

**UNIVERSIDADE FEDERAL DE MINAS GERAIS**

**MATEUS FELIPE DA SILVA**

# **DOCUMENTAÇÃO PARA PROBLEMAS COMBINATÓRIOS**

Matemática Discreta

**Belo Horizonte**

**2017**

## DOCUMENTAÇÃO DO PROGRAMA SOMAX.C

Entrada do programa: Inteiro N menor que 20 e maior que 3 na primeira linha, N inteiros na linha seguinte separados por espaço.

Saída: Palavra “soma” seguida da maior soma encontrada nos sub-arranjo do arranjo de N números. Palavra “Índices” seguida os índices que estarão os números que geraram essa maior soma.

Caso de exclusão: Caso todos números dos índices lidos forem 0, “zero”, o programa imprimirá que a soma é 0 “Zero” e que “Todos os índices são negativos”.

Diagrama de interpretação: A lógica utilizada para solução do problema da soma máxima partiu da verificação dos números dos N índices do arranjo que foi lido.

Para fazer tal verificação, o programa verifica inicialmente índice por índice do arranjo, e acha o maior, salvando a posição deste, após realizar a verificação índice por índice, será verifica de dois em dois índices, achando a maior soma e guardando a posição, depois de três em três índices, até N por N índices (que no caso é quando o programa verifica todo o arranjo lido).

A implementação dessa verificação necessitou de 3 laços de repetição (Figura 1.1):

```
for(j = 0; j < tamanho; j++){
    for(k = 0; k < tamanho; k++){
        somaP = 0;
        for(i = j; i < tamanho-k; i++){
            somaP = somaP + sequencia[i];
        }
        if(somaP > soma){
            soma = somaP;
            inicioSequencia = j;
            fimSequencia = (i-1);
        }
    }
}
```

Figura 1.1

O laço interno inicia-se baseado no J que proveio do laço externo, dessa forma podemos alternar entre cada posição do arranjo, basta apenas verificar k por k índices para conseguirmos acessar soluções unárias, binários, até N-árias soluções.

A condição “IF” verifica a maior soma e então guarda as posições da maior soma encontradas de todos os conjuntos de números testados.

```

if(verifica == tamanho){
    printf("Soma: 0\n");
    printf("Todos os %cndices sao negativos\n",214);
}
else{

```

Figura 1.2

Tal repetição só é executada caso os números salvos nos N índices não forem 0 “Zero” (Figura 1.2), caso satisfeita a condição de ser 0, “zero”, o programa irá imprimir soma igual a zero, e explicará que todos os índices lidos foram iguais a zero.

```

printf("Soma: %d\n",soma);
printf("%cndices: %d a %d\n",214,inicioSequencia+1 , fimSequencia+1);

```

Figura 1.3

Por fim é impressa a maior soma que foi encontrada no programa e os índices que estão essa soma. Foi-se somado (+1) no resultado, pois inicializei o arranjo dos números a partir da posição 0, “zero”. Logo para corrigir apenas se foi necessário somar 1 no início de no fim da sequência.

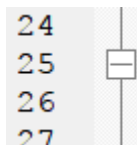
## DOCUMENTAÇÃO PARA O PROGRAMA QMAGICO.C

Entrada do programa: Um inteiro N entre 3 e 5.

Saída: o número de colunas do quadrado mágico, e a constante mágica do mesmo, e o quadrado mágico N x N resolvido.

A solução do quadrado mágico teve de ser dividida em duas para podermos abranger vários tamanhos N x N de soluções.

Foi dividido de maneira, ímpar e par (no caso de par, apenas resolverá para o caso N = 4), mas caso o número seja ímpar, resolverá para qualquer número igual ou maior que 3).

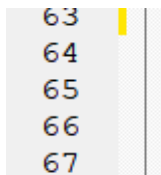


```
if (N % 2 == 0) {  
    /// MATRIX PAR
```

Figura 2.1

Tal divisão foi feita na linha 25 do programa, testando o resto de divisão do número N lido, podendo separar os números ímpares e pares do programa. (Figura 2.1)

Para resolver o quadrado ímpar foi-se utilizada a “técnica do caminhar constante”, que se baseia em cálculos matemáticos definidos que dão a solução para um quadrado mágico ímpar. Tal método consiste em definir o meio da primeira linha do quadrado mágico como sendo o número “1”. (Figura 2.2), linha 66.

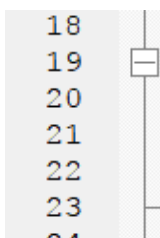


```
int meio = (N / 2);  
contador = 1;  
matrix[0][meio] = contador;
```

Figura 2.2

Definida a posição inicial, deveremos sempre caminhar na matriz no sentido para cima e para direita, preenchendo o próximo número, caso já exista um número nessa posição, deveremos caminhar para baixo na matriz em vez de para cima e para direita. (Figura 2.3).

Para verificar se já existia um número na posição desejada, antes de preencher a matriz, ela foi totalmente preenchida com zeros, assim caso a posição esteja com zero, significa que ela está vazia! (Figura 2.4)



```
for (COL = 0; COL < N; COL++) {  
    for (LIN = 0; LIN < N; LIN++)  
        matrix[COL][LIN] = 0;  
}
```

Figura 2.3

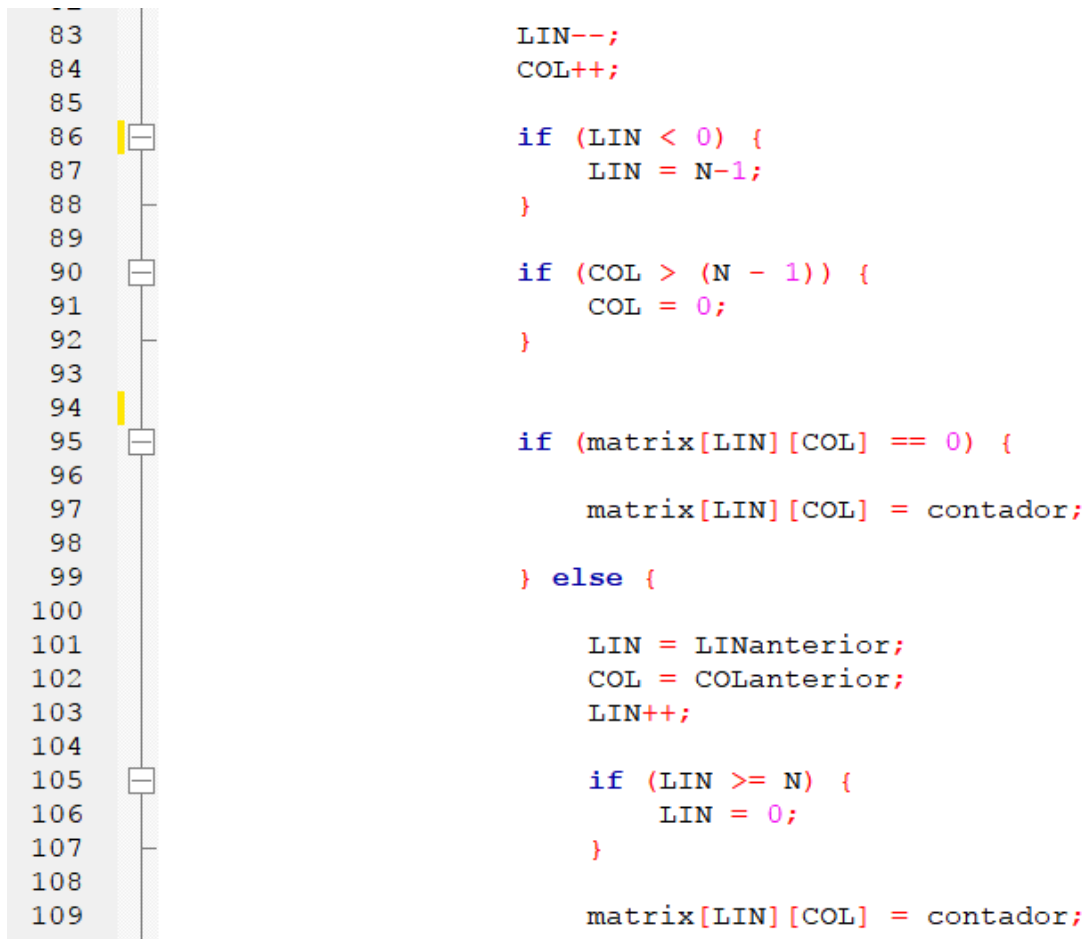


Figura 2.2

Por fim imprimimos a matriz resultado na tela! (Figura 2.4)

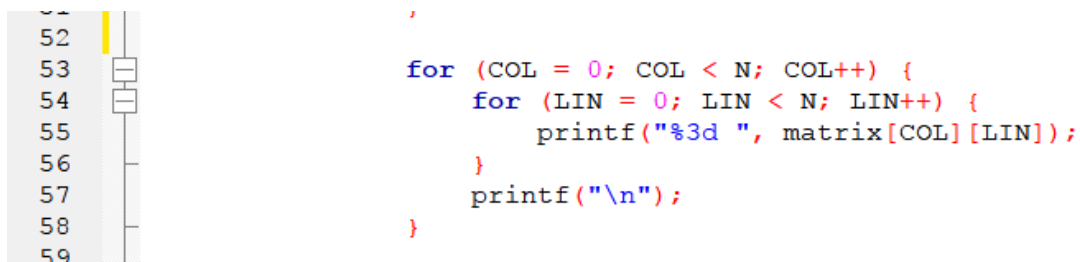


Figura 2.4

Para solução da matriz 4x4 foi utilizado o método das diagonais sequenciais, na qual se preenche apenas os índices das duas diagonais em ordem crescente.

Para realizar tal preenchimento, foi necessário verificar (para diagonal principal) se o índice da coluna era igual o índice a índice das linhas, no caso, estaremos verificando se a posição é  $A_{11}$   $A_{22}$   $A_{33}$   $A_{NN}$ , escolhendo assim a diagonal principal (Figura 2.5), e para

diagonal secundário, utilizou-se o método  $N - 1 - \text{índice da Linha}$ , que retorna os elementos,  $A_{1N} A_{2N-1} A_{3N-2}$ , escolhendo assim a diagonal secundária. (Figura 2.5)

```
if (COL == LIN) {  
    matrix[COL][LIN] = contador;  
} else if (COL == (N - 1 - LIN)) {  
    matrix[COL][LIN] = contador;  
}
```

Figura 2.5

Para finalizar o preenchimento, basta colocarmos nas posições ainda não preenchidas (As que estão com “0”), a sequência de números em ordem decrescente. Foi-se utilizado um contador regressivo para tal situação. (Figura 2.6)

```
} else if (matrix[COL][LIN] == 0) {  
    matrix[COL][LIN] = contador2;  
}
```

Figura 2.6

Por final apenas bastou imprimir a matriz. (Figura 2.4)