Structures de données hiérarchiques

Terminologie

Définition 1

Un **arbre** est un graphe acyclique, non orienté et connexe. Ses **noeuds** en sont les sommets. La **racine** est un noeud particulier de l'arbre. Tout noeud autre que la racine qui n'est relié qu'à un seul noeud est une **feuille**.

Remarque : La racine peut être choisie parmi l'ensemble des sommets sans modification du caractère arborescent. Pour un graphe donné, chaque choix de sommet donne lieu à la représentation d'un arbre particulier (dans le cadre des arbres enracinés dans le plan).

Définition 2

Soit N un ensemble fini d'éléments appelés noeuds. L'ensemble des arbres est défini inductivement par :

- les **feuilles**, graphes contenant un seul sommet n et aucune arête sont des arbres de hauteur 0, et de racine eux-même
- Si n est un élément de N, et si $A_1, ..., A_n$ sont des arbres de hauteurs respectives $h_1, ...h_n$, alors on construit un arbre de racine n et de hauteur $1 + \max(h_i)$ en liant n aux racines des arbres A_i. Ses sous-arbres sont les A_i

Dans le second cas, si $r_1, .., r_n$ sont les racines de l'arbre construit, on dit que ces noeuds sont les **fils** de n, que n est leur **père**, et qu'ils sont **frères**. Si un noeud n' est dans un des sous-arbres A_i, c'est un descendant de n, qui est son **ancêtre**.

La terminologie est construite par analogie avec les arbres généalogiques. On pourra ainsi parler d'ascendants, de descendants, d'ancêtres, de cousins, etc.

Définition 3

- Les noeuds peuvent être **etiquetés**, c'est-à-dire accompagnés d'une information qui leur est associée, et peut être de tout type ;
- Un noeud qui n'a aucun fils est appelé **feuille** ;
- Un noeud qui a au moins un fils est un noeud **interne** ;
- La **profondeur** d'un noeud dans un arbre est le nombre d'arêtes reliant ce noeud et la racine ;
- la **hauteur** d'un arbre est la profondeur la plus grande parmi celles de ses noeuds ;
- L'**arité** (ou **degré**) d'un arbre est le nombre maximal de fils des noeuds de cet arbre ; si l'arité est 2, l'arbre est dit **binaire**, plus généralement, on parle d'arbre n-aire ;
- Un arbre est **complet** si toutes ses feuilles ont la même profondeur, et que tous les noeuds ont même nombre de fils.

Le nombre de noeuds d'un arbre complet binaire de profondeur h est $2^{h+1} - 1$; la profondeur d'un arbre binaire complet de n noeuds est $\log_2(n+1) - 1$. Qu'en est-il pour des arbres ternaires ou n-aires ? Comment compter dans ces arbres les feuilles et les noeuds internes ?

Exercice 1

Proposer un type abstrait de données pour le type de donnée Arbre.

Exemple de type abstrait de donnée Arbre :

Nom	Arbre
Paramètres	Élément
Utilise	Booléen

Opérations	`arbre : Élément -> Arbre`
	`ajoutRacine : Arbre * Liste <Arbres> -> Arbre`
	`ajoutFils : Arbre *Arbre -> Arbre`
	`suppressionFils : Arbre *Arbre -> Arbre`
	`obtenirEtiquetteRacine : Arbre -> Élément`
	`obtenirFils : Arbre -> Liste <Arbres>`
	`estVide : Arbre -> Booléen`

Axiomes	`suppressionFils (ajoutFils(a,b),a)=a`
	`obtenirFils (ajoutRacine(a,l))⊃la`
	`obtenirEtiquetteRacine(arbre (a))=a`

Préconditions Préconditions	suppressionFils(a) : a non vide obtenirEtiquetteFils(a) : a non vide

Comment ajouter ou supprimer un noeud d'un arbre ?

Parcours d'arbres

Un **parcours d'arbre** est la donnée d'un chemin sur l'arbre en tant que graphe permettant d'énumérer (et de traiter le cas échéant) ses noeuds suivant un ordre prédeterminé

Parcours en profondeur

Le parcours en profondeur utilise de façon naturelle la définition inductive des arbres au sein d'un algorithme récursif. On peut également le réaliser à l'aide d'une pile : la pile est initialisée en y mettant uniquement la racine de l'arbre ; tant que la pile n'est pas vide, on dépile un noeud, et on enfile ses fils.

Quelle que soit l'arité de l'arbre, un choix doit être fait sur l'ordre dans lequel les noeuds fils (et sous-arbres associés) et le noeud père sont traités.

- Si on traite le père avant les fils, l'ordre est dit **préfixe**
- Si on traite le père après les fils, l'ordre est dit **postfixe**
- Si on traite le père entre ses fils (dans le cas des arbres strictement binaire), l'ordre est dit **infixe**

Exemple

Au tableau

Exercice 1

Implémenter en Caml et en C ces trois parcours d'arbres.

Remarque

Dans le cas du parcours d'un arbre syntaxique du langage arithmétique, ces parcours permettent d'évaluer une opération arithmétique. Dans ce cadre, l'ordre postfixe, non ambigu, correspond à la **notation polonaise inversée**. L'ordre infixe est ambigu.
L'évaluation se fait comme suit :
Chaque élément est lu, du premier au dernier. S'il s'agit d'un opérande, on l'empile ; si c'est un opérateur (binaire), on dépile deux éléments et on calcule le résultat qu'on empile. A la fin de la lecture de l'expression en notation polonaise inversée, on dispose de l'évaluation de cette expression.

Parcours en largeur

Les noeuds sont parcourus par profondeur croissante. On réalise ce parcours à l'aide d'une file dans laquelle les noeuds sont stockés avant d'être traités. Lorsqu'un noeud est défilé, ses fils sont enfilés, et les noeuds sont visités dans l'ordre où ils ont été enfilés.

File de priorité

Une **file de priorité** est une structure de données de type file, où les éléments sont dotés d'une priorité, qui détermine l'ordre dans lequel ils vont être traités.
Ce type de structure peut être implémenté par un tableau, dont on trie les éléments par priorité croissante. L'accès à l'élément à enlever de la file, et l'opération `défiler` se font alors simplement par accès au dernier élément. En revanche l'insertion est en temps linéaire de la taille de la file. Si on implémente la file de priorité comme une file classique, l'opération `enfiler` est en temps constant, mais `defiler` est linéaire en la taille de la file. Une autre implémentation est celle qui utilise la structure de **tas**.

Arbre binaire ; arbres généraux

Exemple

Les **tries** ou **arbres préfixes** sont des arbres dont l'arité des noeuds est non-bornée (sinon par la taille de l'alphabet A utilisé). Chaque noeud est étiqueté par une lettre de A, distincte de celles des étiquettes des noeuds-frères. Tout noeud est ainsi en bijection avec un mot lisible depuis la racine sur les noeuds qui y mènent. Les tries servent à stocker des tables associatives dont les clefs sont des chaînes de caractères.

Conversion d'arbres généraux en arbres binaires

Un arbre A d'arité quelconque peut être converti en arbre binaire en utilisant l'algorithme qui suit :

- La racine de l'arbre binaire est celle de l'arbre A ;
- le fils gauche d'un noeud dans l'arbre d'origine est le fils gauche de l'arbre binaire obtenu ;
- le frère droit d'un noeud dans A est le fils droit du même noeud dans l'arbre binaire

Tout noeud est alors d'arité 0, 1 ou 2.

L'application ainsi présenté est-elle bijective ? Le cas échéant, quelles modification apporter pour obtenir une bijection (et dans quel ensemble) ? Quelle serait l'application réciproque ?

Exercice 1

> Coder l'algorithme présenté.

Arbre binaires

Un arbre binaire est un arbre n-aire, avec $n = 2$. Les arbres binaires permettent, par exemple, de représenter les opérations arithmétiques.

Exemple

- Arbres binaires et expressions arithmétiques / formules de logique propositionnelle.
- Arbres de décision
- Dendrogramme : soit un ensemble E. la racine est étiquetée par E ; deux noeuds sont étiquetés par des ensembles qui n'ont rien en commun ou dont l'un est inclus dans l'autre ; les feuilles sont étiquetés par des singletons.

Soit A un arbre binaire de hauteur h à f feuilles Alors :

- $f \leqslant 2^h$
- $h \geqslant \lceil \log_2 f \rceil$
- le nombre maximal de noeuds de niveaux p est 2^p

Proposition

Soit A un arbre binaire à i noeuds internes et f feuilles, dans lequel chaque noeud interne a pour arité 2 Alors $f = i + 1$

Arbre binaire de recherche

Définition 4

Soient E un ensemble muni d'une relation d'ordre totale et A un arbre binaire dont les étiquettes sont des éléments de E. L'arbre A est un arbre binaire de recherche si l'étiquette de tout noeud est supérieure strictement aux étiquettes des noeuds de son sous-arbre gauche et strictement inférieure aux étiquettes des noeuds de son sous-arbre droit.

Remarques :

• L'ordre est nécessairement total (par transitivité)

• Si on s'intéresse à la relation stricte, les étiquettes ne peuvent être attribuées qu'une fois.

Ces arbres servent à effectuer des insertions, délétions et recherches fréquentes d'éléments en complexité temporelle optimale. Ils permettent notamment d'implémenter des dictionnaires de manière efficace.

Exercice 2

Concevoir et implémenter les opérations suivantes en $O(h)$:

- Insertion dans un ABR
- Recherche dans un ABR
- Algorithme de suppression. On considère ici qu'on souhaite supprimer un noeud dont on connaît l'étiquette, mais pas les descendants de ce noeud.

Définition 5

Un **arbre bicolore** (ou arbre rouge - noir) est un arbre binaire de recherche, binaire strict, dont les noeuds sont colorés en rouge ou en noir et vérifient les propriétés suivantes :

- la racine est noire ;
- si un noeud est rouge, ses deux enfants sont noirs ;
- les feuilles ne portent pas d'étiquettes ;
- pour chaque noeud, tous les chemins menant de ce noeud à une feuille ont le même nombre de noeuds noirs.

Exercice 3

Montrer qu'aucun chemin vers une feuille n'est plus de deux fois plus long qu'un autre.

Exercice 4

En déduire que la hauteur de l'arbre est logarithmique en son nombre de noeuds internes

Les opérations de recherche, d'insertion, de suppression sont en complexité logarithmique du nombre de noeuds, et sont un peu plus complexes que dans des arbres binaires : il faut garder à tout moment la propriété d'arbre bicolore. L'arbre garde en contrepartie une forme d'équilibre qui garantit le temps des opérations en question, même dans le pire cas.

Exercice 5

> Concevoir et implémenter les opérations de recherche, d'insertion et de suppression dans un arbre bicolore, toujours en en $O(h)$:

L'algorithme de recherche se conçoit sans difficultés. L'insertion et la suppression s'appuient sur la **rotation**, qui est une opération transformant un arbre bicolore en arbre bicolore, qui échange un noeud avec l'un de ses fils. Plus précisément :

- La **rotation droite** fait d'un noeud x dont le fils gauche est y le fils droit de y. Le fils droit de y devient le fils gauche de x.
- La **rotation gauche** :

Exercice 6

> Montrer que les rotations conservent le caractère d'arbre binaire de recherche. Montrer que les rotations gauche et droite sont réciproques. Quel est la complexité d'une rotation ?

.

a Insertion :

On insère le noeud x dans l'arbre comme dans un ABR, et on le colorie en rouge. Puis il faut rétablir les propriétés d'arbre bicolore, ce qui va se faire avec, éventuellement, propagation des modifications.
Notons p le nouveau parent de x. Si p est noir, on n'a rien à modifier (x est forcément inséré avec deux fils feuilles noirs, ce qui ne modifie pas le nombre de noeuds noirs sur le chemin vers les feuilles, ou on est dans le cas des propagations, et les fils de x sont bien noirs).
Si le père p est rouge, on peut en revanche avoir des modifications à effectuer, selon les cas :

- Si p est la racine, on la colorie en noir. *L'arbre est-il bien bicolore ?*
- Si p n'est pas la racine, il possède un frère q et un père r. A ce stade, on sait que p est rouge, donc r est noir.
    - Si q est rouge, on échange les couleurs de p, q et de r. Il faut ensuite propager le rétablissement du caractère "arbre bicolore" vers la racine. (le père de r est-il rouge ? etc.)
    - Si q est noir : sans perte de généralités, on peut supposer que p est le fils gauche de son père (la symétrie permettra de recomposer les cas manquants). A nouveau, on peut décomposer cette situation en deux sous-situations :
        - Si x est le fils gauche de p, on effectue une rotation droite sur les noeuds p et r (r devient donc le fils de p). Le noeud p devient noir, et r devient rouge.
        - Si x est le fils droit de p, on effectue une rotation gauche sur les noeuds p et x, ce qui nous ramène au cas précédent : rotation entre x et r, x mis à noir et r mis à rouge

b Suppression :

Si on souhaite supprimer le noeud x de l'arbre, soit il n'a pas de fils (autre que des feuilles noires non étiquetées), on le supprime (et le remplace par sa feuille noire) sans modifier le caractère d'arbre bicolore de l'arbre ; soit il a un fils, et on remplace le noeud par son fils, puis on rétablit les couleurs si besoin ; soit il a deux fils. Ce dernier cas est plus complexe et va, à l'instar de ce qu'il se passe au cours de l'insertion d'un noeud, nécessiter une disjonction de cas. Dans tous les cas, on va remplacer l'étiquette du noeud à supprimer par celle de son successeur x, qu'on

supprime en recollant son fils s'il existe, puisqu'il ne peut en avoir deux.

Maintenant, si x était rouge, on n'a pas modifié le caractère bicolore de l'arbre en le remplaçant par son fils. S'il était noir, on a supprimé dans le sous-arbre enraciné en x un noeud noir pour tous les chemins vers des feuilles. On rajoute donc une couleur noire au nouveau x. Si x était rouge, il devient noir, et l'arbre reste bicolore. Si le noeud qui a pris la place de x était déjà noir, on se retrouve avec un noeud doublement marqué part la couleur noire. Opérons encore une disjonction de cas :

- Si x est à la racine, on supprime une des deux couleurs noire, l'arbre reste bicolore, et sa hauteur noire diminue.
- Si x n'est pas à la racine, il a un frère qu'on note y et un père p. Alors
  - Si le frère y de x est noir :
    - si ses deux fils sont noirs (on suppose que x est le fils gauche de p par symétrie du problème) : le noeud x devient noir, le noeud y devient rouge, on donne à p le marqueur noir supplémentaire. Si besoin, l'algorithme continue sur le noeud doublement noir créé.
    - si le fils droit de y est rouge : on effectue une rotation gauche entre p et y. Le noeud y prend la couleur du noeud p. les noeuds p, x et le fils droit de y deviennent noirs.
    - si le fils gauche de y est rouge, et si son fils droit est noir, on effectue une rotation droite entre le fils gauche de y et y et on échange leurs couleurs, se ramenant au cas précédent ;
  - si le frère y de x est rouge, on effectue une rotation gauche entre p et y et on échange les couleurs de p et de y. Le nouveau frère de x est noir, ce qui nous ramène à un cas précédent.

Tableau associatif

Un **tableau associatif** ou **dictionnaire** contient un ensemble de valeurs chacun associée à une clef unique. On peut insérer une valeur, avec sa clef, dans un dictionnaire. Si la clef est déjà présente, la valeur précédente est effacée au profit de la nouvelle ; sinon, un nouveau couple est ajouté au dictionnaire. On peut également accéder à un élément en en connaissant la clef ; le supprimer ; en tester la présence ; tester si le dictionnaire est vide.

On peut choisir d'implémenter le tableau associatif par une table de hachage, sous forme de liste chaînée ou de tableau, avec les contraintes déjà vues sur ces structures.

Exercice 1

> Proposer un type abstrait de données pour le dictionnaire.

Nom	Dictionnaire
Paramètres	Clef, Élément
Utilise	Booléen, liste

Opérations	`cree:-> dictionnaire`
	`ajout : Clef*Element*dictionnaire-> dictionnaire`
	`enlever : Clef*Dictionnaire-> Dictionnaire`
	`lire :  Clef*Dictionnaire-> Element`
	`appartient:  Clef*Dictionnaire-> Element`
	`obtenirClefs : Dictionnaire-> Liste de clefs`
	`obtenirValeurs: Dictionnaire-> Liste de élements`

Axiomes	`ajout(clef,b,ajout(clef,a,dic)) = ajout(clef,b,dic)`
	`lire (clef, ajout(clef,a,dic))) = a`
	`appartient (a, ajout(clef,a,dic))=Vrai`
	`appartient (b,enlever(clef,a,dic))=Faux`

Exercice 2

> Proposer un code pour compter les éléments d'un tableau à l'aide d'un dictionnaire. En déduire un algorithme de tri sur des tableaux. Quelle en est la complexité temporelle ? Spatiale ? Dans quel cadre cet algorithme sera particulièrement intéressant ?

Implémentation par une table de hachage

Définition 6

Une **fonction de hachage** est une fonction qui calcule rapidement à partir d'une entrée une **empreinte numérique**, de taille (généralement) inférieure à celle de l'entrée, de manière suffisamment discriminante pour l'identifier.

Il peut s'agir par exemple du calcul d'un reste d'une division euclidienne par un grand nombre premier. La fonction doit renvoyer rapidement un résultat, et minimiser les risques de **collisions**, c'est-à-dire les couples d'entrées produisant la même sortie. En particulier, on attend que deux objets proches aient des images différentes. Si l'ensemble d'arrivée est plus petit que l'ensemble de départ, on ne pourra éviter les collisions.

Exemple : les deux chiffres de contrôle du numéro INSEE (calcul modulo 27).

Définition 7

Une **table de hachage** est un tableau permettant une association entre des clefs et des valeurs.

Un tel tableau n'est pas ordonné en fonction des clef ni des valeurs, mais on accède à l'information liée à une clef en en calculant l'image par une fonction de hachage, qui est l'indice dans le tableau où se trouve l'information cherchée. L'accès à la valeur associée à une clef se fait alors en temps constant. Le problème de collision est écarté par le choix de la fonction de hachage. Si cela n'a pas été possible, il faut pouvoir comparer la clef donnée pour la recherche avec celle liée à l'information trouvée, donc garder dans le tableau la clef avec la valeur.
Vient ensuite un mécanisme de résolution des collisions : on peut avoir dans certains cas à parcourir tout le tableau. L'accès n'est donc en temps constant qu'en moyenne, et linéaire dans le pire cas. Lorsque les collisions sont réparties le plus uniformément possible, l'utilisation des ressources reste telle que prévue.
On peut également chaîner les différentes clés (et leurs valeurs) possédant le même hash.

Implémentation par un Arbre Binaire de Recherche

Lorsqu'il est implémenté par un ABR, le tableau associatif est une donnée persistante. On doit

disposer d'un ordre total sur l'ensemble des clés. Les étiquettes des noeuds de l'ABR sont des couples (clefs, valeurs), et la place du noeud dans l'arbre dépend de la clef.

Tas

Définition 8

Un **tas** (ou **tas-max**, ou **max-heap**) est un arbre binaire **presque complet** (tous ses niveaux sont remplis sauf éventuellement le dernier, dont les feuilles sont le plus à gauche possible) et **ordonné**, dont les étiquettes d'un noeud sont plus grandes (ou égales) que celles des fils.

Remarque : Si on considère que l'étiquette correspond à la priorité, c'est la racine qui contient l'élément de plus haute priorité, d'où l'intérêt pour la représentation des files de priorité.

Exercice 1

> Montrer qu'en numérotant les noeuds de 1 à n dans le cadre d'un parcours en largeur de l'arbre, le noeud associé à i a pour père un noeud associé à $\lfloor \frac{i}{2} \rfloor$, et pour fils (s'ils existent), les noeuds associés à $2i$ et $2i+1$.

Pour chaque cas, proposer un algorithme, puis en donner la complexité.

Algorithme de recherche :

Algorithme d'insertion d'un noeud :

Algorithme de suppression : On considère ici qu'on souhaite supprimer un noeud dont on connaît l'étiquette, mais pas les descendants de ce noeud.

File de priorité

On peut implémenter une file de priorité à l'aide d'un tas en considérant que l'étiquette la plus grande représente un plus haut niveau de priorité.

- Enfiler un couple (clef, valeur) :
- Défiler :

Implémentations des files de priorité

Implémentation	Trouver le max	Insérer	Retirer le max
Tableau non ordonné	$O(n)$	$O(1)$	$O(n)$
Liste non ordonnée	$O(n)$	$O(1)$	$O(1)$ (une fois trouvé)
Tableau ordonné	$O(1)$	$O(n)$	$O(1)$
Liste ordonné	$O(1)$	$O(n)$	$O(1)$
Tas	$O(1)$	$O(\log(n))$	$O(\log(n))$

Tri par tas

Le **tri par tas** est, à l'instar des tris fusion et rapide, un tri par comparaison asymptotiquement optimal. Il se fait en place et n'est pas stable. L'algorithme est le suivant :

- Organiser les données dans un tableau représentant un arbre avec la propriété de tas

- Pour i de 1 à "taille du tableau", extraire la donnée de plus haute étiquette du tas et la ranger dans le tableau à l'indice "taille -i" jusqu'à avoir vidé l'arbre

Quelle est la complexité de ce tri ?