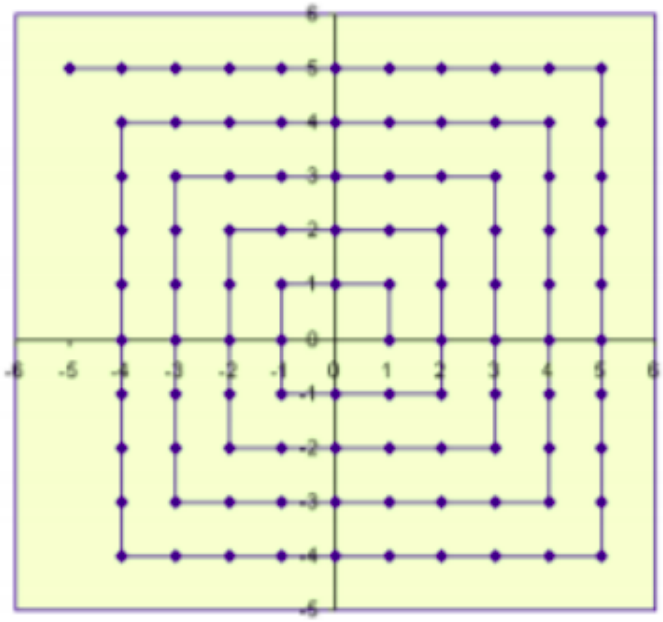
Questão 03

Seja a espiral quadrada como apresentada abaixo. Faça um programa que apresente as coordenadas (x, y)s de um dado ponto n fornecido na entrada. Apresente três algoritmos distintos que executam no pior caso em:

a. $O(1)$

b. $O(n)$



1 O(n)

1.1

Para construir a espiral quadrada, é preciso atentar-se a duas informações: a orientação (se ela está se movimentando pelo eixo X ou pelo eixo Y) e a operação (se ela está crescendo ou decrescendo no eixo). A orientação muda toda vez que o lado supera o último lado do eixo. A operação é definida da seguinte forma: caso o tamanho do lado seja par ela cresce no eixo Y, e caso seja ímpar cresce em X. O tamanho do lado cresce em todo eixo X.

Assim, a primeira implementação baseou-se em quatro variáveis de controle:

- orientation: -1 para o eixo X e 1 para o eixo Y;
- movementSign: -1 para lados com tamanho par e 1 para lados com tamanho ímpar;
- side: começa em 1 e é incrementado toda vez que a orientação passa para o eixo Y;

- sidePosition: variável incremental da posição no lado, começando no 0.

Logo, o algoritmo funciona da seguinte forma:

- As coordenadas X e Y do ponto começam na origem (0,0), a orientação é positiva (deve-se começar operando sobre o eixo X) e o tamanho do lado é 1;
- Após realizar a operação sobre as coordenadas, o sidePosition é incrementado;
- Como o sidePosition alcança o side, ele é zerado e a orientação é trocada (passa-se a operar sobre Y). Como a orientação é do eixo Y, não é preciso incrementar o side;
- A orientação passa a ser positiva, portanto opera-se sobre o eixo Y. A operação continua positiva, portanto a coordenada Y é incrementada;
- sidePosition é incrementado;
- sidePosition alcança o tamanho do lado e é zerado e a orientação deve ser trocada para o eixo X;
- Como voltamos ao eixo X, devemos incrementar o tamanho do Lado e trocar a operação, que como o eixo é par deve ser negativo;
- O ciclo continua até alcançar n.

Dessa maneira, para qualquer ponto n, iremos executar uma quantidade constante de instruções n vezes (calcular as coordenadas desde a origem, ponto 0, até o ponto n). Portanto esse algoritmo é O(n).

1.2

Pensando no comportamento do crescimento da espiral em relação ao plano cartesiano, podemos dividir a espiral em 4 lados: o primeiro se refere às retas horizontais (eixo X) que estão abaixo do eixo X; o segundo se refere às retas verticais (eixo Y) que estão a direita do eixo Y; o terceiro se refere às retas horizontais (eixo X) que estão acima do eixo X; e o quarto se refere às retas verticais (eixo Y) que estão a esquerda do eixo Y. Sempre que estiver nos lados 1 e 2 deve-se incrementar os valores da coordenada X e Y, respectivamente. Da mesma forma, nos lados 3 e 4 decrementa-se os valores da coordenada X e Y, respectivamente. O tamanho do lado cresce em todo eixo X, ou seja, a cada dois segmentos de reta do mesmo tamanho.

Assim, a segunda implementação baseou-se em quatro variáveis de controle:

- vertices: quantidade e vértices;
- index: índice da proxima curva;
- segment: tamanho do segmento atual;
- amountSegment: quantidade de segmentos do mesmo tamanho.

Logo, o algoritmo funciona da seguinte forma:

- As coordenadas X e Y do ponto começam na origem (0,0), a quantidade de vértices é 0, o índice da próxima curva 1, índice da próxima curva 1, tamanho do segmento atual 1 e a quantidade de segmentos do mesmo tamanho 0;

- Calcula-se a direção do segmento (lado da espiral;

- De acordo com o lado incrementa ou decrementa X ou Y;

- Se for alcançado um ponto que é uma curva, a quantidade de segmentos e de vértices é incrementada, e o índice da próxima curva é atualizado;

- A cada dois segmentos do mesmo tamanho, a quantidade de segmentos aumenta e o amountSegment volta a ser 0.

- O ciclo continua até alcançar n.

De maneira similar ao primeiro algoritmo, para qualquer ponto n iremos executar uma quantidade constante de instruções n vezes (calcular as coordenadas desde a origem, ponto 0, até o ponto n). Portanto esse algoritmo é O(n).

2 O(1)

2.1

Podemos encontrar que dado um ponto N, se N é quadrado perfeito então suas coordenadas podem ser descritas da seguinte maneira:

Se N é par:

$$(\frac{\sqrt{N}}{-2}, \frac{\sqrt{N}}{2})$$

Se N é ímpar:

$$(\frac{\sqrt{N}+1}{2}, \frac{\sqrt{N}-1}{-2})$$

Entre um quadrado perfeito ímpar e um par, opera-se positivamente sobre as coordenadas, e entre um quadrado perfeito par e um ímpar, opera-se negativamente sobre as coordenadas. Um quadrado perfeito sempre irá marcar uma curva na espiral, então dado a posição relativa de n sobre o quadrado perfeito, iremos determinar o eixo de movimento (X ou Y).

Assim, a primeira implementação baseou-se em quatro variáveis de controle:

- nearestPS: quadrado perfeito mais próximo;

- distance: diferença do ponto n ao quadrado perfeito mais próximo;

- movementSign: -1 para os nearestPS pares e 1 para os nearestPS ímpares;

- movementAxis: eixo de movimento;

Logo, o algoritmo funciona da seguinte forma:

- Calcula-se o quadrado perfeito mais próximo de n;

- Calcula-se as coordenadas do quadrado perfeito mais próximo;

- Calcula-se a distância de n até o quadrao perfeito mais próximo;

- Se $n > nearestPS$ então o eixo de movimento é Y, caso contrário, X.

- Se o nearesPs é par então o sinal de movimento é negativo, caso contrário, positivo;

- Calcula-se o deslocamento das coordenadas do quadrado perfeito mais próximo até n, onde $coord = coord + (distance * movementSign)$ de acordo com o eixo de movimento determinado;

Dessa maneira, independente do tamanho da entrada n iremos executar uma quantidade constante de instruções. Portanto esse algoritmo é O(1).

2.2

A segunda implementação baseou-se no calculo de posição baseado na quantidade de voltas na espiral, ou seja, a quantidade de quadrados desenhados até o ponto n, e na quantidade de pontos que formam um quadrado ou uma volta.

Assim, a segunda implementação baseou-se em duas variáveis de controle:

- squares: quantidade de voltas na espiral (quantidade e quadrados);

- vertices: quantidades de pontos que formam a espiral, sendo que o primeiro quadrado é formado por 8 pontos;

Logo, o algoritmo funciona da seguinte forma:

- Calcula-se a quantidade de voltas na espiral;

- Calcula-se a quantidade de pontos na espiral, dado $vertices = 8 * sqaures$;

- $quad4 = ((8 + vertices) * squares)/2$;

- $quad1 = quad4 - (vertices * 3)/4$;

- $quad2 = quad4 - (vertices/2)$;

- $quad3 = quad4 - (vertices/4)$;

- Se n é igual a $quad1$ então o ponto se encontra no primeiro quadrante do plano cartesiano, e suas coordenadas são iguais a quantidade de quadrados da espiral;

- Se n é igual a $quad2$ então o ponto se encontra no segundo quadrante do plano cartesiano, e suas coordenadas são $(-squares, squares)$;

- Se n é igual a $quad3$ então o ponto se encontra no terceiro quadrante do plano cartesiano, e suas coordenadas são $(-squares, -squares)$;

- Se n é igual a $quad4$ então o ponto se encontra no quarto quadrante do plano cartesiano, e suas coordenadas são($squares, -squares$);

- Se n é menor do que $quad1$, suas coordenadas são ($squares, squares - quad1 - n$);

- Se n é menor do que $quad2$, suas coordenadas são ($quad2 - n - squares, squares$);

- Se n é menor do que $quad3$, suas coordenadas são ($-squares, quad3 - n - squares$);

- Se n é menor do que $quad4$, suas coordenadas são ($squares - quad4 - n, -squares$);

Dessa maneira, independente do tamanho da entrada n iremos executar uma quantidade constante de instruções. Portanto esse algoritmo é O(1).

2.3

De maneira similar ao primeiro algoritmo, podemos encontrar que dado um ponto N, se N é quadrado perfeito então suas coordenadas podem ser descritas da seguinte maneira:

Se N é par:

$$(\frac{\sqrt{N}}{-2}, \frac{\sqrt{N}}{2})$$

Se N é ímpar:

$$(\frac{\sqrt{N}+1}{2}, \frac{\sqrt{N}-1}{-2})$$

Todos os quadrados perfeitos se encontram na diagonal principal. Caso não seja um quadrado perfeito basta calcular a distância para o quadrado perfeito mais próximo, onde que entre um quadrado perfeito ímpar e um par, opera-se positivamente sobre as coordenadas, e entre um quadrado perfeito par e um ímpar, opera-se negativamente sobre as coordenadas.

Assim, a terceira implementação baseou-se em quatro variáveis de controle:

- nearestPS: quadrado perfeito mais próximo;

- distance: diferença do ponto n ao quadrado perfeito mais próximo;

Logo, o algoritmo funciona da seguinte forma:

- Obtenha o quadrado perfeito mais próximo de n;

- Se n for quadrado perfeito calcule as suas coordenadas;

- Se n não for quadrado perfeito, calcule a distância de n até o quadrado perfeito mais próximo;

- Se o quadrado perfeito mais próximo por par, decrementa o valor da coordenada de Y se a distância for positiva, ou decrementa o valor da coordenada de X se a distância for negativa;

- Se o quadrado perfeito mais próximo por ímpar, incrementa o valor da coordenada de Y se a distância for positiva, ou incrementa o valor da coordenada de X se a distância for negativa;

Dessa maneira, independente do tamanho da entrada n iremos executar uma quantidade constante de instruções. Portanto esse algoritmo é O(1).