Trabalho Prático I - Algoritmos II

Heitor de Paula Santos Damasceno[1] **- Matrícula: 2019006671**

[1]Universidade Federal de Minas Gerais (UFMG)
Instituto de Ciências Exatas - Departamento de Ciência da Computação
Rua Reitor Píres Albuquerque – Pampulha, Belo Horizonte – MG, 31270-901

heitorps123@ufmg.br

1. Explicação Superficial da Resolução Apresentada

O objetivo deste trabalho prático é a implementação de uma solução heurística para o problema da Galeria de Arte. Segue uma breve descrição do problema:

Galeria de Arte

A planta de uma galeria de arte é um polígono simples. Sabendo que uma câmera consegue monitorar qualquer área correspondente a um polígono convexo, qual o número de câmeras necessárias para monitorar completamente essa galeria?

Foi discutida em aula uma solução heurística para esse problema, ou seja, uma solução suficientemente boa, mas não necessariamente a melhor. Essa solução envolve a divisão do polígono que representa a planta da galeria em vários triângulos. Se uma só câmera for posicionada em um vértice de cada triângulo, tentando ao máximo utilizar vértices que fazem parte de vários triângulos, pode-se garantir uma resposta não maior do que $n/3$ câmeras (para um polígono de n vértices).

Mais detalhadamente, os passos da solução são:

1. Utilizar o algoritmo ”Ear Clipping”para triangular o polígono que representa a planta da Galeria de Arte.Esse algoritmo:
   - Checa, para cada sequência de vértices v_i,v_{i+1},v_{i+2}, se há uma mudança de direção para a esquerda caso haja um caminhamento do primeiro para o terceiro (ou seja, é uma dobra para fora e não para dentro).
   - Checa se há algum outro vértice do polígono dentro do triângulo formado pelos três vértices escolhidos.

   Caso a resposta do primeiro item seja sim e a do segundo não, trata-se de uma ”ponta de orelha”, ou seja, uma protuberância (correspondente a um triângulo) que pode ser removida do polígono. Esse procedimento é realizado para todo vértice no início do algoritmo. Após isso, vão se retirando os vértices v_{i+1} dessas orelhas, e atualizando o status dos vértices adjacentes ao retirado (que podem não formar mais orelha com algum vértice, ou pode passar a formar uma orelha com o próximo, ou o anterior). Isso é feito até restarem três vértices no polígono, e os triângulos gerados são salvos em uma lista, completando a triangulação.
2. Constrói um grafo árvore, em que cada vértice é um triângulo proveniente da triangulação, e há uma aresta entre dois vértices se os dois triângulos correspondentes são adjacentes.

3. Colore-se dois vértices de um triângulo aleatório da lista, e realiza-se uma DFS (qualquer caminhamento basta) nesse grafo partindo-se do vértice desse triângulo, e colorindo o vértice restante de cada triângulo adjacente, até o grafo estar todo colorido.

No final disso tudo, o número de vezes que a cor menos representada no grafo aparece é a resposta heurística do problema.

2. Implementação

O trabalho foi implementado na linguagem python, utilizando recursos nativos da linguagem, e as bibliotecas: **Holoviews com o Backend Bokeh** (para a representação visual do trabalho)**, NumPy e FuncTools** (que já estavam incluídas nos casos testes disponibilizados pelo professor).

O código foi devidamente modularizado em funções da forma como o aluno entendeu facilitar a compreensão. Ao final do código são apresentados alguns exemplos interativos dos algoritmos, que representam as etapas destes, sendo que a interação ocorre através de uma janela deslizante.

A seguir são explicados detalhes de implementação do código:

2.1. Estruturas de Dados Utilizadas

Para a implementação dos algoritmos, tornou-se necessário utilizar estruturas de dados nativas de Python, como: **Listas, Dicionários e Conjuntos**.

2.1.1. Listas

Durante o código, Listas foram utilizadas para guardar elementos (**Ex:** variáveis conjunto_pontos, lista_triângulos) e como vetores, para construir o grafo dual do polígono, que é um grafo cujas faces dos triângulos são vértices, e dois vértices estão conectados por uma aresta se as faces são adjacentes (variável Grafo).

2.1.2. Dicionários

Nas ocasiões em que tornou-se necessário associar elementos a um determinado valor (**Ex:** No algoritmo Ear_Clipping, associar ”pontas de orelha”ao vértice da orelha que devia ser retirado, ou no algoritmo da Três_Coloração quando houve a necessidade de associar vértices às cores que receberam) utilizou-se dicionários.

Dicionários são ”mapas”que associam chaves a valores.

2.1.3. Conjuntos

Nas ocasiões em que tornou-se necessário armazenar um conjunto apenas com elementos **distintos**, utilizou-se a estrutura **Conjunto**.

2.2. Algoritmos Utilizados (método de resolução do problema)

Aqui serão detalhados os algoritmos utilizados para resolver o problema da galeria de arte, e como se chegou neles.

2.2.1. produto_vetorial(origem,x,y)

O produto vetorial é uma operação que gera um vetor a partir de outros dois. O sentido desse novo vetor determina se os dois primeiros estão em sentido horário (negativo) ou anti-horário (positivo). Isso é útil para determinar se um caminho que passa por três vértices, por exemplo v_1,v_2 e v_3 tem mudança de direção para a esquerda (para que três vértices formem uma "orelha", a resposta deve ser **sim**).

A definição matemática de produto vetorial dados dois vetores quaisquer $\vec{u}$ = (x_1 , y_1) e $\vec{v}$ = (x_2 , y_2) é:

$$\vec{u}\text{x}\vec{v} = x_1y_2 - x_2y_1$$

A operação é utilizada no algoritmo **Ear Clipping**, ensinado nas aulas.

2.2.2. detectar_vertice_no_triangulo(u,triangulo)

Algoritmo que detecta se um vértice, u por exemplo, está localizado dentro de um triângulo T. Basicamente checa se para todo vértice i de um triângulo, a aresta $i\vec{u}$ está à esquerda de $i(\vec{i+1})$.

Esse algoritmo foi discutido na Lista 4, e é útil no algoritmo **Ear Clipping**.

2.2.3. ear_clipping(conjunto_pontos)

Algoritmo de triangulação de polígonos, O(n^2). Dado um polígono P, ele identifica as pontas de orelhas desse polígono triângulo formado por três vértices consecutivos que não tem nenhum vértice dentro dele,e que, portanto, pode ser removido do polígono. Dessa forma as orelhas são removidas uma a uma, e esses triângulos são armazenados. Após a remoção de todas as orelhas, obtém-se uma divisão do polígono em triângulos (triangulação).

2.2.4. tres_coloracao()

Algoritmo que usa uma dfs (depth-first search) para colorir os vértices do grafo dual ao polígono triangulado. No começo, escolhe-se um vértice aleatório e colore-se os três vértices do triângulo correspondente ao vértice escolhido no grafo dual com cores diferentes. Depois, a partir desse vértice, caminha-se no restante do grafo em profundidade, colorindo os vértices restantes dos vértices correspondentes aos triângulos. Como cada triângulo tem três vértices, e para dois triângulos estarem conectados no grafo construído eles devem ser adjacentes, todo triângulo alcançado após o primeiro passo só terá um vértice a ser colorido, e esse vértice só poderá ser colorido com uma cor para que dois vértices de um mesmo triângulo não tenham cores iguais.

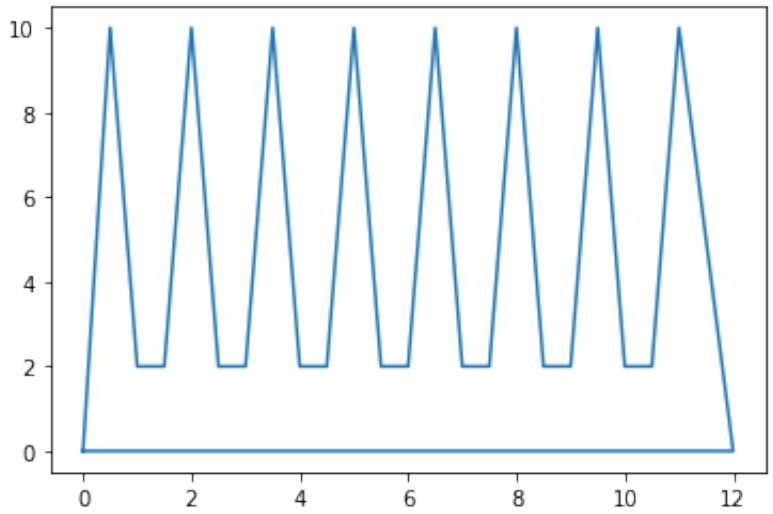
3. Instruções Para Execução

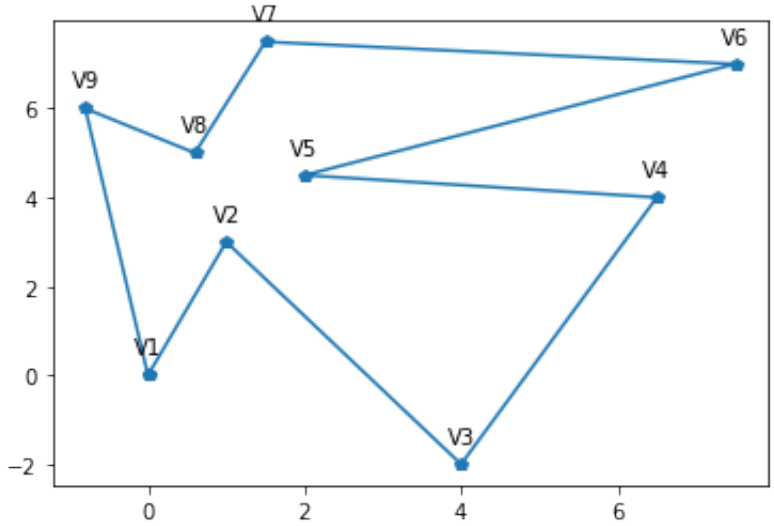
Para resolver o problema da galeria de arte para um polígono, supondo que os pontos de referido polígono já tenham sido adquiridos de alguma forma, é só chamar as funções (em ordem) **ear_clipping**, adquirindo deste a lista de triângulos do polígono,e logo após chamar o algoritmo **tres_coloracao** com esta lista como argumento.

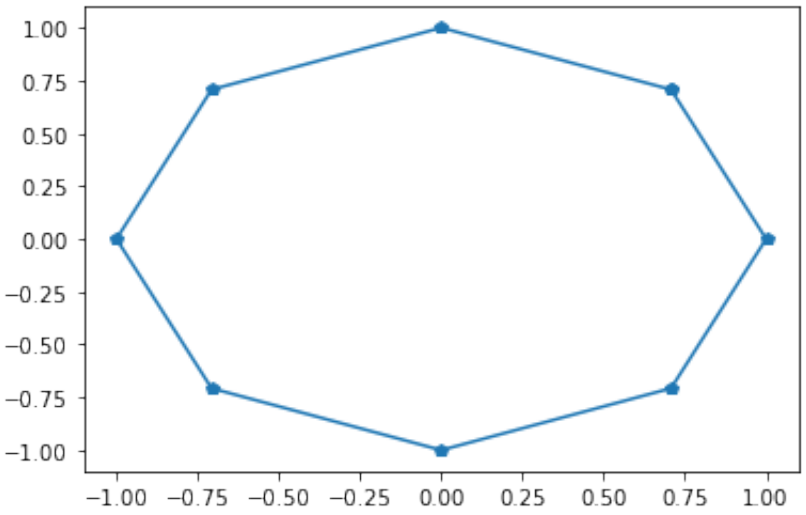
```
hmap,lista_triangulos = ear_clipping(conjunto_pontos)
hmap
... Aqui será imprimida a representação do algoritmo...
hmap = tres_colorir(conjunto_pontos,lista_triangulos)
hmap
... Aqui será imprimida a representação do algoritmo...
```

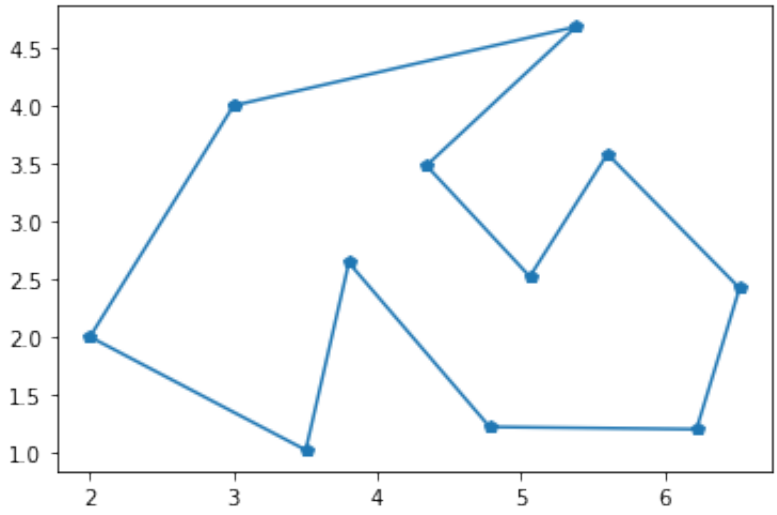
4. Testes Realizados

Os seguintes polígonos foram fornecidos como teste para o trabalho pelo professor.









5. Bibliografia

Notas de aula

Livro ”Computational Geometry - Algorithms and Applications”, 3rd Ed, de Berg et. Al.

Documentação da Biblioteca HoloViews