Module EA4 – Éléments d'Algorithmique II

Outils pour l'analyse des algorithmes

Dominique Poulalhon
dominique.poulalhon@irif.fr

Université Paris Diderot
L2 Informatique & Math-Info
Année universitaire 2019-2020

Arbres Binaires de Recherche

I. Motivations – Rappels

Quelle structure de données pour représenter un ensemble ?

Solution 1 : une liste de ses éléments

Problème : il est naturel de souhaiter que la complexité en espace de la représentation dépende uniquement de l'ensemble représenté
Or ce n'est pas le cas si chaque élément de l'ensemble peut apparaître un nombre quelconque de fois dans la structure qui le représente

Ce n'est pas seulement un problème de complexité en espace : si un ensemble de n éléments est représenté par une liste, avec répétitions, de longueur ℓ, tous les algorithmes de manipulation de l'ensemble auront également une complexité *en temps* dépendant de ℓ et non de n. Par exemple, si la liste est non triée, il faudra parcourir les ℓ éléments avant d'être sûr qu'un élément ne s'y trouve pas.

Donc : il est préférable d'interdire les doublons

Solution 1 (bis) : une liste de ses éléments, sans doublon

C'est la solution que nous avons explorée jusqu'à présent, en comparant les variantes liste chaînée *vs* tableau, et liste non triée *vs* non triée.

Récapitulatif des complexités de différentes opérations que nous avons considérées (complexités dans le pire cas) :

	tableau		liste chaînée	
	non trié	trié	non triée	triée
recherche d'un élément	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$
minimum/maximum		$\Theta(1)$		$\Theta(1)$
sélection du k^e				$\Theta(k)$
union/intersection/...	$\Theta(n^2)$ (sans trier) $\Theta(n \log n)$ (en triant)	$\Theta(n)$	$\Theta(n^2)$ (sans trier) $\Theta(n \log n)$ (en triant)	$\Theta(n)$

Justification rapide, consulter les cours précédents si nécessaire :

- pour la recherche : linéaire *vs* dichotomique
- pour la sélection : sélection rapide (complexité en moyenne, pas au pire) *vs* accès direct ou parcours des k premiers maillons
- pour l'union ou l'intersection : via la fusion de listes triées

Solution 1 (bis) : une liste de ses éléments, sans doublon

C'est la solution que nous avons explorée jusqu'à présent, en comparant les variantes liste chaînée *vs* tableau, et liste non triée *vs* non triée.

Récapitulatif des complexités de différentes opérations que nous avons considérées (complexités dans le pire cas) :

	tableau		liste chaînée	
	non trié	trié	non triée	triée
recherche d'un élément	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$
minimum/maximum		$\Theta(1)$		$\Theta(1)$
sélection du k^e				$\Theta(k)$
union/intersection/...	$\Theta(n^2)$ (sans trier) $\Theta(n \log n)$ (en triant)	$\Theta(n)$	$\Theta(n^2)$ (sans trier) $\Theta(n \log n)$ (en triant)	$\Theta(n)$

$\Longrightarrow$ pour ces opérations, statiques, supériorité nette du tableau trié

Solution 1 (bis) : une liste de ses éléments, sans doublon

pour les opérations statiques, supériorité nette du tableau trié

mais qu'en est-il des opérations *dynamiques* ? un ensemble a vocation à évoluer par ajout ou suppression d'éléments

remarque : chaque suppression nécessite une recherche préalable ; et pour garder une liste sans doublon, chaque insertion également. Cela donne donc :

	tableau		liste chaînée	
	non trié	trié	non triée	triée
recherche d'un élément	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$	$\Theta(n)$
insertion	$+\,\Theta(1)$	$+\,\Theta(n)$	$+\,\Theta(1)$	$+\,\Theta(1)$
suppression	$+\,\Theta(n)$	$+\,\Theta(n)$	$+\,\Theta(1)$	$+\,\Theta(1)$

$\Longrightarrow$ coût linéaire pour chaque forme de liste !

Le tableau est une forme trop rigide pour permettre des évolutions peu coûteuses : un nouvel élément n'a qu'une seule place possible, et lui faire une place au bon endroit ne peut pas se faire uniquement via des modifications locales, il faut déplacer tous les éléments à sa droite.

Moralité : il faudrait trouver une structure ayant à la fois la souplesse de la liste chaînée, le caractère « ordonné » des listes triées, et permettant d'accéder rapidement à n'importe quel élément (peut-être pas en $O(1)$ comme pour les tableaux, mais pas non plus en $\Theta(n)$ comme pour les listes chaînées)
(et toujours sans doublon pour ne pas manipuler des structures de taille inutilement grosse)

Solution 2 : un arbre binaire, « trié », sans doublon

Idée générale :

- (comme dans une liste triée), on veut ranger les éléments – les petits à gauche, et les grands à droite, pour guider la recherche,
- si l'arbre est « à peu près » équilibré, aucun élément ne sera « très loin » de la racine (on peut espérer une hauteur en $\Theta(\log n)$,
- ne pas avoir une seule forme d'arbre de taille n offre une souplesse pour ajouter ou supprimer un élément ; il faut en tirer parti pour pouvoir se contenter de modifications locales, comme les modifications de chaînage d'une liste chaînée.