DS 1

Option informatique, deuxième année

Julien Reichert

Partie I : Graphes

Soit G un graphe non orienté dont les sommets sont les entiers de 1 à 8 représenté par liste d'adjacence ci-dessous.

Sommets	Sommets adjacents
1	(2,3,4)
2	(1,3,4)
3	(1,2,4)
4	(1,2,3,6)
5	(6,7,8)
6	(4,5,7)
7	(5,6,8)
8	(5,7)

On considérera que lors du parcours du graphe les sommets adjacents d'un sommet donné sont rencontrés dans le même ordre qu'ils sont listés dans le tableau précédent.

Question 1 : Représenter graphiquement le graphe correspondant à G.

Question 2 : Ce graphe est-il complet ? Est-il connexe ?

Question 3 : Donner la séquence des sommets de G obtenus lors du parcours en profondeur en commençant sur le sommet 1.

Question 4 : Donner la séquence des sommets de G obtenus lors du parcours en largeur en commençant sur le sommet 1.

Partie II : L'énigme de logique en plein milieu

Question 5 : Résoudre l'instance du jeu *Tracks* figurant en dernière page.

Le principe du jeu *Tracks* est de relier deux bords d'une grille par une unique voie ferrée sans boucle et sans qu'une autre voie ferrée ne figure sur la grille. Les indices figurant devant chaque ligne et chaque colonne indiquent le nombre de cases où la voie ferrée passe, et celle-ci traverse forcément les cases de manière horizontale ou verticale, jamais en diagonale. Quelques éléments de la voie ferrée figurent déjà pour démarrer (et aussi pour fixer les idées).

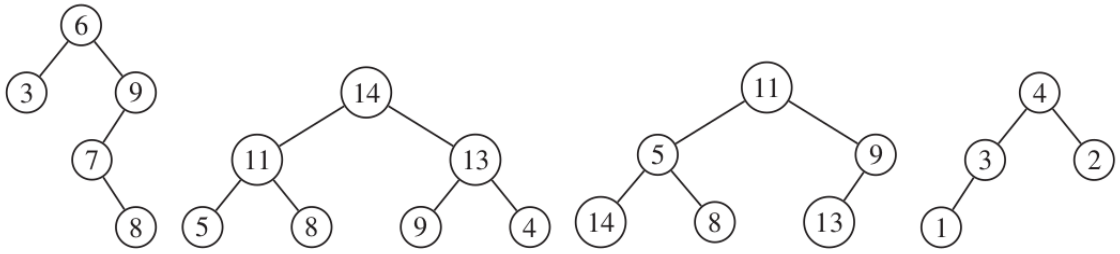
La grille peut être rendue avec la copie, et il est tout à fait envisageable (et recommandé) d'expliquer le raisonnement sur certains points. Inutile de dessiner les traverses, deux rails bien visibles (voire un seul tracé) suffisent. En outre, barrer les cases où il ne peut pas y avoir de rail est une bonne idée...

Partie III : Algorithmique et programmation

Question 6 : Écrire en Caml une fonction récursive de signature `lire_liste : int -> int list -> int` telle que `lire_liste i l` retourne le `i`-ième élément de `l`, pour `i` entre `1` et la taille de `l`.

On considère dans la suite des arbres binaires d'entiers, exprimés en Caml à l'aide du type construit `type arbre = Vide | Noeud of arbre * int * arbre;;`.

Voici quelques exemples de représentations graphiques d'arbres binaires d'entiers, en ne dessinant pas de `Vide`.



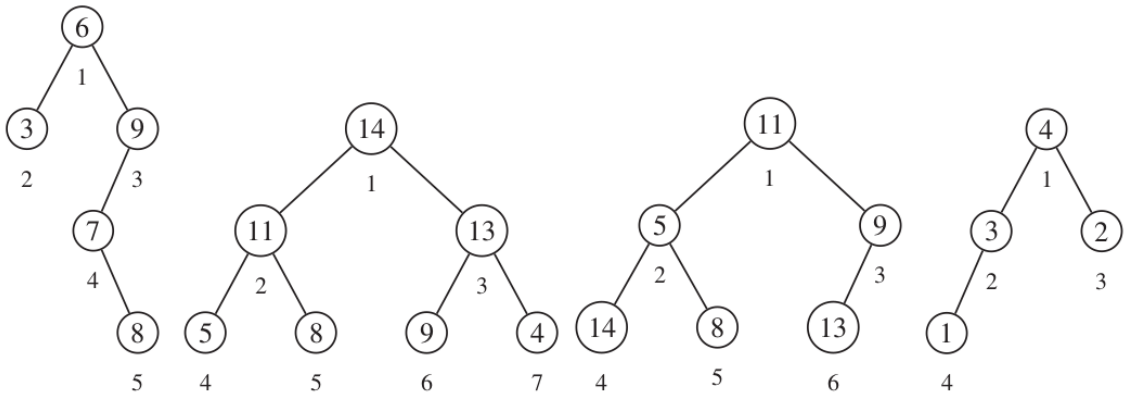
Question 7 : Donner l'expression en Caml des quatre arbres représentés.

Soient `a` un arbre binaire et `n` un nœud de `a`. La hauteur de `n` dans `a` est égale au nombre de nœuds du chemin sans cycle le plus long reliant `n` à un sous-arbre vide. Nous la noterons η(`n`). La hauteur d'un arbre est la hauteur de sa racine. Nous associerons la hauteur 0 à l'arbre binaire vide. La profondeur de `n` dans `a` est la différence entre la hauteur de `a` et celle de `n`. Un niveau dans un arbre binaire est une séquence de gauche à droite de tous les nœuds de même profondeur dans l'arbre.

Question 8 : Donner une définition mathématique de la hauteur d'un arbre binaire impliquant l'arbre vide, le fils gauche et le fils droit de l'arbre.

Question 9 : Soit un ensemble non vide d'étiquettes donné, quelle est la structure d'un arbre binaire contenant ces étiquettes et dont la hauteur est maximale ? Quelle est la structure d'un arbre binaire contenant ces étiquettes et dont la hauteur est minimale ? Justifier.

La numérotation hiérarchique des nœuds d'un arbre binaire `a` consiste à associer à chaque nœud un numéro entre `1` et |`a`| par un parcours en largeur partant de la racine (numéro `1`) et en parcourant chaque niveau de gauche à droite jusqu'au dernier nœud : le plus profond et le plus à droite (numéro |`a`|). Nous noterons N_i(`a`) le nœud de `a` de numéro `i`. Dans les exemples suivants, le numéro de chaque nœud sera noté en-dessous de son étiquette.



Question 10 : Écrire en Caml une fonction récursive (ou faisant appel à des sous-fonctions récursives) de signature `lire : int -> arbre -> int` telle que `lire i a` retourne N_i(`a`) si `i` est entre `1` et |`a`|. Cette fonction devra au plus parcourir une seule fois chaque élément de l'arbre `a`.

Question 11 : Écrire en Caml une fonction récursive (ou faisant appel à des sous-fonctions récursives) qui vérifie si un arbre binaire est un tas (on ne demande pas dans un premier temps de vérifier qu'il est presque complet). Cette fonction devra au plus parcourir une seule fois chaque élément de l'arbre `a`.

Un arbre binaire complet est un arbre binaire dont tous les niveaux sont complets, c'est-à-dire que tous les nœuds d'un même niveau ont deux fils non vides sauf les nœuds du niveau le plus profond qui n'ont aucun fils (c'est-à-dire deux fils vides).

Un arbre binaire presque complet est un arbre binaire dont tous les niveaux sont complets sauf éventuellement le dernier pour lequel les nœuds sont alignés à gauche. En particulier, un arbre binaire complet est ici considéré comme presque complet (en raison de l'adverbe « éventuellement »).

Question 12 : Montrer que, dans un arbre binaire complet non vide, le niveau de profondeur `p` contient 2^p nœuds. En déduire le nombre de nœuds d'un arbre binaire complet en fonction de sa hauteur.

Question 13 : Écrire en Caml une fonction récursive (ou faisant appel à des sous-fonctions récursives) qui détermine si un arbre binaire est complet ou s'il est presque complet sans être complet. On retournera donc par exemple une chaîne de caractères explicite. Cette fonction devra au plus parcourir une seule fois chaque élément de l'arbre `a`.

Question 14 : Soit un nœud de numéro `n` dans le niveau de profondeur `p` d'un arbre binaire presque complet, calculer le nombre de nœuds qui se trouvent à sa gauche dans le niveau de profondeur `p`.

Question 15 : Dans le niveau de profondeur `p + 1` d'un arbre binaire presque complet, quel est le nombre de nœuds qui se trouvent à la gauche des fils du nœud de numéro `n` (`n` faisant partie du niveau de profondeur `p`) ?

Question 16 : Soit un nœud de numéro `n` d'un arbre binaire presque complet, calculer les numéros de ses fils gauche et droit. En déduire le numéro du père du nœud de numéro `n` dans un tel arbre.

Question 17 : Écrire en Caml une fonction récursive (ou faisant appel à des sous-fonctions récursives) de signature `lirebis : int -> arbre -> int` telle que `lirebis i a` retourne N_i(`a`) si `i` est entre `1` et |`a`| en supposant que l'arbre binaire `a` soit presque complet. Cette fonction devra seulement parcourir (une seule fois) la branche de l'arbre `a` où se situe le nœud de numéro `i`.

Question 18 : Écrire finalement en Caml une fonction récursive (ou faisant appel à des sous-fonctions récursives) vérifiant si un arbre binaire est un tas binaire presque complet, le tout en un seul parcours plutôt que deux.

La grille de la partie II