RESOLUCIÓN DEL PROBLEMA DE LA DISTANCIA ENTRE LOS PUNTOS MÁS CERCANOS DE UN CONJUNTO.

ANÁLISIS

Se tiene un conjunto de puntos C= {p_1, p_2, …, p_n }, donde cada p_i se identifica por sus coordenadas p_i=(x_i, y_i), y se encuentran ordenados de modo que $p_i < p_j$ sii $x_i < x_j$.

El objetivo es encontrar: min{ d(p_i, p_j) para todo i, j} , donde

$$d(p_i, p_j) = sqrt((x_i - x_j)^2 + (y_i - y_j)^2)$$

Durante el desarrollo del ejercicio también usaremos la notación: $d_x(p_i, p_j) = sqrt((x_i - x_j)^2)$

DISEÑO DE ALGORITMO BÁSICO

La idea general de un algoritmo básico para resolver el problema sería recorrer cada par de puntos e ir guardando la distancia mínima S que se fuese encontrando. Al final, devolveríamos la distancia mínima encontrada.

El diseño del algoritmo sería:

ALGORITMO S= BASICO(C= {p_1, p_2, …, p_n } : Conjunto de puntos)

```
S= Infinito
Para cada i desde 1 hasta n-1:
        Para cada j desde i+1 hasta n:
                Si (d(pi, pj) < S)
                        S= d(pi, pj)
                Fin-Si
        Fin-Para
Fin-Para
Devolver S
```

DISEÑO DEL ALGORITMO DIVIDE Y VENCERÁS

Las componentes del algoritmo son:

- **División del problema en subproblemas:**
  - El problema original P=C se divide en k=2 subproblemas P1, P2, que se resuelven por separado dando lugar a las soluciones S1 de P1 y S2 de P2.
  - El problema P se divide por la mitad. Asumiendo que c es el índice del punto central en el conjunto C, creamos el problema P1={p_1, p_2, …, p_{c-1}} y el problema P2 los puntos P2= {p_c, p_{c+1}, …, p_n}. Así, el tamaño |P1| y el tamaño |P2| es aproximadamente el mismo, los problemas tienen la misma naturaleza que el problema original P, son independientes y

se pueden resolver por separado. Como los puntos están ordenados por la coordenada X, podremos llamar a P1 “parte izquierda” y a P2 “parte derecha”.

- **Existencia de caso base:**
  - Dejaremos de dividir cuando el número puntos en el conjunto sea |P|<=3. En tal caso, resolvemos con el algoritmo básico.
- **Combinar las subsoluciones:**
  - Idea general: S1 es la distancia mínima de los puntos del subproblema P1, y S2 es la distancia mínima de los puntos del subproblema P2. Para combinar las soluciones S1 de P1 y S2 de P2, podríamos pensar que S=min{S1, S2}. Sin embargo, también hay que considerar que los puntos más cercanos se encuentren uno en P1 y otro en P2. Por tanto, escogeremos los puntos de P1 que disten del centro como mucho min{S1, S2} en su coordenada X, y los puntos de P2 que disten del centro como mucho min{S1, S2} en su coordenada X, y compararemos los puntos seleccionados de la parte izquierda con los puntos seleccionados de la parte derecha.

El diseño del algoritmo DyV sería:

```
ALGORITMO S= DyV( C : Conjunto de puntos ordenados por su coordenada x)
    Si |C| <= 3:
        Devolver S= BASICO( C )
    En otro caso:
        p_c ← Punto central de C
        P1 ← Puntos de C hasta p_c (no incluido)
        P2 ← Puntos de C desde p_c hasta n
        S1 ← DyV(P1)
        S2 ← DyV(P2)
        S ← COMBINAR(p_c, P1, S1, P2, S2 )
        Devolver S
    Fin-En otro caso
```

El método de combinación sería:

```
ALGORITMO S= COMBINAR(p_c : Punto central del problema
                      P1 : Subproblema de P (parte izquierda)
                      S1 : Solución de P1,
                      P2 : Subproblema de P (parte derecha)
                      S2: Solución de P2):

    PARCIAL= min{S1, S2}
    S= PARCIAL
    izq ← índice el punto de P1 más lejano de p_c con d_x(p_izq, pc)<=PARCIAL
    dcha ← índice el punto de P2 más lejano de p_c con d_x(p_dcha, pc)<=PARCIAL
    Para cada punto p_i en P1 con i>=izq:
        Para cada punto p_j en P2 con j<=dcha:
            Si d(p_i, p_j) < S:
                S= d(p_i, p_j)
            Fin-Sin
        Fin-para
    Fin-Para
    Devolver S
```

**** NOTA IMPORTANTE DEL PROFESOR:** Existen otras variantes para la función de combinación que mejoran sustancialmente el orden de eficiencia teórico O. Estas no han sido reflejadas en esta solución al objeto de facilitar la comprensión de la metodología DyV.

DETALLES DE IMPLEMENTACIÓN

En la solución planteada en el código fuente:

- Un punto se representa con una estructura Punto con dos campos (x,y).
- El conjunto de puntos $C=\{c_{ini} \ldots c_{fin-1}\}$ Se representa con un array indexado de ini a fin-1 inclusive.
- El valor Infinito de la distancia se representa con valor "-1" no válido para distancias.

Existen dos ficheros en la implementación:

- **GenerarPuntos.cpp:** Contiene código para generar un número de puntos (dado como parámetro al programa) ordenados por componente x. El número de puntos y los puntos en sí se proporcionan por salida estándar. Un ejemplo de su uso:

  ./GenerarPuntos 100 > dataset100.txt

  Genera un fichero dataset100.txt con un conjunto de 100 puntos aleatorios, ordenados por componente X.

- **Solucion.cpp**: Contiene:
  - Código para leer un conjunto de datos en el formato especificado por GenerarPuntos (función **LeerDatos**).
  - Código para calcular la distancia $d(p_i, p_j)$ (Función **distancia**).
  - Código para calcular la distancia $d_x(p_i, p_j)$ (Función **distanciaEnX**).
  - Código para implementar el algoritmo básico (Función **BASICO**).
  - Código para implementar el algoritmo DyV (Función **DyV**).
  - Código para implementar la función de combinación de DyV (Función **Combinar**).

  Adicionalmente, el programa principal realiza el siguiente flujo de acciones:
  1. Leer datos del conjunto de datos por consola (se guarda en variable **cPuntos**, de tamaño **n**).
  2. Ejecutar el algoritmo básico para encontrar el mínimo de la distancia entre todos los puntos, midiendo también su tiempo de ejecución.
  3. Ejecutar el algoritmo DyV para encontrar el mínimo de la distancia entre todos los puntos, midiendo también su tiempo de ejecución.
  4. Mostrar por consola los resultados obtenidos por **BASICO** y **DyV**, así como sus tiempos de ejecución.

  Un ejemplo de uso de este programa sería:

  ./Solucion < dataset100.txt