Complexité d'algorithmes

Question 1. Les tableaux et les listes

Question 1.1 Evaluer la complexité d'un algorithme qui à partir de deux listes triées A et B construit une liste unique contenant les éléments des deux listes A et B.

Question 1.2 Etant donné un tableau trié d'entiers, évaluer la complexité d'un algorithme qui teste si l'entier x est contenu dans le tableau.

Question 1.3 Soit T un tableau de p entiers. Décrire un algorithme naïf qui retourne le plus petit élément, ainsi que le second plus petit élément d'un tableau en $\frac{3p}{2} + O(1)$ comparaisons. ***Indication : diviser le tableau en deux.***

Il faut décomposer le tableau T en deux tableaux l'un composé des $\lceil \frac{p}{2} \rceil$ premiers éléments et l'autre des $\lfloor \frac{p}{2} \rfloor$ dernier éléments.

1. Il faut calculer les minimums de ces tableaux
   $(min_1, ind_1) \leftarrow \mathcal{A}_1(T[1, \dots \lceil \frac{p}{2} \rceil])$;
   $(min_2, ind_2) \leftarrow \mathcal{A}_1(T[\lceil \frac{p}{2} \rceil + 1, \dots, p])$;
2. Ensuite, il faut parmi ces deux valeurs calculer le minimum
   $$m_1 \leftarrow \min(min_1, min_2)$$
3. Maintenant il faut chercher la deuxième valeur plus petite dans le sous-tableau qui contenant la valeur m
   si $m_1 = min_1$, alors $m' \leftarrow \mathcal{A}_1(T[1, \dots \lceil \frac{p}{2} \rceil], ind_1)$
   sinon $m' \leftarrow \mathcal{A}_1(T[\lceil \frac{p}{2} \rceil + 1, \dots, p], ind_2)$
4. Ensuite, il faut le comparer avec le minimum de l'autre tableau.
   $$m_2 = \min(\{min_1, min_2, m'\} \backslash \{m_1\})$$
5. Retourner m_1, m_2

Complexité : Etape (a) requière $p + O(1)$ opérations.
Etape (b) requière $p/2 + O(1)$ opérations

Question 2. **Comment gérer un cinéma**

Un cinéma possède s salles de cinéma. Chaque semaine, le cinéma propose une liste de films à voir. Chaque film correspond à un nombre fini de séances. Chaque séance se déroule selon un horaire (intervalle de temps) précis. Les films n'ont pas tous la même durée. On précise qu'il est impossible de changer les horaires des séances.

Un étudiant qui a une journée de libre veut voir le maximum de films pendant cette journée (il peut voir un même film plusieurs fois). Ce problème peut être résolu par un algorithme glouton :
Entrée : Un ensemble $\mathcal{I}$ des séances : chaque séance i est caractérisée par l'intervalle (d_i, f_i).
Sortie Un ensemble S des séances.

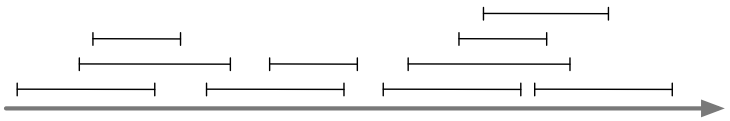


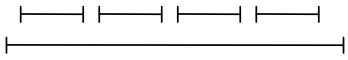
FIGURE 1 – Planning des séances de cinéma

1. Classer les intervalles par valuations v croissantes : $v(i_1) \leq v(i_2) \leq \cdots \leq v(i_n)$
2. Initialiser la recherche avec $S = \emptyset$;
3. Tant que $\mathcal{I}$ n'est pas vide faire
   (a) Choisir $i \in \mathcal{I}$ qui a la plus petite valuation.
   (b) Ajouter i dans S
   (c) Supprimer toutes les séances de $\mathcal{I}$ qui ne sont pas compatible avec i
4. Retourner S

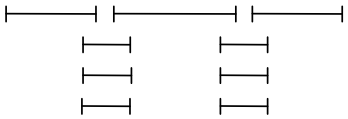
Maintenant il reste à déterminer la valuation.

Question 2.1 Donner un exemple où l'algorithme ne retourne pas la solution optimale :

1. si la valuation v d'un intervalle est sa date de début (d_i).



2. si la valuation v d'un intervalle est sa durée ($f_i - d_i$).



Question 2.2 Montrer que l'algorithme est optimal si la valuation v d'un intervalle est sa date de fin (f_i).

Soit S la solution retournée par le glouton. Soit S^* une solution optimale.

Notons j le premier élément de S inséré par l'algorithme glouton. Nous allons prouver qu'il existe une solution optimale qui contient élément j. Si S^* contient élément j, alors il n'y a rien à prouver.

Sinon, notons i l' élément de S^* tel que i est l'élément de S^* ayant la valeur f_i la plus petite.

Considérons l'ensemble $S' = S^* - \{i\} \cup \{j\}$. Tout d'abord, nous pouvons remarquer que $f_j \leq f_i$: j se termine avant i. Si ce n'était pas le cas, l'algorithme glouton aura choisit la séance i. De plus, cela signifie aussi que la séance j n'intersecte pas avec les séances de $S^* - \{i\}$ car toutes les autres séances de $S^* - \{i\}$ commence après la date f_i.

Comme $|S'| = |S^*|$, S' est une solution optimale qui contient j. Pour prouver que S est une solution optimale, on se restreint aux solutions optimales contenant j et on raisonne par récurrence.

Les ensembles S^* et S ont les k premiers éléments identiques (mais ils différent aux $k+1$ élément).

Soit i le $k+1$ élément de S^* (il n'est pas dans S)

Soit j le $k+1$ élément de S (il n'est pas dans S^*)

Nous allons prouver que i peut être remplacé par le premier élément j de S qui n'est pas dans S^*.

En effet il suffit de remarquer que $f_j \leq f_i$: j se termine avant i. Si ce n'était pas le cas, l'algorithme glouton aura choisit la séance j.

On obtient une autre solution $S' = S^* - \{i\} \cup \{j\}$ telle que $|S'| = |S^*|$ et donc toujours optimale.

Les graphes

Terminologie Un ***graphe non orienté*** G est un couple (V, E) où V est un ensemble de sommets (cercles) et E est un ensemble d'arêtes (trait entre deux cercles). Notons $n = |V|$, $m = |E|$. Les sommets u et v sont dits ***voisins*** s'il y a une arête entre u et v. Nous considérons les notations suivantes.

1. Le ***degré*** de v dans G que l'on notera $d_G(v)$ est le nombre de voisins de v.
2. $\Gamma_G(v)$ est l'ensemble des sommets voisin de v dans G.
3. $G\backslash\{v\}$ est le graphe G privé du sommet de v : son ensemble de sommets est $V\backslash\{v\}$ et son ensemble d'arêtes est $E\backslash\{(u, v) : u \in \Gamma_G(v)\}$.

Question 3. Etant donné un graphe $G = (V, E)$ et un ensemble d'arêtes $M \subseteq E$, évaluer la complexité d'un algorithme qui teste si M est un couplage de G.

Considérons l'algorithme glouton **Entrée :** un graphe $G = (V, E)$, un ensemble d'arêtes
Sortie : un booleen b

1. créer un tableau $Couvert$ de n booléens
2. $b \leftarrow vrai$;
3. Pour chaque sommet v du graphe $Couvert[v] \leftarrow faux$;
4. Pour chaque arête $e = (u, v)$ de M

Si $(Couvert[v] = Couvert[u] = faux)$ alors $Couvert[v] \leftarrow vrai$;$Couvert[u] \leftarrow vrai$;

— sinon Retourner $faux$;

5. Retourner $vrai$;

Instruction 1 : $O(n)$ opérations

Instruction 2 : $O(1)$ opérations

Instruction 3 : $O(n)$ opérations

Instruction 2 : $O(|M|)$ opérations Le nombre d'arêtes de M est majoré par $\frac{n}{2}$.

Donc au total : $O(n)$ opérations

Question 4. Dans un graphe orienté sans boucle, à n sommets, un ***trou noir*** est un sommet de degré sortant nul, et de degré entrant $n-1$. Donner un algorithme qui examine $O(n)$ entrées de la matrice d'adjacence pour décider si un graphe sans boucle possède un trou noir.

Observons que si un graphe contient un trou noir, il peut en contenir qu'un seul.

Observons que la présence d'un arc $x \rightarrow y$ dans G, implique que x n'est pas un trou noir mais que y peut l'être. Et inversement, l'absence de l'arc $x \rightarrow y$ dans G, implique que y n'est pas un trou noir mais que x peut l'être. Tester si G possède l'arc $x \rightarrow y$ implique qu'on peut supprimer au moins un sommet parmi x ou y de la liste des candidats à être un trou noir.

1. $c \leftarrow 0$
2. Pour i allant 1 à $n-1$ faire si il y a d'arc $c \rightarrow i$ alors $c \leftarrow i$
3. si $c == 0$ alors le sommet 0 a 0 arc sortant
4. sinon
   (a) Pour i allant 0 à $c-1$ faire si il y a d'arc $c \rightarrow i$, alors Pas de trou noir dans le graphe $c \leftarrow null$
5. si $c \neq null$, alors c peut être un trou noir.

Cette procédure coûte $2n$ opérations. (Le pire cas est quand le trou noir est le sommet $n-1$) Maintenant, si il existe un sommet c qui ne possède pas d'arc sortant, il faut vérifier que le sommet c a $n-1$ arcs entrants

1. $Okay \leftarrow$ vrai
2. pour i de 0 à $n-1$ faire
   (a) si il n'y a pas d'arc $i \rightarrow c$, alors Okay $\leftarrow$ faux
3. Si $(Okay = vrai)$ alors c est bien un trou noir.

Cette vérification se fait bien en $\mathcal{O}(n)$ opérations.

La coupe maximum d'un graphe

Nous allons considérer un graphe non-orienté $G = (V, E)$ de n sommets, ayant une fonction de poids sur les arêtes $w : E \to \mathbb{N}$.

Soit A un sous-ensemble de sommets. Une ***coupe issue de*** A ***noté*** $\mathbf{c(A, B)}$ d'un graphe non-orienté $G = (V, E)$ est un ensemble d'arêtes qui partagent G en deux sous-ensembles disjoints et distincts (A et $B = V\backslash A$). Plus formellement, pour $B = V\backslash A$,

$$c(A, B) = \{(u, v) \in E \text{ tel que } u \in A,\ v \in B\}$$

Cet exercice traite le problème d'optimisation COUPE MAX : nous avons en entrée un graphe non orienté $G = (V, E)$ avec un poids $w(e)$ sur chaque arête e, et nous voulons partager les sommets en deux ensembles A et $B = V\backslash A$ afin que le poids total des arêtes de la coupe $c(A, B)$ soit aussi grand que possible.

Notation : Soit A et B deux ensembles de sommets disjoints. Le ***poids*** de la coupe $c(A, B)$ ***noté*** $\mathbf{w(A, B)}$ est la somme des poids des arêtes de $c(A, B)$. Nous noterons par $\mathcal{W}(\mathbf{A})$, la somme des poids des arêtes dont les deux extrémités sont dans A. Plus formellement,

$$w(A, B) = \sum_{e \in c(A,B)} w(e) = \sum_{u \in A, v \in B, (u,v) \in E} w(u, v) \qquad \text{et } \mathcal{W}(A) = \sum_{u \in A, v \in A, (u,v) \in E} w(u, v)$$

Nous dirons qu'un sommet u est α-***content*** pour la coupe (A, B) dans le graphe G si et seulement si

$w(A\backslash\{u\}, B \cup \{u\}) \leq \alpha w(A, B)$ si $u \in A$ ou $w(A \cup \{u\}, B\backslash\{u\}) \leq \alpha w(A, B)$ si $u \in B$

Considérons le graphe biparti complet de 8 sommets (voir la figure 2). Toutes les arêtes de ce graphe ont un poids égal à 1 : i.e. $\forall e \in E$, $w(e) = 1$.

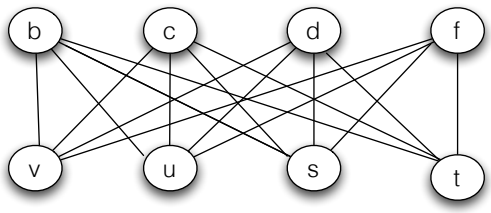


FIGURE 2 – Graphe biparti complet de huit sommets. Toutes ses arêtes ont un poids égal à 1.

Question 5. Compléter les trois tableaux :

A	$B = V\backslash A$	$w(A, V\backslash A)$
$\{b\}$	$\{c, d, f, v, u, s, t\}$	4
$\{b, v\}$		
$\{b, c, v\}$		
$\{b, c, d, f\}$		

A	B	$A\backslash\{b\}$	$w(A\backslash\{b\}, B \cup \{b\})$	$w(A, B)$
$\{b, c, d, f\}$	$V\backslash A$	$\{c, d, f\}$	12	16
$\{b, c, d, v\}$	$V\backslash A$			
$\{b, u, s, t\}$	$V\backslash A$			

A	B	$\mathcal{W}(A)$	$\mathcal{W}(B)$	b est 1-content
$\{b,c,d,f\}$	$V\backslash A$	0	0	oui car ...
$\{b,c,d,v\}$	$V\backslash A$			
$\{b,u,s,t\}$	$V\backslash A$			

A	$B = V\backslash A$	$w(A, V\backslash A)$
$\{b\}$	$\{c,d,f,v,u,s,t\}$	4
$\{b,v\}$	$\{c,d,f,u,s,t\}$	6
$\{b,c,v\}$	$\{d,f,u,s,t\}$	2+3+3
$\{b,c,u\}$	$\{d,f,v,s,t\}$	8
$\{b,v,u\}$	$\{c,d,f,s,t\}$	8
$\{b,c,v,u\}$	$\{d,f,s,t\}$	8
$\{b,c,d,f\}$	$\{v,u,s,t\}$	16

A	B	$A\backslash\{b\}$	$w(A\backslash\{b\}, B\cup\{b\})$	$w(A,B)$
$\{b,c,d,f\}$	$V\backslash A$	$\{c,d,f\}$	12	16
$\{b,c,d,v\}$	$V\backslash A$	$\{c,d,v\}$	8	10
$\{b,c,u,v\}$	$V\backslash A$	$\{c,u,v\}$	8	8
$\{b,u,s,t\}$	$V\backslash A$	$\{u,s,t\}$	12	10

A	B	$\mathcal{W}(A)$	$\mathcal{W}(B)$	b est 1-content
$\{b,c,d,f\}$	$V\backslash A$	0	0	oui car $w(A\backslash\{b\}, B\cup\{b\}) \leq \alpha w(A,B)$
$\{b,c,d,v\}$	$V\backslash A$	3	3	oui car $w(A\backslash\{b\}, B\cup\{b\}) \leq \alpha w(A,B)$
$\{b,c,u,v\}$	$V\backslash A$	4	4	oui car $w(A\backslash\{b\}, B\cup\{b\}) \leq \alpha w(A,B)$
$\{b,u,s,t\}$	$V\backslash A$	3	3	non car $w(A\backslash\{b\}, B\cup\{b\}) > \alpha w(A,B)$

Considérons l'algorithme suivant Algorithme MAX-CUT-LOCAL :

Entrée : Un graphe $G = (V, E)$ et une fonction poids $w : E \to \mathbb{N}$

Sortie : Deux sous-ensembles de sommets A et B.

1. Choisir une partition arbitraire de sommets (A, B) de V.
2. Tant qu'il existe un sommet v qui n'est pas α-***content***
   — Si v est dans A, alors $A \leftarrow A\backslash\{v\}$ et $B \leftarrow B \cup \{v\}$
   sinon $B \leftarrow B\backslash\{v\}$ et $A \leftarrow A \cup \{v\}$
3. Retourner les deux ensembles (A, B)

Question 5.1 Exécuter l'algorithme ayant en entrée le graphe de la figure 1 sachant que $\alpha = 1$ et que la partition initiale est la suivante

1. $A = \{b, c, d\}$ et $B = \{v, u, s, t, f\}$
2. $A = \{v, u, b, c\}$ et $B = \{d, f, s, t\}$

Pour l'exécution sur la partition initiale $A = \{v, u, b, c\}$ et $B = \{d, f, s, t\}$:
Tous les sommets sont contents. L'algorithme retourne cette partition.

Pour l'exécution sur la partition initiale $A = \{b, c, d\}$ et $B = \{v, u, s, t, f\}$: seul le sommet f n'est pas content. L'algorithme modifie la partition. Elle devient $A = \{b, c, d, f\}$ et $B = \{v, u, s, t\}$. Ici, dans la nouvelle partition, tous les sommets sont contents. L'algorithme retourne cette partition.

Par la suite, la solution par l'algorithme MAX-CUT-LOCAL retourne un couple de 2 sous-ensembles de sommets que l'on notera (A, B) tandis qu'une partition correspondant à une coupe maximale sera notée par (A^*, B^*) (c'est-à-dire la coupe $c(A^*, B^*)$ est telle que $w(A^*, B^*)$ soit le grand possible).

Question 5.2 Donner une relation entre $\mathcal{W}(E)$, $\mathcal{W}(A)$, $\mathcal{W}(B)$, $w(A, B)$.

$$\mathcal{W}(E) = \mathcal{W}(A) + \mathcal{W}(B) + w(A, B).$$

Question 5.3 Montrer que pour tout sommet $u \in A$, on a

$$w(\{u\}, A) \leq w(\{u\}, B) + (\alpha - 1)w(A, B).$$

Indication : *u* ***est*** *α***-content.**

Par définition on a $w(A\backslash\{u\}, B \cup \{u\}) = w(A, B) + w(\{u\}, A) - w(\{u\}, B)$. Comme u est α-content, on a $w(A\backslash\{u\}, B \cup \{u\}) \leq \alpha w(A, B)$ si $u \in A$. En combinant les deux équations, on obtient :

$$w(A, B) + w(\{u\}, A) - w(\{u\}, B) \leq \alpha w(A, B)$$

Nous allons supposer pour les prochaines questions que $\alpha = 1$.

Question 5.4 Montrer que pour tout sommet $u \in A$, on a $w(\{u\}, B) \geq \frac{w(\{u\}, \Gamma_G(u))}{2}$.

D'après la question précédente et le fait que $\alpha = 1$, on a

$$w(\{u\}, A) \leq w(\{u\}, B) + 0 \times w(A, B)$$

Commme $w(\{u\}, \Gamma_G(u)) = w(\{u\}, A\backslash\{u\}) + w(\{u\}, B)$, on a

$$w(\{u\}, \Gamma_G(u)) \leq 2 * w(\{u\}, B)$$

Question 5.5 Montrer que $2w(A, B) \geq \mathcal{W}(E)$ et que $2w(A, B) \geq w(A^*, B^*)$.

Comme une arête a deux extrémités on a

- pour A : $w(A, B) = \sum_{u \in A} w(\{u\}, B) \geq \sum_{u \in A} \frac{w(\{u\}, \Gamma_G(u))}{2}$
- pour B : $w(A, B) = \sum_{u \in B} w(\{u\}, A) \geq \sum_{u \in B} \frac{w(\{u\}, \Gamma_G(u))}{2}$

En sommant, on obtient $2w(A, B) \geq \sum_{u \in V} \frac{w(\{u\}, \Gamma_G(u))}{2}$.
Il suffit de constater que $\mathcal{W}(E) \geq w(A^*, B^*)$

Question 5.6 Quelle est la complexité de l'algorithme ? (*indication : évaluer au pire le nombre d'itérations de l'algorithme en fonction du nombre et du poids des arêtes*)

Nombre d'instructions

- L'instruction (1) peut se faire en $\mathcal{O}(n)$ opérations
- Le nombre de fois qu'on peut rentrer dans la boucle "tant que" est majoré par $\mathcal{W}(E)$.
- La condition du "tant que" peut se faire en $\mathcal{O}(\mathcal{W}(E))$ opérations (Mais cela peut se réaliser plus rapidement)

La complexité de l'algorithme est égale à $\mathcal{O}(|E|\mathcal{W}(E))$

Nous ne pouvons pas réduire cet interval, puisque cet algorithme peut retourner une solution à distance 2 de l'optimal. Il suffit de considérer l'exemple de la partie d'entrainement. Supposons que l'algorithme choisit lors de l'instruction 1 comme ensemble $A = \{b, c, v, u\}$ et $B = \{d, f, s, t\}$. Comme tous les sommets sont α-contents, alors l'algorithme retourne la coupe (A, B) avec $w(A, B) = 8$. La solution optimale (A^*, B^*) est la suivante :$A^* = \{b, c, d, f\}$ et $B^* = \{u, v, s, t\}$. Donc nous avons $2w(A, B) = w(A^*, B^*)$.

Graphes ayant une fonction de poids quelconque.

Nous considérons un réel arbitraire $\epsilon > 0$. Nous allons supposer par la suite que

$$\alpha = 1 + \frac{2\epsilon}{n}.$$

Question 5.7 Soit (A, B) la solution obtenue par l'algorithme. Montrer que

1. $\sum_{u\in A} w(\{u\}, A) \leq w(A, B) + \frac{|A|2\epsilon}{n} w(A, B)$
2. $\sum_{u\in B} w(\{u\}, B) \leq w(A, B) + \frac{|B|2\epsilon}{n} w(A, B)$
3. $\mathcal{W}(A) + \mathcal{W}(B) \leq (1 + \epsilon)w(A, B)$

   En appliquant la question précédente ($w(\{u\}, A) \leq w(\{u\}, B)+(\alpha-1)w(A, B)$), on obtient

$$w(\{u\}, A) \leq w(\{u\}, B) + \left(\frac{2\epsilon}{n}\right) w(A, B)$$

   En sommant sur tous les sommets de A (resp. de B), on obtient la première (resp. deuxième) équation.
   En sommant les deux inégalités précédentes (1) et (2), on obtient

$$\sum_{u\in A} w(\{u\}, A) + \sum_{u\in B} w(\{u\}, B) \leq 2w(A, B) + 2\epsilon\frac{|A| + |B|}{n} w(A, B) \qquad (1)$$

Remarquons que $\sum_{u\in A} w(\{u\}, A) = 2\mathcal{W}(A)$ et $\sum_{u\in B} w(\{u\}, B) = 2\mathcal{W}(B)$.
Nous pouvons réécrire l'équation (1)

$$2\mathcal{W}(A) + 2\mathcal{W}(B) \leq 2w(A, B) + 2\epsilon w(A, B)$$

Donc, nous obtenons

$$\mathcal{W}(A) + \mathcal{W}(B) \leq (1 + \epsilon)w(A, B)$$

Question 5.8 Montrer que $\mathcal{W}(E) \leq (2 + \epsilon)w(A, B)$.

Rappellons que $\mathcal{W}(E) = \mathcal{W}(A) + \mathcal{W}(B) + w(A, B)$.
D'après la question précédente,

$$\mathcal{W}(A) + \mathcal{W}(B) + w(A, B) \leq (1 + \epsilon)w(A, B) + w(A, B).$$

Nous pouvons déduire que $\mathcal{W}(E) \leq (2 + \epsilon)w(A, B)$.

Question 5.9 Montrer que $w(A^*, B^*) \leq (2 + \epsilon)w(A, B)$. Qu'en déduisez-vous ?

Il suffit de constater que $\mathcal{W}(E) \geq w(A^*, B^*)$. Comme $(2 + \epsilon)w(A, B) \geq \mathcal{W}(E)$, nous en déduisons

$$(2 + \epsilon)w(A, B) \geq \mathcal{W}(E) \geq w(A^*, B^*)$$

Soit (A_0, B_0) la coupe initiale utilisée par l'algorithme. Nous allons considérer que $A_0 = \{v^{max}\}$ et $B_0 = V\backslash A_0$ tel que le sommet v^{max} respecte la condition suivante

$$\forall v \in V, w(\{v^{max}\}, V\backslash\{v^{max}\}) \geq w(\{v\}, V\backslash\{v\}) \tag{2}$$

Question 5.10 Montrer que $w(A_0, B_0) \geq \frac{2}{n}\mathcal{W}(E)$.

Notons $w(A_0, B_0) = w(\{v^{max}\}, V\backslash\{v^{max}\})$.
Rappellons que $\sum_{v\in V} w(\{v\}, V\backslash\{v\}) = 2\mathcal{W}(E)$.
Par définition du sommet v^{max}, nous obtenons $nw(\{v^{max}\}, V\backslash\{v^{max}\}) \geq 2\mathcal{W}(E)$.
Donc

$$nw(A_0, B_0) \geq 2\mathcal{W}(E)$$

Soit (A_t, B_t) la coupe de l'algorithme après l'itération t de la boucle TANT QUE.

Question 5.11 Montrer que $w(A_{t+1}, B_{t+1}) \geq (1 + \frac{2\epsilon}{n})w(A_t, B_t)$

Si l'algorithme ne se termine pas après l'itération t de la boucle TANT QUE, alors il existe un sommet v_1 non α-***content***.
Supposons que $v_1 \in B_t$ (sans perte de généralité) et qu'il est sélectionné pour l'itération $t + 1$. De ces hypothèses, nous pouvons déduire que $\alpha w(A_t, B_t) < w(A_t \cup \{v_1\}, B_t\backslash\{v_1\})$
Ce qui implique que

1. $\alpha w(A_t, B_t) < w(A_{t+1}, B_{t+1})$
2. $(1 + 2\frac{\epsilon}{n})w(A_t, B_t) < w(A_{t+1}, B_{t+1})$

Question 5.12 Donner une borne sur le nombre d'itérations k de l'algorithme pour que le poids de la coupe double ($w(A_{t+k}, B_{t+k}) \geq 2w(A_t, B_t)$). (***Indication :*** $(1 + 1/x)^x \geq 2$ ***pour*** $x \geq 1$)

Comme on a $w(A_{t+1}, B_{t+1}) \geq (1 + \frac{2\epsilon}{n})w(A_t, B_t)$, nous pouvons généraliser :

$$\begin{aligned} w(A_{t+k}, B_{t+k}) &\geq (1 + \tfrac{2\epsilon}{n})w(A_{t+k-1}, B_{t+k-1}) \\ &\geq (1 + \tfrac{2\epsilon}{n})^2 w(A_{t+k-2}, B_{t+k-2}) \\ &\geq (1 + \tfrac{2\epsilon}{n})^k w(A_t, B_t) \end{aligned}$$

Pour $x \geq 1$, nous avons $(1 + 1/x)^x \geq 2$. Ceci implique que $(1 + \frac{2\epsilon}{n})^{n/2\epsilon} \geq 2$.

Donc pour $k \geq \frac{n}{2\epsilon}$, nous avons $w(A_{t+k}, B_{t+k}) \geq 2w(A_t, B_t)$.

Question 5.13 Soit K le nombre de l'itérations nécessaires pour que l'algorithme termine. Montrer que $K \geq \frac{|V| log_2 \mathcal{W}(E)}{\epsilon}$. Quelle est la complexité de l'algorithme ?

Notons l'entier ℓ tel que $2^\ell \geq \mathcal{W}(E) \geq 2^{\ell-1}$ et tel que $\ell \geq \log_2 \mathcal{W}(E) \geq \ell - 1$.

Nous pouvons remarquer que $(1 + \frac{2\epsilon}{n})^{\ell \frac{n}{2\epsilon}} \geq 2^\ell$ d'après les questions précédentes.

Nous avons les conditions sur K tel que $K \geq \ell \frac{n}{\epsilon}$. Ceci implique que $(1 + \frac{2\epsilon}{n})^K \geq 2^\ell$, que $(1 + \frac{2\epsilon}{n})^K \geq \mathcal{W}(E)$, et que $(1 + \frac{2\epsilon}{n})^K w(A_0, B_0) \geq \mathcal{W}(E)$.

Donc, il y a au plus K itérations de la boucle.