Universidade Federal de Minas Gerais (UFMG)

Algoritmos 1 - DCC207

Trabalho Prático 1

Geometria Computacional

Problema da Galeria de Arte

Alunos

Carlos Eduardo Dias

Arthur Rodrigues Gorgulho

1. Introdução

Este relatório apresenta a implementação de algoritmos de fechamento convexo e polígono, usando o método de varredura angular e varredura vertical, sobre um determinado conjunto de pontos no plano. Os algoritmos são aplicados ao problema da galeria de arte, onde o objetivo é posicionar o máximo de guardas em um polígono de forma que eles não se vejam entre si.

O relatório também inclui a comparação de tempo gasto pelo PL e o fechamento convexo, e a análise de complexidade dos algoritmos implementados.

Nos exemplos, utilizamos o pacote geometrics da biblioteca de metadados.

Por último, apresentamos uma conclusão sobre os resultados obtidos.

2. Métodos

2.1. Fechamento Convexo

A ideia por trás do método varredura de fechamento convexo é que, ao percorrer todos os pontos do conjunto de pontos, é possível descartar pontos que não estão em um ponto extremo do fechamento convexo. Isso é feito ao ordenar os pontos em ordem angular em relação a um ponto pivô e, em seguida, percorrer os pontos e verificar se o último ponto adicionado ou o penúltimo ponto adicionado formam uma curva à esquerda ou à direita.

Caso formem uma curva à direita, o último ponto adicionado é descartado e o último ponto do fechamento convexo é o ponto anterior a ele. Caso formem uma curva à esquerda, o último ponto adicionado é mantido e o próximo ponto é adicionado ao fechamento convexo.

Para o cálculo do ângulo e da orientação dos pontos, foi utilizado o produto vetorial, que é uma operação que pode ser realizada em tempo constante. Dessa forma, a complexidade do algoritmo é $O(n \log n)$, onde n é o número de pontos no conjunto de pontos, devido à ordenação dos pontos em ordem angular.

Abaixo, segue o trecho de código da função que implementa o fechamento convexo:

```
def convexHull(points):
    pivot = min(points, key=lambda p: (p.y, p.x))
    def polar_angle(p):
        dx = p.x - pivot.x
        dy = p.y - pivot.y
        return (dy, dx)
    def cross_product(o, a, b):
        return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x)
    points = sorted(points, key=polar_angle)
    hull = []
    for p in points:
        while len(hull) >= 2 and cross_product(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    return hull
```

2.2. Fechamento Polígono

O fechamento polígono é um fechamento convexo "melhor", no sentido que não inclui pontos que não estão dentro do polígono. Ele é feito ao ordenar os pontos em ordem angular em relação a um ponto pivô e, em seguida, percorrer os pontos e verificar se o último ponto adicionado ou o penúltimo ponto adicionado formam uma curva à esquerda ou à direita.

Caso formem uma curva à direita, o último ponto adicionado é descartado e o último ponto do fechamento polígono é o ponto anterior a ele.

Caso formem uma curva à esquerda, o último ponto adicionado é mantido e o próximo ponto é adicionado ao fechamento polígono.

Para o cálculo do ângulo e da orientação dos pontos, foi utilizado o produto vetorial, que é uma operação que pode ser realizada em tempo constante. Dessa forma, a complexidade do algoritmo é $O(n \log n)$, onde n é o número de pontos no conjunto de pontos, devido à ordenação dos pontos em ordem angular.

Abaixo, segue o trecho de código da função que implementa o fechamento polígono:

```
def polarSort(points):
    pivot = min(points, key=lambda p: (p.y, p.x))
    def polar_angle(p):
        dx = p.x - pivot.x
        dy = p.y - pivot.y
        return (dy, dx)
    def cross_product(o, a, b):
        return (a.x - o.x) * (b.y - o.y) - (a.y - o.y) * (b.x - o.x)
    points = sorted(points, key=polar_angle)
    hull = []
    for p in points:
        while len(hull) >= 2 and cross_product(hull[-2], hull[-1], p) <= 0:
            hull.pop()
        hull.append(p)
    return hull
```

2.3. Exemplos

convexHull.txt

```
convexHull.txt
10
0 0
1 0
2 0
3 0
4 0
4 1
4 2
4 3
4 4
3 4
2 4
1 4
0 4
0 3
0 2
0 1
```

```
convexHull.txt
10
0 0
1 0
2 0
3 0
4 0
4 1
4 2
4 3
4 4
3 4
2 4
1 4
0 4
0 3
0 2
0 1
```

polarSort.txt

```
polarSort.txt
10
0 0
1 0
2 0
3 0
4 0
4 1
4 2
4 3
4 4
3 4
2 4
1 4
0 4
0 3
0 2
0 1
```

```
polarSort.txt
10
0 0
1 0
2 0
3 0
4 0
4 1
4 2
4 3
4 4
3 4
2 4
1 4
0 4
0 3
0 2
0 1
```

2.4. Plot de Polígono de Exemplo

convexHullpolarSortexemplo.txt

```
convexHullpolarSortexemplo.txt
10
0 0
1 0
2 0
3 0
4 0
4 1
4 2
4 3
4 4
3 4
2 4
1 4
0 4
0 3
0 2
0 1
convexHull: [Point(x=0, y=0), Point(x=4, y=0), Point(x=4, y=4), Point(x=0, y=4)]
polarSort: [Point(x=0, y=0), Point(x=1, y=0), Point(x=2, y=0), Point(x=3, y=0), Point(x=4, y=0), Point(x=4, y=1), Point(x=4, y=2), Point(x=4, y=3), Point(x=4, y=4), Point(x=3, y=4), Point(x=2, y=4), Point(x=1, y=4), Point(x=0, y=4), Point(x=0, y=3), Point(x=0, y=2), Point(x=0, y=1)]
convexHull: 0.000000 segundos
polarSort: 0.000000 segundos
```