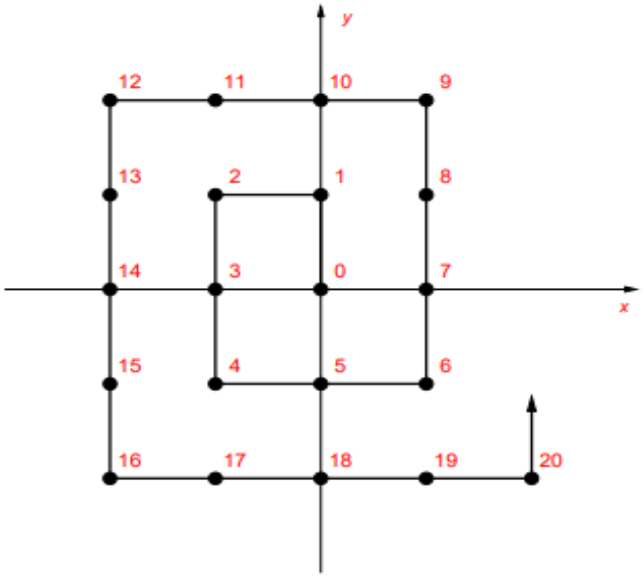
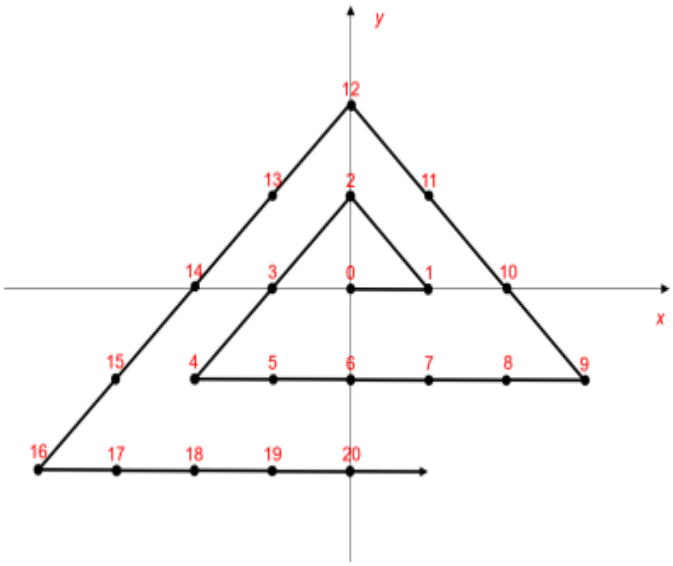
# Trabalho Prático de Matemática Discreta – 2023/2 - TM1

## Aluno: Lucas Affonso Pires, Matrícula: 2023028420

**Parte 1, Especificação:**

O problema a ser solucionado no trabalho prático, envolve dois tipos de espirais, uma quadrada e uma triangular que estão inseridas num plano cartesiano conforme as imagens adiante:



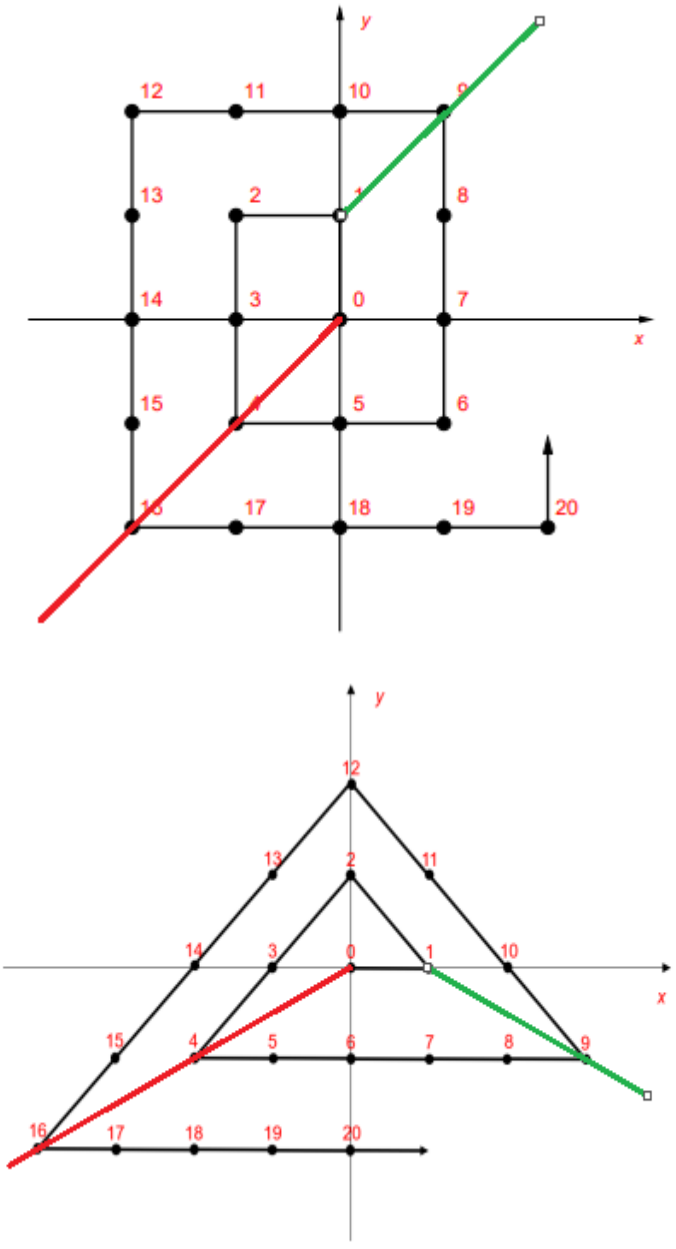




Cada ponto inserido possui uma coordenada inteira no eixo X e no eixo Y, de forma que a cada ponto consecutivo, vai se formando uma espiral. Diante desses casos, o aluno deve implementar um código em C, com custo pelo menos igual ou

inferior à O(n), que seja capaz de receber um número inteiro positivo qualquer e retornar quais são as coordenadas desse número no plano cartesiano de acordo com a espiral em que ele está inserido.

**Parte 2, Projeto:**

**2.1 - Lógica utilizada:** Para resolver o problema, em ambas as espirais, foi utilizado uma lógica parecida, utilizando-se das coordenadas dos números quadrados perfeitos que seguem um padrão. Observe adiante:







Ambos os códigos têm origem com base nas linhas traçadas, onde a vermelha representa a linha que contém os pontos que são quadrados perfeitos pares (0, 4, 16, 36...) e a verde que contém os pontos que são quadrados perfeitos ímpares (1, 9, 25,

49...). A partir daí, existem 4 diferentes possibilidades para qualquer número incluído na espiral:

Caso 1 - ele é um quadrado perfeito par (está na linha vermelha);

Caso 2 - ele é um quadrado perfeito ímpar (está na linha verde);

Caso 3 - ele está após um quadrado perfeito par na espiral e antes de um quadrado perfeito ímpar (na espiral quadrada ele estará na aresta de baixo e da direita, na espiral triangular ele estará na aresta inferior);

Caso 4 - ele está após um quadrado perfeito ímpar na espiral e antes de um quadrado perfeito par (na espiral quadrada ele estará na aresta de cima e da esquerda, na espiral triangular ele estará nas arestas superiores direita e esquerda).

Sabendo dessas 4 possibilidades, basta criar um algoritmo geral que é capaz de calcular o ponto de qualquer uma das possibilidades.

**2.2 - Custo da solução:** Sabendo que a solução implementada utiliza um mesmo algoritmo para cada um dos 4 tipos de possibilidade de ponto na espiral, é perceptível que a solução possui um custo constante Θ (1) independentemente de quão grande seja o número utilizado. O custo definido para definir uma coordenada X ou Y em qualquer um dos casos não supera 6 operações necessárias, necessitando apenas do cálculo da raiz e algumas poucas operações básicas de soma, subtração, divisão e multiplicação. A forma como o código foi implementado para possuir esse custo será exposta adiante.

**2.3 - Implementação do método com custo Θ (1) na espiral quadrada:** Utilizando a lógica apresentada no ponto 2.1, para começar o código, é necessário antes de tudo calcular a raiz do número selecionado. Para isso, cria-se um float raiz que armazena a raiz quadrada do inteiro n recebido.

```
8     scanf("%d", &n);
9
10    raiz = sqrt(n);
```

Após isso, verifica-se o float raiz é um quadrado perfeito (casos 1 e 2) ou não (casos 3 e 4).

```
12    if(raiz == floor(raiz)){
```

Se o número for um quadrado perfeito a float será igual a sua função piso e ele entrará nos casos 1 e 2, caso contrário o número entrará nos casos 3 e 4 por meio de um else.

**Casos 1 e 2:** Nos casos 1 e 2 a lógica é simples, caso o número seja um quadrado perfeito par, seu X e Y é dado pela raiz quadrada negativa de n dividida por 2. Caso o número seja um quadrado perfeito ímpar, seu X é dado pela raiz quadrada de n dividida por 2 e seu Y pela raiz quadrada de n dividida por 2 mais 1.

```
12      if(raiz == floor(raiz)){
13          temp = raiz;
14          if(temp%2 == 0){
15              x = -sqrt(n)/2;
16              y = -sqrt(n)/2;
17          }
18          else{
19              x = sqrt(n)/2;
20              y = (sqrt(n)/2)+1;
21          }
22      }
```

Vale a pena observar que um int temp é igualado a um float raiz, isso foi feito pois apenas um int pode ser divido pelo operador % com o objetivo de verificar se há resto. Se não houver resto, o quadrado perfeito é par, se houver, o quadrado perfeito é ímpar. Outro ponto importante a se observar é o fato de que a raiz de um quadrado perfeito ímpar dividida por 2 resulta em um número não inteiro, porém, como X é um tipo inteiro, ele recebe apenas a parte inteira do número. Por exemplo, para um n que seja 9, sua raiz é igual a 3 e então o X seria igual a 1.5, porém como ele é um inteiro ele passa a valer 1, mantendo a lógica do programa e funcionando corretamente.

**Casos 3 e 4:** Nos casos 3 e 4, são utilizados 2 novos tipos inteiros, lado e passos. Lado refere-se à aresta da espiral quadrada após o quadrado perfeito, visto que a aresta tem o tamanho da raiz quadrada do quadrado perfeito anterior até um número anterior ao próximo quadrado perfeito. Por exemplo, após o número 9 a aresta possui lado 3 até o número anterior ao próximo quadrado perfeito que é 16. Logo, o int lado tem valor igual a raiz. O int passos é responsável por verificar quanto será somado à coordenada da raiz anterior para um número não quadrado perfeito, e verificar se o número já passou de uma aresta para a outra. Portanto, passos é dado subtraindo n do quadrado perfeito anterior.

```
24          lado = raiz;
25          passos = n-(lado*lado);
```

Se o lado for um número par, quer dizer que o número selecionado se encaixa no caso 3. A partir daí, basta verificar em qual das arestas se encontra o número comparando passos a lado. Se passos for igual ou menor que lado, o número se encontra na aresta de baixo, e em qualquer outro caso está na aresta da direita.

```
26          if(lado%2 == 0){
27              if(passos <= lado){
28                  x = ceil((-sqrt(n)/2) + passos);
29                  y = -sqrt(n)/2;
30              }
31              else{
32                  x = sqrt(n)/2;
33                  y = ceil((-sqrt(n)/2)) + (passos-lado);
34              }
35          }
```

Se o lado for um número ímpar, quer dizer que o número selecionado se encaixa no caso 4. A partir daí, basta verificar em qual das arestas se encontra o número comparando passos a lado. Se passos for igual ou menor que lado, o número se encontra na aresta de cima, e em qualquer outro caso está na aresta da esquerda.

```
36          else{
37              if(passos<=lado){
38                  x = floor(sqrt(n)/2 - passos);
39                  y = (sqrt(n)/2)+1;
40              }
41              else{
42                  x = -sqrt(n)/2-1;
43                  y = floor(sqrt(n)/2+1) - (passos-lado);
44              }
45          }
```

Observe que no caso 3, é utilizado a função teto. Isso é apenas para que não haja problema quando a aresta estiver passando pelo ponto 0, que sem a função teto faria com que 2 números estivessem numa mesma coordenada já que sqrt(n) é do tipo double por padrão, causando problemas no código. O mesmo é valido para o caso 4, apenas mudando da função teto para a função piso. Além disso, observe que o X e o Y mudam conforme o número de passos apenas dependendo da aresta em que o número selecionado está. Por exemplo, na aresta inferior o X muda conforme o número de passos, mas o Y mantém-se constante.

**2.4 - Implementação do método com custo Θ (1) na espiral triangular:** Utilizando a lógica apresentada no ponto 2.1, para começar o código, é necessário antes de tudo calcular a raiz do número selecionado. Para isso, cria-se um float raiz que armazena a raiz quadrada do inteiro n recebido.

```
 8      scanf("%d", &n);
 9
10      raiz = sqrt(n);
```

Após isso, verifica-se o float raiz é um quadrado perfeito (casos 1 e 2) ou não (casos 3 e 4).

```
12      if(raiz == floor(raiz)){
```

Se o número for um quadrado perfeito a float será igual a sua função piso e ele entrará nos casos 1 e 2, caso contrário o número entrará nos casos 3 e 4 por meio de um else.

**Casos 1 e 2:** Nos casos 1 e 2 a lógica é simples, caso o número seja um quadrado perfeito par, seu X é dado pela raiz quadrada negativa de n e o Y é dado pela raiz quadrada negativa de n dividida por 2. Caso o número seja um quadrado perfeito ímpar, seu X é dado pela raiz quadrada de n e seu Y pela raiz quadrada negativa de n dividida por 2.

```
12      if(raiz == floor(raiz)){
13          temp = raiz;
14          if(temp%2 == 0){
15              x = -sqrt(n);
16              y = -sqrt(n)/2;
17          }
18          else{
19              x = sqrt(n);
20              y = -sqrt(n)/2;
21          }
22      }
```

Vale a pena observar que um int temp é igualado a um float raiz, isso foi feito pois apenas um int pode ser divido pelo operador % com o objetivo de verificar se há resto. Se não houver resto, o quadrado perfeito é par, se houver, o quadrado perfeito é ímpar. Outro ponto importante a se observar é o fato de que a raiz de um quadrado perfeito ímpar dividida por 2 resulta em um número não inteiro, porém, como X é um tipo inteiro, ele recebe apenas a parte inteira do número. Por exemplo, para um n que seja 9, sua raiz é igual a 3 e então o Y seria igual a -1.5, porém como ele é um inteiro ele passa a valer -1, mantendo a lógica do programa e funcionando corretamente.

**Casos 3 e 4:** Nos casos 3 e 4, são utilizados 2 novos tipos inteiros, lado e passos. Lado refere-se à aresta da espiral triangular após o quadrado perfeito ímpar, visto que a aresta tem o tamanho da raiz do quadrado perfeito ímpar até um número anterior ao quadrado perfeito par. Por exemplo, após o número 9 a aresta possui lado 3 até o número anterior ao próximo quadrado perfeito que é 16. Logo, o int lado tem valor igual a raiz. O int passos é responsável por verificar quanto será somado à coordenada da raiz anterior para um número não quadrado perfeito, e verificar se o número já passou de uma aresta para a outra. Portanto, passos é dado subtraindo n do quadrado perfeito anterior.

Observe que na espiral triangular, o tipo lado não é necessário para descobrir a aresta após o quadrado perfeito par, visto que só a aresta inferior inclui o caso 3.

```
24          lado = raiz;
25          passos = n-(lado*lado);
```

Se o lado for um número par, quer dizer que o número selecionado se encaixa no caso 3. A partir daí, basta verificar o número passos e somá-los à coordenada do quadrado perfeito para descobrir o valor do X.

```
26          if(lado%2 == 0){
27              x = ceil(-sqrt(n)) + passos;
28              y = -sqrt(n)/2;
29          }
```

Se o lado for um número ímpar, quer dizer que o número selecionado se encaixa no caso 4. A partir daí, basta verificar em qual das arestas se encontra o número comparando passos a lado. Se passos for igual ou menor que lado, o número se encontra na aresta superior direita, e em qualquer outro caso está na aresta superior esquerda.

```
30          else{
31              if(passos<=lado){
32                  x = floor(sqrt(n)) - passos;
33                  y = ceil(-sqrt(n)/2) + passos;
34              }
35              else{
36                  x = floor(sqrt(n)) - passos;
37                  y = (ceil(-sqrt(n)/2) + lado) - (passos - lado);
38              }
39          }
```

Observe que no caso 3, é utilizado a função teto. Isso é apenas para que não haja problema quando a aresta estiver passando pelo ponto 0, que sem a função teto faria com que 2 números estivessem numa mesma coordenada já que sqrt(n) é do tipo double por padrão, causando problemas no código. O mesmo é valido para o caso 4, utilizando a função teto e a função piso, para garantir o funcionamento do código. Além disso, observe que o X e o Y mudam apenas conforme o número de passos, sendo utilizados as coordenadas X e Y dos quadrados perfeitos como base. No caso 3, apenas soma-se o número de passos ao X e o Y se mantém constante. No caso 4, o X e o Y utilizam o quadrado perfeito ímpar como base, o X apenas subtrai o número de passos e o Y depende da aresta em que está para mudar.

**Parte 3, Implementação:**

**3.1 - Descrição do código:**

**Espiral quadrada:**

Linhas 1 e 2: Inclusão das bibliotecas necessárias.

Linha 4 e 43: Abertura e fechamento da main.

Linhas 5 e 6: Declaração das variáveis utilizadas.

Linha 8: Comando para receber o número selecionado pelo usuário.

Linha 10: Definição da raiz do número.

Linhas 12 a 22: Casos 1 e 2, explicados acima.

Linhas 23 a 46: Casos 3 e 4, explicados acima.

Linha 48: Comando para printar o resultado.

**Espiral triangular:**

Linhas 1 e 2: Inclusão das bibliotecas necessárias.

Linha 4 e 43: Abertura e fechamento da main.

Linhas 5 e 6: Declaração das variáveis utilizadas.

Linha 8: Comando para receber o número selecionado pelo usuário.

Linha 10: Definição da raiz do número.

Linhas 12 a 22: Casos 1 e 2, explicados acima.

Linhas 23 a 40: Casos 3 e 4, explicados acima.

Linha 42: Comando para printar o resultado.

### 3.2 - Código completo:

**Espiral quadrada:**

```c
#include <stdio.h>
#include <math.h>

int main (){
    int x, y, n, temp, lado, passos;
    float raiz;

    scanf("%d", &n);

    raiz = sqrt(n);

    if(raiz == floor(raiz)){
        temp = raiz;
        if(temp%2 == 0){
            x = -sqrt(n)/2;
            y = -sqrt(n)/2;
        }
        else{
            x = sqrt(n)/2;
            y = (sqrt(n)/2)+1;
        }
    }
    else{
        lado = raiz;
        passos = n-(lado*lado);
        if(lado%2 == 0){
            if(passos <= lado){
                x = ceil((-sqrt(n)/2) + passos);
                y = -sqrt(n)/2;
            }
            else{
                x = sqrt(n)/2;
                y = ceil((-sqrt(n)/2)) + (passos-lado);
            }
        }
        else{
            if(passos<=lado){
                x = floor(sqrt(n)/2 - passos);
                y = (sqrt(n)/2)+1;
            }
            else{
                x = -sqrt(n)/2-1;
                y = floor(sqrt(n)/2+1) - (passos-lado);
            }
        }
    }

    printf("Coordenadas em x e y: (%d, %d)\n", x, y);
}
```

**Espiral triangular:**

```c
#include <stdio.h>
#include <math.h>

int main (){
    int x, y, n, temp, lado, passos;
    float raiz, teste;

    scanf("%d", &n);

    raiz = sqrt(n);

    if(raiz == floor(raiz)){
        temp = raiz;
        if(temp%2 == 0){
            x = -sqrt(n);
            y = -sqrt(n)/2;
        }
        else{
            x = sqrt(n);
            y = -sqrt(n)/2;
        }
    }
    else{
        lado = raiz;
        passos = n - (lado*lado);
        if(lado%2 == 0){
            x = ceil(-sqrt(n)) + passos;
            y = -sqrt(n)/2;
        }
        else{
            if(passos<=lado){
                x = floor(sqrt(n)) - passos;
                y = ceil(-sqrt(n)/2) + passos;
            }
            else{
                x = floor(sqrt(n)) - passos;
                y = (ceil(-sqrt(n)/2) + lado) - (passos - lado);
            }
        }
    }

    printf("Coordenadas em x e y: (%d, %d)\n", x, y);
}
```