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
2018

1 Introdução

Esse trabalho tem como propósito a implementação de uma busca de um elemento em um vetor A com n elementos onde A1 < A2 < ... < Ak > Ak+1 > Ak+2 > ... > An, ou seja, o vetor está ordenado em ordem crescente até o k-ésimo elemento, e a partir desse elemento está ordenado em ordem decrescente. O elemento que deve ser encontrado é o elemento Ak e esse busca deve ser em tempo O(log n), em seguida deve ser implementado um algoritmo para ordenar esse vetor A em tempo O(n).

2 Algoritmo

```c
#include <stdlib.h>
#include <stdio.h>

void printaVetor(int vetor[], int tamanho){
        int i;
        for (i = 0; i < tamanho; i++)
                printf("%d ", vetor[i]);
        printf("\n\n");
}

void radixsort(int vetor[], int tamanho){
        int i;
        int *b;
        int maior = vetor[0];
        int exp = 1;

        b = (int *)calloc(tamanho, sizeof(int));

        for (i = 0; i < tamanho; i++){
                if (vetor[i] > maior)
                        maior = vetor[i];
        }

        while (maior / exp > 0){
                int bucket[10] = {0};
                for (i = 0; i < tamanho; i++)
                        bucket[(vetor[i] / exp) % 10]++;
                for (i = 1; i < 10; i++)
                        bucket[i] += bucket[i - 1];
                for (i = tamanho - 1; i >= 0; i--)
                        b[--bucket[(vetor[i] / exp) % 10]] = vetor[i];
                for (i = 0; i < tamanho; i++)
                        vetor[i] = b[i];
                exp *= 10;
        }
        printf("Vetor ordenado: ");
        printaVetor(vetor, tamanho);

        free(b);
}

int buscaBinaria(int vetor[], int inicio, int fim){
        if(inicio == fim){
         return inicio;
        }
        int meio = inicio + (fim - inicio) / 2;

        if(vetor[meio + 1] < vetor[meio] && vetor[meio - 1] < vetor[meio]){
                return meio;
        } else if (vetor[meio - 1] >= vetor[meio]){
                return buscaBinaria(vetor, inicio, meio);
        } else if (vetor[meio + 1] >= vetor[meio]){
```

```
53                  return buscaBinaria(vetor, meio+1, fim);
54          }
55  }
56
57  int main(){
58          int vetor[] = {1,2,3,7,6,5,4,3,2,1};
59          int tamanho_vetor = sizeof(vetor) / sizeof(vetor[0]);
60          printf("Formato do vetor:  A1< A2< ...< Ak> Ak+1> Ak+2> ... > An \n\n");
61
62          printf("Vetor recebido: ");
63          printaVetor(vetor, tamanho_vetor);
64
65          int k = buscaBinaria(vetor, 0, tamanho_vetor - 1);
66          printf("k encontrado: %d\n", k);
67          printf("A[k] encontrado: %d\n\n", vetor[k]);
68
69          radixsort(vetor, tamanho_vetor);
70          return 0;
71  }
```

3 Descrição do Algoritmo

Neste capítulo será explicado todos as funções do código, as decisões de implementação e as dificuldades encontradas. A ordem de explicação será de acordo com a ordem que foi instanciada no código. Para começarmos veremos a função de printar o vetor no terminal onde é passado o vetor de elementos do tipo inteiro e o tamanho do vetor.

```
1  void printaVetor(int vetor[], int tamanho){
2          int i;
3          for (i = 0; i < tamanho; i++)
4                  printf("%d ", vetor[i]);
5          printf("\n\n");
6  }
```

Em seguida tempos a função para ordenação em tempo O(n), onde foi comprovado em sala pelo professor. Nela é passada também como parâmetro o vetor de elementos do tipo inteiro e o tamanho desse vetor.

```
1  void radixsort(int vetor[], int tamanho){
2          int i;
3          int *b;
4          int maior = vetor[0];
5          int exp = 1;
6
7          b = (int *)calloc(tamanho, sizeof(int));
8
9          for (i = 0; i < tamanho; i++){
10                 if (vetor[i] > maior)
11                         maior = vetor[i];
12         }
13
14         while (maior / exp > 0){
15                 int bucket[10] = {0};
16                 for (i = 0; i < tamanho; i++)
17                         bucket[(vetor[i] / exp) % 10]++;
18                 for (i = 1; i < 10; i++)
19                         bucket[i] += bucket[i - 1];
20                 for (i = tamanho - 1; i >= 0; i--)
21                         b[--bucket[(vetor[i] / exp) % 10]] = vetor[i];
22                 for (i = 0; i < tamanho; i++)
23                         vetor[i] = b[i];
24                 exp *= 10;
25         }
26         printf("Vetor ordenado: ");
27         printaVetor(vetor, tamanho);
28
```

```
29          free(b);
30  }
```

A solução encontrada para encontrar o elemento Ak foi uma busca binária que quebra o problema em vários problemas menores chamados recursivamente. Foi utilizado o meio do vetor como elemento principal a ser comparado para ser dividido o problema. Os parâmetros para a chamada da função é o vetor de elementos do tipo inteiro, a posição do primeiro elemento e do ultimo elemento do vetor, ambos do tipo inteiro.

```
1   int buscaBinaria(int vetor[], int inicio, int fim){
2           if(inicio == fim){
3            return inicio;
4           }
5           int meio = inicio + (fim - inicio) / 2;
6
7           if(vetor[meio + 1] < vetor[meio] && vetor[meio - 1] < vetor[meio]){
8                   return meio;
9           } else if (vetor[meio - 1] >= vetor[meio]){
10                  return buscaBinaria(vetor, inicio, meio);
11          } else if (vetor[meio + 1] >= vetor[meio]){
12                  return buscaBinaria(vetor, meio+1, fim);
13          }
14  }
```

O primeiro if que temos na linha 2 serve para o caso trivial onde o vetor tem um único elemento, logo, ele será o retorno.

```
1   if(inicio == fim){
2       return inicio;
3   }
```

Caso ele não seja um único elemento devemos achar o meio para começarmos a comparação. Foi utilizado uma forma de dividir o meio diferente para não estourar a variável caso o vetor seja muito grande, porém, é a mesma coisa de fazermos (fim - inicio) / 2.

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

Depois de encontrado o meio começamos a fazer as comparações e a divisão do problema em problemas menores. O if seguinte verifica se o Ak encontrado é exatamente o elemento do meio, nesse caso, o elemento do meio do vetor deve ser menor que elemento a sua esquerda e maior que o elemento a sua direita.

```
1   if(vetor[meio + 1] < vetor[meio] && vetor[meio - 1] < vetor[meio]){
2       return meio;
3   }
```

Caso não seja o elemento do meio o algoritmo irá verificar para que lado do vetor a chamada recursiva irá ser chamada. Caso o elemento do meio seja menor ou igual ao elemento da esquerda, significa que o elemento Ak está ao lado esquerdo do desse vetor, logo ele irá fazer a chamada passando como parâmetro o vetor começando em inicio e o fim sendo o meio do vetor original.

```
1   else if (vetor[meio - 1] >= vetor[meio]){
2           return buscaBinaria(vetor, inicio, meio);
3   }
```

Por fim, caso o elemento do meio seja menor ou igual ao elemento da sua direita, significa que o elemento Ak está do lado direito do vetor, logo ele irá fazer a chamada dividindo o problema para o lado direito do vetor original passando o parâmetro o vetor começando do meio + 1 e o fim sendo o fim do vetor original. Veja a junção de todos os ifs:

```
1   if(vetor[meio + 1] < vetor[meio] && vetor[meio - 1] < vetor[meio]){
2           return meio;
3   } else if (vetor[meio - 1] >= vetor[meio]){
4           return buscaBinaria(vetor, inicio, meio);
5   } else if (vetor[meio + 1] >= vetor[meio]){
6           return buscaBinaria(vetor, meio+1, fim);
7   }
```

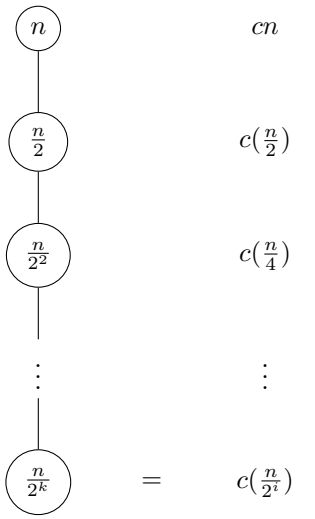
Na função main temos a criação do vetor e seu tamanho utilizando a função sizeof. Em seguida printamos o formato que o vetor está para melhor visualização e printamos o vetor na linha de comando. Em seguida declaramos um inteiro k que irá receber a posição de Ak encontrado desse vetor.

```
int main(){
    int vetor[] = {1,2,3,7,6,5,4,3,2,1};
    int tamanho_vetor = sizeof(vetor) / sizeof(vetor[0]);
    printf("Formato do vetor:  A1< A2< ...< Ak> Ak+1> Ak+2> ... > An \n\n");

    printf("Vetor recebido: ");
    printaVetor(vetor, tamanho_vetor);

    int k = buscaBinaria(vetor, 0, tamanho_vetor - 1);
    printf("k encontrado: %d\n", k);
    printf("A[k] encontrado: %d\n\n", vetor[k]);

    radixsort(vetor, tamanho_vetor);
    return 0;
}
```

Após encontrado o k, printamos esse valor, o elemento nessa posição e chamamos a função radixsort() com seus parâmetros para a ordenação e printamos o vetor ordenado.

4 Análise do algoritmo

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



$O(\log n)$

Quantidade de nós por nível: 1

Custo de cada nível: cn

Quantidade de níveis: $\log_2 n$

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

$i = \log_2 n$

Quantidade de nós no ultimo nível: 1

Custo do ultimo nível: 1

Custo total: $\log_2 n$