# Compte rendu de projet
# Initiation à la Recherche

MARGUERITE Alain
RINCE Romain

Université de Nantes
2 rue de la Houssinière, BP92208, F-44322 Nantes cedex 03, FRANCE


# Table des matières

# 1 Étude des structures de données

## 1 Introduction

L'objectif de ce document est de mener une études sur les différentes structures de données nécessaires aux futurs algorithmes de visualisation. Les calculs de compléxité seront réalisés selon le paramètre $n$ représentant le nombre de boîtes, et $d$ est le nombre de dimensions du problème. Le nombre de boîtes réelement utilisée dans le pavage est représenté par $N$.**Vérifier pour N**

## 2 Définition de la boîte

C'est l'entité du pavage. Les accès à ses attributs sont donc cruciaux. On rappelle qu'une boîte est définie de la manière suivante :

- Un identifiant : Une `String` respectant un format précis (c.f 1.1 du document de spécifications).
- Une liste de coordonnées. Une liste de type `Interval`.
- Une liste des caractéristiques, dans l'ordre et selon les types définis en entête

Ces données seront régulièrement requises durant les algorithmes nécessaires à la visualisation. Il est donc important que leurs accès soient rapides, voir direct. Pour le cas de l'identifiant, s'agissant d'une simple `string` le problème de la structure à utiliser ne se pose pas. Pour la liste des coordonnées en revanche, il s'agit d'une séquence finie de données. Plusieurs possibilités sont alors envisageables :

- Un tableau : L'accès à une coordonnée serait direct. Les opérations d'ajout et de suppression sont revanches coûteuses. Or il est peu probable que ce type d'opération intervienne.
- Une liste : Si l'accès à une coordonnée ne serait pas direct, les opérations d'ajouts de de suppressions sont en temps constant. Ces caractéristiques ne conviennent pas à notre problème.
- Une hashmap : Coûteuse si la fonction de hashage n'est pas appropriée, la HashMap propose cependant un accès direct. Cependant même pour un petit nombre d'éléments, son occupation mémoire peut être conséquente.

- Une TreeMap [Tre]. Implémentation de base des arbres rouges noirs, cette map a la particularité de posséder des clefs triées. Ainsi les complexités de plusieurs opérations telle que l'accès en $\log n$. Dans le cas d'un grand nombre de valeurs à stocker, il est plus performant de construire la TreeMap à partir d'une HashMap.

La création d'une boîte a alors une complexité en $O(d)$. De plus l'accès aux différents attributs de la boîte (identifiant, liste des coordonnées) sera en $O(1)$.

### 2.1 Variables et identifiants

Il est nécessaire de faire le lien de manière efficace entre le nom d'une variable, et sa place dans la structure. En effet comment savoir que la valeur `x` se trouve à la position 4 dans la structure de données.

- Selon le document de spécifications, les variables sont données dans un ordre précis, donné dans l'entête du fichier d'entrée. On peut alors utiliser une map «gobale» renseignant la position de chaque variable. Si cette méthode permet de gagner de l'espace mémoire au sein d'une boîte, le nombre d'accès à cette map risque d'être élevé.
- Utiliser une TreeMap (la clef étant le nom de la variable).

### 2.2 Généricité des caractéristiques

Un problème majeur apparaît pour l'instanciation des boîtes. Elles ont en effet chacune un nombre de caractéristiques potentiellement différents. De plus, on rappelle qu'une caractéristique a plusieurs types. Plusieurs solutions sont envisageables :

- Chaque boîte possède trois maps pour ces trois types de caractéristiques. Cette solution à l'inconveinient d'être coûteuse si par exemple une boîte ne possède que des caractéristiques de type `String` et aucunes de types `Number` ou `Interval`.
- Une map unique contenant des `String` stocke les différentes caractéristiques de la boîte. On aura casté les attributs `Number` ou `Interval` en `String`.
- IL MANQUE UN CAS

## 3 Pavage

L'outil de visualisation peut charger un fichier entrée de manière dynamique ou non. Nous nous plaçons ici dans le cadre où cette option de chargement dynamique n'est pas activée.
L'outil va lire donc de manière séquentielle chaque ligne du fichier d'entrée. Le cahier de spécification exige fréquement un listing de boîtes. Par exemple pour afficher la listres des boîtes concernées par un filtre. La structure du pavage doit être en mesure de répondre de manière efficaces à des requêtes d'un séquence d'un nombre de boîtes selon un ordre particulier. La structure du pavage doit être aussi en mesure de répondre efficacement à la structure de visualisation graphique (cf **lien dym vers QuadTree**). On peut d'emblée envisager plusieurs stratégie pour le stockage du pavage :

- Le fichier d'entrée est lu une unique fois. Les méthodes de structures dans laquelle les boîtes sont stockées sont suffisament performante pour répondre à toutes les spécifications. On peut alors se poser la question si il faut
  - Insérer les boîtes dans la structure puis la trier plus tard.
  - Utiliser un tris par insertion.
- Il est possible que refaire des «passes» sur le fichier d'entrée soit une solution. Dans les cas où les opérations de tris et/ou de recherches seraient trop coûteuse pour la structure du pavage (par exemple lors d'une demande de listing de certaines boîtes). On aurait une complexité en pire cas en $O(n)$.

### 3.1 Stockage du pavage

Le nombre potentiellement très grand de boîtes élimine d'emblée la possibilité de choisir une HashMap. En effet même si la fonction de hashage est judicieusement choisie, l'occupation mémoire requise serait bien trop importante. Les listes sont pas appropriées ici. Une complexité en $n^2$ pour un accès à une boîte n'est pas raisonnable. Les arbres ont l'atout de pouvoir stocker et manipuler un grand nombre de d'entités. Les arbres de recherches sont des arborescences ordonnées permettant un accès en $\log(n)$. Dans le cas où la structure serait triée au fur et à mesure de sa construction. Les arbres de recherche proposent de bonnes performances.

**Arbre binaire** Par exemple l'utilisation d'un arbre binaire de recherche pour la création de n boîtes aurait une complexité de $n^2$ en pire cas. En effet il s'agit du cas où les boîtes arriveraient triées selon l'ordre inverse de celui que l'on souhaite. Il faudrait alors effecter $(p-1)$ comparaisons, pour chaque boîte : $\sum_{p=2}^{n}(p-1)$ Soit $O(n) = \frac{1}{2}n^2 - \frac{1}{2}n$. Cependant dans le meilleur des cas cette opération a une complexité en $n \log n$. La création d'un pavage composé de n boîtes à d dimensions aurait alors une complexité égale à : $(n * d) * n \log(n)$ $\triangle$

**Arbres a-b** Il s'agit d'un arbre de recherche avec les propriétés suivantes :

- $a \leq 2$ et $b \leq 2a - 1$ deux entiers
- La racine a au moins 2 fils (sauf si l'arbre ne possède qu'un noeud) et au plus b fils,
- Les feuilles sont de même profondeur,
- les autres nœuds internes ont au moins a et au plus b fils

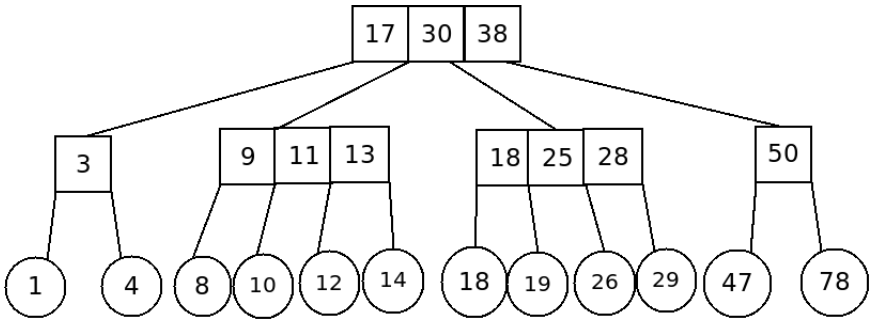


FIGURE 1.1 – a-b tree

L'avantage des arbres a-b est que leurs hauteurs sont sont comprises entres les valeurs suivantes : $\frac{\log n}{\log b} \leq h < 1 + \frac{\log n/2}{\log a}$. Ainsi les opération d'insertions ne serait plus en $n \log n$ mais en $\log n$.

La création d'un pavage composé de n boîtes à d dimensions aurait alors une complexité égale à : $(n * d) * \log(n)$ $\triangle$

# Bibliographie

[Tre] Class treemap. http://docs.oracle.com/javase/1.4.2/docs/api/java/util/TreeMap.html.