Technical Details

胡雨軒

November 13, 2014

Contents

1 La carte de données PairMap

La structure de données dans laquelle on stocke les caractères est un dictionnaire qui a pour clef l'IDS d'un caractère (ou son hash). Il faut une structure qui évite complètement les redondances, c'est-à-dire que la forêt des sinogrammes n'ait pas de doublon quel que soit l'ordre dans lequel les caractères sont entrés et quel que soit le détail des IDS. PairMap n'a pas vocation a être générique : des structures existent déjà pour cela mais à offrir la solution la plus adaptée au stockage des sinogrammes.

Trois stratégies d'optimisation des accès mémoires :

- Un tas ;
- Un accès aléatoire ;
- Un arbre couvrant.

Stratégie du tas

- Chaque caractère possède ses propres composants ;
- La taille d'un tas est 3 ;
- Solution tributaire des IDS ;

- Explosion combinatoire pour mettre à jour ;
- Analyse de caractère rapide.

Stratégie de l'accès aléatoire

- Lecture et écriture rapide si vraiment aléatoire ;
- Chaque sinogramme contient un tableau de référence de composants.

Stratégie de l'arbre couvrant

- Hypothèse : faiblement connecté ;
- Euh… mais ce n'est vraiment pratique à parcourir !

Un sinogramme comporte trois références vers ses composants et pour l'immense majeure partie deux références. Si le sinogramme est une entrée de la table on a un caractère, un point de code et une IDS. Si c'est un sinogramme induit on n'a qu'une IDS. Certains sinogrammes d'un pan Unicode (*unicode block*) sont décomposés en sinogrammes qui n'ont pas de décomposition dans ce pan mais se décomposent dans un autre pan. L'ordre de décomposition des sinogrammes n'est pas respecté globalement mais seulement à l'intérieur d'un pan. Bien qu'un point de code et une IDS renvoient à une même réalité, il faut les voir comme deux sous-clefs, facette d'une même clef.

Tous les caractères ont une IDS mais seulement les caractères explicites ont un point de code. Les caractères implicites n'ont pas de point de code, seulement une IDS. L'IDS peut donc être vue comme une clef principale à côté de laquelle on place lorsque c'est possible un point de code. L'organisation interne de `PairMap` devra donc être un dictionnaire `(K1, V)` avec `K1` une IDS et `V` une référence vers un objet `Node`. Il y a également un dictionnaire `(K2, K1)` avec `K2` un point de code. Le dictionnaire réciproque `(K1, K2)` n'est nécessaire que pour économiser la résolution d'une référence : on peut en effet accéder à `K2` très facilement à partir de `K1` : `K1` → `V.K2`.

L'interface de `PairMap` est finalement un dictionnaire `(K1, K2, V)`.

La classe `TreeMap` peut-elle apporter quelque chose de plus que `HashMap` ?