Algoritmo del par más cercano

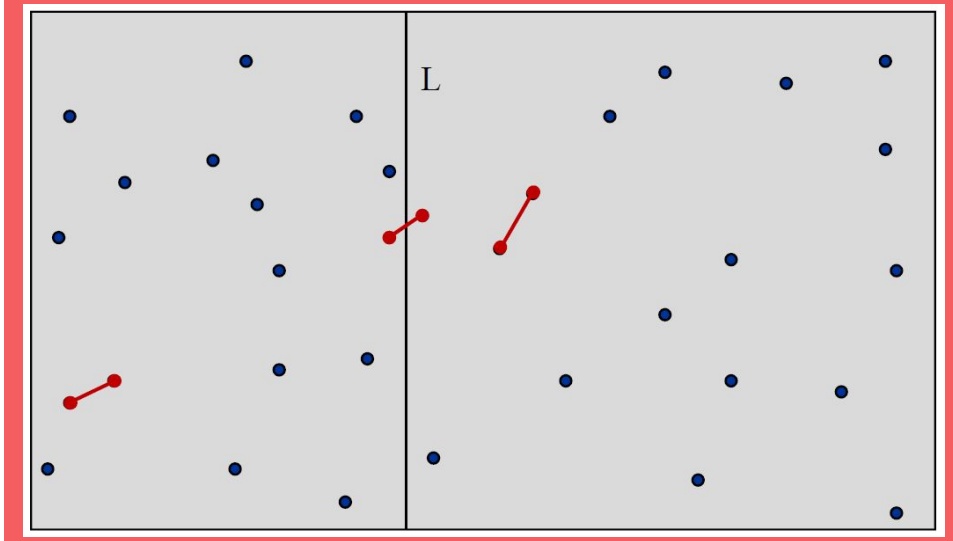
(Algorithm for the closest pair)

Aduanich Rguez Rguez
Alejandro David Carrillo Padrón
Daniel Darias Sánchez

Índice

Introducción

Proporciona una manera de encontrar los dos puntos más cercanos entre sí de un conjunto de puntos dados en dos dimensiones en un caso planar.



Explicación

Método no óptimo
FUERZA BRUTA

Se realiza la comparación de cada punto dentro del vector de puntos con el resto hasta encontrar el caso con el par más cercano.

Método óptimo
DIVIDE Y VENCERÁS

Se divide el vector n veces hasta que el tamaño de cada sección sea 2 que es el tamaño mínimo para hacer comparaciones de dos puntos del algoritmo.

Pseudocódigo

```
bruteForceClosestPair of P(1), P(2), ... P(N)
if N < 2 then
  return ∞
else
  minDistance ← |P(1) - P(2)|
  minPoints ← { P(1), P(2) }
  foreach i ∈ [1, N-1]
    foreach j ∈ [i+1, N]
      if |P(i) - P(j)| < minDistance then
        minDistance ← |P(i) - P(j)|
        minPoints ← { P(i), P(j) }
      endif
    endfor
  endfor
  return minDistance, minPoints
 endif
```

```
closestPair of (xP, yP)
               where xP is P(1) .. P(N) sorted by x coordinate, and
                     yP is P(1) .. P(N) sorted by y coordinate (ascending order)
if N ≤ 3 then
  return closest points of xP using brute-force algorithm
else
  xL ← points of xP from 1 to ⌈N/2⌉
  xR ← points of xP from ⌈N/2⌉+1 to N
  xm ← xP(⌈N/2⌉)x
  yL ← { p ∈ yP : px ≤ xm }
  yR ← { p ∈ yP : px > xm }
  (dL, pairL) ← closestPair of (xL, yL)
  (dR, pairR) ← closestPair of (xR, yR)
  (dmin, pairMin) ← (dR, pairR)
  if dL < dR then
    (dmin, pairMin) ← (dL, pairL)
  endif
  yS ← { p ∈ yP : |xm - px| < dmin }
  nS ← number of points in yS
  (closest, closestPair) ← (dmin, pairMin)
  for i from 1 to nS - 1
    k ← i + 1
    while k ≤ nS and yS(k)y - yS(i)y < dmin
      if |yS(k) - yS(i)| < closest then
        (closest, closestPair) ← (|yS(k) - yS(i)|, {yS(k), yS(i)})
      endif
      k ← k + 1
    endwhile
  endfor
  return closest, closestPair
endif
```

Código y Explicación

```cpp
std::pair<double, points_t> find_closest_brute(const std::vector<point_t>& points) {
	if (points.size() < 2) {
		return { -1, { { 0, 0 }, { 0, 0 } } };
	}
	auto minDistance = std::abs(distance_between(points.at(0), points.at(1)));
	points_t minPoints = { points.at(0), points.at(1) };
	for (auto i = std::begin(points); i != (std::end(points) - 1); ++i) {
		for (auto j = i + 1; j < std::end(points); ++j) {
			auto newDistance = std::abs(distance_between(*i, *j));
			if (newDistance < minDistance) {
				minDistance = newDistance;
				minPoints.first = *i;
				minPoints.second = *j;
			}
		}
	}
	return { minDistance, minPoints };
}
```

```cpp
std::pair<double, points_t> find_closest_optimized(const std::vector<point_t>& xP,
	const std::vector<point_t>& yP) {
	if (xP.size() <= 3) {
		return find_closest_brute(xP);
	}
	auto N = xP.size();
	auto xL = std::vector<point_t>();
	auto xR = std::vector<point_t>();
	std::copy(std::begin(xP), std::begin(xP) + (N / 2), std::back_inserter(xL));
	std::copy(std::begin(xP) + (N / 2), std::end(xP), std::back_inserter(xR));
	auto xM = xP.at(N / 2).first;
	auto yL = std::vector<point_t>();
	auto yR = std::vector<point_t>();
	std::copy_if(std::begin(yP), std::end(yP), std::back_inserter(yL), [&xM](const point_t& p) {
		return p.first <= xM;
	});
	std::copy_if(std::begin(yP), std::end(yP), std::back_inserter(yR), [&xM](const point_t& p) {
		return p.first > xM;
	});
	auto p1 = find_closest_optimized(xL, yL);
	auto p2 = find_closest_optimized(xR, yR);
	auto minPair = (p1.first <= p2.first) ? p1 : p2;
	auto yS = std::vector<point_t>();
	std::copy_if(std::begin(yP), std::end(yP), std::back_inserter(yS), [&minPair, &xM](const point_t& p) {
		return std::abs(xM - p.first) < minPair.first;
	});
	auto result = minPair;
	for (auto i = std::begin(yS); i != (std::end(yS) - 1); ++i) {
		for (auto k = i + 1; k != std::end(yS) &&
		 ((k->second - i->second) < minPair.first); ++k) {
			auto newDistance = std::abs(distance_between(*k, *i));
			if (newDistance < result.first) {
				result = { newDistance, { *k, *i } };
			}
		}
	}
	return result;
}
```

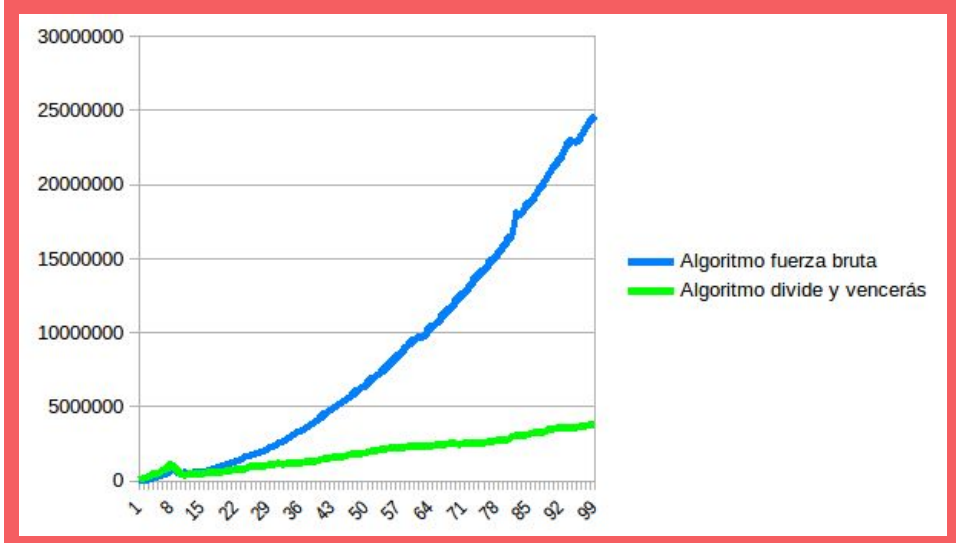
Complejidad

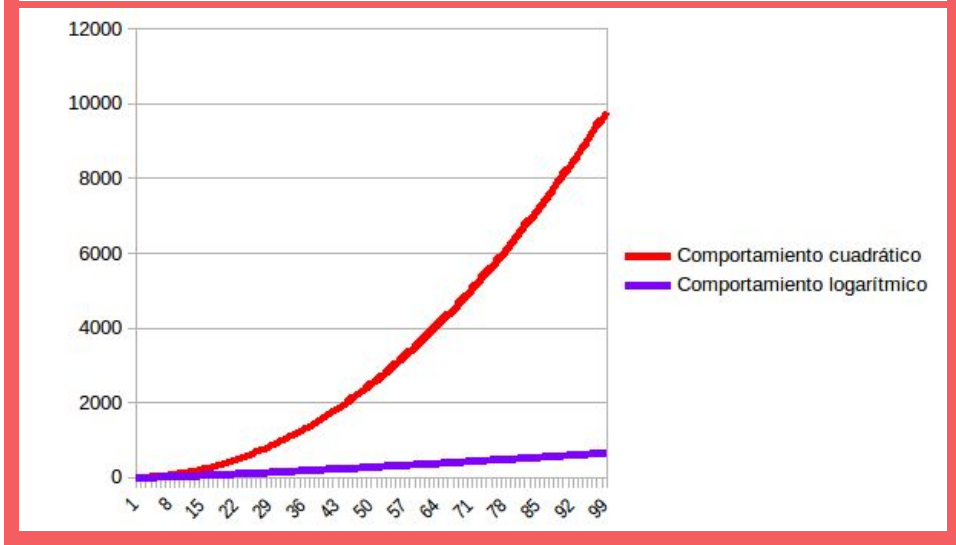
Método no óptimo
FUERZA BRUTA
$O(n^m)$
m = 2 ó 3 según las dimensiones del problema.
n = número de puntos del problema.

```
Closest-Pair (p1, …, pn)
{
   Compute separation line L such that half the points          O(n log n)
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)                                 2T(n / 2)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from separation line L      O(n)

   Sort remaining points by y-coordinate.                       O(n log n)

   Scan points in y-order and compare distance between          O(n)
   each point and next 11 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

Tiempos de Ejecución





Tamaño del Problema	Alg. Fuerza Bruta	Alg. Div y Vencerás	Comportamiento Cuadrático	Comportamiento Logarítmico
10	421677	464150	100	33.2192
19	960821	590863	361	80.7106
29	2.21E+06	1.09E+06	841	140.8814
39	4.01E+06	1.35E+06	1521	206.1306
49	6.32E+06	1.84E+06	2401	275.1207
59	9.22E+06	2.29E+06	3481	347.0759
69	1.20E+07	2.51E+06	4761	421.4881
79	1.58E+07	2.77E+06	6241	497.998
89	2.07E+07	3.44E+06	7921	576.340

Conclusiones

- Efectividad
  - Menor número de comparaciones
  - Menor número de iteraciones
  - Particularmente eficaz en resolución del problema en un plano Euclidiano
- Ventajas de uso.
  - Evitar colisión entre puntos
  - Control de elementos en entorno espacial

¿Preguntas?

Bibliografía

https://en.wikipedia.org/wiki/Closest_pair_of_points_problem

http://www.geeksforgeeks.org/closest-pair-of-points/