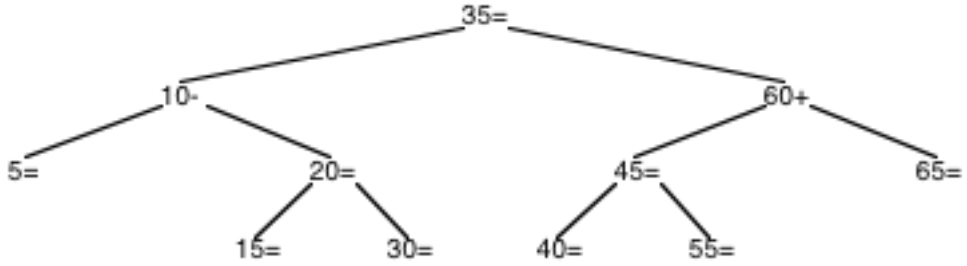
Corrigé de l'examen de Structures de données du 8 février 2003

Exercice 1

Question A

Un arbre AVL est un arbre binaire de recherche qui est H-équilibré. Un arbre binaire de recherche est tel que la clé de tout nœud est plus grande que toutes les clés des nœuds de son sous arbre gauche et plus petite que toutes les clés des nœuds de son sous arbre droit. Un arbre binaire est H-équilibré si en tout nœud, la différence de hauteur entre les sous arbres gauche et droit est au plus de 1. On peut constater sur l'arbre donné qu'il s'agit bien d'un arbre binaire de recherche, et que de plus il est H-équilibré. On peut le décorer avec les déséquilibres (-, =, +).



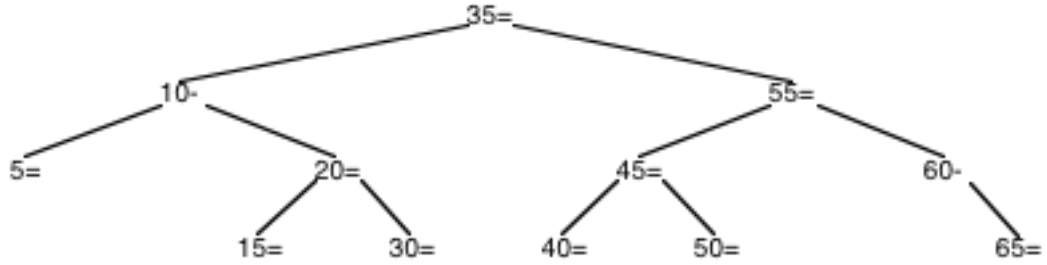
Question B

L'adjonction dans un arbre AVL consiste d'abord à faire une adjonction d'une feuille dans un arbre binaire de recherche, en recherchant l'endroit où doit être le nœud en tant que feuille. Pour cela, partant de la racine, on descend dans l'arbre, à gauche ou à droite, selon que la clé de l'élément ajouté est inférieure ou supérieure à la clé du nœud, jusqu'à rencontrer un arbre vide qui est remplacé par un nouveau nœud contenant l'élément ajouté. Ensuite, on remonte le chemin vers la racine, en corrigeant les déséquilibres (+1 si on remonte depuis la gauche et –1 si on remonte depuis la droite), en vérifiant que le résultat reste dans les limites (-1, 0 ou +1) et en pratiquant la rotation adaptée si ce n'est pas le cas. Dans tous les cas, on arrête la remontée et les corrections lorsque le déséquilibre résultant est nul.

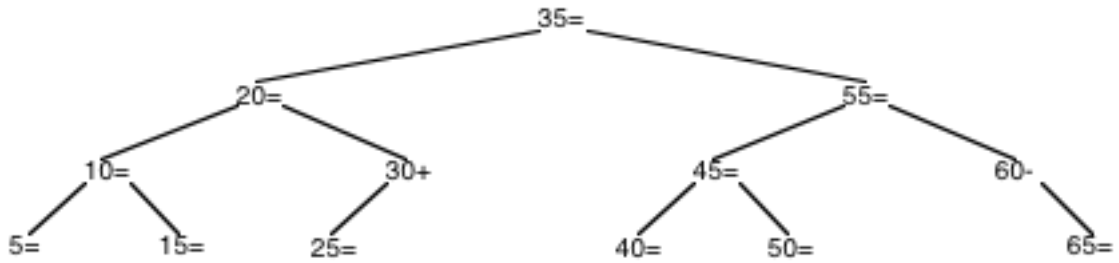
Question C

On part de l'arbre ci-dessus, pour ajouter d'abord 50. 50 étant plus grand que 35, on descend à droite. 50 étant plus petit que 60 on descend à gauche. 50 est plus grand que 45 on descend à droite. 50 est plus petit que 55 on descend à gauche. On est sur un arbre vide qui est donc remplacé par le nœud 50, fils gauche de 55. Ceci porte le

déséquilibre de 55 à +1, celui de 45 à –1, et celui de 60 à +2. Il faut alors pratiquer une rotation gauche-droite puisque le déséquilibre de 45 est –1. Ceci met 55 à la place de 60 avec un déséquilibre nul□ la remontée est donc terminée, et on obtient l'arbre ci-dessous.



Pour ajouter 25 dans ce nouvel arbre, 25 étant plus petit que 35, on descend à gauche. 25 étant plus grand que 10 on descend à droite. 25 étant plus grand que 20 on descend à droite. 25 étant plus petit que 30 on descend à gauche. On est sur un arbre vide qui est donc remplacé par le nœud 25, fils gauche de 30. Ceci porte le déséquilibre de 30 à +1, celui de 20 à –1 et celui de 10 à –2. Il faut alors pratiquer une rotation à gauche puisque le déséquilibre de 30 est –1. Ceci met 20 à la place de 10 avec un déséquilibre nul□ la remontée est terminée et on obtient l'arbre ci-dessous.



Question D

La liste infixée de l'arbre initial est obtenu en mettant chaque nœud après la liste infixée de son sous arbre gauche et avant la liste infixée de son sous arbre droit. On obtient la liste ci-dessous, qui est ordonnée pour un arbre binaire de recherche.

5, 10, 15, 20, 30, 35, 40, 45, 55, 60, 65

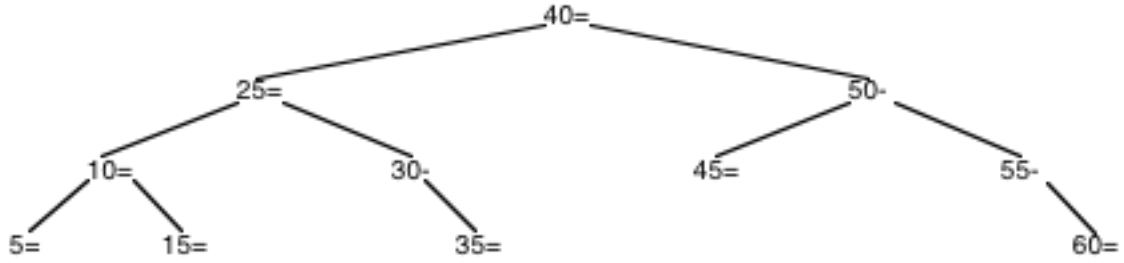
Question E

L'opération de suppression consiste d'abord à rechercher le nœud à supprimer. Pour cela, partant de la racine, on descend dans l'arbre, à gauche ou à droite, selon que la clé de l'élément recherché est inférieure ou supérieure à la clé du nœud, jusqu'à trouver le nœud. Si ce nœud a deux fils, on remplace la valeur par celle qui est à l'extrémité du bord gauche du sous arbre droit, et c'est ce dernier nœud qui est supprimé. La suppression du nœud consiste ensuite à raccrocher le sous arbre éventuellement existant (gauche ou droit) à sa place. Ensuite, on remonte le chemin vers la racine, en corrigeant les déséquilibres (-1 si on remonte depuis la gauche et +1 si on remonte depuis la droite), en vérifiant que le résultat reste dans les limites (-1, 0

ou +1) et en pratiquant la rotation adaptée si ce n'est pas le cas. Dans tous les cas, on arrête la remontée et les corrections lorsque le déséquilibre résultant est non nul.

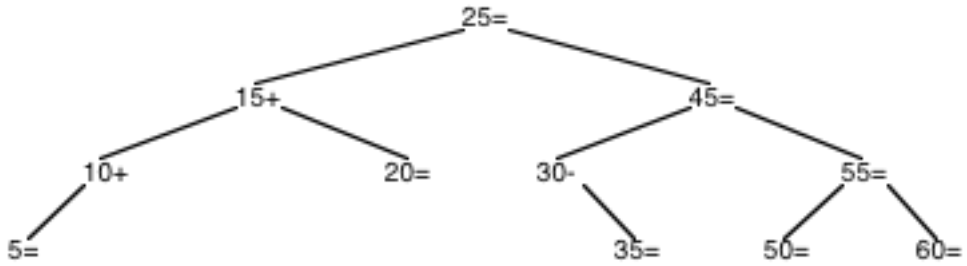
Question F

La suppression de 20 dans l'arbre donné, consiste d'abord à le rechercher. 20 étant plus petit que 25, on descend à gauche. 20 étant plus grand que 15 on descend à droite. On est alors sur le nœud, qui est simplement supprimé, puisqu'il n'a pas de fils. Le déséquilibre de 15 devient +2, ce qui nécessite une rotation à droite puisque le déséquilibre de 10 est +1. Le déséquilibre résultant étant nul, il faut poursuivre la remontée. Le déséquilibre de 25 devient –2, ce qui demande une rotation à gauche puisque le déséquilibre de 40 est –1. On obtient ainsi l'arbre suivant.



Question G

La suppression de 40 dans l'arbre donné, consiste d'abord à le rechercher. 40 étant plus grand que 25, on descend à droite. On est sur le nœud qui a deux fils. On remplace la valeur par celle située à l'extrémité du bord gauche du sous arbre droit, c'est-à-dire, 45, et on supprime ce nœud. Le déséquilibre de 50 devient –2, ce qui nécessite une rotation à gauche, puisque le déséquilibre de 55 est –1. Le déséquilibre résultant étant nul, il faut poursuivre la remontée. Le déséquilibre de 45 devient nul, et celui de 25 également. Comme on est à la racine, on s'arrête.



Exercice 2

Question A

En fait l'opération de recherche proposée ici est analogue à la recherche si une liste est une sous liste d'une autre avec en plus l'information sur la position où se trouve

cette sous liste. On peut donc s'inspirer de l'exercice correspondant (cf. 15.7 du livre).

Question A.1

Lors de l'appel de `Recherche (De => L2, Dans => L1)`, on va donc comparer les 3 éléments de la liste L2 avec les portions de la liste L1 de même longueur, et commencant en 1, puis 2, puis 3, etc. jusque 8, où l'on constatera l'égalité. La valeur retournée sera donc 8.

Lors de l'appel de `Recherche (De => L3, Dans => L1)`, on va donc comparer les 4 éléments de la liste L3 avec les portions de la liste L1 de même longueur, et commencant en 1, puis 2, puis 3, etc. jusque 10, (ou 13 éventuellement). A chaque fois, on constatera l'inégalité, et donc à la fin l'exception `Erreur_Specification` sera levée. Il est certain que la chaîne «□SFIO□ est bien dans le tableau de la liste L1, mais elle est au delà de la fin de la liste, donc elle ne fait pas partie de la liste.

Question A.2

On va faire une fonction interne qui contrôle l'égalité de la liste `De` avec la portion de la sous liste `Dans` commencant en position `En`.

```
function Recherche (De, Dans: Texte) return Positive is
    function Egalite (Decalage: Natural) return Boolean is
    begin
        for I in 1..Longueur (De) loop
            if Ieme (De, I) /= Ieme (Dans, I + Decalage) then
                return False□;
            end if□;
        end loop□;
        return True□;
    end Egalite□;
    for J in 0..Longueur (Dans) – Longueur (De) loop
        if Egalite (J) then return 1 + J; end if□;
    end loop□;
    raise Erreur_Specification□;
end Recherche□;
```

Soient n la longueur de la liste `De` et m la longueur de la liste `Dans`.

1. Si `De` est sous liste de `Dans`, au mieux on le constatera au premier appel de `Egalite`. Il y aura alors n comparaisons. Au pire, on le constatera à la fin, c'est-à-dire lors du (m-n+1)ème appel de `Egalite`, les précédents ayant échoués. La complexité au pire est donc en □(n*(m-n)). En moyenne, on peut dire qu'il y a (m-n)/2 appels de `Egalite`, chacun d'eux demandant n/2 comparaisons, donnant une complexité en moyenne de □(n*(m-n)).
2. Si `De` n'est pas dans `Dans`, il faudra appliquer `Egalite` (m-n) fois. Chacun de ces appels demande au mieux 1 comparaison, au pire n, et en moyenne n/2.

On constate donc que cet algorithme a une complexité en moyenne et au pire en Θ(n*(m-n)).

Question B

La suppression des espaces qui se suivent est analogue à la procédure Unique sur une liste ordonnée, lorsqu'on supprime un élément de la liste lorsqu'il est égal à celui qui le précède (cf. exercice 16.3 question A du livre).

Question B.1

Le résultat de `Supprimer_Les_Espaces_Inutiles (L1)` est sans effet sur la liste `L1`. En effet, dans la portion du tableau qui est la représentation de la liste, il n'y a pas deux espaces qui se suivent. Le seul endroit du tableau où ceci se présente est en dehors de la liste.

Question B.2

Deux méthodes peuvent être envisagées. La première, simple, utilise l'opération `Supprimer` des listes. Elle a l'inconvénient d'être en $\Theta(n^2)$, en particulier si le texte ne contient que des espaces. La seconde est un peu plus compliquée. Il s'agit de déplacer les éléments de la liste lorsque l'on connaît leur place définitive. Sa complexité est alors en Θ(n). Nous ne donnons que cette version dans ce corrigé.

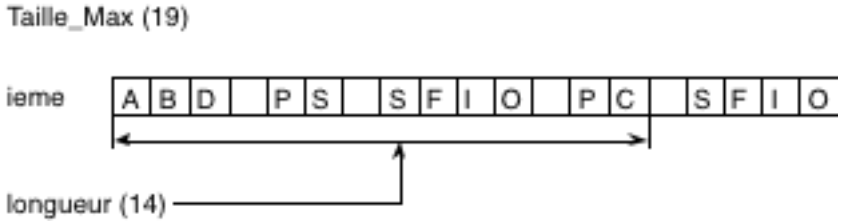
```
procedure Supprimer_Les_Espaces_Inutiles
                              (Dans: in out Texte) is
    I: Positive := 1;
begin
    if Longueur (Dans) = 0 then return; end if;
    for J in 2..Longueur (Dans) loop
        if Ieme (Dans, J) /= ' '
        or else Ieme (Dans, I) /= ' ' then
            I := I + 1;
            if I /= J then
                Changer_Ieme (Dans => Dans, En => I,
                              Val => Ieme (Dans, J));
            end if;
        end if;
    end loop;
    Tronquer (Dans, I);
end Supprimer_Les_Espaces_Inutiles;
```

Question C

Pour la substitution, il faut d'abord localiser `La_chaine` dans le texte. Pour cela, on peut utiliser la fonction étudiée en A. Ensuite, il faut remplacer la portion de texte par la nouvelle. Celle-ci peut être plus courte ou plus longue que l'ancienne, et pour cela il faut déplacer les éléments qui sont derrière.

Question C.1

Nous avons vu que le texte L2 est en 8 dans L1. Il faut le remplacer par un texte plus long (4 au lieu de 3). Il faut donc déplacer la portion derrière lui et on obtient finalement la liste L1 suivante.



Question C.2

Nous prendrons ici une méthode simple. Si le texte de remplacement est plus court que le texte initial, on remplace l'ancien par le nouveau et on supprime les éléments de l'ancien qui sont en trop. Si le texte de remplacement est plus long que le texte initial, on remplace la portion de même longueur et on insère les éléments restant. Evidemment dans ce contexte, les suppressions un par un ou les insertions un par un sont en p*m où p est le nombre de caractères au delà de la portion remplacée et m la différence de longueur entre les deux textes substitués.

```
procedure Substituer (La_Chaine□: in Texte□;
                      Par□: in Texte□;
                      Dans□: in out Texte) is
    Position□: Positive□:= Recherche (De => La_Chaine,
                                      Dans => Dans)□;
    long: Natural□:= Longueur(La_Chaine);
begin
    if Longueur(Dans) – long + Longueur(Par)
                                  > Dans.Taille_Max then
        raise Limite_Implantation; -- trop longue
    end if□;
    if long > Longueur (Par) then
        long□:= Longueur(Par);
        --la plus courte des deux longueurs
    end if;
    -- on repopie d'abord la plus courte des deux longueurs
    for I in 1..long loop
        Changer_Ieme(Dans => Dans, En => Position + I – 1,
                     Val => Ieme (Par, I));
    end loop;
    -- on supprime ceux qui sont en trop, s'il y en a
    for I in long + 1..Longueur (La_Chaine) loop
        Supprimer (Dans, Position + Longueur(Par));
    end loop;
    -- on insère ceux qui restent, s'il y en a
    for I in long + 1..Longueur(Par) loop
        Inserer (Dans => Dans, En => Position + I - 1,
                 Val => Ieme(Par, I));
    end loop;
end Substituer;
```

Exercice 0 (QCM)

2 Dans un B-arbre d'ordre 10, qui contient 100000 valeurs, à combien estimez-vous le nombre d'accès disque pour rechercher un élément?

☐ 10000 ☐ 50 ☐ 17 ☐ 14 ☒ 5

Note: la hauteur du B-arbre est comprise entre $\log_{21}100000$ et $\log_{11}50000$. Ce ne peut être que 5.

3 Dans un B-arbre d'ordre m, qui contient N valeurs, si on applique une recherche dichotomique sur les nœuds, combien y aura-t-il de comparaisons?

☐ N/m ☐ $m \log_m N$ ☒ $\log_2 N$ ☐ $\log_2 N/m$ ☐ $\log_m N$

Note: voir 12.7 du livre.

4 Si on trie une liste contiguë, qui contient 100000 valeurs, par un tri par tas, sur une machine dont le temps de base d'une comparaison ou d'un transfert est de l'ordre de la dizaine de microsecondes, à combien estimez-vous le temps de tri?

☐ 10 jours ☐ 1 jour ☐ 1heure ☒ 1minute ☐ 1 seconde

Note: la complexité est en $n \log_2 n (2Tc+Tt)$.

5 Si on trie une liste contiguë, qui contient 100000 valeurs, par un tri par la méthode de la bulle, sur une machine dont le temps de base d'une comparaison ou d'un transfert est de l'ordre de la dizaine de microsecondes, à combien estimez-vous le temps de tri?

☐ 10 jours ☒ 1 jour ☐ 1heure ☐ 1minute ☐ 1 seconde

Note: la complexité est en $n^2/4(Tc+3Tt)$.

6 Quel est l'ordre de grandeur de la complexité au pire de l'opération qui teste si deux arbres binaires de recherche contiennent les mêmes éléments, en supposant qu'ils en contiennent au plus n?

☐ $\log_2 n$ ☐ $n*(n-1)/2$ ☐ $n* \log_2 n$ ☒ n ☐ $n!$

Note: Comme il s'agit d'arbres binaires de recherche, ils contiennent les mêmes éléments s'ils ont même liste infixe. Donc il faut parcourir les deux arbres selon leur liste infixe et vérifier l'égalité des contenus nœud par nœud.

7 Si on définit `début(x)=cons(premier(x),listevide)`, parmi les cinq égalités suivantes, quelle est celle qui est obligatoirement vraie?

☐ `concaténer(début(x),x)=concaténer(x,fin(x))`

☐ `début(fin(x))=x`

☒ `concaténer(début(x),fin(x))=x`

☐ `début(fin(x))=fin(début(x))`

☐ `fin(début(x))=x`

Note : un raisonnement simple sur les longueurs des listes conduit à éliminer les autres possibilités.

8 Supposons une liste construite à l'aide des types suivants:

```
type Place; type Pt_Place is access Place;
type Place is record
    Contenu: Element;
    Suivant: Pt_Place;
end record;
```

Quel groupe d'instructions implante l'opération *insérer p après q*, où q repère un élément de la liste et p repère l'élément à insérer?

☐ `Q.Suivant := P.Suivant; P.Suivant := Q;`

☐ `P.Suivant := Q; Q.Suivant := P.Suivant;`

☒ `P.Suivant := Q.Suivant; Q.Suivant := P;`

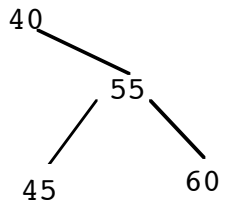
☐ `Q.Suivant := P; P.Suivant := Q.Suivant;`

☐ `Q.Suivant := P.Suivant; P.Suivant := Q.Suivant;`

Note : les autres possibilités entraînent la perte du contenu initial de `Q.Suivant`, sans l'avoir conservé au préalable.

9 Ajouter 45 dans l'arbre binaire de recherche suivant, et expliquer.

45 est plus grand que 40, il doit donc être mis dans le sous arbre droit. Il est plus petit que 55, il doit donc être mis dans le sous arbre gauche de 55. Il est donc mis en fils gauche de 55



10 On dispose d'un fichier de hachage qui contient 100 000 éléments. Chaque bloc disque peut contenir 40 éléments. La fonction de hachage a pour résultat un nombre entre 1 et 1000, équitablement répartis. A combien estimez-vous le nombre d'accès disque pour rechercher un élément ?

☐ 1000 ☐ 40 ☐ 10 ☒ 3 ☐ 1

Note□ en moyenne, les listes de hachage contiennent 100□000/1□000 soit 100 éléments, qui sont donc répartis sur 100/40=2,5 blocs.