Récapitulatif des structures de données en informatique vues en SRI-1ère année

M.C. Lagasquie

24 août 2018

En Informatique, les structures de données sont essentiellement destinées à rassembler un ensemble de données. Ces données peuvent être :

hétérogènes (de nature différente) : par exemple une chaîne de caractères et 3 entiers pour représenter le nom et la date de naissance d'une personne. Pour faire cela, on pourra utiliser des structures de données de type "enregistrement" (`struct` en C, par exemple). Dans ce cas-là, ce qui importe est de rassembler quelques données de nature différente pour en faire un tout manipulable facilement. Le nombre de données est ici en général peu élevé.

homogènes (de même nature) : par exemple un ensemble de personnes. Il s'agit là aussi de rassembler ces données pour en faire un tout manipulable mais ici on va faire face à un problème d'efficacité puisque le nombre de ces données peut être très grand. Les opérations à réaliser sont alors des ajouts ou suppressions de donnée, des recherches de donnée (pour un simple accès ou une modification).

Le document courant est destiné à récapituler les structures de données vues cette année en cours et permettant de traiter le second cas (données homogènes). Elles sont au nombre de 4 :

- tableau (trié ou non trié),
- pile,
- file,
- arbre binaire de recherche (arbre GRD).

Pour chacune d'entre elles, on donnera les opérations possibles avec leur complexité temporelle et les limites ou contraintes d'utilisation de ces structures.

Remarque sur la complexité temporelle des opérations On utilisera la notation de Landau $O(f(n))$ qui majore le temps d'exécution d'un programme quand le nombre de données n tend vers l'infini [1] :

$$T(n) = O(f(n)) \text{ si } \exists c, n_0 \text{ tels que } \forall n > n_0, T(n) \leq c \times f(n)$$

1 Tableau (table)

C'est une structure en général statique (c'est-à-dire dont la taille est fixée au départ du programme et qui ne peut plus évoluer). Cela implique des limites pour l'opération d'ajout :

1. Cette notion sera réutilisée l'an prochain lors du cours IA-RO-CSP.

pas possible de dépasser un nombre maximum de données stockées. Par contre, un avantage important est l'accès direct à chaque donnée *si on connaît son indice* (sa clé d'accès).
Il existe deux moyens d'utiliser un tableau :

— soit non trié : les données sont stockées dans n'importe quel ordre dans le tableau,
— soit trié : les données sont stockées dans un ordre fixé par le programmeur dans le tableau ; on va donc pouvoir utiliser une méthode dichotomique pour accéder aux valeurs.

Le choix d'un de ces moyens influence fortement le temps d'exécution des opérations (N étant le nombre max de données dans le tableau) :

	ajout (avec limite !)	suppression	recherche/modification
Tableau non trié	$O(1)$	$O(N)$	$O(N)$
Tableau trié	$O(log(N)+N)$	$O(log(N)+N)$	$O(log(N))$

Notons qu'il existe des algorithmes de tri permettant de trier un tableau qui ne le serait pas (l'un des plus efficaces est le *tri rapide* de complexité $O(N \times log(N))$).

2 Pile (stack)

C'est une structure en général dynamique[2] (c'est-à-dire dont la taille évolue au cours du programme). En théorie, elle n'a donc pas de limite en nombre de données stockables. Par contre, elle est destinée à représenter un comportement d'ajout et d'accès très particulier, la *dernière donnée ajoutée* jouant un rôle essentiel : c'est la seule donnée directement accessible et pour accéder à une autre donnée il faut la retirer de la structure. On peut faire l'analogie avec une pile d'assiettes : on ajoute sur le sommet de la pile et on enlève depuis le sommet de la pile.
On ne peut donc pas faire de tri, ni accéder directement à n'importe quelle donnée.
De même, l'opération de suppression est contrainte par le fait que c'est toujours la *dernière donnée ajoutée qui sera supprimée* (contrairement à ce qu'on peut faire dans un tableau, où l'on peut choisir de supprimer la donnée que l'on veut).
Le temps d'exécution des opérations est le suivant (N étant le nombre max de données dans la pile) :

	ajout	suppression (avec contrainte !)	recherche/modification
Pile	$O(1)$	$O(1)$	non applicable

Notons qu'on peut toujours faire une recherche dans une pile mais qu'il s'agit là d'une opération destructrice : on déconstruit la pile jusqu'à trouver ce que l'on cherche. C'est pour cela qu'on considère qu'une pile n'est pas faite pour faire de la recherche ou de la modification de données. C'est juste un moyen de stocker des données sachant qu'on les utilisera dans l'ordre inverse de celui du stockage (la plus récente d'abord).

2. Notons qu'on peut quand même la simuler à l'aide d'un tableau.

3 File (queue)

C'est une structure en général dynamique[3] (c'est-à-dire dont la taille évolue au cours du programme). En théorie, elle n'a donc pas de limite en nombre de données stockables. Par contre, elle est destinée à représenter un comportement d'ajout et d'accès très particulier, la *donnée la plus ancienne ajoutée* jouant un rôle essentiel : c'est la seule donnée directement accessible et pour accéder à une autre donnée il faut la retirer de la structure. On peut faire l'analogie avec une file d'attente : on ajoute à la fin de la file et on enlève depuis le début de la file.
On ne peut donc pas faire de tri, ni accéder directement à n'importe quelle donnée.
De même, l'opération de suppression est contrainte par le fait que c'est toujours la *donnée ajoutée depuis le plus longtemps qui sera supprimée* (contrairement à ce qu'on peut faire dans un tableau, où l'on peut choisir de supprimer la donnée que l'on veut).
Le temps d'exécution des opérations est le suivant (N étant le nombre max de données dans la file) :

	ajout	suppression (avec contrainte !)	recherche/modification
File	$O(1)$	$O(1)$	non applicable

Ici aussi, notons qu'on peut toujours faire une recherche dans une file mais qu'il s'agit là d'une opération destructrice : on déconstruit la file jusqu'à trouver ce que l'on cherche. C'est pour cela qu'on considère qu'une file n'est pas faite pour faire de la recherche ou de la modification de données. C'est juste un moyen de stocker des données sachant qu'on les utilisera dans le même ordre que celui du stockage (la plus ancienne d'abord).

4 Arbre binaire de recherche (search binary tree)

C'est une structure en général dynamique[4] (c'est-à-dire dont la taille évolue au cours du programme). En théorie, elle n'a donc pas de limite en nombre de données stockables.
Contrairement à la pile ou à la file, les données sont stockées de manière triée. En effet, chaque nœud de l'arbre doit vérifier une contrainte d'intégrité spécifique : toutes les valeurs stockées dans le sous-arbre gauche du nœud sont inférieures ou égales à la valeur du nœud et toutes les valeurs stockées dans le sous-arbre droit du nœud sont strictement supérieures à la valeur du nœud. Cela donne le moyen d'obtenir un tri des valeurs stockées par simple parcours de tout l'arbre.
Par contre, il n'y a pas d'accès direct aux données (il faut partir de la racine et parcourir l'arbre). Contrairement à la pile ou à la file, ce parcours n'est pas destructif. On peut donc supprimer la donnée de notre choix.
Le temps d'exécution des opérations est le suivant (N étant le nombre max de données dans l'arbre et h la hauteur de l'arbre) :

	ajout	suppression	recherche/modification
Arbre binaire de recherche	$O(h)$	$O(h)$	$O(h)$

avec la contrainte suivante : $1 \leq h \leq N$ et
dans le meilleur des cas (arbre équilibré) $h \approx log(N)$

3. Notons qu'on peut quand même la simuler à l'aide d'un tableau.
4. Notons qu'on peut la simuler à l'aide d'un tableau dans certains cas bien particulier (arbre équilibré encodé sous la forme d'un tas).

5 Tableau récapitulatif

Soit N le nombre max de données, soit $1 \leq h \leq N$ (dans le meilleur des cas, $h \approx log(N)$) :

	ajout	suppression	recherche/modification
Tableau non trié	$O(1)$ avec limite	$O(N)$	$O(N)$
Tableau trié	$O(log(N)+N)$ avec limite	$O(log(N)+N)$	$O(log(N))$
Pile	$O(1)$	$O(1)$ avec contrainte	non applicable
File	$O(1)$	$O(1)$ avec contrainte	non applicable
Arbre binaire de recherche	$O(h)$	$O(h)$	$O(h)$