Aula prática 2 – Estrutura de Dados - UFMG

Aluno: Luiz Felipe de Sousa Faria - 2020027148

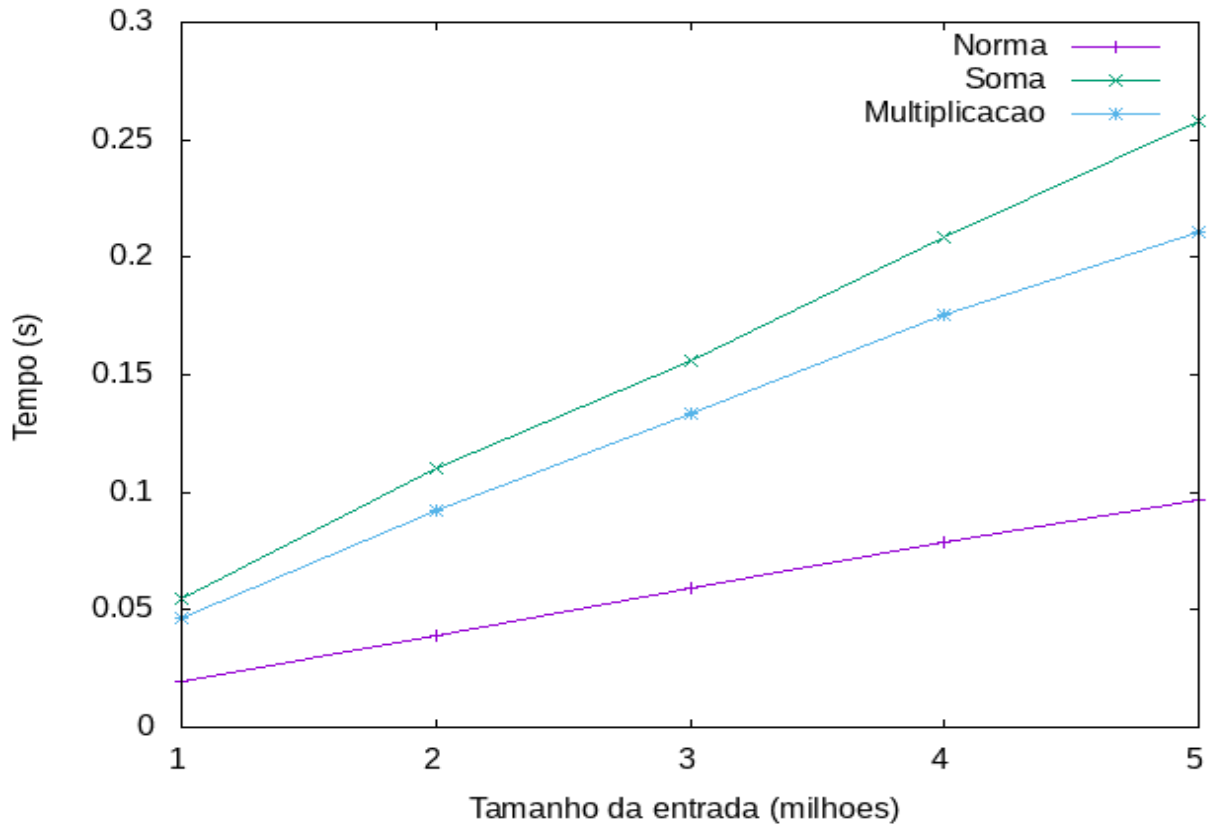
Nota: o código fonte analisado não foi modificado. No entanto, foram identificadas algumas possíveis melhorias para diminuir tempo de execução e memória alocada. As mesmas são citadas durante as análises.

1. Plano de experimento - Resultados Computacionais

Esse experimento analisará o desempenho de três operações vetoriais: soma, norma e multiplicação dos termos. A ideia é comprovar, através de programas que mensuraram o desempenho, que a complexidade dessas funções é realmente linear, conforme análises prévias, e depurar cada função e suas operações.

Os experimentos foram feitos com diferentes tamanhos de entrada para cada uma das funções analisadas. Essas entradas tinham tamanhos na ordem dos milhões. No gráfico abaixo, é possível ver a relação entre **tamanho da entrada** x **tempo gasto para execução** das três funções. O tamanho da entrada varia entre 1-5 milhões. Para a depuração, foi considerado apenas a entrada de tamanho 5 milhões.

2. Gráfico de desempenho



3. Análise de desempenho

Conforme apresentado no gráfico, é possível perceber que as três funções possuem uma linearidade conforme o tamanho da entrada é aumentado. Isso está de acordo com a análise do algoritmo de cada deles.

Todos os algoritmos possuem complexidade *O(n),* na qual tanto no melhor quanto no pior caso, ele precisará percorrer um número **N** (referente ao tamanho dos dois vetores, que será o mesmo em ambos) para realizar a devida operação em cada posição. O gráfico também revela um tempo de execução maior para as funções “soma” e “multiplicação” em relação a “norma”.

4. Resultados da depuração do desempenho

4.1. Soma

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
55.10	0.11	0.11	4	27.55	27.55	inicializaVetorNulo
25.05	0.16	0.05	4	12.52	12.52	acessaVetor
10.02	0.18	0.02	2	10.02	37.57	inicializaVetorAleatorio
10.02	0.20	0.02	1	20.04	47.59	somaVetores
0.00	0.20	0.00	4	0.00	0.00	criaVetor
0.00	0.20	0.00	3	0.00	0.00	defineFaseMemLog
0.00	0.20	0.00	3	0.00	0.00	destroiVetor
0.00	0.20	0.00	1	0.00	0.00	clkDifMemLog
0.00	0.20	0.00	1	0.00	0.00	desativaMemLog
0.00	0.20	0.00	1	0.00	0.00	finalizaMemLog
0.00	0.20	0.00	1	0.00	0.00	iniciaMemLog
0.00	0.20	0.00	1	0.00	0.00	parse_args

4.2. Multiplicação

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
50.09	0.07	0.07	3	23.38	23.38	inicializaVetorNulo
28.62	0.11	0.04	3	13.36	13.36	acessaVetor
14.31	0.13	0.02	1	20.04	20.04	produtoInternoVetores
7.16	0.14	0.01	2	05.01	28.38	inicializaVetorAleatorio
0.00	0.14	0.00	3	0.00	0.00	criaVetor
0.00	0.14	0.00	3	0.00	0.00	defineFaseMemLog
0.00	0.14	0.00	3	0.00	0.00	destroiVetor
0.00	0.14	0.00	1	0.00	0.00	clkDifMemLog
0.00	0.14	0.00	1	0.00	0.00	desativaMemLog
0.00	0.14	0.00	1	0.00	0.00	finalizaMemLog
0.00	0.14	0.00	1	0.00	0.00	iniciaMemLog
0.00	0.14	0.00	1	0.00	0.00	parse_args

4.3. Norma

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
40.07	0.02	0.02	1	20.04	20.04	acessaVetor
40.07	0.04	0.02	1	20.04	20.04	inicializaVetorNulo
20.04	0.05	0.01	1	10.02	10.02	normaVetor
0.00	0.05	0.00	3	0.00	0.00	defineFaseMemLog
0.00	0.05	0.00	1	0.00	0.00	clkDifMemLog
0.00	0.05	0.00	1	0.00	0.00	criaVetor
0.00	0.05	0.00	1	0.00	0.00	desativaMemLog
0.00	0.05	0.00	1	0.00	0.00	destroiVetor
0.00	0.05	0.00	1	0.00	0.00	finalizaMemLog
0.00	0.05	0.00	1	0.00	0.00	iniciaMemLog
0.00	0.05	0.00	1	0.00	20.04	inicializaVetorAleatorio
0.00	0.05	0.00	1	0.00	0.00	parse_args

5. Análise da depuração do desempenho

5.1. Soma

Na função de soma vetorial, são feitas 6 chamadas para funções que inicializam vetores, sendo duas delas feitas com “inicializaVetorAleatorio” (que também chama internamente a função “inicializaVetorNulo”), uma na função “somaVetores”, que garante a consistência do vetor que vai ser gerado da soma, e o restante, na criação dos parâmetros da função. Essas operações compreendem, aproximadamente, 65% do tempo total de execução da função. Poderia ser reduzido criando o vetor C apenas 1 vez (ou na chamada da função, ou internamente nela). O restante do tempo é gasto, majoritariamente, para executar funções de *log* (“acessaVetor”) e para executar a função de somar os vetores. As outras chamadas acabaram tendo pouca significância para o tempo da função, considerando a entrada dada.

5.2. Multiplicação

Na função de multiplicação vetorial, são feitas apenas 5 chamadas para inicialização do vetor. O comportamento de inicialização segue o mesmo da função que soma vetores. No entanto, não é feita a verificação da consistência do vetor a ser escrito, por isso há uma chamada a menos. Isso explica o motivo dessa função apresentar um tempo de execução menor do que a função de soma, como visto no gráfico da sessão 2. O restante do tempo é despendido com funções de *log* e a própria função de multiplicar os vetores.

5.3. Norma

Na função que calcula a norma do vetor, é possível perceber, através da análise, o motivo do tempo de execução dessa função ser menor do que as outras duas. Nela, é

feita apenas uma chamada de inicialização de vetor, que seria o vetor na qual queremos calcular a norma. Com isso, é possível reduzir significativamente o tempo de execução da mesma

5.4. Conclusão

Após a análise das três funções, é possível perceber possíveis melhorias a serem feitas no código. A principal delas seria na redução de chamadas para funções que criam vetores. Na função soma, por exemplo, o vetor na qual será escrito a soma dos outros dois vetores é inicializado duas vezes. Além disso, na função de multiplicação, há uma inicialização do vetor **C** desnecessária, já que o mesmo não é utilizado.

A função "inicializaVetorAleatório" poderia ser refatorada para conseguir gerar um vetor aleatório sem ter que gerar um vetor nulo antes, pois com isso, "percorreríamos" o vetor de tamanho **N** apenas uma vez.

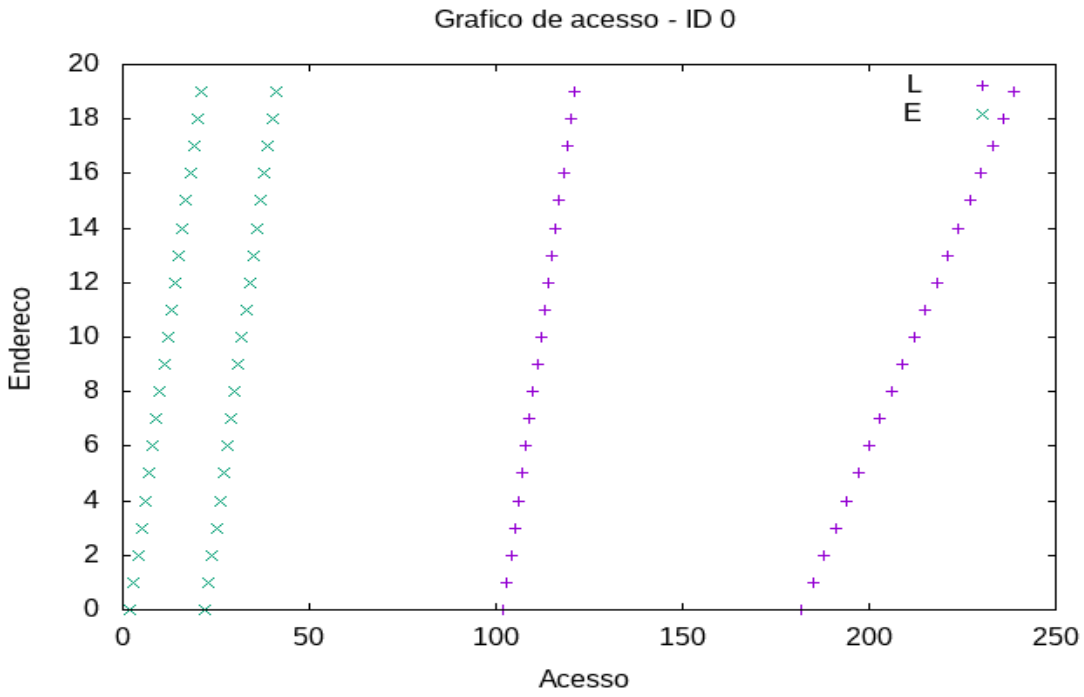
6. Plano de experimento - Localidade de referência

A localidade de referência das três funções foi analisada através de um programa que gera um mapa dos acessos em memória; uma evolução temporal das distâncias de pilha da função; e um histograma contendo as distâncias de pilha. A analise foi feita em uma entrada fixa de tamanho 20 em cada função. É necessário que a entrada seja pequena para visualizar melhor os resultados.

7. Análise da localidade de referência

7.1. Soma

7.1.1 Mapas de acesso das estruturas de dados



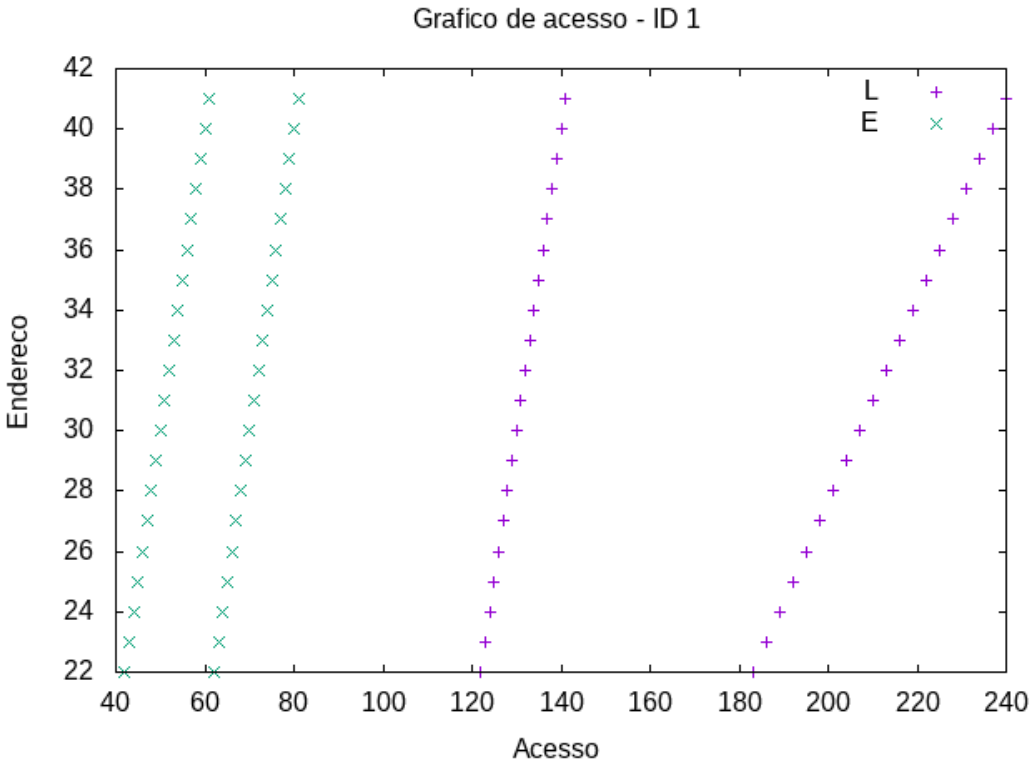


Grafico de acesso - ID 1
Endereco
Acesso
L
E
42
40
38
36
34
32
30
28
26
24
22
40
60
80
100
120
140
160
180
200
220
240

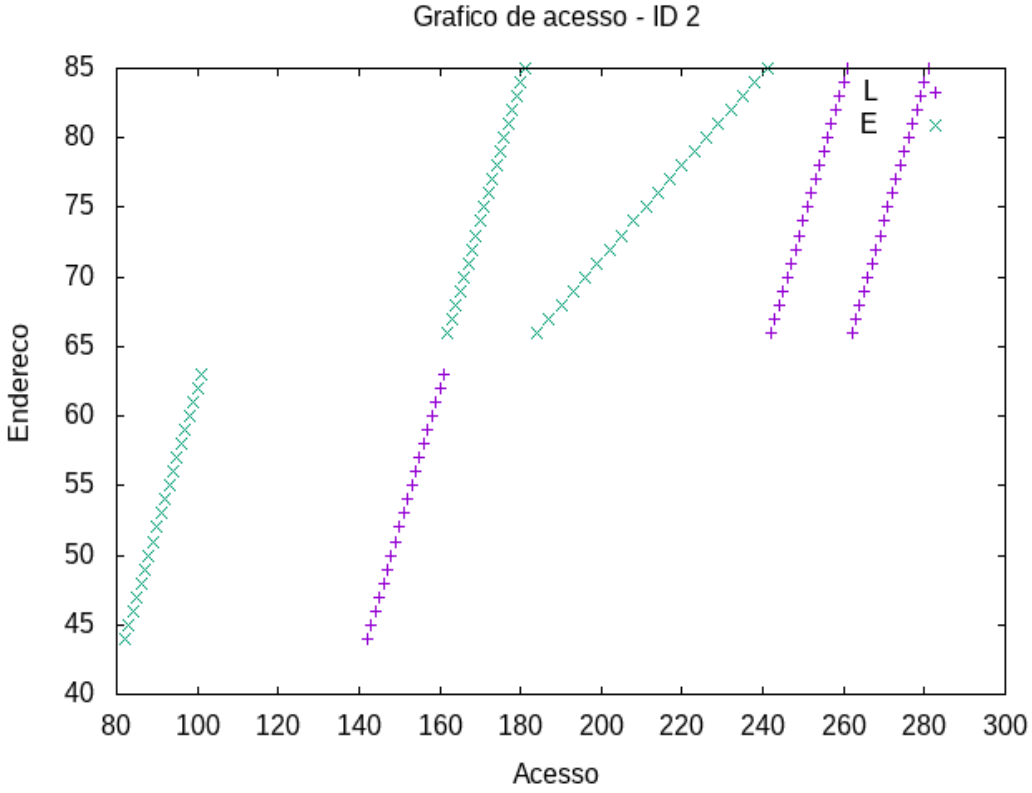
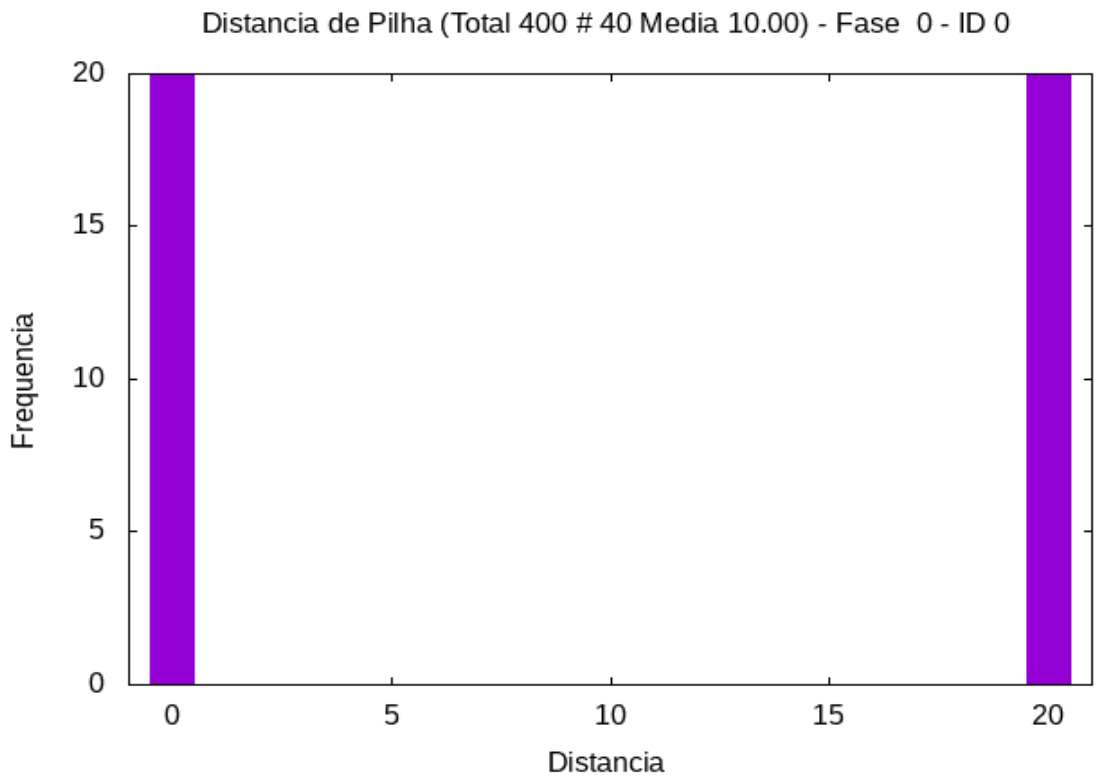
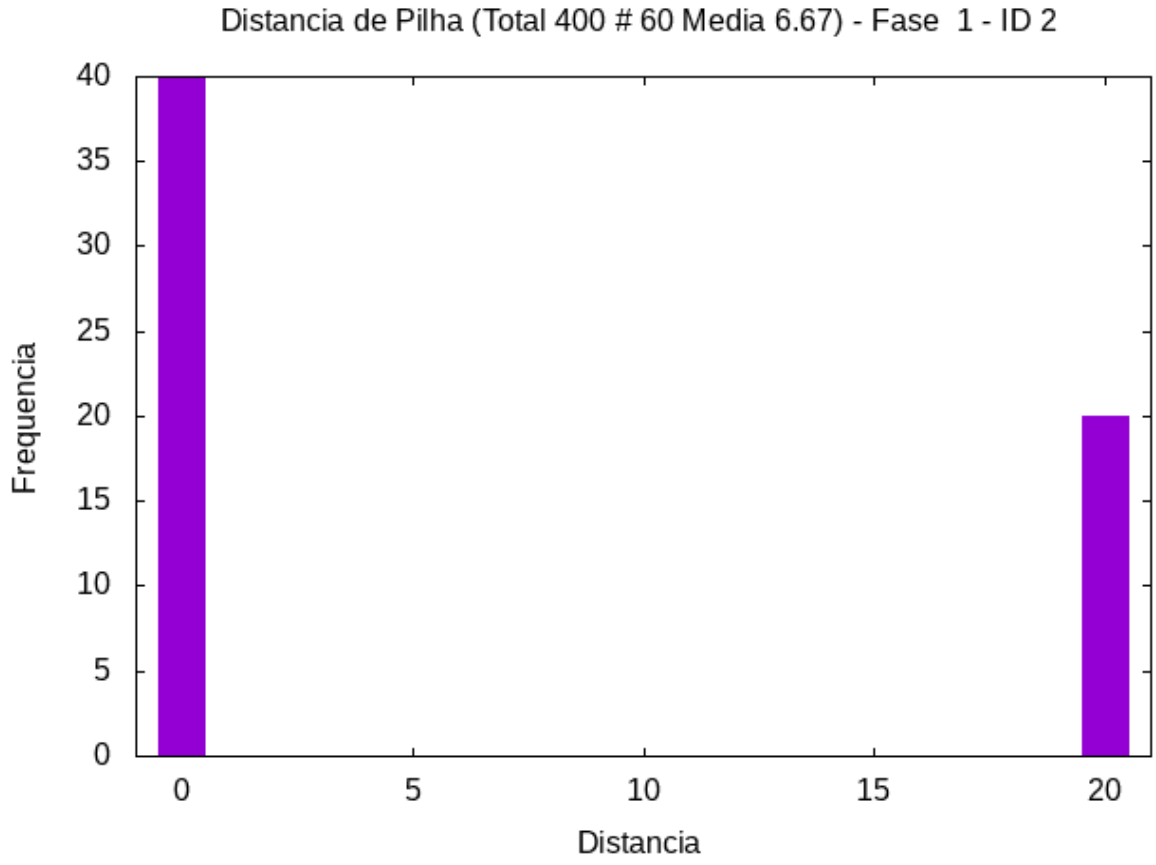


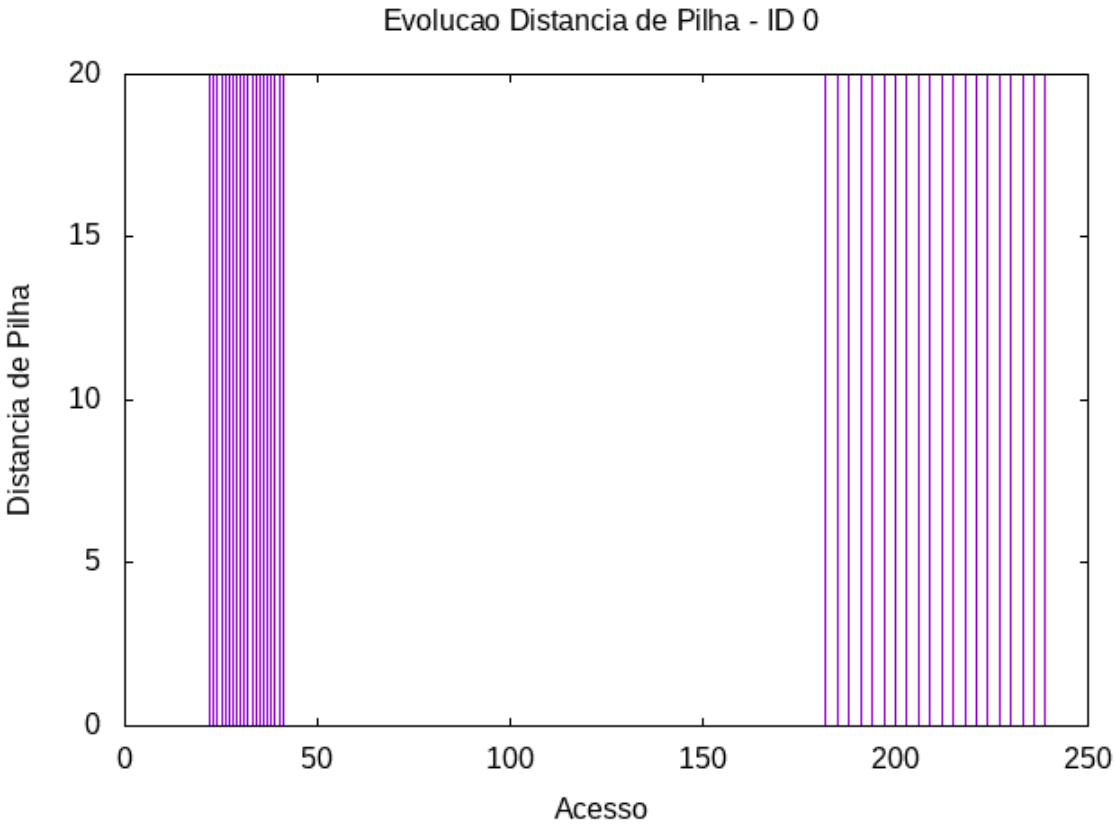
Grafico de acesso - ID 2
Endereco
Acesso
L
E
85
80
75
70
65
60
55
50
45
40
80
100
120
140
160
180
200
220
240
260
280
300

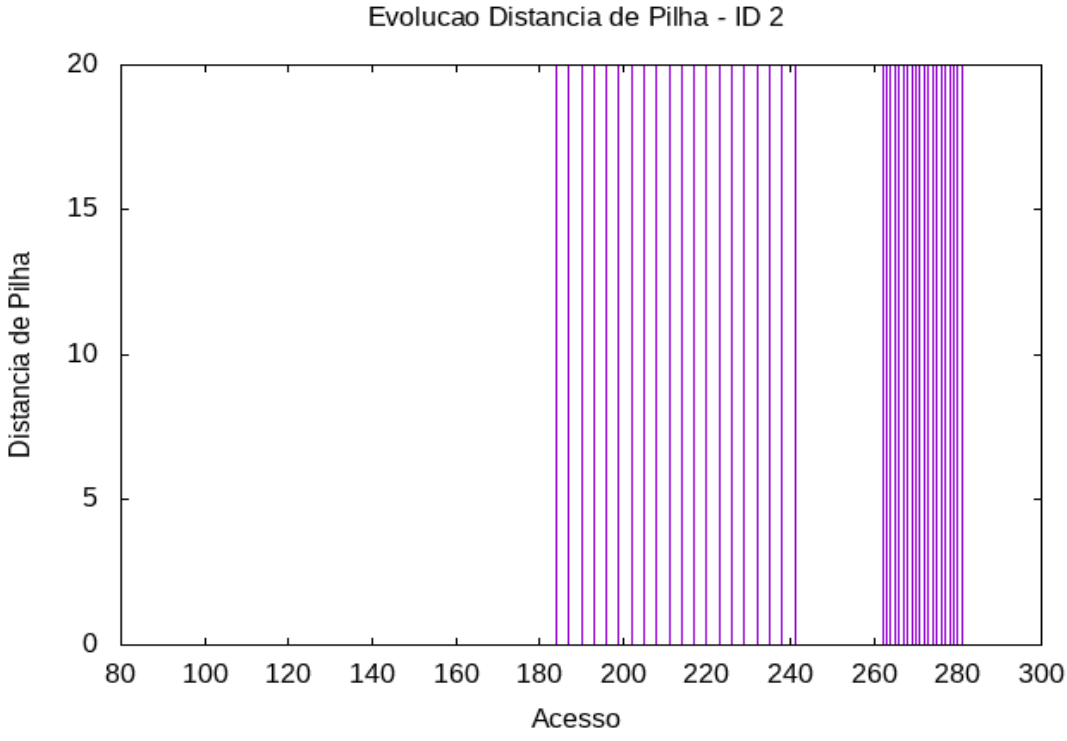
7.1.2 Histogramas de distância de pilha





7.1.3. Evolução temporal de distância de pilha





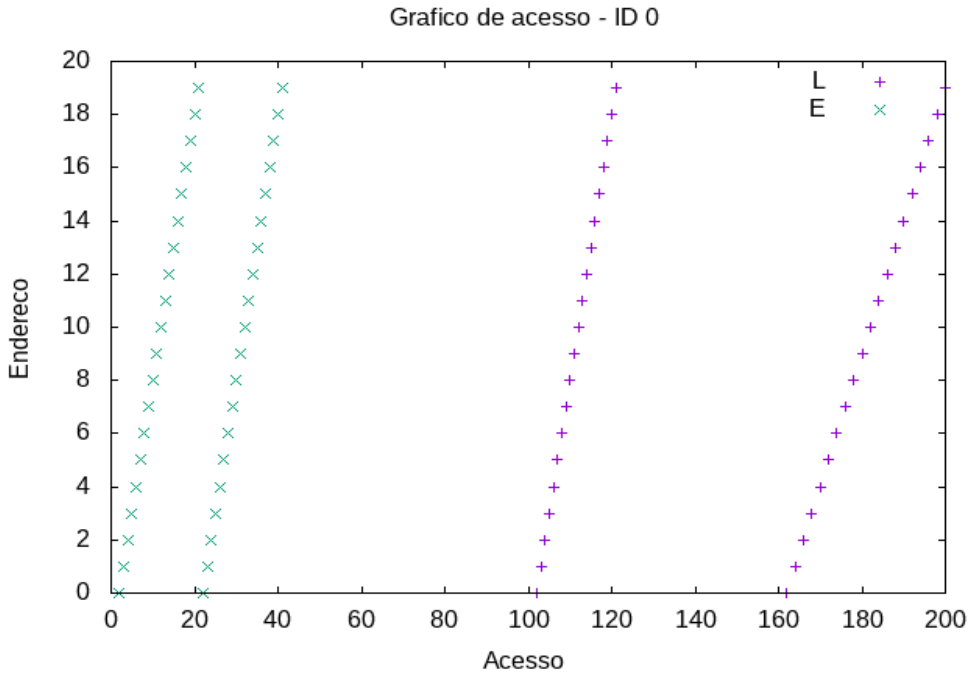
7.1.4. Análise

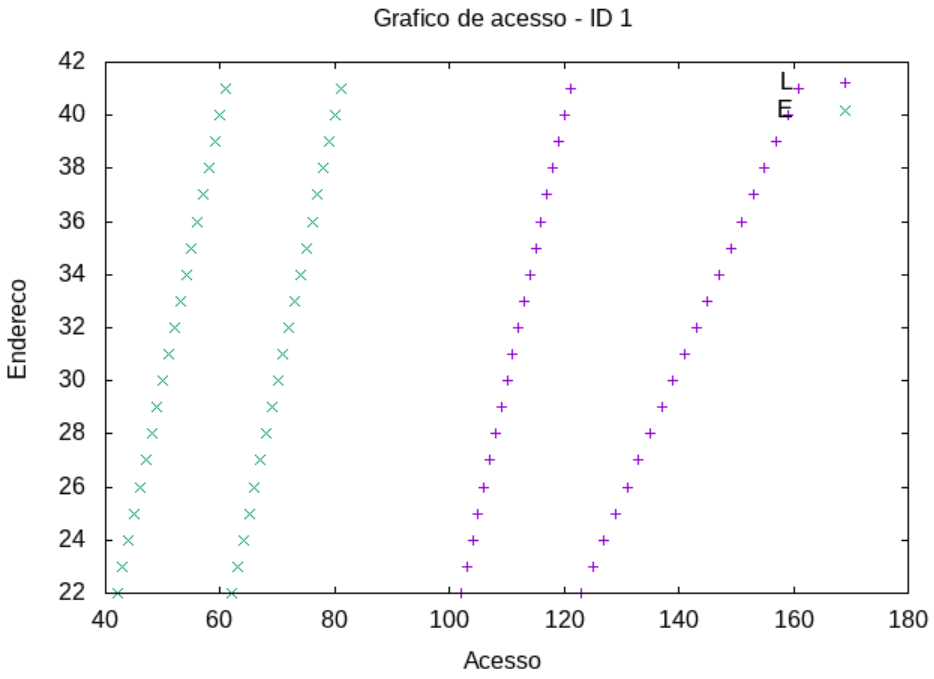
Na operação de soma, é possível perceber, no mapa do ID 1 e 2, o acesso das funções de inicializar vetor no início, o acesso da função de *log* no meio, e da função de soma no final. A função de soma apresenta os acessos mais espaçados, pois ela acessa a posição X de ambos os vetores (1 e 2) na mesma iteração do *loop*.

O mapa de ID 3 representa o vetor na qual foi escrito o resultado da soma de ambos os vetores. Nele, é feito a inicialização do vetor (inicio), a leitura e uma nova inicialização do vetor na função de soma. Isso faz o próximo acesso ser em novos endereços de memória, o que afeta a localidade de referencia e a distancia de pilha. Após isso, essas novas posições são acessadas para escrita dos valores das somas dos outros dois vetores.

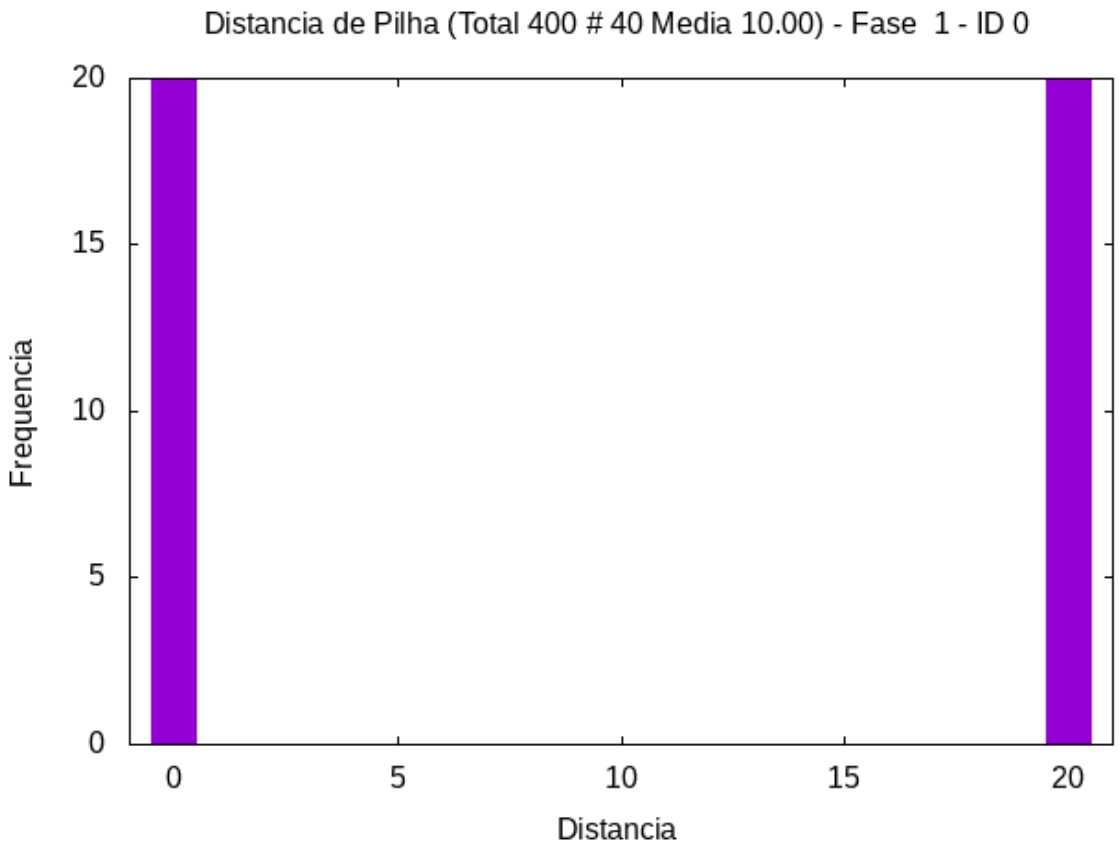
7.2. Multiplicação

7.2.1. Mapas de acesso das estruturas de dados

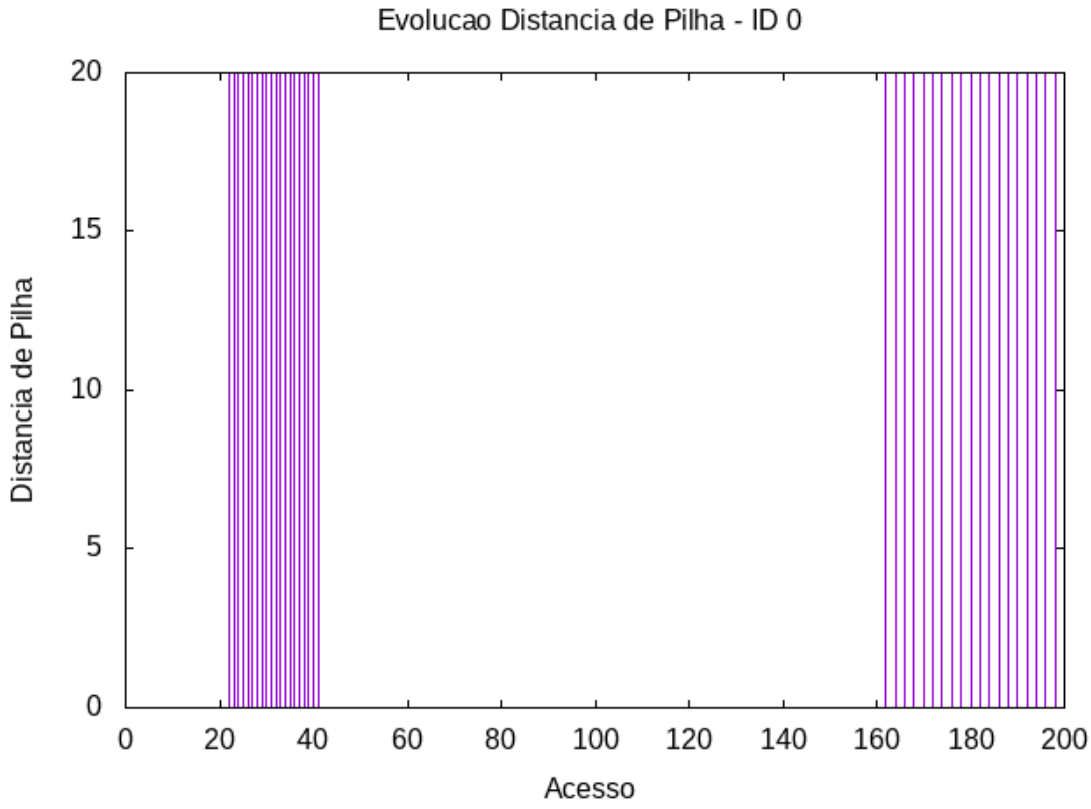




7.2.2. Histogramas de distância de pilha



7.2.3. Evolução temporal de distância de pilha

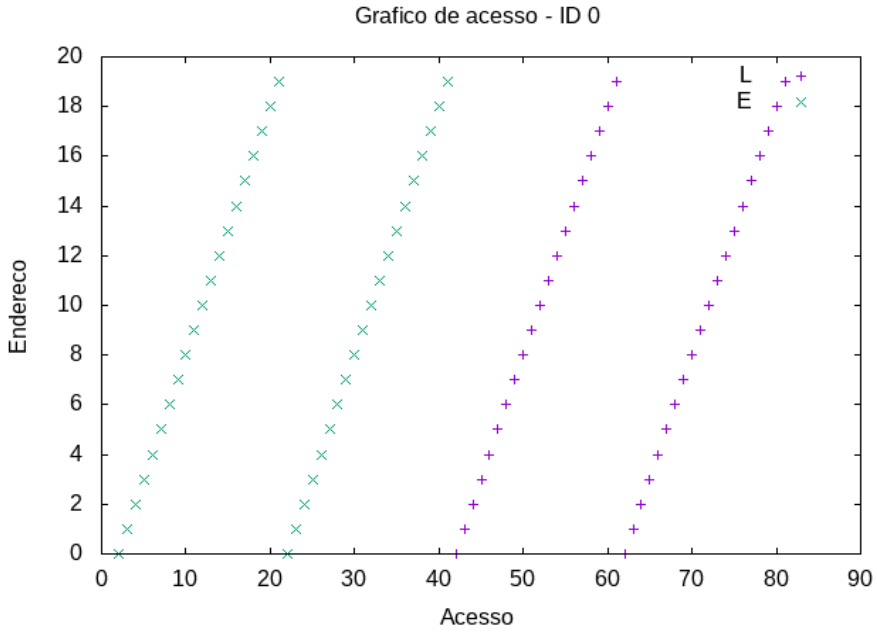


7.2.4. Análise

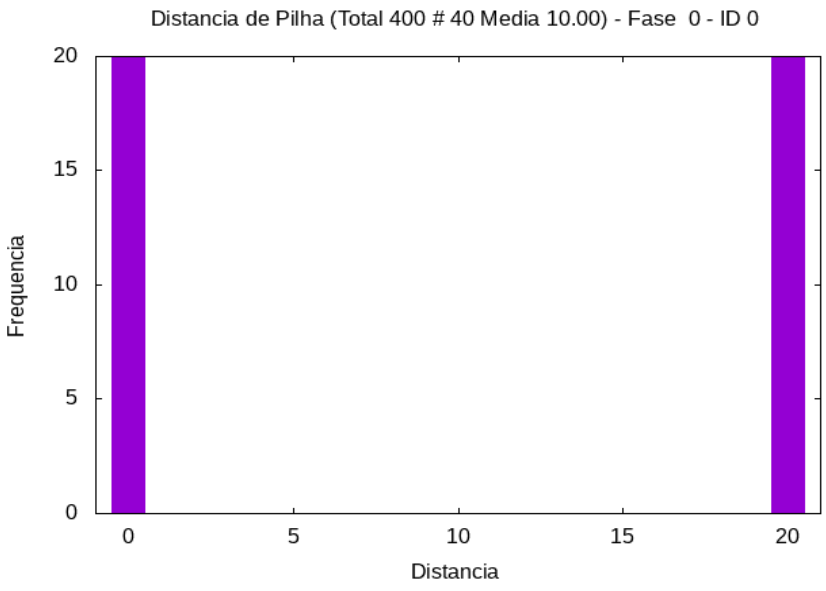
Para o algoritmo do cálculo de multiplicação, é possível perceber no mapa de acessos que o programa apenas faz a inicialização (nula e aleatória) do vetor, depois a leitura de todas as posições na função de *log*, e depois acessa cada posição do vetor para pegar seu valor e multiplicar pelo segundo vetor. Esse último processo é visível no mapa pela linha menos inclinada. Como ele acessa a posição X de ambos os vetores, os acessos acabam ficando mais espaçados. A distância de pilha em 0 representa o primeiro acesso ao vetor, e em 20 quando o vetor é acessado para olhar os valores aleatórios que foram inseridos nele.

7.3. Norma

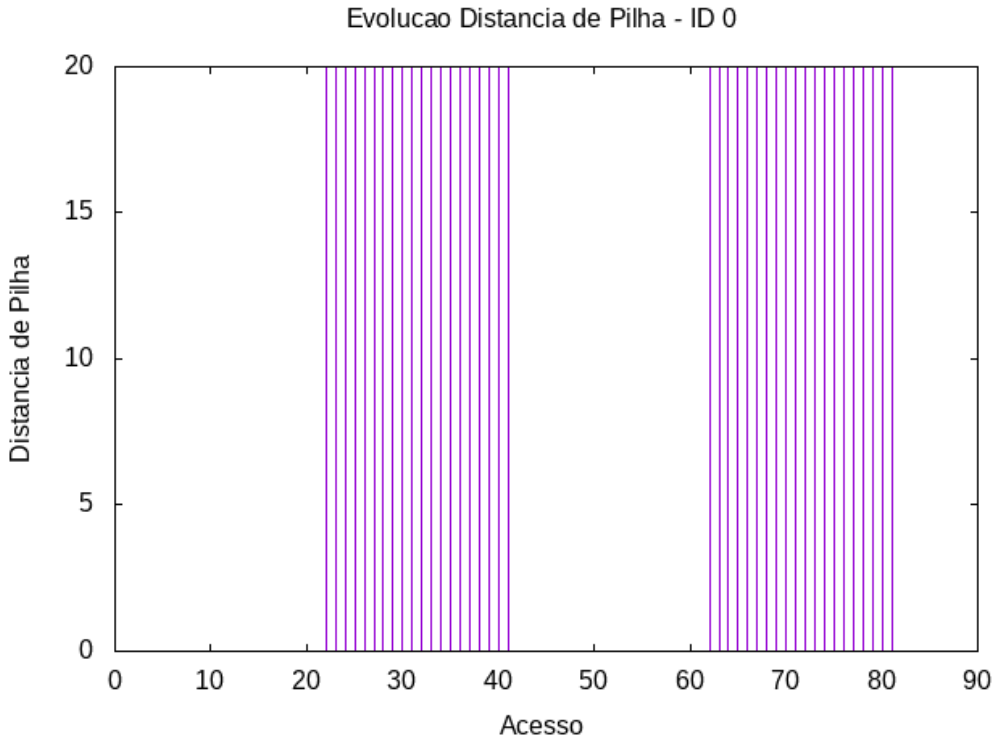
7.3.1. Mapas de acesso das estruturas de dados



7.3.2 Histogramas de distância de pilha



7.3.3. Evolução temporal de distância de pilha



7.3.4. Análise

Para o algoritmo do cálculo de norma, é possível perceber no mapa de acessos que o programa apenas faz a inicialização (nula e aleatória) do vetor e depois a leitura de todas as posições, para fazer a operação. O histograma e a evolução temporal, tanto para a fase de leitura quanto de escrita, são os mesmos. Isso se deve ao fato do vetor ser acessado, por inteiro, duas vezes em cada fase. Refatorando o processo de inicialização e removendo o *log* em memória, a distância de pilha seria menor.