Exercice 2 : nuage de points et quadtree

08 juin

On considère un ensemble de points du plan représentés par le type suivant :
`type point = {x : float; y : float};;`.

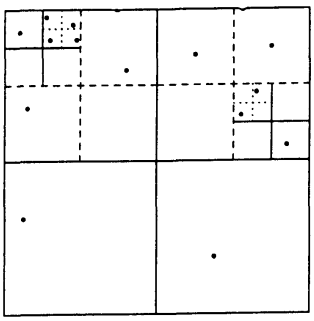
On vous fournit les fonctions suivantes :

1. Les fonctions `sum : point -> point -> point` et `sub : point ->point -> point` qui prennent en entrée deux points de coordonnées (x, y) et (x', y') et qui calculent respectivement $(x + x', y + y')$ et $(x - x', y - y')$.
2. La fonction `mult_scal : float -> point -> point` qui prend en entrées un réel λ et un vecteur (x, y) et qui calcule $(\lambda x, \lambda y)$.

Un nuage de points sera représenté par une cellule carrée subdivisée en quatre sous-cellules. Chaque cellule qui contient deux points ou plus est subdivisée en quatre sous-cellules carrées de même taille, la subdivision s'arrêtant lorsqu'une cellule contient un ou aucun point.

Une représentation structurée de la répartition des points est un arbre d'arité quatre dont les noeuds internes ont quatre fils et dont les feuilles contiennent un ou zéro point.

La figure ci-dessous montre un exemple de division en cellules et l'arbre associé. On adoptera la convention selon laquelle, la cellule en haut à droite est numérotée 0, celle en bas à droite 1, celle en bas à gauche 2 et celle en haut à gauche 3.



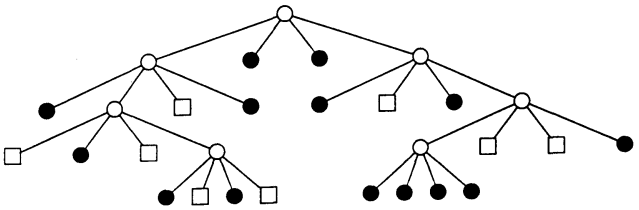


Figure 2

(Dans la figure 2 ci-dessus, les cellules vides sont des carrés et les cellules qui ne contiennent qu'un corps sont des cercles pleins.)

1. Dessiner la division en cellules de deux nuages de points de profondeur 2 qui contiennent respectivement le plus grand nombre de points et le moins grand nombre de points.

   On utilise le type arbre suivant :
   ```
   type arbre = Noeud of filles | Feuille of point | Vide
   and filles = arbre array;;
   ```
   On suppose que la case zéro du tableau `filles` correspond à la cellule en haut à droite, la case 1 en bas à droite, la case 2 en bas à gauche et la case 3 en haut à gauche.

2. Ecrire une fonction `indice_fille : point -> point -> int` qui prend en entrée un point `p` et le centre `p_c` d'une cellule C qui contient le point `p`, cette fonction renvoie alors l'indice de la sous-cellule de C qui contient le point `p`.

3. Ecrire une fonction `position_fille : point -> float -> int -> point` qui prend en entrées un point `p_c` et une taille `d` qui représentent la cellule C centrée sur le point `p_c` et de côté de longueur `d` ainsi qu'un indice `i` compris entre 0 et 3 et qui renvoie le centre de la $i$ème sous-cellule de C.
4. Ecrire une fonction `insere : point -> arbre -> point -> float -> arbre` qui prend en entrée un point `p` à insérer, un arbre dans lequel insérer ce point ainsi que le centre et la taille de la cellule représentée par l'arbre. La fonction renvoie un arbre qui contient le nouveau point en plus de ceux qu'il contenait déjà. L'arbre renvoyé est un quadtree.

   On vous fournit une fonction `min_dist_to_rect : point -> point -> float -> float` telle que `min_dist_to_rect q c half` calcule la distance minimale entre un point `q` et un carré centré en `c` avec demi-côté `half` au sens de la distance euclidienne.

5. Ecrire une fonction `plus_proche_voisin : point -> arbre -> point -> float -> point` qui prend en entrée un point `p`, un arbre ainsi que le centre et la taille de la cellule représentée par cet arbre. La fonction renvoie un point contenu dans l'arbre qui est le plus proche du point `p` au sens de la distance euclidienne.
6. Expliquer comment adapter la solution de la question précédente pour obtenir k plus proches voisins de `p` au lieu d'un seul.