Universidade Estadual de Maringá
Centro de Tecnologia
Departamento de Informática
Bacharelado em Ciência da Computação
5184-32 - Projeto e Análise de Algoritmos

Trabalho de Projeto e Análise de Algoritmos
Projeto 1

Alunos: Diogo Alves de Almeida RA: 95108
Luis Felipe Costa Guimarães RA: 95372

Professor: Rodrigo Calvo

Maringá
2017

1 Introdução

Esse trabalho tem como propósito implementar uma solução alternativa, de melhor eficiência, para verificação de existência e contagem de elementos iguais em vetores ordenados. Primeiramente se é implementada um método fácil, contudo muito custoso para uma base de dados minimamente grande. Já a implementação elaborada diminui esse custo significantemente, fazendo-o muito melhor para bases de dados pequenas até enormes.

2 Algoritmo

```
1  int busca_elaborada(int *vetor, int inicio, int fim, int valor){ //busca binaria
       recursiva
2      int quantia = 0;
3      chamadas++;
4      if (inicio > fim) return 0;
5      else {
6          int meio = (inicio + fim)/2;
7          if ((vetor[inicio] == vetor[fim]) && (vetor[inicio] == valor) && (vetor[
               fim] == valor)){ //se todos os elementos do vetor for o elemento
               procurado
8              quantia += fim - inicio + 1;
9              return quantia;
10         } else if ((vetor[inicio] == vetor[meio]) && (vetor[inicio] == valor) &&
               (vetor[meio] == valor)){ // se o inicio ateh o meio do vetor for o
               elemento procurado
11             quantia +=meio - inicio + 1;
12             if (vetor[meio+1] == valor){ // se ainda tem o elemento procurado
                   na direita do meio
13                 quantia += busca_elaborada(vetor, meio+1, fim, valor);
14             }
15             return quantia;
16         } else if ((vetor[meio] == vetor[fim]) && (vetor[meio] == valor) && (
               vetor[fim] == valor)){ // se o meio ate o fim do vetor tem o
               elemento procurado
17             quantia += fim - meio + 1;
18             if (vetor[meio-1] == valor){ //se ainda tem o elemento procurado
                   na esquerda do meio
19                 quantia += busca_elaborada(vetor, inicio, meio-1, valor);
20             }
21             return quantia;
22         } else if (vetor[meio] == valor){ // se o meio do vetor for o elemento
               procurado
23             quantia++;
24             if((vetor[meio+1] == valor) && (vetor[meio-1] == valor)){ //se o
                   da esquerda e o da direita for o elemento procurado
25                 quantia += busca_elaborada(vetor, meio+1, fim, valor);
26                 quantia += busca_elaborada(vetor, inicio, meio-1, valor);
27             } else if (vetor[meio+1] == valor){ //so o da direita eh o
                   elemento procurado
28                 quantia += busca_elaborada(vetor, meio+1, fim, valor);
29             } else if (vetor[meio-1] == valor){ // so o da esquerda eh o
                   elemento procurado
30                 quantia += busca_elaborada(vetor, inicio, meio-1, valor);
31             }
32         } else if(vetor[meio+1] <= valor){ // se o elemento da direita do
               meio eh menor ou igual o elemento procurado
33             quantia += busca_elaborada(vetor, meio+1, fim, valor);
34         } else if (vetor[meio-1] >= valor){ // se o elemento da esquerda do
               meio eh menor ou igual o elemento procurado
35             quantia += busca_elaborada(vetor, inicio, meio-1, valor);
36         }
```

```
37              }
38      return quantia;
39  }
```

3 Descrição do Algoritmo

A solução apresentada para o problema que o trabalho propõe foi dada através do uso de diversas recursões de uma busca binária com parâmetros o ponteiro do vetor em que estão todos os elementos, ordenados, além disso também temos como parâmetro o início, fim, e o valor que é o elemento a ser procurado no vetor. A cada recursão é feito o cálculo do meio através da seguinte matemática:

```
1 int meio = (inicio + fim) / 2;
```

Nessas recursões são realizadas vários tratamentos de caso, através de múltiplos *ifs*. Primeiramente se é verificado se o início não é maior que o fim. Caso seja, a função retorna 0. Caso não seja, a próxima verificação é de se todos os elementos presentes no vetor são o elemento que nós queremos. O que seria o pior caso possível para uma implementação comum, e não elaborada assim como a implementação apresentada. Caso seja, faz o cálculo de quantos elementos estão presentes no vetor e retorna esse número, sem chamar nenhuma outra recursão e o código é encerrado aí.

Caso a verificação do bloco acima seja falsa, é feita a verificação se o início até o meio do vetor for o elemento procurado. Caso seja, além disso é feita a verificação de se ainda restam elementos a direita do meio. Caso seja verdadeira a verificação é chamada uma recursão passando, de modificado do que foi inicialmente invocado, o valor do início, que é passado como:

```
1 meio+1
```

Caso não seja verdadeira a condicional do bloco acima é feito outra verificação de se o meio até o final do vetor tem o elemento procurado. Caso seja, faz a verificação de se ainda restam elementos a esquerda do meio. Caso exista, é chamada uma nova recursão, passando de modificado do que foi inicialmente invocado, o valor do final, que é passado como:

```
1 meio-1
```

Caso falsa a verificação do bloco acima, verifica se o vetor do meio é o elemento procurado. Caso seja, verifica se o da esquerda do meio e o da direita do meio também são os elementos procurados. Caso verdadeiro é chamado duas recursões, a primeira é para uma busca binária nos elementos a esquerda do meio, chamando uma recursão passando, de modificado do que foi inicialmente invocado, primeiramente o valor de inicio, que é passado como:

```
1 meio+1
```

E após isso, de diferente é passado o valor de fim, que é passado como:

```
1 meio-1
```

Caso falso, verifica se somente os elementos da esquerda são os elementos procurados. Caso verdadeiro, chama uma recursão passando, de modificado do que foi inicialmente invocado o valor de início, que é passado como:

```
1 meio+1
```

Caso falso, verifica se os da direita são iguais os elementos procurados. Caso verdadeiro, chama uma recursão passando, de modificado do que foi inicialmente invocado, o valor de fim, que é passado como:

```
1 meio-1
```

Caso a verificação do bloco acima seja falsa, faz a verificação de se o elemento da direita do meio é menor ou igual do que o elemento procurado. Caso verdadeiro, chama uma recursão passando, de modificado do que foi inicialmente invocado, o valor de início, que é passado como:

```
1 meio+1
```

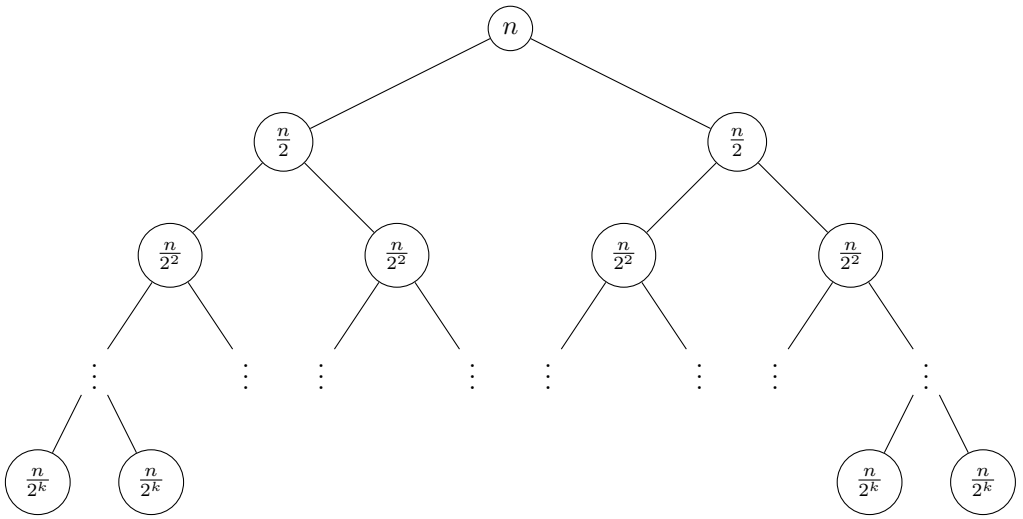
Caso falso, verifica se o elemento a esquerda do meio é maior ou igual ao elemento procurado. Caso verdadeiro, chama uma recursão passando, de modificado do que foi inicialmente invocado, o valor de fim, que é passado como:

```
1 meio-1
```

Após todas essas verificações, o sistema retorna a quantia.

4 Análise do algoritmo

$$T(n) = \begin{cases} constante & , se \quad n = 1 \\ 2T(\frac{n}{2}) + constante & , se \quad n > 1 \end{cases}$$



cn

$2c(\frac{n}{2})$

$4c(\frac{n}{4})$

$2^i c(\frac{n}{2^i})$

$O(\lg n)$

Quantidade de nós por nível: 2^i

Custo de cada nível: cn

Quantidade de níveis: $\lg_2 n$

$\frac{n}{2^i} = 1$

$i = \lg_2 n$

Quantidade de nós no ultimo nível: $2^{\log_2 n} = n \log 2 = n$

Custo do ultimo nível: n

Custo total: $\log_2 n$

5 Gráfico de tempo por elementos

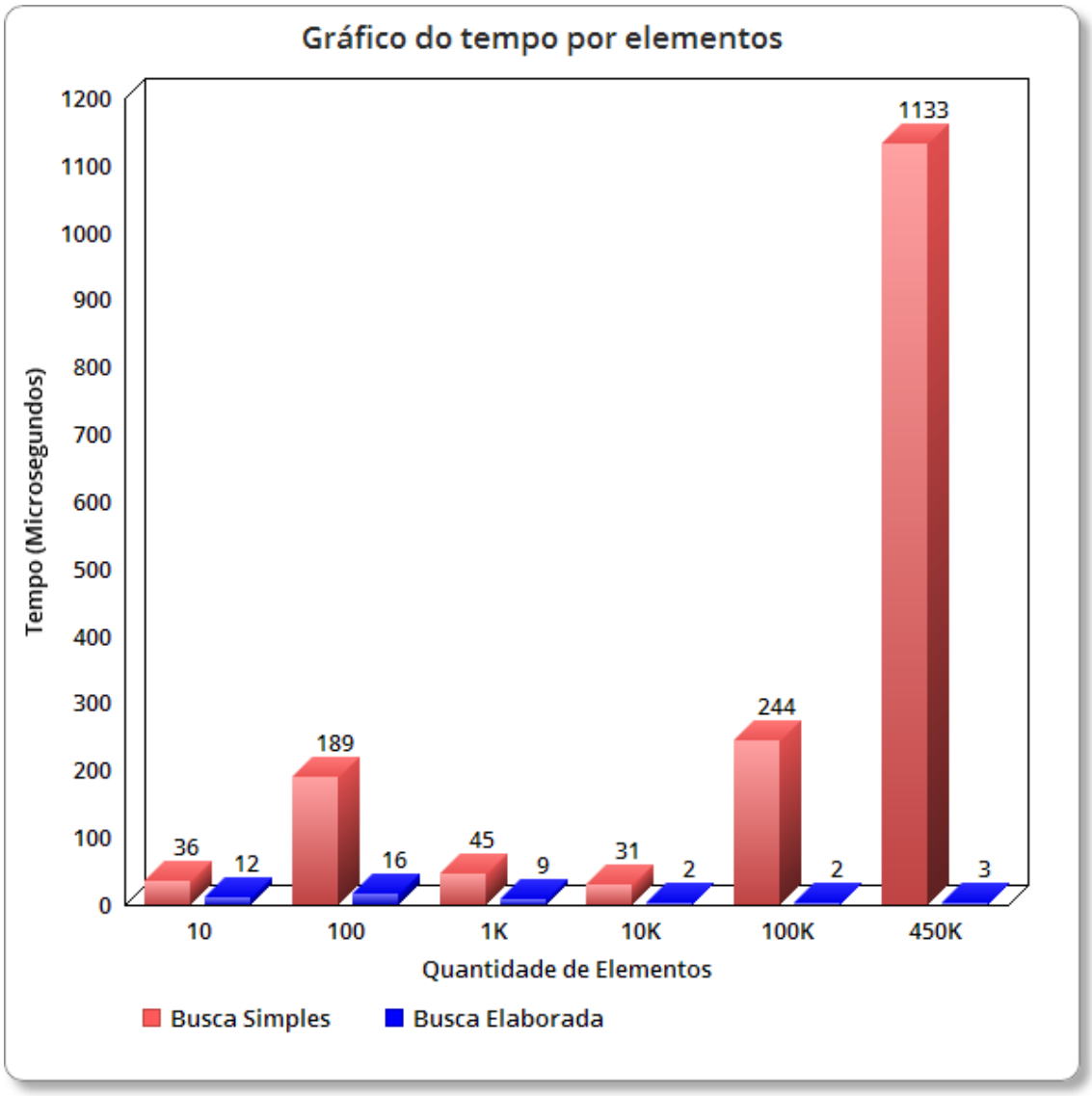


Figura 1: Gráfico de tempo por elementos