Nome: Bruno Marchi Pires
Disciplina: Complexidade de Algoritmos - UDESC
Data: 16/06/2021

Observação: As resoluções dos exercícios 2, 3, 4, 6 encontram-se no final do PDF

Exercício 1:

O algoritmo recursivo com complexidade exponencial, apresentado pelo professor na questão, possui um gráfico aproximado ao **primeiro gráfico da imagem abaixo**.

O algoritmo não recursivo, cuja implementação está no arquivo “lista.c” enviado junto com este PDF, possui complexidade linear, e seu gráfico é o **segundo apresentado na figura abaixo.**

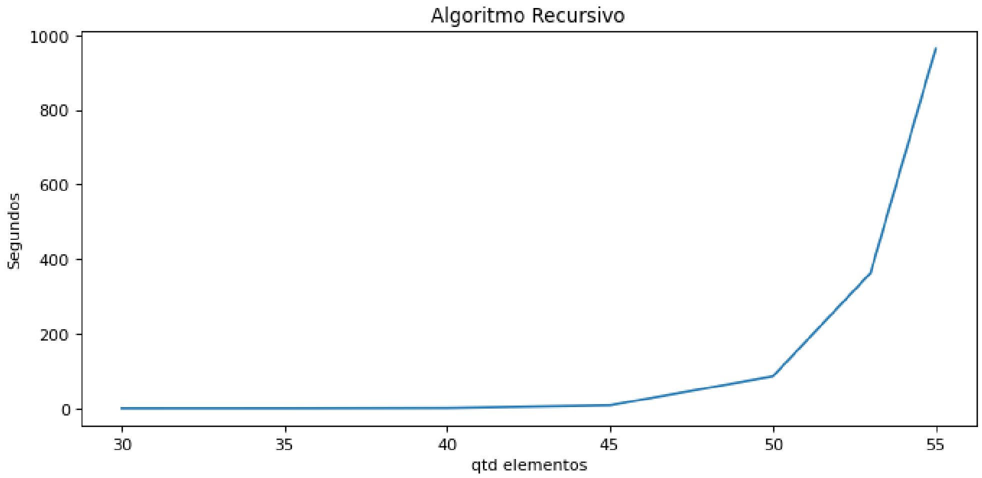
Algumas observações:

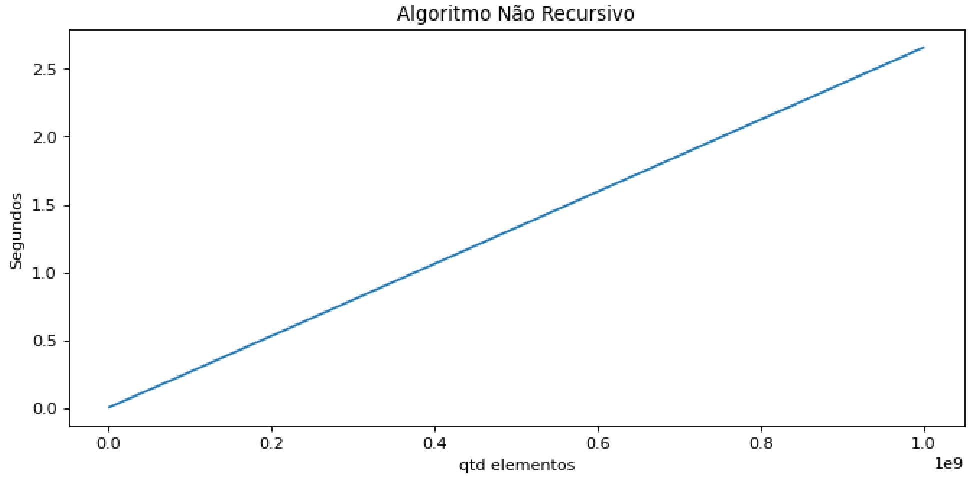
Os gráficos foram desenvolvidos utilizando a linguagem “python”, e as bibliotecas “numpy” e “matplotlib”, mas os dados expressos foram retirados dos algoritmos implementados em C.

Como a complexidade do segundo algoritmo é linear, a quantidade de elementos teve que ser alterada para melhor comparação, pois a máquina executava de maneira instantânea (cálculo do tempo retornava sempre 0), quantidades de elementos que o primeiro algoritmo demorava minutos, tornando impossível a comparação de “n”s iguais.

O eixo x do segundo gráfico, está em uma notação exponencial, onde os valores são multiplicados por 10^9.

Out[17]: [<matplotlib.lines.Line2D at 0x63ec328>]





Exercício 5:

O código está no arquivo chamado “lista.c”, enviado juntamente com este PDF.

Função 1:

```c
Lista *insere_inicio(Lista *lista, int elemento)
{
    Lista *novo = (Lista *)malloc(sizeof(Lista));
    novo->elem = elemento;
    novo->ptr = lista;
    return novo;
}      You, 18 minutes ago • .
```

Função 2:

```c
Lista *insere_final(Lista *lista, int elemento)
{
    Lista *novo = (Lista *)malloc(sizeof(Lista));
    novo->elem = elemento;
    novo->ptr = NULL;
    Lista *aux = lista;

    while (aux->ptr != NULL)
    {
        aux = aux->ptr;
    }
    aux->ptr = novo;
    return lista;
}
```

A complexidade de tempo da função 1, para inserção de elementos no início de uma lista encadeada, é igual a O(1). Todas as operações, sendo elas declaração, alocação e atribuições de variáveis possuem complexidade O(1). Como existe um ponteiro que identifica sempre a primeira posição da lista, qualquer nova inserção exige poucas manipulações e movimentos, não sendo necessário, por exemplo, percorrer a lista.

A complexidade de tempo da função 2, para inserção de elementos no final de uma lista encadeada, é igual a O(n). As operações de declaração, alocação e atribuições de variáveis possuem complexidade O(1), na função ainda existe um “while (aux->ptr != NULL)”, que percorre a lista inteira, até o seu último elemento, para que posteriormente possamos realizar as operações que de fato inserem o novo elemento no final da lista. Este while citado anteriormente possui complexidade igual a O(n), por percorrer toda a lista. Portanto, em suma, temos diversas linhas com complexidade O(1), e um while O(n), sendo possível afirma que a complexidade da função 2 é O(n).

2 →

```
INT POTENCIA (INT b, INT e){
    INT R;                          O(1)
    IF(e==0)                        O(1)
        RETURN 1;
    R = POTENCIA(b, e/2);           T(n/2)
    IF(e%2==0)
                                    O(1)
        RETURN R*R;
    ELSE
                                    O(1)
        RETURN R*R*b;
}
```

COMPLEXIDADE DE ESPAÇO = $O(1)$

$T(n) = T(n/2) + 1 \rightarrow T(1) = 1$

$T(n) = [T(n/2^2) + 1] + 1$

$T(n) = T(n/2^3) + 3$

ooo $\rightarrow \log n$

$T(n) = T(n/2^K) + K$

$T(n) = 1 + \log(n)$ //

PORTANTO

$T(n) = O(\log n) \rightarrow$ TEMPO

3 ⓐ $T(n) = T(n/2) + n$

$T(1) = 1$

ALGUMAS SUBSTITUIÇÕES

$$T(n) = T(n/2) + n$$

$$T(n) = T(n/2^2) + n/2 + n$$

$$T(n) = T(n/2^3) + n/2^2 + n/2 + n$$

$$\ldots$$

$$T(n) = T(n/2^k) + n\sum_{i=0}^{k-1} 1/2^i \quad \text{I}$$

$$\frac{n}{2^k} = 1 \Rightarrow n = 2^k$$

$$\sum_{i=0}^{k-1} 1/2^i = 1/2 + 1/4 + 1/8 + \ldots + 1/2^{k-1} + \underbrace{1/2^k}_{\text{PERTURBAÇÃO}}$$

$$\left[\sum_{i=0}^{k-1} 1/2^i\right] + 1/2^k = 1/2 + 1/2\sum_{i=0}^{k-1} 1/2^i$$

$$\left[\sum_{i=0}^{k-1} 1/2^i\right] - \left[1/2\sum_{i=0}^{k-1} 1/2^i\right] = 1/2 - 1/2^k$$

$$\frac{1}{2}\sum_{i=0}^{k-1} 1/2^i = 1/2 - 1/2^k$$

$$\sum_{i=0}^{k-1} 1/2^i = 1 - \frac{1}{2^{k-1}} \quad //$$

VOLTANDO EM I

$$T(n) = 1 + n\left(1 - \frac{1}{2^{k-1}}\right)$$

$$T(n) = 1 + n\left(1 - \frac{2}{2^k}\right)$$

$$T(n) = 2n - 1 \quad //$$

3b) $T(n) = 2T(n-1) + n$

$T(1) = 1$

Algumas Substituições:

$$T(n) = 2T(n-1) + n$$
$$T(n) = 2[2T(n-2) + n-1] + n$$
$$T(n) = 4T(n-2) + 2(n-1) + n$$
$$T(n) = 8T(n-3) + 4(n-2) + 2(n-1) + n$$

...

∴ Temos → $T(n) = 2^{n-1}\,T(1) + \sum_{i=0}^{n-2} 2^i (n-i)$ → (I)

Encontrando a Fórmula Fechada do Somatório de (I):

$$\sum_{i=0}^{n-2} n2^i - i2^i \rightarrow n\sum_{i=0}^{n-2} 2^i - \sum_{i=0}^{n-2} i2^i \quad \text{(II)}$$

→ Temos 2 somatórios, Precisamos Resolver Ambos

$$\sum_{i=0}^{n-2} 2^i \rightarrow 2^{n-1} - 1 \rightarrow \frac{a_1(q^n - 1)}{q-1}$$ → Fórmula P.G.

(III)

$$\sum_{i=0}^{n-2} i2^i \rightarrow 0 + 2 + 2\cdot 2^2 + 3\cdot 2^3 + 4\cdot 2^4 + \dots + (n-3)\cdot 2^{n-3} + (n-2)\cdot 2^{n-2} + (n-1)\cdot 2^{n-1}$$

↳ Perturbação

$$\left[\sum_{i=0}^{n-2} i2^i\right] + (n-1)\cdot 2^{n-1} = 0 + \sum_{i=0}^{n-2} (i+1)2^{i+1} \rightarrow 2\sum_{i=0}^{n-2} i2^i + 2\sum_{i=0}^{n-2} 2^i$$

$$\left[\cancel{\sum_{i=0}^{n-2} i2^i}\right] + (n-1)\cdot 2^{n-1} = \cancel{2}\sum_{i=0}^{n-2} i2^i + 2\sum_{i=0}^{n-2} 2^i \rightarrow 4^{n-1} - 2$$ → Duas Vezes (III)

$$\sum_{i=0}^{n-2} i2^i = (n-1)\cdot 2^{(n-1)} - 4^{n-1} + 2 \rightarrow -2^{n-1}(3+n) + 2 \quad \text{(IV)}$$

Substituindo IV e III em II:

$$\sum_{i=0}^{n-2} 2^i (n-i) = n\cdot(2^{n-1} - 1) - (n2^{n-1} - 2^{n-1} - 4^{n-1} + 2)$$
$$= 2^{n-1} + 4^{n-1} - n - 2 \quad \text{(V)}$$

Substituindo V em I:

$$T(n) = 2^{n-1} + 2^{n-1} + 4^{n-1} - n - 2 \rightarrow \boxed{T(n) = 8^{n-1} - n - 2}$$

3 ⓒ $T(n) = 4T(n/2) + n$

$T(1) = 1$

ALGUMAS SUBSTITUIÇÕES

$T(n) = 4T(n/2) + n$

$T(n) = 4[4T(n/2^2) + 2n] + n$

$T(n) = 4^2 T(n/2^2) + 2n + n$

$T(n) = 4^3 T(n/2^3) + 4n + 2n + n$

...

$T(n) = 4^k T(n/2^k) + \sum_{i=0}^{k-1} 2^i n$

RESOLVIDO NA 3b

↳ $(2^k - 1)$

$\frac{n}{2^k} = 1$; $n = 2^k$; $k = \log n$

$T(n) = 4^{\log n} + n \cdot (n-1)$

$T(n) = 4^{\log n} + n^2 - n$ //

3 (d)

$$T(n) = T(n/2) + \log_2 n$$

$$T(1) = 1$$

Algumas Substituições

$$T(n) = T(n/2) + \log n$$

$$T(n) = [T(n/2^2) + \log n/2] + \log n$$

$$T(n) = [T(n/2^3) + \underbrace{\log n/2^2] + \log n/2 + \log n}_{\log\left(\frac{n^3}{2^3}\right)}$$

$\therefore$ Temos $\rightarrow$ $T(n) = T(n/2^k) + \log\left(\frac{n^k}{2^k}\right) \rightarrow \log(n^k) - \log(2^k)$

$$\frac{n}{2^k} = 1 \;;\; n = 2^k \;;\; k = \log n$$

$$T(n) = 1 + \log(n^{\log n}) - \log(n) \,//$$

4 $\rightarrow$ $T(n) = 4T(n/2) + n$ PROVAR QUE É $T(n) = 2n^2 - n$

$T(1) = 1$

HIPOTESE DE INDUÇÃO

$$T(2^k) = 2\cdot(2^k)^2 - n$$

$$T(2^k) = 2^{k+1} - 2^k \;//$$

BASE DA INDUÇÃO

$k=0$

$$T(1) = 2\cdot1 - 0$$

$$T(1) = 1$$ PROVADO!

$$1 = 1 \checkmark$$

PASSO INDUTIVO

$\nearrow T(2^k) \rightarrow$ hipótese

$$T(2^{k+1}) = 4T(2^{k+1}/2) + 2^{k+1}$$

$$2(2^{k+1})^2 - 2^{k+1} = 4\cdot(2^{k+1} - 2^k) + 2^{k+1}$$

$$2\cdot(2^{2k+2}) - 2^{k+1} = 2^{2k+3} \underbrace{- 2^{k+2} + 2^{k+1}}_{\rightarrow 2^{k+1}}$$

$$2^{2k+3} - 2^{k+1} = 2^{k+3} - 2^{k+1} \;//$$ PROVADO!

6 →

```
VOID IMPRIMIR (struct ARVORE R){
    IF(R != NULL){              → O(1)
        IMPRIMIR(R->esq)        → T(n/2)
        PRINTF(...)             → O(1)
        IMPRIMIR(R->DIR)        → T(n/2)
    }
}
```

$T(n) = 2T(n/2) + 1$

$T(1) = 1$ → NÃO POSSUI RAMIFICAÇÕES

$T(n) = 2T(n/2) + 1$

$\frac{n}{2^k} = 1 \rightarrow k = \log n$

$T(n) = 2 \cdot [2T(n/2^2) + 1] + 1$

$T(n) = 2 \cdot [2 \cdot [2T(n/2^3) + 1] + 1] + 1$

$T(n) = 2^3 T(n/2^3) + 3$

...

$T(n) = 2^k T(n/2^k) + k$

$T(n) = 2^{\log n} + \log n$

$T(n) = n + \log n$

↳ TEMPO

COMPLEXIDADE DE ESPAÇO = $O(n)$

O ESPAÇO NECESSÁRIO É IGUAL AO TAMANHO DA ÁRVORE DE ENTRADA.

A IMPLEMENTAÇÃO DO ALGORITMO NÃO RECURSIVO NECESSITA TAMBÉM DA IMPLEMENTAÇÃO DE UMA PILHA, O CÓDIGO ESTAVA COM PROBLEMAS, QUE NÃO RESOLVI A TEMPO, POR ISSO OPTEI POR NÃO ENVIAR.