A.M.C.

Memoria Práctica 1:

Estrategias Algorítmicas

Autores:

Miguel Quiroga Campos
Pablo Fernández Ivars
Daniel Linfon Ye Liu
Pablo Gómez Arroyo

Curso 2023/2024

Parte A

Exhaustivo

```
lista[numeroPuntos]
i <- 0

mientras (i < numeroPuntos-1) {                                    //n
        j <- i +1
        mientras (j < numeroPuntos) {                              //n
                si (distancia(i,j) != 0 Y distancia(i,j) < distanciaMinima)
                        distanciaMinima = distancia(i,j)
                j++
        }
        i++
}

devolver distanciaMinima
```

Análisis del coste:

- ***Caso Mejor:*** Este se da cuando entre los dos primeros puntos de la lista se encuentra la menor distancia entre puntos. El coste en este caso sería de **O(n^2),** ya que de todas formas sigue teniendo qué entrar en los bucles para hacer las comparaciones, solo nos ahorramos k operaciones lógicas.

- ***Caso Peor:*** Sucede cuando la distancia más corta entre puntos está entre los dos últimos. El coste es análogo al caso mejor, es decir, **O(n^2),** por lo qué, aun realizando más OE, sigue teniendo el mismo coste.

Exhaustivo Poda

```
lista[numeroPuntos]
i <- 0
heapSortX(lista)


mientras (i < numeroPuntos-1) {                                    //n
        j <- i +1
        mientras (j < numeroPuntos Y i.x - j.x < distanciaMinima) {    //log n
                si (distancia(i,j) != 0 Y distancia(i,j) < distanciaMinima)
                        distanciaMinima = distancia(i,j)
                j++
        }
        i++
}

devolver distanciaMinima
```

Análisis del coste:

- ***Caso Mejor:*** Este se da cuando entre los dos primeros puntos de la lista previamente ordenada por el eje x está la distancia mínima entre dos puntos de toda la lista. El orden sería **O(n*log(n))**, actualizando la distancia mínima una sola vez.

- ***Caso Peor:*** Ocurre cuando los dos puntos entre los qué se encuentra la distancia mínima son los dos últimos de la lista, por lo qué debería de ir actualizando la distancia mínima en cada iteración. Aun así, el coste sigue siendo **O(n*log(n))**, solo ahorrandonos las asignaciones y comparaciones (+k operaciones elementales).

Divide y Vencerás

```
int DyV(Punto lista[], int izq, int der) {

        int distanciaMin

        si (der - izq == 1) { // si sólo hay dos puntos (CASO BASE)      // Caso Base = O(1)

                distanciaMin = distancia(lista[izq], lista[der])

        } sino si (der - izq > 1) { // si hay más de dos puntos

        d1 = DyV(lista, izq, izq + ((der - izq) / 2))                    // T(n/2)
        d2 = DyV(lista, izq + (((der - izq) / 2) + 1), der)              // T(n/2)

        si (d1 <= d2)
                distanciaMin = d1
        sino
                distanciaMin = d2


        puntoMedio = ((lista[der].getx() - lista[izq].getx()) / 2) + lista[izq].getx()

        limiteIzq = puntoMedio - distanciaMin                            // Peor caso: limiteIzq = puntoMedio
        limiteDer = puntoMedio + distanciaMin                            // Peor caso: limiteDer = puntoMedio

        mientras (lista[izq].getx() < limiteIzq)                         // O(n)
                izq++


        mientras (lista[der].getx() > limiteDer)                         // O(n)
                der--

        d3 = Exhaustivo(lista, izq, der)                                 // O(n^2)

        si (d3 < distanciaMin)
                d3 = distanciaMin

    devolver distanciaMin ;
  }
```

Análisis del coste:

si n < 3: T(n) = O(1)

si n >= 3:

- ***Caso Peor:*** Cuando ambos límites (izq y der) son el principio y el final del array.

T(n) = 2T(n/2) + O(n^2) = O(n^2)

- ***Caso Mejor:*** Cuando ambos límites (izq y der) están en el punto medio del array.

T(n) = 2T(n/2) + O(n) = O(n*log(n))

Teorema (conocido como Master Theorem): La solución a la ecuación $T(n) = aT(n/b) + \Theta(n^k)$, con $a \geq 1$ y $b > 1$, es: (Dem. pág. 383 Weiss95)

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$$

Divide y Vencerás Mejorado

```
HeapSort(lista) //por la coordenada x

int DyV(Punto lista[], int izq, int der) {
        int distanciaMin

        si (der - izq == 1) { // si sólo hay dos puntos (CASO BASE)      // Caso Base = O(1)

                distanciaMin = distancia(lista[izq], lista[der])

        } sino si (der - izq > 1) { // si hay más de dos puntos

        d1 = DyV(lista, izq, izq + ((der - izq) / 2))                    // T(n/2)
        d2 = DyV(lista, izq + (((der - izq) / 2) + 1), der)              // T(n/2)

        si (d1 <= d2)
                distanciaMin = d1
        sino
                distanciaMin = d2

        puntoMedio = ((lista[der].getx() - lista[izq].getx()) / 2) + lista[izq].getx()

        limiteIzq = puntoMedio - distanciaMin
        limiteDer = puntoMedio + distanciaMin

        mientras (lista[izq].getx() < limiteIzq)                         // O(n)
                izq++

        mientras (lista[der].getx() > limiteDer)                         // O(n)
                der--

        Punto listaAux[(der-izq)+1] //lista del tamaño de la zona crítica

        j = izq;

        for (i = 0 hasta i < ((der-izq)+1)) {                            // O(n)
                listaAux[i] = lista[j];
                 j++;
        }

        HeapSort(listaAux) //por la coordenada y                         // O(n*log(n))

         j=izq;
                //queda la lista original ordenada por la 'y' sólo en la zona crítica
         for (int i = 0 hasta i < ((der-izq)+1)) {                       // O(n)
                 lista[j] = listaAux[i];
                 j++;
        }

        d3 = Exhaustivo12Pos(lista, izq, der)                            // O(n)

        si (d3 < distanciaMin)
                d3 = distanciaMin

    devolver distanciaMin ;
```

Análisis del coste:

si n < 3: T(n) = O(1)

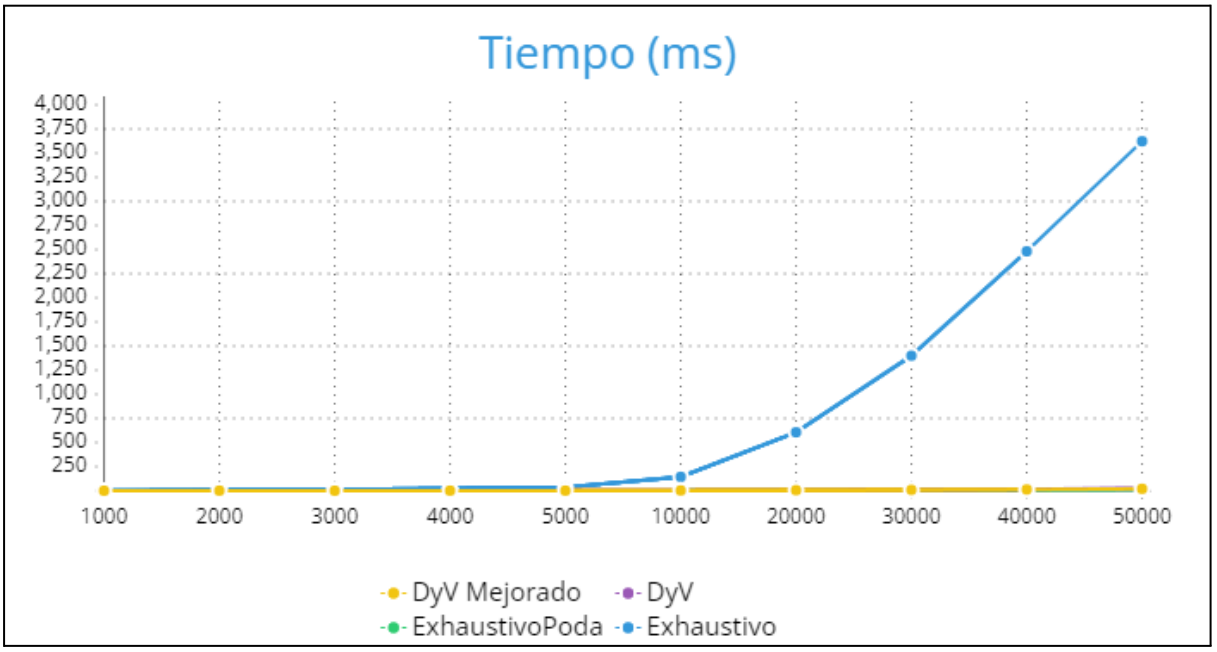
si n >= 3: Para este algoritmo no existe un caso mejor o peor para n >= 3, simplemente evita que se dé el caso peor anterior, por lo que su coste siempre va a ser:

T(n) = 2T(n/2) + O(n) = O(n*log(n))

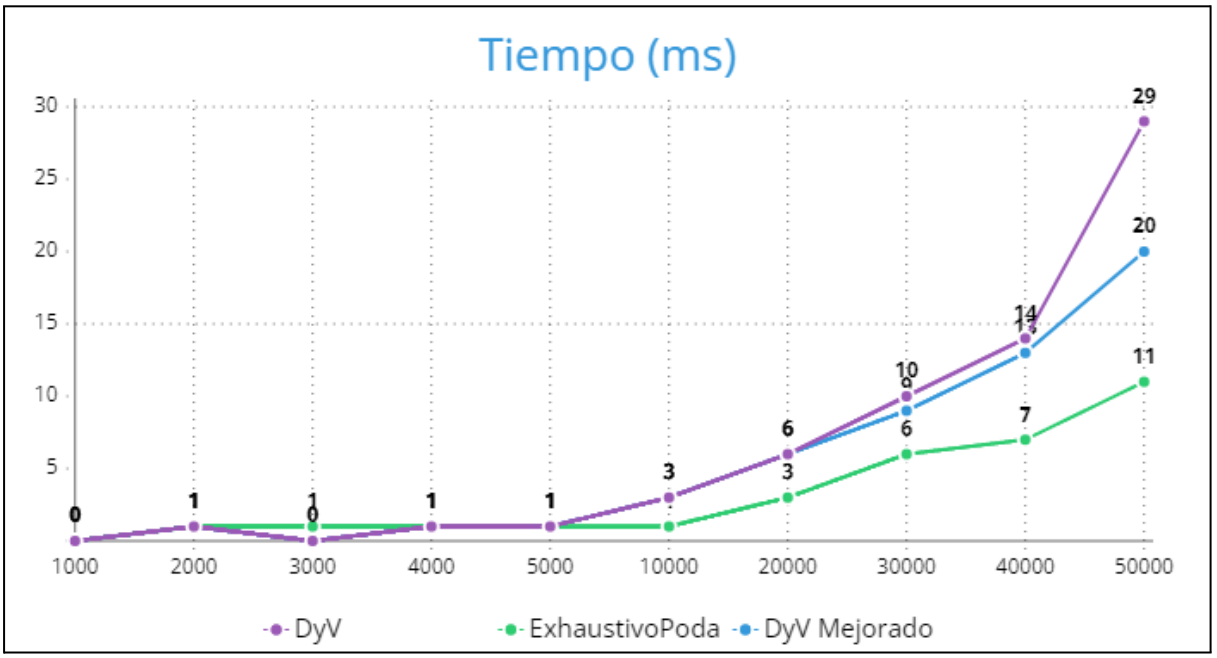
Teorema (conocido como Master Theorem): La solución a la ecuación $T(n) = aT(n/b) + \Theta(n^k)$, con $a \geq 1$ y $b > 1$, es: (Dem. pág. 383 Weiss95)

$$T(n) = \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$$

COMPARACIÓN EXPERIMENTAL



*** Si hacemos zoom en los 3 de abajo:**

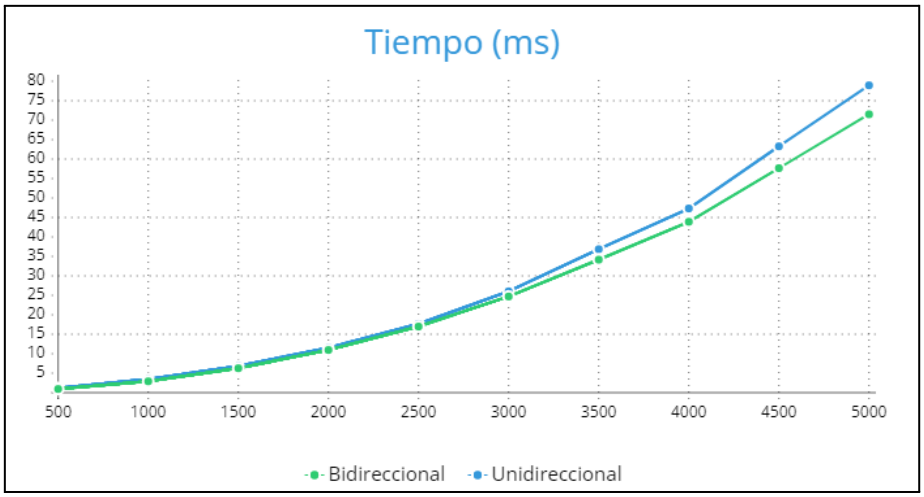


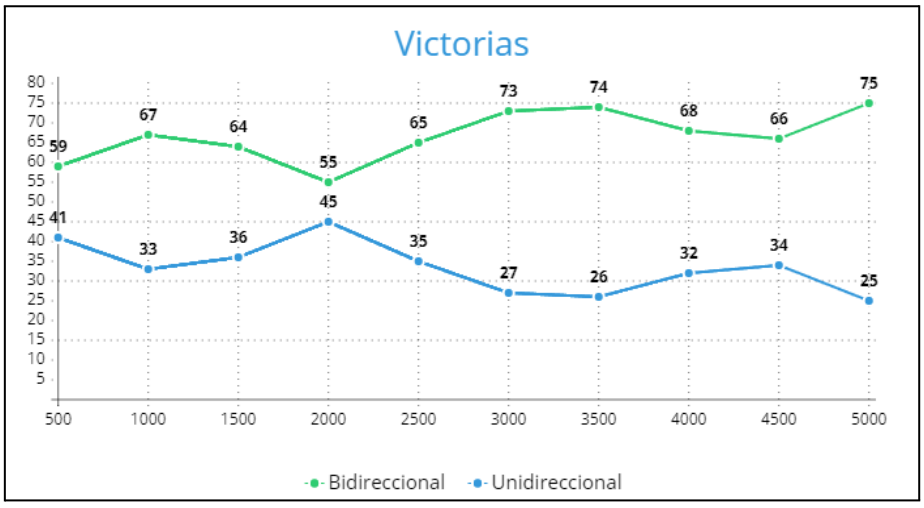
Observamos que los 3 se mantienen en un tiempo muy similar, lo cual era de esperar puesto que todos ellos se acercan mucho a un coste de O(n*log(n)) en casos promedio, no obstante, a medida que la talla aumenta, vemos cómo se va haciendo cada vez más notable la superioridad del algoritmo exhaustivo con poda, seguido del divide y vencerás mejorado. Mientras que el algoritmo Exhaustivo representa la curva acorde a su coste que es de O(n^2) siempre.

Parte B

COMPARACIÓN EXPERIMENTAL

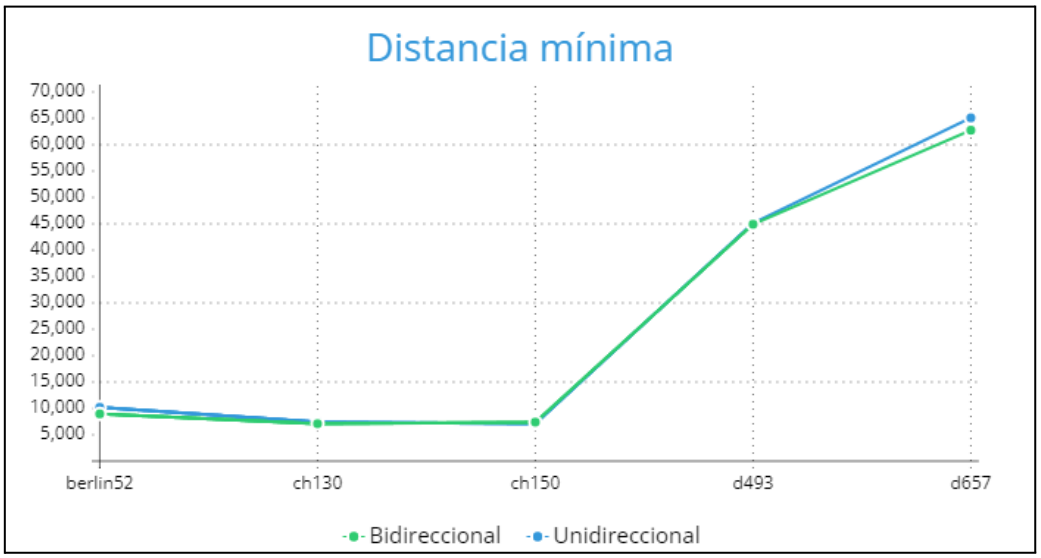
	***********Comparacion de las estrategias**********	
	Unidireccional	Bidireccional
Talla 500-->	41(1.16ms)	59(0.94ms)
Talla 1000-->	33(3.46ms)	67(2.91ms)
Talla 1500-->	36(6.84ms)	64(6.27ms)
Talla 2000-->	45(11.54ms)	55(10.96ms)
Talla 2500-->	35(17.67ms)	65(16.98ms)
Talla 3000-->	27(26.03ms)	73(24.7ms)
Talla 3500-->	26(36.85ms)	74(34.16ms)
Talla 4000-->	32(47.32ms)	68(43.9ms)
Talla 4500-->	34(63.28ms)	66(57.67ms)
Talla 5000-->	25(78.95ms)	75(71.51ms)

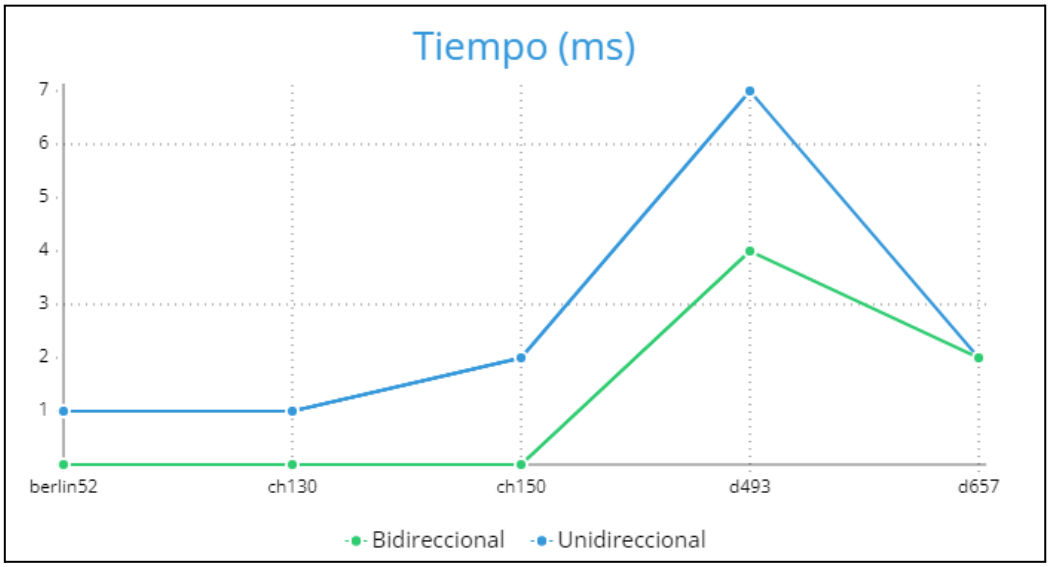




Sobre los datasets proporcionados:

Datasets	berlin52	ch130	ch150	d493	d657
Unidireccional (distancia)	10200'8941	7422'6152	7092'7051	45107'2831	65096'1116
Bidireccional (distancia)	8962'0925	7101'5967	7395'8413	44909'0531	62750'8559
Unidireccional (ms)	1.0	1.0	2.0	7.0	2.0
Bidireccional (ms)	0.0	0.0	0.0	4.0	2.0





CONCLUSIONES

En la primera comparación, con diferentes tallas, observamos que la estrategia de Búsqueda Bidireccional es superior a la Unidireccional, ganando el **66.6%** de las veces y empleando un tiempo ligeramente menor.

Por otro lado, en la segunda comparativa, tenemos resultados mucho más similares, pero que siguen mostrando la ligera superioridad de la estrategia Bidireccional (tanto en distancia como en tiempo), que se hace algo más notable a medida que la talla del conjunto de puntos aumenta. No obstante, al realizarse sólo una única ejecución sobre cada dataset, no son resultados tan significativos como los de la comparativa anterior.