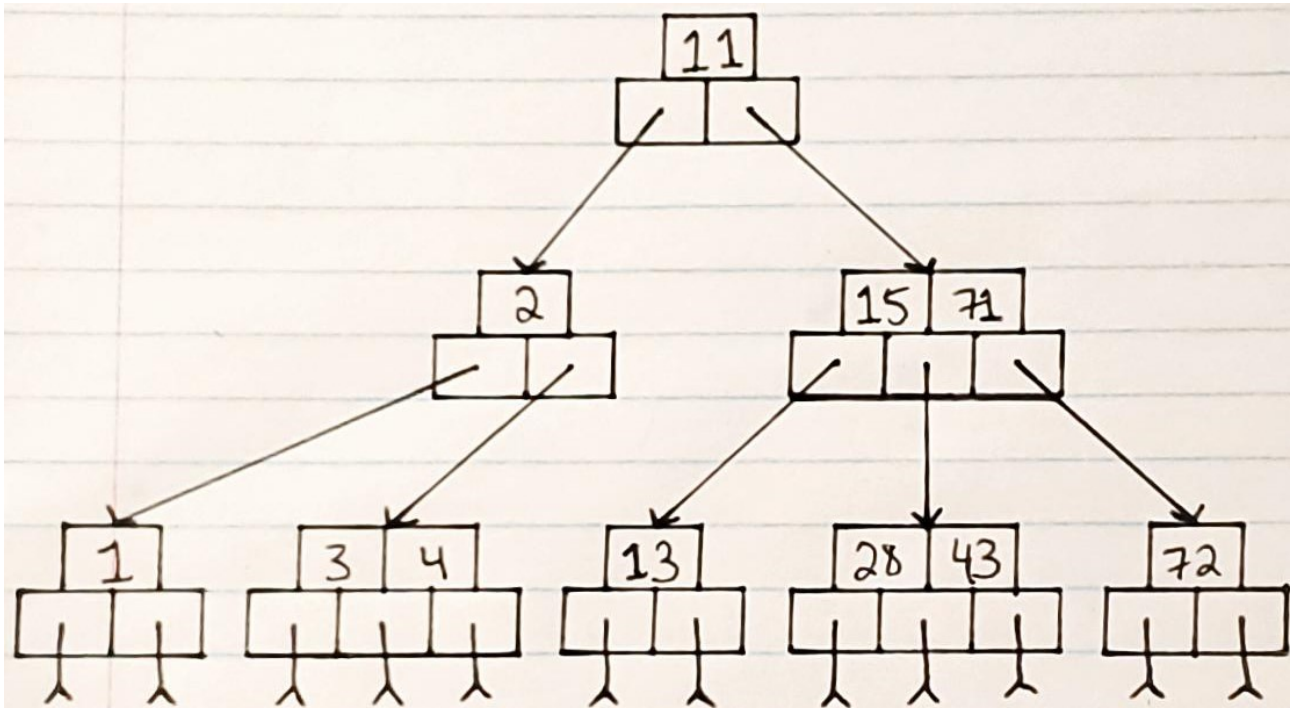
VIGEANT, Dominique. Matricule 20129080.

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 de 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 ; $coutMoyenEchec = \lambda$. 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 λ noeuds avant d'arrêter de comparer ; $coutMoyen = \frac{\lambda}{2}$.

(b)

Chaque noeud d'un arbre cousu a exactement 2 pointeurs (ficelle ou normal, les ficelles des deux extrémités étant null). Les abbréviations suivantes seront utilisées pour la preuve : F = Ficelle (null ou non), P = Pointeur normal, G = Gauche, D = Droite, MPF = Monté par une ficelle, V = visité.

Considérons toutes les combinaisons des 3 états suivants : Type de pointeur (F ou P), Direction du pointeur (G ou D) et Provenance (MPF ou non). N'oublions pas que le parcours qui nous intéresse ici est in-order (GRD). Faisons le parcours en partant de la racine. Lors de son accès, un noeud est considéré comme V seulement s'il n'a pas pas d'enfant (FG, FD), s'il a été accédé par une ficelle (MPF) ou s'il n'a pas d'enfant gauche.

1. PG, PD — V = non, Direction = G.
2. PG, PD, MPF — V = oui, Direction = D.
3. FG, FD — V = oui, Direction = D (successeur in-order).
4. FG, FD, MPF — Impossible puisque le noeud n'a aucun enfant.
5. PG, FD — V = non, Direction = G.
6. PG, FD, MPF — V = oui, Direction = D (succésseur in-order).
7. FG, PD — V = oui, Direction = D.
8. FG, PD, MPF — Impossible puisque le noeud n'a pas d'enfant G.

Remarquons les caractéristiques suivantes du parcours in-order d'un arbre cousu :

- Le parcours se termine lorsque le noeud ayant une FD null est atteint.
- Les noeuds (FG, FD) sont des feuilles et elles ne sont accédées qu'une seule fois chacune.
- Les noeuds (PG, PD) et (PG, FD) sont des noeuds internes et sont accédés deux fois chacun. Ils sont considérés comme visités lors de leur deuxième accès.
- Les noeuds (FG, PD) sont des noeuds internes et sont accédés deux fois chacun. Ils sont considérés comme visités lors de leur premier accès.

Posons

$$L = \text{nombre de feuilles} = \text{nombre de noeuds (FG, FD)}$$

$$I = \text{nombre de noeuds internes} = \text{nombre de noeuds (PG, PD) + (FG, PD) + (PG, FD)}$$

$$N = \text{nombre de noeuds total} = L + I$$

$$\begin{aligned} A = \text{nombre d'accès total} &= L + 2(PG, PD) + 2(FG, PD) + 2(PG, FD) \\ &= L + 2I \\ &= L + I + I \\ &= N + I \end{aligned}$$

La complexité du parcours in-order d'un arbre cousu est égal à la somme du nombre d'accès de chaque noeuds.

$$O(A) = O(N + I) = O(N)$$

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 (la racine) 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 rouge à noir 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 pour l'accès à la clé 1. Cependant, comme Weiss dit au Chapitre 4.5 p.137, les splay trees garantissent qu'une séquence de M opérations sur un ensemble de N noeuds à partir d'un arbre vide coûte au plus $O(M\log N)$.

Toutefois cela n'assure pas qu'une opération distincte ne coûtera pas $O(N)$, mais ce qui est intéressant (et que Weiss explique à la même page) est que si une séquence de M opérations a un coût total de $O(Mf(N))$ dans le pire des cas, généralement on compte le coût amorti comme étant $O(f(N))$. Dans le contexte d'une analyse amortie, les splay trees ont donc un coût de $O(\log N)$ par opération.

Toujours dans Weiss, au Chapitre 11 p.513, il est spécifié que les limites établies par une analyse amortie sont moins fortes que les limites dans le pire cas analogue. La raison est qu'il n'y a pas de garantie pour chaque opération distincte. Le résultat d'un splay tree est donc moins fort que le coût $O(\log N)$ dans le pire cas par opération, ce qui est le résultat pour les arbres rouge-noir selon Weiss au Chapitre 12.2 p.549, mais à long terme le résultat est très similaire.

Pour les arbres rouge-noir, le simple fait d'accéder à un noeud n'implique aucune opération subséquente. Une analyse amortie n'est donc pas nécessaire et on se fie au coût dans le pire cas, qui est $O(\log N)$ puisque la hauteur maximale d'un arbre rouge-noir est $2\log(N+1)$ (Weiss p.549).

Il est possible de donner le coût dans le pire cas de chaque opération pour un splay tree. Celui-ci serait de $O(N)$, le cas mentionné en ii étant un bon exemple. Le problème avec cela est qu'il ne reflète pas la réalité. À chaque "mauvais" accès dans l'arbre, sa hauteur a le potentiel de réduire de manière non négligeable, ce qui généralement réduit le coût des accès suivants.