Memoria de la práctica 3
Algorítmica

FCO. JAVIER SÁEZ MALDONADO
LAURA GÓMEZ GARRIDO
LUIS ANTONIO ORTEGA ANDRÉS
PEDRO BONILLA NADAL
DANIEL POZO ESCALONA

Universidad de Granada
25 de mayo de 2017

Índice

1. Introducción

Un polígono P en el plano euclídeo es un conjunto de n puntos $(x_i, y_i) \in \mathbb{R}^2,\ i = 1, \ldots, n$ llamados vértices, y n segmentos de rectas $\alpha_1, \alpha_2, \ldots, \alpha_n$ llamados lados, que verifican que:

- Los puntos extremos de los lados son dos vértices distintos del polígono.
- Cualquier vértice es el extremo de exactamente dos lados (distintos).

Un polígono del plano se dice **convexo** si cada uno de sus vértices es un vértice de su envolvente convexa.

Por **triangulación** de un polígono plano entenderemos la división de este en un conjunto de triángulos, con la restricción de que dos lados de dos triángulos distintos no se corten o lo hagan en un vértice del polígono.

Diremos que una triangulación de un polígono convexo es **mínima** si la suma de los perímetros de los triángulos que la componen es mínima respecto de cualquier otra triangulación.

2. Problema

Se pide diseñar e implementar un algoritmo *greedy* que solucione el problema de encontrar la triangulación mínima de un polígono convexo. Como salida, se deberá proporcionar el conjunto de aristas que forman la triangulación junto con la suma de los perímetros de los triángulos.

3. Solución

3.1. Descripción del algoritmo

La entrada del algoritmo es un conjunto de n puntos del plano euclídeo, los vértices del polígono convexo que queremos triangular, ordenados. De esta forma, los lados del polígono quedan determinados por el orden, es decir, son:

$$\alpha_i \vee \alpha_{i+1} \forall i \in \mathbb{Z}_n$$

Del mismo modo, las cuerdas son:

$$\alpha_i \vee \alpha_{i+2} \forall i \in \mathbb{Z}_n$$

Los pasos del algoritmo:

1. Si el polígono es un triángulo, no se hace nada.
2. Encontrar la cuerda de longitud mínima.
3. Almacenar la cuerda en el conjunto solución.
4. Eliminar del conjunto de vértices el comprendido entre los dos vértices que une la cuerda.
5. Ejecutar el algoritmo desde el primer paso, siendo esta vez la entrada el conjunto de vértices resultante en el paso anterior.

3.2. Implementación

Para la implementación, hemos escogido el lenguaje de programación `C++`.

3.2.1. Estructuras de datos

Para almacenar los vértices del polígono, hemos escogido el contenedor `list` de la *STL* de `C++`. Además, hemos definido una estructura para almacenar los puntos, con un par de coordenadas en punto flotante de doble precisión y un identificador numérico para la salida por pantalla.

3.2.2. Avance circular de un iterador

```
auto circular_advance = [](auto& forwdIt, auto initValue, auto
    endValue, int n) {
   for(int i=0; i<n; i++)
   {
      forwdIt++;
      if(forwdIt == endValue)
      forwdIt = initValue;
   }

   return forwdIt;
};
```

Esta es una función auxiliar, cuyo propósito es implementar la aritmética de un anillo de restos sobre un contenedor con un iterador que soporte el operador de incremento (++).

La eficiencia teórica de esta función es $O(n)$, sin embargo, siempre se llama con un argumento constante.

3.2.3. Cuerda de longitud mínima

```
auto find_min_string = [&min_distance, &it_min, &min_string,
    &points](auto initial_point_it) {
   auto p0 = initial_point_it;
   auto p = p0;

   do {
      auto prev = p;

      circular_advance(p, points.begin(), points.end(), 2);
      if(min_distance > euclidean_distance(*prev, *p))
      {
         min_distance = euclidean_distance(*prev, *p);
         min_string = std::make_pair(*prev, *p);
         it_min = circular_advance(prev, points.begin(),
             points.end(), 1);
      }

   }while(p != p0);
};
```

Esta función acumula la longitud de la cuerda mínima, empezando en un vértice y recorriendo las cuerdas hasta volver al primero. Si el número de vértices del polígono es impar, solo tiene un ciclo de cuerdas, en cambio tiene dos si es par ($\mathbb{Z}_n/2\mathbb{Z}_n \simeq \mathbb{F}_1$ si n es impar, $\simeq \mathbb{F}_2$ si n es par). Por tanto, en un polígono cuyo número de vértices sea par, ha de llamarse dos veces.

La eficiencia teórica de esta función es $O(n)$.

3.2.4. Bucle principal

```
   while(points.size() > 3)
   {
   auto it = points.begin();
```

```
// Unconditionally find minimum length string starting
// at the first element
find_min_string(it);

// If the polygon's number of edges is even, we need to
// do the same starting at the second one
if(!(points.size()%2)) find_min_string(++it);

sol.push_back(min_string);
points.erase(it_min);
}
```

Este bucle es una transcripción a código de la descripción del algoritmo.

3.2.5. La complejidad del algoritmo

Es un algoritmo de complejidad $O(n^2)$, dado que para un polígono de n nodos, en cada iteración del bucle `while` (n iteraciones) llama a la función *find_min_string*, de complejidad $O(n)$, para después eliminar un vértice.

$$T(n) = \sum_{i=3}^{n} ki = k\frac{n^2+n}{2} \in O(n^2)$$

donde:

- $T(n)$ es el número de instrucciones que ejecuta el algoritmo para un polígono de n vértices.
- k es una constante de proporcionalidad aproximada entre el número de instrucciones que se ejecutan para encontrar la cuerda mínima entre i e i.