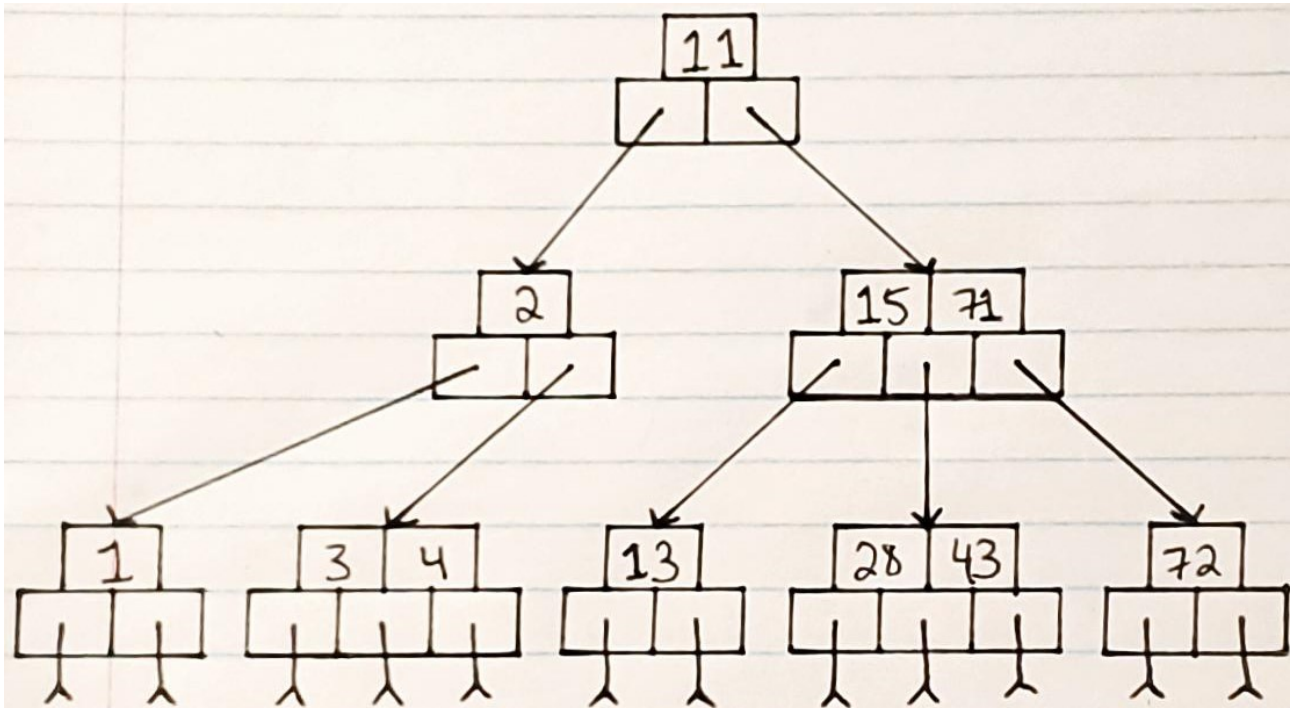
VIGEANT, Dominique. Matricule 20129080.

Question 1. (10 points)

(a)

1. La hauteur du noeud 11 aurait dû être augmentée au niveau 3 afin de laisser un gap de 1 à gauche et un gap 1 à droite.
2. La hauteur du noeud 15 aurait dû être augmentée au niveau 2. Il n'aurait pas été possible d'augmenter le noeud 13 ou le noeud 43 puisqu'il n'y aurait pas eu de gap après le noeud 11 ou avant le noeud 71 respectivement.

(b)



(c)

Hauteur 3 :

Règle (a) : Si le chemin vers lequel on doit descendre est un bloc de 2 enfants ou plus, on ne fait rien.

Hauteur 2 :

Règle (b-2) D-à-G : Si le chemin vers lequel on doit descendre est un bloc de 1 enfant et que son sibling gauche est un bloc de ≥ 2 enfants, on remonte le noeud le plus grand du sibling gauche. Après, on descend le noeud le plus grand vers le sibling droit.

Hauteur 1 : Supprimer le noeud désiré.

Question 2. (10 points)

(a)

- Insertion :
  - Liste non triée : On peut insérer la clé en première place, donc $coutMoyen = 0\lambda + 1 = 1$.
  - Liste triée : En assumant que chaque position est équiprobable de recevoir la clé, $coutMoyen = \lambda$.
- Recherche :
  - Liste non triée : Pour une recherche non réussie, il faut traverser la liste au complet pour se rendre compte de l'échec. Ainsi $coutMoyenEchec = \lambda$. Pour une recherche réussie, il faut en moyenne examiner la moitié des λ noeuds de la liste avant de trouver le noeud voulu ; $coutMoyenSucces = \frac{\lambda}{2}$.
  - Liste triée : Que la recherche soit réussie ou non, il faut en moyenne examiner la moitié des noeuds avant d'arrêter de comparer ; $coutMoyen = \frac{\lambda}{2}$.

(b) Soit un arbre cousu dont tous les noeuds ont entre autres les trois attributs suivant : pointeurGauche, pointeurDroit et visité. pointeurGauche et pointeurDroit peuvent prendre 0 ou 1 comme valeur. 0 signifie que c'est un pointeur normal et 1 signifie que c'est une ficelle (qui peut être null). visité est un booléen qui est initialisé à faux pour chaque noeud et qui devient vrai si on a accédé de façon in-order au noeud. Aussi, l'arbre possède l'état général suivant : mpf est un booléen qui vaut vrai si on vient de monter dans un noeud par une ficelle et faux sinon.

Chaque noeud peut donc faire partie d'un des quatre cas suivants (le nombre de gauche représente pointeurGauche et le nombre de droite représente pointeurDroit) :

00 : Si mpf == faux, on descend à gauche et on garde visité = faux et mpf = faux. Si mpf == vrai, alors on pose visité = vrai et mpf = faux, puis on descend à droite.

11 : Un noeud avec deux ficelles comme pointeurs est nécessairement une feuille. Donc peu importe la valeur de mpf (qui devrait logiquement être faux), on pose visité = vrai et mpf = vrai puis on remonte à droite si la ficelle n'est pas null. Si la ficelle est null, le parcours est terminé.

10 : Si mpf == faux, on pose visité = vrai, on garde mpf = faux puis on descend à droite. mpf == vrai ne devrait pas arriver puisqu'avoir une ficelle à gauche et un pointeur normal à droite signifie que le noeud n'a pas d'enfant à gauche et un enfant à droite. Dans un parcours in-order, il est donc impossible de remonter par une ficelle vers ce noeud.

01 : Si mpf == faux, on garde visité = faux et mpf = faux puis on descend à gauche. Si mpf == vrai, on garde mpf = vrai et on pose visité = vrai, puis on remonte à droite.

Remarquons que dans un parcours in-order, les feuilles sont visitées au maximum une fois et les noeuds internes sont visités au maximum deux fois, c'est-à-dire :

$$\text{nombreDeFeuilles} = \text{nombreDeNoeuds11}$$

$$\begin{aligned}\text{nombreDeNoeudsInternes} &= \text{nombreDeNoeuds01} + \text{nombreDeNoeuds10} + \text{nombreDeNoeuds00} \\ &= N - \text{nombreDeFeuilles}\end{aligned}$$

De plus, le nombre d'opérations pour parcourir un arbre cousu in-order correspond à la somme des visites de chaque noeud. Dans le pire cas, on a donc

$$\begin{aligned}\text{nombreDeVisites} &= 2\text{nombreDeNoeuds01} + 2\text{nombreDeNoeuds10} + 2\text{nombreDeNoeuds00} + \text{nombreDeNoeuds11} \\ &= 2\text{nombreDeNoeudsInternes} + \text{nombreDeFeuilles} \\ &= 2(N - \text{nombreDeFeuilles}) + \text{nombreDeFeuilles} \\ &= 2N - \text{nombreDeFeuilles}\end{aligned}$$

Et ainsi la complexité d'un parcours in-order (GRD) d'un arbre avec ficelles est

$$\begin{aligned}O(\text{nombreDeVisites}) &= O(2N - \text{nombreDeFeuilles}) \\ &= O(N)\end{aligned}$$

Fin de la preuve.

Question 3. (10 points)

(a) Comme indiqué dans le chat 1 de la séance 13 :

```
... faire {
    si (T[v].d + c_vw < T[w].d) alors {
        T[w].d ← T[v].d + c_vw;
        T[w].p ← v;
    }
}
```

(b) Dans le cas de Dijkstra, le champ p peut être interprété comme étant le noeud connu w par lequel arriver à v si on veut passer par le chemin le plus court entre le noeud de départ et v.

(c)

```
Imprimer(Sommet v, Table T) {
    si (T[v].p != "-") alors {
        Imprimer(T[v].p, T);
        Écrire(" à ");
    }
    Écrire(v);
}
```

Algorithme fortement inspiré de Weiss p.378 Figure 9.30.

Question 4. (10 points)

(a)

1. Chaque noeud est coloré rouge ou noir
   Pas de problème ici.
2. La racine est noire
   Mettons la racine noire alors (#1 est toujours correct).
3. Si un noeud est rouge, alors ses enfants sont noirs
   Le noeud est noir.
4. Chaque chemin entre un noeud (n'importe quel) et un autre doit avoir le même nombre de noeuds noirs
   Puisque c'est la racine qui est changée en noir, tous les chemins commençant par la racine ont 1 noeud noir de plus, donc ils ont encore tous le même nombre de noeuds noirs. Pour les chemins qui ne commencent pas par la racine, rien n'est changé donc la condition est encore respectée.

On voit que changer la racine de noir à rouge donne un arbre qui respecte toutes les conditions.

(b) Puisque l'insertion se fait en ordre croissant, la première insertion (1) nécessite 1 opération. Les insertions suivantes nécessitent chacunes 3 opérations :

1. Trouver l'endroit où le noeud doit être inséré, c'est-à-dire comme enfant droit de la racine, et l'y insérer. (2 opérations)
2. Ramener le noeud à la racine, c'est-à-dire effectuer un zig gauche. (1 opération)

La complexité est donc

$$O(3(N-1)+1) = O(N)$$

(c)

L'ami a effectivement raison. Cependant,