Devoir surveillé d'informatique

Consignes

- Les programmes demandés doivent être écrits en C ou en OCaml. Dans le cas du C, on suppose que les librairies standards usuelles (`<stdio.h>`, `<stdlib.h>`, `<stdbool.h>`, `<stdassert.h>`, ...) sont déjà importées.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ Exercice 1 : *Questions de cours*

On considère la fonction d'Ackerman $a : \mathbb{N} \times \mathbb{N} \mapsto \mathbb{N}$ définie par par :

$$\begin{cases} a(0,m) &= m+1 \\ a(n,0) &= a(n-1,1) \text{ si } n > 0 \\ a(n,m) &= a(n-1,a(n,m-1)) \text{ si } n > 0 \text{ et } m > 0 \end{cases}$$

Q1– Calculer $a(1,2)$

Q2– Ecrire en OCaml une fonction `ack: int -> int -> int` qui prend en argument deux entiers positifs n et m et renvoie $a(n,m)$.

Q3– Prouver la terminaison de la fonction `ack` on précisera soigneusement le variant et la relation d'ordre bien fondée utilisée.

❑ Exercice 2 : *Recherche des k premiers maximums d'une liste*

Les fonctions demandées dans cet exercice doivent être écrites en langage OCaml.

On s'intéresse au problème de la recherche des k premiers maximums d'une liste de n entiers. Dans toute la suite de l'exercice on supposera que la liste est *non vide* : $n > 0$ et qu'on extrait moins de maximums qu'il n'y a d'éléments dans la liste c'est à dire que $k \leqslant n$. On cherche donc à écrire une fonction `kmax : 'int list -> int -> 'int list` qui renvoie la liste des `k` premiers maximums de la liste donnée en argument. Par exemples :

— `kmax [2; 5; 1; 8; 3; 0; 4] 2` renvoie `[8, 5]`

— `kmax [7; 8; 8; 1; 6; 3; 2; 9] 3;;` renvoie `[9; 8; 8]`

— `kmax [1; 0; 1; 2; 4]` 4 renvoie `[4; 2; 1; 1]`

■ Partie I : Résolution par recherche successive des maximums

Q4– Ecrire une fonction `extrait_max : int list -> int * int list` qui prend en argument une liste et renvoie le couple composé du maximum de cette liste et de cette liste privée d'un de ses maximums. Par exemple `extrait_max [2; 6; 4; 6; 5]` renvoie `(6, [2; 4; 6; 5])`. On pourra procéder par correspondance de motif et traiter le cas de la liste vide par un `failwith`.

Q5– Donner en la justifiant brièvement la complexité de la fonction `extrait_max`.

Q6– On suppose qu'on extrait les `k` premiers maximums d'une liste à l'aide de `k` applications de la fonction `extrait_max`. Quelle est la complexité de cet algorithme ? (on ne demande *pas* de le programmer)

■ Partie II : Résolution par un tri

Q7– Ecrire une fonction `kpremiers int list -> int -> int list` qui prend en argument une liste et un entier `k` et renvoie la liste composée des `k` premiers éléments de `lst`.

Q8– On propose d'écrire la fonction `kmax` en triant la liste par ordre décroissant puis en prenant ses k premiers éléments. En supposant que l'algorithme de tri utilisé a une complexité en $\mathcal{O}(n \log n)$. Donner la complexité de ce nouvel algorithme (on ne demande *pas* de le programmer).

■ Partie III : Résolution en utilisant un tas

Dans la suite on suppose que la structure de données de tas d'entiers (type `int`), est *déjà implémentée* par un type `int tas` sur lequel on dispose des fonctions suivantes :

— `cree_tas : int -> tas` qui prend en argument un entier `cap` et renvoie un tas binaire vide de capacité `cap`.

— `donne_taille : tas -> int` qui prend en argument un tas et renvoie sa taille (le nombre d'éléments actuellement stocké dans le tas).

— `insere : int -> tas -> unit` qui insère une nouvelle valeur dans le tas. Cette fonction échoue lorsque le tas est plein.

— `donne_min : tas -> int` qui renvoie la valeur minimale contenu dans le tas *sans modifier le tas.*

— `extrait_min : tas -> int` qui renvoie (en le supprimant du tas) le minimum du tas.

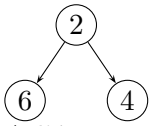
Q9– Rappeler en les justifiant rapidement les complexités des opérations `insere` et `extrait_min` si on suppose que l'implémentation de la structure de tas est réalisée grâce à un tableau.

Q10– Afin d'extraire les `k` premiers éléments d'une liste de taille `n`, on propose créer un tas de taille `k` puis de parcourir récursivement la liste, pour chaque élément rencontré :

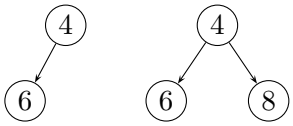
— si le tas n'est pas plein on y insère l'élément

— sinon, on compare l'élément avec le minimum du tas, s'il est plus grand on supprime le minimum du tas et on insère l'élément dans le tas.

Par exemple, si on veut extraire les 3 premiers maximums de la liste `[4; 6; 2; 8; 3; 7; 1; 9; 5]`, après l'insertion des trois premiers éléments, le tas est :



A l'étape suivante, 8 étant plus grand que 2 (le minimum du tas), on extrait 2 du tas et on y insère 8 ce qui donne :



Q11– Poursuivre le déroulement de cet algorithme en faisant figurer comme ci-dessus les étapes de l'évolution du tas.

QuestionProuver que cet algorithme est correct (on pourra utiliser prouver l'invariant valable pour tout `i>=k` : « le tas contient les `k` premiers maximums du sous tableau compris entre les indices `0` et `i-1` »).

a) Ecrire une fonction `int* kmax_heap(int t[], int n, int k)` qui extrait les `k` premiers maximum du tableau `t` de taille `n` et les renvoie dans un tableau de taille `k` en utilisant cet algorithme. Afin de manipuler le tas, on utilisera les fonctions déjà disponibles sur la structure de tas et dont les signatures sont données en début de partie.

```c
int *kmax_heap(int t[], int n, int k)
{
    heap mh = make_heap(k);
    int *mk = malloc(sizeof(int) * k);
    int temp;
    for (int i = 0; i < k; i++)
    {
        insert_heap(t[i], &mh);
    }
    for (int i = k; i < n; i++)
    {
        if (t[i] > mh.tab[0])
        {
            temp = getmin(&mh);
            insert_heap(t[i], &mh);
        }
    }
    for (int i = 0; i < k; i++)
    {
        mk[i] = mh.tab[i];
    }
    return mk;
}
```

b) Donner en la justifiant la complexité de ce nouvel algorithme en fonction de k et n.