Práctica 2

Raúl Granados López
Hossam El Amraoui Leghzali
Javier Montaño Rubio

Método Domina: Eficiencia

Función S= Domina(P1=(x0, ..., xn), P2=(x0, ..., xn)) → O(n)

domina = true → O(1)

Para cada coordenada x de los puntos, **hacer:** → O(n)

Si P1.coordenada[x] < P2.coordenada[x], **hacer:** → O(1)

domina = false → O(1)

Devolver domina

O(1)

O(n)

Método Domina: Implementación

```cpp
struct Punto{
    vector<int> coordenadas;

    // Devuelve si se domina a p2
    bool Domina (const Punto & p2) { //O(n)

        bool domina = true;

        for (int i=0; i<coordenadas.size() && domina; i++) { //O(n)
            if (coordenadas[i] < p2.coordenadas[i]) {
                domina = false;
            }
        }

        return domina;
    }
};
```

Método Básico: Eficiencia

Función S = BASICO (P: Conjunto de puntos (x0, x1, ..., xn)) → $O(n^3)$

Para cada punto pi=(x0, ..., xn) en P, **hacer:** → nº veces = n

Para cada punto pj=(x0, ..., xn) en P, **hacer:** → nº veces = n

Si Domina(pi, pj), **hacer:** → $O(n)$

Borrar pj de P → $O(n)$

$O(n)$

$O(n^2)$

$O(n^3)$

Devolver P

Método Básico: Implementación

```
vector<Punto> MetodoBasico (vector<Punto> p, int n){

    for (auto it1 = p.begin(); it1 != p.end(); ++it1){

        vector<Punto>::iterator it2 = p.begin();

        while (it2 != p.end()){

            if ((*it1).Domina(*it2) && it1 != it2){

                p.erase(it2);

                if (it1 > it2)
                    --it1;

            }else
                ++it2;

        }

    }

    return p;

}
```

Divide y Vencerás 1: Eficiencia

Función S= DyV1(P: Conjunto de puntos (x0, x1, ..., xn)) $\rightarrow O(n^3)$

Si |P| =1, **hacer:**

Devolver S= BASICO(P)

En otro caso:

Partir P en dos subconjuntos de igual tamaño P1 y P2

S1= DyV1(P1) $\rightarrow T(n/2)$

S2= DyV1(P2) $\rightarrow T(n/2)$

S= Fusionar1(S1, S2) $\rightarrow O(n^3)$

Devolver S

$$T(n) = 2T(n/2) + n^3$$

$$2T(n/2)+n^3$$

$$n = 2^m; \quad m = \log_2(n)$$

$$\underbrace{T(2^m) = T(2^{m-1})}_{PH} + \underbrace{2^{3m}}_{PNH}$$

$$T(2^m) - T(2^{m-1}) = 0 \qquad 2^{3m} = b^m \cdot q(m) \begin{cases} b = 8 \\ q(m) = 1 \\ d = 0 \end{cases}$$

$$P_H(x) = x - 2$$

$$P(x) = (x-2)\cdot(x-b)^{d+1} = (x-2)(x-8) \begin{cases} r_1 = 2 & M_1 = 1 \\ r_2 = 8 & M_2 = 1 \end{cases}$$

$$t_m = c_{10} \cdot 2^m \cdot m^0 + c_{20} \cdot 8^m \cdot m^0$$

$$t_n = c_{10} \cdot n + c_{20} \cdot 2^{3\log_2(n)} = c_{10} \cdot n + c_{20} \cdot 2^{\log_2(n^3)} = c_{10} \cdot n + c_{20} \cdot n^3 \Rightarrow$$

$$\Rightarrow \boxed{O(n^3)}$$

Divide y Vencerás 1: Implementación

```
vector<Punto> DyV1 (vector<Punto> p, int n) {
    vector<Punto> puntosNoDominados;

    if (n == 1)
        puntosNoDominados.push_back(p[0]);
    else {

        vector<Punto>::const_iterator it = p.cbegin() + n/2;
        vector<Punto> p1(p.cbegin(), it);
        vector<Punto> p2(it, p.cend());

        vector<Punto> temp1 = DyV1(p1, n/2);
        vector<Punto> temp2 = DyV1(p2, n - n/2);

        Fusiona1 (puntosNoDominados, temp1, temp2);

    }

    return puntosNoDominados;
}
```

```
void Fusiona1 (vector<Punto> & puntosNoDominados, const vector<Punto> & p1, const vector<Punto> & p2) {

    vector<Punto> aux = p1;
    for (int i=0; i < p2.size(); ++i)
        aux.push_back(p2[i]);

    for (int i=0; i<aux.size(); i++) {  //O(n^3)
        bool dominado=false;

        for (int j=0; j<aux.size() && !dominado; j++) {  //O(n^2)
            if (aux[j].Domina(aux[i]) && i!=j) { //O(n)
                dominado = true;
            }
        }

        if (!dominado)
            puntosNoDominados.push_back(aux[i]);

    }

}
```

Divide y Vencerás 2: Eficiencia

Función S= DyV2(P: Conjunto de puntos (x0, x1, ..., xn)) $\rightarrow O(n^3)$

Si |P| =1, **hacer:**

Devolver S= BASICO(P)

En otro caso:

Partir P en dos subconjuntos de igual tamaño P1 y P2

S1= DyV1(P1)

S2= DyV1(P2)

S= Fusionar2(S1, S2)

$T(n)=2T(n/2)+n^3$

Devolver S

Divide y Vencerás 2: Implementación

```cpp
vector<Punto> DyV2 (vector<Punto> p, int n) {
    vector<Punto> puntosNoDominados;

    if (n == 1)
        puntosNoDominados = p;

    else {
        vector<Punto>::const_iterator it = p.cbegin() + n/2;
        vector<Punto> p1(p.cbegin(), it);
        vector<Punto> p2(it, p.cend());

        vector<Punto> temp1 = DyV2(p1, n/2);
        vector<Punto> temp2 = DyV2(p2, n - n/2);

        Fusiona2 (puntosNoDominados, temp1, temp2);
    }

    return puntosNoDominados;
}
```

```cpp
void Fusiona2 (vector<Punto> & puntosNoDominados, const vector<Punto> & p1, const vector<Punto> & p2) {

    puntosNoDominados = p1;
    for (int i=0; i < p2.size(); ++i)
        puntosNoDominados.push_back(p2[i]);

    puntosNoDominados = MetodoBasico (puntosNoDominados, puntosNoDominados.size ());

}
```

Eficiencia general