UNIFEI - UNIVERSIDADE FEDERAL DE ITAJUBÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

SIN110 - ALGORITMOS E GRAFOS

RESOLUÇÃO DOS EXERCÍCIOS E09 DO DIA 16/10/2015

ITAJUBÁ

2015

Exercícios E09 – 16/10/15

Aluna: Karen Dantas **Número de matrícula:** 31243

1)

```
int encontra_soma_x (float *vet, float x, int tam, int inicio, int fim){
1        if(inicio == tam)
2                return achou;
3        else{
4                if (tam==1){
5                        achou = 1;
6                        return achou;
7                }
8                else{
9                         if(fim > 0){
10                                if (x == vet[inicio] + vet[fim]){
11                                        achou = 1;
12                                        return achou;
13                                }
14                                encontra_soma_x(vet, x, tam, inicio, fim-1);
15                        }
16                        encontra_soma_x(vet, x, tam, inicio+1, tam-1);
17                }
18        }
19 }
```

Vale ressaltar que fiz o programa em linguagem C, logo, a primeira posição do vetor começa em 0 e não em 1. E a variável 'achou' é uma variável global que tem valor inicial 0.

Demonstração por indução:

- <u>Base indutiva</u> (inicio == tam): esse caso é verificado na linha 1, se o índice 'inicio' do vetor for igual ao tamanho do vetor ('tam') quer dizer que todos os números do vetor foram somados entre si. Caso haja apenas um elemento no vetor e ele for igual a 'x', é retornada a variável 'achou' com valor verdadeiro. Isso é feito na linha 4.
- <u>Hipótese indutiva</u>: suponho que, para vetores com n números reais sendo $n \geq 1$, a função encontra_soma_x retornará 1 caso haja dois números que se somados resultam em 'x' e 0 caso contrário.
- <u>Passo indutivo</u>: quero então provar que a função encontra_soma_x funciona corretamente para uma entrada com um vetor com n elementos. Enquanto a variável 'fim' for maior que 0, será executada a linha 10 verificando se o número que está na posição 'inicio' somado ao número que está na posição 'fim' é igual a variável 'x'. Se não for, as chamadas recursivas das linhas 14 e 16 vão "andando" com os marcadores de posição

‘inicio’ e ‘fim’ até que se encontre uma soma entre dois números igual a ‘x’ ou se constate que não há soma que resultará em ‘x’.

Fórmula: $T(n) = \Theta(1)$, se $n=1$

$T(n) = T(\lfloor n/2 \rfloor) + O(n)$, se $n > 1$

Utilizando o teorema Mestre, identifiquei nessa equação o terceiro caso analisado na equação geral de recorrências: $f(n) = af(n/b) + g(n)$, com $a < b^k$, com as constantes $a = 1$, $b = 2$ e $k = 1$, tendo como resultado: $T(n) = O(n)$.

---

2) EncontraCelebridade (S, Matriz)

```
1    se |S| = 1
2          então k ← Matriz[i, j]
3    senão
4          se (Matriz[i, j] = 1)
5                então aux ← i
6          senão aux ← j
7          S ← tira aux de S
8          k ← EncontraCelebridade (S, Matriz)
9          se k > 0 então
10               se (Matriz[aux, k] ≠ 1) ou (Matriz[k, aux] ≠ 0)
11                     então k ← 0
12   devolve k
```

Considerei que i e j são variáveis globais que indicam quaisquer duas pessoas do conjunto dado.

Demonstração por indução:

- Base indutiva (|S| = 1): esse caso é verificado na linha 1. Se o conjunto ‘S’ tiver apenas um elemento, ele é celebridade.
- Hipótese indutiva: suponho que, para dado conjunto ‘S’, a função EncontraCelebridade retornará a celebridade do conjunto ‘S’ se houver e, caso contrário, retornará 0.
- Passo indutivo: quero então provar que a função EncontraCelebridade funciona corretamente para dado conjunto ‘S’. Na linha 4, é verificado se o elemento na posição Matriz[i, j] é igual a 1. Se for, a pessoa ‘i’ não é celebridade, caso contrário, a pessoa ‘j’ não é celebridade. Na linha 7, é tirada a pessoa ‘aux’ do conjunto ‘S’ e ‘k’ recebe a chamada recursiva da função. Na linha 9, é verificado se ‘k’ é maior que 0, se sim, quer dizer que ‘k’ pode ser celebridade, assim, é verificado na linha 10 se Matriz[aux, k] é diferente de 1 ou se Matriz[k, aux] é diferente de 0. Em caso afirmativo, o ‘k’ recebe 0, ou seja, não há celebridade no conjunto, caso contrário, é retornado a celebridade ‘k’.

Esse algoritmo como pedido no exercício tem comportamento linear, pois, vai sendo removida do conjunto ‘S’ (dentre as n), uma pessoa que foi constatado não ser celebridade.

---

3) int BuscaTernaria (int *vet, , int x, int esq, int dir){

```
1       if (esq == dir){
2           if (x == vet[esq])
3               return esq;
4           else
5               return -1;
6       }
7       else {
8           meio = (( ⌊esq+dir⌋ )/3)+1;
9           if (x == vet[meio])
10              return meio;
11          else
12              if (x < vet[meio])
13                  return BuscaTernaria (vet, x, esq, meio-1);
14              else
15                  return BuscaTernaria (vet, x, meio+1, dir);
16       }
17 }
```

Vale ressaltar que fiz o programa em linguagem C, logo, a primeira posição do vetor começa em 0 e não em 1. E a variável ‘meio’ é uma variável global.

Demonstração por indução:

- Base indutiva (esq == dir): esse caso é verificado na linha 1, se o índice ‘esq’ do vetor for igual ao índice ‘dir’ do vetor quer dizer que, ou número não foi encontrado e será retornado -1, ou que há apenas um elemento no vetor e, nesse caso, se ele for igual a ‘x’ é retornada a sua posição na linha 3, caso contrário, é retornado –1.
- Hipótese indutiva: suponho que, para vetores ordenados com n números, a função BuscaTernaria dividirá vetor em dois conjuntos com n/3 e 2n/3 elementos e retornará a posição do número ‘x’ caso ele seja encontrado e retornará -1 caso contrário.
- Passo indutivo: quero então provar que a função BuscaTernaria funciona corretamente para uma entrada com um vetor ordenado com n elementos. Através da linha 8, é encontrado o novo ‘meio’ do vetor dividindo-se a soma da esquerda com a direita por 3 e, como a linguagem C considera o piso da divisão, soma-se 1 ao resultado para obter-se o teto. Na linha 9, é verificado se o número que está na posição ‘meio’ é igual a ‘x’ e, caso seja, é retornada a posição de ‘x’ senão é verificado se ‘x’ é maior ou menor que o número na posição ‘meio’. Se for menor faz-se uma chamada recursiva passando como ‘dir’ a

variável ‘meio’ menos 1, caso contrário, faz-se uma chamada recursiva passando como ‘esq’ a variável ‘meio’ mais 1.

<u>Fórmula</u>: $T(n) = \boldsymbol{\theta}(1)$, se $n=1$

$T(n) = T(\lfloor n/3 \rfloor) + \boldsymbol{\theta}(1)$, se $n > 1$

Utilizando o teorema Mestre, identifiquei nessa equação o segundo caso analisado na equação geral de recorrências: $f(n) = af(n/b) + g(n)$, com $a = b^k$, com as constantes $a = 1$, $b = 3$ e $k = 0$, tendo como resultado: $T(n) = O(n^0 \log n)$, logo, $T(n) = O(\log n)$.

<u>Busca Binária:</u>

<u>Fórmula:</u> $T(n) = \boldsymbol{\theta}(1)$, se $n = 1$

$T(n) = T(\lceil n/2 \rceil + \boldsymbol{\theta}(1)$, se $n > 1$

Utilizando o teorema Mestre, identifiquei nessa equação o segundo caso analisado na equação geral de recorrências: $f(n) = af(n/b) + g(n)$, com $a = b^k$, com as constantes $a = 1$, $b = 2$ e $k = 0$, tendo como resultado: $T(n) = O(n^0 \log n)$, logo, $T(n) = O(\log n)$.

Portanto, pode-se concluir que as buscas binárias e ternárias possuem a mesma complexidade no pior caso.

---

4) BuscaEmVetorInfinito (vet, x)

```
 se (vet[i] != ∞ e achou != -1)
        i=i+1
        se (vet[i+1] = ∞)
                então dir ← i
                        achou ←1
        senão devolve BuscaEmVetorInfinito (vet, x)
senão se (esq = dir)
            então se x = vet[esq] então devolve esq
                        senão devolve -1
        senão meio ← ⌊(esq+dir)/2⌋
        se (x = vet[meio])
                então devolve meio
        senão se (x < vet[meio])
                        então dir ← meio -1
                senão
                        então esq ← meio + 1
                BuscaEmVetorInfinito (vet, x)
```

<u>Explicação:</u> Esse algoritmo, primeiramente, irá buscar o índice do último elemento do vetor antes dos infinitos e, após encontrado esse índice, irá fazer busca binária para encontrar o

número ‘x’. A variável ‘dir’ é uma variável global que é inicializada na linha 4. As variáveis ‘esq’, ‘i’ e ‘achou’ são variáveis globais que, inicialmente, possuíam valores 1, 0 e -1, respectivamente.

Demonstração por indução:

- Base indutiva (esq == dir): esse caso é verificado na linha 7, se o índice ‘esq’ do vetor for igual ao índice ‘dir’ do vetor quer dizer que ou número não foi encontrado e será retornado -1, ou que há apenas um elemento no vetor e, nesse caso, se ele for igual a ‘x’ é retornada a sua posição na linha 8, caso contrário, é retornado –1. Na linha 1, enquanto o elemento do vetor na posição ‘i’ for diferente de infinito e a variável achou for igual a -1 quer dizer que não foi encontrado o tamanho do vetor ainda.
- Hipótese indutiva: suponho que, para um vetor de tamanho infinito com n números ordenados sendo que desconheço esse n, a função BuscaEmVetorInfinito retornará a posição do número ‘x’ caso ele seja encontrado e retornará -1 caso contrário.
- Passo indutivo: quero então provar que a função BuscaEmVetorInfinito funciona corretamente para uma entrada com um vetor de tamanho infinito com n elementos ordenados. Na linha 1, é procurada a posição do último elemento do vetor antes dos infinitos através de condições e chamada recursiva da linha 6. A partir da linha 7 é feita a busca binária. Na linha 10 é encontrado o ‘meio’ do vetor dividindo-se a soma da esquerda com a direita por 2. Na linha 11, é verificado se o número que está na posição ‘meio’ é igual a ‘x’ e, caso seja, é retornada a posição de ‘x’ senão é verificado se ‘x’ é maior ou menor que o número na posição ‘meio’. Se for menor a variável ‘dir’ recebe a variável ‘meio’ menos 1 e faz-se uma chamada recursiva da função, caso contrário, a variável ‘esq’ recebe a variável ‘meio’ mais 1 e faz-se uma chamada recursiva da função.

Fórmula:

$$T(n) = \Theta(n), \quad \text{se } n=1$$
$$T(n) = T(\lfloor n/2 \rfloor) + \Theta(n), \quad \text{se } n>1$$

Utilizando o teorema Mestre, identifiquei nessa equação o segundo caso analisado na equação geral de recorrências: $f(n) = af(n/b) + g(n)$, com $a = b^k$, com as constantes a = 1, b = 2 e k = 1, tendo como resultado: $T(n) = O(n^1 \log n)$, logo, $T(n) = O(n \log n)$ que é proporcional a $O(\log n)$ conforme foi pedido no exercício.