SOLUTIONS SÉRIE 9 (Chapitres 10 et 11)

Question # 1

Démontrez que le nombre de déplacements de disques nécessaires pour résoudre le problème des tours de Hanoi (avec n disques) doit être supérieur ou égal à $2^n - 1$.

Solution :

Nous allons démontrer par induction que le nombre $M(n)$ de déplacements doit satisfaire $M(n) \geq 2^n - 1$.

Cas de base : pour $n = 1$, on a que $M(1) = 2^1 - 1 = 1$ comme il se doit.

Supposons que c'est vrai pour tout $i = 1, \ldots, n$. Démontrons que cela doit être vrai pour $i = n + 1$.

Pour pouvoir déplacer le plus grand disque, il faut que les n plus petits disques soient sur une autre tige. Mais le nombre de déplacements de disques pour effectuer cela, selon l'hypothèse inductive, doit être $\geq 2^n - 1$. Le déplacement du plus grand disque vers la tige destination requiert au moins un déplacement. Après avoir déplacé le plus grand disque à la tige de destination, il faut déplacer les n plus petits disques de leur tige à la tige destination (au dessus du plus grand disque) et cela nécessite au moins $2^n - 1$ déplacements selon l'hypothèse inductive. Le nombre $M(n + 1)$ de déplacements doit donc satisfaire :

$$M(n+1) \;\geq\; 2^n - 1 + 1 + 2^n - 1 \;=\; 2^{n+1} - 1\,.$$

Question # 2

Un algorithme qui utilise un nombre polynomial de fois un autre algorithme à temps polynomial est-il assurément un algorithme à temps polynomial ?

Solution :

Non. Considérez un algorithme A qui, pour une entrée x de n bits ne fait que produire la sortie x, x de $2n$ bits (constituée, par exemple, de la chaîne x concaténée avec elle-même). Le temps d'exécution de cet algorithme est en $\Theta(n)$ (donc polynomial). Supposons maintenant que la sortie de A est redirigée vers elle-même. La taille de $A(A(x))$ est alors de $4n$, la taille de $A(A(A(x)))$ est de $8n$, la taille de $A^k(x)$ est alors de $2^k n$ ce qui deviens $2^n n$ lorsque $k = n$. Il est donc important de s'assurer que la taille des instances générées par A reste bornée par un polynôme en la taille n de l'instance initiale.

Question # 3

Un problème p peut être résolu par un algorithme ayant un temps d'exécution $O(n^{\lg(n)})$. Laquelle des affirmations suivantes est vraie ?

A) Le problème est nécessairement traitable.

B) Le problème est nécessairement intraitable.

C) Aucune de ces réponses n'est vraie.

Solution :

A) Faux. Un problème est traitable s'il peut être résolut en temps polynomial. Premièrement, $n^{\lg(n)}$ croît infiniment plus rapidement que n'importe quel polynôme (on peut vérifier que $lim_{n\to\infty} \frac{n^k}{n^{\lg(n)}} = 0$). Deuxièmement, on ne peut pas savoir si cet algorithme résout le problème en temps polynomial en raison de la notation O : rien n'empêche l'agorithme d'être exactement en $n^{\lg(n)}$. On ne peut donc pas conclure s'il existe un algorithme qui résout le problème en temps polynomial ou non.

B) Faux. Un problème intraitable ne peut pas être résolut en temps polynomial. Tel que mentionné en (i), il est impossible de savoir si notre algorithme est polynomial ou non à cause de la notation O. Par exemple, rien n'empêche notre algorithme d'être en n^2 car $n^2 \in O(n^{\lg(n)})$. Or, même si notre algorithme n'était pas polynomial, il pourrait exister un autre algorithme qui résout le même problème en temps polynomial. On ne peut donc pas conclure qu'il n'existe aucun algorithme polynomial pour résoudre ce problème.

C) Vrai. On ne peut pas affirmer avec certitude que le problème est traitable ou intraitable.

Question # 4

Trouvez un minorant pour un algorithme devant lister tous les nombres premiers contenus dans un vecteur $V[1..n]$.

Solution :

Tout algorithme devant résoudre ce problème doit analyser chacun des n nombres dans le vecteur V. Il doit donc s'exécuter en temps $\Omega(n)$.

Question # 5

Trouvez un minorant pour un algorithme devant énumérer tous les mots de passe possibles formés de n caractères choisis parmi $\{a..z, A..Z, 0..9\}$.

Solution :

Il y a $2 \times 26 + 10 = 62$ caractères et donc 62^n mots de passe possibles de longueur n. Le minorant pour l'algorithme est donc $\Omega(62^n)$.

Question # 6

Trouvez un minorant pour un algorithme qui doit multiplier une matrice de dimensions $a \times b$ par une matrice de dimensions $b \times c$.

Solution :

Tout algorithme devant multiplier ces matrices doit lire les ab entrées de la première matrice de même que les bc entrées de la deuxième matrice. L'algorithme doit donc s'exécuter en $\Omega(ab + bc)$. De plus, cet algorithme doit produire une matrice de dimensions $a \times c$ contenant ac entrées. L'algorithme doit donc s'exécuter en $\Omega(ac)$. En combinant ces deux minorants, nous obtenons $\Omega(ab + bc + ac)$.

Question # 7

Considérez l'algorithme de force brute pour résoudre le problème de décider si un nombre n est premier ou composé (Algorithme 1). Cet algorithme est-il suffisant pour conclure que le problème mentionné appartient à la classe P ?

Algorithme 1 : ForceBruteEstCompose(n)

```
1 pour i = 2..⌊n/2⌋ faire
2     si n mod i = 0 alors
3         retourner «oui»
4 retourner «non»
```

Solution :

Non. Même si, à première vue, le temps d'exécution de l'algorithme semble être $O(n)$ ce n'est pas le cas. Ici, l'entrée est un entier n. La taille d'un entier est exprimée en nombre de bits utilisés pour coder n. Soit b le nombre de bits utilisés pour coder n. Étant donné que $b = \lfloor \lg n \rfloor + 1$, on obtient $O(n) = O(2^b)$. Donc notre algorithme n'est pas polynomial en la taille de l'entrée. Par conséquent, l'existence de cet algorithme n'est pas suffisante pour conclure que le problème mentionné appartient à la classe P.

Question # 8

Le problème de partition *se définit de la façon suivante : Soit A un ensemble d'entiers. Existe-t-il un sous-ensemble $B \subset A$ tel que la somme des éléments dans B est égale à la somme des éléments dans A qui ne sont pas dans B ?*

$$\sum_{b \in B} b = \sum_{c \in A \setminus B} c$$

Le problème du sac à dos décision *se définit de la façon suivante : Soit n objets avec des poids $w_1, \ldots, w_n$ et des valeurs $v_1, \ldots, v_n$. Soit un sac à dos de capacité maximale W et un nombre V. Existe-t-il un sous ensemble de ces objets tel que le poids total est au plus W et tel que la valeur totale est au moins V ?*

A) *Prouvez que le problème du sac à dos version décision appartient à la classe NP.*
Solution :
Le problème du sac à dos version décision appartient à la classe NP car il existe un système de preuves vérifiables en temps polynomial pour ce problème. Un certificat au problème du sac à dos version décision est un sous-ensemble d'objets S. Pour vérifier si S est bien une solution à l'instance x telle que

$$x = \langle [w_1, \ldots, w_n], [v_1, \ldots, v_n], W, V \rangle \,, \tag{1}$$

il faut vérifier si la somme des poids des objets dans S est inférieure ou égale à W et si la somme des valeurs des objets dans S supérieure ou égale à V. Cette vérification s'effectue en temps polynomial. En effet, il suffit de boucler sur les éléments de S pour calculer son poids total et sa valeur totale.

B) *Prouvez que le problème de partition est réductible polynomialement au problème de décision pour le problème du sac à dos.*
Solution :
Il faut exploiter une propriété du problème de partition. Soit B un certificat au problème de partition. Nous avons $\sum_{b \in B} b = \sum_{c \in A \setminus B} c = \frac{1}{2} \sum_{a \in A} a$.

On crée une instance du problème de sac à dos de la façon suivante. On crée un vecteur w de $|A|$ éléments tel que $w[i] = A[i]$. On crée ensuite un vecteur v de $|A|$ éléments tel que $v[i] = A[i]$. On fixe les variables W et V à la valeur $\frac{1}{2} \sum_{a \in A} a$.

Pour que la réduction soit valide nous devons prouver que

$$\textit{Partition}(A) = Vrai \iff \textit{Sac-à-Dos-Décision}(w, v, W, V) = Vrai$$

On prouve d'abord que s'il existe une solution au problème partition, le sac-à-dos a aussi une solution. Soit B le certificat du problème de partition. Il existe donc un sous-ensemble d'élément B dont la somme est $\frac{1}{2} \sum_{a \in A} a$. Ces éléments sont une solution au problème du sac-à-dos car leurs poids est égal à $W = \frac{1}{2} \sum_{a \in A} a$ (ce qui est au plus W) et leur valeur est égale à $V = \frac{1}{2} \sum_{a \in A} a$ (ce qui est au moins V).

On prouve ensuite que si le sac-à-dos a une solution, il existe bien une solution au problème de partition. Soit B l'ensemble des objets dans le sac à dos. On sait que la somme des poids est au plus $W = \frac{1}{2} \sum_{a \in A} a$ et la somme des valeurs est au moins $V = \frac{1}{2} \sum_{a \in A} a$. Puisque le poids d'un objet est égal à sa valeur, la solution du sac-à-dos n'a pas d'autre choix d'avoir un poids d'exactement W et une valeur d'exactement V.

$$V \leq \sum_{i \in B} v[i] = \sum_{i \in B} w[i] \leq W$$

Et puisque que $V = W = \frac{1}{2} \sum_{a \in A} a$, nous avons donc que les objets dans le sac-à-dos forme un certificat au problème de partition.

C) Sachant que le problème de partition est NP-complet, que pouvez-vous conclure pour le problème du sac à dos version décision ?

Solution :

Le problème du sac à dos version décision est dans la classe NP (prouvé en A). Le problème de partition est NP-complet. Il est aussi polyomialement réductible au problème du sac à dos décision (prouvé en B). Nous concluons donc que le problème du sac à dos décision est NP-complet.

Question # 9

Le problème de satisfiabilité *(SAT) se définit de la façon suivante : Soit $L = \{l_1, l_2, \ldots, l_n\}$, un ensemble de littéraux qui peuvent prendre des valeurs booléennes : `vrai` ou `faux`. Soit $C = \{c_1, c_2, \ldots, c_m\}$, un ensemble de clauses où chaque clause est une disjonction de littéraux. Existe-t-il une assignation de valeurs aux littéraux de L qui satisfasse la conjonction des clauses de C ?*

Par exemple, soit l'instance suivante :

$$L = \{l_1, l_2, l_3, l_4\}, \tag{2}$$

$$C = \{\{l_1, \neg l_2\}, \{\neg l_1, l_2, \neg l_3, l_4\}, \{\neg l_1, \neg l_3\}, \{l_4\}\}. \tag{3}$$

Cette instance représente le problème de satisfiabilité suivant :

$$(l_1 \vee \neg l_2) \wedge (\neg l_1 \vee l_2 \vee \neg l_3 \vee l_4) \wedge (\neg l_1 \vee \neg l_3) \wedge (l_4). \tag{4}$$

Une solution à cette instance est $l_1 =$ `vrai`, $l_2 =$ `vrai`, $l_3 =$ `faux`, $l_4 =$ `vrai`.

Le problème de satisfiabilité à 3 littéraux *(3SAT) est identique au problème SAT, mais avec la contrainte supplémentaire que toute clause $c \in C$ doit porter sur exactement 3 littéraux.*

Sachant que SAT est un problème NP-complet, prouvez que 3SAT est NP-complet.

Solution :

3SAT est dans la classe NP. Un certificat à 3SAT est une assignation de valeurs booléennes aux littéraux de L. Pour vérifier que ce certificat est valide il suffit de vérifier que les m clauses, qui contiennent au plus 3 littéraux, sont satisfaites. Il faut donc, pour chaque littéral de chacune des clauses, vérifier s'il est évalué à `vrai` (donc satisfait) considérant la solution. Si au moins un littéral par clause est satisfait alors la clause est évaluée à `vrai` (donc satisfaite). Si toutes les clauses sont satisfaites alors l'affectation est bien un certificat du problème. Le temps requis pour cette vérification est donc en $O(m)$.

Prouvons qu'il existe une réduction polynomiale de SAT vers 3SAT. Soit L l'ensemble de littéraux d'une instance SAT et soit C l'ensemble des clauses de cette même instance. Nous créeons L_3 l'ensemble des littéraux et C_3 l'ensemble des clauses d'une instance de 3SAT qui encode SAT ainsi :

1. *Ajouter tous les littéraux de L dans L_3.*
2. *Pour chacune des clauses $c_i \in C$ faire :*
   (a) *Si $|c_i| = 1$ alors ajouter deux nouveaux littéraux x_i et y_i dans L_3 et créer les clauses suivantes dans C_3 :*

$$c_i \cup \{x_i, y_i\},$$
$$c_i \cup \{x_i, \neg y_i\},$$
$$c_i \cup \{\neg x_i, y_i\},$$
$$c_i \cup \{\neg x_i, \neg y_i\}.$$

   (b) *Si $|c_i| = 2$ alors ajouter un littéral z_i dans L_3 et créer les clauses suivantes dans C_3 :*

$$c_i \cup \{z_i\},$$
$$c_i \cup \{\neg z_i\}.$$

(c) *Si* $|c_i| = 3$ *alors ajouter* c_i *à* C_3.

(d) *Si* $|c_i| \geq 4$ *alors*

i. *Posons* $c_i = \{l_1, \ldots, l_k\}$.

ii. *Ajouter les littéraux* $z_{i,1}, \ldots, z_{i,k-3}$.

iii. *Ajouter les clauses suivantes :*

$$\begin{aligned} &\{l_1, l_2, \neg z_{i,1}\}, \\ &\{z_{i,1}, l_3, \neg z_{i,2}\} \\ &\ldots, \\ &\{z_{i,k-4}, l_{k-2}, \neg z_{i,k-3}\} \\ &\{z_{i,k-3}, l_{k-1}, l_k\}. \end{aligned}$$

Dans notre algorithme de réduction, nous parcourons d'abord les n *littéraux de* L. *Ensuite, nous parcourons les* m *clauses de notre instance SAT qui contiennent au plus* n *littéraux. La réduction, telle que décrite, est donc en* $O(nm)$. *Nous avons donc un algorithme polynomial pour réduire une instance de SAT en une instance de 3SAT.*

Pour que la réduction soit valide nous devons prouver que

$$\textit{SAT}(L, C) = \texttt{vrai} \iff \textit{3SAT}(L_3, C_3) = \texttt{vrai}$$

Supposons que toutes les clauses de C *sont satisfaites. Prouvons que toutes les clauses de* C_3 *sont satisfaites. Prenons chacune des clauses* $c_i \in C$ *:*

— *Si* $|c_i| = 1$ *alors les* 4 *clauses ajoutées à* C_3 *seront satisfaites car dans chacune de ces* 4 *clauses l'unique littéral de* c_i *est présent.*

— *Si* $|c_i| = 2$ *alors les* 2 *clauses ajoutées à* C_3 *seront satisfaites car elles contiennent tous les littéraux de* c_i.

— *Si* $|c_i| = 3$ *alors* c_i *est ajoutée directement à* C_3. *Elle reste donc satisfaite.*

— *Si* $|c_i| \geq 4$ *alors un ensemble de clauses est ajouté à* C_3.

  — *Supposons que* l_1 *ou* l_2 *est à* `vrai`. *Dans ce cas, il suffit d'affecter* `vrai` *aux littéraux* $z_{i,j}$ *(*$\forall 1 \leq j \leq k-3$*) pour satisfaire toutes les clauses ajoutées.*

  — *Supposons* l_{k-1} *ou* l_k *eset à* `vrai`. *Dans ce cas, il suffit d'affecter* `faux` *aux littéraux* $z_{i,j}$ *(*$\forall 1 \leq j \leq k-3$*) pour satisfaire toutes les clauses ajoutées.*

  — *Supposons que l'un des littéraux* $l_{j'}$ *(*$3 \leq j' \leq k-2$*) soit à* `vrai`. *Dans ce cas, il suffit d'affecter* `faux` *aux littéraux* $z_{i,j}$ *qui "précèdent"* $l_{j'}$ *(i.e.,* $j < j'$ *et* `vrai` *aux littéraux* $z_{i,j}$ *qui "succèdent"* $l_{j'}$ *(i.e.,* $j > j'$*).*

Nous avons donc que toutes les clauses de notre instance transformée C_3 *sont satisfaites lorsque les clauses de l'instance originale sont satisfaites.*

Supposons que toutes les clauses de C_3 *sont satisfaites. Prouvons que toutes les clauses de* C *sont satisfaites. Il est suffisant de remarquer qu'il est impossible de satisfaire les clauses de* C_3 *sans satisfaire les littéraux de* L, *i.e., ceux de l'instance SAT originale. Ainsi, si toutes les clauses dans* C_3 *de l'instance transformée sont satisfaites, les clauses de l'instance originale dans* C *sont satisfaites.*

Puisque 3SAT est dans NP *et puisqu'il existe une réduction polynomiale de SAT, un problème* NP*-complet, vers 3SAT, nous concluons que 3SAT est* NP*-complet.*
CQFD.

Question # 10

Utilisez la technique du retour arrière pour trouver un cycle hamiltonien dans le graphe de la figure 1.

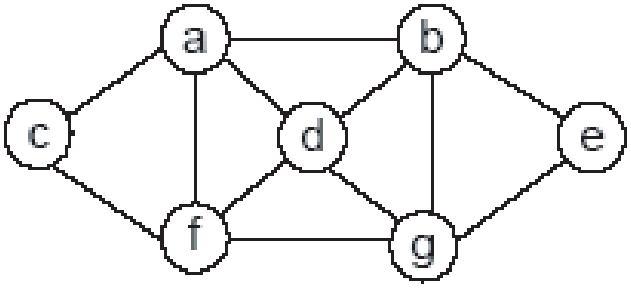

FIGURE 1 – Trouver un cycle hamiltonien sur ce graphe

Solution :
La technique du retour arrière nous donne la solution de la figure 2.

Question # 11
Utilisez la technique du retour arrière pour générer toutes les permutations de $\{1, 2, 3, 4\}$.

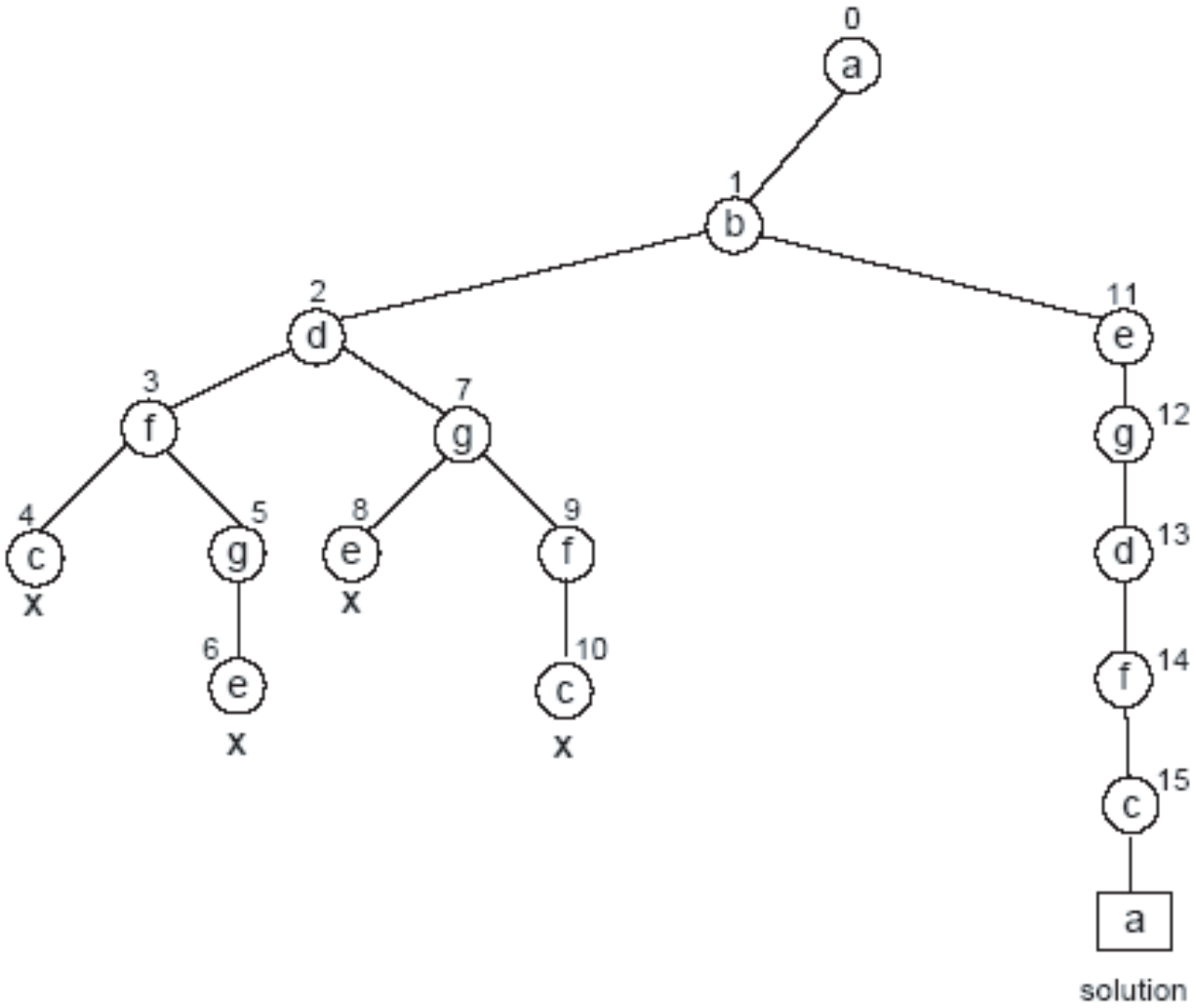

FIGURE 2 – La technique du retour arrière pour trouver un cycle hamiltonien sur le graphe de la figure 1

Solution :
La technique du retour arrière nous donne l'arbre de la figure 3.

Question # 12
Quelle structure de données utiliseriez-vous pour garder la trace des noeuds actif dans un algorithme de "branch-and-bound" pour la version "best-first" ?
Solution :
Un monceau et un monceau inversé pour les problèmes de maximisation et de minimisation respectivement.

Question # 13
Résolvez l'instance suivante du problème du sac à dos avec l'algorithme de "branch-and-bound" pour $W = 16$ *:*

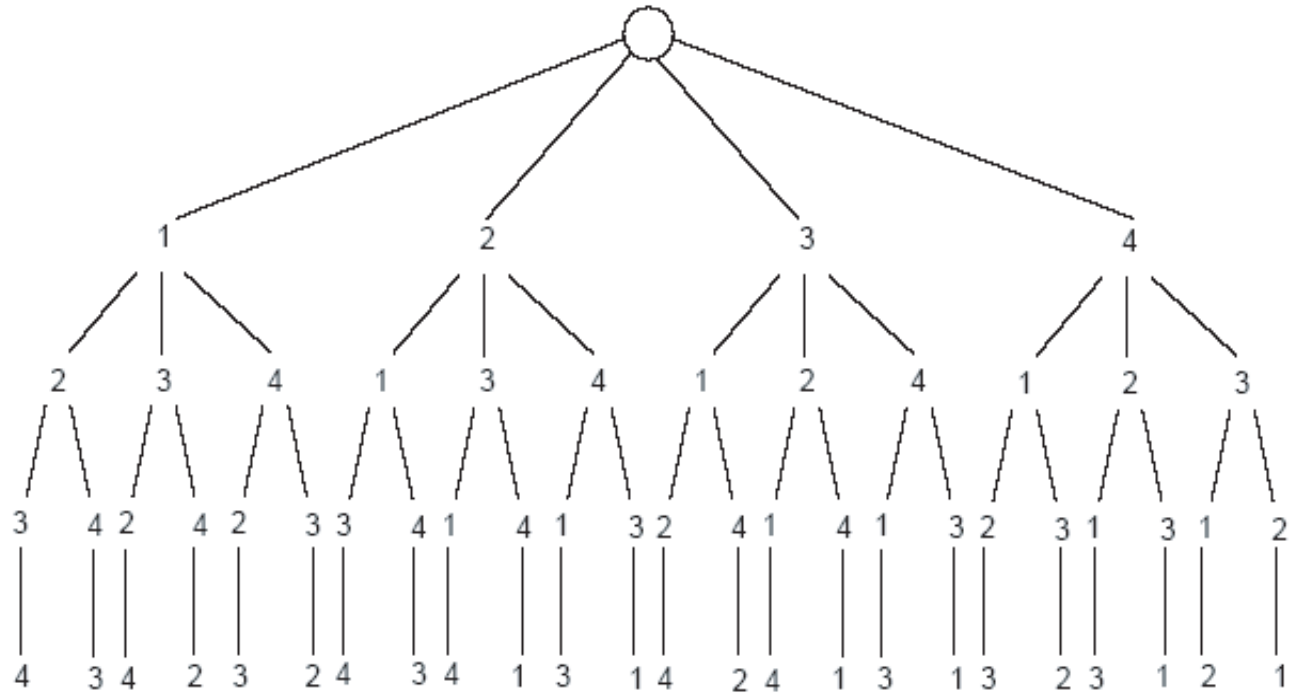

FIGURE 3 – Technique du retour arrière sur les permutations de 1, 2, 3 et 4.

objet	poids	valeur
1	10	100
2	7	63
3	8	56
4	4	12

Solution :

La solution optimale est : $\{$*objet 2, object 3*$\}$ *pour une valeur de 119. Voir la figure 4.*

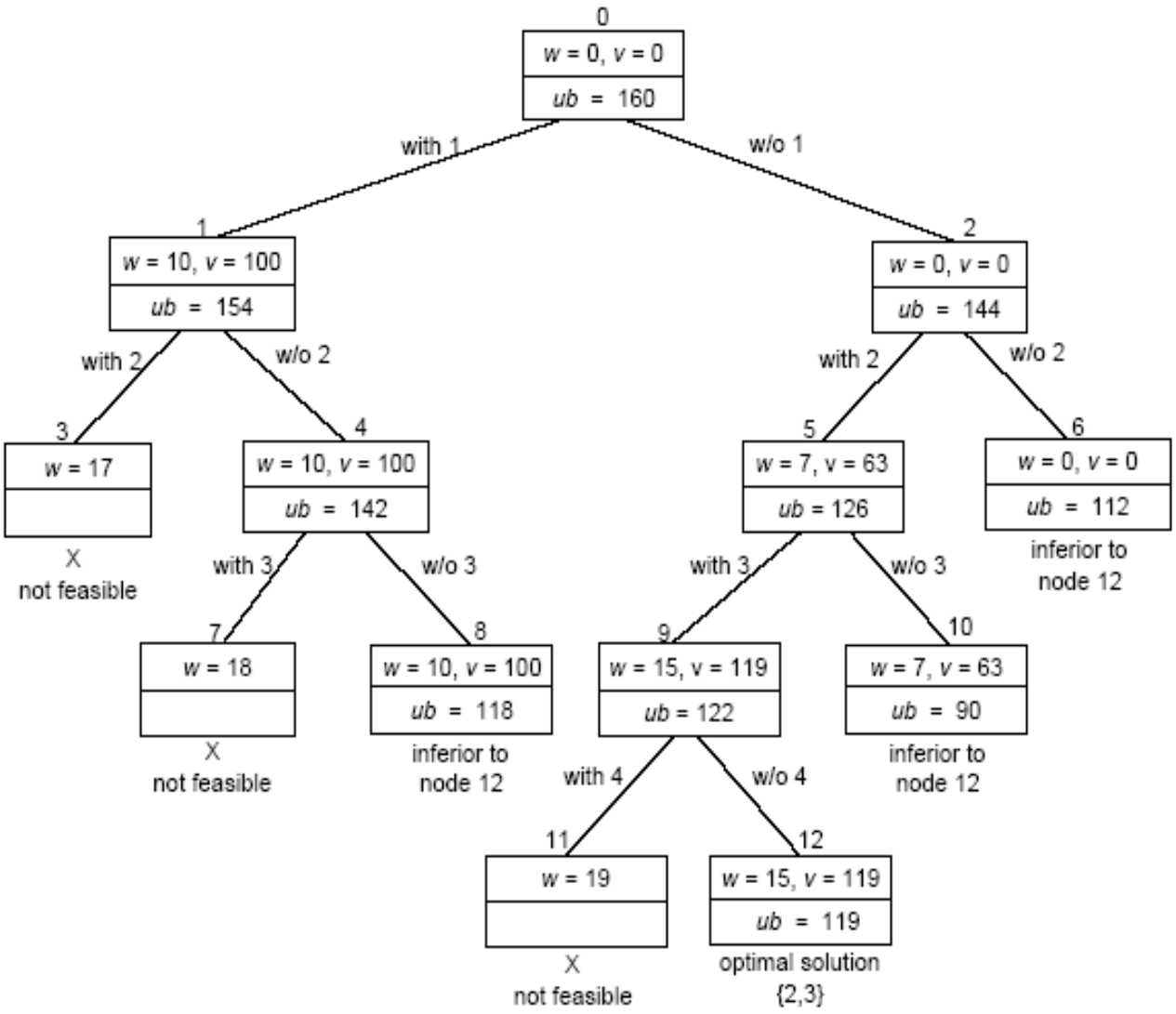

FIGURE 4 – Branch-and-bound sur le problème du sac à dos

Question # 14

A) Pour le problème du sac à dos, élaborez une façon de calculer les bornes qui est meilleure que celle utilisée au Chapitre 11.

Solution :

On suppose que les objets sont triés en ordre décroissant de leur rapport : $v_1/w_1 \geq v_2/w_2 \ldots \geq v_n/w_n$ *et on considère les objets dans cet ordre lors de la sélection. Soit un sous-ensemble* $S = \{i_1, \ldots, i_k\}$ *représenté par un noeud dans l'arbre de "branch-and-bound"; le poids et la valeur de* S *sont* $w(S) = w_1 + \ldots + w_k$ *et* $v(S) = v_1 + \ldots v_k$ *respectivement. On peut calculer la borne supérieure* $Ub(S)$ *de n'importe quel sous-ensemble obtenu en ajoutant un objet aux objets de* S *de la façon suivante. On ajoute à* $v(S)$ *la valeur des objets suivants* i_k *tant que le poids total ne dépasse pas la capacité du sac. Lorsqu'on rencontre un objet qui viole cette condition, on détermine la fraction* f *requise de cet objet pour remplir le sac et on ajoute la valeur de cette fraction de l'objet à notre borne supérieur. Ex. :* $Ub(S) = v(S) + v_{k+1} + v_{k+2} + \frac{1}{2}v_{k+3}$.

B) *Utilisez votre nouveau calcul de borne pour résoudre l'instance de la question 13.*
Solution :
La solution optimale est : $\{$*objet 2, object 3*$\}$ *pour une valeur de 119. Voir la figure 5.*

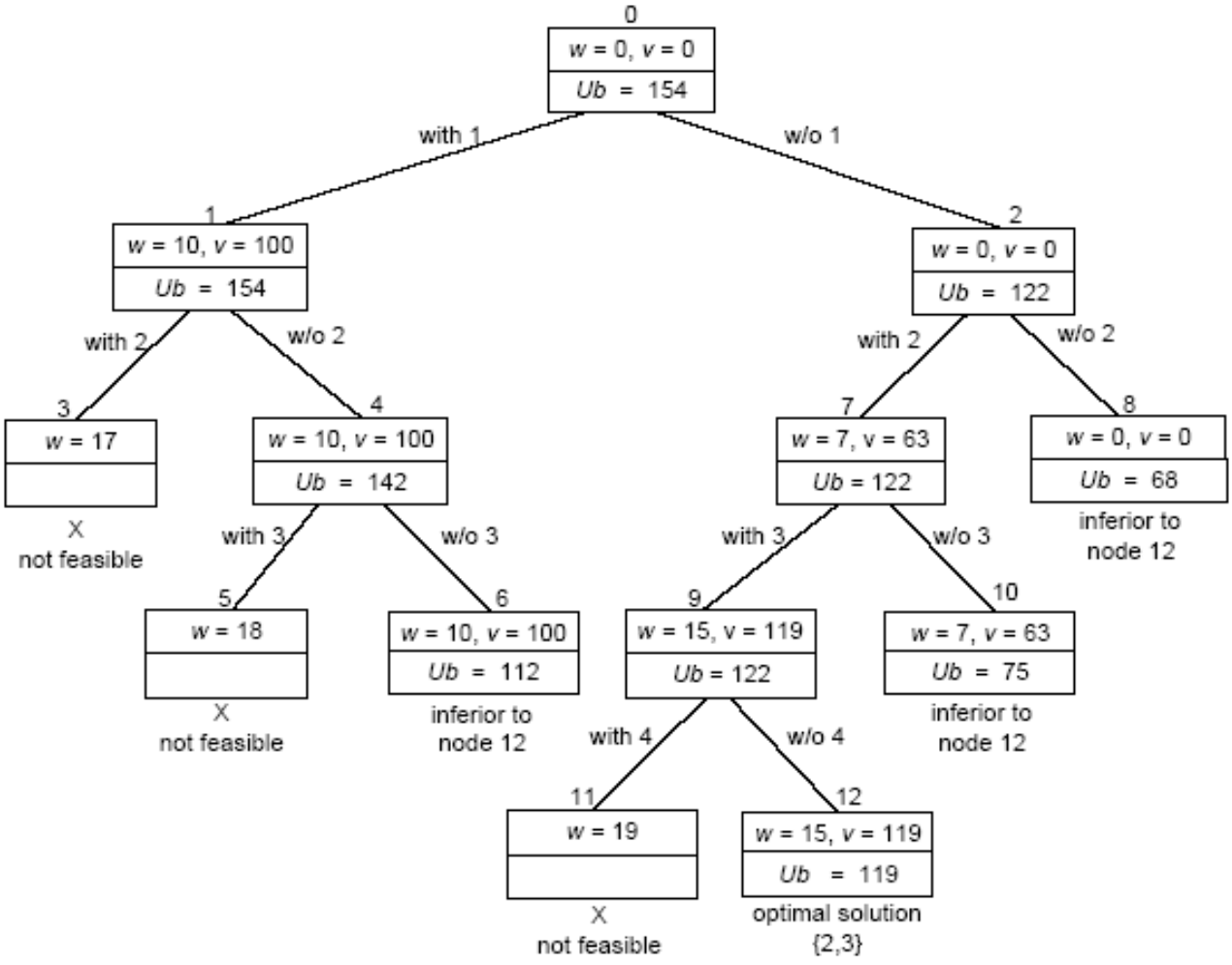

FIGURE 5 – Branch-and-bound avec une borne améliorée sur le problème du sac à dos