Tas et files de priorités

OPTION INFORMATIQUE - TP n° 3.2 - Olivier Reynet

À la fin de ce chapitre, je sais :

- ☞ définir un tas-min et un tas-max
- ☞ trier un tableau pas tas
- ☞ utiliser un tas pour créer une file de priorité
- ☞ appliquer les tas à l'algorithme de Dijsktra
- ☞ programmer impérativement en OCaml

A Implémentation d'un tas max et tri par tas

A1. Créer une variable `heap_test` de type `Array` contenant les entiers de 1 à 10 dans l'ordre décroissant. On génèrera automatiquement cette variable (pas in extenso). Est-ce un tas ? Si oui, de quel type ?

A2. Écrire une fonction de signature `swap : 'a array -> int -> int -> unit` qui échange la place de deux éléments dans un type `Array`. Tester cette fonction sur le tableau `heap_test`.

Comme en OCaml les `Array` sont indexés à partir de 0, on choisit de placer la racine du tas en 0. Puis, si le père est k alors les fils seront en $2k+1$ et $2k+2$. Si un fils est en k, alors le père est en $\lfloor \frac{k-1}{2} \rfloor$.

Il faut noter qu'un tas peut ne pas être rempli. Dans ce cas, on prendra soin de relever l'indice de la première case non remplie.

A3. Écrire une fonction de signature `up : 'a array -> int -> unit` qui, si cela est possible, fait monter un élément d'après son indice dans le tas tout en préservant la structure du tas.

A4. Écrire une fonction de signature `down : 'a array -> int -> int -> unit` qui, si cela est possible, fait descendre un élément dans un tas max tout en préservant la structure du tas. Le premier entier de la signature est l'indice de la première case non remplie du tas, le second l'indice de l'élément à faire descendre.

On cherche à créer un tas de deux manières différentes :

- soit en considérant que le premier élément du tableau est un tas que l'on fait grossir avec les éléments de droite du tableau,
- soit en considérant que les feuilles sont des tas à un élément que l'on fait grossir au fur avec les éléments de gauche du tableau. On remarque dans ce cas que quelle que soit la taille n du tas, l'élément d'indice $n/2$ est toujours la première feuille. On va donc insérer les $n/2$ premiers éléments dans leurs feuilles et faire apparaître la structure de tas.

A5. Effectuer à la main la transformation du tableau `[|0; 1; 2; 3; 4; 5; 6; 7; 8; 9|]` en tas-max par les deux méthodes.

A6. Écrire une fonction de signature `heap_make_up : 'a array -> unit` qui transforme un tableau en un tas en faisant monter les éléments. Quelle la complexité de cette fonction ?

A7. Écrire une fonction de signature `heap_make_down : 'a array -> unit` qui transforme un tableau en un tas en faisant descendre les éléments. Quelle est la complexité de cette fonction [1] ?

A8. Écrire une fonction de signature `heap_sort : 'a array -> unit` qui effectue le tri par tas d'un tableau. Quelle la complexité de cette fonction ?

A9. Tester le tri par tas sur un tableau de 10000 valeurs entières choisies aléatoirement entre 0 et 1000000.

B Implémentation d'une file de priorités

Pour implémenter une file de priorités à partir d'un tas, il est nécessaire de définir le type de données que contient la file, puis de programmer les opérations de base de la file de priorités. Les opérations sur une file de priorités sont :

1. créer une file vide,
2. insérer dans la file une valeur associée à une priorité (ENFILER),
3. sortir de la file la valeur associée la priorité maximale (ou minimale) (DÉFILER).

On considère une file de priorités dont les éléments sont des couples (v, p) et p est un entier naturel. **Plus p est faible, plus la priorité est élevée**.

Par ailleurs, on se dote des types et des exceptions suivantes :

```
type 'a qdata = {value: 'a; priority: int};;
type 'a priority_queue = {mutable first_free: int; heap: 'a qdata array};;
exception EMPTY_PRIORITY_QUEUE;;
exception FULL_PRIORITY_QUEUE;;
```

B1. De quel type est le tas nécessaire à la construction de cette file de priorités ?

B2. Écrire une fonction de signature `up : 'a array -> int -> unit` qui, si cela est possible, fait monter un élément d'après son indice dans le tas tout en préservant la structure du tas. On veillera à bien faire monter en fonction de la priorité.

B3. Écrire une fonction de signature `down : 'a array -> int -> int -> unit` qui, si cela est possible, fait descendre un élément dans un tas max tout en préservant la structure du tas. Le premier entier de la signature est l'indice de la première case non remplie du tas, le second l'indice de l'élément à faire descendre. On veillera à bien faire descendre en fonction de la priorité.

B4. Écrire une fonction de signature `make_priority_queue : int -> 'a * int -> 'a priority_queue` qui crée une file de priorité à n éléments initialisés à l'aide d'un type `qdata` donné en paramètre et dont le premier élément libre est le premier du tas.

B5. Écrire une fonction de signature `insert : 'a priority_queue -> 'a * int -> unit` qui enfile un élément dans la file de priorités. On veillera à préserver la structure du tas et à renvoyer une exception si l'opération n'est pas possible.

B6. Écrire une fonction de signature `get_min : 'a priority_queue -> 'a` qui renvoie l'élément `value` de priorité maximale. On veillera à préserver la structure du tas et à renvoyer une exception si l'opération n'est pas possible.

1. On suppose que le nombre de nœuds ayant la hauteur h est majoré par $\lceil \frac{n}{2^{h+1}} \rceil$