AL5 – Algorithmique
Examen de 2e session

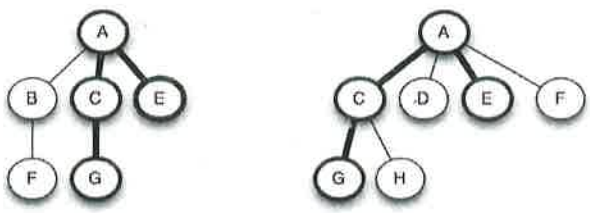
Mard 21 juin 2016 8h30–11h30

Documentation autorisée : deux feuilles A4 recto verso manuscrites. Toute autre documentation et appareil sont interdits. **Les téléphones portables doivent être éteints et déposés dans votre sac et vous devez poser vos sacs et vos vestes à l'avant de la salle avant de commencer.**

> **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. Cela veut dire que les noms de variables doivent être bien choisis pour représenter ce qu'elles sont censées contenir et que les principales étapes doivent être accompagnées d'explications. Une partie de la note portera sur la clarté de présentation de vos réponses (mais pas leur longueur).

Exercice 1 : Parcours d'arbres (6 points)

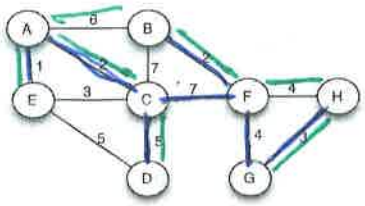
L'objectif de cette question est de donner un algorithme qui prend en entrée deux arbres (d'arité variable) et retourne deux listes contenant les différences entre les deux arbres.
Chaque sommet `s` est un objet avec les attributs `s.valeur` pour sa valeur et `s.fils` pour la liste des fils de `s`. Si `s.fils` est vide alors le sommet est une feuille. Toutes les valeurs sont distinctes à l'intérieur d'un arbre. Vous pouvez supposer que les fils sont triés par ordre de valeur.
L'algorithme prend en entrée deux sommets `A1, A2` (qui sont la racine de leurs arbres respectifs) et retourne une paire d'ensembles de sommets `(AbsA1, AbsA2)`. Un sommet est commun aux deux arbres s'il a la même valeur et se trouve au même endroit dans le sous-arbre commun. `AbsA1` contient l'ensemble des sommets de `A2` absents de `A1` (et vice versa pour `AbsA2`). Si un sous-arbre est absent d'un des deux arbres, il suffit de mettre sa racine dans l'ensemble correspondant. Dans l'exemple ci-dessous, l'algorithme doit retourner les ensemble `AbsA1={D,H,F}` et `AbsA2={B}`. (L'ordre n'est pas important.)



1. Donner un algorithme qui trouve les différences entre deux arbres selon les spécifications ci-dessus.
2. Analyser la complexité de votre algorithme si `A1` a n_1 sommets, `A2` a n_2 sommets, et l'arbre commun a n sommets.
3. Donner l'ordre de parcours des sommets dans les deux arbres et le résultat de l'algorithme sur les arbres donnés ci-dessus.

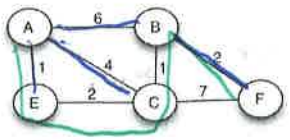
Exercice 2 : Kruskal et Union Find (4 points)

Appliquer l'algorithme de Kruskal pour trouver un arbre couvrant minimal du graphe valué ci-dessous. Utiliser un algorithme basé sur Union-Find pour détecter les cycles. (Utiliser la variante de Union-Find avec compression de chemin). Donner, pour chaque étape de l'algorithme de Kruskal, la structure Union-Find obtenue.



Exercice 3 : Dijkstra (4 points)

Donner l'exécution de l'algorithme de Dijkstra sur le graphe ci-dessous, avec A comme sommet de départ. Donner pour chaque étape le sommet sélectionné et le tas-min (sous forme d'arbre) qui contient les priorités des sommets candidats à cette étape.

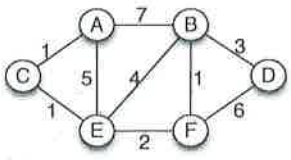


Exercice 4 : Plus courts chemins (6 points)

On souhaite concevoir un algorithme qui prend en entrée un graphe non-orienté valué `(G, w)` et calcule les plus courts chemins à partir d'un sommet `s` vers tous les sommets du graphe, chaque chemin passant par au plus `k` sommets intermédiaires.

Le graphe `G` est constitué de sommets numérotés de 0 à $n-1$. Pour chaque sommet `i`, `G[i]` est une liste d'entiers (la liste d'adjacence). Les poids des arêtes sont donnés par une matrice `w`, et le poids de l'arête `(i,j)` est `w[i][j]`.

L'algorithme retourne une liste `L` de `n` chemins, un pour chaque sommet `t` du graphe. `L[t]` contient le plus court chemin de `s` à `t` passant par au plus `k` sommets intermédiaires. Un chemin de `s` à `t` est une liste de sommets du chemin dont le premier élément est `s` et le dernier élément est `t`. S'il n'existe pas de chemin de `s` à `t` passant par au plus `k` sommets intermédiaires, cette liste doit être vide.



1. Donner le résultat attendu sur le graphe ci-dessus avec `k = 2` à partir du sommet `A`.
2. Sur quel algorithme vu en cours votre algorithme est-il basé ? Expliquer brièvement comment le modifier. En citant les propriétés vues en cours, justifier pourquoi cette approche donne un résultat correct.
3. Donner l'algorithme.
4. Analyser sa complexité en fonction du nombre de sommets n, du nombre d'arêtes m, et de la longueur des chemins k.