Trabalho Prático 3
Algoritmos 1

João Antonio Oliveira Pedrosa
Matrícula: 2019006752

[1] Universidade Federal de Minas Gerais
Belo Horizonte - MG - Brasil

`joao.pedrosa@dcc.ufmg.br`

1. Introdução

O trabalho propõe a resolução do seguinte problema:

Descobrir uma configuração de câmeras que cubra toda a área de uma galeria de arte representada por um polígono, utilizando triangulação de polígonos.

2. Implementação

Para a resolução do problema, duas classes fundamentais foram criadas, Point e Polygon:

3. Point

Essa classe possui apenas dois atributos, x e y, representando as coordenadas x e y do ponto. Os operadores de soma, subtração e multiplicação foram sobrecarregados para responder, respectivamente, à soma coordenada por coordenada, subtração coordenada por coordenada e produto vetorial. Duas funções que recebem a classe Point como parâmetro também foram implementadas. São elas:

- $counter_clock(p1, p2, p3)$: $Retornaseospontosp_1$, p_2 *e* p_3, *seguindo essa ordem, estão em sentido anti-horário.*

Como a resposta pode ser adquirida através de uma simples função recursiva com memorização, não julguei necessário fazer um sistema de classes para essa tarefa, a main do programa realiza todo o processamento necessário para a resolução do problema. Uma descrição em pseudo-código de tudo o que a função Main faz pode ser vista no **Algoritmo 2**.

Algorithm 1 Solução Programação Dinamica

INPUT: Vetor de Preços dos Bilhetes P, Vetor de Tempo das escalas T, Vetor de Descontos D
OUTPUT: Valor mínimo para realização de todas as escalas

```
d ← Nº de Descontos
n ← Nº de Escalas
t ← Tempo máximo para realização dos descontos
M = Matriz de Tamanho n × d, inicializado com todas as posições = −1
procedure DP(int posAtual, int descAtual, int tempoUtilizado)
    if posAtual = n then
        return 0
    end if
    if M[posAtual][descAtual] ≠ −1 then
        return M[posAtual][descAtual]
    end if
    a = DP(posAtual + 1, 0, 0)                        ▷ a = Resultado se esperarmos
    b = ∞            ▷ Inicialmente b = ∞, iremos checar se somos obrigados a esperar
    if descAtual + 1 < n and tempoUtilizado + T[posAtual] < t then
        b = DP(posAtual + 1, descAtual + 1, tempoUtilizado + T[posAtual])
    end if
    M[posAtual][descAtual] = min (a, b) + P[posAtual] ∗ (1 − (D[posAtual]/100))
return M[posAtual][descAtual]
end procedure
return DP(0, 0, 0)
```

4. Análise de Complexidade

- A leitura da entrada têm complexidade $\mathcal{O}(N + T + D)$.

- A função recursiva calcula cada estado da tabela M uma única vez. Portanto tem complexidade $O(N \cdot D)$.

Sendo assim, separando o algoritmo nessas 2 partes, temos uma complexidade de tempo de $\mathcal{O}((N + T + D) + N \cdot D)) \in \mathcal{O}(N \cdot D)$.

Em relação a complexidade de expaço, armazenamos dois vetores de tamanho N, um de tamanho D e uma matriz de tamanho $N \cdot D$, portanto temos complexidade de espaço também $\mathcal{O}(N \cdot D)$.