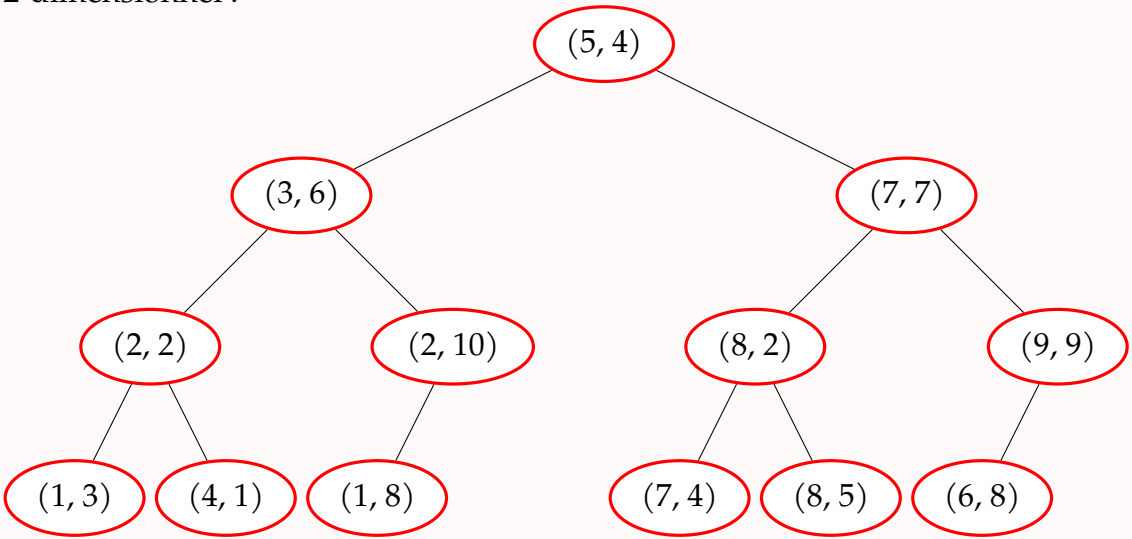
LYCÉE LECONTE DE LISLE

Arbres k-dimensionnels

Vincent Picard

1
Principe

Il est plus facile de représenter la situation pour $k = 2$. Voici un exemple d'arbre 2-dimensionnel :

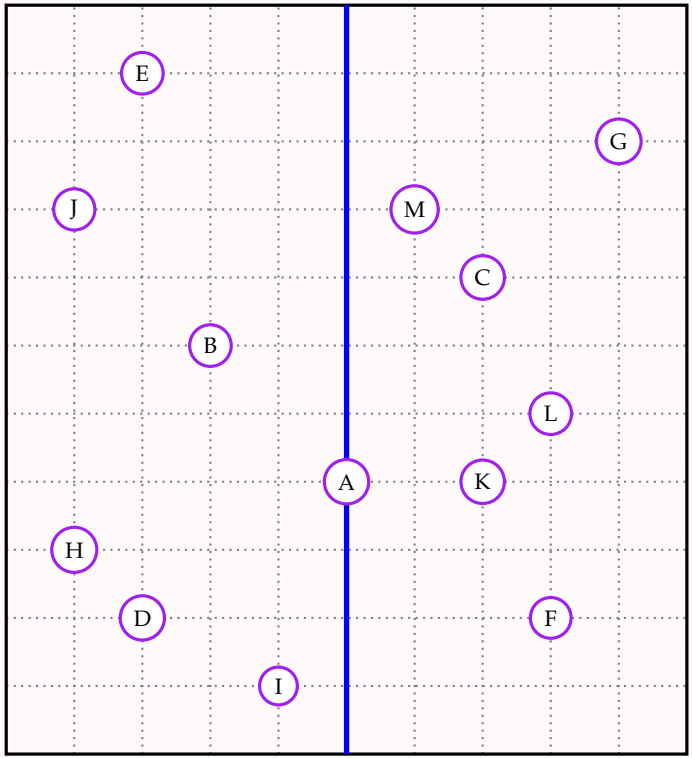


Cet arbre sert à conserver un ensemble de points du plan.

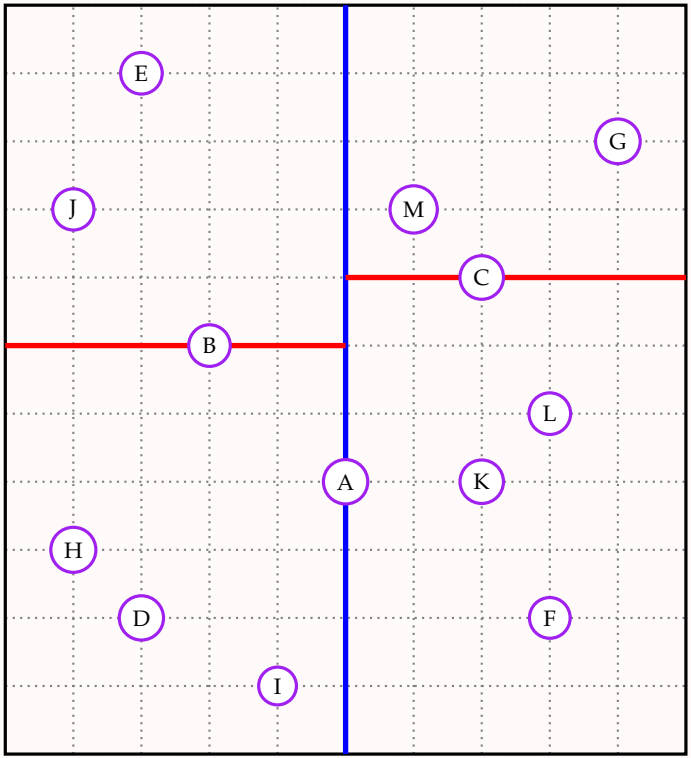
C'est une **généralisation en dimension 2 de l'arbre binaire de recherche** :

- Tous les points à gauche d'un point de profondeur paire a une abscisse plus petite.
- Tous les points à gauche d'un point de profondeur impaire a une ordonnée plus petite.

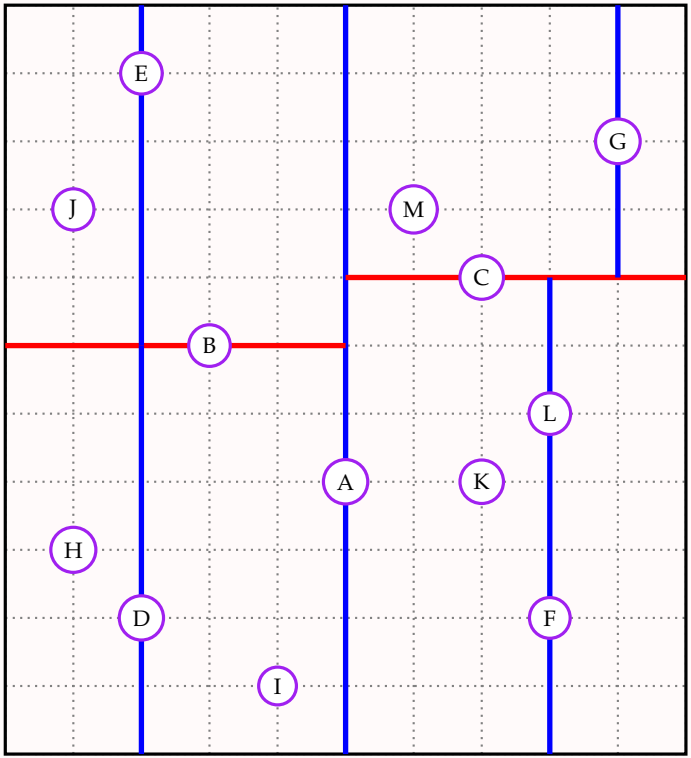
Chaque nœud (= point) divise l'espace restant en 2, soit **verticalement** (profondeurs paires), soit **horizontalement** (profondeurs impaires).



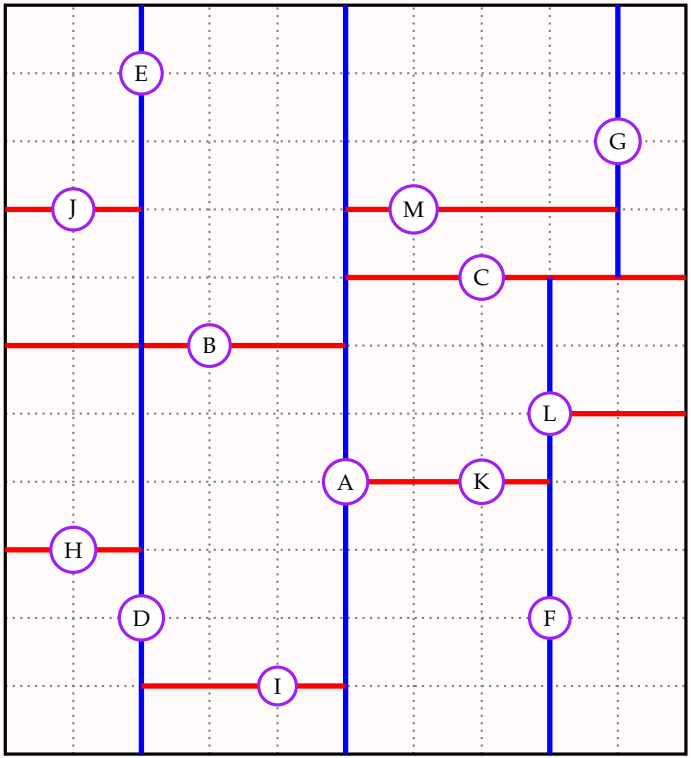
Chaque nœud (= point) divise l'espace restant en 2, soit **verticalement** (profondeurs paires), soit **horizontalement** (profondeurs impaires).



Chaque nœud (= point) divise l'espace restant en 2, soit **verticalement** (profondeurs paires), soit **horizontalement** (profondeurs impaires).



Chaque nœud (= point) divise l'espace restant en 2, soit **verticalement** (profondeurs paires), soit **horizontalement** (profondeurs impaires).



2
Quelques fonctions classiques

Codage des points en OCaml

```
type point = int array;;

let distance p q = (* attention : c'est la distance au carré ! *)
    let k = Array.length p in
    let somme = ref 0 in
    for i = 0 to k-1 do
        somme := !somme + (p.(i) - q.(i)) * (p.(i) - q.(i))
    done;
    !somme
;;
(* renvoie le point le plus proche de q *)
let best q x y_opt = match y_opt with
    | None -> x
    | Some py ->
            if distance q x <= distance q py then
                x
            else
                py
;;
```

Recherche dans un kd-tree

On procède comme avec les arbres binaires de recherche :

```
(* teste si le point q est dans l'arbre t *)
let existe q t =
    let k = Array.length q in
    let rec aux prof t = match t with
    | Vide -> false
    | Noeud (p, _, _) when p = q -> true
    | Noeud (p, g, d) ->
            if q.(prof mod k) < p.(prof mod k) then
                aux (prof + 1) g
            else
                aux (prof + 1) d
    in aux 0 t
;;
```

Insertion dans un kd-tree

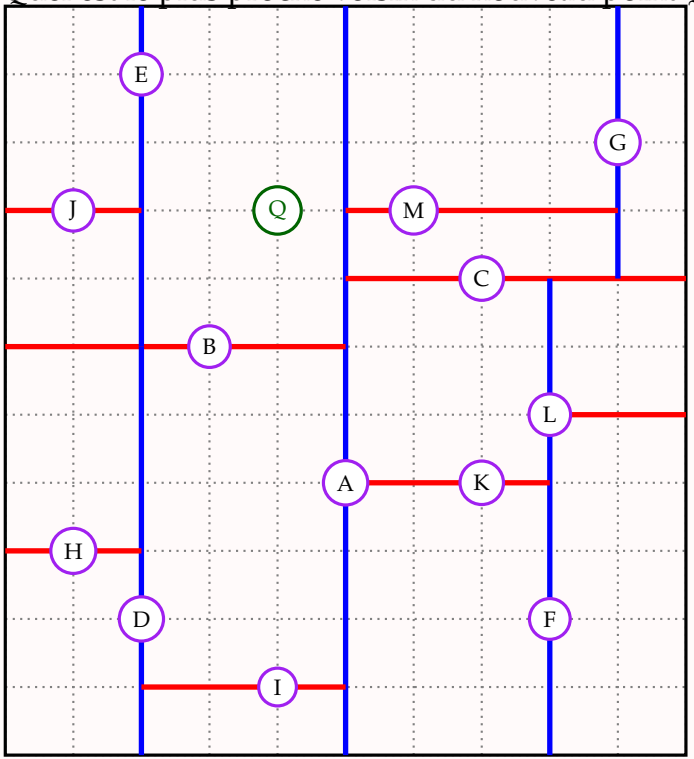
On procède comme avec les arbres binaires de recherche :

```
let insere q t =
    let k = Array.length q in
    let rec aux prof t = match t with
    | Vide -> Noeud(q, Vide, Vide)
    | Noeud (p, g, d) ->
            if q.(prof mod k) < p.(prof mod k) then
                Noeud (p, aux (prof + 1) g, d)
            else
                Noeud (p, g, aux (prof + 1) d)
    in aux 0 t
;;
```

3
Recherche du plus proche voisin

Recherche du plus proche voisin

Quel est le plus proche voisin du nouveau point $Q = (4, 8)$?



Algorithme

On procède de manière récursive :

- On cherche du côté de l'hyperplan où le point Q se trouve (fils gauche ou droite).
- On garde le meilleur point β, entre la racine et le point trouvé.
- On teste si la boule de centre Q et de rayon $Q\beta$, touche l'hyperplan.
  - Si la boule touche ou traverse : il faut verifier dans l'autre demi-espace.
  - Sinon, il ne sert à rien de regarder dans l'autre demi-espace.

Code OCaml

```
let carre x = x * x;;

(* Renvoie le point (option) le plus proche de q.
Renvoie None si l'arbre est vide *)
let plus_proche q t =
    let k = Array.length q in
    let rec aux prof t = match t with
    | Vide -> None
    | Noeud (p, g, d) ->
            if q.(prof mod k) < p.(prof mod k) then
                let pp_gauche = aux (prof + 1) g in
                let b = best q p pp_gauche in
                if distance b q < carre (q.(prof mod k) - p.(prof mod k)) then
                    (Some b)
                else
                    let pp_droite = aux (prof + 1) d in
                    Some (best q b pp_droite)
```

```
            else (* cas symétrique *)
                let pp_droite = aux (prof + 1) d in
                let b = best q p pp_droite in
                if distance b q < carre (q.(prof mod k) - p.(prof mod k)) then
                    (Some b)
                else
                    let pp_gauche = aux (prof + 1) g in
                    Some (best q b pp_gauche)
    in aux 0 t
;;
```