Práctica 3

Algoritmos *greedy*

Laura Gómez Garrido, Pedro Bonilla Nadal, Javier Sáez Maldonado, Daniel Pozo Escalona, Luis Ortega Andrés

2º Doble Grado Informática y Matemáticas

Índice

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 su coste (suma de las longitudes de las cuerdas generadas).

Conceptos

Polígono convexo

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

Triangulación

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.

Triangulación mínima

Una triangulación se dice **mínima** si es un elemento minimal del conjunto de triangulaciones posibles, con el preorden inducido por el orden en $\mathbb{R}$ de la suma de las longitudes de sus diagonales.

Conceptos

Polígono convexo

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

Triangulación

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.

Triangulación mínima

Una triangulación se dice **mínima** si es un elemento minimal del conjunto de triangulaciones posibles, con el preorden inducido por el orden en $\mathbb{R}$ de la suma de las longitudes de sus diagonales.

Conceptos

Polígono convexo

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

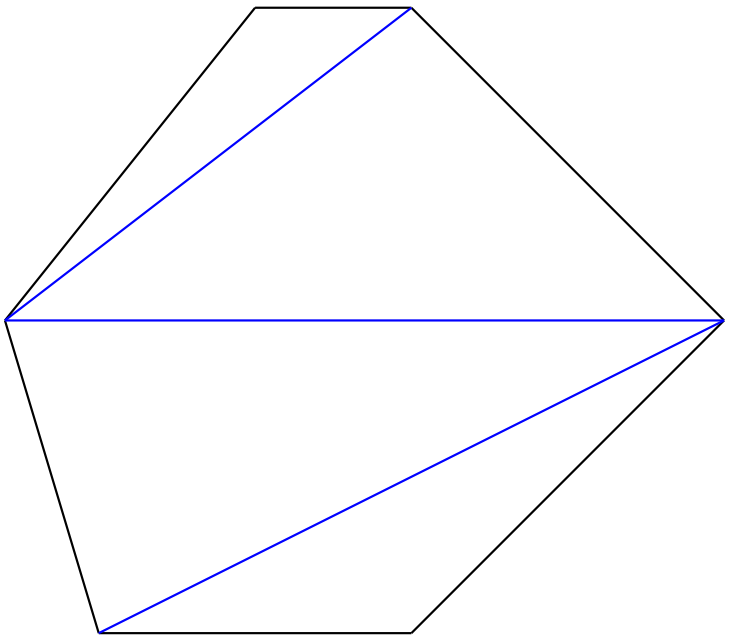
Triangulación

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.

Triangulación mínima

Una triangulación se dice **mínima** si es un elemento minimal del conjunto de triangulaciones posibles, con el preorden inducido por el orden en $\mathbb{R}$ de la suma de las longitudes de sus diagonales.

Ejemplo de polígono triangulado



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:

$$\mathcal{L} = \{\alpha_i \vee \alpha_{i+1} \ : i \in \mathbb{Z}_n\}$$

Del mismo modo, las cuerdas son:

$$\mathcal{C} = \{\alpha_i \vee \alpha_{i+2} \ : i \in \mathbb{Z}_n\}$$

Descripción 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.

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 (++).

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);
};
```

Busca la cuerda de mínima longitud recorriendo los nodos hasta volver al nodo inicial.

Si identificamos los vértices $\alpha_1, \ldots, \alpha_n$ del polígono con los elementos de $\mathbb{Z}_n$, y observamos:

- Si n es par $\implies \mathbb{Z}_n/2\mathbb{Z}_n \simeq \mathbb{F}_2$
- Si n es impar $\implies \mathbb{Z}_n/2\mathbb{Z}_n \simeq \mathbb{F}_1$

Podemos concluir que un polígono con un número par de vértices tiene dos ciclos de cuerdas, y uno con un número impar de vértices, uno solo.
Por tanto, para comparar las longitudes de todas las cuerdas de un polígono, debe llamarse a la función anterior dos veces en el primer caso, y una en el segundo.

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);
}
```

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.

Alternativa

Existen implementaciones de este algoritmo que se ejecutan en tiempo $O(n \log(n))$, algunas están descritas en:

Dickerson, Matthew T. (1997). *«Fast greedy triangulation algorithms»*. Computational Geometry (Elsevier) 67-86