a função LoseLose em termos de tempo de execução e distribuição das chaves. A função Polinomial apresenta resultados mais estáveis e eficientes, com menos colisões e índices vazios. Em contrapartida, a função LoseLose, embora mais simples de implementar, apresenta um desempenho inferior devido à maior taxa de colisões e à distribuição menos eficiente das chaves.