Prácticas de Algorítmica.
3º de Grado en Ingeniería Informática.
Curso 2019-2020.

Práctica 4.

Objetivos.

Con esta práctica se pretende que el alumno implemente un algoritmo basado en la técnica de los algoritmos voraces.

Enunciado:

El alumno deberá implementar un algoritmo voraz para obtener una aproximación poligonal de una curva digital con mínimo error. En el tema de programación dinámica se han estudiado dos métodos óptimos para obtenerlas y se han visto todos los conceptos relacionados con las aproximaciones poligonales de una curva digital.

Se suministra en el material un ejemplo completo de cómo desarrollar un algoritmo para obtener aproximaciones poligonales que se explicará con detalle en clase de prácticas. Este ejemplo contiene lo siguiente:

- Clase ***Point*** que representa un punto en 2D.
- Clase ***Line*** que representa una recta en 2D.
- Clase ***Matrix*** que representa una matriz, necesaria para hacer cálculos auxiliares.
- Clase ***DigitalCurve*** que representa una curva digital compuesta de una serie de puntos. Puede ser abierta o cerrada.
- Clase ***Algorithm*** que es una clase abstracta de la cual ha de heredar cualquier método para obtener aproximaciones poligonales que queramos implementar. Esta clase tiene implementadas la mayoría de las funciones comunes que usará cualquier método de obtención de aproximaciones poligonales.
- Clase ***SuppressionCollinearPointsMethod*** que es una clase heredada de la clase Algorithm. En esta clase se implementa un método para obtener la aproximación poligonal con el menor número de puntos que produce un error 0. Este método obtiene dichos puntos usando un método de la clase Algorithm que elimina los puntos alineados (collinearPointsElimination).
- Programa principal (***main***.***cpp***) en el cual se muestra como usar el método implementado.
- ***CmakeList***.***txt*** para generar el makefile y el ejecutable.
- Ejemplos de curvas digitales (tienen extensión txt) para probar el programa.

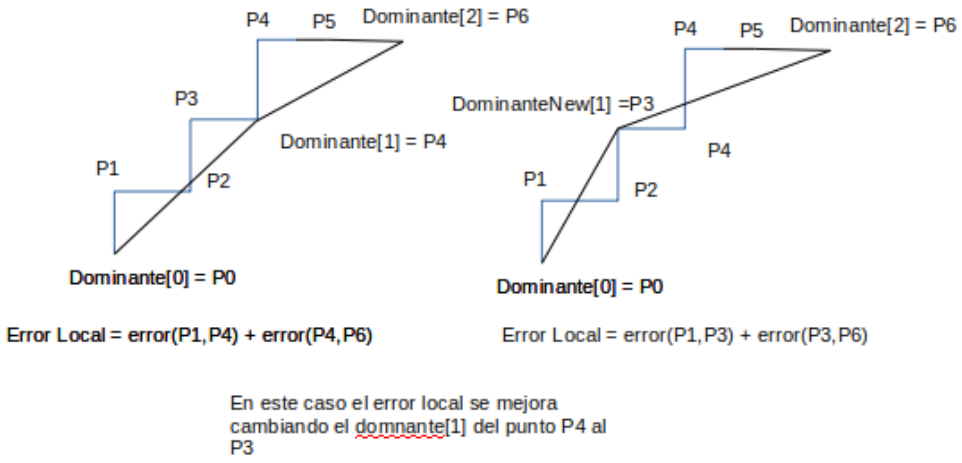
El alumno deberá implementar una clase para implementar un método voraz. Dicha clase se llamará ***GreedyMethod*** y se suministra su correspondiente ***greedymethod***.***hpp*** completo y parte del ***greedymethod***.***cpp*** donde el alumno tiene que completar el método ***apply***().

La forma de proceder es la siguiente:

- En primer lugar, en el programa principal habrá que introducir el fichero que contiene la curva digital y cuantos puntos queremos que tenga la aproximación poligonal. Si la curva es cerrada, como ocurre en todas las curvas suministradas, el punto de comienzo y el final son iguales, por tanto esto hay que tenerlo en cuenta. Si quisiéramos obtener una aproximación de 4 puntos, habrá que introducir 5.
- En el método en sí hay que seguir los siguientes pasos:
  - Hay que eliminar los puntos alineados y nos quedarán una serie de puntos que se llaman dominantes (***_dominantPointPosition***).
  - De los dominantes que nos quedan hay que seleccionar el número de puntos que queremos que contenga la aproximación polygonal y eliminar el resto. Si por ejemplo nos quedan 13 puntos(0, 1, 2, 3, 4, …., 12) y hay que seleccionar 5 nos quedaremos con

los puntos (0, 3, 6, 9, 12). Es decir, se eligen empezando por el 0 y saltando de 3 en tres puntos. El 3 se obtiene de Entero(13/(5-1)) = 3. Hay que tener en cuenta que el último también se selecciona. Ahora nos queda el número de dominantes que nos interesa.

- Ahora estos dominantes que nos quedan se van moviendo, para mejorar su posición, usando un método voraz de la siguientes forma:
  - Empezamos por el punto dominante[1] (el primero es el 0). Este punto dominante crea dos errores, primero con respecto al dominante[0] (error(dominante[0], dominante[1]) y otro con respecto al dominante[2] (error(dominante[1], dominante[2]). El error total que se genera será la suma de los dos.
  - Ahora se busca cual sería la posición ideal del punto dominante[1], la que genera menos error total, entre todos los puntos de la curva comprendidos entre dominante[0] y dominante[2] y ese punto será el nuevo punto dominante[1]. Aquí el algoritmo voraz busca el óptimo local. (Ver figura)



  - Para el punto dominante[2] se busca su posición óptima entre el nuevo dominante[1] y el punto dominante[3].
  - Para el punto dominante[i] se buscará su posición óptima entre el nuevo dominante[i-1] y el dominante[i+1].
  - El último punto dominante de la curva (coincide con el primero) se optimizará entre el nuevo penúltimo dominante y el nuevo dominante[1]. Aquí hay que desplazar el último y el primero, ya que coinciden.
  - Al finalizar estos pasos la posición de los puntos dominantes ha mejorado y el error acumulado total (ISE) ha disminuido.

Si se hace hasta aquí, la nota máxima será un 9.

Parte opcional:
Realizar varias pasadas (anteriormente solo se hace una) en el proceso de optimización hasta que el error acumulado total (ISE) no cambie. **Nota máxima: 10.**

Fecha de comienzo: 4 de noviembre.
Fecha máxima de entrega: 18 de noviembre.