Structures de données

Projet 2 : utilisation de tas binaires pour le tri d'entiers

Dans le cadre des travaux pratiques d'algorithmique et de programmation en C, la réalisation de deux projets (un sur les listes, un sur les arbres) est attendue pour vérifier votre compréhension des différents concepts abordés pendant les cours magistraux et les travaux dirigés, ainsi que votre capacité à les utiliser de façon autonome face à des problèmes concrets.

Ce document décrit le second projet, qui consiste à concevoir et à implémenter les algorithmes permettant l'utilisation d'une structure de données basées sur les arbres binaires : les tas binaires. Cette structure de données présente différentes propriétés qui la rendent utile pour la résolution de plusieurs problèmes, en particulier celui du tri qui sera l'objectif de ce projet.

La section 1 décrit le projet dans son ensemble, et présente le concept de tas binaire. La section 2 donne précisément les fonctionnalités attendues. La section 3 contient les consignes et les différents points qui seront pris en compte pour l'évaluation. Et finalement, la section 4 propose plusieurs idées d'améliorations supplémentaires pour aller plus loin.

Bonne chance !

1) Présentation du projet

Les arbres binaires sont des structures de données arborescentes qui permet de stocker une collection d'objets, de type donné arbitraire, en établissant une hiérarchie : chaque objet disposant d'au plus deux objets enfants (accessibles par des pointeurs) de sorte que l'ensemble — pris à partir de son premier élément, appelé racine — forme un arbre.

Sur la base de cette structure de données primitive, il est possible de définir des propriétés particulières que l'arbre binaire doit respecter afin d'ajouter des « fonctionnalités » à l'arbre pour résoudre des problèmes. Par exemple, vous avez étudié les arbres binaires de recherche (ABR) qui sont conçus de telle sorte à faciliter, par exemple, la recherche d'un élément à l'intérieur (remarquez qu'en utilisant les propriétés des ABR, il n'est en effet plus nécessaire d'examiner chaque élément de l'arbre binaire).

Nous nous proposons ici d'étudier les tas binaires. Un tas binaire est un arbre binaire qui doit satisfaire à deux propriétés :

- Il doit être complet : c'est-à-dire que tous les niveaux doivent être intégralement remplis, à l'exception éventuelle du dernier dans lequel les feuilles sont calées à gauche.

- Il faut que la valeur de chaque nœud[1] soit supérieure ou égale à tous ses enfants.

Quand ces deux propriétés sont réunies, on parle d'un « **tas-max** ». On peut imaginer une autre version, très similaire, où la valeur de chaque nœud devrait être inférieure ou égale à tous ses enfants, on parle dans ce cas de « **tas-min** ».

Ces deux définitions coexistent et sont deux variantes des tas binaires. **Dans le cadre de ce projet, on se concentrera sur les tas-max** (en gardant à l'esprit que le fonctionnement et les algorithmes dans les deux cas restent identiques, et qu'il faut juste changer l'opération de comparaison pour passer de l'un à l'autre).

Voici un exemple de tas binaire :

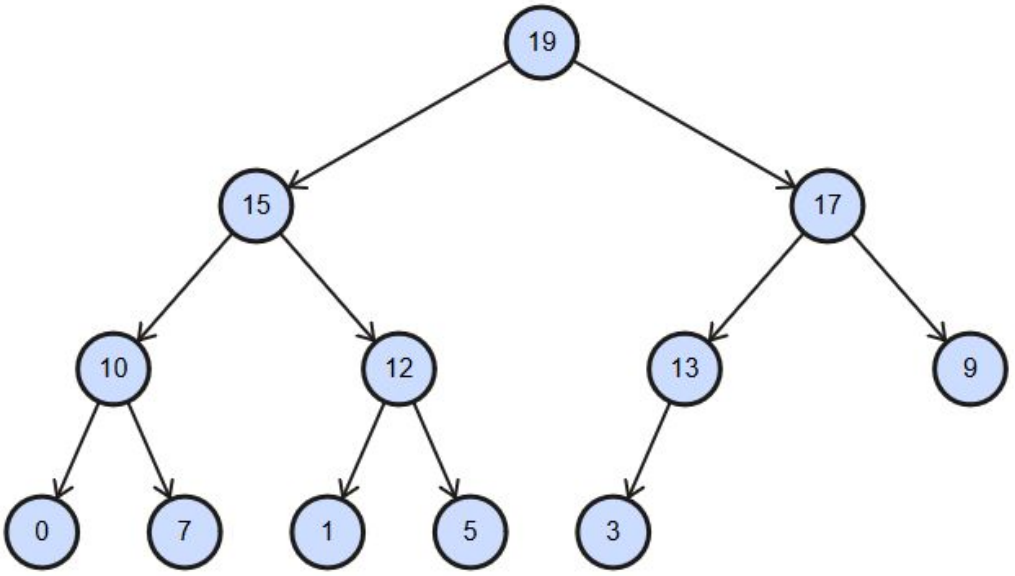

Figure 1 : exemple de tas binaire (ici, un tas-max)

Cette structure de données permet notamment de réaliser de façon relativement performante une opération courante : le tri. L'algorithme de tri utilisant cette structure est appelé **tri par tas** (*heapsort* en anglais).

L'objectif de ce projet est de s'intéresser à la représentation des tas binaires en mémoire (qui seront ici stockés sous la forme de tableaux), aux différentes opérations élémentaires d'un tas binaire, et enfin d'utiliser ces dernières pour réaliser un algorithme de tri par tas.

2) Fonctionnalités attendues : ce que vous devez faire

Vous devez réaliser un programme qui permet de manipuler un tas binaire (un tas-max) d'entiers, c'est-à-dire qui est en mesure de l'afficher et d'appliquer dessus les différentes opérations qui sont décrites ci-dessous.

[1] On considérera ici qu'il s'agit d'entiers, mais on peut utiliser n'importe quel type sur lequel une relation d'ordre est bien définie.

Pour y parvenir, vous devrez définir une structure de données en C — semblable à un arbre binaire dans sa forme, mais implémenté à l'aide d'un tableau — correspondant au tas binaire ; et définir des fonctions qui mettent en œuvre les opérations demandées.

Il est enfin nécessaire d'avoir un programme de démonstration utilisant cette structure qui demande à l'utilisateur les opérations qu'il ou elle souhaite réaliser avec un tas binaire.

2.1) Représentation d'un tas binaire en mémoire

Comme présenté dans l'introduction, les tas binaires sont des arbres binaires qui vérifient plusieurs propriétés et sur lesquels certaines opérations (notamment le tri) sont relativement efficaces.

Pour pouvoir les manipuler et appliquer sur ces derniers les différentes opérations demandées dans les sections suivantes, **il sera représenté en mémoire à l'aide d'un tableau**. Cette représentation diffère de la vision où on définit une structure représentant un nœud qui contient des références (en C des pointeurs) vers ses enfants.

Ici, puisque l'arbre est complet (propriété d'un tas binaire) il est possible d'utiliser un tableau pour stocker toutes les valeurs présentes dans le tas binaire. En effet, sa propriété de complétude garantit que le tableau de comportera pas de « trous », et permet donc de stocker directement les valeurs de l'arbre de façon contiguë en mémoire.

Pour ce faire, on utilisera une structure comportant la taille n du tas binaire et un tableau comportant n valeurs (ici il s'agit d'entiers). On applique ensuite la logique suivante (également représentée à la figure 2) pour représenter l'arbre binaire :

- on place la valeur de la racine à l'index 0 du tableau ;
- étant donné un nœud ayant pour index i, la valeur de son sous-arbre gauche est à l'index $2i+1$, et la valeur de son sous-arbre droit est à l'index $2i+2$;
- on peut enfin remarquer qu'il est possible d'accéder à la valeur du parent d'un nœud ayant pour index i en accédant à $(i-2)/2$ si i est pair, et à $(i-1)/2$ si i est impair.

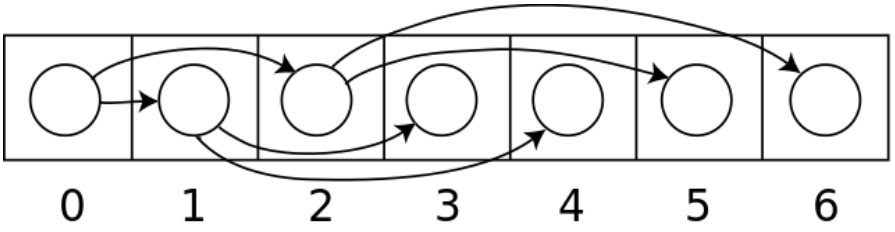

Figure 2 : représentation d'un arbre binaire complet à l'aide d'un tableau
(crédit : Dcoetzee — Wikimedia Commons — Domaine public)

Étape 1 : Définir une structure en C qui permet de représenter un arbre binaire complet en mémoire avec le fonctionnement décrit dans cette section.

Exemple : l'arbre binaire complet représenté à la figure 3 peut être représenté par le tableau 1.

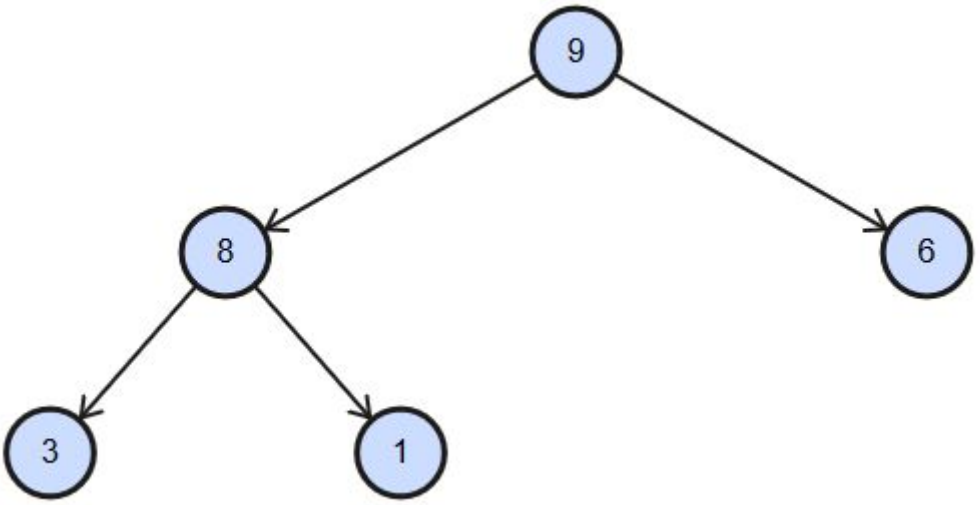

Figure 3 : exemple d'arbre binaire complet (ici, c'est un tas-max)

Valeur du nœud	9	8	6	3	1
Index	0	1	2	3	4

Tableau 1 : représentation de l'arbre binaire complet de la figure 1 avec un tableau

2.2) Opérations élémentaires sur les tas binaires

Pour chacune des opérations élémentaires suivantes, il faut concevoir et implémenter en C un algorithme qui la réalise sur un tas binaire donné en entrée (qui sera représenté par la structure conçue dans la section 2.1) :

Étape 2 : Renvoyer le plus grand élément du tas binaire.

Exemple : pour le tas binaire de la figure 1, la valeur renvoyée sera 19.

Étape 3 : Ajouter un élément donné à un tas binaire donné. Il faut que l'arbre reste un tas binaire (c'est-à-dire continue de respecter les propriétés) une fois l'ajout effectué.

Indication : pour concevoir cet algorithme, regardez d'abord ce qu'il faut faire sur l'arbre (en travaillant sur un dessin) pour ajouter un élément, puis regarder ce qui se passe avec la représentation avec un tableau.

Exemple : nous ajoutons la valeur 18 au tas binaire de la figure 1, le tas binaire résultant sera alors celui représenté à la figure 4.

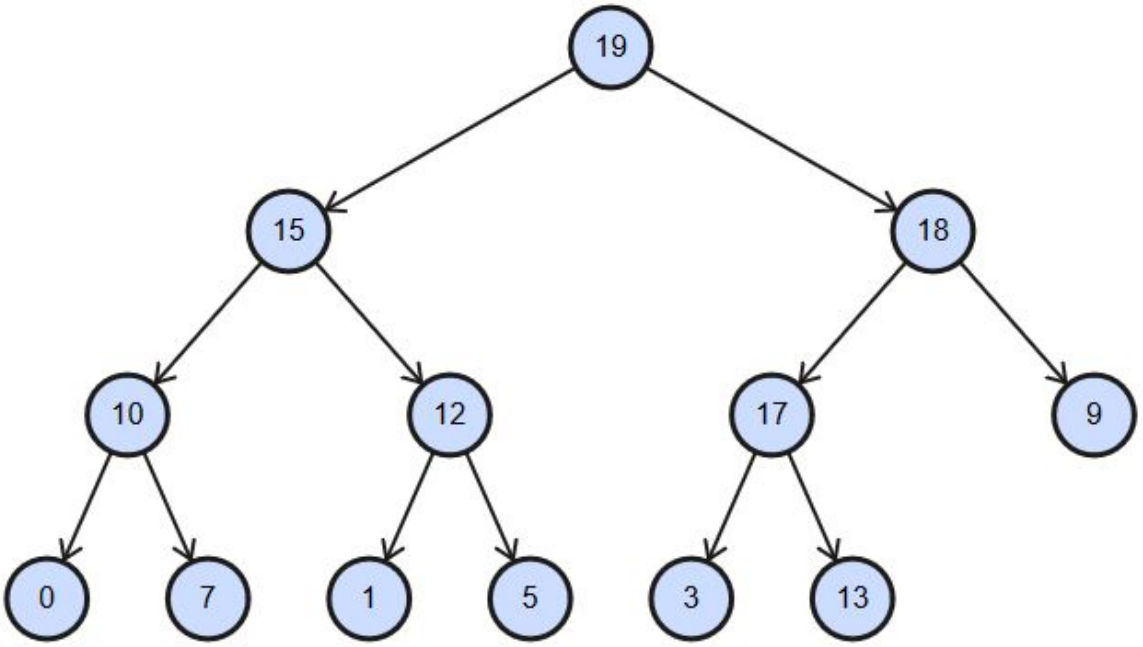

Figure 4 : tas binaire après ajout de la valeur 18 au tas de la figure 1

Étape 4 : Retirer la racine d'un tas binaire donné. Il faut que l'arbre reste un tas binaire (c'est-à-dire continue de respecter les propriétés) une fois le retrait effectué.

Indication : procédez avec raisonnement similaire à celui de l'étape 3.

Exemple : nous retirons la racine au tas binaire de la figure 1, *le tas binaire résultant sera alors celui représenté à la figure 5.*

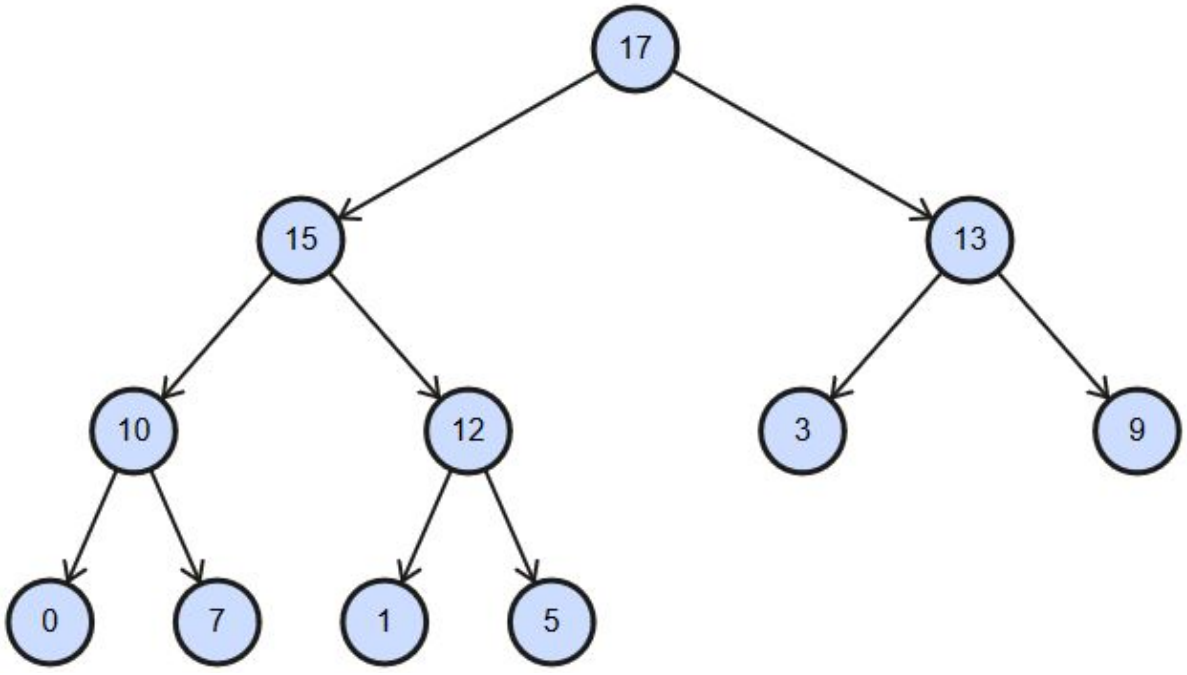

Figure 5 : tas binaire après retrait de la racine (19) du tas de la figure 1

2.3) Tri par tas

En associant les différentes opérations élémentaires de la section 2.2, il est possible de réaliser un algorithme de tri (dans notre cas, trier les entiers d'un tableau dans un ordre croissant).

L'objectif de cette section est donc de concevoir et implémenter en C des algorithmes qui réalisent les opérations décrites à l'étape 5 et 6.

Étape 5 : En utilisant les opérations élémentaires décrites précédemment, et à partir d'un tableau d'entiers donné en entrée de l'algorithme, réaliser à la suite les actions suivantes :

- Créer un tas binaire contenant les valeurs de ce tableau.
- Utiliser ce tas binaire pour extraire les différentes valeurs triées.
- Renvoyer un tableau trié dans l'ordre croissant comportant les valeurs du tableau donné en entrée.

Étape 6 : Il n'est en fait pas nécessaire d'utiliser un tableau intermédiaire pour réaliser le tri ; il est possible de travailler directement sur le tableau donné en entrée (on parle de **tri en place**). Adapter l'algorithme de l'étape 5 pour réaliser un tri en place.

Indication : partez du tableau donné en entrée, transformez ce tableau en tas (en travaillant directement sur les valeurs de ce dernier) et appliquez la même logique que dans l'étape 5 pour réaliser le tri sur le tas ainsi obtenu.

3) Instructions et critères d'évaluation

Vous devrez réaliser ce projet par groupe de deux, sur une période d'à peu près deux semaines. À la fin, vous présenterez votre travail à votre enseignant sous la forme d'un rapport de quelques pages et d'une soutenance.

Concernant les livrables et les deadlines, le code source final et le rapport devront être envoyés (par les moyens indiqués par vos enseignants) avant la date finale indiquée par vos enseignants.

Votre projet sera évalué selon différents critères détaillés ci-dessous (à peu près dans l'ordre décroissant, les éléments en haut de cette liste sont plus importants que ceux en bas) :

- L'aspect fonctionnel : votre programme doit fonctionner et toutes les fonctionnalités attendues décrites dans la section précédente doivent être implémentées.
- La qualité technique de votre projet. C'est-à-dire la propreté et la lisibilité de votre code source (soyez organisé et cohérent) ainsi que la qualité de vos algorithmes.
- La qualité de votre prestation lors de la soutenance. Essayez d'être clair et intéressant pour vos spectateurs.
- Intérêt, pertinence et qualité éditoriale de votre rapport. Il doit avoir une longueur de quelques pages et doit détailler votre approche et votre réalisation. En particulier, un

retour est attendu sur les fonctionnalités que vous avez implémentées, les algorithmes importants que vous avez conçus, les difficultés rencontrées et les améliorations possibles.
- L'interface utilisateur et l'expérience utilisateur de votre programme. Une interaction ergonomique et agréable sera valorisée.
- Toutes les améliorations supplémentaires que vous aurez réalisées. Plusieurs idées sont proposées dans la section suivante.

4) Pour aller plus loin…

Après avoir réalisé les fonctionnalités attendues, vous pouvez — seulement si vous le souhaitez, cela n'est pas obligatoire — continuer d'ajouter plus de fonctionnalités et d'améliorations pour rendre votre projet encore mieux.

Voici quelques idées (cette liste n'est ni exhaustive, ni particulièrement triée, n'hésitez pas à être imaginatif) :

- Nous avons traité ici du cas des tas binaires en nous intéressant au tas-max. Il pourrait être intéressant, à des fins de complétude, d'implémenter également les différentes opérations pour un tas-min (dont la logique sera par ailleurs très similaire).
- En utilisant à la fois un tas-min et un tas-max, calculer de façon performante la médiane pour un flux d'entiers. C'est-à-dire que l'utilisateur rentre au fur et à mesure des entiers arbitraires à ajouter dans un tableau, et vous devez renvoyer — à chaque étape — la médiane de ce tableau.
- Dans ce projet, nous avons étudié l'algorithme de tri par tas. Mais il existe une multitude d'autres algorithmes permettant de réaliser le tri avec des logiques différentes et présentant d'autres avantages ou inconvénients. Il serait intéressant d'effectuer des recherches pour établir une liste de différents algorithmes de tri usuels en les comparant mutuellement.