Practica Opcional 2 Estructura de Datos y Algoritmos

*Búsqueda del umbral óptimo en un algoritmo Divide y Vencerás (DYV)*

**Nombre de los integrantes del grupo:**

* Daniel Serrano Torres
* Juan Luis Pérez

**Introducción**

En esta práctica trata de abordar un problema computacional complejo utilizando técnicas algorítmicas llamadas: divide y vencerás.

El problema trata de un espacio de dos dimensiones con puntos en él, por tanto son representados con coordenadas (x, y). Se busca hallar el par de puntos más cercanos entre sí.

**Algoritmo divide y vencerás (DV)**

**Descripción de la división del código en módulos**

**Archivo *main.cpp***

int main() // Funcion Principal del programa

void inicioAlgoritmoDirecto(Lista<Punto> &lista\_de\_puntos) // Implementación publica del algoritmo Directo con cuenta de tiempos

Solucion inicioAlgoritmoDYV(Lista<Punto> &lista\_de\_puntos) // Implementación publica del algoritmo DYV con cuenta de tiempos

**Archivo *mezcla.cpp***

void mezclam(Punto array\_puntos[], int a, int m, int b,Comparador menor); // Implementacion privada del algoritmo mergesort que mezcla dos listas

bool menorX(const Punto &p1, const Punto &p2); // Función que compara dos puntos por su abscisa. Su definición es compatible con el tipo Comparador

bool menorigualX(const Punto &p1, const Punto &p2); // Función que compara dos puntos por su abscisa. Su definición es compatible con el tipo Comparador

bool menorY(const Punto &p1, const Punto &p2);

Lista<Punto> mezcla(Lista<Punto> &l1, Lista<Punto> &l2, Comparador menor); // Función de mezcla que recibe la función que compara dos puntos

void mergeSort(Punto array\_puntos[], int a, int b); // Implementacion privada del algoritmo mergesort

void OrdenacionMergeSort(Lista<Punto> &l,Comparador menor); // Operación publica del algoritmo mergesort

void deListaAarray(Lista<Punto> &l, Punto array\_puntos[]);//Transforma una lista a un array

void deArrayALista(Lista<Punto> &l, Punto array\_puntos[]); // Transforma un array a lista

void partirListam(Lista<Punto> &original, int medio , Lista<Punto> &izquierda , Lista<Punto> &derecha); // Dada una lista original y un punto medio , parte en dos listas dejando 0-medio elementos en izquierda y medio+1 hasta n elementos en derecha

**Archivo *dyv.cpp***

Solucion solucionDirecta(Lista<Punto> &puntos, int n); // Dada una lista con dos o tres puntos , devuelve los dos puntos más cercanos , su distancia y una lista con los puntos ordenados crecientemente por su coordenada y.

Lista<Punto> merge(Lista<Punto> &l1, Lista<Punto> &l2); // Dadas dos listas de puntos ordenadas por la coordenada y ,devuelve otra lista con su mezcla ordenada también por la coordenada y.

Lista<Punto> filtraBanda(Lista<Punto> &l, double d, double x); // Dada una lista l de puntos , una distancia d y una abcisa x , devuelve la lista de los puntos de l cuya abscisa diste a lo sumo d de x en valor absoluto

void recorreBanda(Lista<Punto> &l, Punto &p1, Punto &p2, double &d); // Recorre una lista l de puntos , comparando cada uno con los 7 siguientes si los hubiera . Devuelve el par de puntos con menor distancia, y dicha distancia. Si l tiene un punto o ninguno, devuelve una distancia +infinito.

Solucion eligeMinimo(const Solucion &s1, const Solucion &s2,const Punto &p1, const Punto &p2, double d); // Dadas tres soluciones , cada una consistente en un par de puntos y su distancia , devuelve el par más cercano de los tres y su distancia .

Solucion parMasCercano(Lista<Punto> &puntos, int n,int umbral); //Llamada a funcion del algoritmo DYV

**Archivo *faux.cpp***

bool iguales(Solucion &s,Solucion &s1) // Funcion de testing utilizada durante el desarrollo de la practica

Lista<Punto> generarListaDePuntos(int n); //Rellena una lista con N puntos aleatorios

void imprimeListadePuntos(Lista<Punto> &l); //Imprime una lista de puntos de forma adecuada

void imprimeUnicoPunto(const Punto &p1); //Funcion que muestra un punto de forma adecuada -- Auxiliar

**Archivo *humbral.cpp***

bool directaMenor(float tiempoDirecto, float tiempoDV);

Función que indica si el tiempo del algoritmo directo es menor que el del algoritmo de divide y venceras.

int humbral(int numero\_de\_puntos);

Función que obtiene el número de puntos a partir del cual el algoritmo directo es más eficiente

**Instrucciones para compilar el código**

Se ha utilizado un entorno de desarrollo integrado como NetBeans o Xcode, etc. De todas maneras puede ser compilado mediante el uso del compilador g++ .

Las únicas dependencias son las propias al uso de las librerías internas del propio lenguaje C++.

**Cálculo del umbral experimental**

El modo empleado para obtener el umbral donde el algoritmo

**Método empleado para obtener los tiempos y generar las graficas**

Se ha utilizado un cálculo del tiempo tardado por cada algoritmo, y con los datos obtenidos por el programa se han generado las gráficas en una hoja de cálculo.

Las gráficas siguientes corresponden a un estudio de 6 casos, con número de puntos: 100, 500, 1000, 5000, 10000, 20000 respectivamente. Todos los casos han sido realizados con ambos algoritmos, cuadrático (directo) y divide y vencerás (DV).

En esta última gráfica se representan ambos algoritmos junto para ver de una forma clara y sencilla las diferencias en cuestión del tiempo tardado por ambos para el mismo tamaño de problema (número de puntos).

Se puede apreciar rápidamente que cuando el número de puntos del espacio del problema aumenta significativamente el algoritmo cuadrático es mucho más lento.