Atividade Prática 8

1 - Estruturas de Dados a serem monitoradas

Nosso código irá envolver, novamente, a Pilha implementada por Filas Circulares. Como exploramos na AP 6, a Pilha é completamente dependente da Fila Circular para seu funcionamento. Dessa maneira, estaremos monitorando a Pilha. Porém, consequentemente, estaremos monitorando a Fila também.

2 - Funções a serem instrumentadas

As funções empilha(), desempilha() e printPilha() serão instrumentadas. Elas utilizam as funções enfila, desenfila(), filaFrente() e filaVazia().

3 - Definir fases de monitoramento

No nosso programa main, iremos criar a Pilha(e consequentemente duas Filas Circulares), empilhar 50 números, desempilhar 20 números para depois imprimirmos a Pilha.

- Fase 0: Criação da Pilha e das Filas.
- Fase 1: Empilhamento dos elementos.
- Fase 2: Desempilhamento dos elementos.
- Fase 3: Impressão da Pilha.

4 - Instrumentar o Código

O código foi instrumentado com o memlog.

```
6   Pilha::Pilha(){
7       this->id = idPilha;
8       ESCREVEMEMLOG((long int) (&(this->fila1)), sizeof(double), this->id);
9       idPilha++;
10  }
```

```
21  void Pilha::empilha(int valor){
22      this->fila2.enfila(valor);
23      ESCREVEMEMLOG((long int) (&(this->fila2.array[fila2.filaFrente()])), sizeof(double), this->fila2.id);
24
25      while(this->fila1.filaVazia() == false){
26          LEMEMLOG((long int) (&(this->fila1.array[fila1.filaFrente()])), sizeof(double), this->fila1.id);
27          this->fila2.enfila(this->fila1.filaFrente());
28          this->fila1.desenfila();
29      }
30
31      FilaCircular filaAuxiliar = this->fila1;
32      this->fila1 = this->fila2;
33      this->fila2 = filaAuxiliar;
34
35      std::cout << valor << " Empilhado\n";
36  }
```

```
void Pilha::desempilha(){
    if(this->fila1.filaVazia() == true){
        return;
    }
    else{
        this->fila1.desenfila();
        LEMEMLOG((long int) (&(this->fila1.array[fila1.filaFrente()])), sizeof(double), this->fila1.id);
        ESCREVEMEMLOG((long int) (&(this->fila1.array[fila1.filaFrente()])), sizeof(double), this->fila1.id);
    }

    std::cout << "Desempilhando\n";
}
```

```
51  void Pilha::printPilha(){
52      FilaCircular filaAuxiliar = this->fila1;
53
54      while(filaAuxiliar.filaVazia() == false){
55          std::cout << filaAuxiliar.filaFrente() << " ";
56          LEMEMLOG((long int) (&(filaAuxiliar.array[filaAuxiliar.filaFrente()])), sizeof(double), filaAuxiliar.id);
57          ESCREVEMEMLOG((long int) (&(filaAuxiliar.array[filaAuxiliar.filaFrente()])), sizeof(double), filaAuxiliar.id);
58          filaAuxiliar.desenfila();
59      }
60      std::cout << "\n";
61  }
```

```
13
14  char lognome[1000];
15
16  int main(){
17      lognome[0] = 'l';
18      lognome[1] = 'o';
19      lognome[2] = 'g';
20
21      iniciaMemLog(lognome);
22      ativaMemLog();
23
24      defineFaseMemLog(0);
25      Pilha pilha;
26
27      defineFaseMemLog(1);
28      for(int i = 0; i < 50; i++){
29          pilha.empilha(i);
30      }
31
32      defineFaseMemLog(2);
33      for(int i = 20; i < 40; i++){
34          pilha.desempilha();
35      }
36
37      std::cout << "Pilha após todas operações:\n";
38      defineFaseMemLog(3);
39      pilha.printPilha();
40  }
```

5 - Definir o Plano de Experimentos

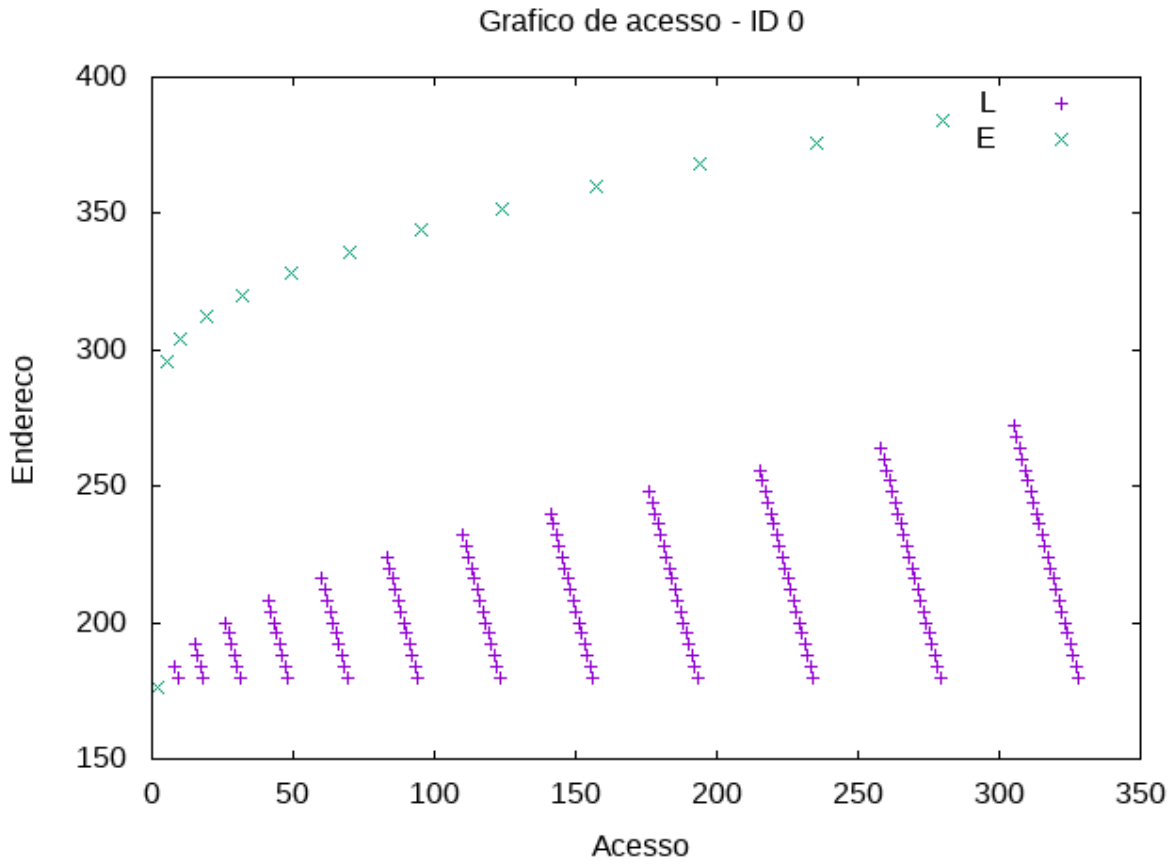
Como pode ser visto no código acima, iremos:

- Criar 7 Pilhas.
- Empilhar 25 elementos em uma Pilha.
- Desempilhar 13 elementos em uma Pilha.
- Exibir uma Pilha.

6 - Executar os Experimentos

Executamos os experimentos, gerando nosso arquivo do memlog.

7 - Gerar as Visualizações



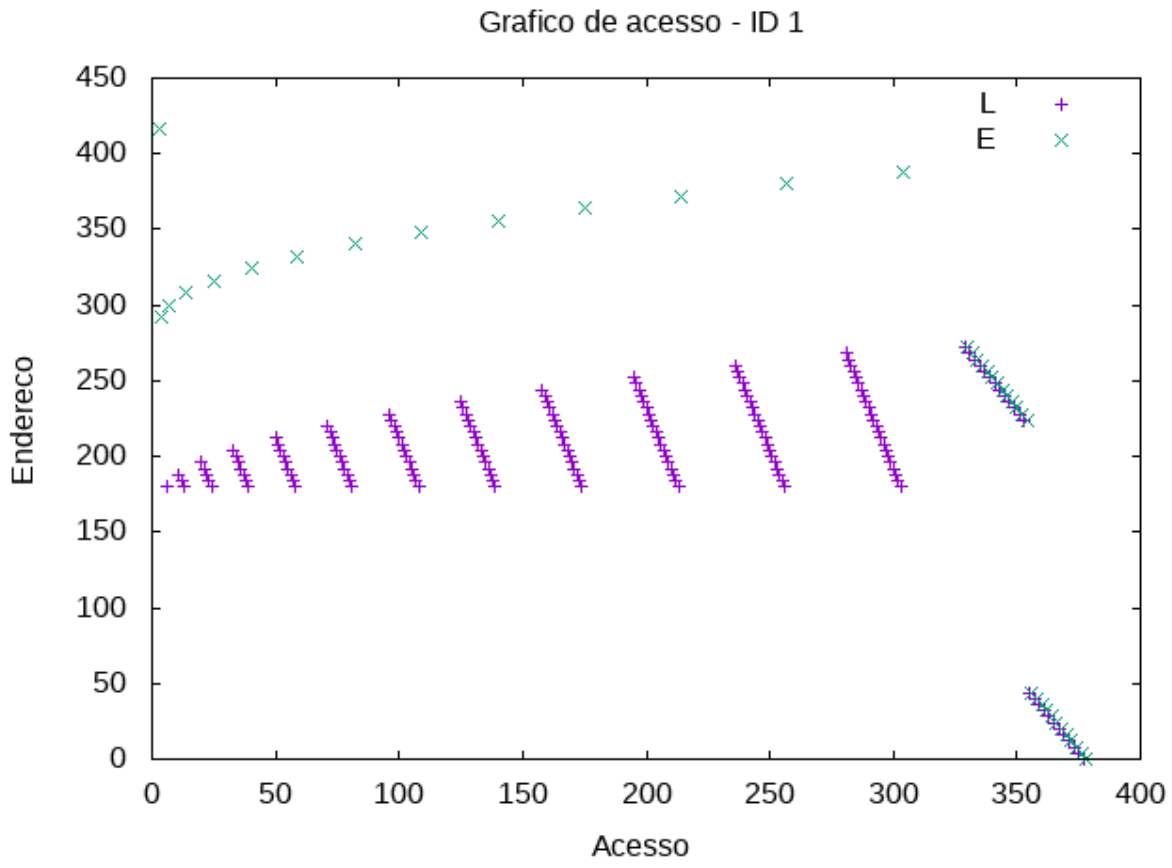
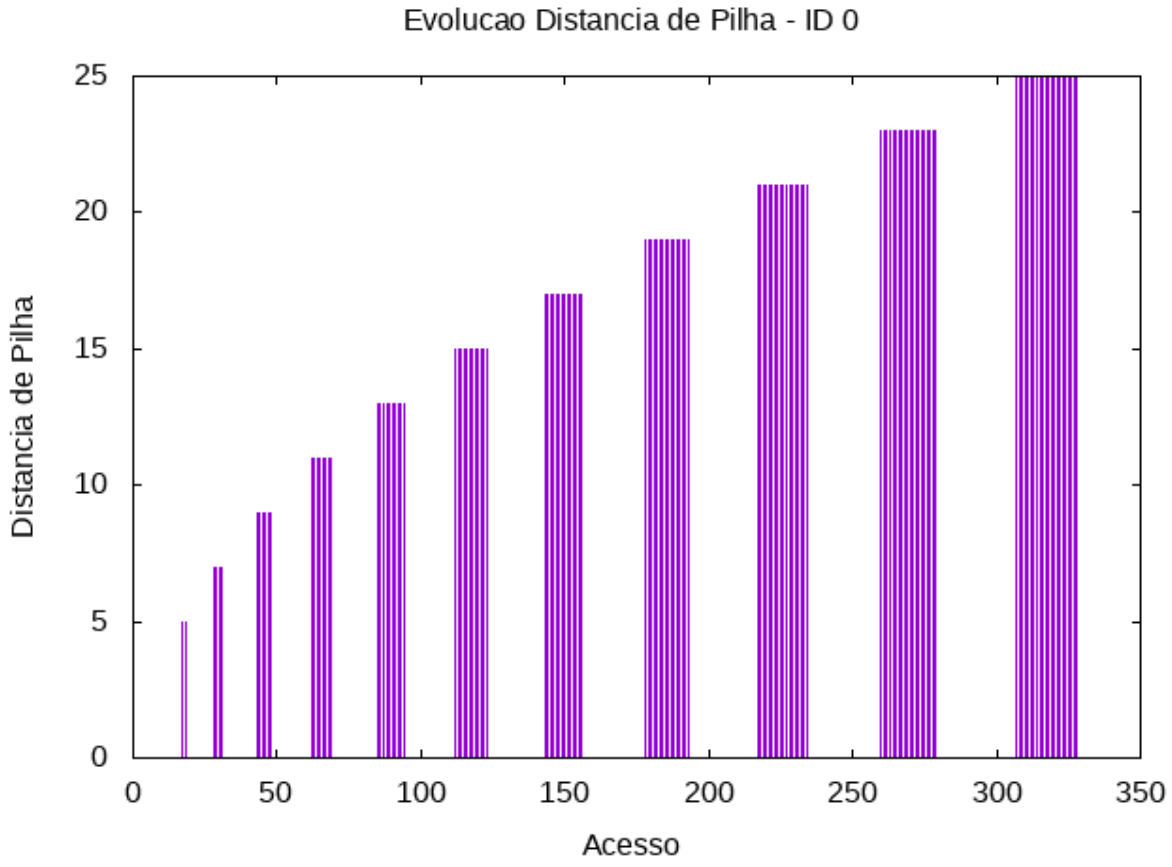
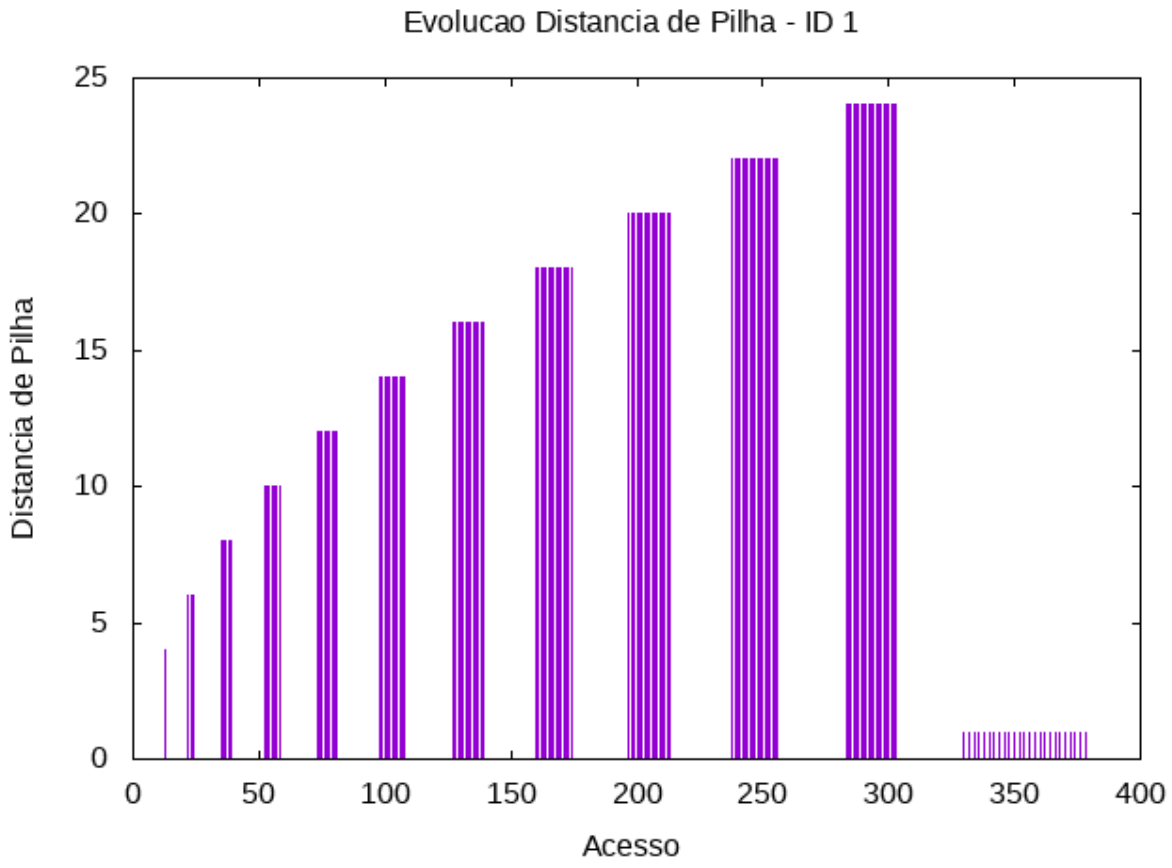
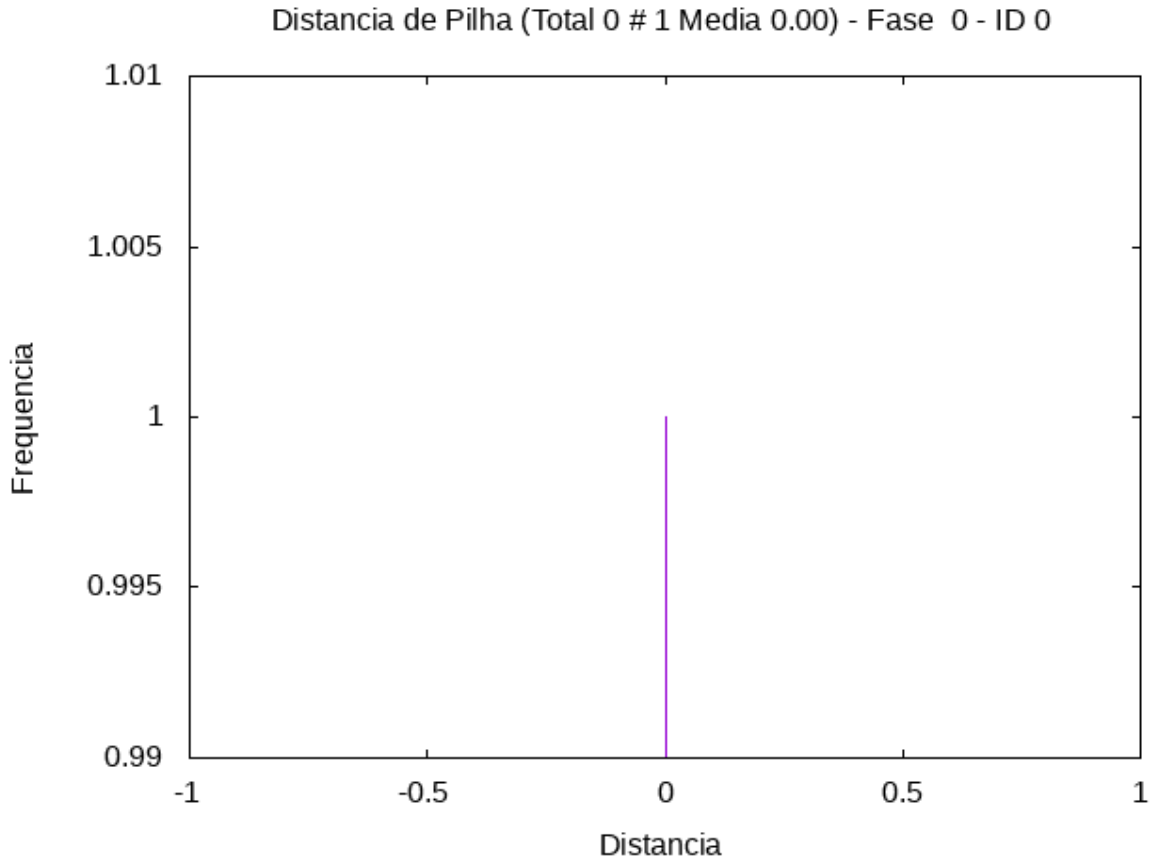
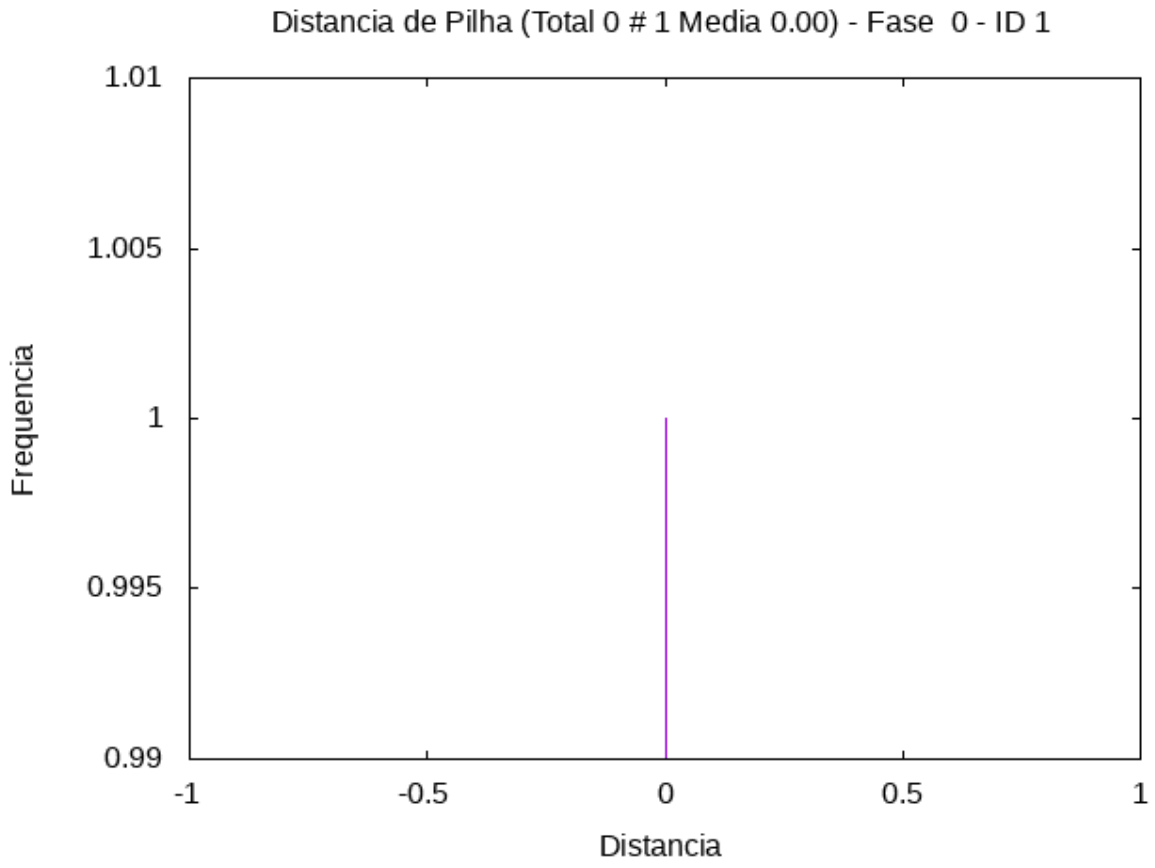


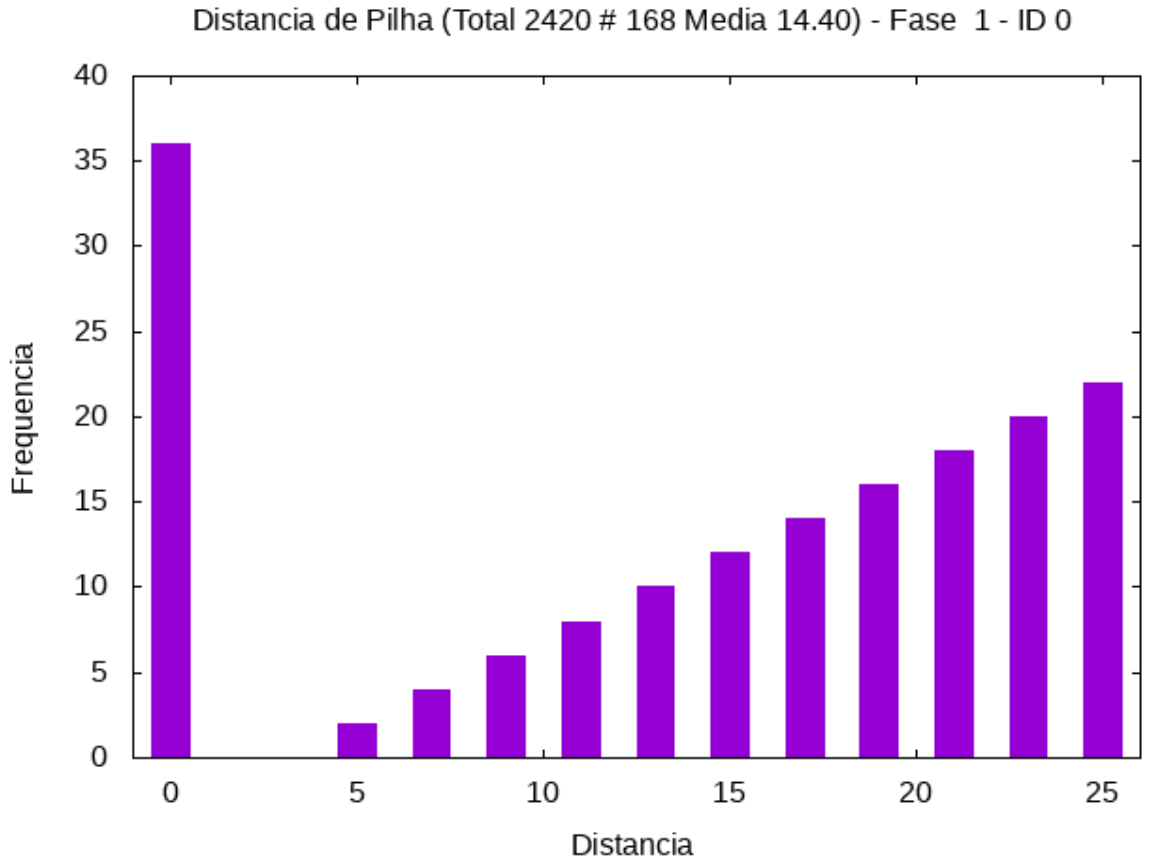
Grafico de acesso - ID 1
Endereco
Acesso
L
E
0
50
100
150
200
250
300
350
400
450

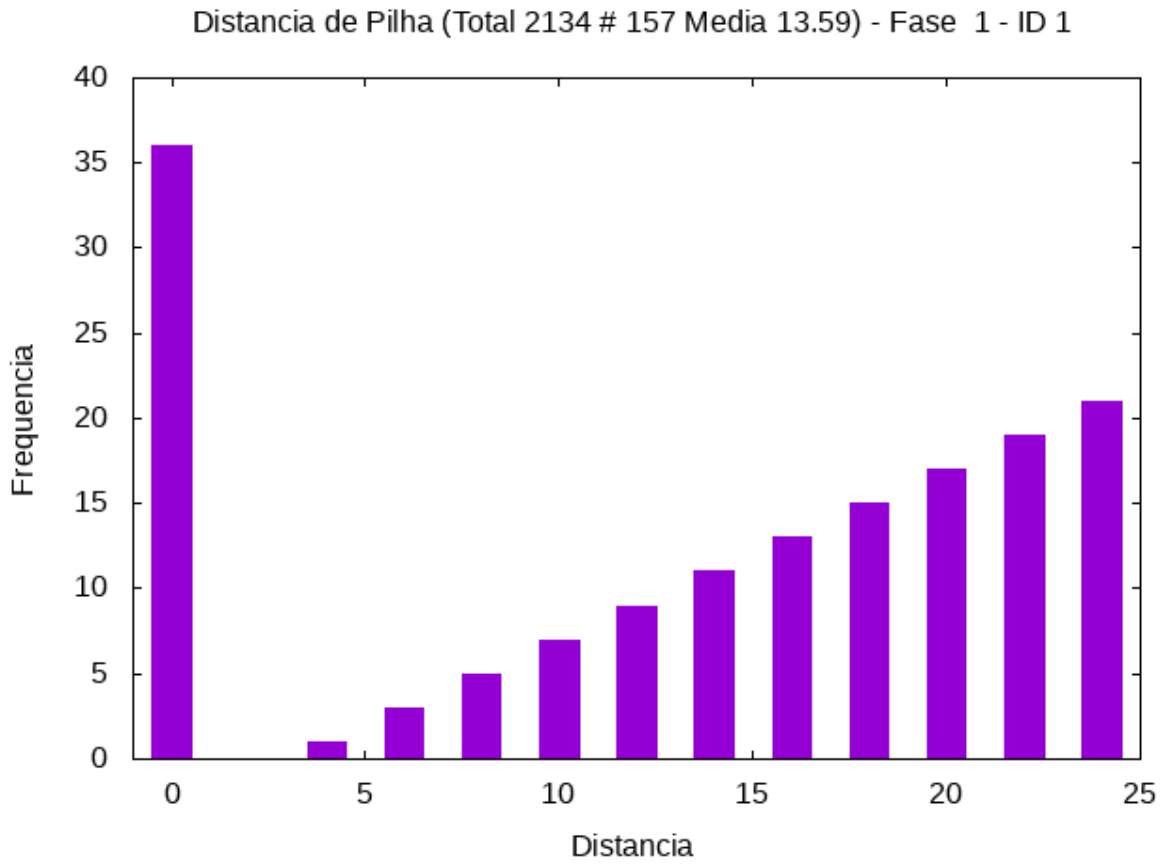

Evolucao Distancia de Pilha - ID 0
Distancia de Pilha
Acesso
0
5
10
15
20
25
50
100
150
200
250
300
350

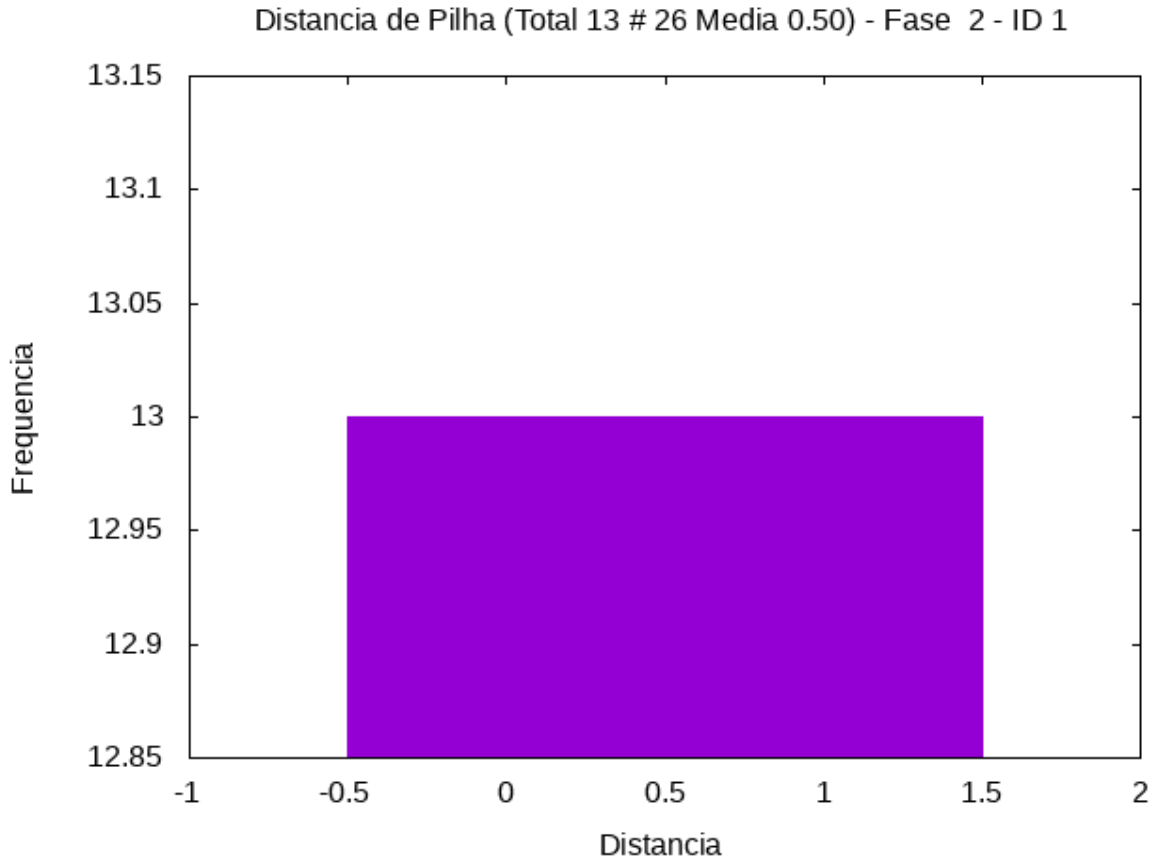

Evolucao Distancia de Pilha - ID 1
Distancia de Pilha
Acesso
0
5
10
15
20
25
0
50
100
150
200
250
300
350
400

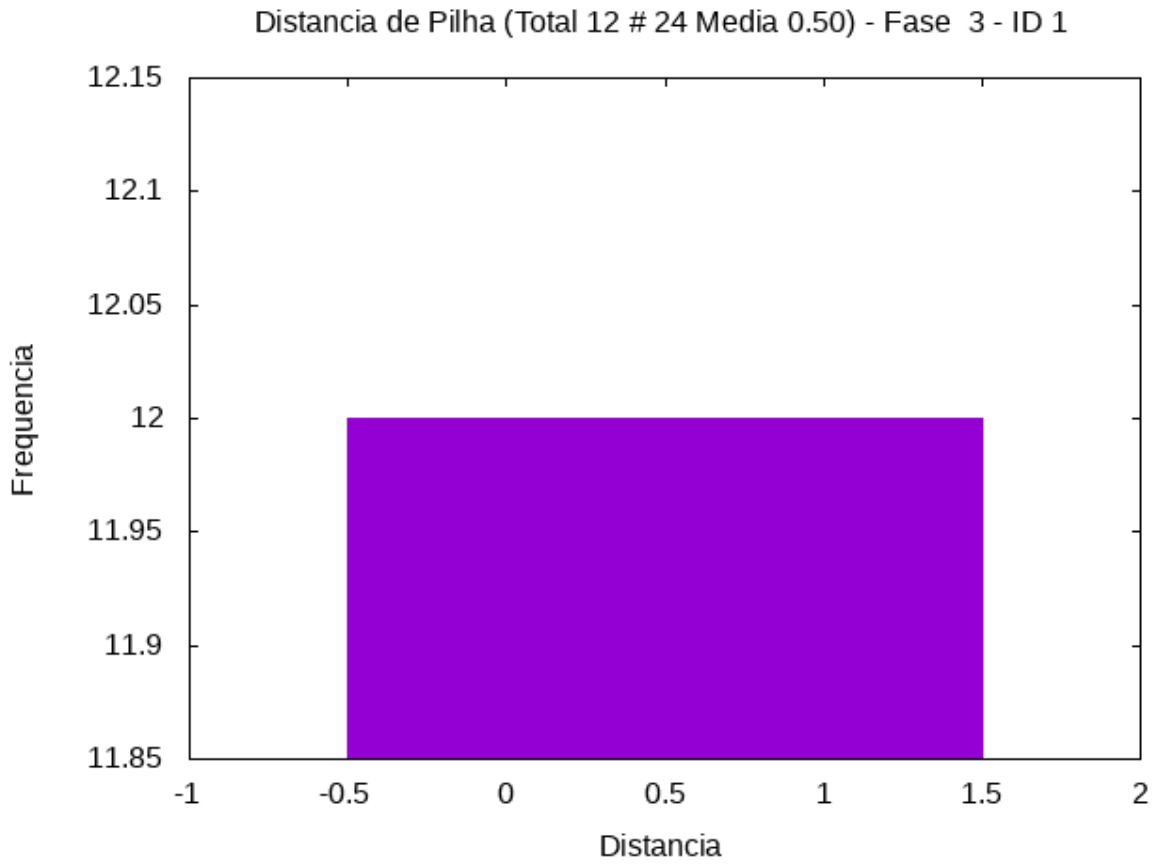

Distancia de Pilha (Total 0 # 1 Media 0.00) - Fase 0 - ID 0
Frequencia
Distancia
0.99
0.995
1
1.005
1.01
-1
-0.5
0
0.5
1


Distancia de Pilha (Total 0 # 1 Media 0.00) - Fase 0 - ID 1
Frequencia
Distancia
1.01
1.005
1
0.995
0.99
-1
-0.5
0
0.5
1


Distancia de Pilha (Total 2420 # 168 Media 14.40) - Fase 1 - ID 0
Frequencia
Distancia
40
35
30
25
20
15
10
5
0
0
5
10
15
20
25


Distancia de Pilha (Total 2134 # 157 Media 13.59) - Fase 1 - ID 1
Frequencia
Distancia
0
5
10
15
20
25
30
35
40


Distancia de Pilha (Total 13 # 26 Media 0.50) - Fase 2 - ID 1
Frequencia
Distancia
12.85
12.9
12.95
13
13.05
13.1
13.15
-1
-0.5
0
0.5
1
1.5
2



8 - Analisar os Resultados e Visualizações

Primeiramente, o gráfico de acesso nos permite visualizar com clareza cada operação relacionada às duas filas circulares, de ID 0 e 1.

Tanto na fila 0 e na fila 1, a instância única de escrita em cada momento se refere ao empilhamento de um valor. A cadeia longa de leituras se refere à cópia de seus valores para a criação da fila auxiliar utilizada no método de empilhamento.

Por fim, exclusivamente na fila 1, a cadeia de escritas em conjunto com leituras se refere ao método de impressão da pilha, onde a fila é copiada uma vez e tem todos seus valores acessados.

Ademais, a Distância de Pilha também permite tais análises. A da fase 0 indica sua alocação única. A da fase 1 reflete o distanciamento da parte de trás da fila que nós observamos a modo de que mais elementos são empilhados. A fase 2 se demonstra constante pois, o desenfilamento apenas diminui o topo da pilha, mas o array da fila ainda possui o elemento, por isso não há diminuição da distância. A fase 3 também permanece constante, porque os elementos são apenas acessados para sua impressão.