Programmation Fonctionnelle Avancée

Cours 8

kn@lri.fr
http://www.lri.fr/~kn

université PARIS-SACLAY

Résumé de l'épisode précédent

On connaît plusieurs façons de représenter des dictionnaires. On pose k la taille des clés et n le nombre d'éléments :

	Accès	Données triées	Persistante
Liste de paires	$O(k\,n)$	✔	✔
Tableau de paires	$O(k \log(n))$	✔	✘
`Map` (ABR)	$O(k \log(n))$	✔	✔
`Hashtbl`	$O(k)$ (amorti)	✘	✘

- Souvent, la taille des clés est négligée (considérée comme une constante). C'est valide pour des entiers 64bits, mais pas pour tous les types de clés (listes, chaînes de caractères, nombres en précision arbitraire...).
- Ici, Persistante=✘ signifie que la seule façon de rendre la structure de donnée persistante est de faire une copie (O(n)).
- Persistante ⇒ Mutable, car il suffit de prendre une référence vers la structure persistante.

Plan

Corrigé commenté du TP 7, EXO 1

On fait un corrigé commenté du TP (~15 à 20 minutes)

Plan

Préliminaires

On cherche une structure de donnée de dictionnaire telle que:

	Accès	Données triées	Persistante
`????`	$O(k)$	✔	✔

On va faire l'hypothèse générale que les clés sont des *chaînes de caractères*. Ce n'est pas réducteur :

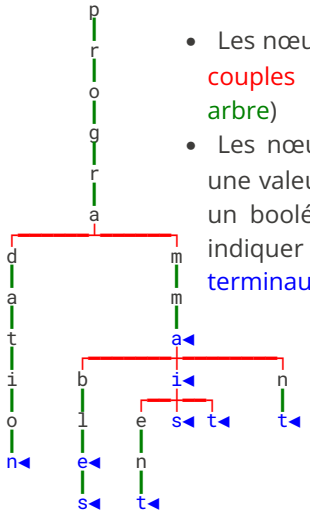
- Entiers, flottants : chaînes dont les caractères sont les bits
- n-uplet $(v_1, \ldots, v_n)$: `"`v_1`:…:`v_n`"`

Dictionnaire français

Observons une portion de dictionnaire (le livre, pas la structure de données)

```
//Arbre complet :
...
p─r─o─g─r─a─d─a─t─i─o─n
          └─m─m─a
                ├─b─l─e
                │     └─s
                ├─i
                │ ├─e─n─t
                │ ├─s
                │ └─t
                ├─n─t
                ⋮
```


Arbre final



- Les nœuds sont des listes de couples (caractère, sous-arbre)
- Les nœuds contiennent soit une valeur (dictionnaires) soit un booléen (ensemble) pour indiquer qu'ils sont terminaux.

Trie

Concept proposé en informatique pour la première fois par René de la Briandais en 1959 !

L'année suivante, Edward Fredkin re-découvre le concept et appelle la structure de donnée *trie* (jeu de mot sur *tree* et *re trie val*). La prononciation originale est « tri » mais beaucoup de personnes disent « traille » pour éviter la confusion avec les arbres (ex: si on parle de structure de données et qu'on compare les « trie » (traille) aux « binary search tree » (tri)).

Complexité de la recherche ?

Soit une chaîne `key = "c0 c1 … ck-1"` et un trie `t`

- Prendre le caractère c_i, et :
  - Chercher dans la liste de paires du nœud courant une paire (c_i, t')
  - Si cette paire existe, chercher récursivement c_{i+1} dans t'
  - Si cette paire n'existe pas, non trouvé
- En fin de chaîne, si le nœud courant est terminal, trouvé, sinon non-trouvé

On effectue k étapes. Dans chacune d'elle, traverse une liste de taille au plus S où S est la taille de l'alphabet i.e. le nombre de caractères différents. C'est une constante pour un type de clé donné (2 pour des entiers, 256 pour des chaînes, ...) donc la complexité totale est $O(k)$.

Taille du dictionnaire en pratique ?

Même si c'est une constante, une taille de dictionnaire de 256 peut être problématique.

Mais en pratique, cette situation ne se produit pas ! Si on considère le dictionnaire des mots français, seuls 26 caractères sont utilisés (ou un peu plus avec les diacritiques).

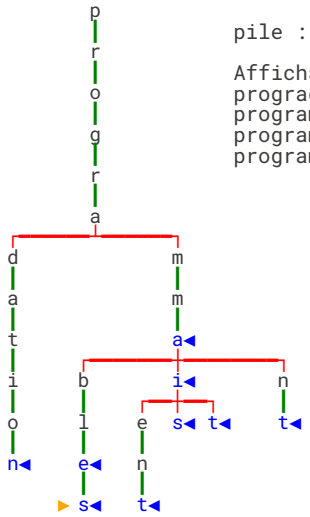
De plus, seul le premier nœud contient 26 caractères (car il y en français des mots qui commencent par chacune des lettres) `[('a', t1); … ; ('z', t26)]`. Mais dès le niveau inférieur, certaines lettres sont absents de certains nœuds (par exemple `t26` ne contient pas « x » car il n'y a pas de mot « zx... » en français.

Parcourir les clés

Soit l'algorithme suivant :

- Conserver une liste, initialement vide
- Pour un nœud donné :
  - Si le nœud est terminal, renverser la pile et l'afficher
  - Pour chaque caractère de la liste, l'ajouter à la pile et parcourir le sous-arbre correspondant

Illustration




```
pile : ['s';'e';'l';'b';'a';'m';'m';'a';'r';'g';'o'; r'; 'p']

Affichage :
progradation
programma
programmable
programmables
```


Démo

- On commence une implémentation minimale (sera poursuivie en TP)
- On donne une application : la complétion automatique de mot

Plan