Trabalho Prático

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil
joaoteixeira@ufmg.br

1. INTRODUÇÃO

Este documento apresenta a implementação de três algoritmos para geração de espirais matemáticas em linguagem C, desenvolvidos como requisito do trabalho prático da disciplina. O objetivo principal foi criar soluções computacionais eficientes para:

1. *Espiral Quadrada (conforme tabela fornecida)*
2. *Espiral Triangular (conforme tabela fornecida)*
3. *Espiral Criativa (padrão original)*

2. MÉTODO DE IMPLEMENTAÇÃO

2.1 Abordagem Geral

Para cada espiral, adotamos estratégias distintas:

Espiral Quadrada:

- *Cálculo matemático direto baseado em camadas concêntricas*
- *Fórmula otimizada que determina coordenadas sem iteração completa*
- *Complexidade: $O(\sqrt{n})$*

Espiral Triangular:

- *Uso de vetores de direção para controle de movimento*
- *Sistema de passos que alterna entre três eixos principais*
- *Complexidade: $O(n)$*

Espiral Criativa:

- *Padrão original de zigue-zague vertical*
- *Crescimento progressivo da amplitude*
- *Complexidade: $O(n)$*

2.2 Ferramentas de Validação

- *Visualização Gráfica: GeoGebra para confirmação dos padrões*
- *Validação Cruzada: Comparação com tabelas de referência*

3. DESENVOLVIMENTO TÉCNICO

```
/espirais
├── espirais.h          # Definições comuns
├── espquadrada.c       # Implementação quadrada (O(√n))
├── esptriangular.c     # Implementação triangular (O(n))
├── minhasesp.c         # Espiral criativa (O(n))
├── main.c              # Programa principal
└── Makefile            #Script de compilação
```

3.2 Implementações-Chave

Espiral Quadrada (trecho principal):

```c
Ponto espiral_quadrada(int n) {
    int x = 0, y = 0;
    int lado_do_quadrado = 0;

    for(; lado_do_quadrado * lado_do_quadrado <= n; ++lado_do_quadrado)
        ; // encontra o lado do quadrado de area n

    --lado_do_quadrado;
```

- Calcula a camada atual através de aproximação por quadrados perfeitos
- Complexidade: O(√n) para determinação da camada

Lógica de Posicionamento:

- Divide o problema em quadrantes baseado na paridade da camada
- Atualiza coordenadas de forma incremental

Otimizações:

- Elimina necessidade de iteração completa por ponto
- Cálculo matemático direto das coordenadas
- Controle preciso de transições entre camadas

```c
if(lado_do_quadrado % 2 == 1) {
    ++y;
    n -= lado_do_quadrado * lado_do_quadrado;
    x += lado_do_quadrado / 2;
    y += lado_do_quadrado / 2;

    if(n <= lado_do_quadrado) {
        x -= n;
    }
    else {
        x -= lado_do_quadrado;
        n -= lado_do_quadrado;
        y -= n;
    }
}
```

3.3 Espiral Triangular (esptriangular.c)

```c
Ponto espiral_triangular(int n) {
    if (n == 0) return (Ponto){0, 0};

    int x = 0, y = 0;
    int lado = 0, passo = 0;
    int dx[] = {1, -1, -1};
    int dy[] = {0, 1, -1};
    int incremento[] = {4, 2, 2};
    int lados[] = {1, 1, 2};

    for (int i = 1; i <= n; i++) {
        if(passo == lados[lado])
        {
            lados[lado] += incremento[lado];
            lado = (lado + 1) % 3;
            passo = 0;
        }

        x += dx[lado];
        y += dy[lado];
        passo++;
    }

    return (Ponto){x, y};
}
```

- Sistema de Direções: 3 vetores (horizontal + 2 diagonais) formando ângulos de 120°
- Controle de Ciclos:
  - `ciclo[]` define o tamanho inicial de cada segmento
  - `incremento[]` expande os segmentos a cada camada (padrão 4-2-2 para triângulos equiláteros)

3.3 Espiral Criativa (minhaesp.c)

```
int dx[] = {1, 0, -1, -1, 1};  // Vetores de direção X
int dy[] = {0, 1, 0, -1, -1};   // Vetores de direção Y
```

- Sequência de 5 movimentos distintos formando o zigue-zague
- Combina deslocamentos laterais e verticais

Controle de Ciclos:

```c
if(passo == lados[lado])
{
    lados[lado] += incremento[lado];
    lado = (lado + 1) % 5;
    passo = 0;
}
```

- Sistema de estados para gerenciar padrões complexos
- Crescimento adaptativo do tamanho dos segmentos

Complexidade:
- O(n) linear, pois cada ponto requer cálculo constante
- 5 estágios distintos de movimento com crescimento controlado

3.3 Elementos Comuns

Estrutura de Dados:

```c
typedef struct {
    int x;
    int y;
} Ponto;
```

- Padronização para representação de coordenadas
- Retorno unificado para todas as funções

Validação:

- Tratamento explícito do caso n=0 em todas implementações
- Verificação de limites em cada operação de incremento
- Precisão matemática garantida por aritmética inteira

4. ANÁLISE DE COMPLEXIDADE

Algoritmo	Melhor Caso	Pior Caso	Justificativa
Espiral Quadrada	$\Theta(1)$	$O(\sqrt{n})$	Cálculo direto por camadas
Espiral Triangular	$\Theta(1)$	$O(n)$	Iteração linear controlada
Espiral Criativa	$\Theta(1)$	$O(n)$	Padrão regular com crescimento

5. CONCLUSÃO

Este trabalho prático permitiu a implementação e análise de três diferentes algoritmos para geração de espirais matemáticas, proporcionando valiosos aprendizados em otimização de algoritmos e programação em C.

Principais Realizações:

- Implementação bem-sucedida da Espiral Quadrada com complexidade O(√n), superando o requisito mínimo de O(n)
- Desenvolvimento da Espiral Triangular com padrão preciso conforme tabela fornecida
- Criação de uma Espiral Criativa original com padrão de zigue-zague vertical
- Documentação completa e código devidamente comentado

Desafios e Soluções:

1. Precisão Matemática:
    - *Problema*: Discrepâncias iniciais nos pontos de transição
    - *Solução*: Revisão detalhada das fórmulas e implementação de testes unitários
2. Otimização de Performance:
    - *Problema*: Implementação inicial com complexidade O(n) para a espiral quadrada
    - *Solução*: Reformulação usando propriedades matemáticas de quadrados perfeitos
3. Validação Cruzada:
    - *Problema*: Dificuldade em verificar padrões complexos visualmente
    - *Solução*: Uso do GeoGebra para plotagem automática dos pontos

Aprendizados Principais:

- A importância da análise matemática preliminar antes da implementação
- O valor dos testes sistemáticos para garantir correção
- Como documentação clara facilita a manutenção e revisão
- A eficácia de visualizações gráficas para debug de algoritmos geométricos

6. REFERÊNCIAS BIBLIOGRÁFICAS

1. ROSE, Kenneth H. *Matemática Discreta e Suas Aplicações*. 6ª edição.
    - Relevância: Livro-texto adotado na disciplina, utilizado como base teórica para os conceitos de sequências matemáticas e análise de algoritmos.
2. CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. 3. ed. Rio de Janeiro: Elsevier, 2012.
    - Aplicação: Fundamentos de análise de complexidade de algoritmos (Cap. 3).
3. KERNIGHAN, B. W.; RITCHIE, D. M. *A Linguagem de Programação C*. 2. ed. Rio de Janeiro: Campus, 1989.
    - Uso: Referência para técnicas de implementação em C.

4. Documentação oficial do *GeoGebra*. Disponível em: https://www.geogebra.org/manual/pt/