Trabalho Pratico 1

Operações com matrizes alocadas dinamicamente

João Vitor Ferreira

Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte – MG – Brasil

joaovitorferreira@ufmg.br

1. Introdução

O problema proposto foi o de implementar um código no qual realiza as principais funções em matrizes, a soma, multiplicação e transposição. Deve-se alocá-las dinamicamente.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection.

2.1. Estrutura de Dados

A implementação teve como parâmetro o uso de matrizes contendo dados do tipo Double em sua composição. Com o espaço necessário sendo alocado dinamicamente. Ela foi desenvolvida a partir de uma struct “mat_tipo”, dentro do arquivo “mat.h”. com isso foram criadas as funções necessárias para que se fosse possível realizar as operações propostas, sendo elas, criação da matriz,

inicialização com valores iguais a 0, preenchimento com valores lidos de um arquivo, acesso dos valores, impressão da Matriz, retornar o elemento de determinada posição, cópia da Matriz, destruir e desalocar a matriz e por fim as funções responsáveis pelas operações feitas sobre as matrizes soma, multiplicações e transposição.

2.2. Classes

A modularização foi implementada, por tanto ocorreu e 1 struct responsável por lidar com os paramentos da matriz tais como o as posições x e y e o id

2.3. Funções

Foram implementadas 13 funções com paramentos e funcionamento diferentes.

criaMatriz (mat_tipo *mat, int tx, int ty, int id), recebe uma matriz por referência, um valor do tipo Double tx e ty e um int contendo o id da matriz, seu funcionamento atribui para as instancias mat->tamx e mat->tamy da matriz os valores provindos de tx e ty, respectivamente, e para mat->id o id, após isso é realizada a alocação da matriz através do malloc.

inicializaMatrizNula (mat_tipo *mat), recebe uma matriz como referência, e com dois for alinhado atribui para todas as posições da matriz o valor 0.

atribuiValoresMatriz (mat_tipo *mat, char arq[]), recebe uma matriz como referência e um char contendo o nome do arquivo, esta função inicialmente realiza uma chamada para a função inicializaMatrizNula, após abre o arquivo

determinado na variável arq, realiza a leitura dos dois primeiros valores por fscanf, aos quais são referentes ao tamanho da matriz, e com 1 loop realiza a leitura dos elementos no arquivo e atribui eles a matriz a posição correspondente da matriz, após esse loop o arquivo é fechado.

acessaMatriz(mat_tipo *mat)), recebe uma matriz como referência, e acessa todas as posições da matriz.

imprimeMatriz(mat_tipo *mat), recebe uma mat_tipo por referência, e realiza a impressão na tela dessa matriz, com uma formatação que denota o número da linha e coluna, juntamente com uma indentação alinhada dos valores presente em cada posição, aos quais são acessados com 2 loops alinhados.

criaArquivoresultado(mat_tipo *mat, char arq[]), recebe além da matriz o nome do arquivo que o resultado sera salvo, nesta função após fazer a abertura do arquivo é escrito nele os valores de mat->tamx e mat->tamy seguidos de uma quebra de linha, após são impressos todos os valores presentes na matriz após serem feitas as operações respeitando a quebra de linha ao final da linha.

escreveElemento(mat_tipo *mat, int x, int y, double v), é recebido a matriz o número de linha e coluna e um valor, é realizada a atribuição do valor v a posição [x][y] na matriz.

leElemento(mat_tipo *mat, int x, int y), é passada a matriz e as posições x e y, esta função somente retorna o valor contido na posição mat->m[x][y].

copiaMatriz(mat_tipo *src, mat_tipo *dst, int dst_id), são passadas duas matrizes, o tamanho x e y da primeira o id da segunda matriz, a segunda é inicializada com

zeros pela função inicializaMatrizNula é feita a cópia dos valores da primeira para a segunda.

somaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c), são passadas por referência 3 matrizes, é criada e inicializada com zeros a matriz c e após é feita a atribuição na matriz c da soma dos valores de cada posição da matriz a com b, ou seja, c->m[i][j] = a->m[i][j] + b->m[i][j].

multiplicaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c), são passadas 3 matrizes, e é feita a atribuição em c da multiplicação de a pôr b.

transpoeMatriz(mat_tipo *a), é passada uma matriz, e nela é feita a troca das linhas pelas colunas.

destroiMatriz(mat_tipo *a), a matriz passada por referência tem sua memória desalocada e a invalidação de seus índices id, tamx e tamy.

2.4. Formato de Entrada e Saída

O formato de entrada utilizado foi a análise dos atributos passada por parâmetro na linha de comando, portando deve ser passada o parâmetro que indica qual operação deverá ser realizada, soma(-s), multiplicação(-m) e transposição(-t), e os nomes dos arquivos onde se encontram as matrizes sendo que para a soma e multiplicação são necessários 2 arquivos precedidos dos parâmetro -1 e -2 e para a transposição somente é necessário 1 arquivo precedido de -1.

3. Analise Complexidade

3.1. Tempo

Inicialmente iremos seguir algumas preposições as estruturas auxiliares mais básicas consideraremos com o $O(1)$, e o custo da funções são:

criaMatriz (mat_tipo *mat, int tx, int ty, int id) = existem 5 operações de atribuição portanto cada uma vale $O(1)$, e 1 loop, que vale $O(n)$

$$5 \cdot O(1) + \theta(n)$$
$$= \max(\theta(1), \theta(n))$$
$$= \theta(n)$$

inicializaMatrizNula (mat_tipo *mat), há 2 operações de inicialização, 1 de atribuição e 2 loop em serie, portanto:

$$2 \cdot O(1) + O(1) + \theta(n^2)$$
$$= \max(\theta(1), \theta(n^2))$$
$$= \theta(n^2)$$

atribuiValoresMatriz (mat_tipo *mat, char arq[]), Há 11 funções de inicialização e atribuição e 1 loop

$$11 \cdot O(1) + \theta(n)$$
$$= \max(\theta(1), \theta(n))$$
$$= \theta(n)$$

acessaMatriz(mat_tipo *mat)), Há 5 funções de inicialização e atribuição e 2 loops em sequência

$$5 \cdot O(1) + \theta(n^2)$$

$$= \max(\theta(1), \theta(n^2))$$

$$= \theta(n^2)$$

imprimeMatriz(mat_tipo *mat), Há 7 funções de inicialização e atribuição e 1 for isolado e 2 for em sequência

$$7 \cdot O(1) + \theta(n) + \theta(n^2)$$

$$= \max(\theta(1), \theta(n), \theta(n^2))$$

$$= \theta(n^2)$$

criaArquivoresultado(mat_tipo *mat, char arq[]), Há 7 funções de inicialização e atribuição e 1 loop

$$7 \cdot O(1) + \theta(n)$$

$$= \max(\theta(1), \theta(n))$$

$$= \theta(n)$$

escreveElemento(mat_tipo *mat, int x, int y, double v), Ha somente funções 2 funções de atribuição

$$2 \cdot O(1)$$

$$= O(1)$$

leElemento(mat_tipo *mat, int x, int y), Ha somente funções 2 funções de atribuição

$$2 \cdot O(1)$$

$$= O(1)$$

copiaMatriz(mat_tipo *src, mat_tipo *dst, int dst_id), é chamada à função cria matriz que possui complexidade $\theta(n)$ após a inicia matriz nula $\theta(n^2)$, após 2 for em serie e 2 atribuições

$$2 \cdot O(1) + \theta(n) + \theta(n^2) + \theta(n^2)$$

$$= \max(\theta(1), \theta(n^2), \theta(n^2))$$

$$= \theta(n^2)$$

somaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c), é chamada à função cria matriz que possui complexidade $\theta(n)$ após a inicia matriz nula $\theta(n^2)$, após 2 for em serie e 1 atribuição

$$O(1) + \theta(n) + \theta(n^2) + \theta(n^2)$$

$$= \max(\theta(1), \theta(n^2), \theta(n^2))$$

$$= \theta(n^2)$$

multiplicaMatrizes(mat_tipo *a, mat_tipo *b, mat_tipo *c), é chamada à função cria matriz que possui complexidade $\theta(n)$ após a inicia matriz nula $\theta(n^2)$, após 3 for em serie e 1 atribuição

$$O(1) + \theta(n) + \theta(n^2) + \theta(n^3)$$
$$= \max(\theta(1), \theta(n^2), \theta(n^3))$$
$$= \theta(n^3)$$

transpoeMatriz(mat_tipo *a), Há 4 funções de inicialização e atribuição e 2 loops em sequência

$$4 \cdot O(1) + \theta(n^2)$$
$$= \max(\theta(1), \theta(n^2))$$
$$= \theta(n^2)$$

destroiMatriz(mat_tipo *a), Há 4 funções de inicialização e atribuição e 1 for

$$4 \cdot O(1) + \theta(n)$$
$$= \max(\theta(1), \theta(n))$$
$$= \theta(n)$$

3.2. Espaço

O espaço usado é único ou seja um valor somente poderá ser atribuído a uma posição, desconsiderando que ele apareça em outras posições na aleatoriedade, portanto somente sera alocada a quantidade de espaço necessária para todos

os elementos, como estamos trabalhando com matrizes. Com isso temos que o espaço sera de $\theta(n^2)$ para o programa todo. Foram realizados 10 testes com diferentes tamanhos de matrizes, se alternando de 500 em 500, iniciando em 500 a finalizando em 5000

4. Estratégias de Robustez

Para garantir o correto funcionamento do código foram adicionados errosAssert, dos quais param o código toda vez que um erro é localizado. Os erros que foram adicionados estão relacionados com a alocação e abertura dos arquivos. No caso da abertura do arquivo caso ele está vazio um erro retornara avisando que não há nada no arquivo. E no caso da alocação caso o valor não tenha sido corretamente alocado um erro irá ser retornado.

5. Testes

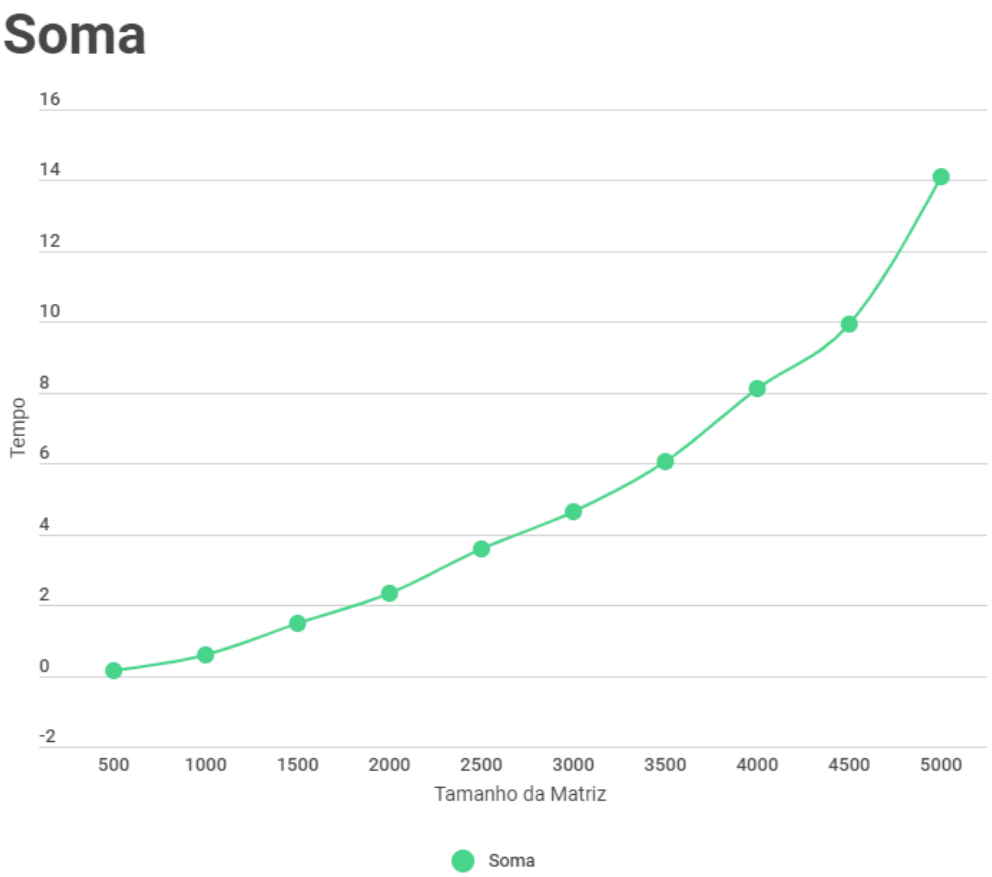
Foram realizados 10 testes com diferentes tamanhos de matrizes, se alternando de 500 em 500, iniciando em 500 a finalizando em 5000. As entradas e saídas estão disponíveis No seguinte link: https://ufmgbr-my.sharepoint.com/:f:/g/personal/joaovitorferreira_ufmg_br/ElAdhxn2NqxOrdnAuhsL8VoBZk5ydqhLuusVPYXTOiOYKA?e=1Py1SW, não estão sendo disponibilizadas juntamente aos arquivos do TP0 devido ao Moodle somente aceitar arquivos até 20mb e minhas entradas estarem no total de 3,66GB, sendo A e B os valores usados no teste. Os arquivos foram nomeados com índices, os de entrada variam de m1 até m22, enquanto os de saída varia de res1 até res33. Juntamente destes testa estará a saída gprof.

Farei a análise do tempo de execução e gprof, esse teste foi feito sobre uma matriz de tamanho 5000x5000.

5.1. Soma

5.1.1 Tempo

Dimensão	Tempo de execução em segundos	Crescimento
500	0,139175528	-
1000	0,595092392	0,455916864
1500	1,498495011	0,903402619
2000	2,338212073	0,839717062
2500	3,588380553	1,250168480
3000	4,649898185	1,061517632
3500	6,053216431	1,403318246
4000	8,107226474	2,054010043
4500	9,923210719	1,815984245
5000	14,099163914	4,175953195



5.1.2 Gprof

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
72.35	2.37	2.37	4	593.24	593.24	inicializaMatrizNula
10.38	2.71	0.34	2	170.21	763.46	atribuiValoresMatriz
9.16	3.01	0.30	4	75.09	75.09	acessaMatriz
4.27	3.15	0.14	1	140.18	140.18	criaArquivoresultado
3.97	3.28	0.13	1	130.16	723.41	somaMatrizes

0.00	3.28	0.00	4	0.00	0.00	criaMatriz
0.00	3.28	0.00	3	0.00	0.00	defineFaseMemLog
0.00	3.28	0.00	3	0.00	0.00	destroiMatriz
0.00	3.28	0.00	2	0.00	0.00	le_dado_matriz_x
0.00	3.28	0.00	2	0.00	0.00	le_dado_matriz_y
0.00	3.28	0.00	1	0.00	0.00	clkDifMemLog
0.00	3.28	0.00	1	0.00	0.00	desativaMemLog
0.00	3.28	0.00	1	0.00	0.00	finalizaMemLog
0.00	3.28	0.00	1	0.00	0.00	iniciaMemLog
0.00	3.28	0.00	1	0.00	0.00	parse_args

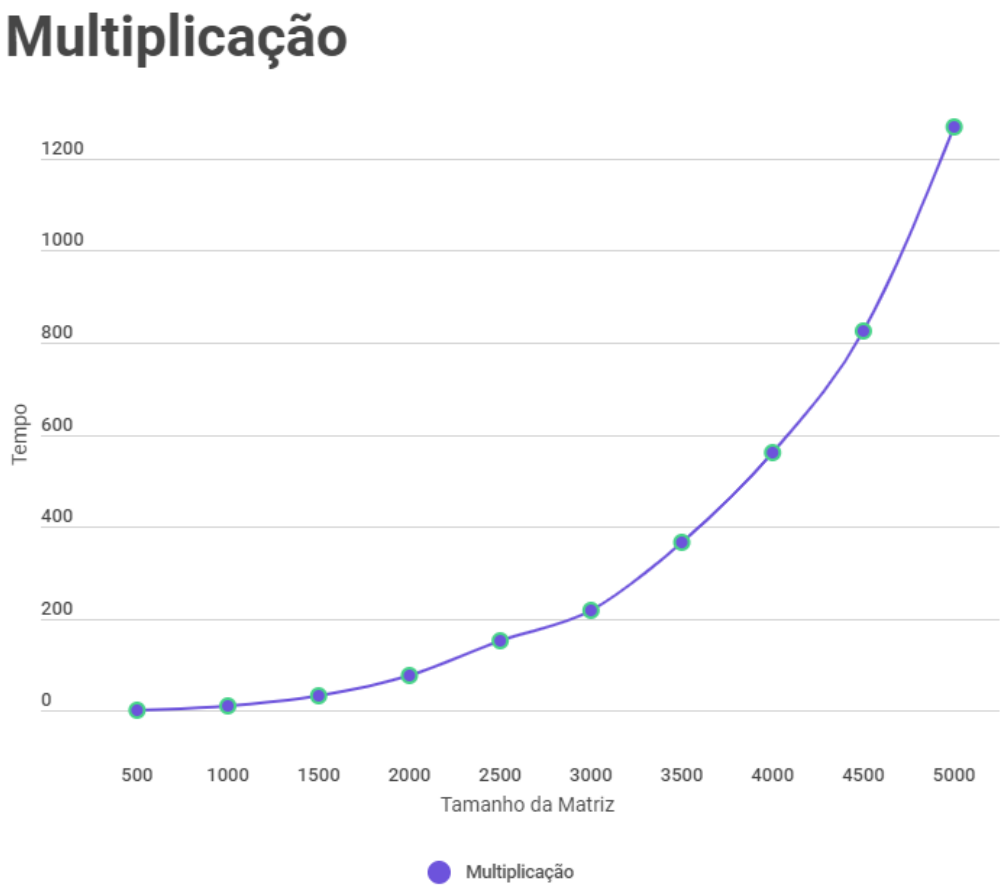
5.1.3 Analise

Após analisar a tabela, gráfico e resultado gprof, podemos inferir que o crescimento da função pé exponencial, e no pior caso analisado a matriz de tamanho 5000x5000, temos que o tempo total de execução 14 segundos e a maior parte desse tempo vem da funções somaMatrizes e inicializaMatrizNula. Sendo as duas que geram o resultado acima em sua maior parte

5.2 Multiplicação

5.2.1 Tempo

Dimensão	Tempo de execução em segundos	Crescimento
500	1,024875439	-
1000	9,209711724	8,184836285
1500	31,163369089	21,953657365
2000	76,512557276	45,349188187
2500	151,563006255	75,050448979
3000	218,434890627	66,871884372
3500	367,718079370	149,283188743
4000	560,134205233	192,416125863
4500	824,848848762	264,714643529
5000	1268,271472766	443,422624004



5.2.2 Gprof

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
99.83	728.50	728.50	1	728.50	728.80	multiplicaMatrizes
0.16	729.67	1.17	4	0.29	0.29	inicializaMatrizNula
0.06	730.10	0.43	2	0.22	0.51	atribuiValoresMatriz
0.05	730.49	0.39	4	0.10	0.10	acessaMatriz
0.02	730.64	0.15	1	0.15	0.15	criaArquivoresultado

0.00	730.64	0.00	4	0.00	0.00	criaMatriz
0.00	730.64	0.00	3	0.00	0.00	defineFaseMemLog
0.00	730.64	0.00	3	0.00	0.00	destroiMatriz
0.00	730.64	0.00	2	0.00	0.00	le_dado_matriz_x
0.00	730.64	0.00	2	0.00	0.00	le_dado_matriz_y
0.00	730.64	0.00	1	0.00	0.00	clkDifMemLog
0.00	730.64	0.00	1	0.00	0.00	desativaMemLog
0.00	730.64	0.00	1	0.00	0.00	finalizaMemLog
0.00	730.64	0.00	1	0.00	0.00	iniciaMemLog
0.00	730.64	0.00	1	0.00	0.00	parse_args

5.2.3 Analises

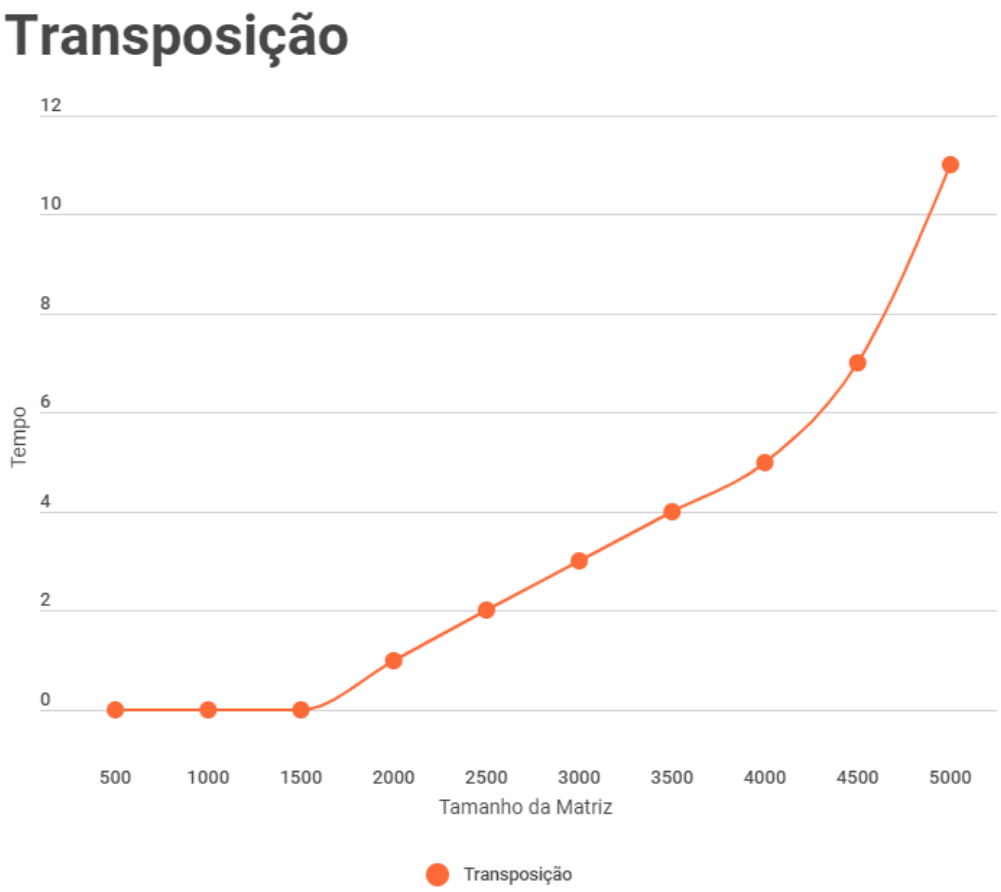
Após analisar a tabela, gráfico e resultado gprof, podemos inferir que o crescimento da função é exponencial e cresce rapidamente, dado que a função e Ômega n ao cubo, e no pior caso analisado a matriz de tamanho 5000x5000, temos que o tempo total de execução 21 minutos e a maior parte desse tempo vem da função multiplicaMatrizes.

5.3 Transposição

5.3.1 Tempo

Dimensão	Tempo de execução em segundos	Crescimento

500	0,111172882	-
1000	0,404853575	0,293680693
1500	0,923617300	0,518763725
2000	1,628952394	0,705335094
2500	2,114831255	0,485878861
3000	3,155710917	1,040879662
3500	4,517754450	1,362043533
4000	5,955114970	1,437360520
4500	7,381269059	1,426154089
5000	11,082230910	3,700961851



5.3.2 Gprof

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
27.87	0.27	0.27	1	270.34	410.51	atribuiValoresMatriz
24.77	0.51	0.24	1	240.30	240.30	transpoeMatriz
17.55	0.68	0.17	1	170.21	170.21	criaArquivoresultado
15.48	0.83	0.15	2	75.09	75.09	acessaMatriz
14.45	0.97	0.14	1	140.18	140.18	inicializaMatrizNula
0.00	0.97	0.00	3	0.00	0.00	defineFaseMemLog

0.00	0.97	0.00	1	0.00	0.00	clkDifMemLog
0.00	0.97	0.00	1	0.00	0.00	criaMatriz
0.00	0.97	0.00	1	0.00	0.00	desativaMemLog
0.00	0.97	0.00	1	0.00	0.00	destroiMatriz
0.00	0.97	0.00	1	0.00	0.00	finalizaMemLog
0.00	0.97	0.00	1	0.00	0.00	iniciaMemLog
0.00	0.97	0.00	1	0.00	0.00	le_dado_matriz_x
0.00	0.97	0.00	1	0.00	0.00	le_dado_matriz_y
0.00	0.97	0.00	1	0.00	0.00	parse_args

5.3.3 Analise

Analisando os dados temos uma função exponencial e que tem tempo de execução de 11 segundo, com a função atribuiValoresMatriz usando a maior parte deste tempo, essa análise para matrizes de tamanho 5000x5000

6. Análise Experimental

A analise deve ser feita de todas as 3 funções. A matriz escolhida para ser realizada a conta foi a matriz de tamanho 10x10, todos os arquivos usados estarão disponíveis no seguinte link: https://ufmgbr-my.sharepoint.com/:f:/g/personal/joaovitorferreira_ufmg_br/ElAdhxn2NqxOrdnAuhsL8VoBZk5ydqhLuusVPYXTOiOYKA?e=1Py1SW, devido ao tamanho extrapolar o limite máximo de envio do Moodle.

6.1 Soma

6.1.1 Acesso

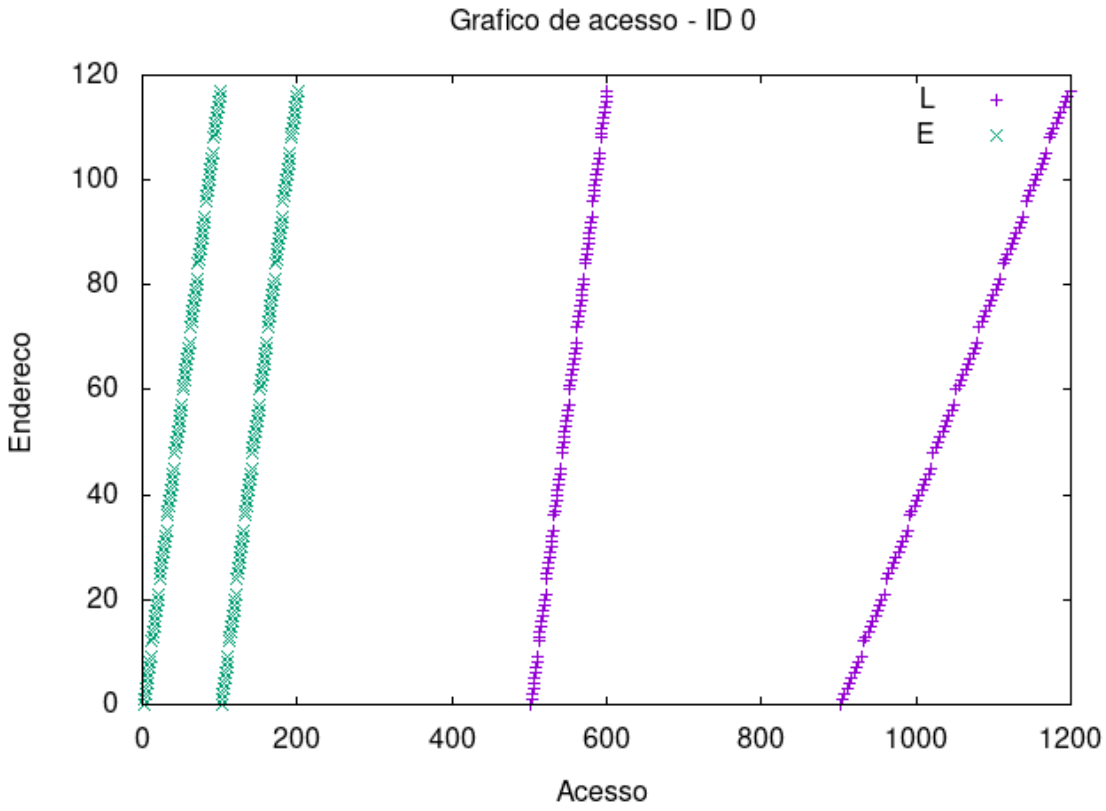


Grafico de acesso - ID 0
Endereco
Acesso
L
E
0
20
40
60
80
100
120
0
200
400
600
800
1000
1200

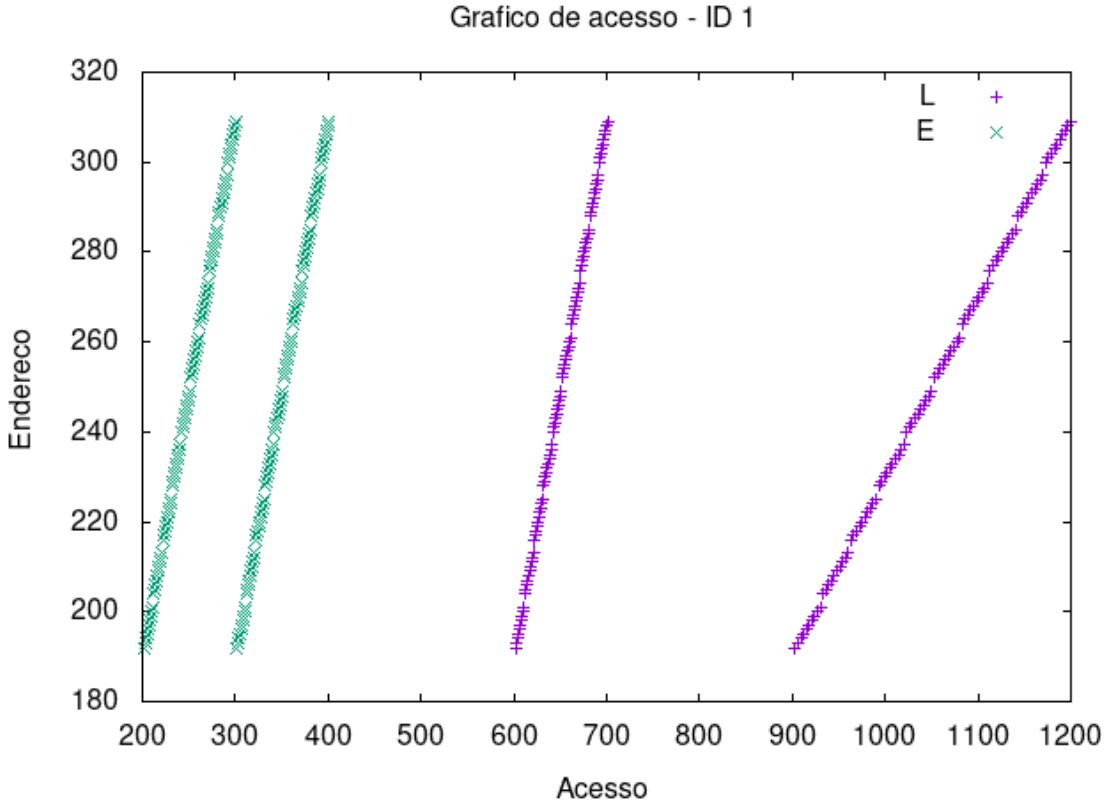
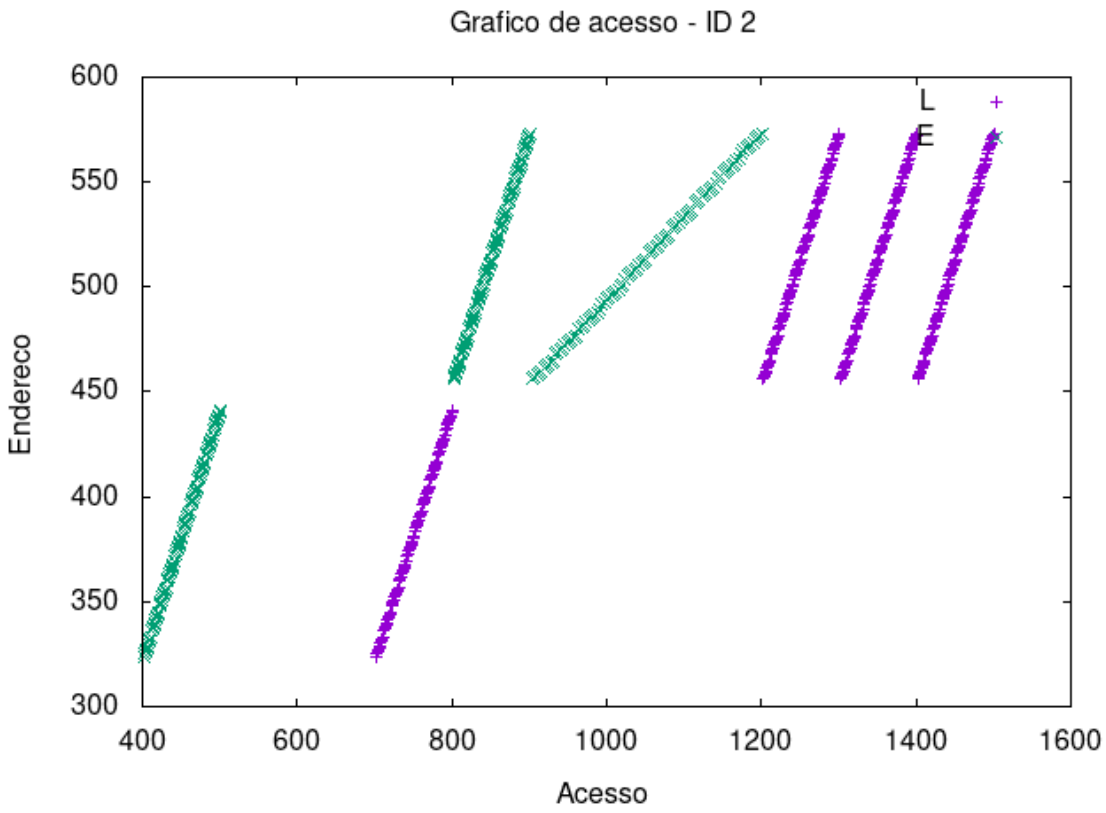
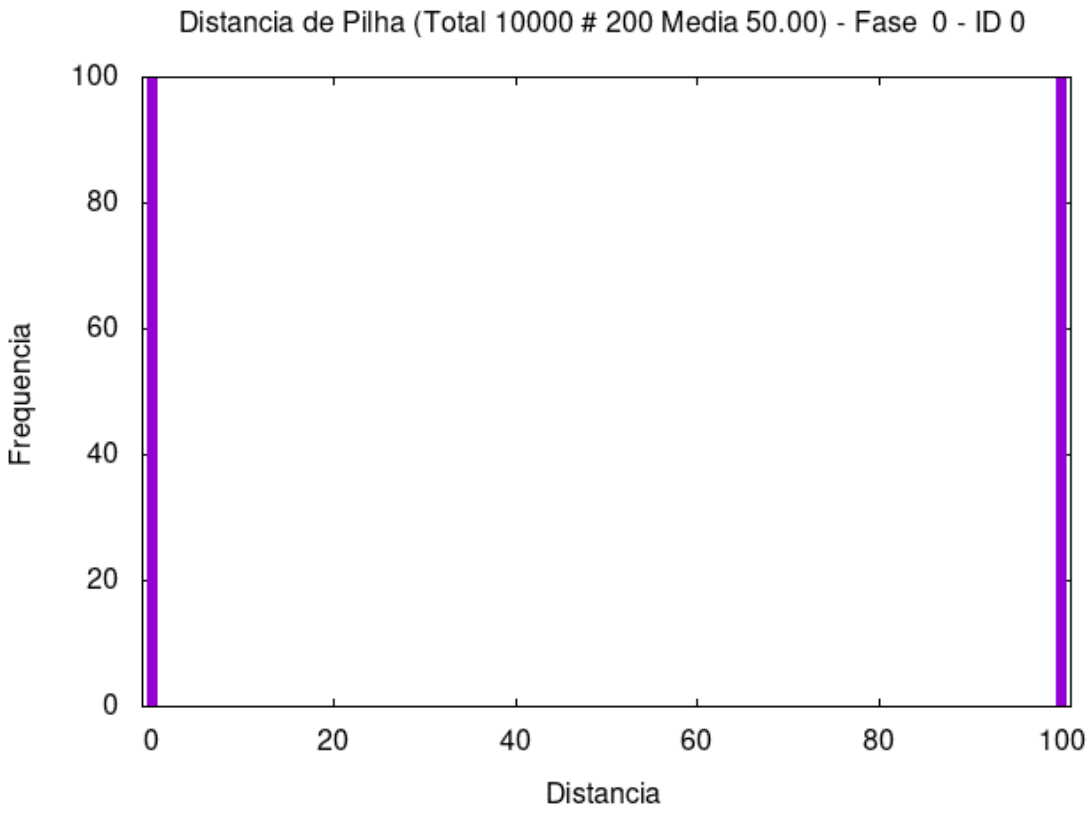


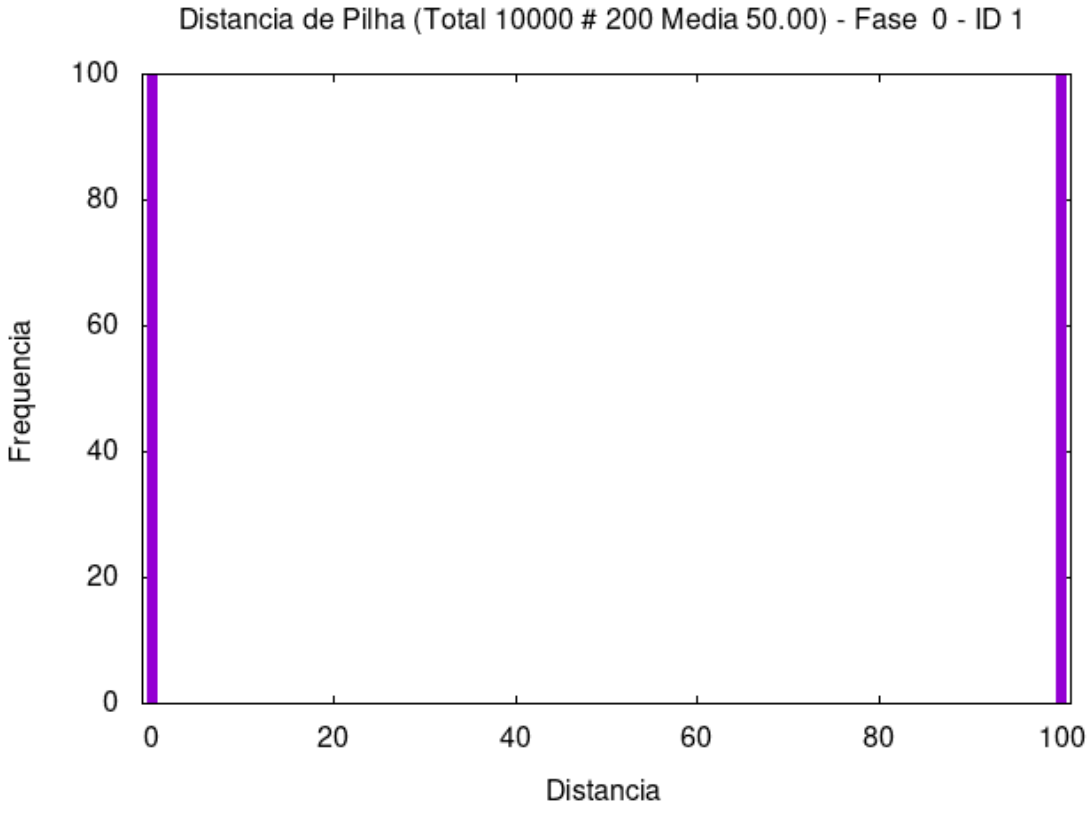
Grafico de acesso - ID 1
Endereco
Acesso
L
E
180
200
220
240
260
280
300
320
200
300
400
500
600
700
800
900
1000
1100
1200

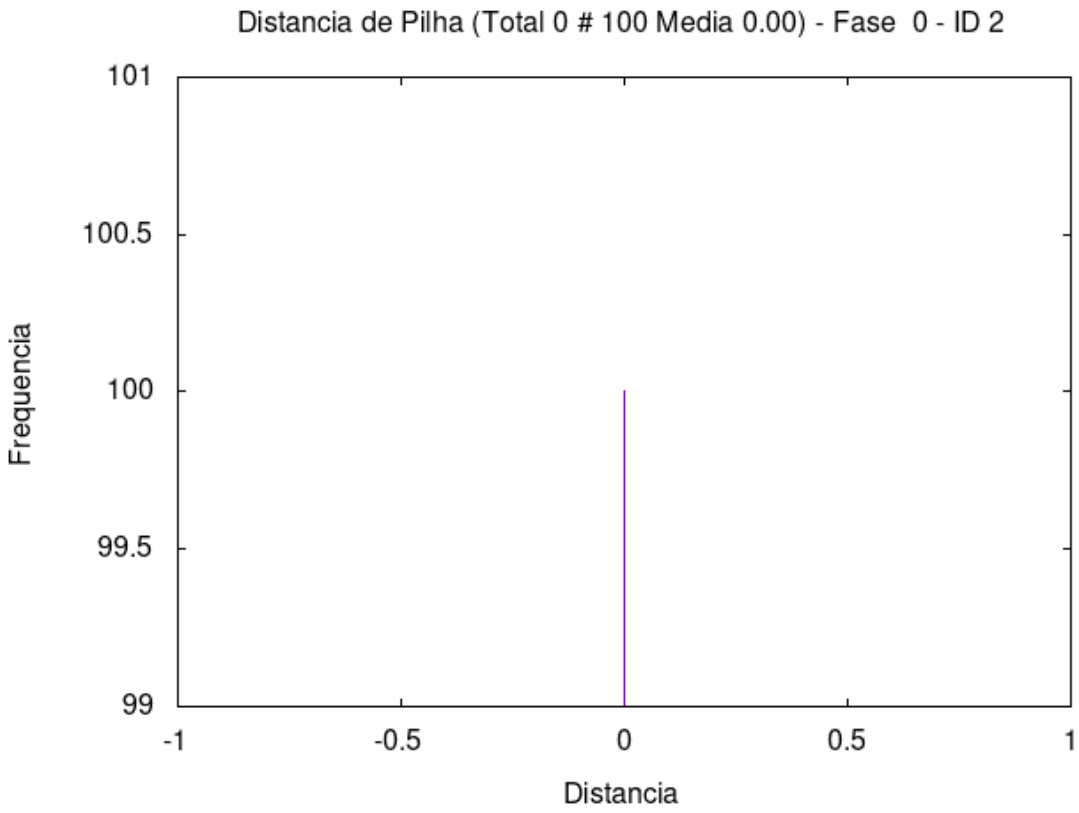


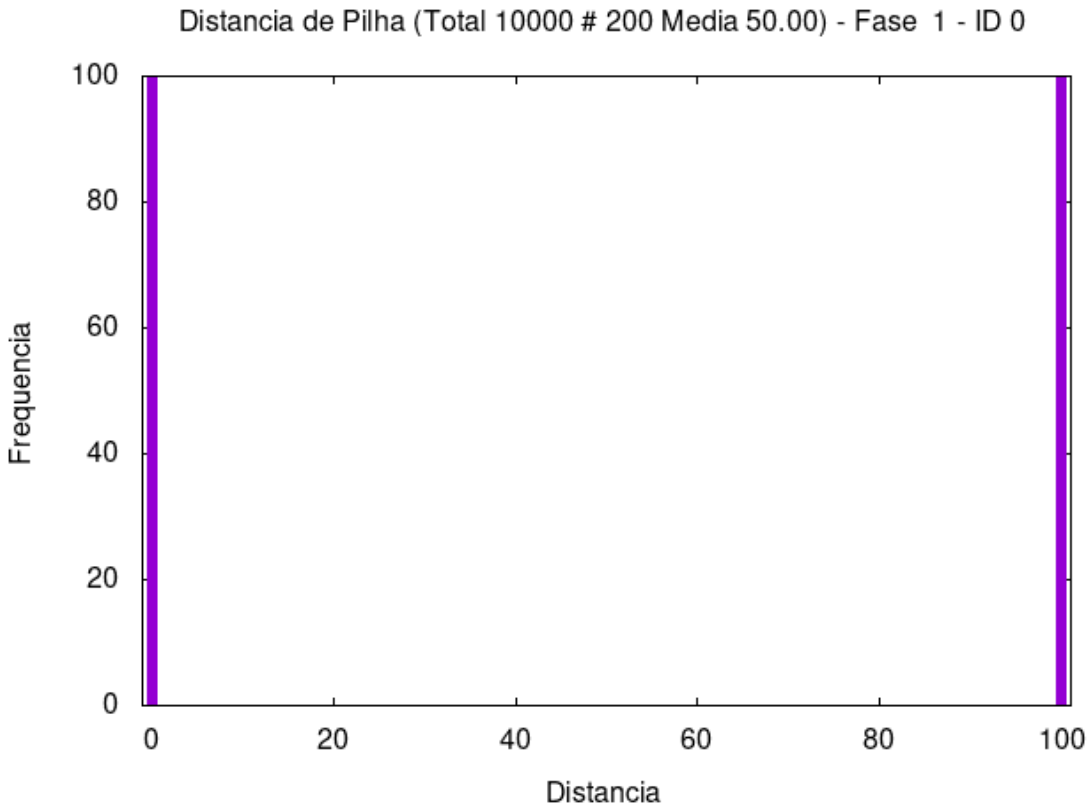
Analisando os dois primeiros gráficos vimos que há uma certa uniformidade na escrita e leitura das matrizes a e b. porém no 3 gráfico esse cenário é alterado devido a matriz c, ter sido preenchida e alocada de maneira diferente das outras além de herdar a soma dos valores dá outras duas

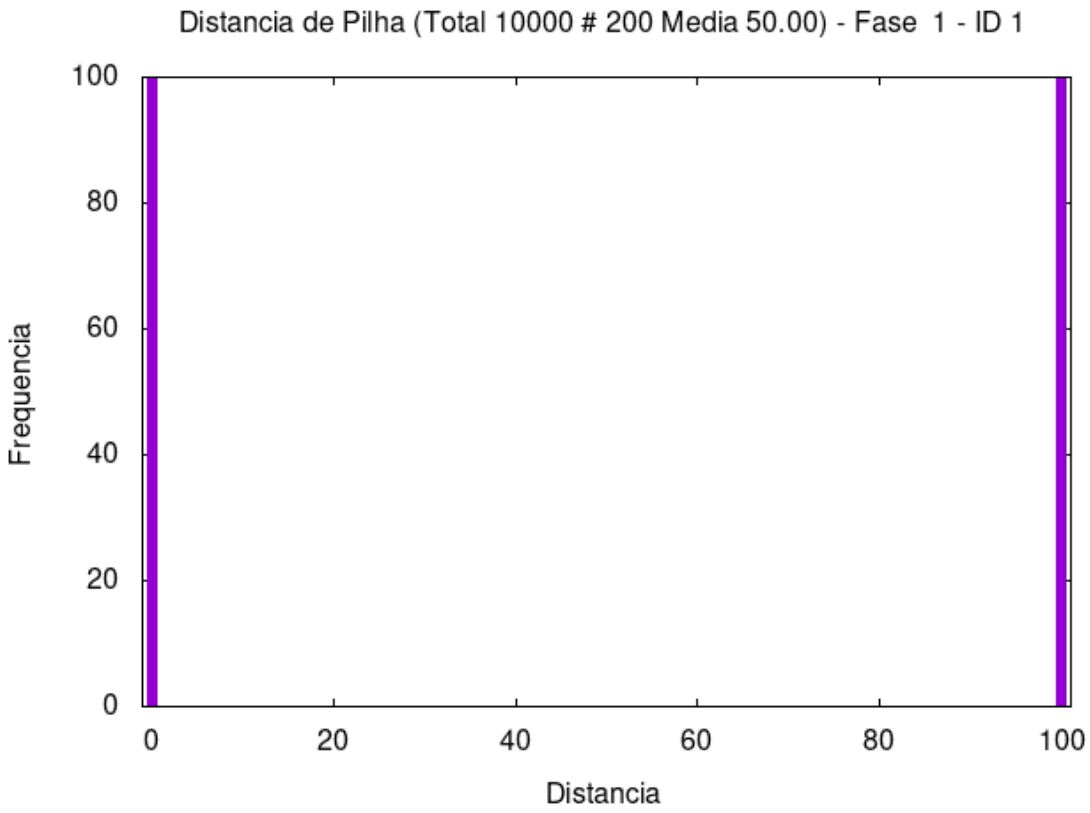
6.1.2 Distância de pilha

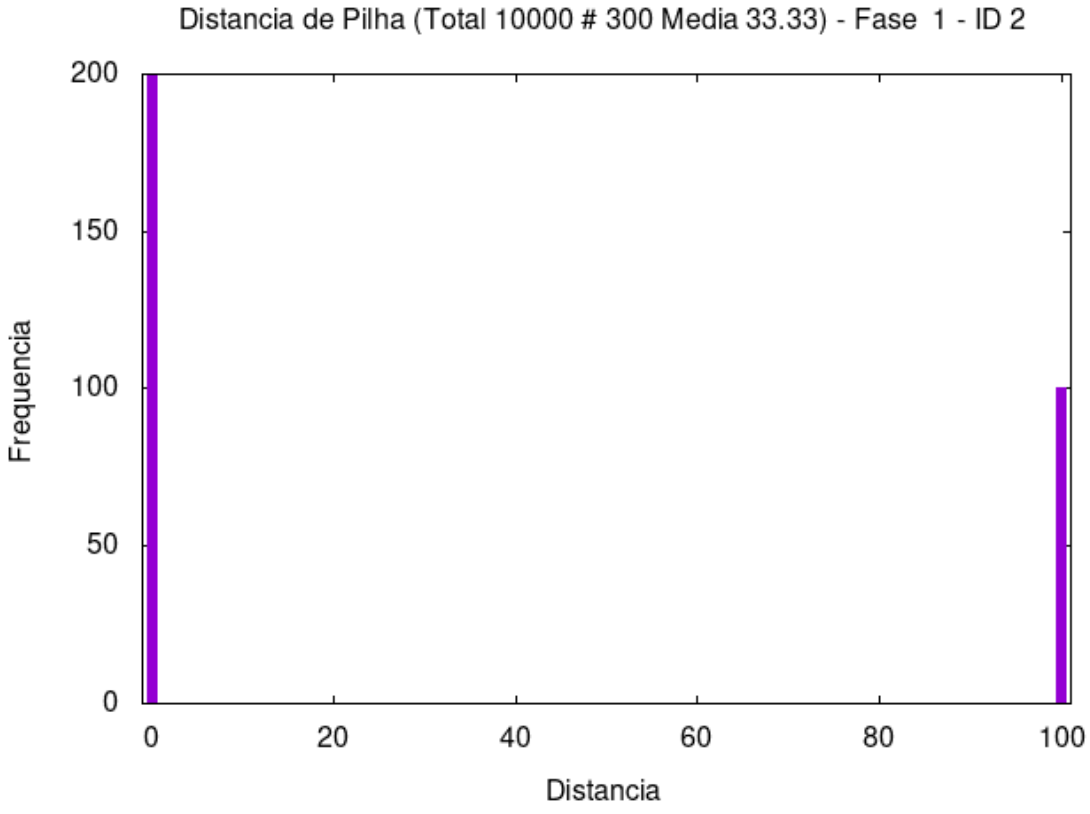

Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 0 - ID 0
Frequencia
Distancia
0
20
40
60
80
100

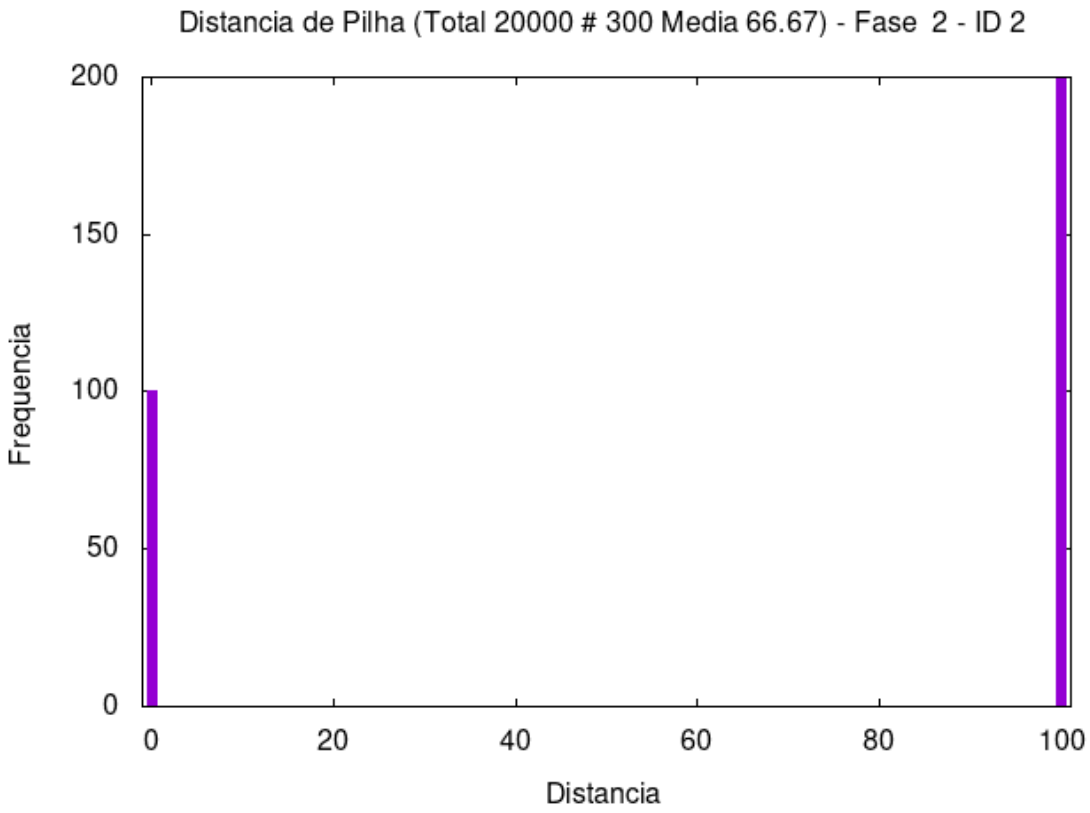

Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 0 - ID 1
Frequencia
Distancia
0
20
40
60
80
100


Distancia de Pilha (Total 0 # 100 Media 0.00) - Fase 0 - ID 2
Frequencia
101
100.5
100
99.5
99
-1
-0.5
0
0.5
1
Distancia


Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 1 - ID 0
Frequencia
100
80
60
40
20
0
0
20
40
60
80
100
Distancia

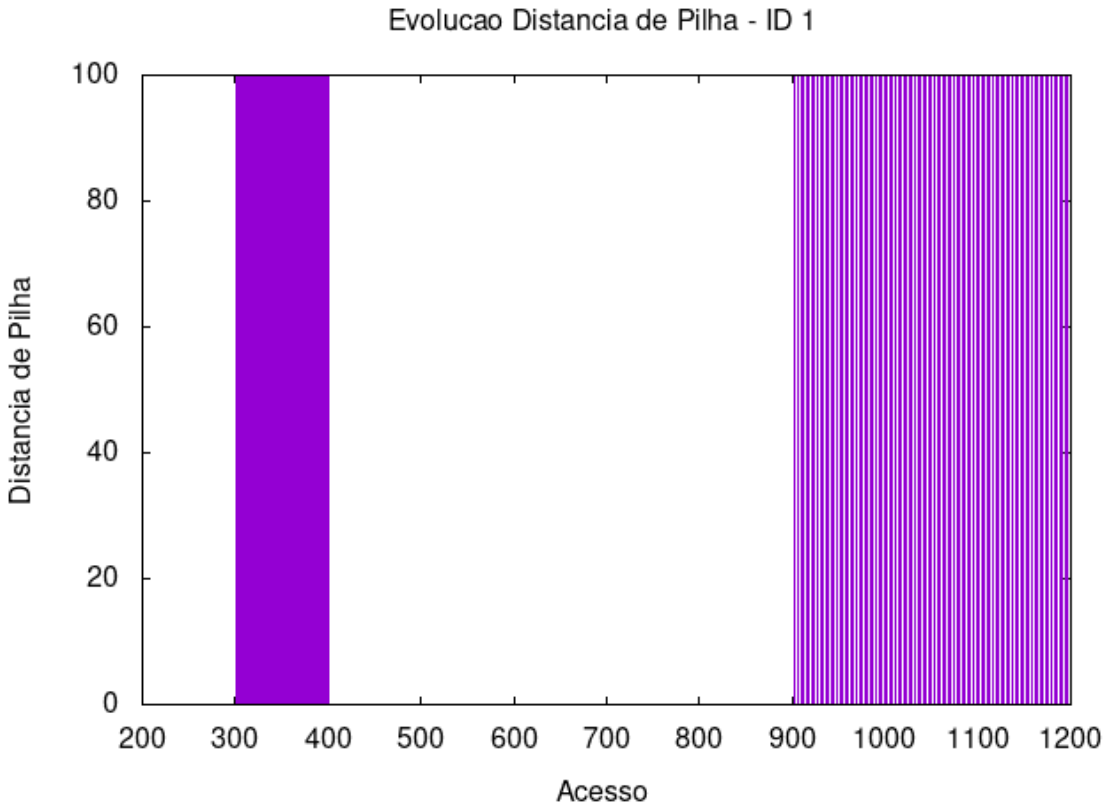

Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 1 - ID 1
Frequencia
Distancia
0
20
40
60
80
100

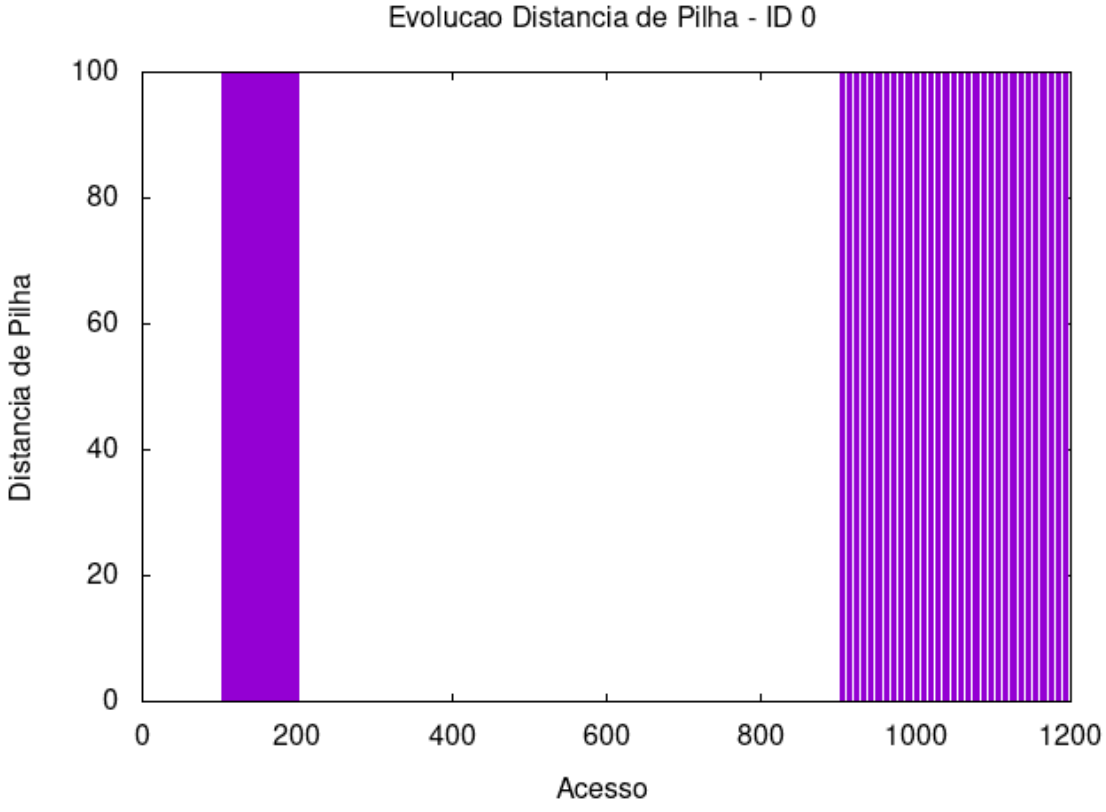

Distancia de Pilha (Total 10000 # 300 Media 33.33) - Fase 1 - ID 2
Frequencia
Distancia
0
50
100
150
200
20
40
60
80

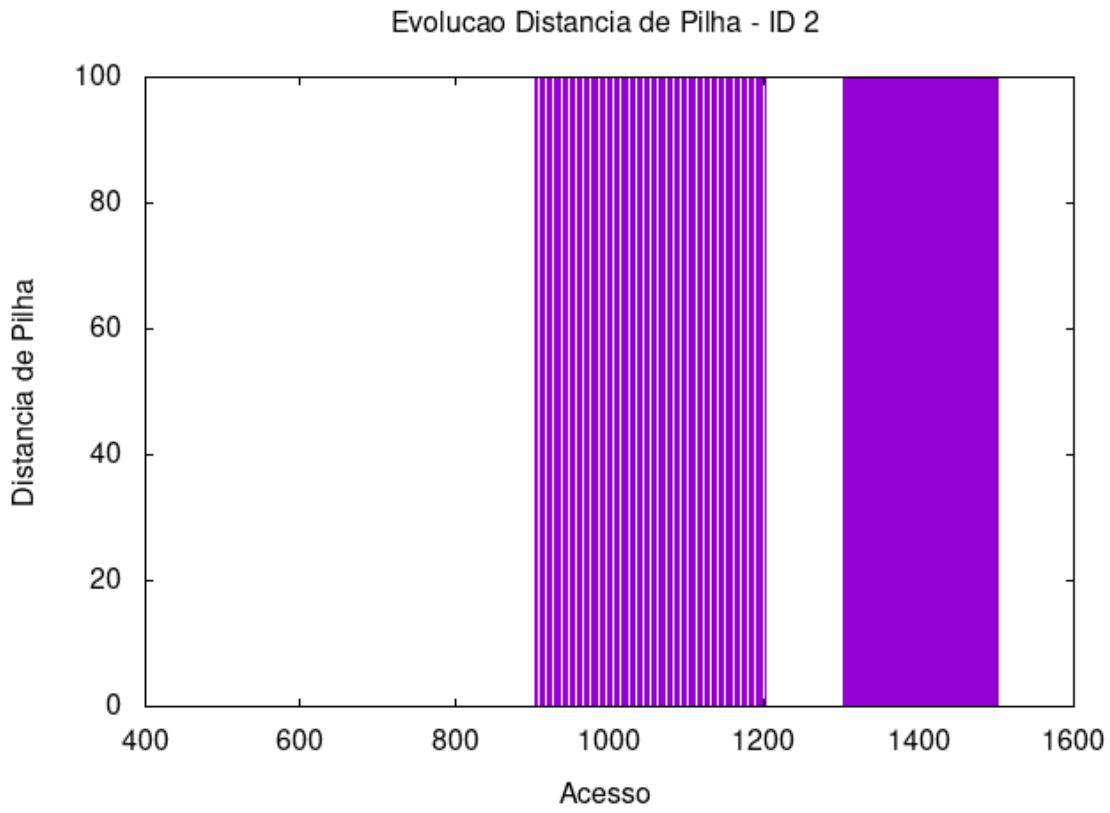


Na primeira fase temos que as matrizes são adicionadas na pilhas, sendo ‘a’, ‘b’ e ‘c’, respectivamente. Já nas fases seguintes vemos a somar ocorrer e ser transplantada para a matriz ‘c’.

6.1.3 Evolução distância de pilha


Evolucao Distancia de Pilha - ID 1
Distancia de Pilha
Acesso
0
20
40
60
80
100
200
300
400
500
600
700
800
900
1000
1100
1200


Evolucao Distancia de Pilha - ID 0
Distancia de Pilha
Acesso
0
20
40
60
80
100
0
200
400
600
800
1000
1200



A evolução é notada que no inicio dos dois primeiros gráfico vemos o acesso a matriz, e logo após vemos o acesso as linhas da matriz que são intercaladas se alinharmos os gráfico, no último gráfico vemos a inserção dos valores da soma nas posições da ultima matriz e a impressão

6.2 Multiplicação

6.2.1 Acesso

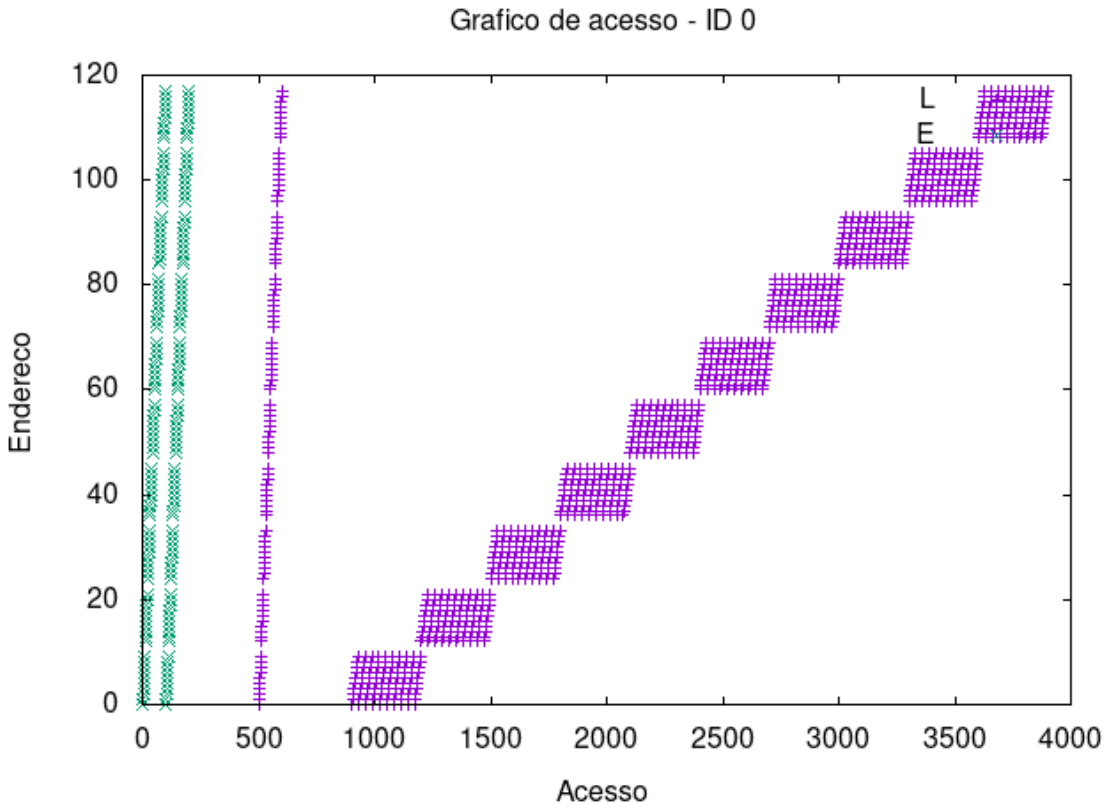


Grafico de acesso - ID 0
Endereco
Acesso
L
E
0
20
40
60
80
100
120
0
500
1000
1500
2000
2500
3000
3500
4000

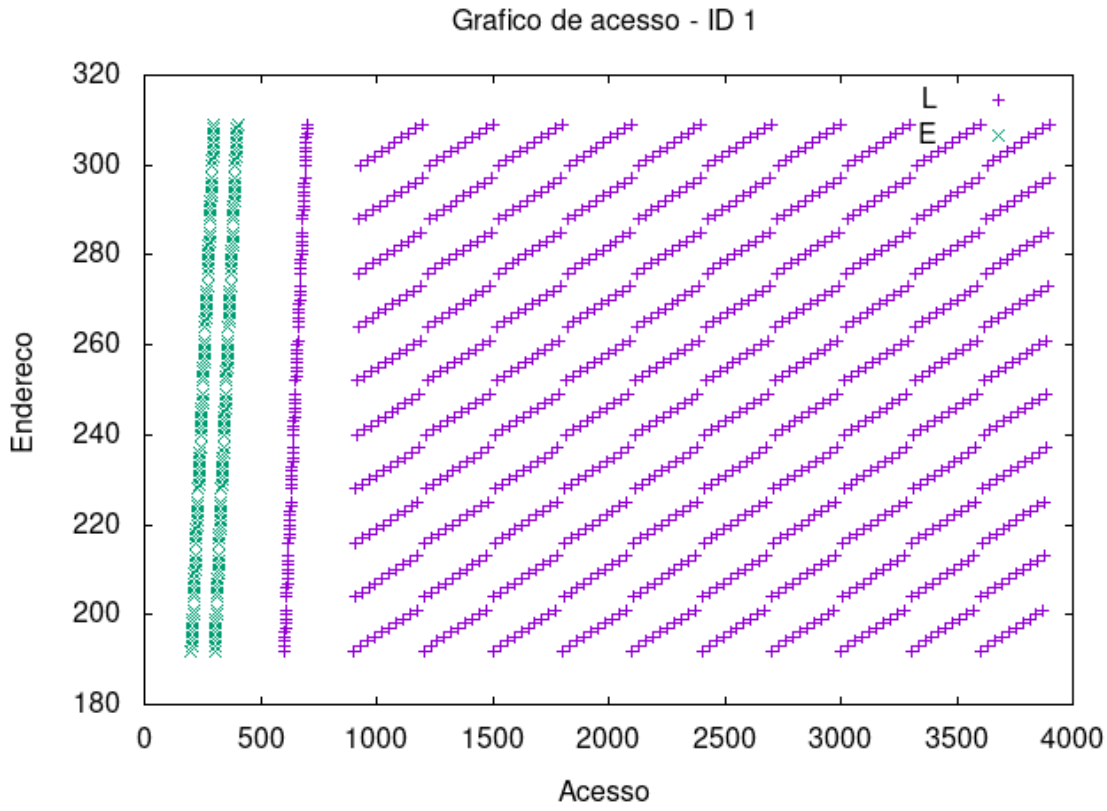
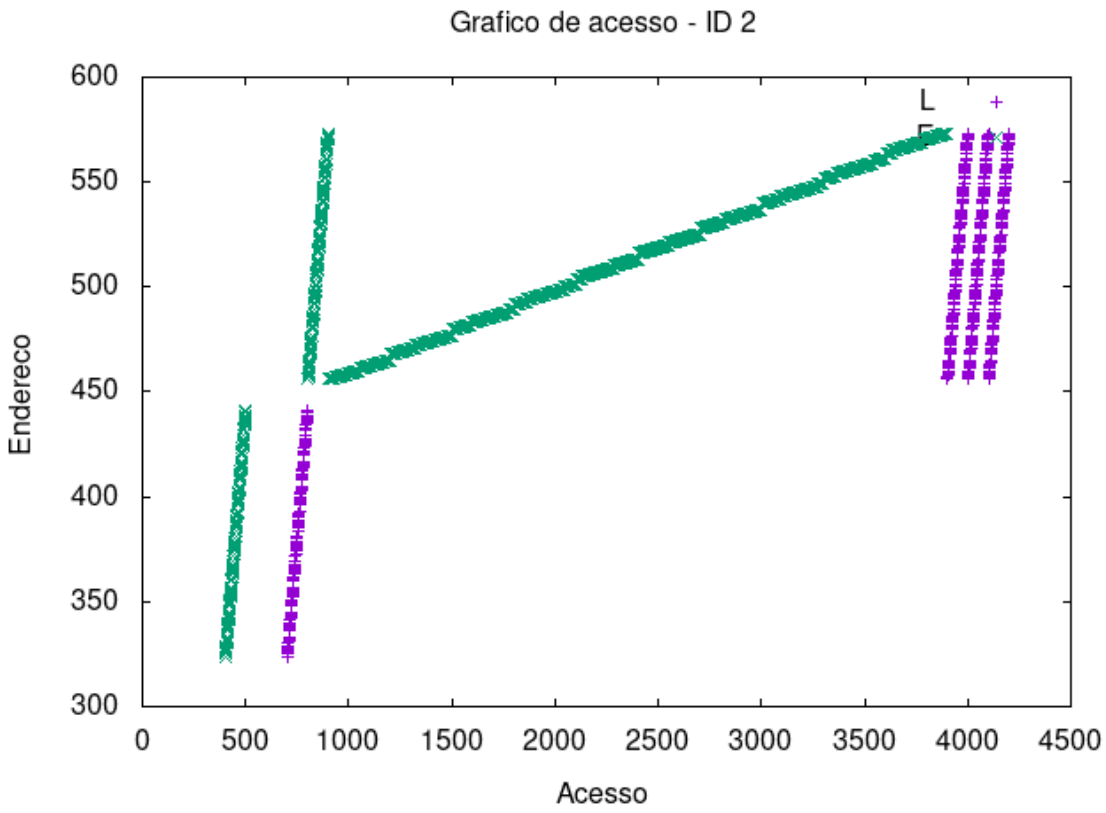
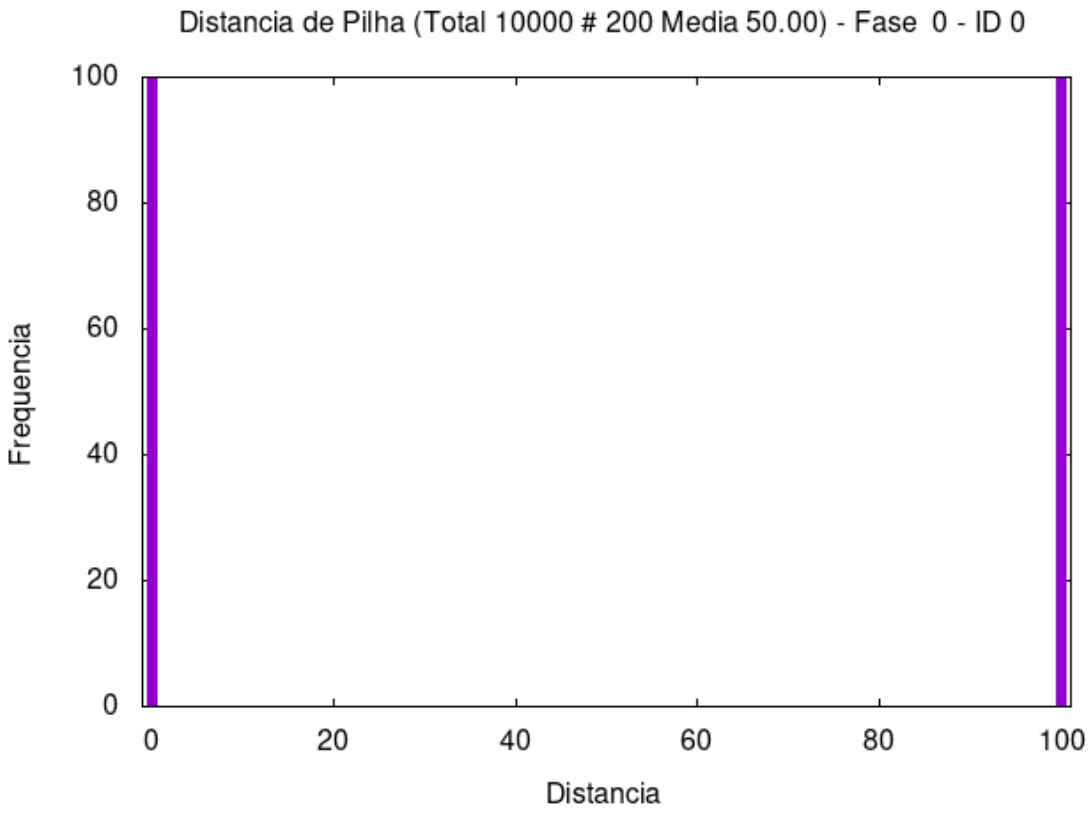


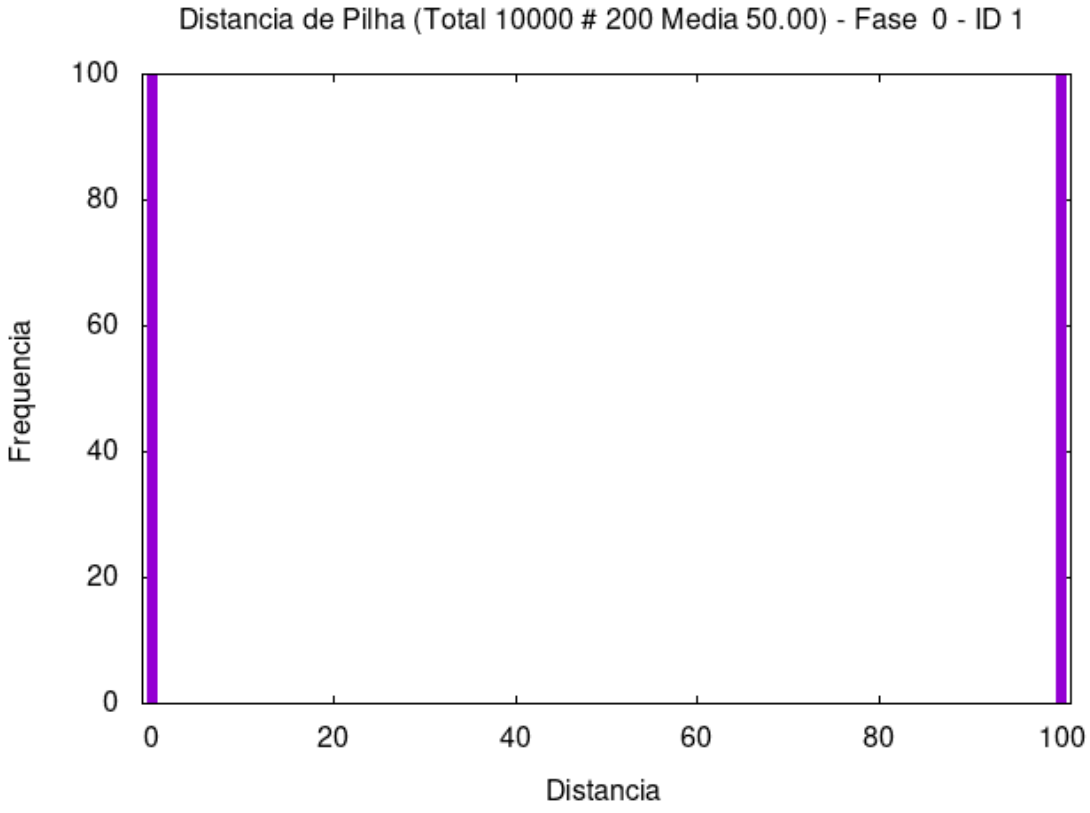
Grafico de acesso - ID 1
Endereco
Acesso
L
E
180
200
220
240
260
280
300
320
0
500
1000
1500
2000
2500
3000
3500
4000

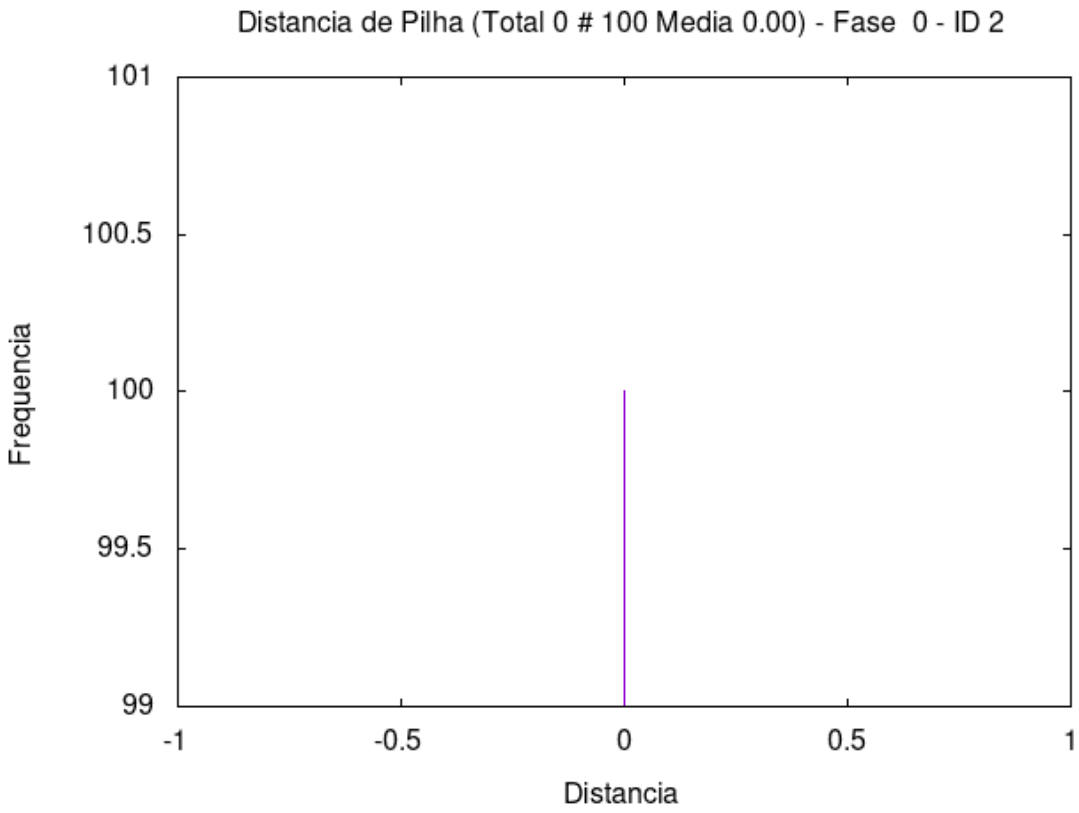


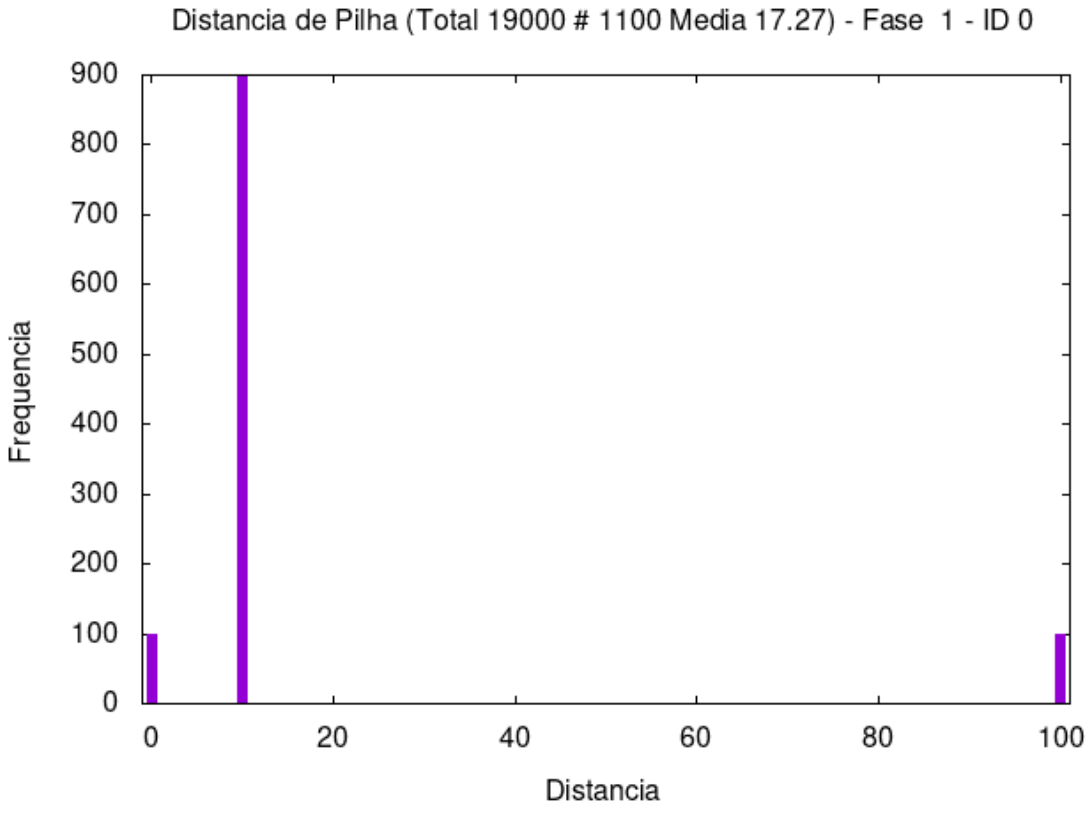
Vemos que o acesso do primeiro gráfico é extremamente organizado é possível ver cada posição da matriz claramente, visto que há somente o acesso. Já no sendo gráfico temos a leitura das posições intercaladas para realizar a multiplicação e soma dos valores. Já no ultimo vemos a escrita dos valores na matriz auxiliar e a leitura para impressão.

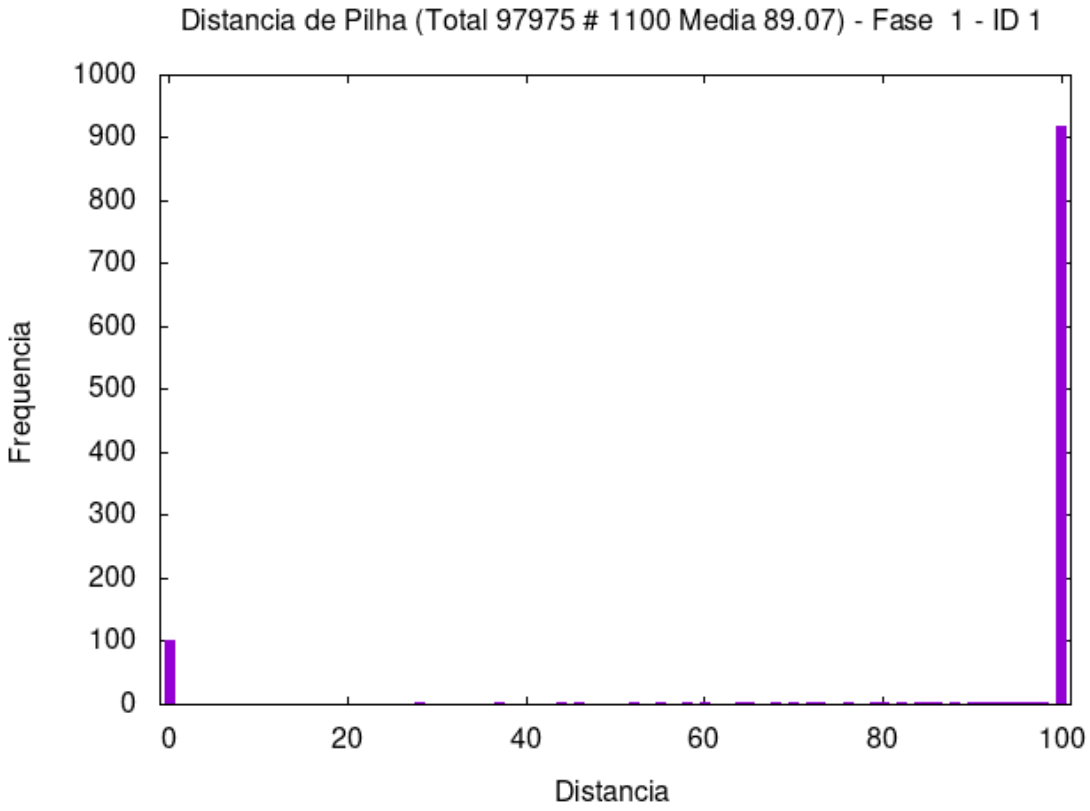
6.2.2 Distância de pilha

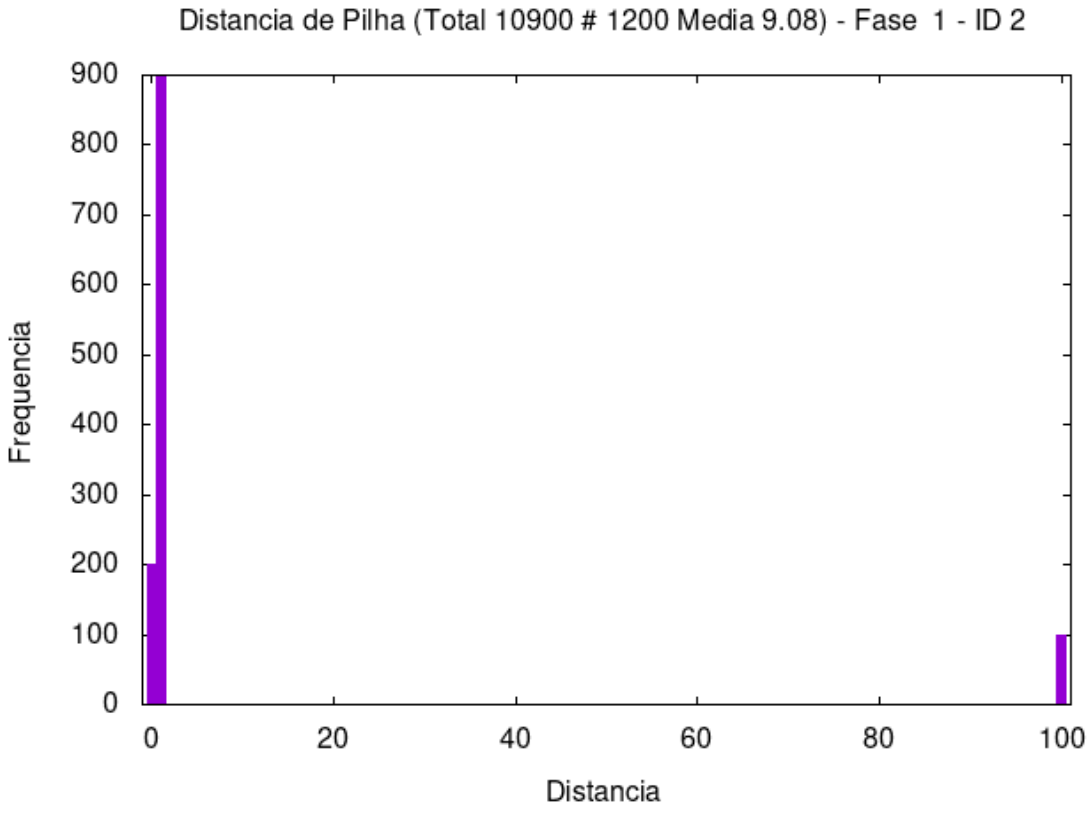

Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 0 - ID 0
Frequencia
Distancia
0
20
40
60
80
100

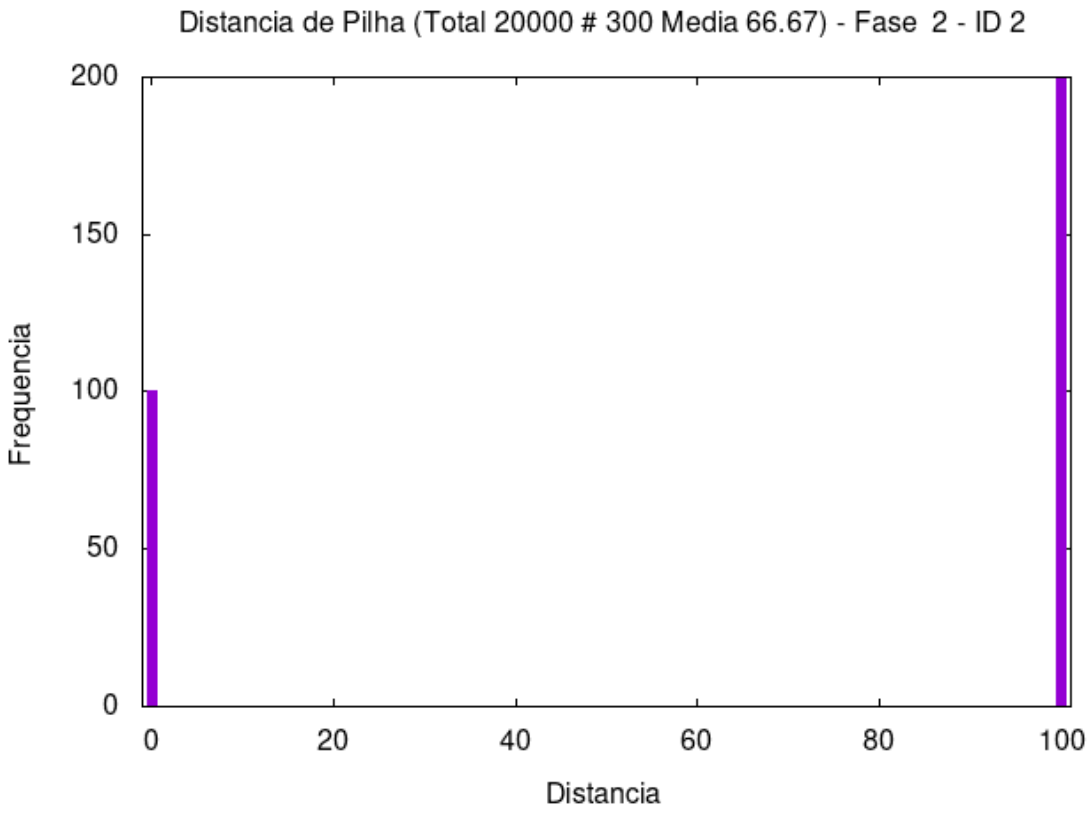

Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 0 - ID 1
Frequencia
Distancia
0
20
40
60
80
100


Distancia de Pilha (Total 0 # 100 Media 0.00) - Fase 0 - ID 2
Frequencia
Distancia
101
100.5
100
99.5
99
-1
-0.5
0
0.5
1


Distancia de Pilha (Total 19000 # 1100 Media 17.27) - Fase 1 - ID 0
Frequencia
Distancia
900
800
700
600
500
400
300
200
100
0
0
20
40
60
80
100

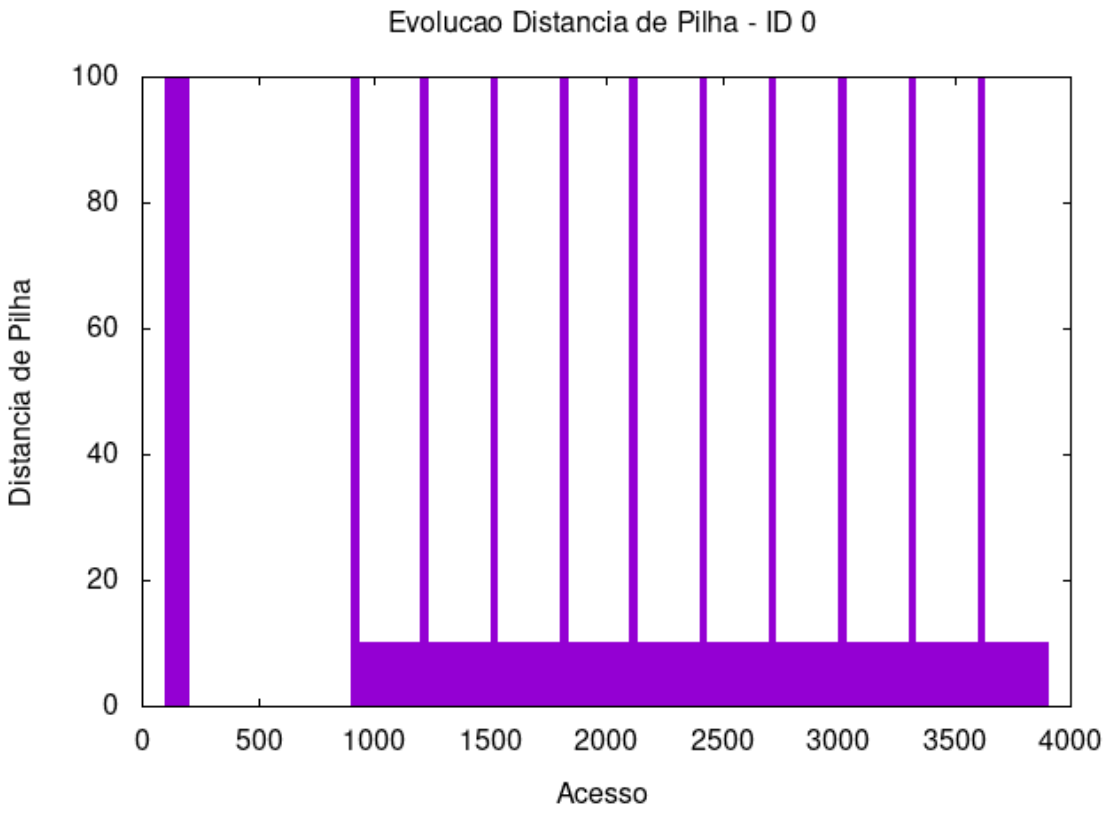

Distancia de Pilha (Total 97975 # 1100 Media 89.07) - Fase 1 - ID 1
Frequencia
Distancia
0
100
200
300
400
500
600
700
800
900
1000
0
20
40
60
80
100

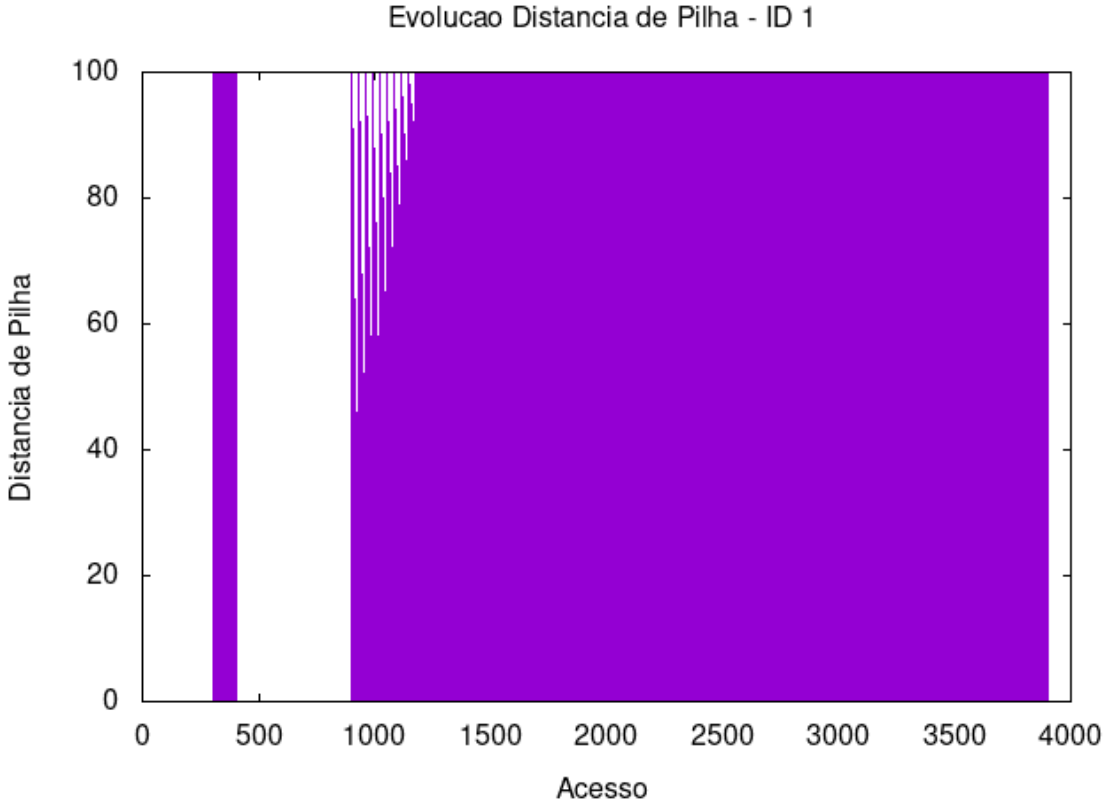

Distancia de Pilha (Total 10900 # 1200 Media 9.08) - Fase 1 - ID 2
Frequencia
Distancia
0
100
200
300
400
500
600
700
800
900
0
20
40
60
80
100

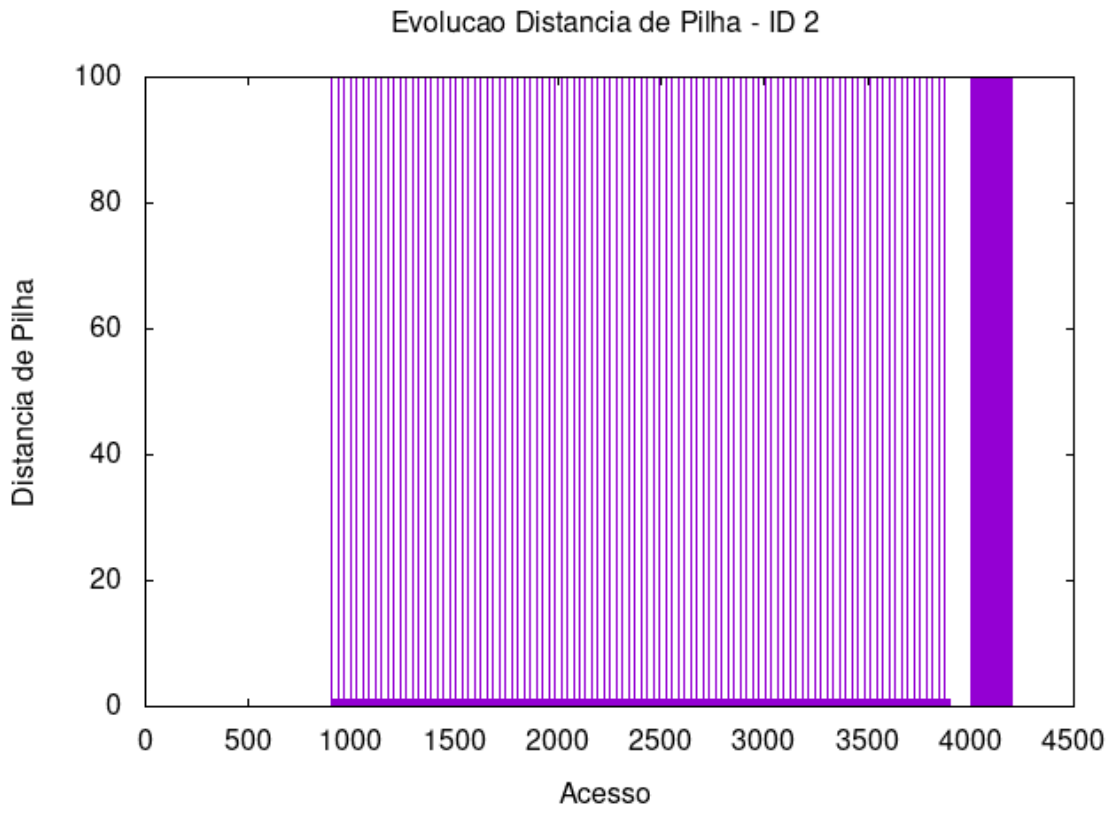


Na fase 1 notamos o acesso da posições de cada matriz. Na fase 1 vemos os acessos alternados de linhas e colunas para a realização da operação. E na fase 3 temos a impressão.

6.2.3 Evolução distância de pilha


Evolucao Distancia de Pilha - ID 0
Distancia de Pilha
Acesso
0
20
40
60
80
100
0
500
1000
1500
2000
2500
3000
3500
4000

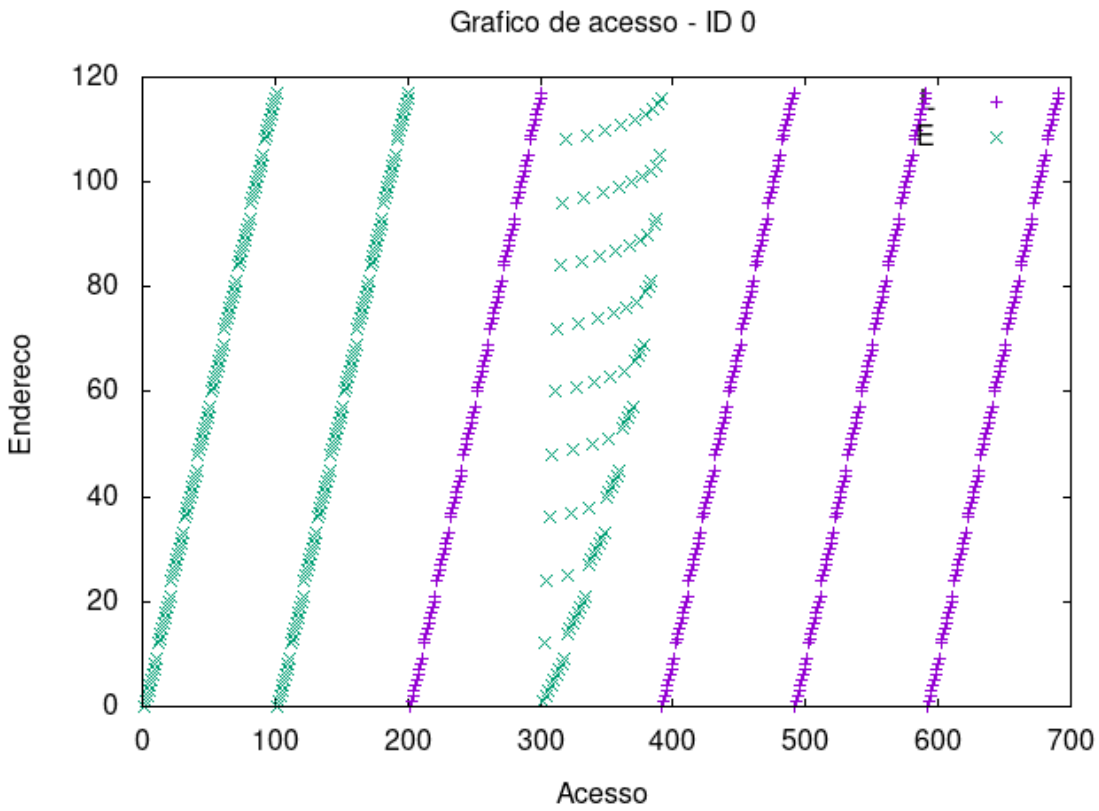

Evolucao Distancia de Pilha - ID 1
Distancia de Pilha
Acesso
0
20
40
60
80
100
0
500
1000
1500
2000
2500
3000
3500
4000



Na evolução é notado que o acesso é feito conforme a matriz é percorrida e uma coluna é acessada e depois a linha inteira, e depois a próxima linha. E no fim há uma alternância de redução de 1 posição. Por fim é visto a passagem do resultada para a matriz e a impressão.

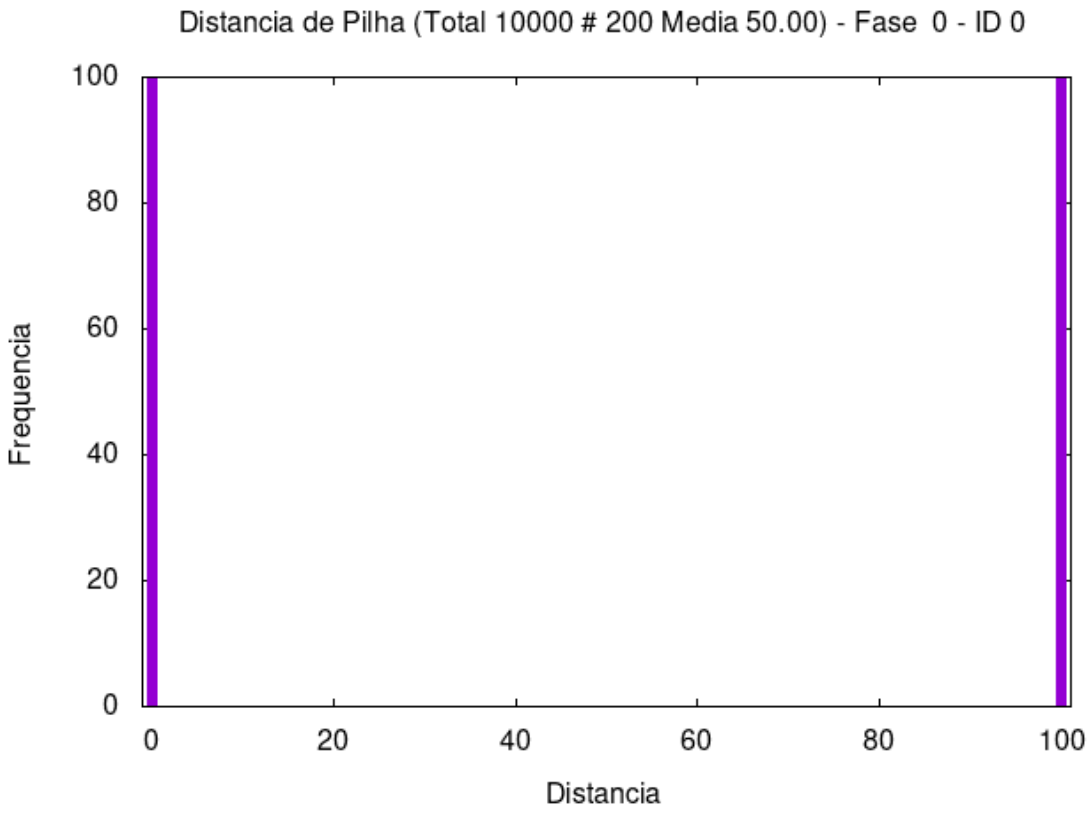
6.3 Transposição

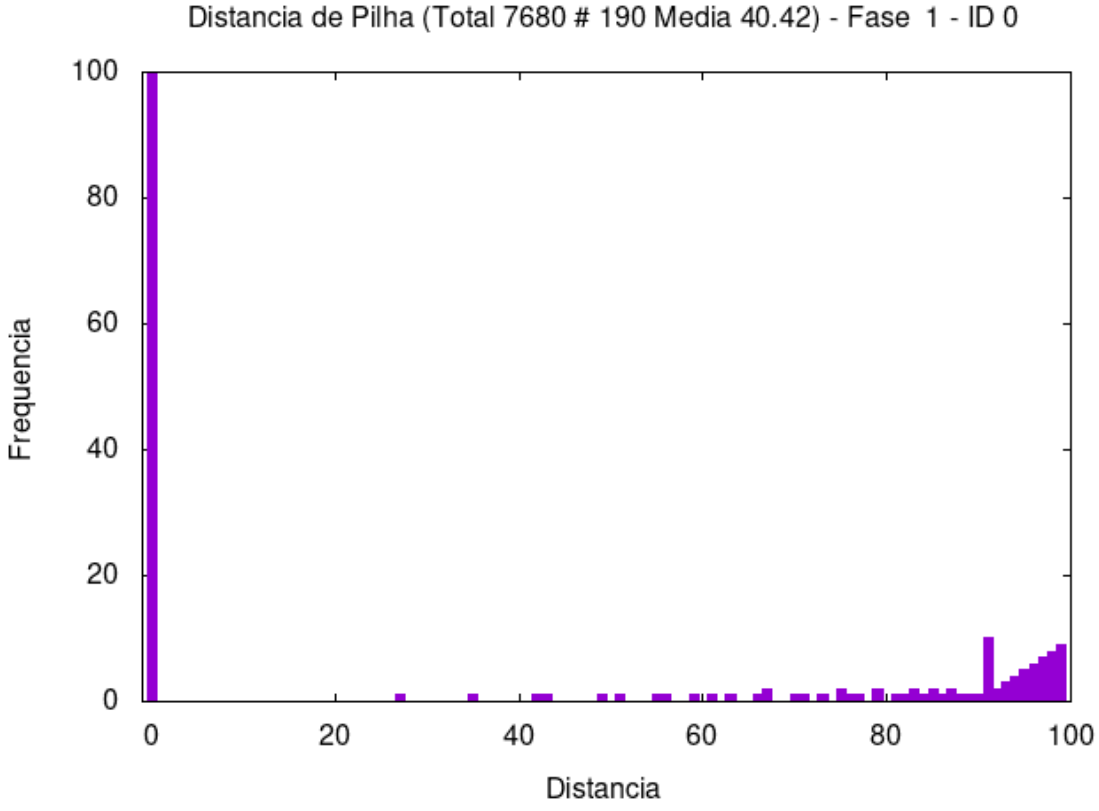
6.3.1 Acesso

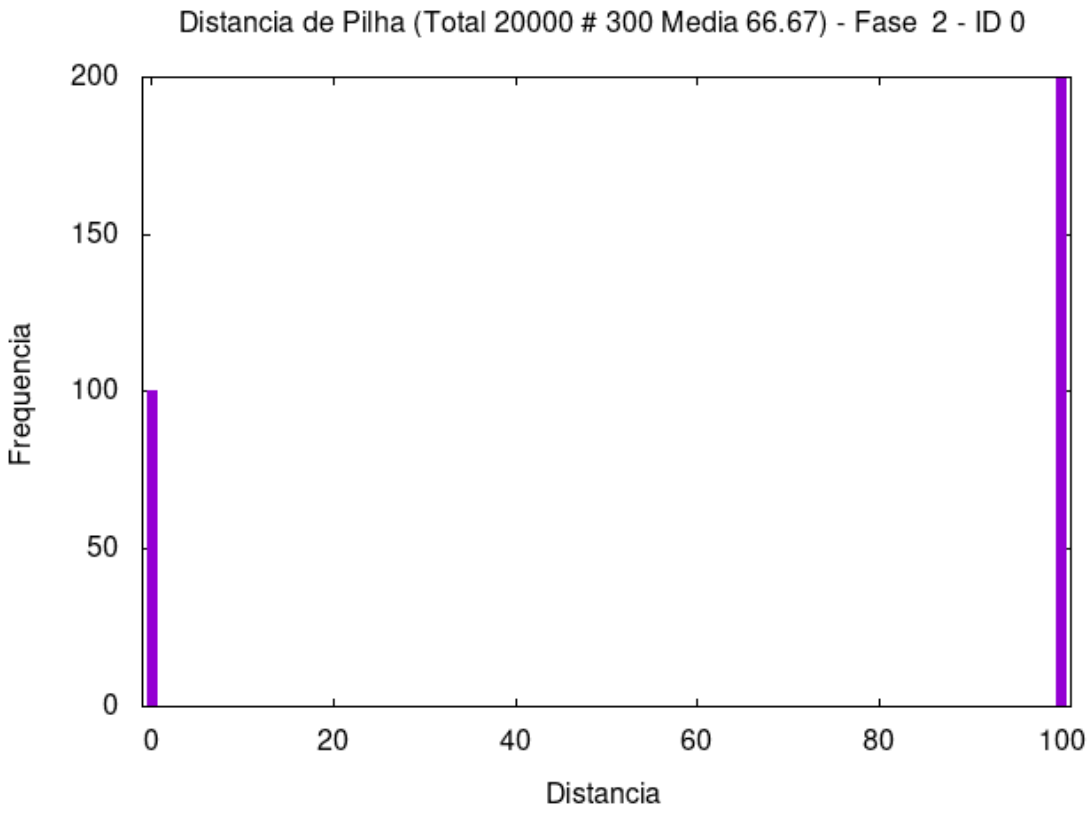


Observando este gráfico de acesso da transposição podemos notar que a leitura é feita linearmente em todos os pontos, já a escrita parece seguir uma lógica diferente por volta da faixa de acesso 300, essa anomalia é devido a funcionalidade da função de transposição, que faz a troca dos valores das linhas com os da colunas e etc. Logo a escrita pode ser notada que é feita seguindo a ordem de baixo para cima temos a primeira coluna, segundo até chegarmos na coluna e linha 10

6.3.2 Distância de pilha


Distancia de Pilha (Total 10000 # 200 Media 50.00) - Fase 0 - ID 0
Frequencia
Distancia
0
20
40
60
80
100

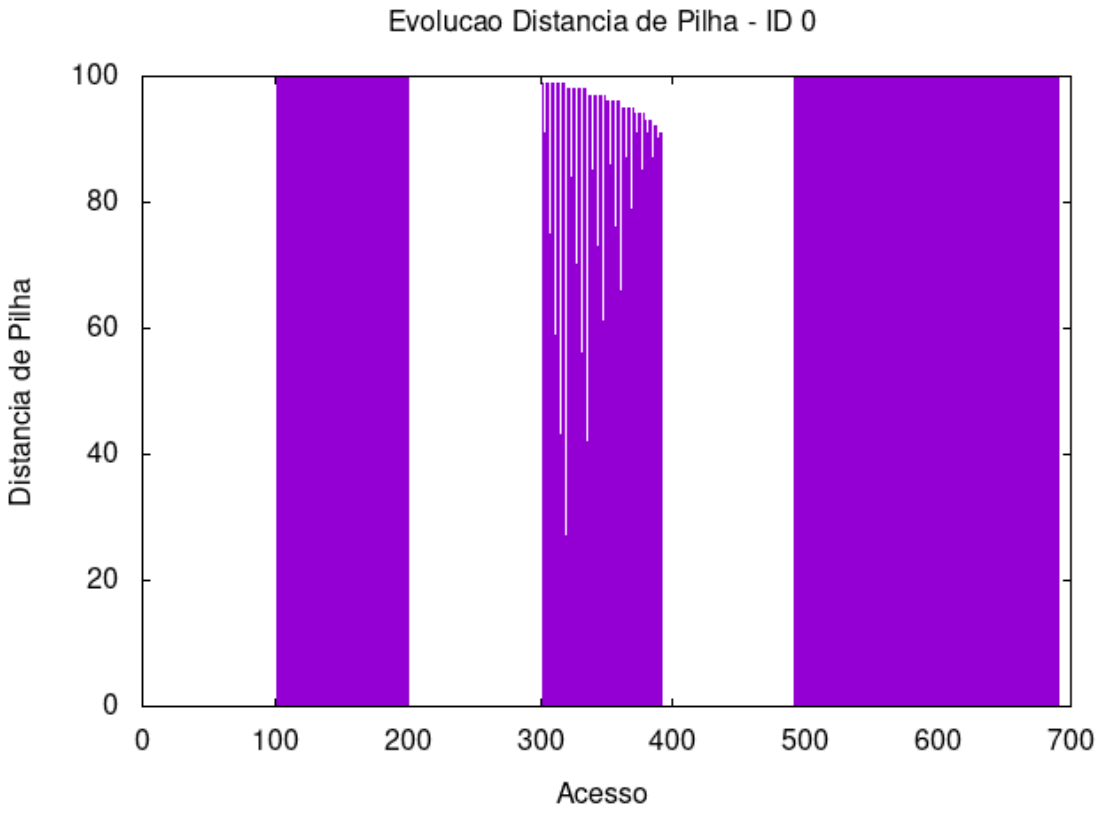

Distancia de Pilha (Total 7680 # 190 Media 40.42) - Fase 1 - ID 0
Frequencia
Distancia
0
20
40
60
80
100



No primeiro gráfico temos a alocação de todos os elementos da matriz na pilha, no segundo gráfico nota-se que temos a representação da troca dos valores pela parte da pilha crescente próxima da distância 90. No último gráfico temos a impressão de todos os valores acontecendo

6.3.3 Evolução de distância de pilha

.



Na evolução as mudanças são ainda mais notaveis, no primiero bloco temos a leitura de todos os elementos. Já no segunfo notamos que a evolução varia devido ao fato de que esta sendo feito o swap da linha e coluna, portanto ele percorre as colunas e as escreve nas linhas o que é notado nele, e por fim é representado a impressão

7. Conclusão

Após a implementação do programa, pode-se notar que havia três partes distintas para o˜ desenvolvimento do trabalho. A primeira era de se fazer a conversão e alocação das matrizes dinamicamente. A segunda adaptar o preenchimento das matrize para serem lidas por arquivos que são passados na

linha de comando do programa e por fim implementar o analisamen para gerar os resultados computacionais.

O grande ponto deste trabalho era de se conseguir implementar todos estes requisitos e resultar num programa considerado eficiente e que rode de forma satisfatório, além do fato de conseguir realizar as principais operações nas matrizes que estão alocadas dinamicamente na memória.

Contudo fica visível a necessidade do uso de técnicas de estrutura de dados para e de teste para poder inferir sobre a qualidade da implementação visto que está nos possibilitam compreender o real funcionamento e custo que o programa demanda. Também é notável que o programa possui um funcionamento que lembra o do exponencial, visto que o custo é drasticamente incrementado a cada aumento dos valores de entrada. No entanto essa condição não pode ser efetivamente comprovada.

8. Bibliografia

Ziviani, N. (2006). *Projetos de Algoritmos com Implementações em Java e C++:˜ Capítulo 3: Estruturas de Dados Básicas´* . Editora Cengage.

9. Instruções para compilação e execução

A compilação do código pode ser feita através do Makefile, basta adicionar os arquivos contendo a quantidade de linhas e colunas da matriz e depois de uma quebra de linhas os valores da posições. Estes arquivos foram nomeados de m1.txt e m2.txt e executam todas as operações com as matrizes. E retornam na pasta os arquivos res1.txt, res2.txt e res3.txt que são respectivamente a saída da soma, multiplicação e transposição das matrizes de entrada.

Caso opte por realizar a analise de desempenho, reali8zaroutros testes, gerar os gráficos de acesso, distância de pilha evolução de pilha e os arquivos gprof deve-se entrar no arquivo Makefile e descomentar todas as linhas que contenham o “#”. O caso especial dos gráficos se faz necessário utilizar o analisamem que é disponibilizado na aba de Material adicional da Metaturma no Moodle de Estrutura de Dados 2022/1