UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIAS DA COMPUTAÇÃO

Documentação - Trabalho Prático de Matemática Discreta
Turma TW

Aluno: Luís Gustavo Costa da Silva
Matrícula: 2019054803

Belo Horizonte
2019

1. Soma Máxima (somamax.c)

Para chegar em um algoritmo implementável com o objetivo de resolver o problema, analisei a estrutura de diversos vetores e suas somas máximas. Percebi que é impossível um sub-vetor resposta ter um número negativo em sua extremidade, ao mesmo tempo que é perfeitamente possível que tenham números negativos em sua constituição.

No início tinha pensado em um algoritmo que analisasse todos os sub-vetores possíveis e escolhesse um, porém a complexidade seria cúbica de tempo de execução, levando-me a pensar em outra forma de executar o algoritmo. Tinha em mente que um número negativo não poderia interromper o cálculo da soma máxima, e para iniciar a contagem do vetor o número precisaria ser positivo.

O algoritmo implementado não possui nenhuma função além da 'main'. A princípio ele requisita a entrada o usuário, 'n', representando o tamanho do vetor desejado. É esperado que a entrada sempre satisfaça a expressão " 3 ≤ n ≤ 20 ", ajustando a entrada de acordo com o valor digitado: ou igualando 'n' a 3 caso ele seja menor que 3 ou igualando a 20 caso ele seja maior que 20.

```c
#include <stdio.h>

int main(){

    int n = 0, r_init = 0, r_end = 0, init_aux = 1, i = 0;
    float entrada, s_total = 0, s_aux = 0;

    scanf("%d", &n);
    if(n < 3) n = 3;
    if( n > 20) n = 20;
```

Imagem 1: Biblioteca utilizada, alocação de variáveis e entrada *'n'*. As variáveis *"r_init"* representa o range (índice) inicial da resposta, *"r_end"* o final, enquanto *"s_total"* é a soma total. As variáveis *"s_aux"* e *"init_aux"* são auxiliares para a resolução do problema, e a variável *'i'* é de controle.

Em seguida, o código inicia seu bloco principal, um loop controlado que vai fazer a entrada dos valores do vetor, ao mesmo tempo que ele calcula a soma máxima do sub-vetor. Em tentativas anteriores, eu aloquei o vetor dinamicamente, mas percebi que não era estritamente necessário para a resolução do problema.

Após a entrada do valor contido teoricamente na posição 'i' do vetor, ele define a soma auxiliar como sendo o valor anterior de soma auxiliar (inicializada com 0) somado à entrada.

Em seguida, entra-se em um bloco condicional, onde ele analisa se a soma auxiliar é menor que zero e, caso verdade, reseta a soma auxiliar e define a posição inicial do sub-vetor, sendo ainda não definitivo. Em continuação, inicia-se outro bloco condicional, esse comparando a soma auxiliar com a total (inicializada com 0). Caso a soma auxiliar for maior que a total, a soma definitiva é atualizada, assim como os índices definitivos.

```c
    for(i = 0; i < n; i++){
        scanf("%f", &entrada);

        s_aux += entrada;

        if(s_aux < 0){
            s_aux = 0;
            init_aux = i+2;
        }
        if(s_aux >= s_total){
            s_total = s_aux;
            r_init = init_aux;
            r_end = i+1;
        }
    }
```

Imagem 2: Bloco principal do código somamax.c.

Por fim, o código possui um bloco condicional que analisa a soma total, para ajustar os valores definitivos para o caso trivial, onde todas as entradas do vetor são menores que zero.

```c
    if(s_total <= 0){
        s_total = 0;
        r_init = 1;
        r_end = n;
    }

    printf("Soma: %.1f\nIndices: %d a %d\n", s_total, r_init, r_end);
    return 0;
}
```

Imagem 3: Ajuste para o caso trivial, e saída final do código.

O algoritmo implementado possui complexidade linear em todos os casos, sendo melhor representada pela notação assintótica Θ(n), para o tempo de execução. Sua complexidade de espaço é Θ(1), pois o espaço ocupado é independente da entrada.

2. Quadrado Mágico (qmagico.c)

A princípio, o algoritmo para a implementação do quadrado mágico foi muito difícil para a implementação. Com isso, montei algumas funções que eu usaria em qualquer algoritmo desse problema. Somaram-se 5 funções, ***sum_line*** que calcula a soma da linha *n* da matriz, ***sum_col*** que calcula a soma da coluna *n* da matriz, ***sum_diag*** que calcula a soma da diagonal da matriz, ***sum_rev_diag*** que calcula a soma da diagonal inversa da matriz e ***check*** que retorna 1 caso a matriz for um quadrado mágico.

A princípio comecei a procurar por semelhanças entre os quadrados mágicos, chegando a conclusão quase que imediata que o problema se dividia em dois: quadrados mágicos pares e quadrados mágicos ímpares. Procurei fazer o determinante das matrizes moduladas pelos quadrados mágicos, e cheguei a conclusão que quadrados mágicos de ordem “n” possuem o mesmo valor absoluto dos determinantes, mas nada além disso. Comecei então a pesquisar métodos

matemáticos para a formação de quadrados mágicos, e juntei alguns métodos diferentes para a formação dessas matrizes. Para as matrizes 3 e 5 escolhi implementar o método siamês, visto que esse aparenta ser o mais simples para a implementação.

Para o quadrado de ordem 4 foi simples perceber um método para criação. Preenchendo o quadrado de forma linear, de 1 até 16, bastava inverter os 2 números que ficam entre as extremidades dos lados e trocar com a outra dupla da extremidade oposta. Por exemplo, temos o quadrado:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Invertendo os números que ficam entre as extremidades dos lados, temos:

1	3	2	4
9	6	7	12
5	10	11	8
13	15	14	16

E agora trocando as duplas das extremidades opostas temos:

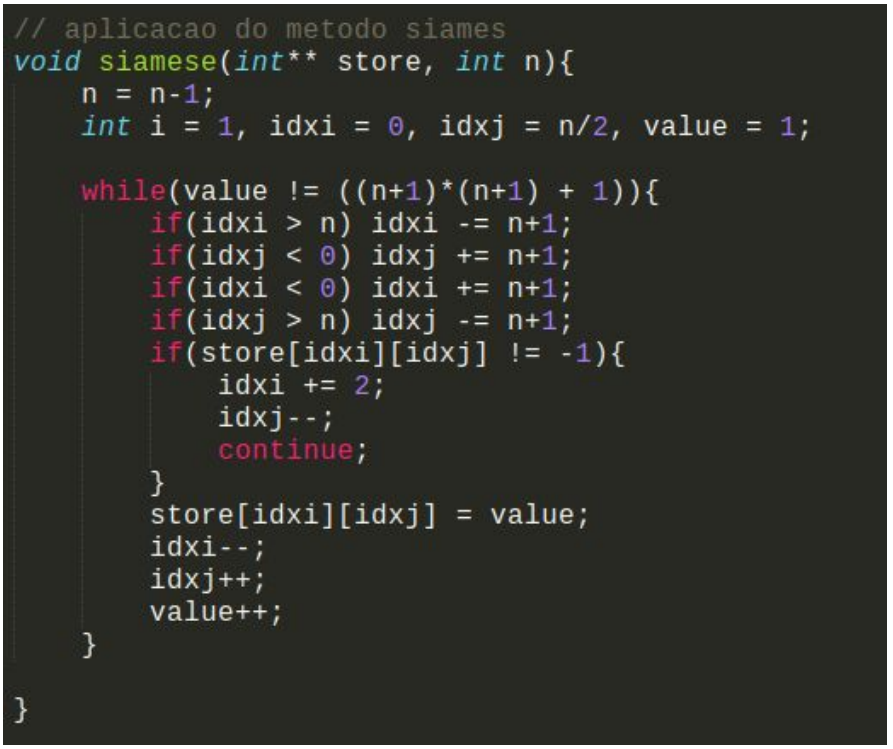
1	15	14	4
12	6	7	9
8	10	11	5
13	3	2	16

Esse método é bem simples de implementar, então vou usá-lo.

Para o quadrado mágico de ordem 6 achei os métodos LUX e Strachey, mas escolhi o método de Strachey, visto que ele utiliza o método siamês e a implementação seria mais simples também.

2.1. Implementação

A implementação do método siamês foi feita através da seguinte função:

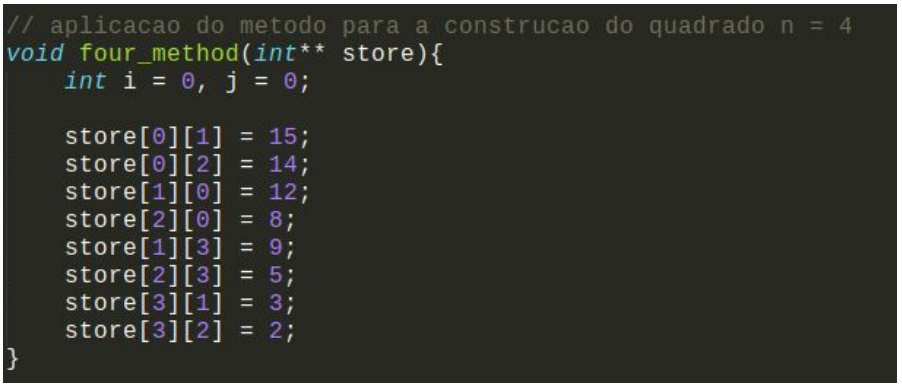


```
// aplicacao do metodo siames
void siamese(int** store, int n){
    n = n-1;
    int i = 1, idxi = 0, idxj = n/2, value = 1;

    while(value != ((n+1)*(n+1) + 1)){
        if(idxi > n) idxi -= n+1;
        if(idxj < 0) idxj += n+1;
        if(idxi < 0) idxi += n+1;
        if(idxj > n) idxj -= n+1;
        if(store[idxi][idxj] != -1){
            idxi += 2;
            idxj--;
            continue;
        }
        store[idxi][idxj] = value;
        idxi--;
        idxj++;
        value++;
    }

}
```

Imagem 4: Função siamese. Possui entrada de uma matriz onde os resultados serão armazenados (store) e a quantidade de linhas/colunas da matriz (n). Com isso, utilizando das variáveis idxi e idxj é possível executar o algoritmo através da utilização de condicionais.

Essa função vai ser utilizada para a resolução dos quadrados de ordem 3 e 5, e uma função bem similar será utilizada para a resolução do quadrado mágico 6x6.

Para a implementação do quadrado 4x4 foi feita a seguinte função:



```
// aplicacao do metodo para a construcao do quadrado n = 4
void four_method(int** store){
    int i = 0, j = 0;

    store[0][1] = 15;
    store[0][2] = 14;
    store[1][0] = 12;
    store[2][0] = 8;
    store[1][3] = 9;
    store[2][3] = 5;
    store[3][1] = 3;
    store[3][2] = 2;
}
```

Imagem 5: Função four_method. Ela apenas muda a posição dos valores de acordo com o raciocínio explicado anteriormente, bem simples. Ela armazena os resultados na matriz store.

Para o quadrado mágico de 6x6 foi usado o método de Strachey, que funciona para quadrados mágicos de ordem 4k +2, onde k é um número natural. Com isso, para a matriz 6x6, k era 1. A resolução por Strachey consiste em dividir o quadrado 6x6 em 4 quadrantes menores A, B, C, D, que ficam dispostos desta forma:

$$\begin{matrix} A & C \\ D & B \end{matrix}$$

Cada quadrante deverá ser um quadrado mágico, onde A armazenará os valores de 1 a 9, B os valores de 10 a 18, C os valores de 19 a 27 e D os valores de 28 a 36. Para a formação desses quadrados, implementei uma função chamada “strachey_siamese”:

```
// aplicacao do metodo siames para quadrantes de strachey
void strachey_siamese(int start, int end, int store[][3], int n){
    n = n-1;
    int i = 1, idxi = 0, idxj = n/2;

    while(start != (end + 1)){
        if(idxi > n) idxi -= n+1;
        if(idxj < 0) idxj += n+1;
        if(idxi < 0) idxi += n+1;
        if(idxj > n) idxj -= n+1;
        if(store[idxi][idxj] != -1){
            idxi += 2;
            idxj--;
            continue;
        }
        store[idxi][idxj] = start;
        idxi--;
        idxj++;
        start++;
    }

}
```

Imagem 6: Função strachey_siamese. Possui como entrada os números de início e fim, uma matriz 3x3 de entrada e o n da matriz (3). Foi feita com base na função siamese apresentada mais cedo,

Seu funcionamento, em essência, funciona igual o algoritmo implementado para a resolução dos quadrados 3 e 5, porém essa função permite a determinação dos números que formarão o quadrado mágico, porém só funciona com matrizes alocadas estaticamente.

Após isso, juntei os quadrantes da maneira que o método instrui, com a seguinte função:

```
void unite(int** store, int A[][3], int C[][3], int D[][3], int B[][3], int n){
    int i = 0, j = 0;

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            if(i < 3 && j < 3) store[i][j] = A[i][j];
            if(i < 3 && j >= 3) store[i][j] = C[i][j-3];
            if(i >= 3 && j < 3) store[i][j] = D[i-3][j];
            if(i >= 3 && j >= 3) store[i][j] = B[i-3][j-3];
        }
    }
}
```

Imagem 7: Função unite. Possui como entrada a matriz onde serão armazenadas as mudanças (store), as 4 matrizes que serão compostas a matriz final e o n do quadrado mágico.

Após a organização do conjunto de dados, o método instrui que deve-se trocar as k colunas mais à esquerda dos quadrantes A e D, portanto fiz isso com a seguinte função:

```
void invertAD(int** store){
    int aux = 0, i = 0;
    for(i = 0; i < 3; i++){
        aux = store[i][0];
        store[i][0] = store[i+3][0];
        store[i+3][0] = aux;
    }
}
```

Imagem 8: Função invertAD. Possui como entrada a função onde serão armazenadas as mudanças, e troca as colunas mais à esquerda dos quadrantes A e D. Só funciona para quadrados mágicos de ordem 6, o que é pertinente para o trabalho.

Em continuidade, o método pede para trocarmos as k-1 colunas mais a direita dos quadrantes C e B, mas como o k da matriz 6x6 é 1, não é necessário a execução desse passo. Em seguida, é necessário a inversão dos valores (1,1) e (1,0) dos quadrantes A e D, que executei com a seguinte função:

```
void invertmiddle(int** store){
    int aux = 0;
    aux = store[1][1];
    store[1][1] = store[4][1];
    store[4][1] = aux;

    aux = store[1][0];
    store[1][0] = store[4][0];
    store[4][0] = aux;
}
```

Imagem 9: Função invertmiddle. Possui como entrada a matriz onde serão executadas as mudanças, e troca a posição (1,1) dos quadrantes A e D e a posição (1,0) dos mesmos quadrantes. Só funciona para quadrados mágicos de ordem 6, o que é pertinente para o trabalho.

Com isso, tem-se um quadrado mágico 6x6. A solução foi modularizada para propósitos de fácil leitura e melhor entendimento e desenvolvimento do código.

2.2 Chamada das funções

A devida formação dos quadrados mágicos foi feita dentro da função main, da seguinte forma. Primeiro é feita a alocação de variáveis, a entrada, o cálculo da soma mágica e a alocação dinâmica da matriz.

```
int main(){
    int n = 0, i = 0, j = 0, sum_mag = 0, count = 1;
    int qA[3][3], qB[3][3], qC[3][3], qD[3][3];

    scanf("%d", &n);

// calculo da soma mágica
    sum_mag = n*(n*n + 1)/2;

// alocacao
    int **matriz = (int**)(malloc(n * sizeof(int*)));
    for(i = 0; i < n; i++){
        matriz[i] = (int*)(malloc(n * sizeof(int)));
    }

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            matriz[i][j] = -1;
        }
    }
```

Imagem 10: Início da função main. Preparações iniciais para a solução do problema. Os includes utilizados foram as bibliotecas “stdio.h” e “stflib.h”, para a devida entrada e saida de dados e a manipulação de memória. A entrada será armazenada na variável “n”, as variáveis “i” e “j” serão as controladoras dos laços durante o código, “sum_mag” armazenará a soma mágica do quadrado mágico, “count” será utilizada para preencher o quadrado de ordem 4 e as variáveis “qA”, “qB”, “qC” e “qD” são as matrizes quadrantes utilizadas na solução do quadrado mágico 6x6.

Em seguida, a solução é arranjada na forma de um *switch*, sendo os casos de teste variando de acordo com a entrada.

```
switch(n){

    case 3:

        siamese(matriz, n);

        break;

    case 4:

        for(i = 0; i < n; i++){
            for(j = 0; j < n; j++){
                matriz[i][j] = count;
                count++;
            }
        }

        four_method(matriz);

        break;

    case 5:

        siamese(matriz, n);

        break;
```

Imagem 11: Inicialização do switch. Casos 3, 4 e 5, onde são montadas as soluções para n = 3, 4 e 5. para a solução onde n = 4, a matriz solução é reinicializada usando a variável count de acordo com o que foi explicado anteriormente.

```
    case 6:

        for(i = 0; i < 3; i++){
            for(j = 0; j < 3; j++){
                qA[i][j] = -1;
                qB[i][j] = -1;
                qC[i][j] = -1;
                qD[i][j] = -1;
            }
        }

        strachey_siamese(1,9,qA,3);
        strachey_siamese(10,18,qB,3);
        strachey_siamese(19,27,qC,3);
        strachey_siamese(28,36,qD,3);

        unite(matriz, qA, qC, qD, qB, n);

        invertAD(matriz);
        invertmiddle(matriz);

        break;

}
```

Imagem 12: Caso 6 do switch. Todos os quadrantes são inicializados com -1, para o funcionamento da função strachey_siamese, chamada em seguida juntamente com as outras funções que montam o quadrado.

Por fim é feita a saída da resposta, juntamente com a liberação da memória da matriz resultado.

```c
    printf("n = %d, Soma = %d\n", n, sum_mag);

    for(i = 0; i < n; i++){
        for(j = 0; j < n; j++){
            printf("%d\t", matriz[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    for(i = 0; i < n; i++){
        free(matriz[i]);
    }
    free(matriz);

    return 0;
}
```

Imagem 13: Saída do programa e desalocação da matriz

2.3. Checagem

Implementei algumas funções somente para a análise do quadrado mágico, visando checar se ele é realmente mágico.

```c
// calcula a soma da linha
int sum_line(int linha, int n, int** matriz){
    int i = 0, sum = 0;
    for(i = 0; i < n; i++){
        sum += matriz[linha][i];
    }
    return(sum);
}

// calcula a soma da coluna
int sum_col(int coluna, int n, int** matriz){
    int i = 0, sum = 0;
    for(i = 0; i < n; i++){
        sum += matriz[i][coluna];
    }
    return(sum);
}

// calcula a soma da diagonal
int sum_diag(int n, int** matriz){
    int i = 0, sum = 0;
    for(i = 0; i < n; i++){
        sum += matriz[i][i];
    }
    return(sum);
}

// calcula a soma da diagonal reversa
int sum_rev_diag(int n, int** matriz){
    int i = 0, sum = 0;
    for(n = n - 1; n >= 0; n--){
        sum += matriz[i][n];
        i++;
    }
    return(sum);
}
```

Imagem 14: Funções implementadas para calcular a soma da linha x, coluna x, diagonal e diagonal inversa da matriz

```c
// checa as condições para o quadrado mágico
int check(int n, int** matriz, int sum_mag){
    int i = 0, linhas = 1, colunas = 1, diagonais = 1;
    for(i = 0; i < n; i++){
        linhas = linhas && (sum_line(i, n, matriz) == sum_mag);
    }
    for(i = 0; i < n; i++){
        colunas = colunas && (sum_col(i, n, matriz) == sum_mag);
    }
    diagonais = diagonais && (sum_diag(n, matriz) == sum_mag) && (sum_rev_diag(n, matriz) == sum_mag);

    return(linhas && colunas && diagonais);
}
```

Imagem 15: Função check. Seu objetivo é checar se todas as somas são iguais, o que implica na matriz analisada ser mágica.

2.4. Considerações Finais

O algoritmo implementado tem complexidade linear (Θ(n)) nos casos em que a entrada n = 3 ou 5, porém sua complexidade sobre para n² para os casos 4 e 6, sem considerar a alocação, inicialização e desalocação operações importantes para a resolução do problema. Portanto, em modo geral, o algoritmo possui complexidade no pior caso de n², sendo representada por O(n²). A complexidade de espaço do algoritmo é Θ(n²), pois o programa aloca dinamicamente uma matriz e as outras variáveis não dependem da entrada.

3. Referências

https://en.wikipedia.org/wiki/Siamese_method ; Método Siamês utilizado na formação dos quadrados 3 e 5.

https://en.wikipedia.org/wiki/Strachey_method_for_magic_squares ; Método de Strachey usado na formação do quadrado 6.

https://en.wikipedia.org/wiki/Conway%27s_LUX_method_for_magic_squares ; Método LUX.