Université Paris Diderot – Paris 7
L3 Informatique

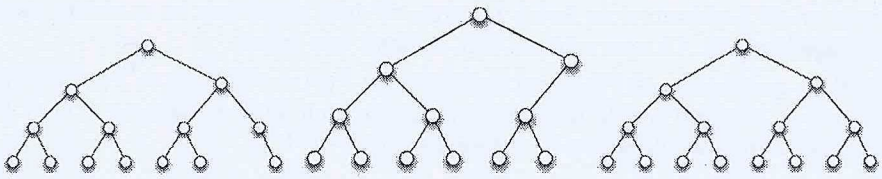
Algorithmique
Année 2014–2015, 1er semestre

AL5 Algorithmique
Examen de 2e session

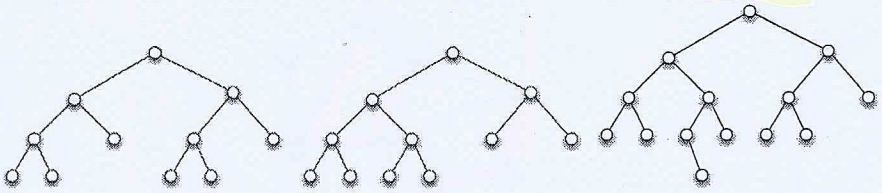
Mardi 30 juin 2015 8h30–11h30
Documentation autorisée : une feuille A4 recto verso manuscrite.
Toute autre documentation ou appareil sont interdits.

Indications : Le barème est donné à titre indicatif seulement. Justifiez vos réponses et expliquez vos algorithmes. Les algorithmes doivent être aussi efficaces que possible et présentés de façon lisible et claire. Une partie de la note portera sur la clarté de présentation de vos réponses (mais pas la longueur). Un bon choix de noms de variables et de noms d'algorithmes est essentiel. Ne pas négliger les explications courtes qui permettent de comprendre le fonctionnement de vos algorithmes.

Exercice 1 (6 points) Un arbre binaire est appelé parfait si toutes ses feuilles sont sur deux niveaux consécutifs et tous les niveaux excepté le dernier doivent être totalement remplis et si le dernier n'est pas totalement rempli alors il doit être rempli de gauche à droite. L'objectif de cet exercice est de concevoir un algorithme pour déterminer si un arbre est parfait, et de donner des éléments pour démontrer qu'il est correct. Reportez-vous aux exemples suivants pour vérifier vos réponses dans tous les cas de figure.



(a) non-parfait (b) non-parfait (c) parfait



(d) non-parfait (e) parfait (f) non-parfait

Notre point de départ pour concevoir l'algorithme est l'algorithme de parcours suivant.

```
def Parcours(root):
    Q = File((root,0)) - Crée une file contenant une paire
    while not Q.empty():
      (next,d) = Q.pop()
      print(next)
      if next.left :
         Q.push((next.left, d+1))
      if next.right :
         Q.push((next.right, d+1))
```

Pour vérifier qu'un arbre est parfait, il faut vérifier que tous les niveaux, sauf le dernier, sont complètement remplis, et que le dernier niveau est rempli de gauche à droite.

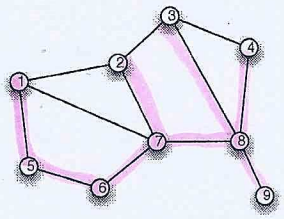
1. Quel type de parcours l'algorithme `Parcours` effectue-t-il ?
2. Quel est le contenu de la file lorsque l'algorithme affiche le sommet le plus à gauche d'un niveau ? Que pouvez-vous dire sur le nombre de sommets dans la file à ce moment-là ?
3. Comment pouvez-vous détecter que le sommet qui s'affiche est les plus à gauche d'un niveau ?
4. Quelle condition suffit-il de vérifier pour déterminer si un niveau est complètement rempli ? A quel moment du parcours pouvez-vous la vérifier ?
5. Quelle condition suffit-il de vérifier pour déterminer si un niveau est le dernier de l'arbre ? A quel moment du parcours pouvez-vous la vérifier ?
6. Quelle condition suffit-il de vérifier pour déterminer si un niveau est rempli de gauche à droite ? A quel moment du parcours pouvez-vous la vérifier ?
7. Modifier l'algorithme de parcours afin qu'il retourne un booléen VRAI si l'arbre est parfait, et FAUX sinon. Expliquer comment votre algorithme vérifie les conditions identifiées ci-dessus. Vous pouvez ajouter des variables pour maintenir les valeurs nécessaires pour vérifier les conditions ci-dessus mais vous devez maintenir l'ordre de parcours des éléments. Prenez soin d'écrire le code avec des noms de variables qui évoquent clairement chacune des conditions.
8. Donner la complexité de votre algorithme en fonction du nombre de noeuds dans l'arbre.
9. Donner l'exécution de votre algorithme sur les arbres (b) (d) et (f) ci-dessus en expliquant brièvement quelle condition n'est pas vérifiée et comment votre algorithme le détecte.

Exercice 2 (4 points)

Dans un AVL, le déséquilibre d'un sommet est donné par la hauteur du fils gauche moins la hauteur du fils droit.

1. Donner un AVL contenant au moins 12 sommets dans lequel tous les sommets (sauf les feuilles) ont un déséquilibre +1. Chaque sommet doit contenir une valeur.
2. Rappelons que la série de Fibonacci se définit par $F(0) = F(1) = 1$, et $F(n) = F(n-1) + F(n-2)$ pour $n \geq 2$. Montrer que le nombre de sommets d'un AVL de hauteur n dont tous les sommets internes ont un déséquilibre de +1 est égal à $F(n)$.

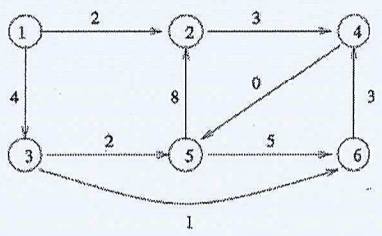
Exercice 3 (6 points)



1. Donner un arbre couvrant du graphe non-valué ci-dessus.
2. Supposons qu'on vous donne un algorithme `kruskal(G,P)` qui exécute l'algorithme de Kruskal sur un graphe valué `G` dont les poids sont donnés par `P`. L'algorithme prend en entrée un graphe dont les sommets sont numérotés de 0 à n-1. `G` est un tableau qui contient à l'indice u la liste des sommets adjacents à u et `P` est une matrice $n \times n$ qui à l'indice `[u,v]` contient le poids de l'arête (u, v) si l'arête est dans le graphe, et `None` sinon. (Vous pouvez supposer que pout tout entier x, la comparaison `x <= None` retourne toujours `Vrai`). L'algorithme retourne un liste d'arêtes. Montrer comment utiliser l'algorithme de Kruskal pour trouver un arbre couvrant d'un graphe non-valué. Donner la complexité de cet algorithme. (La complexité doit tenir compte du temps d'exécution de Kruskal.)

3. Donner un algorithme simple qui prend en entrée un graphe connexe non-valué et retourne un arbre couvrant du graphe. Le graphe est donné sous forme de liste d'adjacence et les sommets sont numérotés de 0 à n-1. L'algorithme doit retourner la liste des arêtes de l'arbre. Expliquer comment fonctionne l'algorithme. Donner la complexité de votre algorithme. *N.B. Pour avoir les points pour cette question, votre algorithme doit être plus simple et plus efficace que l'algorithme de Kruskal.*

Exercice 4 (4 points)



1. Donner l'exécution de l'algorithme de Dijkstra sur le graphe orienté ci-dessus. Le sommet de départ est 1. A chaque itération, donner l'ensemble des candidats considérés ainsi que leur priorité.
2. Quelle est la complexité de l'algorithme de Dijkstra si on maintient la file de priorité dans un tas ? Quelle est la complexité si on maintient la file de priorité dans une liste triée ? Vous devez tenir compte dans les deux cas du temps pour mettre à jour les priorités et pour réordonner la file de priorité, ainsi que pour retirer l'élément de priorité minimum ou maximum.