Module EA4 – Éléments d'Algorithmique II

Outils pour l'analyse des algorithmes

Dominique Poulalhon
dominique.poulalhon@irif.fr

Université de Paris
L2 Informatique & DL Bio-Info, Jap-Info, Math-Info
Année universitaire 2020-2021

dernier TP cette semaine (toujours avec une permanence sur discord jeudi matin et vendredi matin)

dernier TD la semaine prochaine ; groupes INFO1 et INFO2 sur discord (probablement mardi 11 à 8h30)

contrôle n° 3 mercredi prochain (12 mai) sur moodle, de 15h à 16h30

Quelques applications des tris

Applications du tri en géométrie : 1. Calcul de l'enveloppe convexe

enveloppe convexe d'une partie $\mathcal{P}$ du plan : plus petite partie convexe $\mathcal{C}$ contenant $\mathcal{P}$

Applications du tri en géométrie : 1. Calcul de l'enveloppe convexe

enveloppe convexe d'une partie $\mathcal{P}$ du plan : plus petite partie convexe $\mathcal{C}$ contenant $\mathcal{P}$

si $\mathcal{P}$ est un ensemble fini de points (on parle de *nuage* de points), $\mathcal{C}$ est un polygone dont les sommets sont des points du nuage

Applications du tri en géométrie : 1. Calcul de l'enveloppe convexe

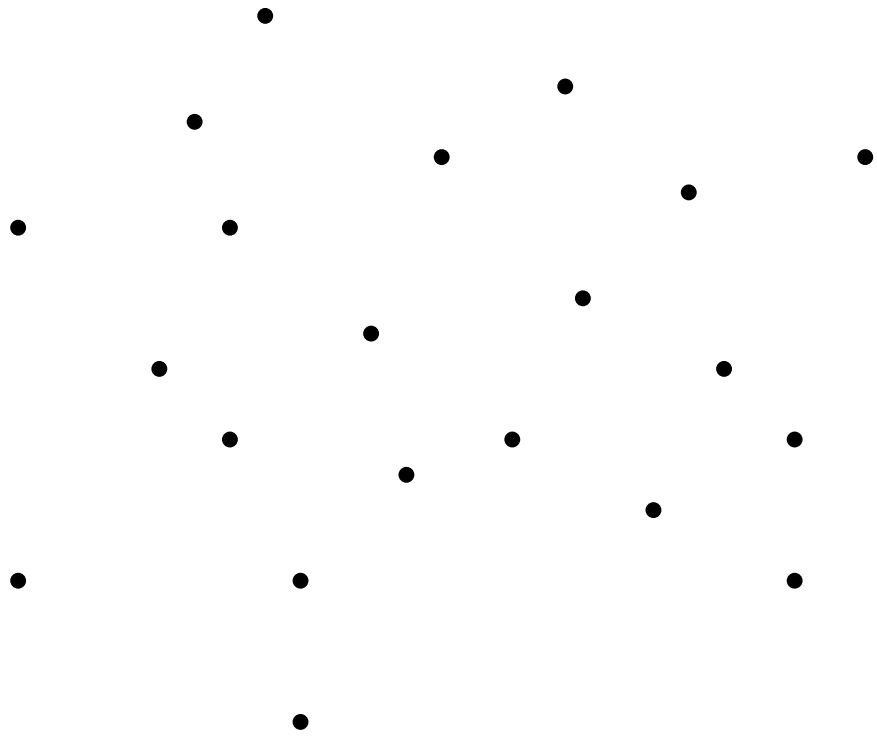
enveloppe convexe d'une partie $\mathcal{P}$ du plan : plus petite partie convexe $\mathcal{C}$ contenant $\mathcal{P}$

si $\mathcal{P}$ est un ensemble fini de points (on parle de *nuage* de points), $\mathcal{C}$ est un polygone dont les sommets sont des points du nuage

enveloppe_convexe(nuage)

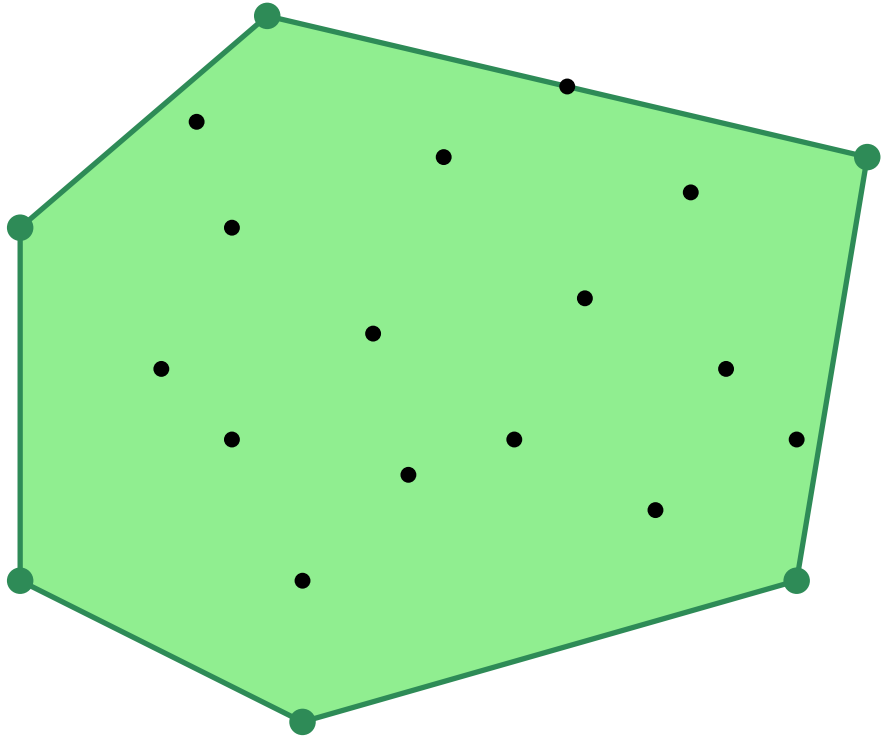
étant donné un `nuage` de points du plan, déterminer l'enveloppe convexe des points du `nuage`

Caractérisation des arêtes de l'enveloppe



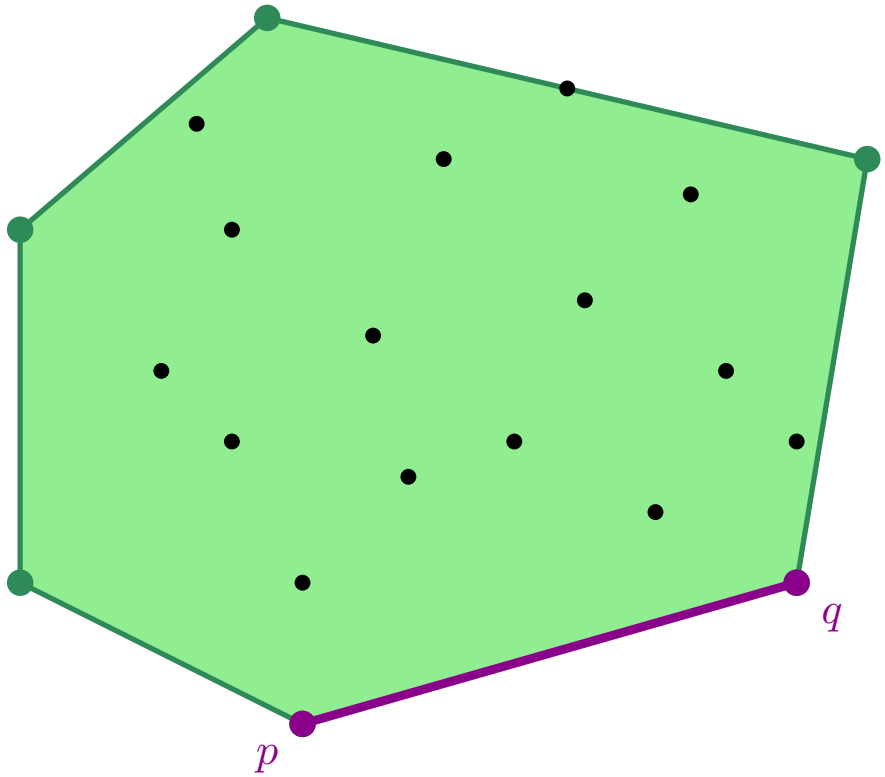
Un nuage de points

Caractérisation des arêtes de l'enveloppe



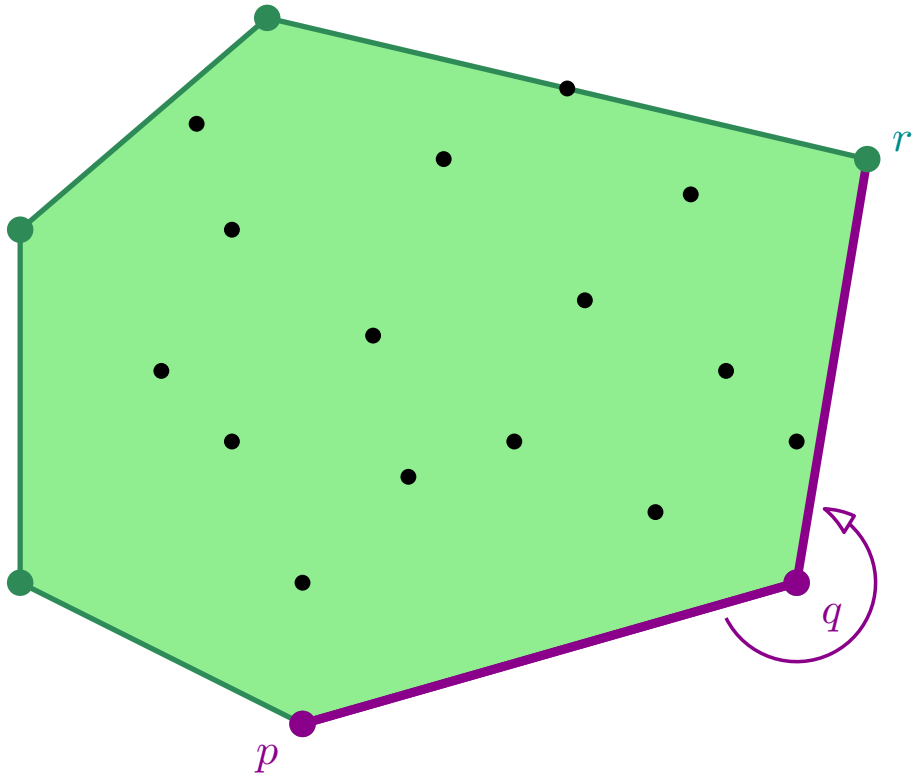
Son enveloppe convexe

Caractérisation des arêtes de l'enveloppe



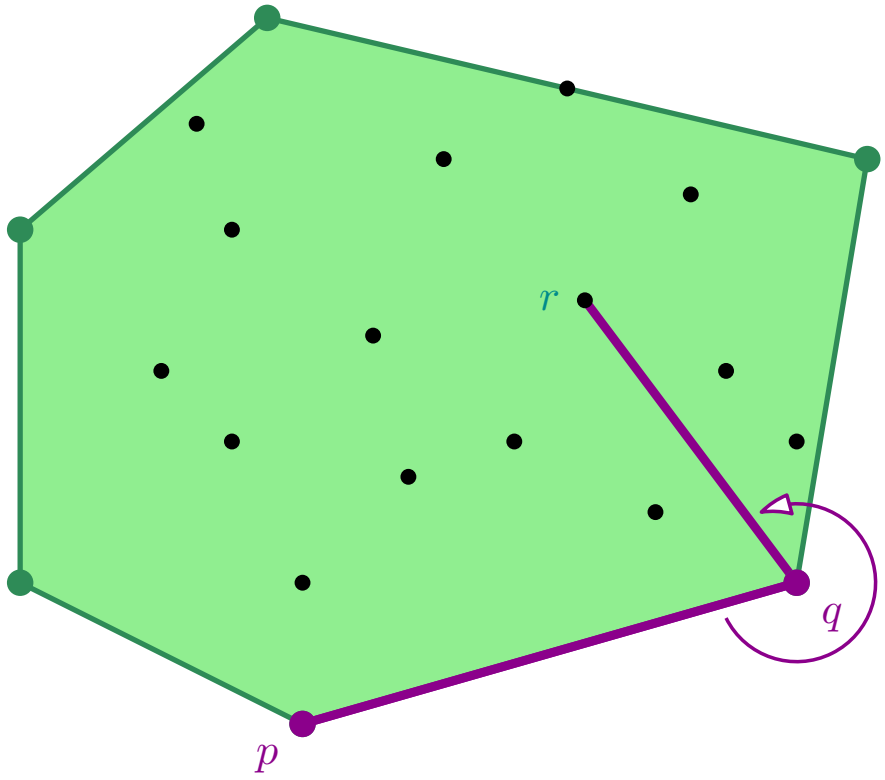
Une arête [pq] de l'enveloppe (dans le sens direct)

Caractérisation des arêtes de l'enveloppe



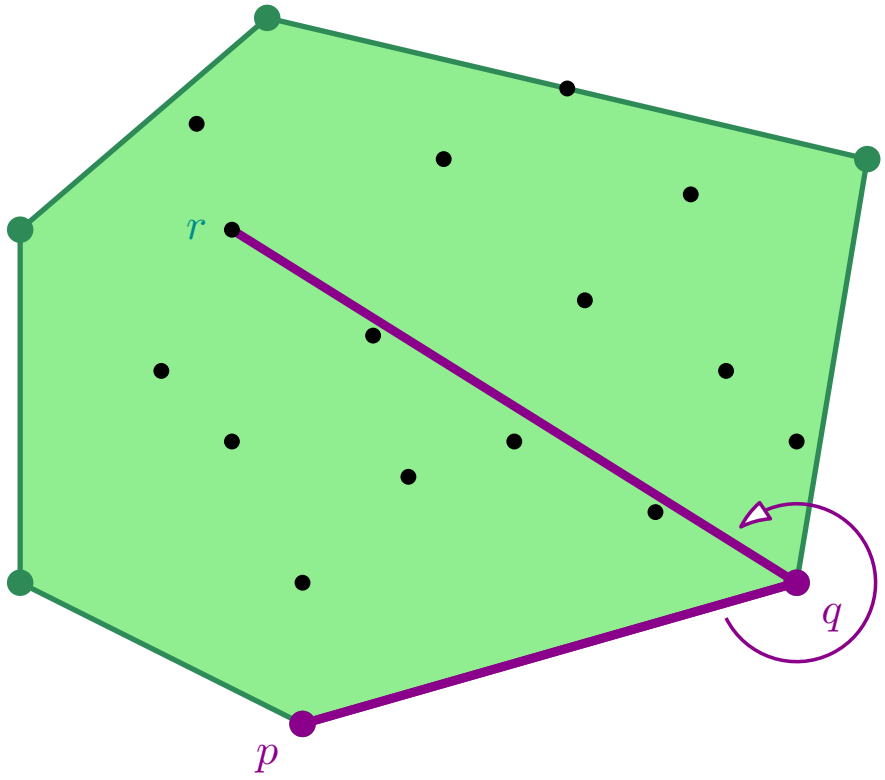
Tous les angles $\widehat{pqr}$ « tournent à gauche »

Caractérisation des arêtes de l'enveloppe



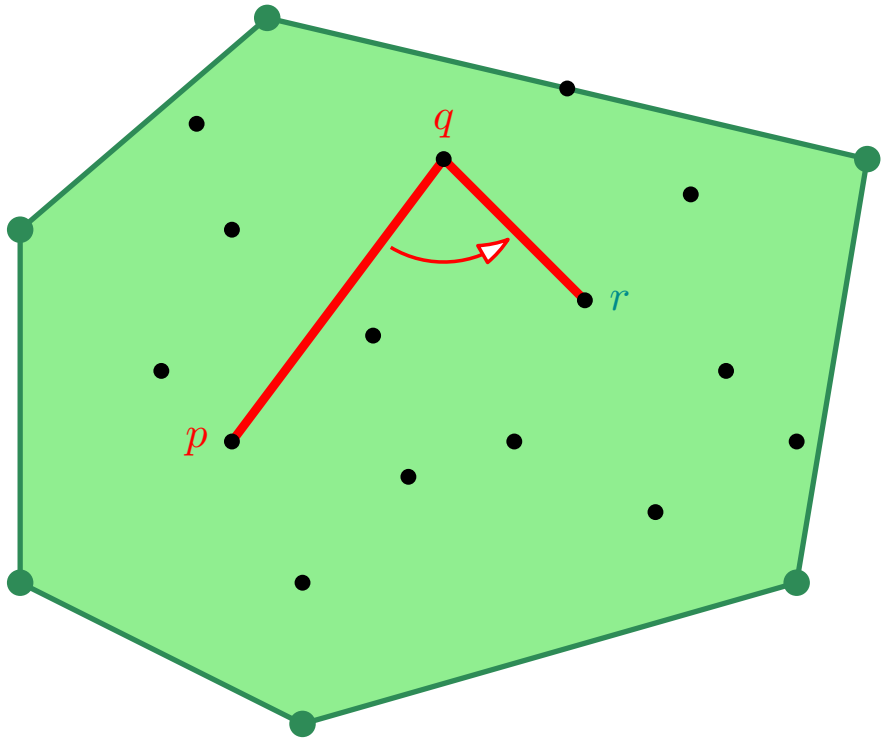
Tous les angles $\widehat{pqr}$ « tournent à gauche »

Caractérisation des arêtes de l'enveloppe



Tous les angles $\widehat{pqr}$ « tournent à gauche »

Caractérisation des arêtes de l'enveloppe



... contrairement au cas où [pq] n'est pas une arête de l'enveloppe

Enveloppe convexe d'un nuage – méthode naïve

```
def enveloppe_convexe_naive(nuage) :
  tous_les_couples =     # tous les couples de points du nuage
          [ (p,q) for p in nuage for q in nuage if p != q ]
  aretes_enveloppe = []
  for (p, q) in tous_les_couples :
    for r in nuage : # r contredit-il la caractérisation pour [pq]?
      if tourne_a_droite(p, q, r) : break
    else : # ie si la boucle termine normalement, [pq] ∈ enveloppe
      aretes_enveloppe += [(p,q)]
  return aretes_enveloppe
```

Enveloppe convexe d'un nuage – méthode naïve

```
def enveloppe_convexe_naive(nuage) :
  tous_les_couples =     # tous les couples de points du nuage
          [ (p,q) for p in nuage for q in nuage if p != q ]
  aretes_enveloppe = []
  for (p, q) in tous_les_couples :
    for r in nuage : # r contredit-il la caractérisation pour [pq]?
      if tourne_a_droite(p, q, r) : break
    else : # ie si la boucle termine normalement, [pq] ∈ enveloppe
      aretes_enveloppe += [(p,q)]
  return aretes_enveloppe
```

Lemme

enveloppe_convexe_naive(nuage) retourne une liste formée des arêtes de l'enveloppe convexe de nuage en temps $\Theta(n^3)$ (au pire et en moyenne)

Enveloppe convexe d'un nuage – par balayage

Idée : pour être plus efficace, il ne faut pas considérer *tous* les couples mais essayer de « tourner » autour du nuage

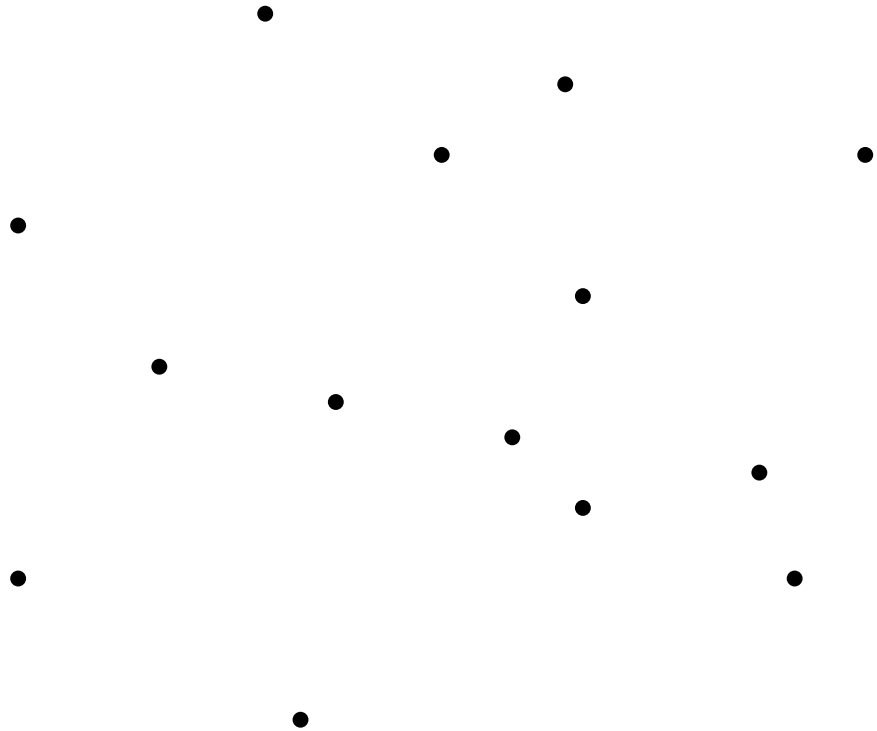
Plus précisément :

- partir d'un point « extrémal » p_0 – par exemple celui d'ordonnée minimale – qui appartient nécessairement à l'enveloppe
- considérer ensuite les points un par un – $p_1, p_2, \ldots p_{n-1}$ pour déterminer si p_i appartient à l'enveloppe convexe du nuage $\{p_0, p_1, \ldots, p_i\}$

Question : dans quel ordre faut-il considérer les points du nuage ?

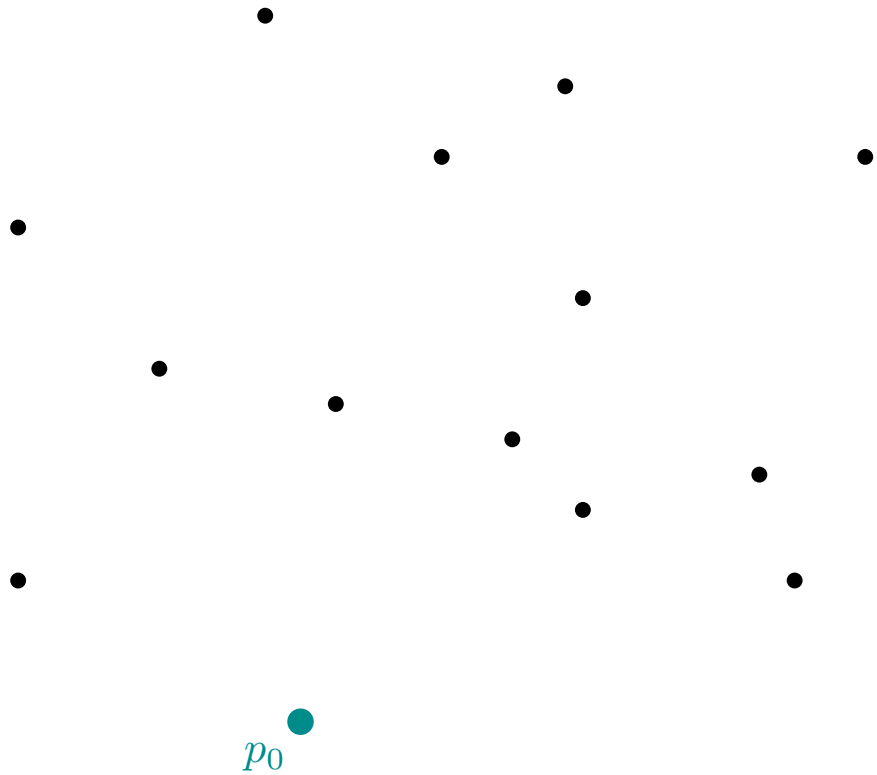
Enveloppe convexe d'un nuage – par balayage

Exemple :



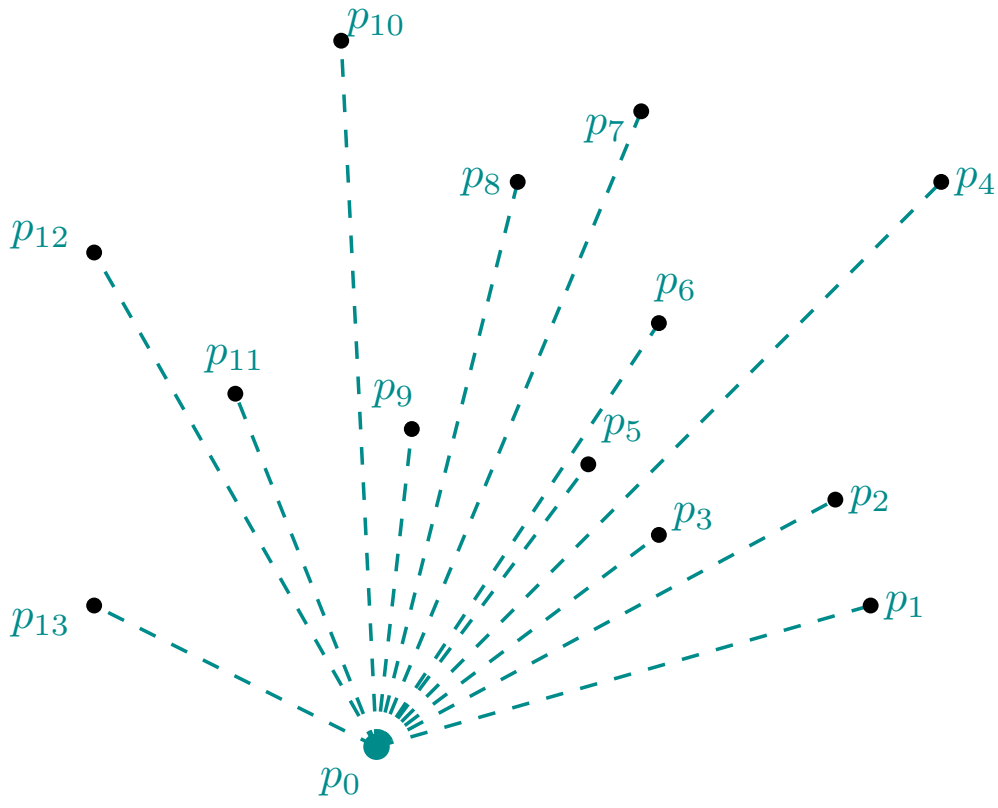
Enveloppe convexe d'un nuage – par balayage

Exemple :



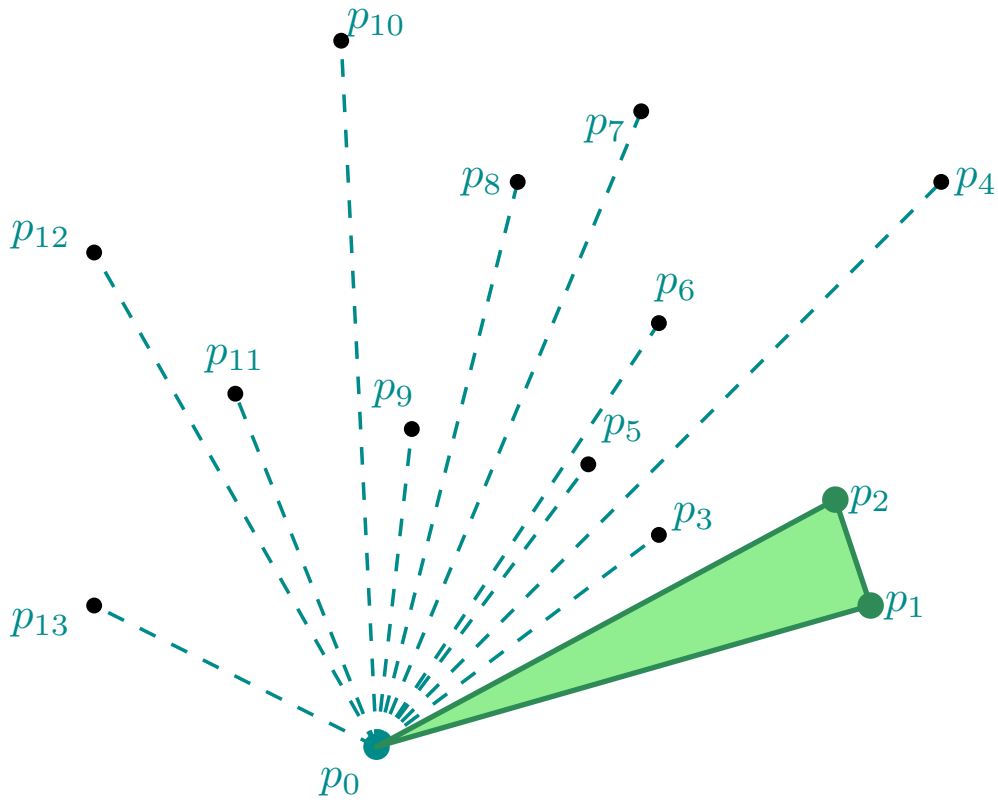
Enveloppe convexe d'un nuage – par balayage

Exemple :



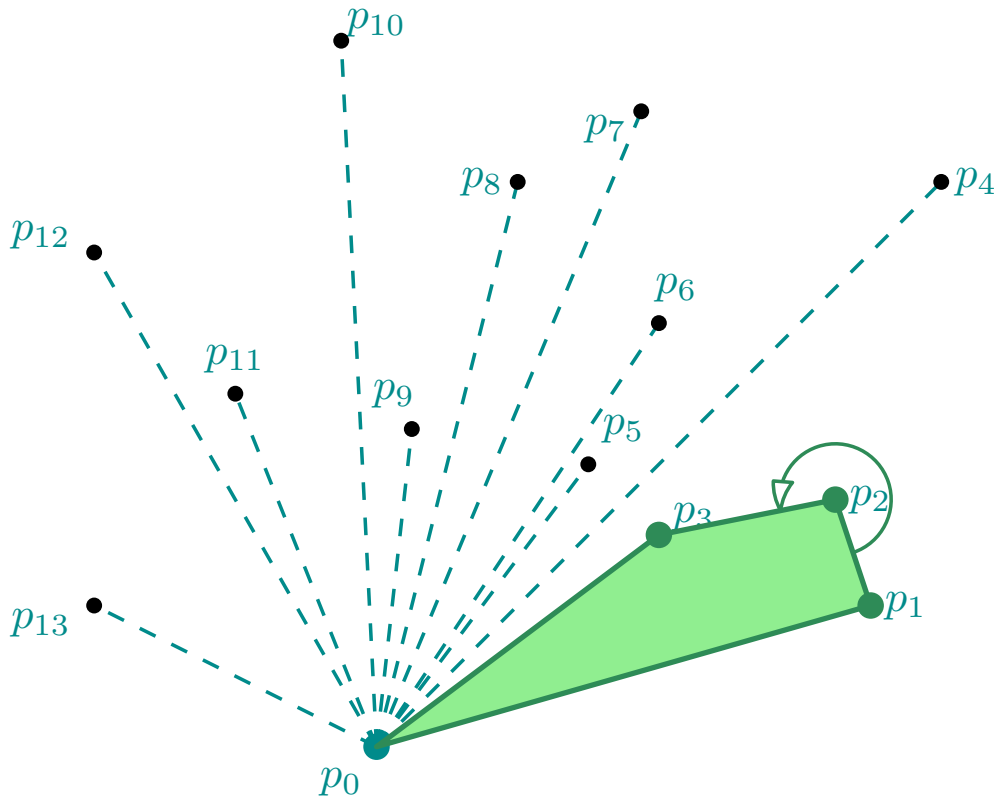
Enveloppe convexe d'un nuage – par balayage

Exemple :



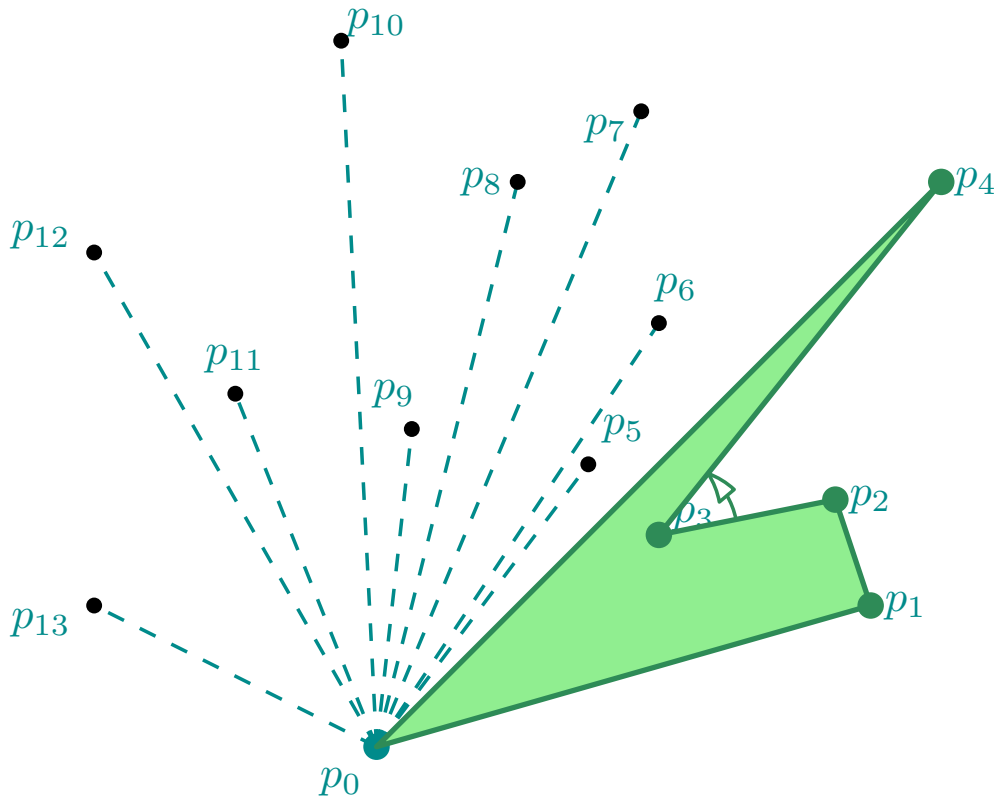
Enveloppe convexe d'un nuage – par balayage

Exemple :



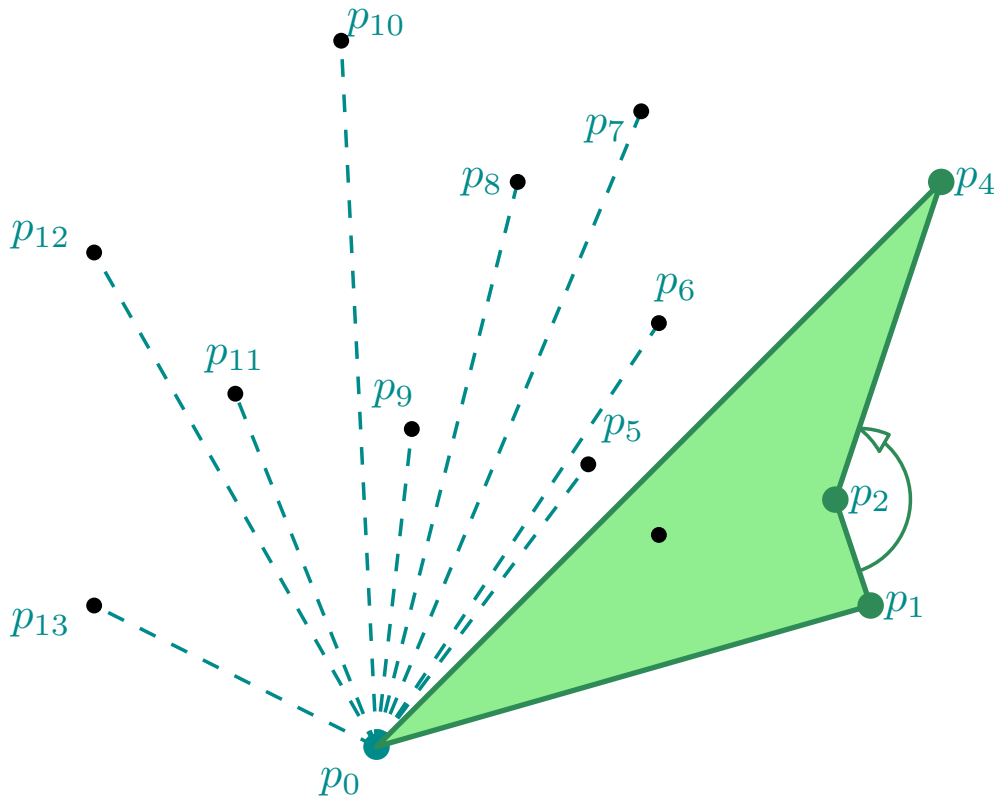
Enveloppe convexe d'un nuage – par balayage

Exemple :



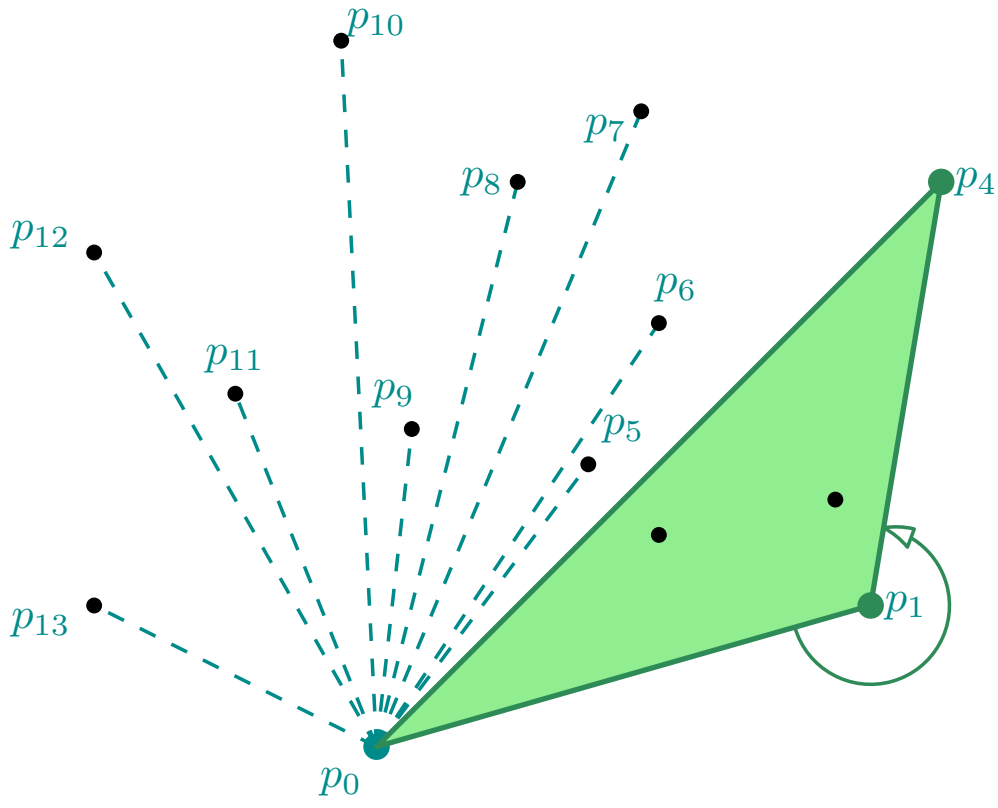
Enveloppe convexe d'un nuage – par balayage

Exemple :



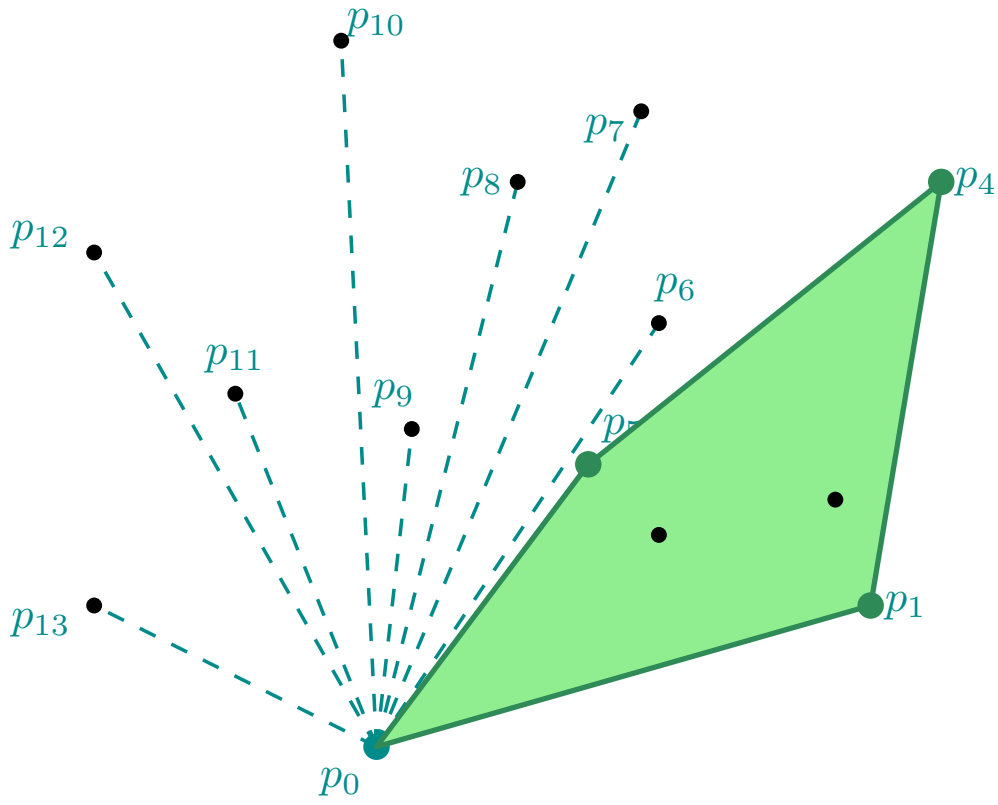
Enveloppe convexe d'un nuage – par balayage

Exemple :



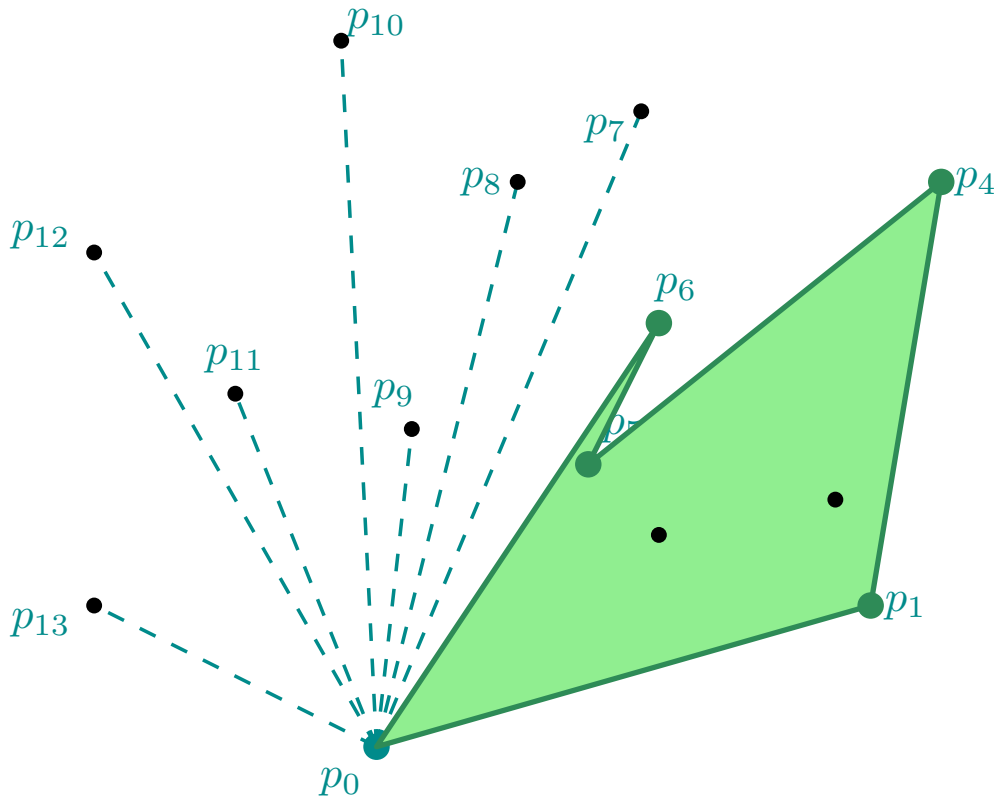
Enveloppe convexe d'un nuage – par balayage

Exemple :



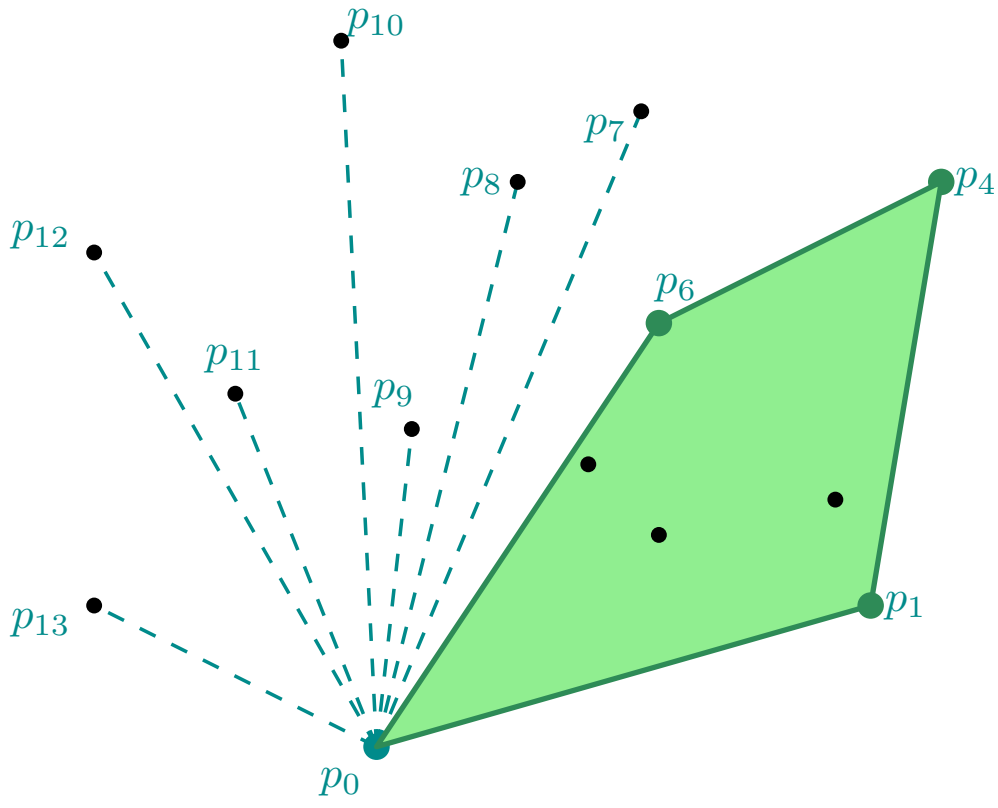
Enveloppe convexe d'un nuage – par balayage

Exemple :



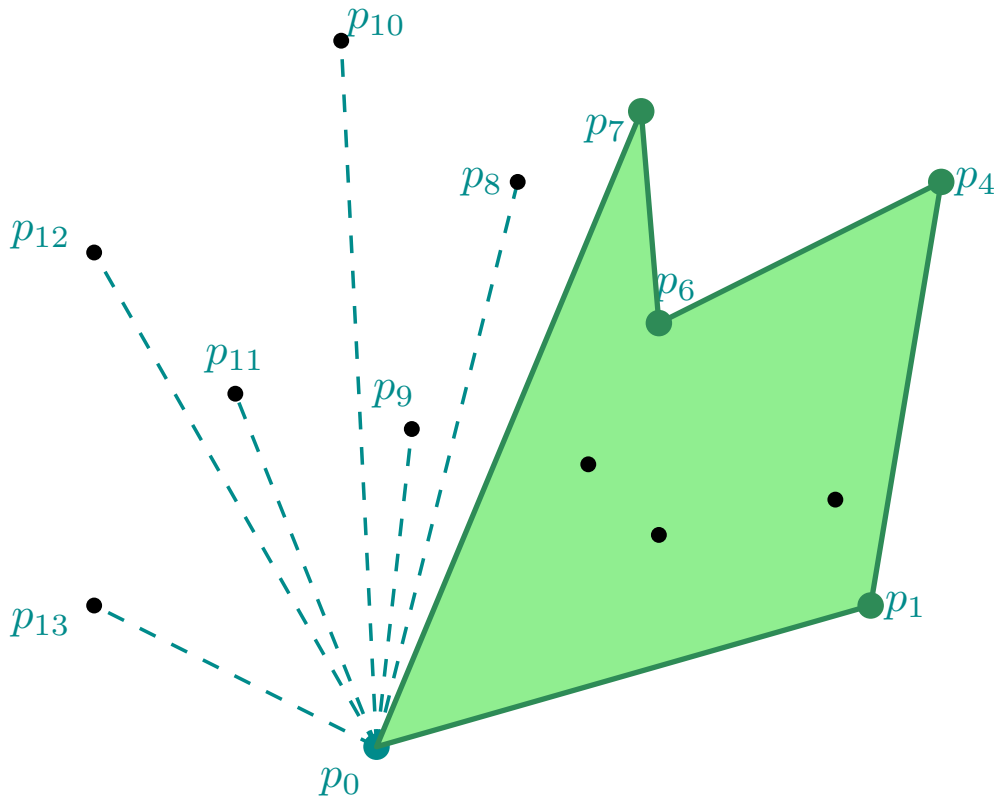
Enveloppe convexe d'un nuage – par balayage

Exemple :



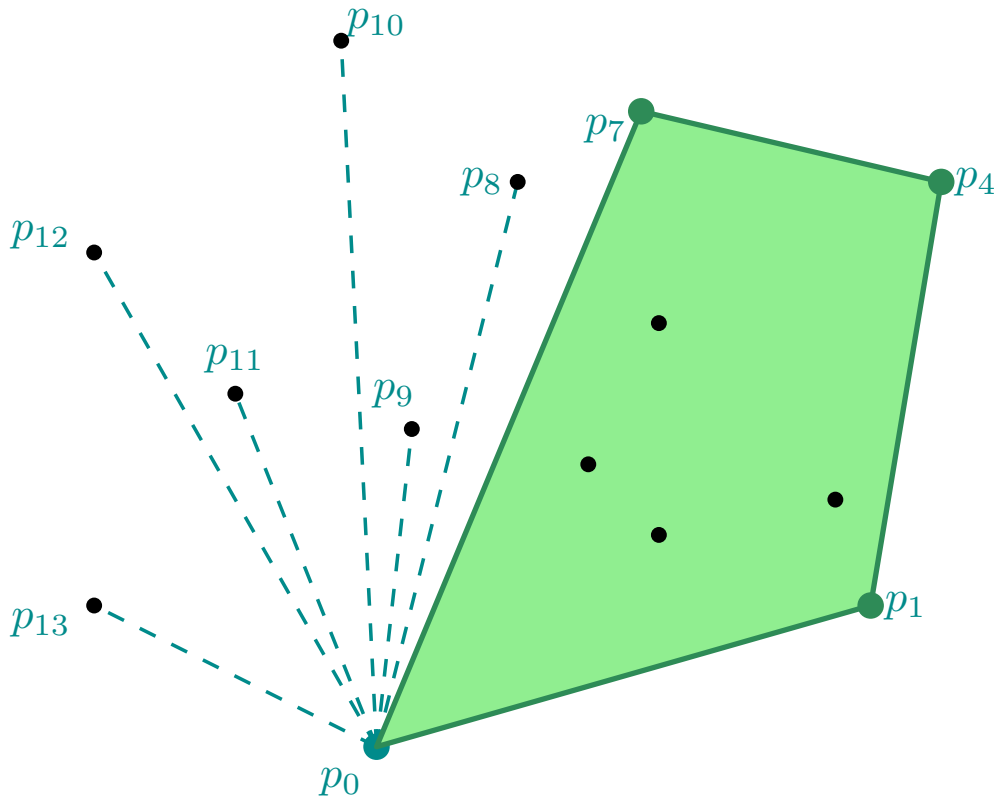
Enveloppe convexe d'un nuage – par balayage

Exemple :



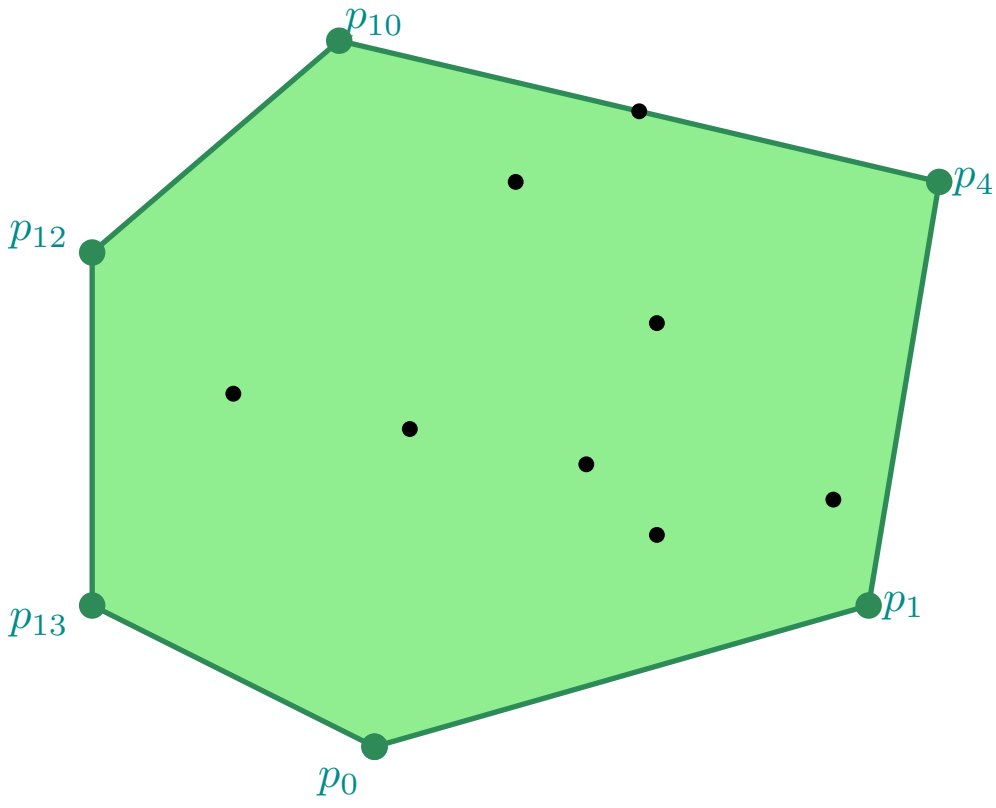
Enveloppe convexe d'un nuage – par balayage

Exemple :



Enveloppe convexe d'un nuage – par balayage

Exemple :



Enveloppe convexe d'un nuage – par balayage

```
def enveloppe_convexe_par_balayage(nuage) :
  p0 = point_le_plus_bas(nuage)
  angles = [ angle_polaire(point, p0) for point in nuage ]
  nuage_trié = trier_selon_angles(nuage, angles)
  pile = [ nuage_trié[0], nuage_trié[1], nuage_trié[2] ]
  for point in nuage_trié :
    while tourne_a_droite(pile[-2], pile[-1], point) :
      pile.pop()
    pile.append(point)
  return pile
```

Enveloppe convexe d'un nuage – par balayage

```
def enveloppe_convexe_par_balayage(nuage) :
  p0 = point_le_plus_bas(nuage)
  angles = [ angle_polaire(point, p0) for point in nuage ]
  nuage_trié = trier_selon_angles(nuage, angles)
  pile = [ nuage_trié[0], nuage_trié[1], nuage_trié[2] ]
  for point in nuage_trié :
    while tourne_a_droite(pile[-2], pile[-1], point) :
      pile.pop()
    pile.append(point)
  return pile

def trier_selon_angles(nuage, angles) :
  # exemple de « decorate-sort-undecorate »
  return [ point
       for (angle, point) in sorted(zip(angles, nuage)) ]
```

Théorème

`enveloppe_convexe_par_balayage(nuage)` produit la liste des sommets de l'enveloppe convexe en temps $\Theta(n \log n)$

Démonstration

- point le plus bas : c'est juste un `min` $\Longrightarrow$ $\Theta(n)$
- tri selon l'angle : $\Theta(n \log n)$
- double boucle : $\Theta(n)$ car chacun des n points est, au pire, sorti une fois de la pile

Applications du tri en géométrie : 2. Points les plus proches

points_les_plus_proches(nuage)

étant donné un nuage de points du plan, déterminer les deux points du nuage les plus proches l'un de l'autre

Applications du tri en géométrie : 2. Points les plus proches

points_les_plus_proches(nuage)

étant donné un nuage de points du plan, déterminer les deux points du nuage les plus proches l'un de l'autre

problème presque équivalent :

distance_minimale(nuage)

étant donné un nuage de points du plan, déterminer la distance minimale entre deux points du nuage

Cette distance minimale est appelée *maille* du nuage de points

distance_minimale(nuage)

étant donné un nuage de points du plan, déterminer la distance minimale entre deux éléments du nuage

```
def distance_minimale_naive(nuage) :
  toutes_les_distances =
     [ distance(p,q) for p in nuage for q in nuage if p != q ]
  return min(toutes_les_distances)
```

distance_minimale(nuage)

étant donné un nuage de points du plan, déterminer la distance minimale entre deux éléments du nuage

```
def distance_minimale_naive(nuage) :
  toutes_les_distances =
     [ distance(p,q) for p in nuage for q in nuage if p != q ]
  return min(toutes_les_distances)
```

Lemme

distance_minimale_naive(nuage) calcule la distance minimale entre deux points du nuage en temps $\Theta(n^2)$

distance_minimale(nuage)

étant donné un `nuage` de points du plan, déterminer la distance minimale entre deux éléments du `nuage`

Approche ***« diviser pour régner »*** :

- découper le problème en sous-problèmes de taille inférieure
- résoudre ***récursivement*** le ou les sous-problèmes
- résoudre le problème initial à l'aide des résultats des sous-problèmes

distance_minimale(nuage)

étant donné un `nuage` de points du plan, déterminer la distance minimale entre deux éléments du `nuage`

Approche ***« diviser pour régner »*** :

- séparer `nuage` en deux sous-listes `gauche` et `droite`
- résoudre ***récursivement*** le ou les sous-problèmes
- résoudre le problème initial à l'aide des résultats des sous-problèmes

distance_minimale(nuage)

étant donné un `nuage` de points du plan, déterminer la distance minimale entre deux éléments du `nuage`

Approche ***« diviser pour régner »*** :

- séparer `nuage` en deux sous-listes `gauche` et `droite`
- calculer `d1 = distance_minimale(gauche)`
  et `d2 = distance_minimale(droite)`
- résoudre le problème initial à l'aide des résultats des sous-problèmes

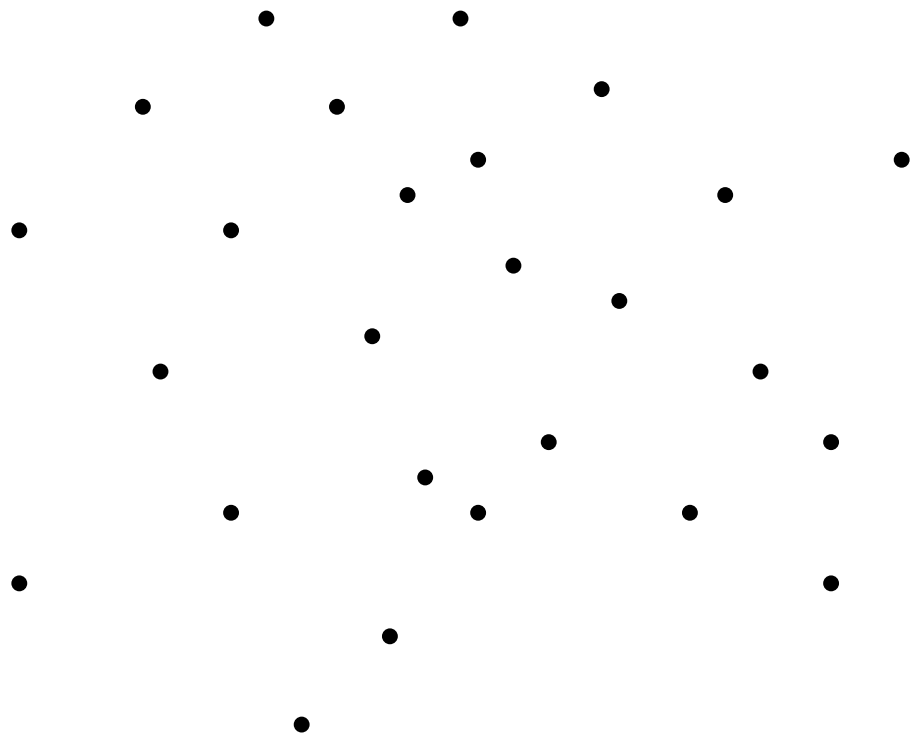
distance_minimale(nuage)

étant donné un `nuage` de points du plan, déterminer la distance minimale entre deux éléments du `nuage`

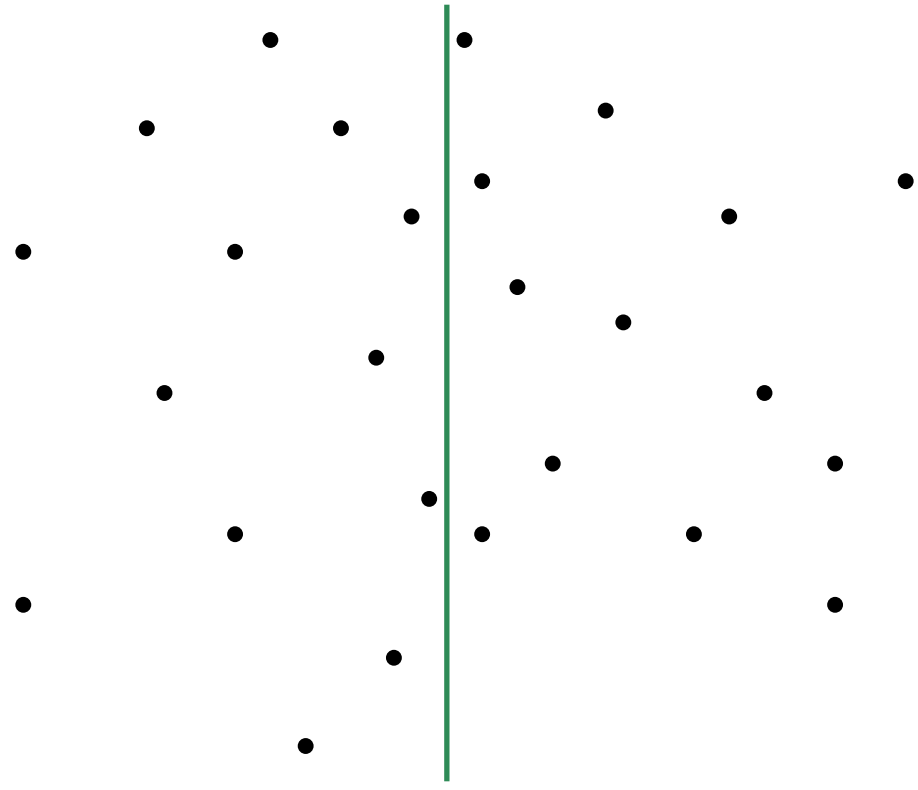
Approche « ***diviser pour régner*** » :

- séparer `nuage` en deux sous-listes `gauche` et `droite`
- calculer `d1 = distance_minimale(gauche)`
  et `d2 = distance_minimale(droite)`
- chercher s'il existe `p1` dans `gauche` et `p2` dans `droite` plus proches que `min(d1, d2)`

Points les plus proches – *« diviser pour régner »*

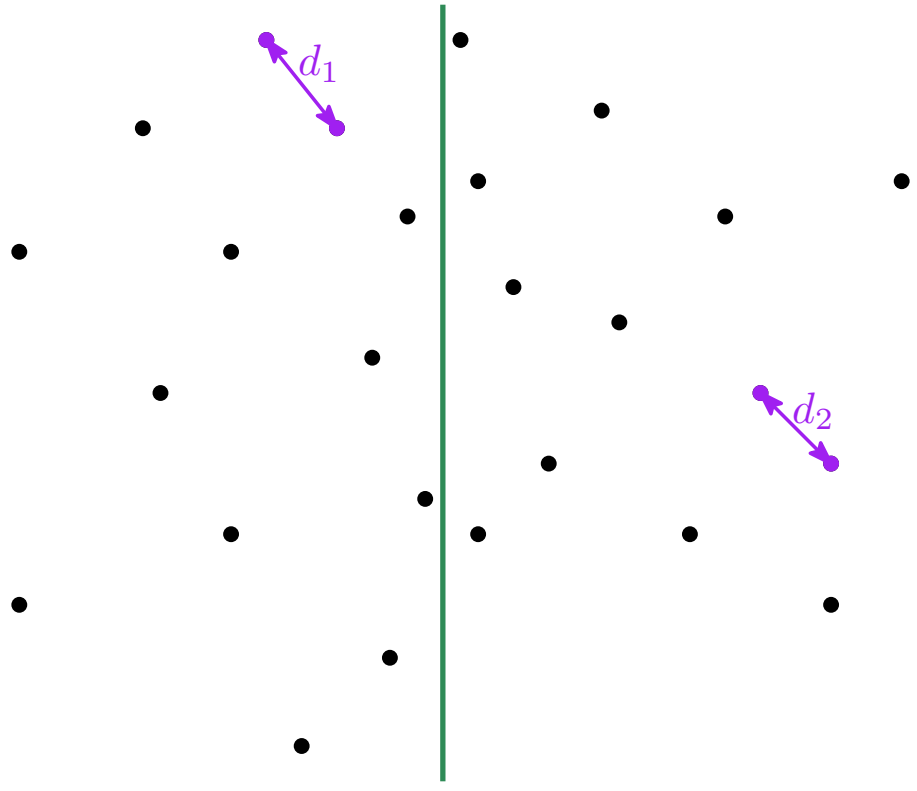


Points les plus proches – *« diviser pour régner »*



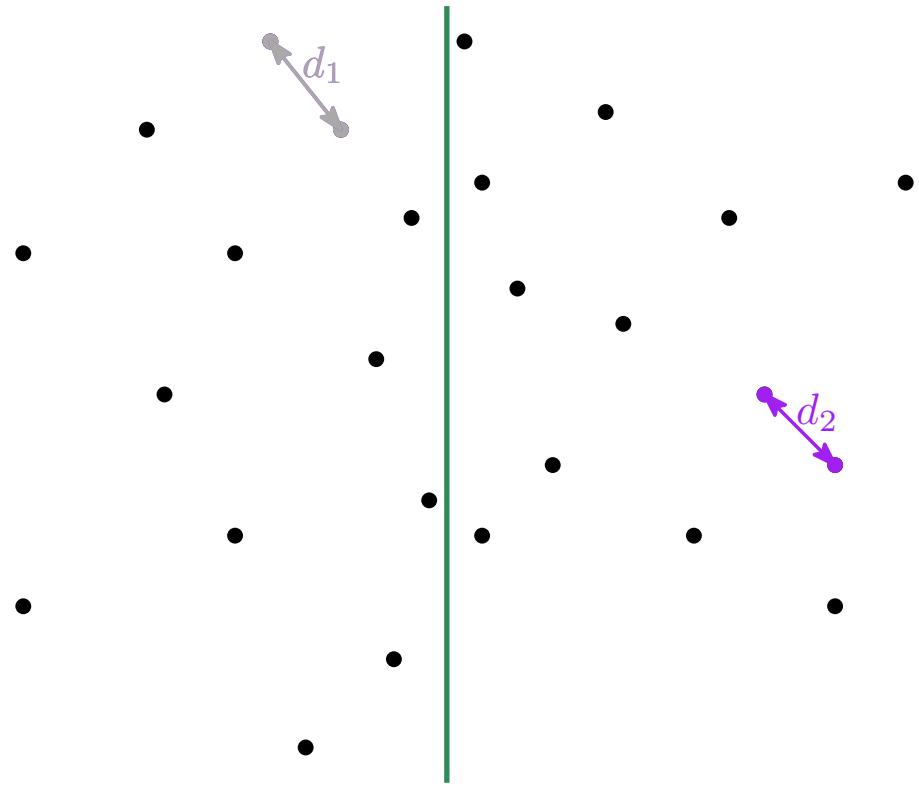
Partitionnement `gauche` – `droite`

Points les plus proches – *« diviser pour régner »*

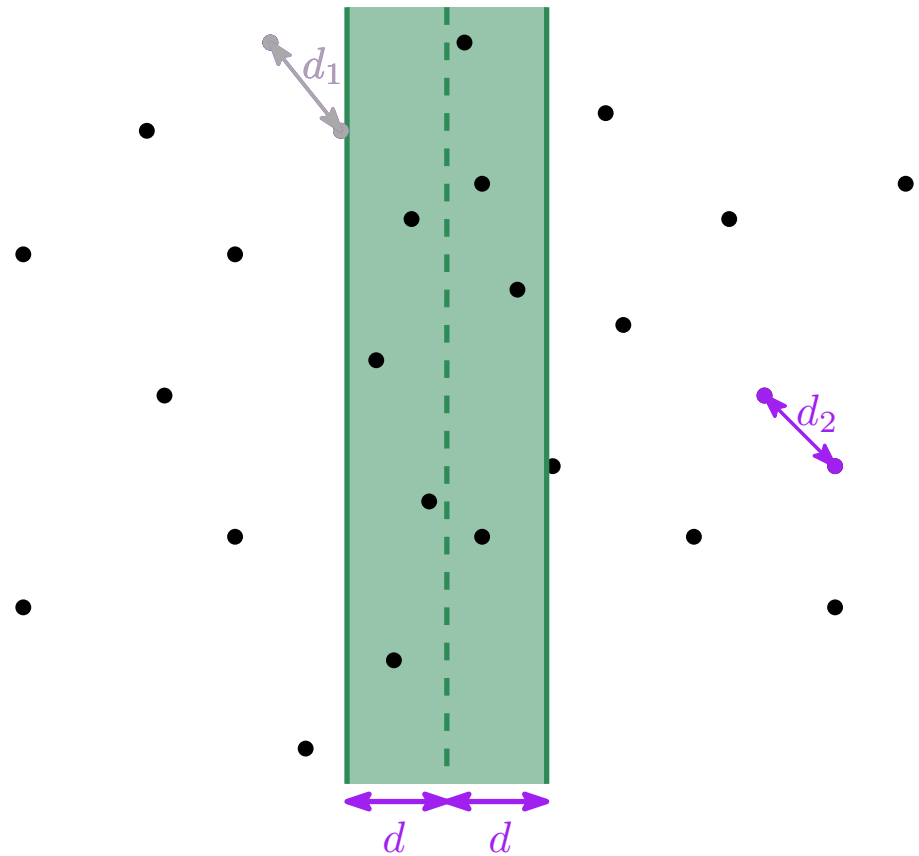


Appels récursifs sur `gauche` et `droite`

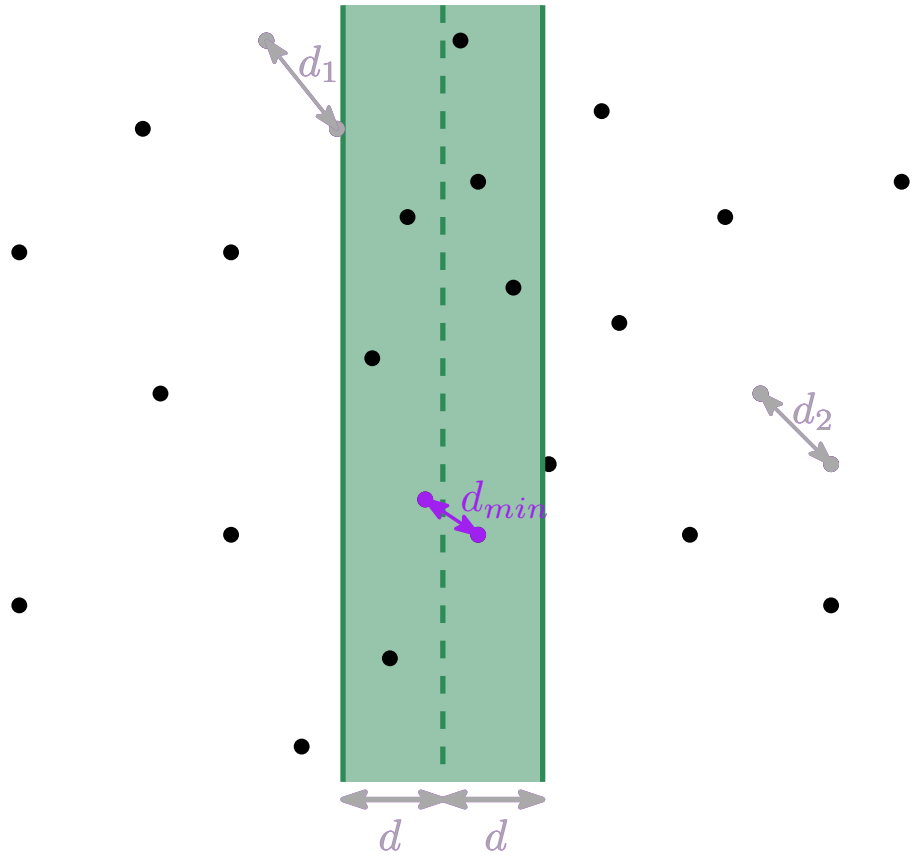
Points les plus proches – *« diviser pour régner »*



Calcul de $\mathrm{d} = \min(\mathrm{d1}, \mathrm{d2})$

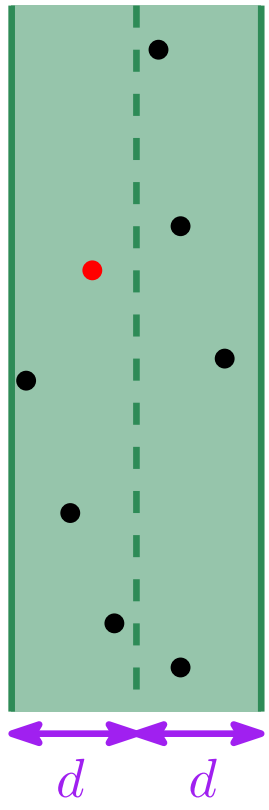


Extraction de la bande médiane de largeur 2d

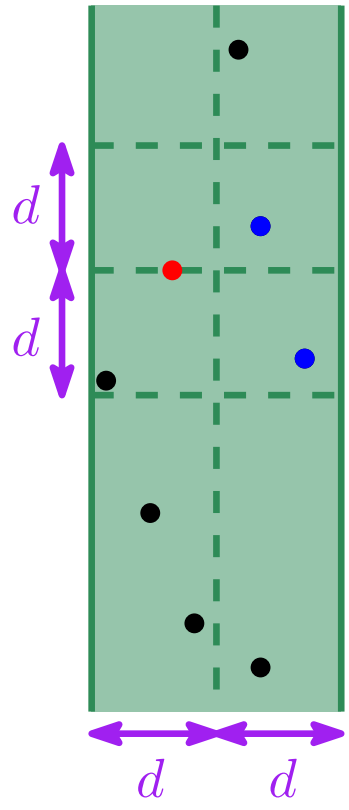


Recherche dans la bande médiane

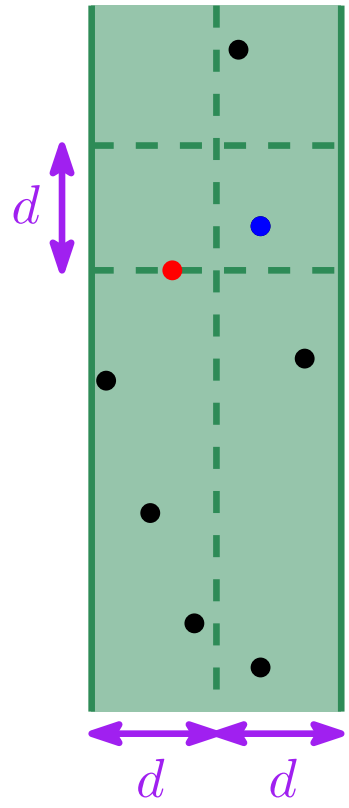
Comment trouver (`p1`, `p2`) efficacement ?



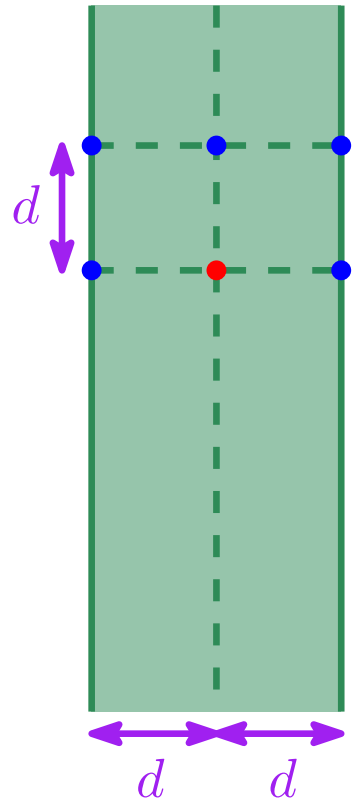
Comment trouver (`p1`, `p2`) efficacement ?



Comment trouver (`p1`, `p2`) efficacement ?



Comment trouver (`p1`, `p2`) efficacement ?



Comment optimiser l'algorithme ?

Pour le partitionnement gauche–droite

Trier ***une fois pour toutes*** la liste des points selon les abscisses

$\Longrightarrow$ étant donné `L_x`, le partitionnement a un coût constant

Comment optimiser l'algorithme ?

Pour le partitionnement gauche–droite

Trier ***une fois pour toutes*** la liste des points selon les abscisses

$\Longrightarrow$ étant donné `L_x`, le partitionnement a un coût constant

Pour la recherche des couples (p1, p2)

Trier ***une fois pour toutes*** la liste des points selon les ordonnées

$\Longrightarrow$ étant donné `L_y`, la recherche a un coût linéaire

Comment optimiser l'algorithme ?

Pour le partitionnement gauche–droite

Trier ***une fois pour toutes*** la liste des points selon les abscisses
$\Longrightarrow$ étant donné `L_x`, le partitionnement a un coût constant

Pour la recherche des couples (p1, p2)

Trier ***une fois pour toutes*** la liste des points selon les ordonnées
$\Longrightarrow$ étant donné `L_y`, la recherche a un coût linéaire

$$C_{totale}(n) = C_{tris}(n) + C_{rec}(n) = \Theta(n \log n) + C_{rec}(n)$$

$$C_{rec}(n) = 2C_{rec}\left(\frac{n}{2}\right) + O(n)$$

$$\Longrightarrow C_{totale}(n) \in \Theta(n \log n)$$