Annexe D

Tas et files de priorités

Hugo Salou MPI*

Dernière mise à jour le 8 mars 2023

L'objectif d'une file de priorité est de récupérer l'élément de priorité minimale. On organise cette structure de données sous forme d'un arbre tournois. [1] Un arbre tournois est un arbre dont la priorité d'un nœud est supérieur à celle de ses fils. On impose une structure supplémentaire, l'arbre doit être parfait : l'arbre est complet jusqu'à l'avant dernier niveau, où il est replis à gauche. On définit plusieurs opérations sur cette file de priorité (de type `fp`, où les éléments sont de type `elem`) :

— `insérer` : `fp` $\to$ `elem` $\to$ `fp` qui insère un élément,

— `lire_min` : `fp` $\to$ `elem` qui récupère l'élément de priorité minimale,

— `supprimer_min` : `fp` $\to$ `fp` qui supprime l'élément de priorité minimale,

— (`diminuer_priorité` : `fp` $\to$ `elem` $\to$ `fp`), [2]

— `créer` : ( ) $\to$ `fp`.

On définit un type `btree`, représentant un arbre binaire, et on implémente les opérations ci-dessous en OCaml.

```
type 'a btree =
| Node of 'a * 'a btree * 'a btree
| Empty
```

Code 1 – Définition du type `btree`

Pour l'opération `créer`, on retourne `Empty` (cela donne une complexité en $\Theta(1)$). Pour l'opération `insérer`, on insère l'élément comme feuille (de manière à conserver la propriété de l'arbre parfait), et on inverse le nœud avec son parent jusqu'à ce que la propriété soit vérifiée ($\Theta(\log_2 n)$). Pour l'opération `lire_min`, on lit la racine ($\Theta(1)$). Pour l'opération `supprimer_min`, on permute la racine et le dernier nœud (*i.e.* le nœud le plus à droite de hauteur maximale), et on restore la structure d'arbre tournois en permutant un nœud et son fils de valeur minimale, et en répétant ($\Theta(\log_2 n)$). Pour trouver le dernier nœud, on garde en mémoire cet emplacement. On peut aussi implémenter cet algorithme avec un tableau (▷ TP), ou avec une liste triée (mais la complexité est moins bien).

1. Un arbre tournois n'est pas un arbre binaire de recherche.
2. Cette opération est parfois omise car trop compliquée à implémenter.