Analisis de Algoritmos
Tarea 1

Nombres de los alumnos del grupo

April 11, 2025

1 Fuerza Bruta

1.1 Implementación

La implementación propuesta es la que se puede apreciar en la **Figura 1**.
Se utilizo un Vector para almacenar el conjunto de puntos, se declara una variable minDistance encargada de almacenar el minimo valor, tiene 2 ciclos anidados que se encargan de recorrer todos los puntos del conjunto buscando la menor distacia en ellos.
En la seccion de correctitud se detalla el funcionamiento del algoritmo.

```cpp
double calculateDistance (std::pair<double, double> p1, std::pair<double, double> p2) {
    return sqrt(pow(p1. first - p2. first , 2) + pow(p1.second - p2.second , 2));
};

int main() {

    //  Creamos un vector que almacene los puntos
    std::vector<std::pair<int, int>> dots = {{1, 2}, {3, 4}, {5, 6}, {7, 8}};
    double minDistance = std::numeric_limits<double>::max();

    // Calcularemos la distancia minima entre 2 puntos y almacenamos la minima distancia
    for (size_t i = 0; i < dots.size(); i++)
    {
        for (size_t j = i+1; j < dots.size(); j++)
        {
            double distance = calculateDistance(dots[i], dots[j]);
            if (minDistance == 0 || distance < minDistance)
            {
                minDistance = distance;
            }
        }
    }
```

Figure 1: Implementación fuerza bruta.

1.2 Correctitud

Nuestro algoritmo recibe un conjunto $S = \{(X_i, Y_i)\} \mid i \in [0, n-1]$ de n puntos en el plano y devuelve la mínima distancia entre ellos.

Input: Conjunto $S = \{(X_i, Y_i)\} \mid i \in [0, n-1]$ de n puntos en el plano.
Output: La distancia mínima entre puntos distintos del conjunto.

Main:

Declara una variable `minDistance` inicializándola con un número grande
for cada punto (X_i, Y_i) en el conjunto S **do**
    for cada punto (X_j, Y_j) tal que $j > i$ **do**
        Calcular la distancia euclidiana entre (X_i, Y_i) y (X_j, Y_j)
        if esta distancia es menor que `minDistance` **then**
            Reemplazar `minDistance` con esta distancia
        end if
    end for
end for
Devolver `minDistance`

Para analizar la correctitud del algoritmo, utilizaremos **Invariante de ciclo**
Antes del primer ciclo:
Al iniciar el primer ciclo minDistance almacena un valor tan grande que puede ser reemplazado con cualquier distancia.
En el ciclo:
En cada ciclo, se comparan 2 puntos, si su distancia es menor a la que esta almacenada en minDistance, actualizara esta variable. Esto garantiza que minDistance siempre mantenga la menor distancia encontrada.
Al salir del ciclo:
Al finalizar el ciclo habremos revisado todas las posibles combinaciones de pares de puntos en S, por lo que minDistance almacenara la minima distancia entre cualquier par de puntos en S.

1.3 Complejidad Computacional

El primer ciclo del código propuesto recorre i desde 0 hasta $n-1$, con lo que se realizan $n-1$ iteraciones. Luego, el segundo ciclo interno comienza con $j = i+1$, por lo tanto, para la primera iteración del primer ciclo se hacen $n-1$ comparaciones, luego $n-2$, luego $n-3$, y así sucesivamente, hasta llegar a 1. Esto se puede expresar como la suma:

$$(n-1) + (n-2) + (n-3) + \cdots + 2 + 1 = \frac{(n-1) \cdot n}{2}$$

Por lo tanto, la complejidad del algoritmo es $\mathcal{O}(n^2)$.

2 Dividir para Vencer

2.1 Diseño

El diseño de *Divide and Conquer* se basa en el algoritmo de **GeeksforGeeks**[1]. El Algoritmo es el siguiente:

Input: Conjunto $S = \{(X_i, Y_i)\} \mid i \in [0, n-1]$ de n puntos en el plano.
Output: La distancia mínima entre puntos distintos del conjunto.

Primero, el algoritmo ordena los puntos del conjunto S según su coordenada X. Una vez ordenado, se busca el punto medio y se divide el conjunto en dos subconjuntos:

- $P_{\text{izq}} = \{(X_0, Y_0), \ldots, (X_{\lfloor n/2 \rfloor - 1}, Y_{\lfloor n/2 \rfloor - 1})\}$
- $P_{\text{der}} = \{(X_{\lfloor n/2 \rfloor}, Y_{\lfloor n/2 \rfloor}), \ldots, (X_{n-1}, Y_{n-1})\}$

Luego, de manera recursiva, se busca la distancia mínima en los conjuntos P_{izq} y P_{der}:

$$d = \min(d_{\text{izq}}, d_{\text{der}})$$

Con esto, sabemos que la distancia mínima del conjunto completo debe ser menor o igual a d.

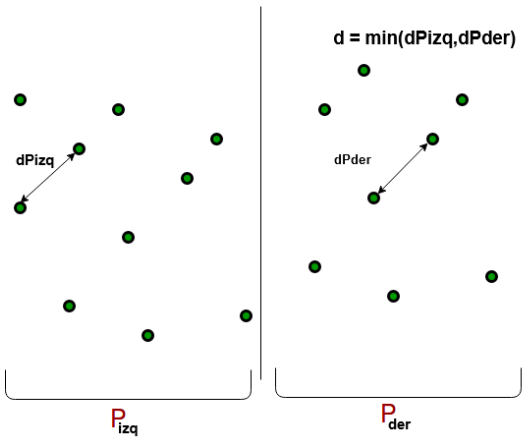

Figure 2: Division del Conjunto.

Ahora, consideramos los pares que están en ***la frontera***, es decir, aquellos en los que un punto pertenece a la parte izquierda y otro a la derecha. Imaginando una línea vertical que pasa por $X_{\lfloor n/2 \rfloor}$ como se aprecia en la **Figura 3**, analizamos

los pares de puntos cuya coordenada x está a una distancia menor que d de esta línea.

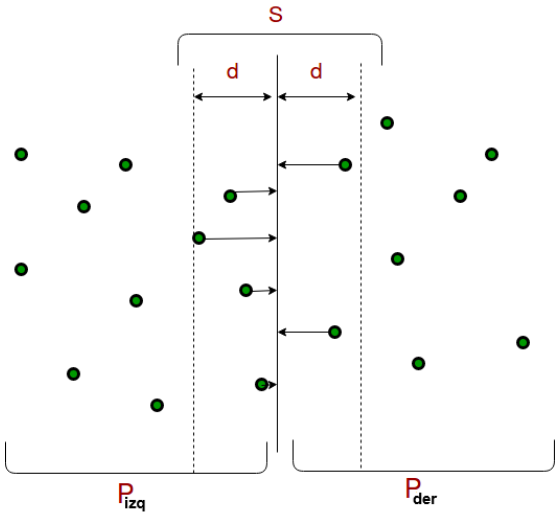

Figure 3: Frontera del conjunto dividido.

Para los puntos en la frontera, primero los ordenamos por su coordenada Y. Luego, para cada punto, lo comparamos con los siguientes 6 o 7 puntos más cercanos a lo largo del eje Y. Esto se debe a que, al ordenar por Y, los puntos en la franja vertical de ancho $2d$ están distribuidos en un área donde, geométricamente, no pueden existir más de 6 puntos a menos de d de distancia entre sí. Esta propiedad se basa en la densidad máxima de puntos en un rectángulo de dimensiones $d \times d$.

Finalmente, la distancia mínima entre los puntos del conjunto S será:

$$\min(d_{\text{izq}}, d_{\text{der}}, d_{\text{frontera}})$$

2.2 Correctitud

Demostraremos la correctitud del Algoritmo *divide and conquer* para encontrar la distancia mínima entre dos puntos funciona correctamente usando inducción matemática.

Hipótesis

Sea $P(n)$: "El algoritmo encuentra correctamente la distancia mínima entre dos puntos distintos de un conjunto de n puntos en el plano".

Caso base: $n = 2$ o $n = 3$

Cuando hay solo 2 o 3 puntos, el algoritmo compara todas las posibles distancias directamente. En este caso, claramente devuelve la distancia mínima de forma correcta. Por lo tanto, $P(2)$ y $P(3)$ son verdaderos.

Paso inductivo

Supongamos que el algoritmo funciona correctamente para todo conjunto de k puntos, con $k \leq n$. Queremos probar que también funciona para un conjunto de $n + 1$ puntos. El algoritmo:

- Ordena los puntos por coordenada x.
- Divide el conjunto en dos mitades.
- Aplica el algoritmo recursivamente en cada mitad. Por la hipótesis inductiva, cada mitad devuelve la distancia mínima correcta.
- Luego compara los puntos cercanos a la línea divisoria (frontera), y revisa solo los que pueden estar a una distancia menor que la mínima encontrada hasta ahora.

Finalmente, devuelve el mínimo entre:

- la distancia mínima en la mitad izquierda,
- la distancia mínima en la mitad derecha,
- y las distancias entre puntos cercanos a la frontera.

Esto asegura que el resultado es correcto también para $n + 1$ puntos.

Conclusión

Por inducción matemática, el algoritmo funciona correctamente para cualquier cantidad de puntos $n \geq 2$.

2.3 Complejidad Computacional

El algoritmo en sencillos pasos hace lo siguiente.
primero

1. Ordena todos los puntos del conjunto por su coordenada X,
2. luego divide el conjunto en dos mitades, despues
3. recursivamente encuentra la distancia en las dos mitades
4. combina los resultados considerando los puntos cercanos a la frontera

5. Comparar todos los pares en la frontera dentro de una distancia d del eje divisorio
6. y finalmente, ordenar esa franja por coordenada Y e compara solo hasta 6-7 vecinos siguientes.

El ordenar los puntos por su coordenada x tiene complejidad O(n $\log n$),

3 Implementación

Implemente su algoritmo `divide_and_conquer`, comparando sus respuestas con las de su implementación de `brute_force`.

4 Análisis Experimental

4.1 Diseño

Diseñe un análisis experimental calculando los tiempos de ejecución en nanosegundos de sus dos soluciones para un conjunto de n puntos cuyas coordenadas enteras están elegidas al azar en un cuadrado de 100×100, variando la cantidad n de puntos entre las potencias de dos de $2^3 = 8$ a $2^9 = 512$.

4.2 Realización

Realice el análisis experimental diseñado, guardando los resultados obtenidos.

5 Mejora

Observando el tiempo de ejecución para valores grandes de n, proponga y evalúe una versión trivialmente mejorada de los dos algoritmos.

6 Gráfico

Construya un gráfico que muestre cómo varían los tiempos de ejecución de sus cuatros soluciones en nanosegundos, variando la cantidad n de puntos entre las potencias de dos de $2^3 = 8$ a $2^9 = 512$.

References

[1] GeeksforGeeks. *Closest Pair of Points using Divide and Conquer algorithm*. Accedido el 9 de abril de 2025. Feb. 13, 2023. URL: `https://www.geeksforgeeks.org/closest-pair-of-points-using-divide-and-conquer-algorithm/`.