Trabalho Prático - Matemática Discreta

Lucas Rafael Costa Santos

Outubro 2023 Belo Horizonte-MG

1 Introdução

Este trabalho visa a implementação de um algoritmo que, a partir de um número dado, retorne as coordenadas de um ponto em uma espiral quadrada e em uma espiral triangular. Além disso, o aluno deverá identificar a complexidade assintótica presente no código, sendo que a complexidade máxima é O(n).

2 Especificação

2.1 Espiral Quadrada

Este algoritmo tem como objetivo calcular as coordenadas (x, y) de uma espiral quadrada. Para isso, o programa requer como entrada um número inteiro, e em caso de ser um valor negativo, o programa exibe uma mensagem de erro e interrompe sua execução. Se o número fornecido for válido, as variáveis x e y são calculadas com base nos vértices da espiral. Então, o ponto escolhido e as coordenadas calculadas são impressos na tela.

2.2 Espiral Triangular

Assim como na espiral quadrada, este algoritmo também tem como objetivo calcular as coordenadas (x, y) de uma espiral, sendo desta vez uma triangular. Desse modo, o programa também requer como entrada um número inteiro, que caso seja negativo, faça com que o programa exiba uma mensagem de erro e interrompa sua execução. Se o número fornecido for válido, as variáveis x e y são calculadas, porém agora, com base nos vértices, meio e topo da espiral triangular. Por fim, o ponto escolhido e as coordenadas calculadas são impressos.

3 Projeto

3.1 Espiral Quadrada

Ao observarmos a espiral quadrada, é possível perceber que seus vértices (pontos de mudança de direção), possuem uma característica bem peculiar, os vértices superior direito e inferior esquerdo são sempre quadrados perfeitos que possuem raízes que crescem de dois em dois. Ou seja, as raízes dos vétices superiores direitos serão números ímpares (1,3,5,7,...), enquanto as raízes dos vértices inferiores esquerdos serão pares (0,2,4,6,...).Sendo assim, é possível definir todos esses pontos. Além disso, por meio da tabela diponibilizada conseguimos encontrar uma relação entre os valores das coordenadas.

Para quadrados perfeitos ímpares temos: (x e y são inteiros)

$$x = \frac{\sqrt{n}}{2} \tag{1}$$

$$y = \frac{\sqrt{n}}{2} + 1 \tag{2}$$

Já para quadrados perfeitos pares, temos:

$$x = -\frac{\sqrt{n}}{2} \tag{3}$$

$$y = -\frac{\sqrt{n}}{2} \tag{4}$$

Também foi possível calcular o valor dos vértices que não são quadrados perfeitos, pois o valor do vértice será igual ao quadrado perfeito anterior a este vértice mais a raíz do mesmo. Dessa maneira, obtemos as seguintes coordenadas:

Vértices superiores esquerdos:

$$x = -\frac{\sqrt{n}}{2} + 1 \tag{5}$$

$$y = \frac{\sqrt{n}}{2} + 1 \tag{6}$$

E para os vértices inferiores diretos:

$$x = \frac{\sqrt{n}}{2} \tag{7}$$

$$y = -\frac{\sqrt{n}}{2} \tag{8}$$

Agora que identificamos todos os vértices, o próximo passo consiste em verificar se o ponto 'n' é um desses vértices. Se 'n' não for um vértice, é necessário determinar a sua posição relativa em relação aos vértices existentes. Depois, calculamos as coordenadas, mantendo 'x' ou 'y' constante, dependendo da situação, e, em seguida, calculamos a diferença entre o vértice anterior e o valor de 'n'.

Quando 'n' está entre um vértice inferior direito e um superior direito:

$$x = \frac{\sqrt{n}}{2} \tag{9}$$

$$y = -\frac{\sqrt{n}}{2} + (n - chao) \tag{10}$$

Se 'n' está entre um vértice Superior direito e um superior esquerdo:

$$x = \frac{\sqrt{n}}{2} - (n - chao) \tag{11}$$

$$y = \frac{\sqrt{n}}{2} + 1 \tag{12}$$

Se 'n' está entre um vértice Superior esquerdo e um inferior esquerdo:

$$x = -\frac{\sqrt{n}}{2} + 1 \tag{13}$$

$$y = \frac{\sqrt{n}}{2} + 1 - (n - chao) \tag{14}$$

Quando 'n' está entre um vértice inferior esquerdo e um inferior direito:

$$x = -\frac{\sqrt{n}}{2} + (n - chao) \tag{15}$$

$$y = -\frac{\sqrt{n}}{2} \tag{16}$$

3.2 Espiral Triangular

Para a espiral triangular, implementamos a mesma lógica, porém, ao invés de termos quatro vértices, teremos dois vértices inferiores, um vétice de topo e um valor que ficará no meio da base do triângulo. Esses valores se ralacionam diretamente com àqueles que já obtivemos com os vértices do quadrado, sendo então, os quadrados perfeitos os vértices inferiores e ou outros vértices o topo ou o meio. Assim, ao obtermos estes ponto e analizarmos as relações na tabela disponibilizada, conseguimos as seguintes coordenadas.

Vértices inferiores esquerdos (quadrados perfeitos pares):

$$x = -\sqrt{n} \tag{17}$$

$$y = -\frac{\sqrt{n}}{2} \tag{18}$$

Vértices inferiores direitos (quadrados perfeitos ímpares):

$$x = \sqrt{n} \tag{19}$$

$$y = -\frac{\sqrt{n}}{2} \tag{20}$$

Topo:

$$x = 0 \tag{21}$$

$$y = \frac{\sqrt{n}}{2} + 1 \tag{22}$$

Meio:

$$x = 0 \tag{23}$$

$$y = -\frac{\sqrt{n}}{2} \tag{24}$$

Agora que identificamos todos os vértices, topo e meio, assim como na espiral quadrada, o próximo passo consiste em verificar se o ponto 'n' é um desses pontos. Se 'n' não for um destes pontos, é necessário determinar a sua posição relativa em relação aos vértices existentes. Depois, calculamos as coordenadas dependendo de sua posição.

Quando 'n' está entre o meio e o vértice inferior direito

$$x = n - chao \tag{25}$$

$$y = -\frac{\sqrt{n}}{2} \tag{26}$$

Se 'n' está entre o vértice inferior direito e o topo:

$$x = \sqrt{n} - (n - chao) \tag{27}$$

$$y = -\frac{\sqrt{n}}{2} + (n - chao) \tag{28}$$

Se 'n' está entre o topo e o vértice inferior esquerdo:

$$x = 0 - (n - chao) \tag{29}$$

$$y = \frac{\sqrt{n}}{2} + 1 - (n - chao) \tag{30}$$

Quando 'n' está entre o vértice inferior esquerdo e o meio:

$$x = -\sqrt{n} + (n - chao) \tag{31}$$

$$y = -\frac{\sqrt{n}}{2} \tag{32}$$

4 Implementação dos Algoritmos

4.1 Espiral Quadrada

4.1.1 Função main

Na função 'main', inicialmente declaramos três variáveis inteiras (n, x, y), e então atribuímos um valor a 'n' com a função scanf. Em seguida, criamos uma nova variável 'resultado' que receberá o valor retornado pela função 'calculaCoordenadas'. Caso o resultado seja igual a -1, o programa deverá exibir uma mensagem de erro e a execução será encerrada. Por outro lado, se o resultado for diferente de -1, a função 'imprime' é chamada, sendo responsável por imprimir os valores de 'n', 'x' e 'y'.

```
int main() {
    int n, x, y;
    scanf("%d", &n);

    int resultado = calculaCoordenadas(n, &x, &y);

    if (resultado == -1) {
        printf("Ponto invalido\n");
        exit(EXIT_FAILURE);
    }

    imprime(n, x, y);

    return 0;
}
```

Listing 1: *Função main*

4.1.2 Função imprime

A Função 'imprime' é responsável por imprimir nosso resultado com o auxílio da função 'printf'.

```
void imprime(int n, int x, int y) {
    printf("Ponto: %d Coordenadas: (%d, %d)", n, x, y);
}
```

Listing 2: *Função imprime*

4.1.3 Função calculaCoordenadas

É na função 'calculaCoordenadas' que se encontra toda a lógica do código. Começamos verificando se o valor 'n' é menor que 0. Se for o caso, retornamos '-1' para indicar um erro, o qual será tratado pela a função principal, 'main'. Em seguida, calculamos a raiz quadrada de 'n' usando a função 'sqrt' e o valor inteiro da metade da raiz.

Agora, entramos nas verificações. Primeiro, verificamos se 'n' é um vértice. Se for um vértice, atribuímos os valores adequados às coordenadas 'x' e 'y'. Caso 'n' não seja um vértice, criamos as variáveis 'chao' e 'teto' para representar os limites inferior e superior de 'n'. Esses limites correspondem aos vértices mais próximos.

A partir do momento em que temos esses limites, podemos calcular as coordenadas de 'n' com base na diferença entre o valor de 'n' e os valores dos vértices mais próximos.

Por fim, a função retorna 0 para indicar que o cálculo das coordenadas foi bem-sucedido.

```c
int calculaCoordenadas(int n, int *x, int *y) {
    if (n < 0) {
        return -1; // Erro: Valor incorreto
    }

    int raiz = sqrt(n); // Calcula a parte inteira da raiz de n
    int metadeRaiz = raiz / 2; //Calcula a metade da raiz

    if (n == raiz * raiz) {
        // Vertice Inferior esquerdo | Superior direito
        *x = (raiz % 2 == 0) ? -metadeRaiz : metadeRaiz;
        *y = (raiz % 2 == 0) ? -metadeRaiz : metadeRaiz + 1;
    }
   else if (n == raiz * raiz + raiz) {
        // Vertice Inferior direito | Superior esquerdo
        *x = (raiz % 2 == 0) ? metadeRaiz : -(metadeRaiz + 1);
        *y = (raiz % 2 == 0) ? -metadeRaiz : metadeRaiz + 1;
    }

    else {
        // Nao e vertice

        // Calcula os limites (Vertices)
        int chao = raiz * raiz + raiz;
        int teto = (n > chao) ? (raiz + 1) * (raiz + 1) : raiz * raiz;
        int temp;
        // Armazena o menor valor em chao e o maior em teto
        if (chao > teto) {
            temp = chao;
            chao = teto;
            teto = temp;
        }
        // Calcula as raizes dos limites
        int raiz1 = sqrt(chao);
        int raiz2 = sqrt(teto);

        // Calcula as coordenada de acordo com os limites

        // Verifica se n esta entre um Vertice Inferior direito e um Superior direito
        if ((chao == raiz1 * raiz1 + raiz1 && raiz1 % 2 == 0) && (teto == raiz2 * raiz2 && raiz2 % 2 != 0)) {
            *x = metadeRaiz;
            *y = -metadeRaiz + (n - chao);
        }
        // Verifica se n esta entre um Vertice Superior direito e um Superior esquerdo
        else if ((chao == raiz1 * raiz1 && raiz1 % 2 != 0) && (teto == raiz2 * raiz2 + raiz2 && raiz2 % 2 != 0)) {
            *x = metadeRaiz - (n - chao);
            *y = metadeRaiz + 1;
        }
        // Verifica se n esta entre um Vertice Superior Esquerdo e um Inferior esquerdo
        else if ((chao == raiz1 * raiz1 + raiz1 && raiz1 % 2 != 0) && (teto == raiz2 * raiz2 && raiz2 % 2 == 0)) {
            *x = -(metadeRaiz + 1);
            *y = metadeRaiz + 1 - (n - chao);
        }
        // Verifica se n esta entre um Vertice Inferior esquerdo e um Inferior direito
        else if ((chao == raiz1 * raiz1 && raiz1 % 2 == 0) && (teto == raiz2 * raiz2 + raiz2 && raiz2 % 2 == 0)) {
            *x = -metadeRaiz + (n - chao);
            *y = -metadeRaiz;
```

```
        }
    }

    return 0;
}
```

Listing 3: *Função calculaCoordenadas*

4.2 Espiral Triangular

4.2.1 Função main

A função 'main' da espiral triangular é exatamente igual à função presente na espiral quadrada.

```
int main() {
    int n, x, y;
    scanf("%d", &n);

    int resultado = calculaCoordenadas(n, &x, &y);

    if (resultado == -1) {
        printf("Ponto invalido\n");
        exit(EXIT_FAILURE);
    }

    imprime(n, x, y);

    return 0;
}
```

Listing 4: *Função main*

4.2.2 Função imprime

A Função 'imprime' também é exatamente igual à função presente na espiral quadrada.

```
void imprime(int n, int x, int y) {
    printf("Ponto: %d Coordenadas: (%d, %d)\n", n, x, y);
}
```

Listing 5: *Função imprime*

4.2.3 Função calculaCoordenadas

Na função 'calculaCoordenadas', observamos algumas pequenas variações em relação à abordagem usada na espiral quadrada. Em vez de verificar especificamente os vértices, nesta função, examinamos o topo, o meio e os vértices inferiores do conjunto numérico. Além disso, os cálculos das coordenadas também seguem uma lógica diferente. No entanto, em termos gerais, a função mantém semelhanças notáveis com a abordagem empregada na espiral quadrada.

```
int calculaCoordenadas(int n, int *x, int *y) {
    if (n < 0) {
        return -1; // Erro: Valor incorreto
    }

    int raiz = sqrt(n); // Calcula a parte inteira da raiz de n
    int metadeRaiz = raiz / 2; //Calcula a metade da raiz

    if (n == raiz * raiz) {
        // Vertice Inferior esquerdo | Inferior direito
        *x = (raiz % 2 == 0) ? -raiz : raiz;
        *y = (raiz % 2 == 0) ? -metadeRaiz : -metadeRaiz;
    }
   else if (n == raiz * raiz + raiz) {
        // Topo | Meio
        *x = 0;
        *y = (raiz % 2 != 0) ? metadeRaiz + 1 : -metadeRaiz;
    }

    else {
```

```
21          // Nao e vertice, meio ou topo
22
23          // Calcula os limites (Vertices, Meio ou Topo)
24          int chao = raiz * raiz + raiz;
25          int teto = (n > chao) ? (raiz + 1) * (raiz + 1) : raiz * raiz;
26          int temp;
27          // Armazena o menor valor em chao e o maior em teto
28          if (chao > teto) {
29              temp = chao;
30              chao = teto;
31              teto = temp;
32          }
33          // Calcula as raizes dos limites
34          int raiz1 = sqrt(chao);
35          int raiz2 = sqrt(teto);
36
37          // Calcula as coordenadas de acordo com os limites
38
39          // Verifica se n esta entre o Meio e o Vertice Inferior direito
40          if ((chao == raiz1 * raiz1 + raiz1 && raiz1 % 2 == 0) && (teto == raiz2 * raiz2 && raiz2 %
    2 != 0)) {
41              *x = n - chao;
42              *y = -metadeRaiz;
43          }
44          // Verifica se n esta entre o Vertice Inferior direito e o Topo
45          else if ((chao == raiz1 * raiz1 && raiz1 % 2 != 0) && (teto == raiz2 * raiz2 + raiz2 &&
    raiz2 % 2 != 0)) {
46              *x = raiz - (n - chao);
47              *y = -metadeRaiz + (n - chao);
48          }
49          // Verifica se n esta entre o Topo e o Vertice Inferior esquerdo
50          else if ((chao == raiz1 * raiz1 + raiz1 && raiz1 % 2 != 0) && (teto == raiz2 * raiz2 &&
    raiz2 % 2 == 0)) {
51              *x = 0 - (n - chao);
52              *y = metadeRaiz + 1 - (n - chao);
53          }
54          // Verifica se n esta entre o Vertice Inferior esquerdo e o Meio
55          else if ((chao == raiz1 * raiz1 && raiz1 % 2 == 0) && (teto == raiz2 * raiz2 + raiz2 &&
    raiz2 % 2 == 0)) {
56              *x = -raiz + (n - chao);
57              *y = -metadeRaiz;
58          }
59      }
60
61      return 0;
62  }
```

Listing 6: *Função calculaCoordenadas*

5 Complexidade Assintótica

Como os dois códigos possuem implementações semelhantes, ambos possuem a mesma complexidade. Sendo assim, podemos analisa-los da seguinte maneira:

- Leitura da entrada: A leitura da entrada envolve apenas a função scanf, que por sua vez, tem uma complexidade constante em relação ao tamanho da entrada, portanto, é O(1).
- Chamada da função calculaCoordenadas:: A complexidade da função calculaCoordenadas depende principalmente de cálculos matemáticos e verificações condicionais. Sendo assim, devemos analisar as partes principais do código dessa função:
  - Verificação de valores fora dos limites: A verificação de 'n' para garantir que não seja negativo é O(1), pois é uma operação simples de comparação.
  - Cálculo da raiz quadrada de n: A complexidade de calcular a raiz quadrada de n é geralmente O(1) em muitas implementações da biblioteca padrão do C. Portanto, aqui iremos considerar este passo como O(1).
  - Determinação da raiz mais próxima: A determinação da raiz mais próxima (usando a variável raiz) é O(1) porque envolve apenas operações simples de comparação.

  - Cálculo das coordenadas: O código inclui várias verificações condicionais e cálculos baseados em condições. No entanto, o número total de operações em cada caso condicional é limitado e não depende diretamente de n. Portanto, cada caso condicional pode ser considerado como O(1).

- Impressão de saída: A função imprime também envolve apenas operações de impressão no console, e a complexidade é, portanto, O(1).

A complexidade total do programa é a soma das complexidades das partes individuais, que são todas O(1). Portanto, podemos concluir que complexidade como um todo é O(1).

6 Conclusão

Após a implementação dos algoritmos, tanto o código 'espquadrada.c' quanto 'esptriangular.c' funcionam perfeitamente e retornam valores lógicos para entradas do tipo 'int' maiores ou iguais a zero. Além disso, ao considerar a função 'sqrt' como uma operação de complexidade constante (O(1)), pudemos concluir que a complexidade dos nossos códigos é, de fato, constante (O(1)).
Por fim, com realização deste projeto foi possível perceber como a análise de complexidade funciona na prática e como implementação de algoritmos com uma menor complexidade pode ser um diferencial para termos um programa mais eficiente.