TP 4 : Modules Set et Map

Exercice 1 - Le module `Set`

Pour cette exercice, nous allons manipuler des ensembles d'entiers en OCaml par le biais du module `Set`. On peut définir leur type par :

```
module Int =
  struct
    type t = int
    let compare = fun x y -> x - y
  end ;;

module IntSet = Set.Make(Int) ;;
```

1.1 Écrire une fonction récursive `range: int -> int -> IntSet.t` qui, sur la donnée de deux entiers a et b, renvoie un ensemble contenant tous les entiers compris entre a et b (inclus).

1.2 Écrire une fonction `nub: int list -> int list` qui, sur la donnée d'une liste ℓ, renvoie ℓ sans les doublons. Par exemple, `nub [1;1;2;3;1;4]` retournera `[1;2;3;4]`

note : utiliser une fonction auxiliaire `nub_aux: int list -> IntSet.t -> int list` qui prend comme argument supplémentaire l'ensemble des entiers déjà vu.

1.3 Écrire une fonction `from_list: int list -> IntSet.t` qui, sur la donnée d'une liste ℓ, renvoie l'ensemble des entiers présents dans ℓ.

1.4 On définit la fonction `f: int list -> int list` par

```
let f = IntSet.elements (from_list l) ;;
```

Que fait la fonction `f` ? Quel est son coût pour une liste de taille n en entrée ?

Exercice 2 - Manipulation de graphes

Pour cet exercice, on utilsera le type `graph` vu en cours.

2.1 Écrire une fonction `add_vertex: int -> graph -> graph` qui, sur la donnée d'un entier `i` et d'un graphe `g`, renvoie un nouveau graphe constitué de `g` auquel on a ajouté un sommet numéroté `i` (sans successeurs).

2.2 Écrire une fonction `add_edge: int -> int -> graph -> graph` qui, sur la donnée de deux sommets `u` et `v`, et d'un graphe `g`, renvoie un nouveau graphe constitué de `g` auquel on a ajouté une arête de `u` à `v`.

2.3 Écrire une fonction `create_from_lists: int list -> (int * int) list -> graph` qui crée un graphe à partir d'une liste de sommets et d'une liste d'arêtes passées en arguments.

2.4 Écrire une fonction `is_successor` telle que `is_successor u v g` renvoie `true` si `v` est un successeur de `u` dans `g`, et `false` sinon.

2.5 Écrire `nb_vertices: graph -> int` renvoyant le nombre de sommets d'un graphe.

2.6 Écrire la fonction `nb_edges: graph -> int` renvoyant le nombre d'arêtes d'un graphe.