Examen de TP : Pairing Heaps

Durée : 3 heures.

1 Avant-propos

Ce TP est à rendre via TOMUSS à l'issue des 3h de la séance. Il est à réaliser seul·e. Vous êtes autorisés à utiliser tous les outils de la librairie standard (conteneurs, algorithmes, ...). Vous êtes également autorisés à vous rendre en ligne pour chercher de la documentation. Vous n'êtes par contre pas autorisés à publier votre code en ligne ou récupérer celui d'autrui.

Le sujet est probablement trop long pour les 3h de la séance. Ne vous affolez pas, allez le plus loin possible, le barème sera déterminé en fonction des travaux rendus.

2 Archive de base

L'archive qui vous est fournie est munie d'un Makefile. Ce Makefile est muni du nécessaire pour compiler votre code, mais aussi d'une règle pour générer l'archive de votre projet. Vous pouvez ainsi utiliser la commande

```
make archive
```

pour créer l'archive. Cette commande empaquetera tous les fichiers cpp et hpp présents de le répertoire du Makefile. Vous pourrez ensuite soumettre votre archive sur TOMUSS.

Dans cette archive, vous trouverez :
- un fichier de test ;
- des fichiers `noeud.[hc]pp` pour décrire les noeuds ;
- des fichiers `pheap.[hc]pp` pour décrire l'arbre.

Les fichiers `hpp` contiennent une interface à respecter pour que le fichier de test fonctionne. Vous êtes libre de rajouter tous les attributs et toutes les méthodes que vous jugerez utiles aux classes.

3 Pairing Heaps simplement chaînés

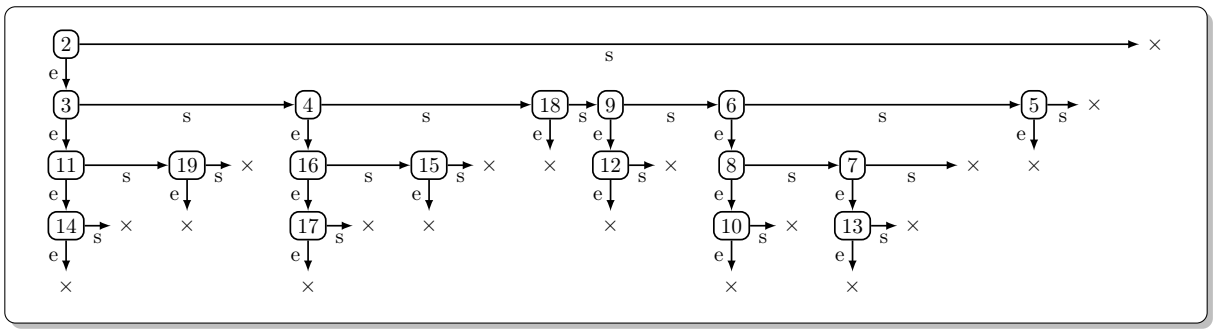
Un pairing heap est une structure de données pour encoder une file à priorités via un arbre. Cette structure offre une complexité O(1) pour l'insertion, la fusion de deux files et la consultation de l'élément le plus prioritaire. Le retrait de l'élément le plus prioritaire est réalisé en $O(\log n)$. Elle permet également la fusion de deux files à priorité.

3.1 Principe

Un pairing heap est un arbre tel que chaque nœud a une clé plus petite que tous les nœuds en dessous de lui, comme pour un tas binaire. Cet arbre n'est pas binaire : chaque nœud peut avoir un nombre variable d'enfants, chaînés entre eux à la manière d'une liste chaînée. Ainsi chaque nœud de l'arbre contient :
- une clé définissant la priorité ;
- une valeur associée à la clé ;
- l'adresse du nœud suivant (s) dans la liste de ses frères ;
- l'adresse du premier de ses enfants (e) ;

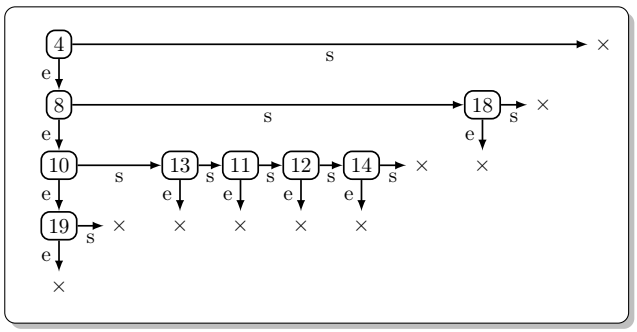
Vous trouverez également dans le code fourni une adresse supplémentaire pour le noeud précédent. Ne vous en préoccupez pas pour l'instant, cette adresse sera utilisée dans la seconde partie pour le double chaînage.

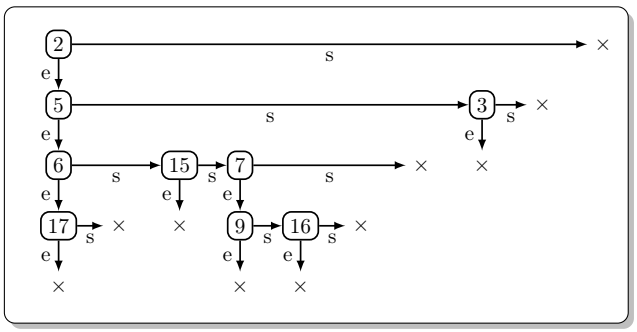
Dans tous les schémas qui suivent, seule la clé est représentée. La valeur est associée à la clé, mais n'intervient pas du tout dans la structure de l'arbre.



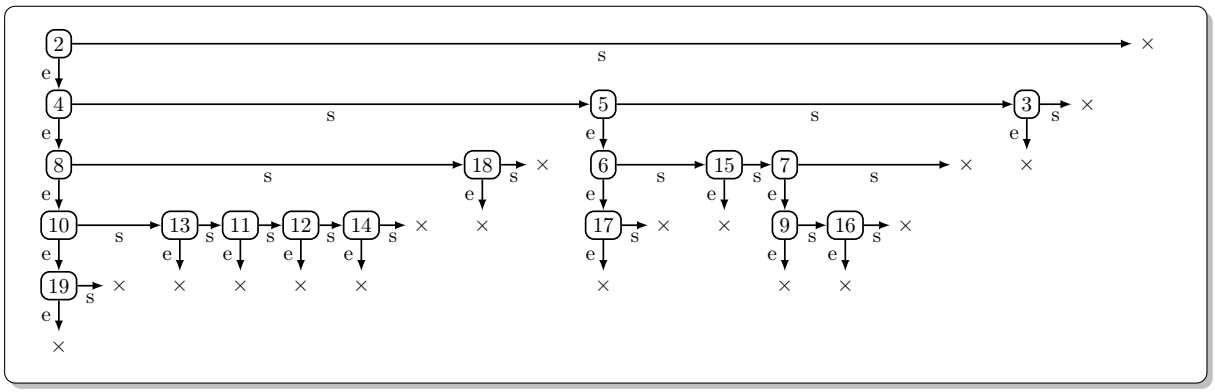
3.2 Fusion

L'opération centrale des pairing heaps est la fusion de deux arbres. Cette opération consiste à comparer les racines des deux arbres, puis à inserer l'arbre ayant la plus grande racine en tête des enfants de la racine de l'autre arbre. Par exemple si les deux arbres sont les suivants :





Les racines des arbres sont 2 et 4, et $2 < 4$. L'arbre de racine 4 est donc inséré en tête des enfants de l'arbre de racine 2, pour donner au final



3.3 Insertion

L'insertion n'est qu'un cas particulier de fusion : il suffit de créer un nouveau nœud pour le couple clé – valeur à insérer, puis de fusionner l'arbre initial avec de nœud.

3.4 Suppression du minimum

Le minimum étant à la racine de l'arbre, il est facile de consulter sa clé et sa valeur. Pour le supprimer, il est par contre nécessaire de déterminer le minimum des enfants pour obtenir la nouvelle racine. La solution naïve pour réaliser cette action consiste à prendre le premier enfant, puis itérativement fusionner les autres enfants avec cet arbre pour obtenir l'arbre final. Si la liste des enfants est longue, cette opération sera coûteuse, et si le premier enfant a la clé minimale, la liste de ses enfants sera étendue avec tous ses frères, ce qui la rendra encore plus longue. La complexité de cette opération est la longueur de la liste. Pour améliorer les requêtes sur ces arbres, il est donc nécessaire de contrôler la longueur des listes d'enfants.

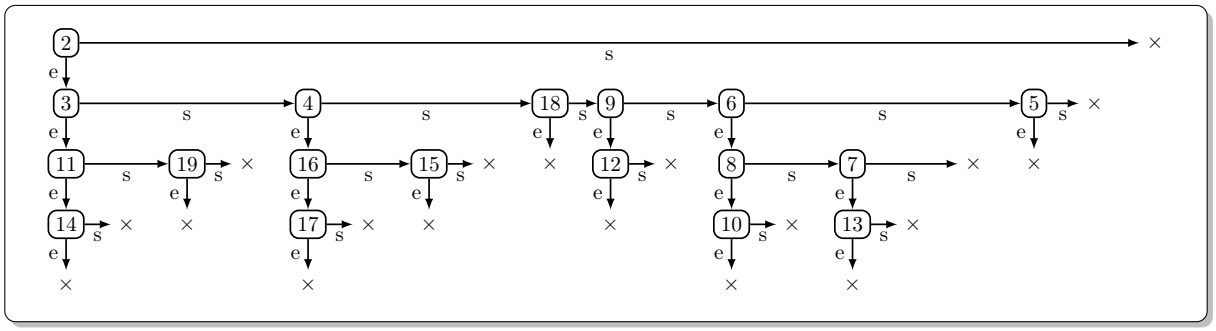
Le mot « pairing » du nom de la structure provient de la stratégie utilisée pour la suppression de minimum. Plutôt que de fusionner d'un coup tous les enfants de la racine supprimée, cette stratégie réalise cette opération en deux temps :

1. les k enfants sont fusionnés deux par deux pour former $k/2$ arbres ;
2. les $k/2$ arbres sont fusionnés ensembles pour former l'arbre final.

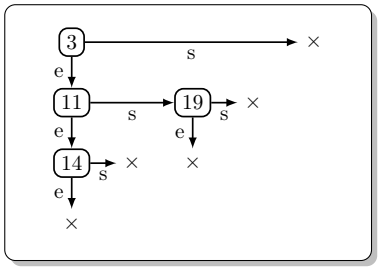
Dans le cas d'un nombre d'enfants impair, l'un des enfants est laissé seul lors de la première étape, et est fusionné avec les autres lors de la seconde étape.

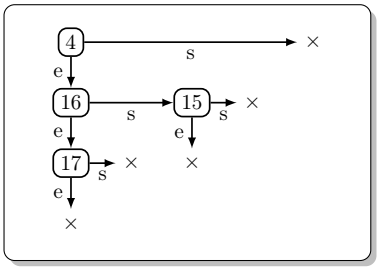
Pour éviter d'avoir des tableaux intermédiaires pour stocker tout ça, il est possible de ne stocker qu'un arbre courant, initialisé à la fusion des deux premiers enfants. Vous pouvez ensuite considérer les deux enfants suivants, les fusionner entre eux, puis à l'arbre courant, et ainsi de suite. Vous pouvez aussi avoir une approche récursive qui permet de faire le tout en quelques lignes.

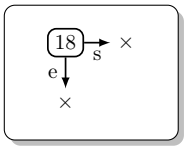
Par exemple sur l'arbre du premier exemple

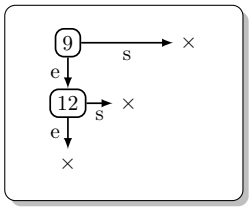


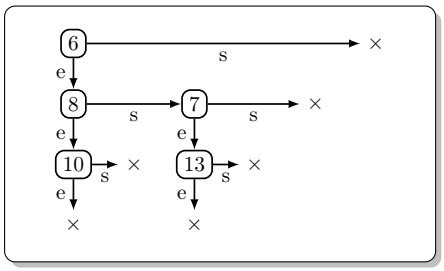
après suppression de la racine les enfants sont

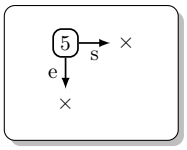




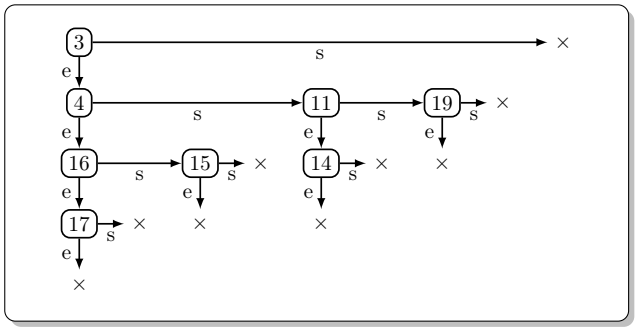


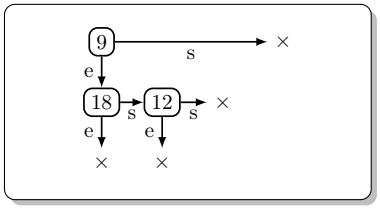


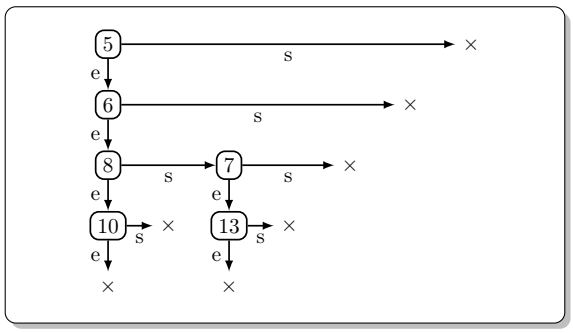




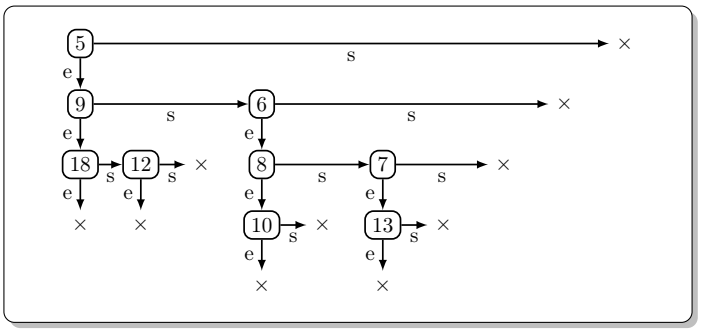
Ces enfants sont fusionnés deux à deux pour créer trois arbres



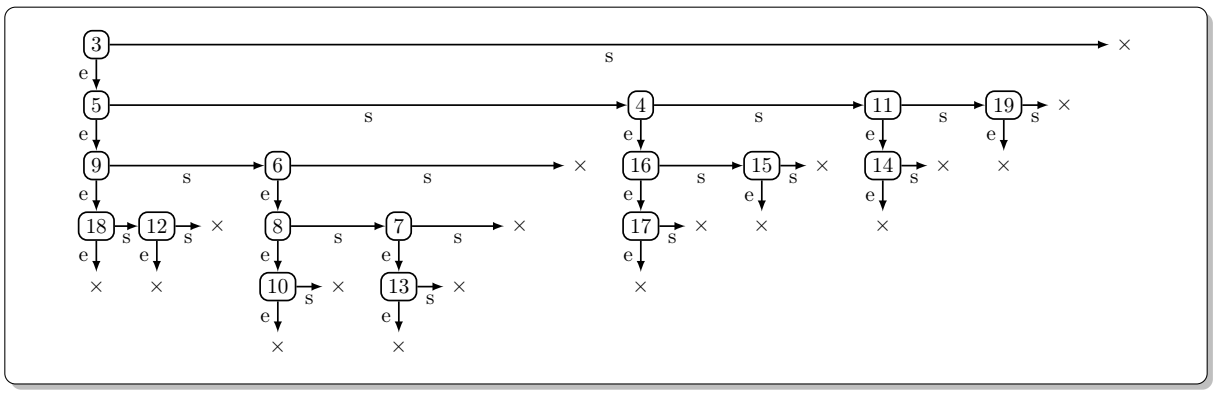




Ces paires sont enfin fusionnées pour former l'arbre final.



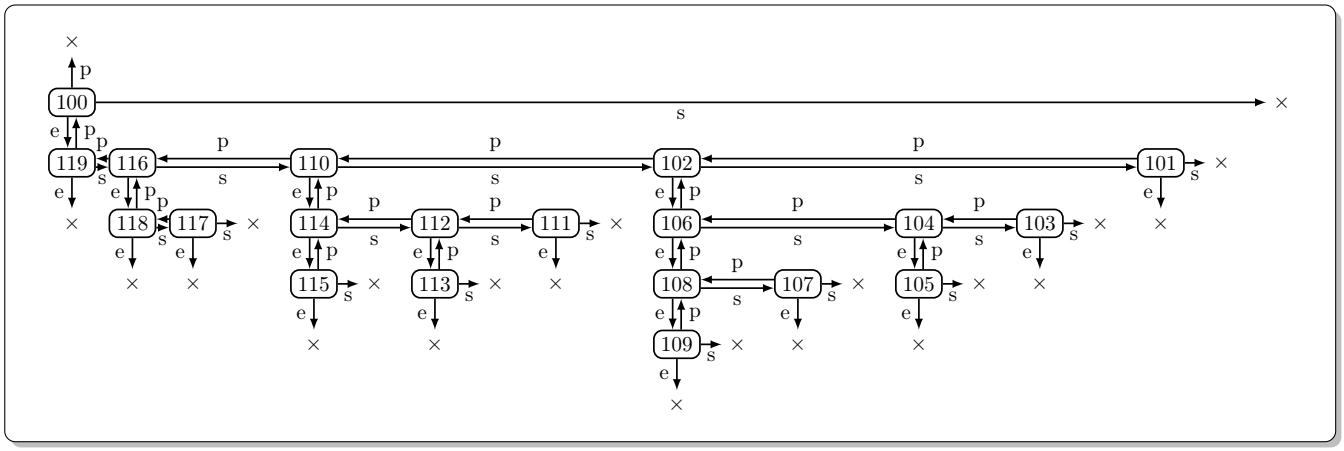
puis



4 Pairing Heaps doublement chaînés

Il est possible d'améliorer la structure pour permettre de retirer des nœuds en plein milieu de la structure. Il est ainsi possible de les retirer, de changer leur clé, puis de les réinsérer. Pour réaliser cette action avec les pairing heaps simplement chaînés, le seul problème est que lors du retrait d'un nœud, il est nécessaire de faire en sorte que son prédécesseur dans la liste de ses frères, ou son parent s'il est en tête de liste doit être mis à jour. Les pointeurs actuellement stockés dans la structure ne permettent pas de déterminer rapidement le prédécesseur ou le parent.

Le plus efficace pour retrouver le prédécesseur ou le parent est simplement de stocker dans chaque nœud un pointeur supplémentaire pour indiquer le prédécesseur, ou le parent dans le cas du premier enfant. La différence se fait en regardant à partir du prédécesseur si le nœud est son enfant ou son suivant.



Avec ce double chaînage, il est alors possible de coder une fonction qui retire un nœud fourni par son adresse. Une suppression de minimum est alors réalisée sur le sous-arbre enraciné à ce nœud, et le sous-arbre réorganisé est à nouveau fusionné avec le reste de l'arbre.

Le changement de clé d'un nœud est réalisé en supprimant le nœud puis en le réinsérant.

Pour ne pas risquer de casser ce qui marche déjà en vous emmêlant les pointeurs, copiez le fichier des pairing heaps simplement chaînés pour le sauvegarder, et garder cet acquis. Lorsque vous atteignez cette partie, décommentez le code permettant de valider le double chaînage dans `pheap.cpp`, méthode $\texttt{valider}_{n}oeud$.