PROJETO E ANÁLISE DE ALGORITMOS - LABORATÓRIO 1

Nome: João Victor de Mesquita Cândido dos Santos **RA:** 102028

Explicação do método.

A resolução do problema foi feita através de uma soma dos n primeiros termos de uma progressão aritmética(PA) para que o algoritmo pudesse encontrar a solução exata da posição da cobra no tempo correto (0,5s).

A soma dos n primeiros termos de uma PA é dado pela equação a seguir:

$$S_n = \frac{(a1+a2)*n}{2} \quad (1)$$

No caso do exercício, S_n significa o nosso valor de L e $a1$ e $a2$ representa a primeira e a segunda volta na matriz, assim substituindo em (1) temos:

$$L = \frac{4(N-1)+4(N-2n+1)*n}{2} \quad (2)$$

Fazendo as manipulações temos que:

$$2L = (4N - 4 + 4N - 8n + 4) * n \quad (3)$$

$$2L = 8Nn - 8n^2 \quad (4)$$

$$L = 4Nn - 4n^2 \quad (5)$$

$$L - 4Nn + 4n^2 = 0 \quad (6)$$

Assim foi possível perceber que a equação 6 resultou em uma equação de segundo grau, do qual obteremos raízes para n, utilizando apenas uma das raízes através da resolução da equação teremos:

$$n = \frac{N-\sqrt{N^2-L}}{2} \quad (7)$$

A equação (7) nos diz então quantas voltas completas a cobra fez, e substituindo esse valor de n encontrado em (7) na equação (2) se obtém a quantidade de quadrados que a cobra já andou na matriz.

```
raiz= sqrt((N*N)-L);
n_aux= (N-raiz)/2;
laux= (4*N*n_aux)-(4*n_aux*n_aux);
```

As três linhas de código acima representam os cálculos descritos anteriormente.

A partir do momento que a cobra percorreu uma volta completa na matriz, obteremos uma submatriz da qual essa irá conter o restante que falta para cobra andar na matriz original, ou seja, se temos uma matriz 8x8 e a cobra precisa andar 53 quadrados nessa matriz, ela vai completar duas voltas completas, com duas voltas completas a cobra percorreu um total de 48 quadrados, sobrando 5 quadrados para percorrer, como a cobra já deu duas voltas completas temos então

uma submatriz 4X4 e é através dessa nova matriz com 5 quadrados restando para percorrer iremos analisar qual a posição final da cobra.

```
L=L-laux;
N=N-2*n_aux; //nova dimensão
x=n_aux;
y=n_aux+1;
```

Na primeira linha do código encontramos o quanto falta para a cobrar percorrer, a segunda linha nos diz qual é a dimensão da nova submatriz e x e y significam respectivamente a coluna e a linha de onde está a cobra em relação a matriz principal.

Iniciando a lógica, temos que na parte superior da matriz a cobra pode percorrer um espaço total L de tamanho N, na parte direita a cobra pode correr um espaço L de N-1, na parte de baixo também um espaço N-1 e na parte esquerda N-2.

Assim, a primeira condição é fazer as verificações através da nova matriz e o restante que a cobra necessita percorrer.

Caso o valor que a cobra necessita percorrer (L) for maior que o máximo que ela pode percorrer na parte superior da matriz isso significa que a cobra pode estar na região direita,baixo ou esquerda, para isso precisamos continuar verificando assim subtraímos N do valor que a cobra precisa correr (L) e ir pra próxima verificação, caso esse restante (L=L-N) for maior do que N-1 quer dizer que a cobra ou está na parte de baixo ou na parte esquerda, para isso subtraímos do novo L (L=L-N) o valor que já foi percorrido (N-1), assim (L=(L-N)-(N-1)) e continuamos verificando, caso esse novo valor de L for maior que N-1 novamente isso significa que a nossa cobra para na região esquerda da matriz e subtraímos novamente N-1 que foi o que a cobra percorreu assim (L=(L-N)-(N-1)-(N-1)).

```
if(L>N){

        L=L-N;
        if(L>N-1){
          L=L-(N-1);
          if(L>N-1){
            L=L-(N-1);
```

Como a cobra está na região esquerda, atualizamos os valores de linha e coluna.

Aqui temos duas condições, a primeira é caso o valor final que resta da cobra for igual a N-2, ou seja, quando a cobra chega no final da coluna esquerda.

```
if(L-(N-2)==0){
        y=N;
        x=x+1;
        printf("%lld ",y);
        printf("%lld",x);
       }
```

E a segunda condição é para quando a cobra estiver em qualquer posição da região esquerda.

```
        else{

          y=(N-L)+n_aux;
          x=x+1;
          printf("%lld ",y);
          printf("%lld",x);
    }
```

Caso o final da cobra não esteja na região esquerda, isso significa que ela está na região de baixo da nova matriz, que apresenta a seguinte condição:

```
else{
          y=y+(N-1);
          x=N-(L-x);
          printf("%lld ",y);
          printf("%lld",x);
       }
```

Caso o final da cobra não esteja na região de baixo isso significa que ela está na parte direita da nova matriz, que apresenta a seguinte condição:

```
else{
        y=y+L;
        x=x+N;
        printf("%lld ",y);
        printf("%lld",x);
      }
```

E caso o final da cobra não esteja do lado direito, isso significa que ela para na região de cima da matriz que apresenta a seguinte condição:

```
else{
      printf("%lld ",y);
      x=x+L;
      printf("%lld",x);
      }
```

Análise de Complexidade:

```
int main()
{
  unsigned long long int N,L,n_aux,x=0,y=0,laux;
  double raiz;
  scanf("%lld",&N);
```

```
  scanf("%lld",&L);
  raiz= sqrt((N*N)-L);  // O custo dessa linha é O(4) ou 4O(1)
  n_aux= (N-raiz)/2; // O custo dessa linha é O(3) ou 3O(1)
  laux= (4*N*n_aux)-(4*n_aux*n_aux); // O custo dessa linha é O(6) ou 6O(1)
  L=L-laux; // O custo dessa linha é de O(2) ou 2O(1)
  N=N-2*n_aux; //O custo dessa linha é de O(3) ou 3O(1)
  x=n_aux; //como se trata de uma atribuição temos o custo de O(1)
  y=n_aux+1; // O custo dessa linha é de O(2) ou 2O(1)

  if(L>N){ // O custo dessa linha é de O(1)
     L=L-N; //O custo dessa linha é de O(2) ou 2O(1)
     if(L>N-1){ // O custo dessa linha é de O(2) ou 2O(1)
       L=L-(N-1); //O custo dessa linha é de O(3) ou 3O(1)
       if(L>N-1){ // O custo dessa linha é de O(2) ou 2O(1)
         L=L-(N-1); //O custo dessa linha é de O(3) ou 3O(1)
         if(L-(N-2)==0){ //O custo dessa linha é de O(3) ou 3O(1)
            y=N; //O custo dessa linha é de O(1)
            x=x+1; //O custo dessa linha é de O(2) ou 2O(1)
            printf("%lld ",y);
            printf("%lld",x);
         }
         else{
            y=(N-L)+n_aux; // O custo dessa linha é de O(3) ou 3O(1)
            x=x+1; //O custo dessa linha é de O(2) ou 2O(1)
            printf("%lld ",y);
            printf("%lld",x);
         }
       }
       else{
          y=y+(N-1); //O custo dessa linha é de O(3) ou 3O(1)
          x=N-(L-x); //O custo dessa linha é de O(3) ou 3O(1)
          printf("%lld ",y);
          printf("%lld",x);
       }
     }
     else{
        y=y+L; //O custo dessa linha é de O(2) ou 2O(1)
        x=x+N; //O custo dessa linha é de O(2) ou 2O(1)
        printf("%lld ",y);
        printf("%lld",x);
     }
  }
  else{
     printf("%lld ",y);
     x=x+L; //O custo dessa linha é de O(2) ou 2O(1)
     printf("%lld",x);
     }
return 0;
```

```
}
```

Analisando o código podemos fazer a somatória do custo de todas as linhas assim foi obtido um valor total de O(57) ou melhor dizendo, 57*O(1), assim a complexidade total do código será uma constante vezes O(1), resultando então uma **complexidade constante**.