Relátorio de Estrutura de Dados

Yan Rodrigues da Silva
Ciência da computação
495532

December 7, 2021

1 Especificações da máquina e organização do trabalho

O relátorio do trabalho de estrutura de dados é construído, de modo a mostrar os datalhes de implementação dos algoritmos, assim como, os resultados obtidos a partir de suas execuções.

1.1 Especificações da máquina

Detalhes técnicos da maquina:

- Processor: Intel® Core i3-4010U CPU @ 1.70GHZ x 4.
- Memory: 7.5 GB.
- Disk Capacity: 256.1 GB
- SO: Ubuntu 20.04
- OS Type: 64 bit

Para o problema de ordenação apresentado na seção 3.2 do projeto, o valor máximo de entrada foi 600.000.000 e não 1.000.000.000 como o documento pedia originalmente. Minha máquina não conseguiu processar o valor proposto inicialmente.

1.2 Organização do trabalho

Detalhes de organização:

- Todos os algoritmos estão contidos na pasta *src/lib*.
- Os arquivos de testes estão contidos na pasta *src/tests*.
- Veja o arquivo Makefile para poder executar algum dos testes.

2 Heap

Heap é uma estrutura de dados que é geralmente representada como uma árvore binária, mas construída em cima de uma lista. Para a implementação do heap máximo, foi definido uma struct Heap contendo os campos: capacity, length e um ponteiro para uma lista (*list).

O campo capacity define um limite para a quantidade de elementos que podem ser inseridos no vetor. Caso um elemento tente ser inserido e a capcidade da Heap esteja no máximo não será possivel inserir esse elemento. O campo length, no que lhe concerne, auxilia na inserção de elementos e informa também quantos elementos a estrutura possui.

Operações implementadas:

1. CreateHeap → Criar
2. maxHeapifyUp → Subir
3. maxHeapifyDown → Descer
4. Insert → Inserir
5. Remove → Remover
6. Heapify → Construir

3 Ordenação

3.1 Algoritmos de ordenação

Algoritmos de ordenação são desenhados e implementados para, dado um vetor qualquer A $= [a_1, a_2, \ldots, a_n]$, determinarmos um arranjo A' de A tal que $a'_i \leq a'_{i+1}$ para todo $i \in [1, n-1]$. Trabalharemos com os algoritmos *InsertionSort* e *HeapSort* para os problemas de ordenação.

Sabemos que os algoritmos *InsertionSort* e *HeapSort* possuem complexidade: $\mathcal{O}(n^2)$ e $\Theta(n \cdot \log n)$ respectivamente, portanto é de se esperar que *HeapSort* possua melhores resultados em relação ao *InsertionSort*.

3.2 Resultados obtidos

Detalhes importantes:

1. $\Lambda = [0, 600.000.000]$ é intervalo de entradas dos algoritmos.
2. Para o InsertionSort os valores de 10.000.000 e 1.000.000.000 não foram computados, pois passaram de 24h e ainda estavam executando.

InsertionSort:

Figure 1: Gráfico do InsertionSort

HeapSort:

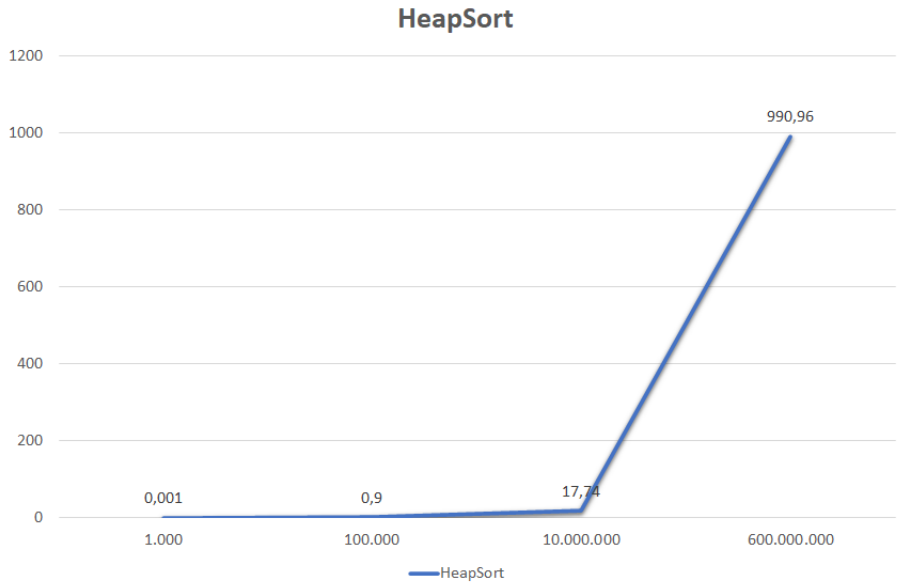

Figure 2: Gráfico do HeapSort

4 Tabela de Dispersão

Para a construção da tabela de dispersão foram criadas duas estruturas: *Node* e *HashTable*. A estrutura *Node* vai armazenar uma chave inteira e um valor do tipo *string* e um ponteiro para a próxima estrutura *Node*.

A estrutura *HashTable*, por sua vez, possui 3 campos: capacity, collision e um vetor de ponteiros list, para a estrutura *Node*. O campo capacity define a quantidade de elementos que list vai possuir, o campo collision é utilizado para calcular a quantidade de colisões.

4.1 Resultados obtidos

Método da divisão:

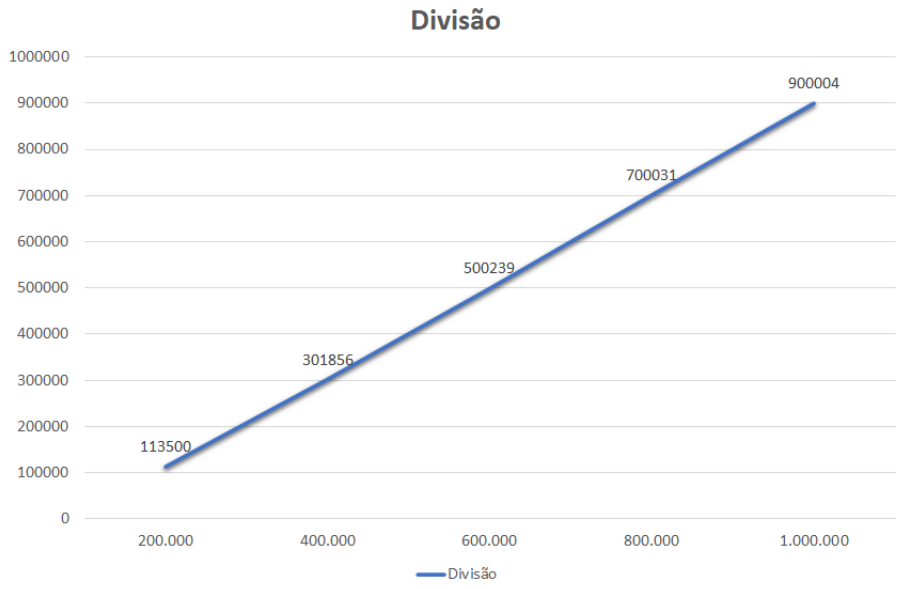

Figure 3: Método da Divisão

Método da multiplicação:

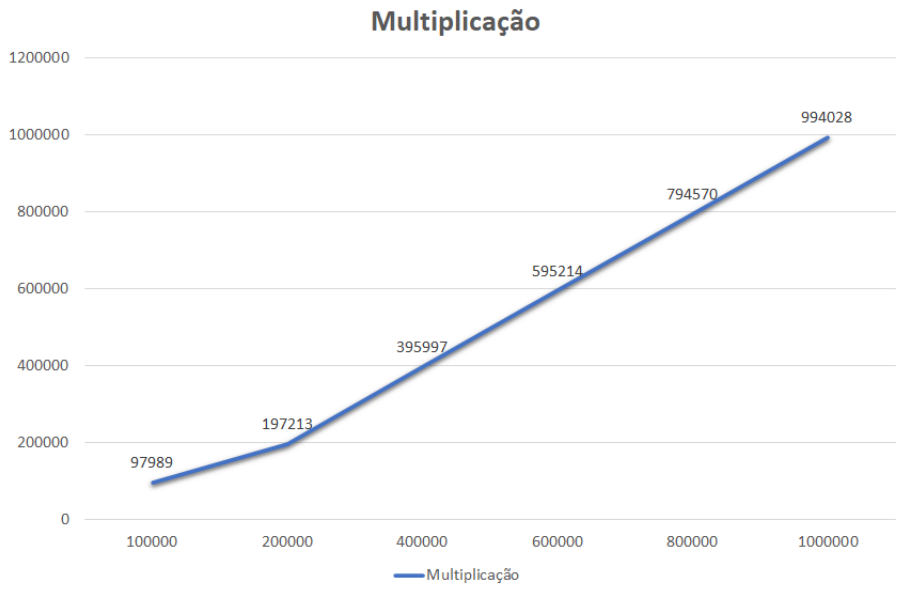

Figure 4: Método da Multiplicação

Método da Dobra:

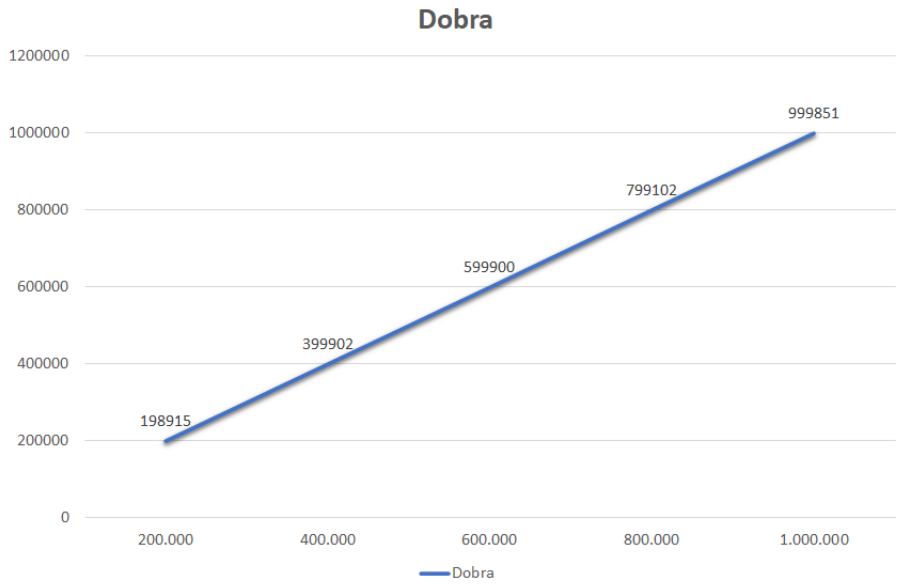

Figure 5: Método da Dobra

Comparando todos os métodos:

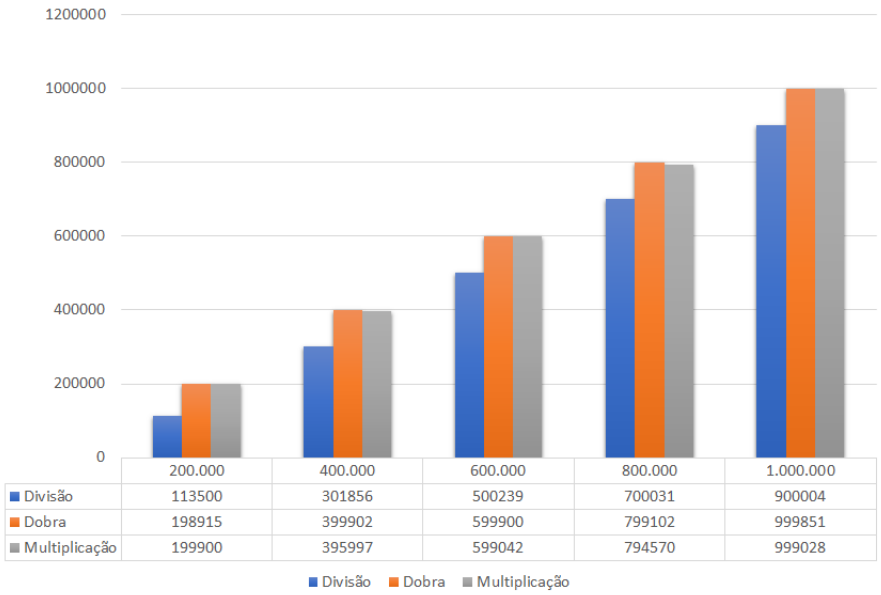

Figure 6: Visualizando todos métodos