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
23 de mayo de 2017

Índice

1. Introducción **2**

2. Problema **2**

3. Solución **2**
3.1. Algoritmo Greedy 2
3.1.1. Circular advance 2
3.1.2. Find Min String 3
3.1.3. NOSE QUE TITULO PONER LMAO 4
3.1.4. La complejidad del algoritmo 4

1. Introducción

Un polígono P en el plano es un conjunto de n puntos (x_i, y_i) llamados vértices, y n segmentos de rectas $1, 2, ..., n$ llamados lados, verificando 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, para cada dos puntos cualesquiera del interior del polígono, el segmento que los une está contenido en el mismo.
Por **triangulación** de un polígono plano entenderemos la división de éste en un conjunto de triángulos, con la restricción de que cada lado de un triángulo se reparta entre dos triángulos adyacentes (obviamente, sólo pertenece a uno si está en la frontera 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 a 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. Algoritmo Greedy

El algoritmo realizado se vale de dos funciones principales, *circular_advance* y *find_min_string*.

3.1.1. Circular advance

```
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;
};
```

TODO: Explicar codigo.

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

3.1.2. Find Min String

```
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);
};
```

La eficiencia teórica de la función es $O(n^2)$.

3.1.3. NOSE QUE TITULO PONER LMAO

```
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);
}
```

Dado un polígono de n nodos, n puede ser par o no serlo, en el caso de que no sea par, la función *find_min_string* recorrera todos los nodos sin problemas. Sin embargo, en el caso de que sea par, la función solo recorrera la mitad de los nodos. Por ello, se vuelve a llamar a la función para recorrer los nodos faltantes.

3.1.4. 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) realiza la función *find_min_string* la cual realiza n iteraciones en las que en cada una de ellas llama a *circular_advance* con 2 como parámetro ($O(2)$).

4. Conclusión