TECHNIQUES ALGORITHMIQUES ET PROGRAMMATION

Algorithme A* – Tas binaire

1 Tas binaire minimum

On représentera un tas binaire (*heap*) par la structure suivante :

```
typedef struct{
  void* *array;
  int n, nmax;
  int (*f)(const void*, const void*);
} *heap;
```

où `array[]` est le tableau de stockage des objets du tas (chacun d'eux étant un `void*` stocké à partir de l'indice 1, l'indice 0 n'étant pas utilisé), `n` est le nombre d'objets stockés dans le tas, `nmax` est le nombre maximum d'objets stockables dans le tas, et enfin `f()` la fonction de comparaison de deux objets permettant de définir la notion de tas minimum (fonction similaire à `qsort()`).

Question 1. *Rappelez la définition d'un tas binaire ainsi que le principe de l'implémentation par un tableau. En particulier, précisez pour un élément d'indice i du tableau quels sont les indices de ses fils et de son père.*

Question 2. *Rappelez le principe de l'ajout d'un élément dans le tas. Donnez sa complexité en fonction du nombre n d'éléments du tas.*

Question 3. *Même question pour la suppression du minimum.*

On suppose qu'on utilise un tableau dynamique pour `array[]` avec la stratégie suivante : lorsque le tas est plein, on double la taille du tableau (et qui est donc recopié). Initialement, le tas a une taille de 1.

Question 4. *Avec cette stratégie, donnez la complexité pour remplir un tas de taille n, c'est-à-dire la complexité totale pour réaliser n ajouts à partir d'un tas vide et de taille* 1.

2 En TP

Téléchargez les fichiers correspondant au TP à partir de la page de l'UE disponible ci-après :

http://dept-info.labri.fr/~gavoille/UE-TAP/

Vous aurez à compléter `heap.c` que vous testerez avec `./test_heap` après avoir compilé avec `make -B test_heap` toujours sur le même `Makefile`. Vous n'avez pas à modifier `heap.h` mais simplement lire les spécifications des fonctions qui sont à programmer dans `heap.c`. Il est préférable de placer les fichiers `heap` dans le même répertoire contenant `Makefile` et les fichiers `tools`.