Aluno: Igor Joaquim da Silva Costa
Matricula: 2021032218

1 - Problema soma máxima

Estratégia usada: calcular a soma de cada subconjunto pertencente ao conjunto potência do vetor apresentado.

Especificação: a solução consiste em duas partes principais, a primeira sendo gerar todos os sub-vetores possíveis a partir do vetor de entrada, e a segunda sendo a computação da soma de cada um desses sub-vetores. O resultado deve apresentar, além da soma máxima, os indexes correspondentes ao início e ao fim do sub-vetor com a maior soma.

Projeto e Implementação: para a primeira parte, i.e gerar todos os sub-vetores, a solução escolhida foi usar duas estruturas de repetição alinhadas, sendo a primeira correspondente ao início do sub-vetor e a segunda, o final, garantindo que o index final comece na posição seguinte ao index inicial.

Para a segunda parte, a função "int soma_intervalo" recebe o index inicial e final e retorna o valor desse sub-vetor. A cada novo sub-vetor, a variável "soma_max" é atualizada com o maior valor da soma encontrado até então, além do index inicial e final serem armazenados para a soma correspondente. Após realizar todas as somas possíveis, as variáveis serão impressas.

2 - Quadrado mágico

2.1 Solução com permutação para quadrados pequenos

Estratégia usada: Para um quadrado mágico NxN, são geradas N sequências de N elementos onde a soma de cada sequência equivale a soma máxima, dada pela fórmula[1]:

$$Smax(n) = \frac{n(1+n^2)}{2}$$

Após isso, são geradas permutações nas linhas e nas colunas da matriz, até que ela se torne um quadrado mágico.

Especificação: A solução consiste em 4 passos:

1. Gerar N sequencias de N elementos com soma máxima.
2. Permutar os elementos da matriz, sem mudar a soma das linhas.
3. Decidir se um quadrado é mágico
4. Se o item 3 for falso, retornar ao item 2.

Projeto e Implementação:

1) Para ser possível realizar o item 1, foi implementada uma estrutura "iterador_t", que representa uma variável de problemas típicos de combinatória, como por exemplo, no caso de um quadrado mágico 3x3, com $\mathrm{i} \in [1{,}3]$ e $\mathrm{xi} \in [1{,}9]$ e $\mathrm{x(i)} < \mathrm{x(i{+}1)}$:

$$\mathrm{x1} + \mathrm{x2} + \mathrm{x3} = 15$$

Onde cada xi é um elemento da mesma linha do quadrado.

Para implementar esse funcionamento, dado um número variável de elementos por soma, a estrutura possui os seguintes atributos:

int valor_atual == número que a variável representa no momento.

int valor_referencia == número relativo ao x1.

int * possíveis_valores == lista com todos os valores que a variável pode assumir.

int tamanho == quantidade de valores que a variável pode assumir.

int is_completamente_iterado == booleano que diz se a variável assumiu todos os valores possíveis.

Para lidar com essa estrutura, temos 3 funções auxiliares. A primeira sendo **"void preenche_iterador**", que inicializa o vetor possíveis_valores com os possíveis valores que a variável pode assumir, igualando a 0 os valores já assumidos ou que não respeitam a condição x(i) < x(i+1).

A segunda é a função **"int is_iterador_nulo",** que retorna um booleano True se todas as posições do array possíveis_ valores = 0. A terceira, **"void incrementa_iterador",** incrementa o valor atual e diminui possíveis_valores, até que o valor atual se iguale ao valor máximo.

Por fim, é usada uma função recursiva para ir incrementando o valor de cada variável da sequência, onde a última variável tem o valor incrementado até que ele se iguale ao valor n^2, nesse caso, o valor da penúltima variável é incrementado e a última variável se reinicializa com o valor_atual + 1 em relação à sua antecessora. Isso ocorre até o passo base, que corresponde à variável x1 se igualar a n^2. Esse comportamento é estabelicido pela função **"int incrementa_lista_iteradores".**

Ao final de cada incremento na sequência, é verificado se a soma dos valores da sequência se iguala a soma máxima. Em caso positivo, essa sequência é retornada. Em caso negativo, a função recursiva é chamada até que o caso base seja alcançado. Esse comportamento acontece na função **"int get_combinacao_N"**. No caso da função, x1 == N que é passado por parâmetro.

Para decidir qual N deve ser passado por parâmetro, temos duas escolhas. A primeira é escolhendo sempre o número seguinte ao menor número já usado em alguma outra soma, pela função **"int get_prox_menor_valor",** que retorna o menor elemento da matriz que representa o quadrado mágico + 1. Entretanto, existem casos onde a combinação retornada não é válida para se formar as N linhas de um quadrado mágico, por exemplo:

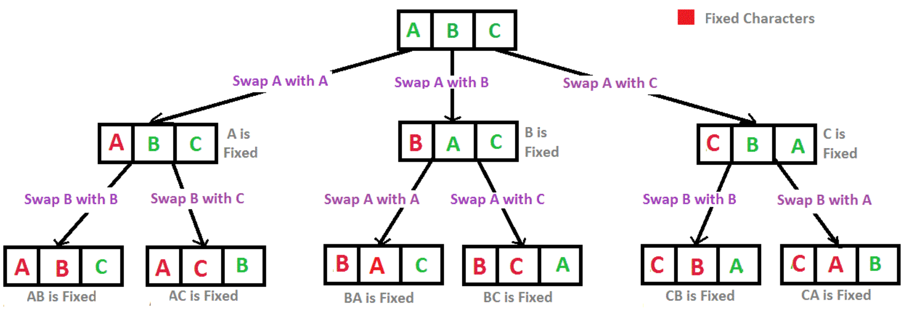
```
1 2 15 16
3 4 13 14
5 6 11 12
0 0 0 0
```

Onde não existem soma condizente com o quadrado mágico.

Para resolver esse impasse, o valor passado com x1 na função get_combinação é uma variável aleatória $\in [1, \frac{n^2}{2}]$. Caso não seja possível formar as N somas, o processo se reinicia até encontrar uma combinação aleatória que forme as N somas. Comportamento gerado na função "void gera_quadrado".

2) Para permutar os elementos da matriz, sem mudar a soma das linhas, foi usado uma ideia de permutação de strings[2], onde, por exemplo, a string "ABC" pode se tornar "CBA". Ou seja, as permutações da matriz são uma série de permutações em suas colunas linhas, codificadas por permutações em string.

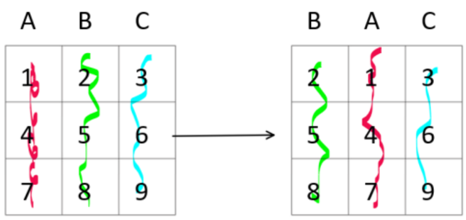
Para isso, é gerada uma lista com todas as permutações possíveis de uma string de tamanho N, representada pela estrutura "lista_permutadores_t", que armazena uma lista de string "char **permutacoes". Para gerar cada uma das permutações, foi usada uma função recursiva[2] que se comporta assim:



Recursion Tree for Permutations of String "ABC"

A cada passo da árvore, a permutação é armazenada na lista. Função **"void gera_lista_permutacoes".**

Por exemplo, para se permutar as colunas de uma matriz 3x3, tendo como entrada a string "BAC", temos:



O comportamento para as permutações das linhas e dos elementos das linhas é semelhante (funcs : permuta_elementos_linha, permuta_linha_matrix, permuta_colunas_matrix).

3) e 4)

Para definir se um quadrado é mágico, basta verificar que suas linhas, colunas e diagonais se igualam a soma máxima(func e_quadrado_magico). O comportamento de juntar o passo 2 e o passo 4 se dá na função **"void permutador",** que permuta a matriz até que ela se torne um quadrado mágico.

2.2 Soluções para quadrados grandes

Estratégia usada: A solução dada no tópico 2.1 só é efetiva em quadrados nxn com n<4, isso porque, como exemplo:

11	12	13	14	15
1	2	16	21	25
3	4	17	19	22
5	6	7	23	24
8	9	10	18	20

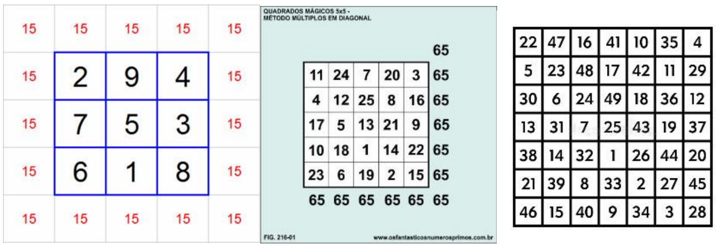
Esse quadrado 5x5 com 5 somas iguais a soma máxima nas linhas, porém ele não pode ser configurado de forma que se torne um quadrado mágico-devido como a forma que as sequências foram geradas - usando apenas números próximos entre si. Por esse motivo, foram tomadas outras estratégias para os quadrados 5x5 e 6x6.

2.2.1 Quadrado Impar – 5x5[3]

Estratégia usada: em 1688, o matemático francês Simon de la Loubère[4] desenvolveu o De la Loubère method[5], um método para construir quadrados mágicos ímpares, a partir de um padrão avistado:

69	34	80	45	1	47	12	58	23
79	44	9	46	11	57	22	68	33
8	54	10	56	21	67	32	78	43
18	55	20	66	31	77	42	7	53
19	65	30	76	41	6	52	17	63
29	75	40	5	51	16	62	27	64
39	4	50	15	61	26	72	28	74
49	14	60	25	71	36	73	38	3
59	24	70	35	81	37	2	48	13

Que ocorre em quadrados mágicos ímpares:



O padrão se dá ao analisarmos as posições da sequência de 1 até n^2, onde o 1 é colocado no quadrado central de uma linha, o 2 é colocado no quadrado abaixo e à direita do 1 (envolvendo as bordas do quadrado como necessário),o 3 é colocado no quadrado diagonalmente abaixo e à direita de 2, e assim por diante. Se a posição já tiver sido ocupada, descem dois quadrados e o padrão continua[6]. A prova para isso pode ser achada nas referências [6] e [7].

Especificação: Executar o seguinte algoritmo[8]:

1. Colocar o número 1 na posição (n/2, n -1)
2. A posição do próximo número é calculada diminuindo o número da linha do número anterior em 1 e aumentando o número da coluna do número anterior em 1. A qualquer momento, se a posição da linha calculada tornar-se -1, ela será arredondada para n-1. Da mesma forma, se a posição da coluna calculada se tornar n, ela mudará para 0.
3. Se o quadrado mágico já contém um número na posição calculada, a posição da coluna calculada será diminuída em 2 e a posição da linha calculada será incrementada em 1.
4. Se a posição da linha calculada for -1 e a posição da coluna calculada for n, a nova posição seria: (0, n-2).

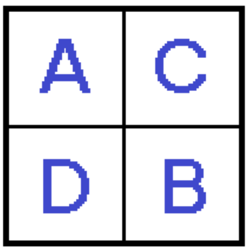
Projeto e Implementação: Implementar o algoritmo acima na função “**void quadrado_magico_impar”**

2.2.1 Quadrado Par – 6x6

Estratégia usada: Durante as décadas, foram se descobrindo formúlas para montar quadrados nxn onde n é par, uma delas remete a versão do método das bordas desenvolvidos pelos japoneses e adaptado pelos europeus[10]. Para a solução desse problema, escolhi o método descrito em [9].

Especificação: Executar o seguinte algoritmo:

1. Separar o quadrado N em 4 quadrados menores, nesse padrão:



2. Cada quadrado corresponderá à um quadrado mágico 3x3, porém, seguindo a ordem alfabética, cada elemento de cada quadrado será somado com o valor do quadrado anterior. Por exemplo, o quadrado A será um quadrado mágico 3x3 comum, já que não possui anterior, o quadrado B será um quadrado mágico 3x3 onde cada elemento é somado com 9 == maior valor de A, e assim por diante.
3. Inverter as posições (0,0), (1,1), (2,0) dos quadrados A e B. Explicação em [11].

Projeto e Implementação: Implementar o algoritmo acima na função "**void quadrado_magico_par_simples"**

[1] Prova matemática da fórmula da soma das linhas de um quadrado mágico

[2]permutações de string em C

[3]Quadrado magico impar

[4]Simon de la Loubère

[5] De la Loubère method

[6]Prova 1, quadrados mágicos impares

[7]Prova 2, quadrados mágicos ímpares

[8]Codigo referencia- quadrado magico impar

[9]Quadrados Mágicos Pares e Pares duplos

[10]Europa do século 15 e quadrados mágicos

[11]Numbers: Their Tales, Types, and Treasures.