UNIVERSIDADE FEDERAL DE MINAS GERAIS
DEPARTAMENTO DA CIÊNCIA DA COMPUTAÇÃO

Trabalho Prático Final Algoritmos e Estruturas de Dados I

Aluno: Rafael Lázaro Monteiro Matrícula: 2017435036

Curso: Engenharia de Sistemas

Sumário

Tabela de Figuras

1. INTRODUÇÃO

O trabalho descrito neste documento teve como objetivo a utilização de conceitos de geometria computacional para avaliação de uma série de elementos geométricos, dentre eles pontos, retas e polígonos, que foram passados pelos professores da disciplina.

Para a implementação do programa foi utilizado o conceito de "Tipo Abstrato de Dados" (TAD) para implementação de uma biblioteca para as funções utilizadas na análise geométrica dos polígonos.

O trabalho é composto de três documentos, sendo o "TADGeometria.h" e "TADGeometria.c" os dois documentos que compõem a nova biblioteca de funções, e "main.c" que é o documento responsável pelo funcionamento do programa e aonde ocorrem as operações relativas ao documento.

2. IMPLEMENTAÇÃO

A implementação do trabalho consiste no desenvolvimento da biblioteca de funções para a análise geométrica dos dados passado, e por um programa para leitura e e extração de dados do arquivo.

Tomando os arquivos para implementação do projeto temos o “TADGeometria.h” que está implementado da maneira mostrada abaixo (observe que a imagem mostra o programa escrito em uma plataforma de programação online disponibilizado pela universidade de Harvard, esta chamada “CS50.ide”, no modo noturno apenas para melhor contraste e exibição da linguagem);

```
1  //ponto, representado por coordenadas cartesianas x e y
2  typedef struct
3  {
4      double x;
5      double y;
6  } ponto;
7  //linha, representada por uma sequencia de 2 a 100 pontos/vertices
8  typedef struct
9  {
10     ponto l[99];
11     int numVertices;
12 } linha;
13 //poligono, representada por uma sequencia de 3 a 100 pontos/vertices
14 typedef struct
15 {
16     ponto p[99];
17     int numVertices;
18 } poligono;
19 void criaPonto(ponto*, double x, double y);
20 //retorna TRUE se os pontos forem identicos
21 int pontoCoincide(ponto P, ponto Q);
22
23 void imprimePonto(ponto P);
24
25 void criaLinha(linha*l, int numVertices, ponto*vertices);
26
27 //verificase a linha poligonal tem interseção com o poligono
28 int linhaInterceptaPoligono(linha l, poligono P);
29
30 //verifica se a linha é simples (sem auto-interseções)
31 int linhaSimples(linha L);
32
33 void criaPoligono (poligono *p, int numVertices, ponto*vertices);
34
35 //verifica se o ponto esta no interior do poligono
36 int pontoEmPoligono(ponto P, poligono Pol);
37
38 //verifica se o poligono é simple (sem auto-interseções)
39 int poligonoSimples (poligono Pol);
40
```

Figura 1:Programa escrito na plataforma CS50.ide.

Como pode ser observado no código são declaradas e definidas as estruturas para o ponto, linha e para polígonos, e as demais funções são apenas declaradas assim como passado no corpo do trabalho.

As funções acima declaradas foram implementadas no arquivo “TADGeometria.c” que é mostrada abaixo. Como mostrado abaixo as funções são:

a) Função “CriaPonto” desenvolvida para pegar os pontos lidos do arquivo de entrada, e os salva-los em array de estruturas do tipo ponto que é alocado na função principal.

```
 8
 9 void criaPonto(ponto *p, double x, double y)//Função para criar ponto
10 {
11     p->x=x;
12     p->y=y;
13 }
14
```

Figura 2: Função CriaPonto.

b) Função “pontoCoincide”, verifica se os pontos passados como parâmetros para a mesma são iguais.

```
15
16 int pontoCoincide(ponto p, ponto q)//função para testar se dois pontos são iguais
17 {
18     if ((p.x==q.x) && (p.y==q.y))
19     {
20         printf("o x e o y do ponto são ( %f , %f ) \n", p.x,q.y);
21         return 1;
22     }
23     else
24     {
25         return 0;
26     }
27 }
```

Figura 3: Função pontoCoincide.

c) Função “imprimePonto”, desenvolvida para impressão dos pontos criados acima.

```
29 void imprimePonto(ponto p)//Função para imprimir pontos
30 {
31     printf("(%lf, %lf)", p.x,p.y);
32 }
```

Figura 4: Função imprimePonto.

d) Função “criaLinha” desenvolvida para pegar os pontos dos vértices da linha lidos do arquivo de entrada, e os salva-los em array de estruturas do tipo linha que é alocado na função principal.

```
33
34 void criaLinha(linha *l,int numVertices,ponto*vertices)//Função para criar linhas
35 {
36     (*l).l[numVertices].x=(*vertices).x;
37     (*l).l[numVertices].y=(*vertices).y;
38 }
```

Figura 5: Função criaLinha.

As funções para calcular as intercepções foram desenvolvidas baseadas nas definições geométricas apresentadas pelo Professor Herondino Fialho da UNIFAP(Universidade Federal do Amapá)[7] . A metodologia é descrita abaixo.

Dados dois segmentos formados pelos pontos p1p2 e p3p4, respectivamente e com:

$$p_1 = (x_1, y_1), p_2 = (x_2, y_2), p_3 = (x_3, y_3) \ e \ p_4 = (x_4, y_4)$$

O ponto de interseção entre eles é dado por:

$$p_1 + u(p_2 - p_1) = p_3 + v(p_4 - p_3)$$

Esta igualdade dá origem a um sistema com duas equações e duas incógnitas (*u* e *v*):

$$\begin{cases} x_{int\,esec} = x_1 + u(x_2 - x_1) \\ y_{int\,esec} = y_1 + u(y_2 - y_1) \end{cases} \qquad \begin{cases} x_{int\,esec} = x_3 + v(x_4 - x_3) \\ y_{int\,esec} = y_3 + v(y_4 - y_3) \end{cases}$$

Desenvolvendo o sistema, temos:

$$u = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}$$

$$v = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}$$

Calculados os parâmetros *u* e *v*, podemos determinar o ponto de interseção. Como resultado dos cálculos se os denominadores forem zero então isso quer dizer que as retas são paralelas logo não se interceptam.

Nas funções abaixo as variáveis “u” e “v” são representadas por “s” e “t” estes são parâmetros que garantem que a interseção não seja impropria, isto é, estejam sobre as retas, os valores de $0 \leq s \leq 1$ e $0 \leq t \leq 1$ garante que os pontos de interseção estejam sobre as retas.

A descrição acima é válida para todas as funções que avaliam se há a intercepção de um ente geométrico com um outro pois as avaliações foram feitas avaliando-se retas e semirretas provindas de pontos dos entes geométricos.

As funções utilizando a metodologia apresentada são mostradas abaixo

e) Função “linhaInterceptaPoligono” função desenvolvida para verificar se uma linha intercepta um polígono.

```
40
41 int linhaInterceptaPoligono(linha L, poligono P)//Função para determinar se uma linha intercepta um poligono
42 {
43     int i,j;
44     double det, s, t;
45     for(i=0; i<L.numVertices-1; i++)//Loop para calcular se há interseção
46     {
47         for(j=0; j<P.numVertices-1; j++)
48         {
49             det=(P.p[j+1].x - P.p[j].x) * (L.l[i+1].y - L.l[i].y) - (P.p[j+1].y - P.p[j].y) * (L.l[i+1].x - L.l[i].x);
50             s=(((P.p[j+1].x - P.p[j].x) * (P.p[j].y - L.l[i].y)) - ((P.p[j+1].y - P.p[j].y) * (P.p[j].x - L.l[i].x))) / det;
51             t=(((L.l[i+1].x - L.l[i].x) * (P.p[j].y - L.l[i].y)) - ((L.l[i+1].y - L.l[i].y) * (P.p[j].x - L.l[i].x))) / det;
52             if((s>=0 && s<=1) && (t>=0 && t<=1))
53                 return 1; // Se ocorrer a interseção retorna 1
54         }
55     }
56     return 0; //Não havendo interseção retorna 0
57 }
```

Figura 6: Função linhaInterceptaPoligono.

f) Função “linhaSimples”, criada para verificar se uma linha intercepta ela mesma, ou seja, é não-simples.

```
int linhaSimples(linha L)
{
    int i,j;
    double det, s, t;
    for(i=0; i<L.numVertices-1; i++)
    {
        //semireta 1
        for(j=0; j<L.numVertices-1; j++)
        {
            //semireta 2
            det=(L.l[j+1].x - L.l[j].x) * (L.l[i+1].y - L.l[i].y) - (L.l[j+1].y - L.l[j].y) * (L.l[i+1].x - L.l[i].x);
            s=(((L.l[j+1].x - L.l[j].x) * (L.l[j].y - L.l[i].y)) - ((L.l[j+1].y - L.l[j].y) * (L.l[j].x - L.l[i].x))) / det;
            t=(((L.l[i+1].x - L.l[i].x) * (L.l[j].y - L.l[i].y)) - ((L.l[i+1].y - L.l[i].y) * (L.l[j].x - L.l[i].x))) / det;
            if((s>0 && s<=1) && (t>0 && t<=1))
                return 1;   //Se ocorrer interseção retorna 1
        }
    }
    return 0;   //Não havendo interseção retorna 0
}
```

Figura 7: Função linhaSimples.

g) Função “criaPoligono” função criada para pegar os dados lidos para os pontos dos vértices do polígono e colocar no array de estruturas do tipo polígono.

```
void criaPoligono(poligono *p, int numVertices, ponto *vertices)//Função para criar poligono
{
    (*p).p[numVertices].x=(*vertices).x;
    (*p).p[numVertices].y=(*vertices).y;
}
```

Figura 8: Função criaPoligono.

h) Função “pontoEmPoligono” função criada para verificar se o ponto se encontra no interior do polígono.

```
int pontoEmPoligono(ponto P, poligono Pol)//Função para verificar se um ponto esta dentro do poligono
{
    int i, count=0;
    double det, s, t;
    ponto AUX;
    AUX.x=INT_MAX;   //Criar um semireta apartir do ponto X usnado o valor maximo do programa
    AUX.y=P.y;  //Mantem o valor do Y
    for(i=0; i<Pol.numVertices-1; i++)
    {
        det=(Pol.p[i+1].x - Pol.p[i].x) * (AUX.y - P.y) - (Pol.p[i+1].y - Pol.p[i].y) * (AUX.x - P.x);
        s=(((Pol.p[i+1].x - Pol.p[i].x) * (Pol.p[i].y - P.y)) - ((Pol.p[i+1].y - Pol.p[i].y) * (Pol.p[i].x - P.x))) / det;
        t=(((AUX.x - P.x) * (Pol.p[i].y - P.y)) - ((AUX.y - P.y) * (Pol.p[i].x - P.x))) / det;
        if((s>=0 && s<=1) && (t>=0 && t<=1))
            count+=1;
    }
    if(count%2!=0)
    {
        return 1; //Se o ponto está dentro do poligono retorna 1
    }

    else
    {

        return 0; //Se o ponto está fora do poligono retorna 0
    }
}
```

Figura 9: Função pontoEmPoligono.

i) Função “poligonoSimples” criada para verificar se o polígono é simples, isto é sem interseções entre as arestas do mesmo.

```
int poligonoSimples(poligono Pol)// Função para verificar se o polígono é simples (sem auto-interseções)
{
    int i,j;
    double det, s, t;
    for(i=0; i<Pol.numVertices-1; i++)
    {
        for(j=0; j<Pol.numVertices-1; j++)
        {
            det=(Pol.p[j+1].x - Pol.p[j].x) * (Pol.p[i+1].y - Pol.p[i].y) - (Pol.p[j+1].y - Pol.p[j].y) * (Pol.p[i+1].x - Pol.p[i].x);
            s=(((Pol.p[j+1].x - Pol.p[j].x) * (Pol.p[j].y - Pol.p[i].y)) - ((Pol.p[j+1].y - Pol.p[j].y) * (Pol.p[j].x - Pol.p[i].x))) / det;
            t=(((Pol.p[i+1].x - Pol.p[i].x) * (Pol.p[j].y - Pol.p[i].y)) - ((Pol.p[i+1].y - Pol.p[i].y) * (Pol.p[j].x - Pol.p[i].x))) / det;
            if((s>0 && s<=1) && (t>0 && t<=1))
                return 1;   //Se há interseção retorna 1
        }
    }
    return 0; //Não Havendo interseção retorna 0
}
```

Figura 10: Função poligonoSimples.

O arquivo “main.c” é a parte integrante do projeto responsável pelas operações no arquivo, isto é a leitura dos dados, e alocação destes nos arrays específicos destinados a cada tipo de dado.

O primeiro Loop é responsável criação e alocação dos pontos a partir dos dados lidos do arquivo de entrada, o segundo Loop cria as linhas e o terceiro é responsável pela criação dos polígonos.

```
26          fscanf(ARQ,"%d", &numPontos);
27          p=calloc(numPontos,sizeof(ponto));
28          for(i=0; i<numPontos; i++)
29          {
30              fscanf(ARQ,"%lf %lf",&x,&y);
31              criaPonto(&p[i],x,y);
32          }
33          fscanf(ARQ,"%d", &numLinhas);
34          l=calloc(numLinhas,sizeof(linha));
35          for(i=0; i<numLinhas; i++)
36          {
37              fscanf(ARQ,"%d",&numverticelinha);
38              l[i].numVertices=numverticelinha;
39              verticeLinhas=calloc(numverticelinha,sizeof(ponto));
40              for(j=0; j<numverticelinha; j++)
41              {
42                  fscanf(ARQ,"%lf %lf",&x,&y);
43                  criaPonto(&verticeLinhas[j],x,y);
44                  criaLinha(&l[i],j,&verticeLinhas[j]);
45              }
46          }
47          fscanf(ARQ,"%d",&numPoligonos);
48          Po=calloc(numPoligonos,sizeof(poligono));
49          for(i=0; i<numPoligonos; i++)
50          {
51              fscanf(ARQ,"%d",&numverticepoligonos);
52              Po[i].numVertices = numverticepoligonos;
53              verticePoligono = calloc(numverticepoligonos,sizeof(ponto));
54              for(j=0; j<numverticepoligonos; j++)
55              {
56                  fscanf(ARQ,"%lf %lf", &x, &y);
57                  criaPonto(&verticePoligono[j],x,y);
58                  criaPoligono(&Po[i], j, &verticePoligono[j]);
59              }
60          }
```

Figura 11: Parte do programa “main.c” responsável pela leitura dos dados e alocação.

Já o trecho do código mostrado abaixo é responsável pela realização dos testes sobre as geometrias especificadas, esse trecho lê as strings de cada teste, e de acordo com essa string são realizados os testes.

```
61          fscanf(ARQ,"%d",&numTeste);
62          opcao=calloc(numTeste,sizeof(char));
63          for(i=0; i<numTeste; i++)
64          {
65              fscanf(ARQ,"%d", &tipoteste);
66              fscanf(ARQ,"%s",opcao);
67              if(strcmp(opcao,"LINSIMP")==0)
68              {
69                  fscanf(ARQ,"%d",&aux1);
70                  if(linhaSimples(l[aux1-1])==0)
71                  {
72                      printf("Linha %d: simples\n", aux1);
73                  }
74                  else
75                  {
76                      printf("Linha %d: nao simples\n", aux1);
77                  }
78              }
79              else
80              {
81                  if(strcmp(opcao,"POLSIMP")==0)
82                  {
83                      fscanf(ARQ,"%d", &aux1);
84                      if(poligonoSimples(Po[aux1-1])==0)
85                      {
86                          printf("Poligono %d: simples\n", aux1);
87                      }
88                      else
89                      {
90                          printf("Poligono %d: nao simples\n", aux1);
91                      }
92                  }
93                  else
94                  {
95                      if(strcmp(opcao,"LINPOL")==0)
96                      {
97                          fscanf(ARQ,"%d %d", &aux1, &aux2);
98                          if(linhaInterceptaPoligono(l[aux1-1], Po[aux2-1])==1)
99                          {
100                             printf("Linha %d: intercepta o poligono %d\n", aux1, aux2);
101                         }
102                         else
103                         {
104                             printf("Linha %d: nao intercepta o poligono %d\n", aux1, aux2);
105                         }
106                     }
107                     else
108                     {
109                         if(strcmp(opcao,"PTOPOL")==0)
110                         {
111                             fscanf(ARQ,"%d %d", &aux1, &aux2);
112                             if(pontoEmPoligono(p[aux1-1], Po[aux2-1])==1)
113                             {
114                                 printf("Ponto %d: dentro do poligono %d\n", aux1, aux2);
115                             }
116                             else
117                             {
118                                 printf("Ponto %d: fora do poligono %d\n", aux1, aux2);
119                             }
120                         }
121                     }
122                 }
123             }
124         }
125     }
```

Figura 12: Trecho responsavel pela realização dos testes sobre as geometrias especificadas.

3. TESTES

Os resultados dos testes realizados pelo programa são mostrados abaixo.

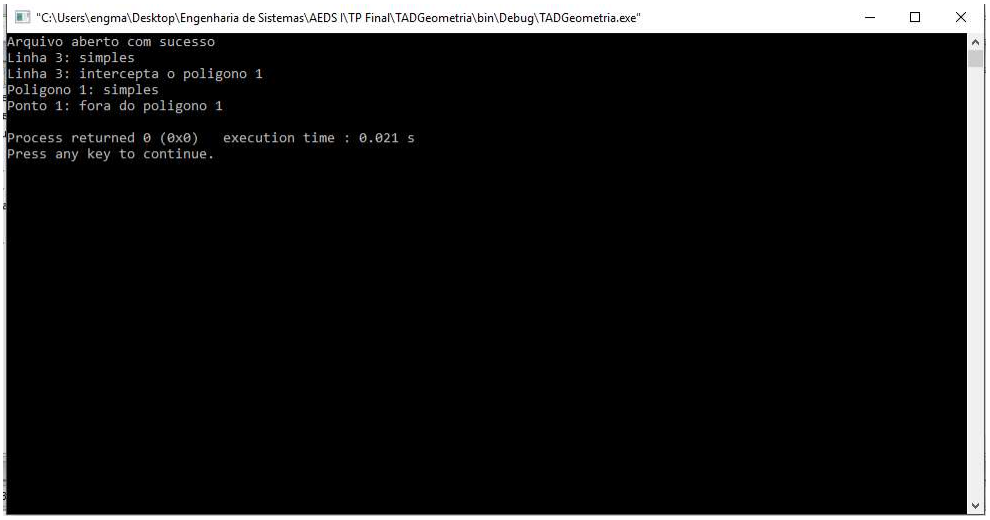

Figura 13:Resultado obtido da simulação do trabalho prático.

Como pode ser visto os resultados obtidos diferem dos resultados que eram esperados segundo o trabalho, isso é devido ao modo que as funções foram implementadas, e também aos próprios dados, como está explicado abaixo:

O polígono 1 tem os seguintes pontos (10, 10), (10, 20), (20, 20) e (20, 10), estes pontos configuram um polígono simples como o mostrado abaixo:

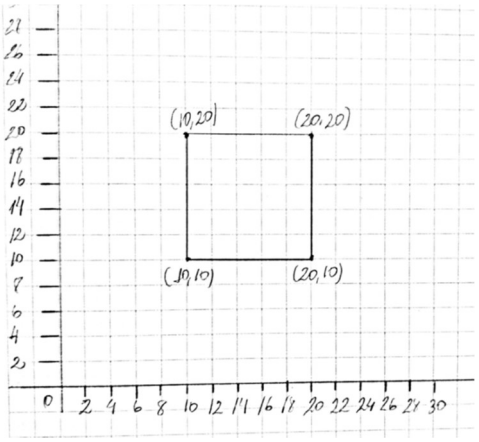

Figura 14: Esquema mostrando que o polígono é um polígono simples.

Já a linha 3 intercepta o polígono pois esta apresenta os pontos (3,6) e (10,12), sendo o segundo ponto coincidente com a aresta da do polígono como mostrado no esquema abaixo.

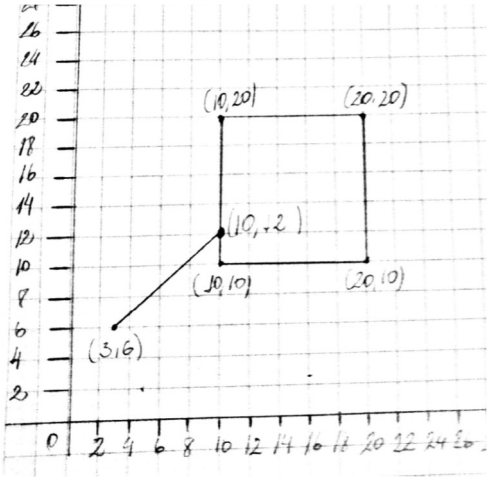

Figura 15: Esquema mostrando que a linha 3 intercepta o poligono1

4. CONCLUSÃO

O objetivo do trabalho foi obtido pois houve a compilação do mesmo e também houve a realização das análises dos polígonos passados para serem analisados e como mostrado na seção anterior alguns resultados diferiram do esperado no documento do trabalho, mas como mostrado isso pode ser explicado com relação a técnica de implementação utilizada assim como possível variação do arquivo de entrada fornecido.

5. BIBLIOGRAFIA

1. Backes, A.; “LINGUAGEM C COMPLETA E DESCOMPLICADA”; Editora Elsevier. Rio de Janeiro 2013.
2. De Berg, M., Chieng, O., van Kreveld, M., Overmars, M.; “ Computational Geometry Algorithms and Applications”, Third Edition, Springler.
3. “http://www.cs.princeton.edu/introalgsds/71primitives”.
4. “http://www.cs.princeton.edu/introalgsds/72hull”.
5. Alsuwaiyel, M. H.;“ALGORITHMS - DESIGN TECHNIQUES AND ANALYSIS” Information & Computer Science Department (KFUPM) July, 1999
6. Fialho, Herondino- “Algoritmos Geometricos”.