TRABALHO PRÁTICO 03

ARTHUR LINHARES MADUREIRA
2021031599

1) INTRODUÇÃO

O trabalho objetiva simular o funcionamento de um dicionário. Para isso, foram utilizadas duas estruturas diferentes (hash e árvore) que devem fornecer os mesmos resultados. Com isso, objetiva-se entender melhor o funcionamento e a implementação dessas duas estruturas de dados.

2) MÉTODOS

- **CLASSE SIGNIFICADOS:** essa classe possui um array que contém diferentes strings correspondentes às definições de um verbete. Como métodos, ela possui a função de adicionar um novo significado e imprimir os significados do array.
- **CLASSE VERBETE:** possui uma string correspondente à palavra e um atributo do tipo *Significados* para armazenar os significados referentes. Além disso, possui apontadores que são utilizados tanto na implementação da AVL (filhoEsquerdo e filhoDireito) quanto para a lista encadeada do hash (próximo)
- **CLASSE LISTAENCADEADA:** trata-se de uma lista encadeada de verbetes. Possui métodos para inserir um novo item alfabeticamente, remover os itens que possuem significado e imprimir os elementos da lista.
- **CLASSE HASHTABLE:** responsável por implementar o hash. Possui os métodos de inserir um verbete, remover todos os verbetes com significado e imprimir os verbetes da hash. Tendo em vista que serão utilizadas letras maiúsculas e minúsculas, a constante foi definida como 52.
- **CLASSE DICIONARIO:** contém os métodos referentes às duas estruturas de dados presentes no trabalho (hash e árvore balanceada).

3) ANÁLISE DE COMPLEXIDADE

3.1) ANÁLISE DE TEMPO

➔ AVL

OPERAÇÃO	MELHOR CASO	CASO MÉDIO
INSERÇÃO	O (log n)	O (log n)
DELEÇÃO	O (log n)	O (log n)
BUSCA	O (1)	O (log n)

➔ HASH

OPERAÇÃO	CASO MÉDIO
INSERÇÃO	O(1 + N/52)
DELEÇÃO	O(1 + N/52)
BUSCA	O(1 + N/52)

3.2) COMPLEXIDADE DE ESPAÇO

A complexidade de espaço é O(N) para hash e para AVL.

4) ESTRATÉGIAS DE ROBUSTEZ

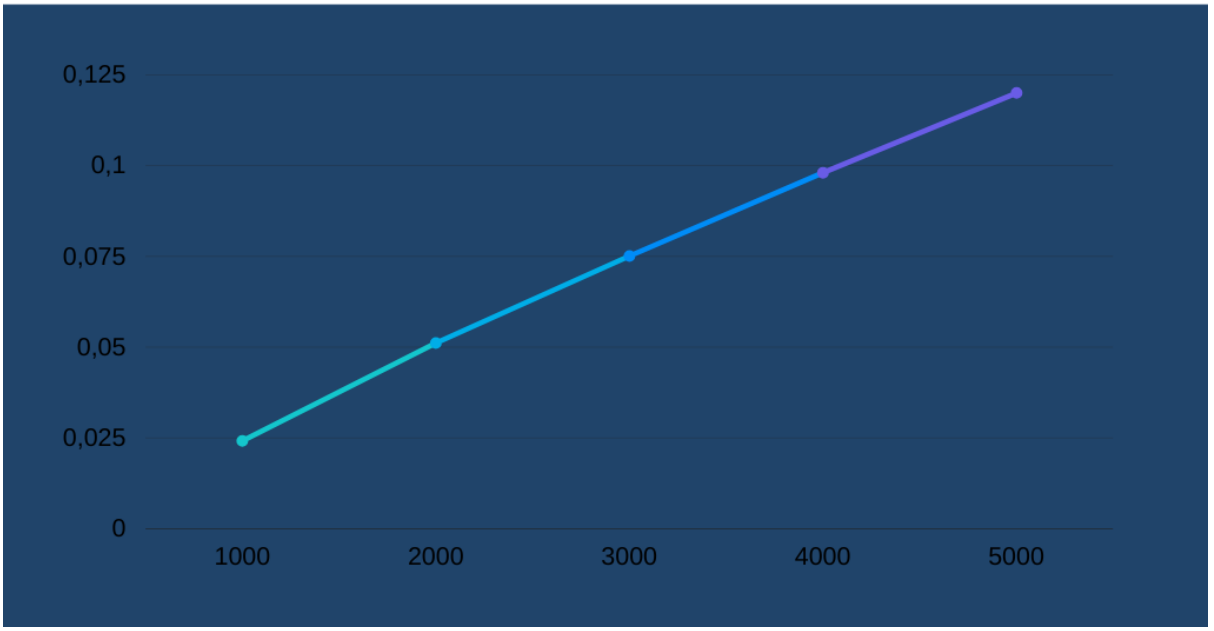
- **CHECAGEM DOS TIPOS:** se for fornecido um tipo não disponível de estrutura de dados, é acionado um *erroAssert*.
- **CHECAGEM DO ARQUIVO DE ENTRADA:** se o arquivo de entrada não for aberto corretamente, é acionado um *erroAssert*.

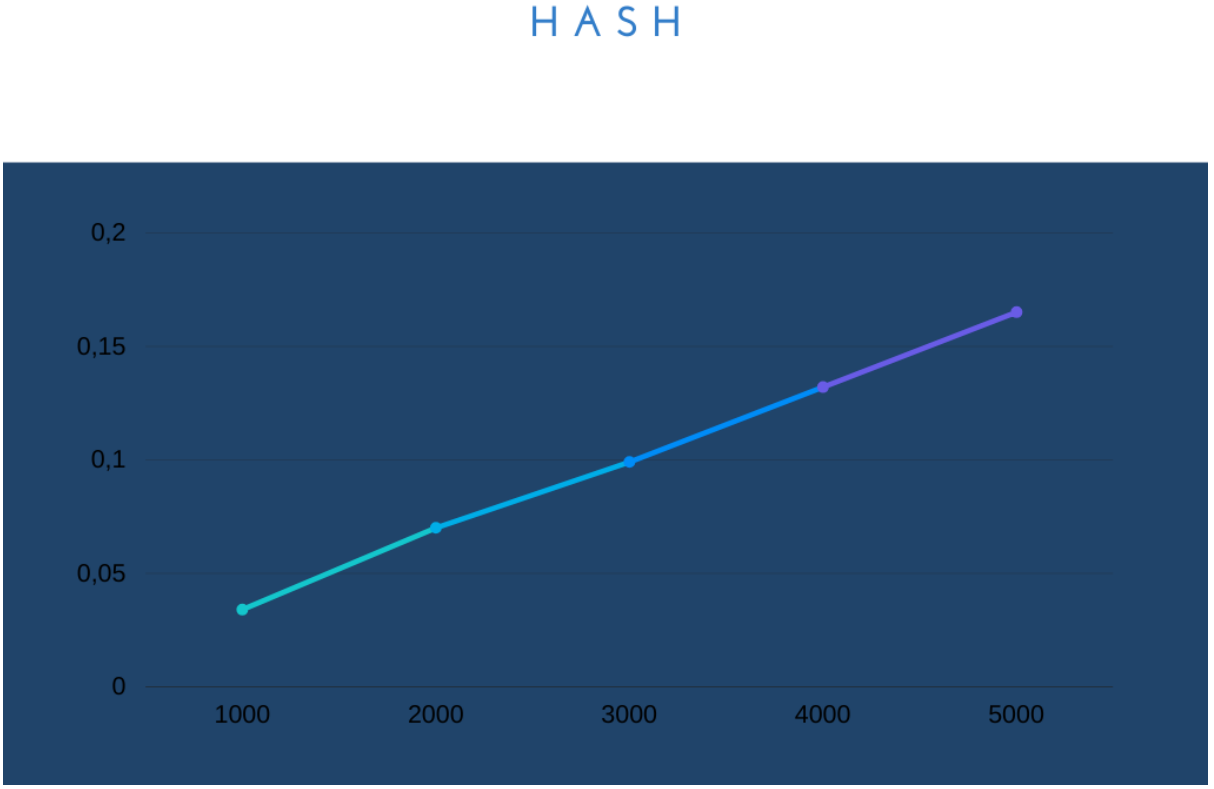
5) ANÁLISE EXPERIMENTAL

5.1) ANÁLISE DO TEMPO

Para realizar essa análise, vamos testar as duas estruturas de dados com diferentes tamanhos de entradas. Os resultados encontram-se expostos nos gráficos abaixo:

A V L

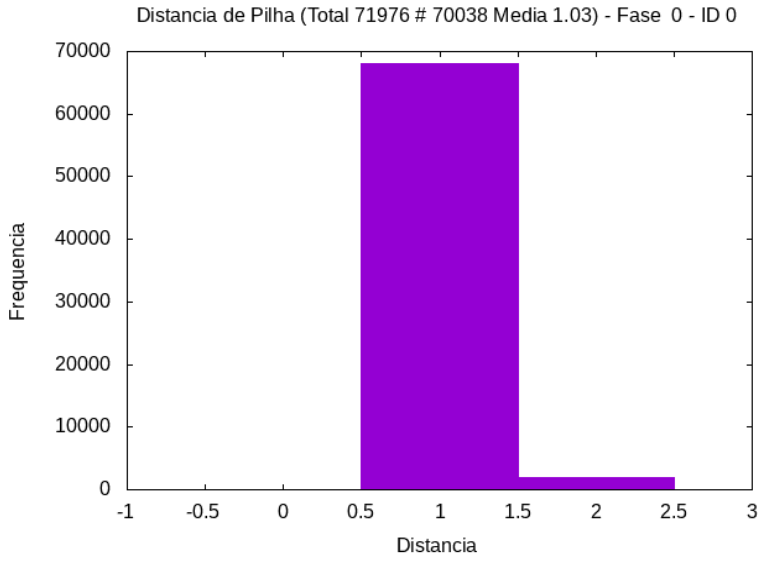


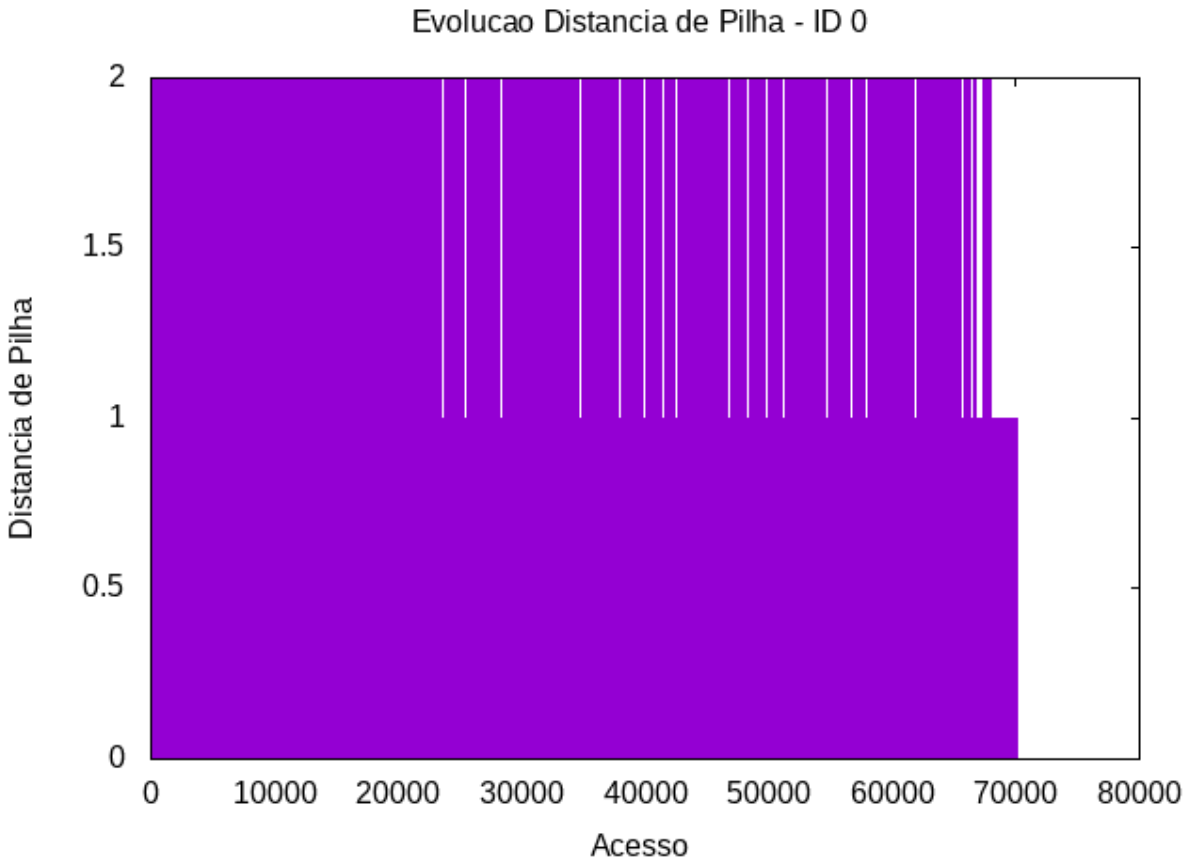


Por meio da análise dos gráficos, nota-se que a árvore AVL foi significativamente superior ao hash nos casos de teste.

5.2) ANALISAMEM

➔ **HASH**




Evolucao Distancia de Pilha - ID 0
Distancia de Pilha
2
1.5
1
0.5
0
0
10000
20000
30000
40000
50000
60000
70000
80000
Acesso

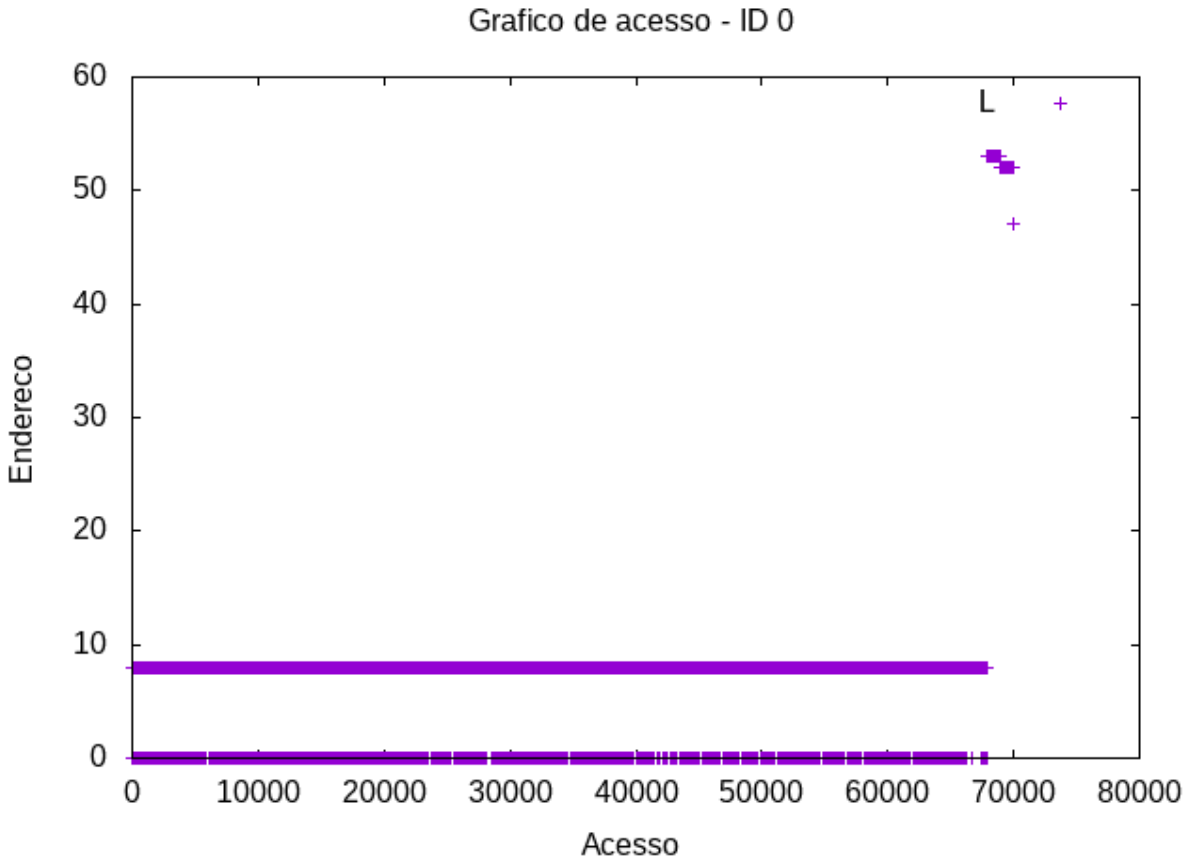
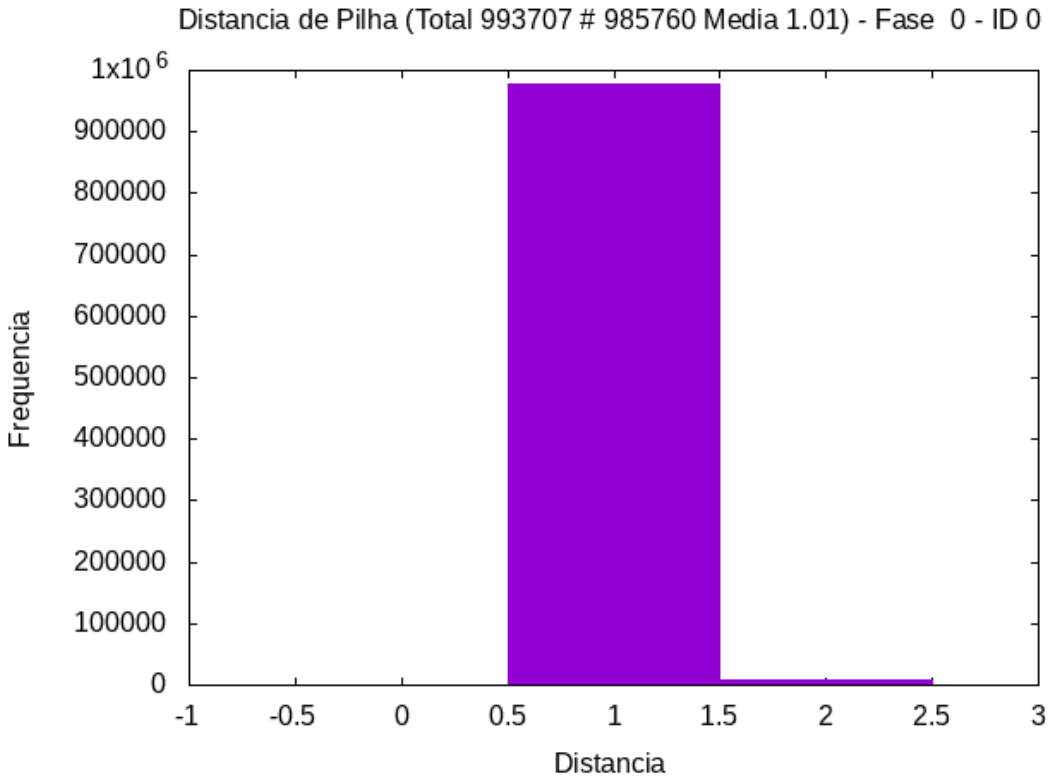
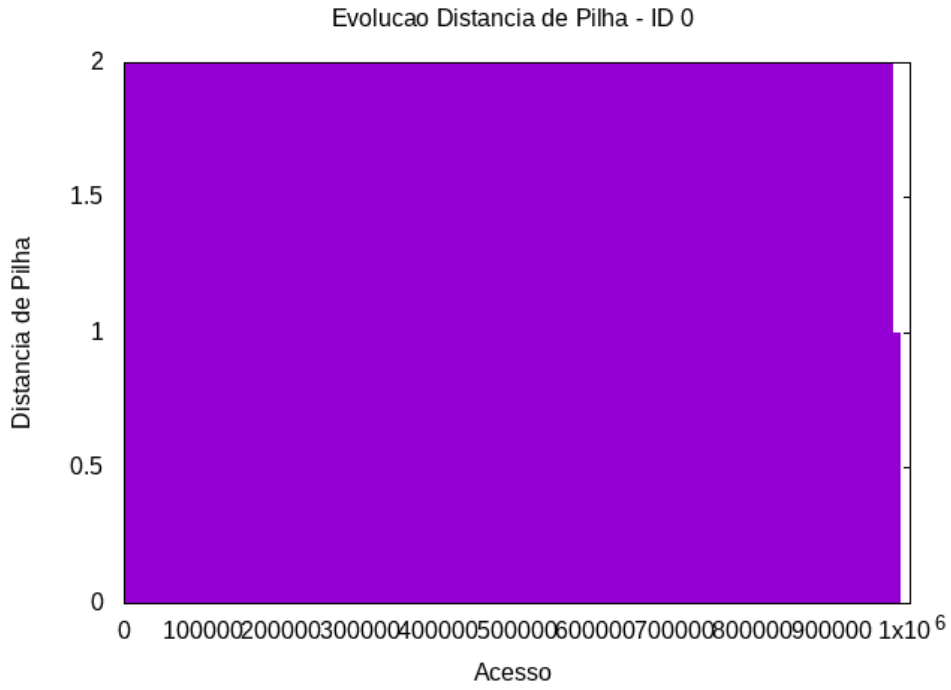
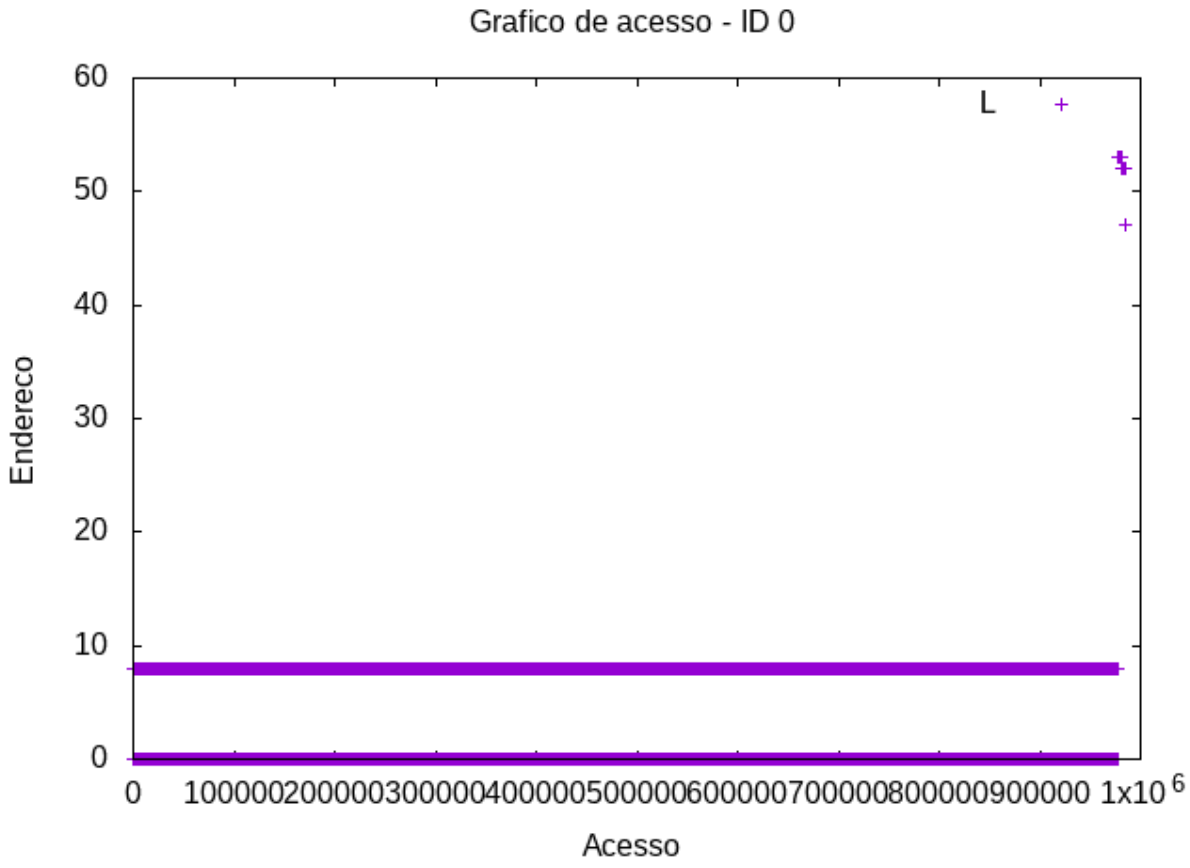


Grafico de acesso - ID 0
Endereco
60
50
40
30
20
10
0
L
0
10000
20000
30000
40000
50000
60000
70000
80000
Acesso

➔ **AVL**


Distancia de Pilha (Total 993707 # 985760 Media 1.01) - Fase 0 - ID 0
Frequencia
1x10 6
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
-1
-0.5
0
0.5
1
1.5
2
2.5
3
Distancia


Evolucao Distancia de Pilha - ID 0
Distancia de Pilha
2
1.5
1
0.5
0
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
1x10 6
Acesso



Os gráficos de ambas aplicações ficaram semelhantes. Isso ocorreu devido ao fato deles acontecerem de maneiras similares no que tange o acesso à memória.

6) CONCLUSÃO

Com a realização do trabalho, foi possível compreender melhor a aplicação das estruturas de dados propostas e como elas são utilizadas para gerar algoritmos de busca eficientes.

A tabela hash mapeia elementos de determinado tipo em um local da tabela. Já a árvore AVL é um modelo aprimorado da árvore binária de busca, já que ela evita a formação de árvores pouco benéficas do ponto de vista da eficiência dos métodos de busca.

Tendo isso em vista, é notório que as duas estruturas de dados são eficiente para simularem o funcionamento de um dicionário.

7) BIBLIOGRAFIA

Ziviani, N. (2006). Projetos de Algoritmos com Implementações em Java e C++ . Editora Cengage.

INSTRUÇÕES PARA COMPILAÇÃO E EXECUÇÃO

- Para compilar o programa, utilize o comando “make”
- Para executar o programa, digite ./bin/tp3 -i input.txt -t tipo (pode-se escolher o nome do arquivo de entrada e o tipo deve ser *hash* ou *arv*)
- Para ativar o memlog, use a flag -p.