Licence Informatique **J1IN601 – Algorithmique 3**
Université Bordeaux 1

TP noté avril 2012

Modalités du TP

L'archive fournie contient l'ensemble des fichiers nécessaires pour le TP. Il est interdit d'ajouter un fichier et le seul fichier que vous pouvez modifier est le fichier `splay.c`. Il est notamment interdit de modifier le fichier `splay.h`.

Chaque étudiant doit faire le TP seul. Vous pouvez vous inspirer des fichiers produits lors des différents TP ainsi que des corrections fournies. Toute autre source d'« inspiration » (voisins, internet...) et strictement interdite.

À la fin de l'épreuve vous enverrez un email à votre chargé de TP usuel (`frederic.mazoit@labri.fr`, `kaninda.musumbu@labri.fr` ou `sylvain.salvati@labri.fr`), dont le sujet sera « TP noté d'algorithmique 3 » et dont le corps contiendra vos nom et prénom ainsi que le fichier `splay.c` que vous aurez produit. **Tout écart dans le format de l'email et tout email reçu après l'heure de fin de l'épreuve seront sanctionnés.**

Présentation du sujet

Le but de ce TP est d'implémenter une variante d'arbre binaire de recherche appelée « splay tree » ou arbre évasé en français.

Cette structure possède une fonction `splay` qui prend un arbre binaire de recherche `T` et une clef `key` et telle que

- `splay(T, key)` donne un arbre `T'` qui représente le même ensemble de clés que `T`;
- si `key` est la clé d'un objet qui figure dans `T`, alors `key` est la clé de la racine de `T'`;
- sinon, la clef de la racine de `T'` est soit `key-` soit `key+`, où `key-` est la plus grande clé dans `T` inférieure a `key`, et `key+` est la plus petite clé supérieure à `key`.

La fonction splay est implémentée à l'aide de rotations.

Toutes les autres opérations sont implémentées en utilisant cette fonction splay.

Sujet

Dans ce sujet, on demande d'implémenter plusieurs fonctions sur ces « splay trees ». Pour chaque fonction `riri` à implémenter, le fichier `prof_splay.o` contient une fonction `prof_riri` équivalente. Si besoin, vous pourrez utiliser ces fonctions.

1. Implémenter `splay_create(f)`.
2. Implémenter `splay_destroy(f)`.
3. Implémenter `splay_find(m, key)`.
   Indication : Vous pourrez commencer par faire un appel à `splay_splay(m, key)`. Ensuite, si la clef apparaît dans l'arbre, alors on sait qu'elle se trouve au niveau de la racine.
4. Implémenter `splay_insert(m, obj)`.
   Indication : On crée un nœud contenant `obj` dont le sous-arbre gauche (`left`) contient les clefs inférieures à celle de `obj` et dont le sous-arbre droit (`right`) ne contient que des clefs supérieures à celle de `obj`. On peut facilement obtenir `left` et `right` à partir du résultat de `splay_splay(m, (m->f)(obj))`.

5. En vous inspirant de la question précédente, donner une implémentation de la fonction `splay_coupe` dont le prototype est le suivant :

   ```
   splay splay_coupe(splay A, int key);
   ```

   et qui
   – rend un splay contenant les clefs de `A` strictement supérieures à `key` ;
   – retire de `A` les clefs strictement supérieures à `key`.

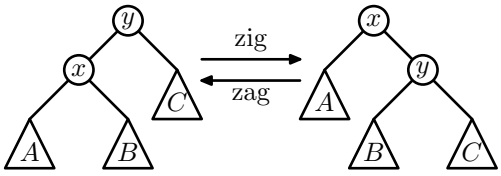
6. Implémenter `splay_delete(m, key)`.
   Indication : Dans `splay_find` et `splay_insert`, on a utilisé `splay_splay` pour ne travailler qu'au niveau de la racine. Vous pouvez utiliser une stratégie similaire pour `splay_delete` en faisant au besoin plusieurs appels à `splay_splay`.

7. En vous inspirant de la question précédente, donner une implémentation de la fonction `splay_union` dont le prototype est le suivant :

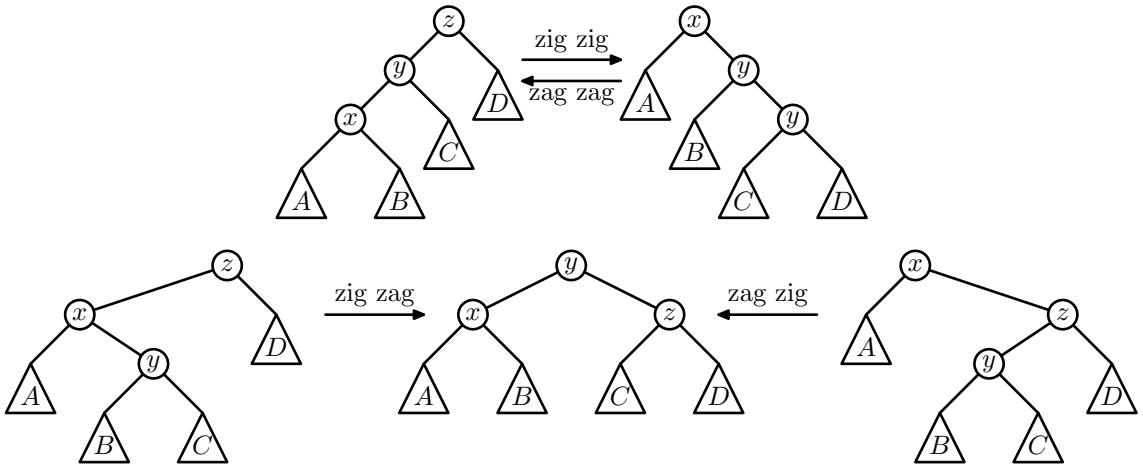
   ```
   splay splay_union(splay A, splay B);
   ```

   et qui
   – si les clefs de `A` sont inférieures aux clefs de `B`, crée un arbre contenant des objets de `A` et de `B` en détruisant `A` et `B` ;
   – rend `NULL` sinon.
   On pourra utiliser avantageusement la bibliothèque `limits.h` qui défini notamment les constantes `INT_MIN` et `INT_MAX` qui sont respectivement les plus petit et les grands entiers représentables par le type `int`.

8. Donner une implémentation première implémentation (appelée `splay_cheap_splay`) de la fonction splay en n'utilisant que des rotations simples (zig et zag).



9. La précédente implémentation n'a pas un bon comportement théorique. Son coût amorti est plus important que $O(\log n)$. Sleator et Tarjan on montré que si on utilise au plus une rotation simple (zig ou zag) et sinon que des doubles rotations (zig zig, zig zag, zag zig et zag zag), alors on obtient un bien meilleur comportement avec notamment un coût amorti en $O(\log n)$.



   Implémenter cette stratégie dans la fonction `splay_splay`.