Devoir maison n°9

À faire pour le lundi 10/02

⋆ ⋆ ⋆

Pour ce devoir, vous devrez rendre un fichier de code source, contenant les fonctions et les tests que vous aurez écrit. Il est demandé que le fichier source puisse se compiler correctement. Tout code erroné encore présent dans le fichier devra être commenté. Vous nommerez votre fichier au format `DM9_NOM_Prenom.ml`, par exemple `DM9_CARRÉ_Nathaniel.ml`.

Le code source devra inclure des commentaires de code expliquant le fonctionnement des différentes parties du code, et, lorsque c'est pertinent, des invariants de boucle assurant la correction des fonctions. Il devra également contenir des tests des différentes fonctions écrites.

Dans ce devoir, on cherche à mettre en œuvre des arbres k-dimensionnels, et à mesurer les performances, tant au niveau de la création que de la recherche de plus proche voisin. On suppose que les données manipulées sont des éléments de $\mathbb{R}^k$.

On représente un point de $\mathbb{R}^k$ en OCaml par un tableau de flottants de taille k :

```
type point = float array
```

Lorsqu'on manipule un tableau de points, on supposera sans le préciser systématiquement que les points ont la même dimension.

Pour un élément $(x_0, x_1, \ldots, x_{k-1}) \in \mathbb{R}^k$, on dira que l'**attribut** a est x_a.

1 Création d'arbre k-dimensionnel

On définit le type d'arbre suivant :

```
type tree = Nil | Node of tree * int * tree
```

Pour un tableau de données `data` de taille n, un arbre k-dimensionnel est la donnée d'un arbre `t` de type `tree` de taille n, contenant les indices entre 0 et $n-1$, et d'une copie `d` de `data` où les éléments ont été permutés de telle sorte que pour tout nœud interne `Node(l, i, r)` de `t`, de profondeur p :

- les indices de `l` sont inférieurs ou égaux à i et les indices de `r` sont supérieurs ou égaux à i ;
- les valeurs de l'attribut $p \mod k$ des éléments d'indices de `l` sont inférieures ou égales à la valeur de l'attribut $p \mod k$ de `data.(i)` qui est strictement inférieure aux valeurs de l'attribut $p \mod k$ des éléments d'indices de `r`.

Autrement dit, on dispose d'un arbre k-d sur les indices des éléments, qui sont placés à la bonne position dans la copie des données.

Lorsqu'on demande de modifier un tableau de données pour la suite, on s'assurera que seules des permutations ont lieu, et que toutes les données sont encore présentes dans le tableau.

Question 1 Écrire une fonction `swap : 'a array -> int -> int -> unit` qui prend en argument un tableau `tab` et deux indices i et j et échange les éléments `tab.(i)` et `tab.(j)` dans le tableau.

1.1 Partition aléatoire

On rappelle que `Random.int n` renvoie un entier choisi aléatoirement et uniformément dans $[\![0, n-1]\!]$. Si le fichier est compilé, on pourra utiliser la commande `Random.self_init ()` pour initialiser la graine.

Question 2 Écrire une fonction `partition : point array -> int -> int -> int -> int` telle que si `data` est un tableau de points de taille n, `lo` et `hi` sont des indices tels que $0 \leqslant$ `lo` $<$ `hi` $\leqslant n$ et `a` un entier compris entre 0 et $k-1$, alors `partition data lo hi a` modifie le tableau `data` entre les indices `lo` (inclus) et `hi` (exclu) tel que :

– un point **pivot** p est choisi aléatoirement et uniformément parmi les points d'indices entre `lo` et `hi`. Notons i_p l'indice où il se trouve après modification ;
– pour $i \in [\![\texttt{deb}, i_p]\!]$, l'attribut a de `data.(i)` est inférieur ou égal à l'attribut a de p ;
– pour $i \in [\![i_p + 1, \texttt{fin} - 1]\!]$, l'attribut a de `data.(i)` est strictement supérieur à l'attribut a de p.

La fonction renvoie l'indice i_p où a été placé le pivot.

Question 3 Écrire une fonction `kd_tree_aux : point array -> int -> int -> int -> tree` telle que si `data` est un tableau de points de taille n, `lo` et `hi` sont des indices tels que $0 \leqslant$ `lo` $<$ `hi` $\leqslant n$ et `a` un entier compris entre 0 et $k-1$, alors `kd_tree_aux data lo hi a` modifie le tableau `data` entre les indices `lo` (inclus) et `hi` (exclu) et renvoie un sous-arbre k-d contenant les indices entre `lo` et `hi`. On choisira la racine de chaque nœud aléatoirement.

Question 4 En déduire une fonction `kd_tree : point array -> point array * tree` qui construit un arbre k-d pour l'ensemble d'un tableau de données, et renvoie une copie des données et l'arbre ainsi construit.

1.2 Création de données

Pour tester les différentes fonctions, on travaille dans l'hypercube $[0, 100]^k$. On rappelle que `Random.float x` renvoie un flottant choisi aléatoirement et uniformément entre 0 et x.

Question 5 Écrire une fonction `create_point : int -> point` qui prend en argument un entier k et crée un point aux coordonnées aléatoires dans $[0, 100]^k$.

Question 6 Écrire une fonction `create_data : int -> int -> point array` qui prend en argument deux entiers n et k et crée un tableau de n données dans $[0, 100]^k$.

1.3 Analyse de l'arbre

Question 7 Écrire une fonction `height : tree -> int` qui calcule la hauteur d'un arbre.

Question 8 Écrire une fonction `infix : tree -> int list` qui calcule la liste des étiquettes d'un arbre, dans l'ordre d'un parcours en profondeur infixe. La fonction devra avoir une complexité linéaire en la taille de l'arbre.

Question 9 Pour $k = 3$ et $n = 100\,000$, vérifier que l'arbre k-d obtenu a un parcours infixe croissant. Quelle est la hauteur de l'arbre obtenu ?

2 Recherche de plus proche voisin

On se limite dans ce devoir à la recherche **DU** plus proche voisin. En particulier, on n'utilisera pas de file de priorité pour le parcours de l'arbre.

Question 10 Écrire une fonction `delta : point -> point -> float` qui calcule la distance euclidienne entre deux points.

Question 11 Écrire une fonction `nearest_neighbor : point array -> tree -> point -> int * int` telle que si `(data, t)` forment un arbre k-d et si `x` est un point, alors `nearest_neighbor data t x` renvoie un couple (i, m) tel que `data.(i)` est le point du tableau le plus proche de x, et m est le nombre de nœuds de l'arbre k-d qui ont été parcourus pour trouver i.

Question 12 Pour $n = 100\,000$, quel est le nombre moyen de nœuds parcourus pour trouver le plus proche voisin d'un point aléatoire, pour $k = 3$, $k = 10$, $k = 20$? Que remarque-t-on ?

Ce phénomène est appelé « malédiction de la dimension ».

3 Arbre équilibré

Dans cette partie, on souhaite construire un arbre k-d le plus équilibré possible. Pour ce faire, lorsqu'on doit choisir la racine d'un nœud interne, on choisit le nœud dont l'attribut courant est médian parmi les données du sous-arbre.

L'algorithme de sélection rapide, qui ressemble à l'algorithme de tri rapide, consiste à partitionner le tableau et à recommencer récursivement dans une des deux moitiés obtenues, jusqu'à ce que l'élément d'un indice i donné soit bien placé, les éléments plus petits que i étant avant et ceux plus grand que i étant après. Lorsqu'on veut partitionner selon la médiane, on choisit i comme étant le milieu de l'intervalle courant [1].

Question 13 Reprendre la construction de l'arbre k-d pour suivre ce principe. On pourra réutiliser la fonction `partition`.

Question 14 Comparer la hauteur des arbres obtenus par la méthode précédente et par cette méthode. Comparer le nombre moyen de nœuds explorés lors de la recherche du plus proche voisin. Le gain est-il significatif ?

⋆ ⋆ ⋆

1. À noter : on n'obtient techniquement pas la médiane, mais le deuxième quartile.