SCC0201 - Introdução à Ciência da Computação II
Universidade de São Paulo - Instituto de Ciências Matemáticas e de Computação
Outubro, 2020

Prova 1 - Análise de Algoritmos

Diógenes Silva Pedro - 11883476

1. Formule a equação de recorrência que representa o número de com-parações e operações aritméticas realizadas no pior caso, excluindo da contagem operações relativas a linhas com instruções **for** e **while**. Considere a variável a como sendo as operações aritméticas, e c as comparações.

   Antes de fazermos a equação de recorrência transformando o algoritmo em recursivo faremos a análise no algoritmo de forma iterativa, para assim após transformarmos o algoritmo em recursivo podermos comparar as funções.

   Temos o código iterativo:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
    if (i > j) {                                //c
        mat[i][j]= (i+j)/2.0;                   //2a
    }
    else if (i < j) {                           //c
        mat[i][j]= (i+j)/4.0;                   //2a
    }
    else {
        double soma = 0.0;

        for (int a = 0; a < n; a++) {
            if (i-1 >= 0)                      //a + c
                soma += mat[i-1][a];           //2a
        }

        mat[i][j]= soma;

        }
    }
}
```

Figura 1: Código do Moa

A seguir temos uma matriz que nos auxiliará a explicar o código:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Figura 2: Matrix m × n

Temos 4 casos possíveis:

(a) Onde $i = j$ e $i = 0$ esse caso aparecerá 1 vez por matriz independente de seu tamanho, na 1° posição da matriz.

  i. Temos então para esse caso as operações equivalentes as comparações do **if** e **else if** sendo $2c$ além de $n(a + c)$ referente ao if do for mais interno.

  ii. Assim para o caso 1 teremos $n(a + c) + 2c$ operações.

(b) Para as próximas diagonais sempre poderemos somar a linha anterior já que $i > 0$.

  i. Como estamos falando de uma diagonal, novamente teremos $2c$ equivalente ao **if** e **else if** porém agora dentro do for mais interno teremos $n(3a + c)$ operações.

  ii. Assim para o caso 2 teremos $n(3a + c) + 2c$. operações.

(c) Para $i < j$ o **if** será comparado e entraremos no **else if**.

  i. Assim para o caso 3 teremos $2a + 2c$ operações.

(d) Para $i > j$ entraremos no **if**.

  i. Assim para o caso 4 teremos $2a + c$ operações.

Veremos agora quantos casos aparecem para uma matriz genérica $n * n$:

$$\begin{bmatrix} 1 & 3 & \dots & 3 \\ 4 & 2 & \dots & 3 \\ \vdots & \vdots & \ddots & 3 \\ 4 & 4 & 4 & 2 \end{bmatrix}$$

Podemos perceber na matriz acima que o número de vezes que o caso 1 aparece é 1

Sabemos que em uma matriz $n * n$ o número de diagonais é igual a n, então para o número de vezes que o caso 2 aparece é $n - 1$, pois já tiramos a diagonal do caso 1.

Podemos reparar olhando a matriz a cima que o número de vezes que o caso 3 e o caso 4 aparecem são iguais. Para calcularmos quantas são essas vezes usaremos um exemplo numéricos antes da fórmula genérica.

Para uma matriz 4×4 teremos 16 casas sendo 4 delas diagonais então teremos $16 - 4 = 12$ casas disponíveis para os casos 3 e 4 porém ainda temos que dividir por 2 como podemos ver acima.

$$\text{Logo: } \frac{12}{2} = 6. \text{ Temos } \frac{4 \times x}{2} = 6$$

$$\therefore x = 3$$

Agora para a fórmula genérica temos então:

$$\frac{n(n-1)}{2} = \text{n° de casos 3 ou 4}$$

Agora que sabemos o número de ocorrências dos casos para uma matriz quadrada de tamanho **n** qualquer juntaremos todas as operações para então gerarmos a formula geral da função iterativa. Seguiremos a ordem do algoritmo e adicionaremos a quantidade de operações do **if**, do **else if** e do **else** nessa sequência. Temos então a seguinte equação:

$$f(n) = \frac{n\ (n-1)\ (2a+c)}{2} + \frac{n\ (n-1)\ (2a+2c)}{2} + 1(n(a+c)+2c) + (n-1)(n(3a+c)+2c)$$

$$f(n) = \frac{(n^2-n)(2a+c)}{2} + \frac{(n^2-n)(2a+2c)}{2} + an + ac + 2c + (n-1)(3an + cn + 2c)$$

$$f(n) = \frac{4an^2 + 3cn^2 - 4an - 3cn}{2} + an + ac + 2c + 3an^2 + cn^2 + 2cn - 3an - cn - 2c$$

$$2f(n) = 4an^2 + 3cn^2 - 4cn - 3cn + 2an + 2cn + 4c + 6an^2 + 2cn^2 + 4cn - 6an - 2cn - 4c$$

$$2f(n) = 10an^2 + 5cn^2 - 8an + cn$$

$$f_{iterativa}(n) = 5an^2 + \frac{5cn^2}{2} - 4an + \frac{cn}{2}$$

Faremos agora a análise pelo ponto de vista da recorrência, com isso precisamos transformar nossa função iterativa em uma função recursiva.

Transformando a função em recursiva:

```c
void recursiveAlloc(double **mat, int n, int i){

    if (i >= n) return;

    for (int j = 0; j < n; j++) {
        if (i > j) {                          //c
            mat[i][j]= (i+j)/2.0;             //2a
        }
        else if (i < j) {                     //c
            mat[i][j]= (i+j)/4.0;             //2a
        }
        else {
            double soma = 0.0;
            for (int a = 0; a < n; a++) {
                if (i-1 >= 0)                 //a + c
                    soma += mat[i-1][a];      //2a
            }

            mat[i][j] = soma;
        }
    }

    recursiveAlloc(mat, n, i + 1);

    //Obs: Como o próprio exercício diz no enunciado os fors e whiles deverão
    //ser excluidos da contagem. Por isso a operação aritmética da chamada
    //recursiva, além do comparação do caso base, não serão contadas para a
    //análise do algoritmo. Pois essas seriam as mesmas operações do for mais
    //externo do algoritmo interativo mostrado anteriormente.
}
```

Figura 3: Código Recursivo

Como sabemos o número de casos será igual a função iterativa, porém agora teremos que mudar a hora da análise pois teremos que olhar a nossa função para cada linha da matriz.

Lembrando da nossa matriz de casos poderemos utiliza-lá para tentar escrever uma equação utilizando todos as operações que a função possa a vir fazer.

$$\begin{bmatrix} 1 & 3 & \dots & 3 \\ 4 & 2 & \dots & 3 \\ \vdots & \vdots & \ddots & 3 \\ 4 & 4 & 4 & 2 \end{bmatrix}$$

Começaremos pelo começo do nosso vetor na linha 0 da matriz, teremos:

$$f(0) = 0(2a + c) + (n - 1)(2a + 2c) + 1(n(a + c) + 2c) + f(1)$$

Conseguimos entender o porque da função acima pois não vemos nenhum caso 4 na primeira linha, além de que como a nossa primeira linha é especial usaremos o caso

1 nela, ao invés do caso 2, que será utlizada no resto das fórmulas.

Agora avançaremos para f(1), ela conterá as operações da 2° linha além das operações da 1° primeira. A partir de agora usaremos do caso 2 para as diagonais, assim temos:

$$f(1) = 1(2a + c) + (n - 2)(2a + 2c) + 1(a(3a + c) + 2c) + f(2)$$

$$f(2) = 2(2a + c) + (n - 3)(2a + 2c) + 1(a(3a + c) + 2c) + f(3)$$

Como podemos ver as operações são sempre realizadas antes da chamada recursiva da próxima função, o que implica em uma recursão de cauda.

Temos para a última linha da matriz a seguinte equação:

$$f(n - 1) = (n - 1)(2a + c) + 0(2a + 2c) + 1(n(3a + c) + 2c) + f(n)$$

Pela recursão de cauda f(n) será a função que retornará graças ao caso de borda, então f(n) conterá todas as operações da função recursiva, temos então:

Podemos também perceber um padrão na função pois o caso 4 aparece primeiramente 0 vezes e no final aparece (n-1) vezesm, pois em todas as vezes temos 1 diagonal. A quantidade de vezes em que o caso 3 e 4 aparecem serão calculados através da soma de PA. Temos então:

$$f_{recursiva}(n) = \sum_{i=0}^{n-1} i(2a + c) + \sum_{i=n-1}^{0} i(2a + 2c) + 2cn + an + cn + (n - 1)(3a + c)$$

$$\sum_{i=0}^{n-1} = \frac{(0 + n - 1)n}{2} = \frac{n^2 - n}{2}$$

$$\sum_{i=n-1}^{0} = \frac{(n - 1 + 0)n}{2} = \frac{n^2 - n}{2}$$

$$f_{recursiva}(n) = \frac{(n^2 - n)(2a + c)}{2} + \frac{(n^2 - n)(2a + 2c)}{2} + 2cn + an + cn + 3an^2 + cn^2 - 3an - cn$$

$$f_{recursiva}(n) = \frac{4an^2 + 3cn^2 - 4an - 3cn}{2} + 2cn + an + cn + 3an^2 + cn^2 - 3an - cn$$

$$2f_{recursiva}(n) = 4an^2 + 3cn^2 - 4an - 3cn + 4cn + 2an + 2cn + 6an^2 + 2cn^2 - 6an - 2cn$$

$$2f_{recursiva}(n) = 10an^2 + 5cn^2 - 8an + cn$$

$$\therefore f_{recursiva}(n) = 5an^2 + \frac{5cn^2}{2} - 4an + \frac{cn}{2}$$

2. Otimize a solução do código do melhor jeito que conseguir.
   Minha solução foi:

```c
void generateMatrix(double **matrix, int size, double sum){
    double realsum = 0;

    for (int i = 0; i < size; i++){
        sum = 0;
        for (int j = 0; j < size; j++){
            if (i == j){                          //c
                matrix[i][j] = realsum;
            }
            else if(i < j){                       //c
                matrix[i][j] = (i + j) / 4.0;     //2a
            }
            else
                matrix[i][j] = (i + j) / 2.0;     //2a

            sum += matrix[i][j];                  //a
        }
        realsum = sum;
    }
}
```

Figura 4: Código do Dio

3. Analizaremos agora o código visto acima a partir da contagem direta de operações.

   Definiremos os 3 casos possíveis:

   (a) Caso 1:
   Primeiramente checamos se estamos na diagonal, caso sim temos: $n(a + c)$ operações.

   (b) Caso 2:
   Agora checamos se $i < j$ caso sim temos: $n(3a + 2c)$ operações.

   (c) Caso 3:
   Nesse caso estamos no **else** logo $i > j$, assim temos: $n(3a + 2c)$ operações.

   Agora descobrimeros quantos aparições temos por caso:

   (a) Caso 1 (Diagonal):
   Sempre teremos 1 caso por linha logo $1(n(a + c))$

   (b) Caso 2 ($i < j$):
   Como descobrimos anteriormente teremos:

   $$\frac{n(n-1)(3a+2c)}{2}$$

   (c) Caso 3 ($i > j$):
   Igual ao caso acima. Vale ressaltar que o n vem do for mais externo, $n - 1$ é devido ao n do for mais interno menos o caso único da diagonal

Para a equação geral de contagem da função seguindo a sequência **if**, **else if** e **else**, temos:

$$f(n) = 1(n(a+c)) + \frac{n(n-1)(3a+2c)}{2} + \frac{n(n-1)(3a+2c)}{2}$$

$$f(n) = an + cn + \frac{(n^2-n)(3a+2c)}{2} + \frac{(n^2-n)(3a+2c)}{2}$$

$$f(n) = an + cn + \frac{3an^2+2cn^2-3an-2cn}{2} + \frac{3an^2+2cn^2-3an-2cn}{2}$$

$$f(n) = an + cn + \frac{6an^2+4cn^2-6an-4cn}{2}$$

$$2f(n) = 2an + 2cn + 6an^2 + 4cn^2 - 6an - 4cn$$

$$2f(n) = 6an^2 + 4cn^2 - 4an - 2cn$$

$$\therefore f(n) = 3an^2 + 2cn^2 - 2an - 2cn$$

Podemos olhar no gráfico a seguir a diferença entre o algoritmo original mostrado na questão 1 e o algoritmo mostrado na questão 2.

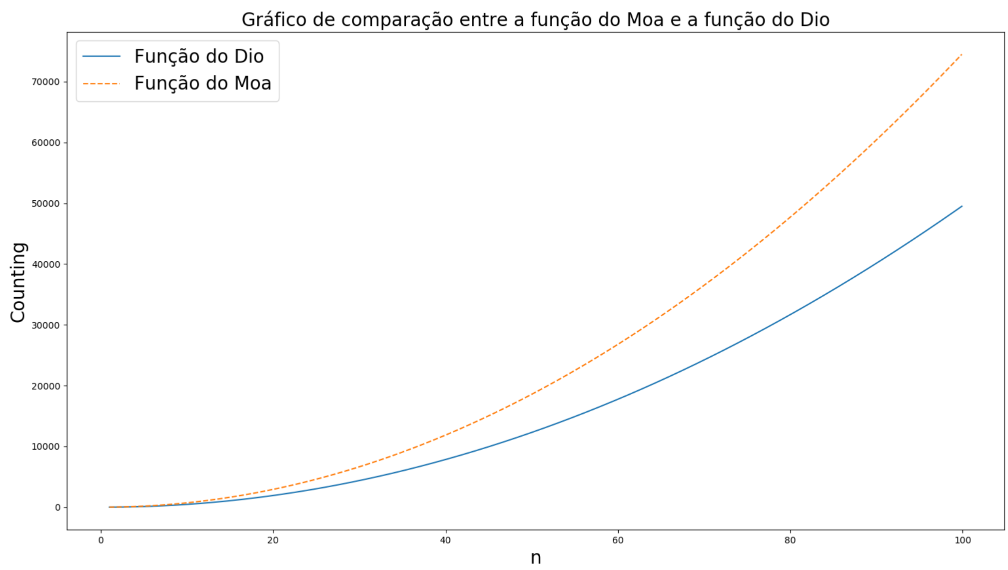

Figura 5: Gráfico de Comparação

Podemos ver que mesmo com poucas mudanças podemos ganhar uma otimização no algoritmo saindo de ≈ 70000 operações onde $n = 100$, para ≈ 50000 operações.