TP D'OPTION INFORMATIQUE 6
Révisions

1 Algorithmes de tri

1. Implémenter le tri par insertion, le tri rapide et le tri fusion, sur liste et sur tableau. Donner les complexités dans le meilleur et le pire cas.
2. Implémenter le tri par tas sur un tableau. Donner sa complexité.

2 Parcours de graphe

Implémenter un parcours de graphe en largeur puis en profondeur, sur un graphe représenté par tableau de listes d'adjacences. Donner la complexité.

3 Logique

1. Établir la table de vérité de $\phi = (p \leftrightarrow q) \vee (q \wedge r)$ et en déduire une FNC de ϕ.
2. Démontrer en déduction naturelle $(p \to q) \to (\neg p \vee q)$ et $(\neg p \vee q) \to (p \to q)$.

4 Arbres et écriture binaire

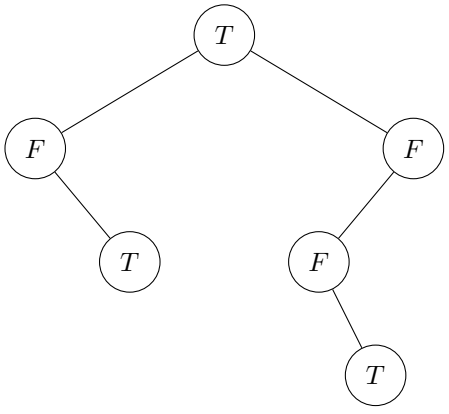
on se propose de représenter un ensemble fini d'entiers naturels par un arbre binaire contenant des booléens, de la façon suivante : pour déterminer si un entier n est dans l'ensemble, on parcourt son écriture binaire de droite à gauche pour en déduire un chemin depuis la racine. A chaque étape, un bit 0 indique de continuer avec le fils gauche, et un bit 1 indique de continuer avec le fils droit. Le nombre n est alors dans l'ensemble si le chemin décrit existe, et qu'il mène à un noeud de valeur booléenne `true`. On convient qu'une écriture binaire ne commence jamais par 0, et que 0 a une écriture vide, et correspond donc au booléen à la racine de l'arbre.
On définit ainsi le type Ocaml suivant :

```
type ens = Vide | Noeud of ens * bool * ens;;
```

1. Donner l'ensemble d'entiers représenté par l'arbre

   ```
   Noeud (Noeud (Vide, false, Noeud (Vide, true, Vide)), true,
    Noeud (Noeud (Vide, false, Noeud (Vide, true, Vide)), false, Vide))
   ```

   dont la représentation graphique est :



2. Dessiner l'arbre représentant l'ensemble $\{1; 2; 4; 7; 11\}$.

3. Implémenter les opérations suivantes de la structure d'ensemble :
   - `appartient : ens -> int -> bool` testant l'appartenance d'un nombre à l'ensemble ;
   - `ajoute : ens -> int -> ens` ajoutant un nombre à l'ensemble ;
   - `construit : int list -> ens` traduisant une liste de nombres en ensemble ;
   - `union : ens -> ens -> ens` réalisant l'union de deux ensembles.
4. Déterminer la complexité de la fonction `appartient` en fonction de l'entier cherché dans l'ensemble.
5. La représentation minimale d'un ensemble n'a jamais de feuille (ie de noeud dont les fils sont vides) de booléen `false`. Lorsqu'on supprime un élément d'un ensemble, il est possible de faire apparaitre de telles feuilles. Écrire une fonction `elaguer : ens -> ens` prenant en entrée un ensemble, et renvoyant le même ensemble élagué, ie n'ayant aucune feuille de valeur `false`.
6. Implémenter les opérations suivantes :
   - `retire : ens -> int -> ens` privant l'ensemble d'un élément, et élaguant le résultat ;
   - `intersection : ens -> ens -> ens` renvoyant l'intersection élaguée de deux ensembles.
7. Écrire une fonction `ens_vers_liste : ens -> int list` renvoyant la liste des entiers présents dans un ensemble.
8. Expliquer pourquoi dans cette représentation, un fils gauche ne contient jamais `true` à sa racine.
9. On en déduit que dans cette représentation, la moitié des booléens stockés ne porte aucune information. Proposer une variante de cette représentation dans laquelle chaque noeud correspond bien à un entier différent, et modifier l'implémentation des opérations en conséquence.