

CEFET-MG - Campus II

Departamento de Engenharia Elétrica
Computação de Alto Desempenho

Lista I

Alunos: Antônio Augusto Diniz Sousa
Professor: Eduardo Henrique da Rocha Coppoli

13 de Maio
2019

CEFET-MG - Campus II

Departamento de Engenharia Elétrica
Computação de Alto Desempenho

Lista I

Lista I apresentado à Disciplina de Computação de Alto Desempenho do Curso de Engenharia Elétrica do Centro Federal de Educação Tecnológica de Minas Gerais, como requisito parcial para conclusão das matérias eletivas do curso de Engenharia de Computação da mesma instituição.

Alunos: Antônio Augusto Diniz Sousa

Professor: Eduardo Henrique da Rocha Coppoli

13 de Maio
2019

Sumário

1	Questão 1	1
2	Questão 2	5
2.1	a	5
2.2	b	6
2.3	Observações	6
3	Questão 3	7
4	Questão 4	8

1 Questão 1

Nessa questão foi solicitada a confecção de um código que gerasse diversos números aleatórios, ordenasse a lista gerada, e separasse em faixas uniformes.

Como tratou-se de um código muito extenso, ele se encontra no github do trabalho. Segue abaixo as imagens referentes a execução do código para uma lista de 256 números.

Figura 1: Lista de Números Aleatórios gerada

Números aleatórios gerados:									
41	123	16	242	48	135	221	117	27	232
49	69	227	11	71	41	103	242	68	70
199	218	156	161	196	107	157	154	169	87
253	201	201	6	180	249	140	146	103	166
123	151	234	87	154	42	119	2	29	179
71	220	142	219	118	76	71	12	221	231
99	219	169	45	216	94	31	102	239	133
5	99	22	239	178	175	18	34	176	47
212	238	4	92	202	121	167	10	132	133
232	222	89	138	5	50	231	35	151	207
168	148	44	189	124	213	101	141	246	22
179	204	251	182	33	199	40	199	200	164
69	178	131	157	61	127	206	30	162	95
228	67	242	17	247	111	221	93	243	205
106	168	146	102	95	178	38	127	122	238
36	190	153	166	92	205	31	35	234	192
121	200	250	108	208	242	210	175	72	191
125	177	104	16	16	190	193	54	62	52
29	97	233	181	1	62	123	31	96	95
214	217	40	209	62	247	196	10	167	5
192	29	181	41	44	196	230	228	241	38
17	15	134	249	187	126	56	56	156	144
150	115	98	189	69	151	173	2	152	78
6	89	106	178	129	142	112	97	107	98
126	124	113	251	118	45	123	166	92	24
55	241	131	144	167	191				

Com a lista gerada, demonstrada pela Figura 1, ordenou-se através de um algoritmo *Merge Sort*, obtendo o resultado da Figura 2

Figura 2: Lista Ordenada

Números aleatórios ordenandos									
1	2	2	4	5	5	5	6	6	10
10	11	12	15	16	16	16	17	17	18
22	22	24	27	29	29	29	30	31	31
31	33	34	35	35	36	38	38	40	40
41	41	41	42	44	44	45	45	47	48
49	50	52	54	55	56	56	61	62	62
62	67	68	69	69	69	70	71	71	71
72	76	78	87	87	89	89	92	92	92
93	94	95	95	95	96	97	97	98	98
99	99	101	102	102	103	103	104	106	106
107	107	108	111	112	113	115	117	118	118
119	121	121	122	123	123	123	123	124	124
125	126	126	127	127	129	131	131	132	133
133	134	135	138	140	141	142	142	144	144
146	146	148	150	151	151	151	152	153	154
154	156	156	157	157	161	162	164	166	166
166	167	167	167	168	168	169	169	173	175
175	176	177	178	178	178	178	179	179	180
181	181	182	187	189	189	190	190	191	191
192	192	193	196	196	196	199	199	199	200
200	201	201	202	204	205	205	206	207	208
209	210	212	213	214	216	217	218	219	219
220	221	221	221	222	227	228	228	230	231
231	232	232	233	234	234	238	238	239	239
241	241	242	242	242	242	243	246	247	247
249	249	250	251	251	253				

Após a ordenação da lista, criou-se também um método que separa esses números em faixas. Para visualizar essas faixas, desenvolveu-se outro método para imprimir as faixas, para facilitar na visualização dos números pertencentes a cada faixa, conforme Figura 3.

Figura 3: Faixas Geradas

Imprimindo faixa 0									
1	2	2	4	5	5	5	6	6	10
10	11	12	15	16	16	16	17	17	18
22	22	24	27	29	29	29	30	31	31
31	251								
Imprimindo faixa 1									
33	34	35	35	36	38	38	40	40	41
41	41	42	44	44	45	45	47	48	49
50	52	54	55	56	56	61	62	62	62
113									
Imprimindo faixa 2									
67	68	69	69	69	70	71	71	71	72
76	78	87	87	89	89	92	92	92	93
94	95	95	95						
Imprimindo faixa 3									
96	97	97	98	98	99	99	101	102	102
103	103	104	106	106	107	107	108	111	112
113	115	117	118	118	119	121	121	122	123
123	123	123	124	124	125	126	126	127	127
Imprimindo faixa 4									
129	131	131	132	133	133	134	135	138	140
141	142	142	144	144	146	146	148	150	151
151	151	152	153	154	154	156	156	157	157
161									
Imprimindo faixa 5									
161	162	164	166	166	166	167	167	167	168
168	169	169	173	175	175	176	177	178	178
178	178	179	179	180	181	181	182	187	189
189	190	190	191	191					
Imprimindo faixa 6									
192	192	193	196	196	196	199	199	199	200
200	201	201	202	204	205	205	206	207	208
209	210	212	213	214	216	217	218	219	219
220	221	221	221	222					
Imprimindo faixa 7									
227	228	228	230	231	231	232	232	233	234
234	238	238	239	239	241	241	242	242	242
242	243	246	247	247	249	249	250	251	251
253									

Para finalizar a visualização do algoritmo, criou-se uma função para contabilizar a quantidade de elementos pertencentes a cada faixa, conforme de-

monstrado pela Figura 4.

Figura 4: Frequência de cada faixa

```
Faixa 0 a 32 - Frequencia = 32
Faixa 32 a 64 - Frequencia = 31
Faixa 64 a 96 - Frequencia = 24
Faixa 96 a 128 - Frequencia = 40
Faixa 128 a 160 - Frequencia = 31
Faixa 160 a 192 - Frequencia = 35
Faixa 192 a 224 - Frequencia = 35
Faixa 224 a 256 - Frequencia = 31
```

Após a criação do algoritmo, testou-se diversos níveis de otimização, para diversos tamanhos de listas. O resultado pode ser visto através da Tabela 1.

Tabela 1: Resultado para diversos níveis de otimização

Número item	256	2560	25600	256000	2560000	25600000
Sem Otimização	0,002s	0,002s	0,013s	0,076s	0,652s	7,092s
Level 1	0,002s	0,002s	0,008s	0,053s	0,367s	3,865s
Level 2	0,002s	0,002s	0,007s	0,053s	0,366s	3,883s
Level 3	0,002s	0,002s	0,008s	0,053s	0,352s	3,772s
Level 4	0,002s	0,002s	0,007s	0,051s	0,352s	3,657s
Level 5	0,002s	0,002s	0,007s	0,051s	0,352s	3,542s

Com esses testes, pode-se concluir que os níveis de otimização reduzem consideravelmente o tempo de execução de um algoritmo, principalmente para entradas grandes. No exemplo utilizado, a maior diferença deu-se passando do Level 0 para o Level 1, reduzindo cerca de 50% do tempo de execução.

2 Questão 2

Nessa questão foi solicitada a conversão de dois números para o padrão de ponto flutuante estabelecido pela norma 754 da IEEE. Tal norma separa os tipos de dados em três, conforme sua precisão, sendo:

- Single precision (*float*) 32 bits (precisão 24 bits)
- Double precision (*double*) 64 bits (precisão 53 bits)
- Double extended precision 80 bits (precisão 63 bits)

O padrão também define como deve ser escrita o número, tendo sua representação numérica dada pela

$$(-1)^S M \times 2^E \quad (1)$$

Onde:

- Bit de sinal s determina se número é negativo ou positivo;
- Mantissa M é um valor fracionário no intervalo $[1.0, 2.0)$, na representação normalizada;
- E representa o expoente

Agora, sabendo um pouco sobre a norma utilizada, podemos fazer as conversões.

2.1 a

Numero a ser convertido é: 0,00752.

Por questões de facilidade, irei utilizar a precisão simples, ou single precision, por precisar de um número menor de bits.

De acordo com a norma, o numero pertencente a essa precisão possui a seguinte forma:

- 1 bit para o sinal.
- 8 bits para o expoente.
- 23 bits para a representação da mantissa.

Seguiremos agora um passo a passo para facilitar o entendimento dos cálculos executados:

1. Converter o número de decimal para binário:

$$0,00752 = 0,000000011110110011010100101010100001000..._2$$

2. Deslocar a vírgula:

$$0,00752 = 1,1110110011010100101010100001000..._2 \times 2^{-8}$$

3. Normalizar o expoente:

$$-8+127 = 119 = 1110111_2$$

4. Juntar tudo

$$0,00752_{10} = 0\ 01110111\ 11101100110101001010101_{IEEE754}$$

2.2 b

Numero a ser convertido é: -688,25.

Por questões de facilidade, também irei utilizar a precisão simples nesse exercício.

Seguiremos o mesmo passo a passo utilizado na questão anterior:

1. Converter o número de decimal para binário:

$$-688,25 = 1010110000,01_2$$

2. Deslocar a vírgula:

$$-688,25 = 1,01011000001_2 \times 2^9$$

3. Normalizar o expoente:

$$9+127 = 136 = 10001000_2$$

4. Juntar tudo e completar o número de bits com 0 a esquerda.

$$-688,25_{10} = 1\ 10001000\ 00000000000001011000001_{IEEE754}$$

2.3 Observações

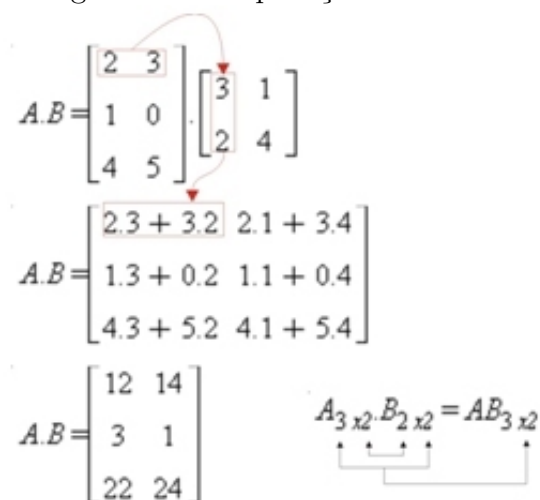
Para calcular a conversão de decimal para binário da parte inteira, utilizou-se a própria calculadora do sistema operacional. Já para calcular a da parte fracionada, utilizou-se uma planilha no *LibreOffice Calc*, na qual está disponibilizada no github da disciplina.

3 Questão 3

Primeiramente, para solucionar esse problema, é necessário saber o funcionamento de uma operação de multiplicação de matrizes.

O produto de matrizes é resultado da multiplicação da coluna da primeira matriz, pela linha da segunda, conforme Figura 5

Figura 5: Multiplicação de matrizes



$$A.B = \begin{bmatrix} 2 & 3 \\ 1 & 0 \\ 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}$$

$$A.B = \begin{bmatrix} 2.3 + 3.2 & 2.1 + 3.4 \\ 1.3 + 0.2 & 1.1 + 0.4 \\ 4.3 + 5.2 & 4.1 + 5.4 \end{bmatrix}$$

$$A.B = \begin{bmatrix} 12 & 14 \\ 3 & 1 \\ 22 & 24 \end{bmatrix}$$

$$A_{3 \times 2} \cdot B_{2 \times 2} = AB_{3 \times 2}$$

Considerando matrizes quadradas de ordem n , para calcular um termo é necessário n multiplicações e $n - 1$ adições. Como a matriz resultante possui n^2 , o total de operações necessárias para fazer um produto de 2 matrizes é $n^2 \times (2n - 1)$, ou seja, $2n^3 - n^2$.

Para ter 1Mflop é necessário que a máquina consiga calcular o produto de duas matrizes de ordem superior a 100 em 1 segundo.

Já para ter 1Gflop, é necessário o produto de matrizes de ordem superior a 1000.

4 Questão 4