Sujet CCINP : `SET COVER` et arbres de Braun (OCaml)

Exercice A

On considère le problème d'optimisation `SET COVER-opti` suivant :

$$\begin{cases} \textbf{Entrée} : \text{Un ensemble } E \text{ fini et } E_1, ..., E_n \text{ des sous ensembles de } E. \\ \textbf{Solution} : \text{Une couverture de } E, \text{ i.e. un sous ensemble } I \subset [\![1, n]\!] \text{ tel que } \bigcup_{i \in I} E_i = E. \\ \textbf{Optimisation} : \text{Minimiser } |I| \text{ (qu'on appelle } \textit{taille de la couverture}). \end{cases}$$

1. Décrire le problème de décision `SET COVER` associé à `SET COVER - opti`.
2. Montrer que le problème `SET COVER` est dans NP.

On dit qu'un sous-ensemble X de sommets d'un graphe G est une couverture par sommets de G si toute arête de G a au moins l'une de ses extrémités dans X. On admet dans la suite la NP-complétude du problème de décision `VERTEX COVER` décrit comme suit :

$$\begin{cases} \textbf{Entrée} \ : \text{Un graphe } G \text{ et un entier } k. \\ \textbf{Question} : \text{Existe-t-il une couverture par sommets de } G \text{ de taille inférieure à } k \, ? \end{cases}$$

Pour toute instance $I_V = (G = (S, A), k)$ de `VERTEX COVER`, on associe l'instance I_S de `SET COVER` définie par : $E = A$ et les sous ensembles de E sont les $E_s = \{a \in A \,|\, a \text{ est incidente à } s\}$ pour tout $s \in S$.

3. Montrer que si I_V est une instance positive de `VERTEX COVER` alors I_S est une instance positive de `SET COVER`. La réciproque est-elle vraie ? Justifier.
4. Montrer que le problème `SET COVER` est NP-complet.

On se contente donc de résoudre le problème `SET COVER-opti` de manière approchée via cet algorithme :

```
1 I ← ∅ et U ← E
2 tant que U ≠ ∅
3 |   si U n'intersecte aucun E_i alors
4 |   |   renvoyer "couverture impossible"
5 |   sinon
6 |   |   Déterminer i tel que E_i ∩ U est maximal
7 |   |   Ajouter i à I
8 |   |   U ← U \ E_i
9 renvoyer I
```

5. On considère l'instance suivante de `SET COVER-opti` : $E = [\![1, 8]\!]$, $E_1 = \{5, 6, 7, 8\}$, $E_2 = \{3, 4\}$, $E_3 = \{1\}$, $E_4 = \{2\}$, $E_5 = \{1, 3, 5, 7\}$, $E_6 = \{2, 4, 6, 8\}$. Montrer que cet algorithme glouton peut renvoyer une couverture de taille 4. Quelle est la taille de la couverture optimale ?
6. Montrer que cet algorithme est polynomial. On explicitera auparavant des choix de structures pour représenter I, U, E et les E_i.
7. En s'inspirant de la question 5, montrer que cet algorithme n'est pas un algorithme d'approximation à facteur constant pour `SET COVER-opti`.

Exercice B

Le sujet fournit un code compagnon fournissant le type décrit par l'énoncé ainsi qu'une partie des fonctions décrites ci-après. Il est à compléter avec les fonctions demandées.

Si t est un arbre, on note $|t|$ sa taille. Un *arbre de Braun* est un arbre binaire qui est :

- Soit l'arbre vide.
- Soit de la forme $N(r, g, d)$ avec r une étiquette et g et d deux arbres de Braun vérifiant

$$|d| \leq |g| \leq |d| + 1$$

On se limite dans ce sujet sujet au cas où les étiquettes des arbres de Braun sont des entiers et on représente de tels arbres à l'aide du type suivant en OCaml :

```
type braun = E | N of int * braun * braun
```

1. Pour tout $n \in [\![1, 6]\!]$, dessiner les squelettes des arbres de Braun de taille n. Que remarque-t-on ?

On admet que la hauteur h et la taille n d'un arbre de Braun vérifient la relation $h = \Theta(\log n)$.

2. Écrire une fonction `hauteur : braun -> int` calculant la hauteur d'un arbre de Braun. On demande une complexité logarithmique en la taille de l'arbre en entrée, qu'on justifiera brièvement.

Un *tas de Braun* est un arbre de Braun vérifiant de surcroît que l'étiquette d'un noeud est toujours inférieure ou égale à celles de ses fils éventuels. Dans la suite, on implémente des fonctions sur les tas de Braun de façon à les utiliser pour implémenter une file de priorité. Le code compagnon met par ailleurs à disposition deux tas de Braun `t1` et `t2` pour les tests.

3. Écrire une fonction `minimum : braun -> int` renvoyant l'élément minimal d'un tas de Braun. Dans le cas où l'arbre est vide, on renverra `max_int`.

4. Le code compagnon fournit une fonction `inserer : braun -> int -> braun`. Justifier que `inserer a x` est un tas de Braun et qu'il contient l'élément `x` en plus de ceux présents dans `a`.

On suppose que l'on dispose d'une fonction `fusionner : braun -> braun -> braun` prenant en entrée deux tas de Braun t et t' tels que $|t'| \leq |t| \leq |t'| + 1$ et qui renvoie un tas de Braun contenant les éléments de t et de t'.

5. Écrire alors une fonction `extraire_min : braun -> int*braun` prenant en entrée un tas de Braun t et qui renvoie un couple (m, t') avec m le minimum de t et t' un tas de Braun contenant les étiquettes de t moins une occurrence de m. On lèvera une exception en cas d'arbre vide.

On cherche à présent à implémenter les fonctions nécessaires au bon fonctionnement de la fonction la fonction `fusionner` fournie dans le code compagnon.

6. Écrire une fonction `extraire_element : braun -> int*braun` prenant en entrée un tas de Braun t et renvoyant (x, t') tels que x est un élément quelconque de t et t' est un tas de Braun contenant les éléments de t sauf une occurrence de x. On pourra s'inspirer du fonctionnement de la fonction `inserer` et on lèvera une exception si l'arbre est vide.

7. Écrire une fonction `remplacer_min : braun -> int -> braun` prenant en entrée un tas de Braun t et un entier x et renvoyant un tas de Braun contenant les éléments de t, moins une occurrence du minimum de t, plus une occurrence de x.

8. Expliquer brièvement la correction de la fonction `fusionner` fournie par l'énoncé.

9. Quelles sont les complexités de `minimum`, `inserer` et `extraire_minimum`, c'est-à-dire des opérations de base sur une file de priorité implémentée avec un tas de Braun ?