Algorithmique avancée
Corrigé de l'examen du 29 janvier 2002, 8h00-11h00

Jean-Michel Dischler et Frédéric Vivien

Aucun document n'est autorisé

1 Enregistrement d'un CD sur cassette (21 points)

Nous avons à notre disposition une cassette de 60 minutes : sur chacune de ses faces nous pouvons donc enregistrer 30 minutes de musique. Nous souhaitons enregistrer sur cette cassette des morceaux provenant d'un CD contenant n morceaux pour une durée totale de 78 minutes : nous allons donc devoir choisir quels morceaux mettre sur la cassette et lesquels ne pas enregistrer, sachant que nous nous imposons qu'un morceau du CD soit enregistré au plus une fois sur la cassette.

Notations. Pour faciliter l'écriture des algorithmes, nous supposons que tous les morceaux durent un nombre entier de minutes et qu'ils sont tous de durée non nulle ! Nous notons n le nombre de morceaux et, afin d'être général, d la durée maximale enregistrable sur une *face* de la cassette (dans l'exemple ci-dessus, $d = 30$). Finalement, nous notons d_i la durée du i^{e} morceau (pour $1 \leq i \leq n$).

Le choix des morceaux à enregistré est déterminé par la politique d'enregistrement. Plusieurs politiques sont envisageables.

1.1 Un maximum de musique, quitte à couper des morceaux (2 points)

Dans cette partie nous cherchons à maximiser la durée totale de musique enregistrée sur la cassette, sachant que l'on s'autorise à couper un morceau (et ce autant de fois que nécessaire).

1. Proposez un algorithme glouton de résolution du problème. (1 point)

   Proposition de notation : si le morceau i n'est pas entièrement enregistré sur une face de la cassette mais seulement un fragment de longueur f, on pourra utiliser la notation : $i(\frac{f}{d_i})$.

   L'algorithme est présenté figure 1.

2. Quelle est sa complexité ? (1 point)

   Chacune des deux boucles « tant que » considère chaque morceau au plus une fois et il y a n morceaux. la complexité totale est donc en $O(n)$.

1.2 Un maximum de musique mais sans couper de morceaux (19 points)

Dans cette partie nous cherchons à maximiser la durée totale de musique enregistrée sur la cassette, sachant que l'on *interdit* de couper un morceau : un morceau figure en entier sur une face ou n'est pas enregistré du tout.

1. **Relation de récurrence.** On note $\mathit{durée}(i, t_1, t_2)$ la durée maximale de musique que l'on peut enregistrer sur la cassette, sachant que l'on peut mettre au maximum t_1 minutes de musique sur la première face et t_2 sur la deuxième, et sachant que l'on n'enregistre que des morceaux parmi les i premiers. $\mathit{durée}(n, d, d)$ nous donnera donc la solution de notre problème.

   Proposez une formule de récurrence définissant $\mathit{durée}(i, t_1, t_2)$. (3 points)

   Indication : dans une solution optimale, soit le i^{e} morceau est enregistré sur la première face, soit il est enregistré sur la deuxième, soit il n'est enregistré sur aucune des deux faces.

```
GLOUTON-COUPEUR(n, d)
  t_1 ← d
  t_2 ← d
  face_1 ← ∅
  face_2 ← ∅
  pour i ← 1 à n faire d'_i ← d_i
  i ← 1
  tant que t_1 > 0 et i ≤ n faire
    si d'_i ≤ t_1
      alors
        t_1 ← t_1 − d'_i
        face_1 ← face_1 ∪ {i}
        i ← i + 1
      sinon
        d'_i ← d'_i − t_1
        face_1 ← face_1 ∪ {i(t_1/d_i)}
        t_1 ← 0
  tant que t_2 > 0 et i ≤ n faire
    si d'_i ≤ t_2
      alors
        t_2 ← t_2 − d'_i
        face_2 ← face_2 ∪ {i}
        i ← i + 1
      sinon
        d'_i ← d'_i − t_2
        face_2 ← face_2 ∪ {i(t_2/d_i)}
        t_2 ← 0
```

FIG. 1 – Algorithme glouton de remplissage des faces d'une cassette.

Note sur la correction. Toutes les questions suivantes dépendent de la formule établie à cette question ci. Ces questions seront corrigées d'après **votre** réponse à cette question : par exemple, si l'algorithme que vous proposerez à la question 3a est bien la transcription de votre formule de récurrence sous la forme d'un algorithme de programmation dynamique, vous obtiendrez le maximum de points à cette question, même si votre formule de récurrence est fausse.

Considérons les trois cas :

(a) *Le i^e morceau est enregistré sur la première face. Alors, durée$(i, t_1, t_2) = d_i +$ durée$(i-1, t_1 - d_i, t_2)$ (avec ce morceau on enregistre une durée d_i de musique, et il nous reste toujours une durée de t_2 libre sur la deuxième face, alors que l'on a utilisé d_i minutes des t_1 que l'on avait sur la première face). Ceci n'est bien évidemment possible que si la durée restant sur la première face (t_1) est supérieure à la durée du morceau (d_i) : $t_1 \geq d_i$.*

Nous notons δ_1 la durée totale obtenue dans cette configuration :

$$\delta_1 = \begin{cases} 0 & \text{si } d_i > t_1 \\ d_i + \text{durée}(i-1, t_1 - d_i, t_2) & \text{sinon} \end{cases}$$

(b) *Le i^e morceau est enregistré sur la deuxième face. Alors, durée$(i, t_1, t_2) = d_i +$ durée$(i-1, t_1, t_2 - d_i)$. Ceci n'est bien évidemment possible que si la durée restant sur la deuxième face (t_2) est supérieure à la durée du morceau (d_i) : $t_2 \geq d_i$.*

Nous notons δ_2 la durée totale obtenue dans cette configuration :

$$\delta_2 = \begin{cases} 0 & \text{si } d_i > t_2 \\ d_i + \text{durée}(i-1, t_1, t_2 - d_i) & \text{sinon} \end{cases}$$

(c) *Si le i^e morceau n'est enregistré sur aucune des deux faces, la durée maximale que l'on peut obtenir en enregistrant certains des i premiers morceaux est égale à durée maximale que l'on peut obtenir en enregistrant certains des $i-1$ premiers morceaux, puisque l'on n'utilise pas le i^e :* $durée(i, t_1, t_2) = durée(i-1, t_1, t_2)$.
Nous notons δ_3 la durée totale obtenue dans cette configuration :

$$\delta_3 = durée(i, t_1, t_2).$$

Nous avons donc trois configurations possibles. On choisit bien évidemment celle qui maximise notre fonction de coût, c'est-à-dire celle qui maximise la durée totale de musique enregistrée. On obtient donc la formule :

$$durée(i, t_1, t_2) = \max\{\delta_1, \delta_2, \delta_3\}.$$

2. **Résolution récursive.** (3 points)

(a) Proposez un algorithme de résolution récursive du problème calculant simplement la récurrence établie à la question précédente. (1 point)

RÉSOLUTION-RÉCURSIVE(i, t_1, t_2)
  si $d_i > t_1$
    alors $\delta_1 = 0$
    sinon $\delta_1 = d_i +$ RÉSOLUTION-RÉCURSIVE$(i-1, t_1 - d_i, t_2)$
  si $d_i > t_2$
    alors $\delta_2 = 0$
    sinon $\delta_2 = d_i +$ RÉSOLUTION-RÉCURSIVE$(i-1, t_1, t_2 - d_i)$
  $\delta_3 =$ RÉSOLUTION-RÉCURSIVE$(i-1, t_1, t_2)$.
  renvoyer $\max\{\delta_1, \delta_2, \delta_3\}$

On résout le problème par l'appel : RÉSOLUTION-RÉCURSIVE*(n, 30, 30).*

(b) Montrez que cet algorithme est de complexité super-polynomiale dans le pire cas. (2 points)
Indication : vous pourrez supposer ici que tous les morceaux durent une minute.
Nous supposons ici que tous les morceaux durent une minute.
Nous notons $T(n, t_1, t_2)$ la complexité du problème avec n morceaux, une face de t_1 minutes et une de t_2. Si n, t_1 et t_2 sont tous les trois plus grand que 1, l'appel RÉSOLUTION-RÉCURSIVE*(i, t_1, t_2) engendre les trois appels récursifs :* RÉSOLUTION-RÉCURSIVE*($i-1$, t_1-1, t_2),* RÉSOLUTION-RÉCURSIVE*($i-1$, t_1, t_2-1) et* RÉSOLUTION-RÉCURSIVE*($i-1$, t_1, t_2). D'où la relation de récurrence sur la complexité :*

$$T(n, t_1, t_2) = T(n-1, t_1-1, t_2) + T(n-1, t_1, t_2-1) + T(n-1, t_1, t_2).$$

En remarquant que l'on a évidemment $T(n-1, t_1-1, t_2) \geq T(n-1, t_1-1, t_2-1)$, $T(n-1, t_1, t_2-1) \geq T(n-1, t_1-1, t_2-1)$, et $T(n-1, t_1, t_2) \geq T(n-1, t_1-1, t_2-1)$, on obtient :

$$T(n, t_1, t_2) \geq 3 \times T(n-1, t_1-1, t_2-1).$$

D'où l'on déduit :

$$T(n, t_1, t_2) = \Omega(3^{\min(n, t_1, t_2)}).$$

3. **Résolution par programmation dynamique « pure ».** (4 points)

(a) Proposez un algorithme basé sur le principe de la programmation dynamique et résolvant notre problème. Cet algorithme utilisera bien évidemment la relation de récurrence précédemment établie. (3 points)
On note d la durée totale de musique enregistrable sur une face. L'algorithme est présenté figure 2.
On peut remarquer que cet algorithme est bien défini : le calcul de durée(i, t_1, t_2) ne fait appel qu'à des cases du tableau de la forme durée($i-1$, t'_1, t'_2), cases calculées lors de l'itération précédente de la première boucle.

```
PROGRAMMATION-DYNAMIQUE(n)
  pour t1 ← 1 à d faire
    pour t2 ← 1 à d faire
      durée(0, t1, t2) ← 0
  pour i ← 1 à n faire
    pour t1 ← 1 à 30 faire
      pour t2 ← 1 à 30 faire
        si d_i > t1
          alors δ1 = 0
          sinon δ1 = d_i + durée(i − 1, t1 − d_i, t2)
        si d_i > t2
          alors δ2 = 0
          sinon δ2 = d_i + durée(i − 1, t1, t2 − d_i)
        δ3 = durée(i − 1, t1, t2).
        durée(i, t1, t2) ← max{δ1, δ2, δ3}
  renvoyer durée(n, d, d)
```

FIG. 2 – Calcul de la durée maximale enregistrable par programmation dynamique « pure ».

(b) Quelle est la complexité de cet algorithme ? (1 point)
Avec les notations définies à la question précédente, la complexité de cet algorithme est :

$$T(n, d) = \Theta(n \times d^2).$$

4. **Résolution par recensement.** (5 points)
   (a) Proposez un algorithme basé sur le principe de recensement (variante de la programmation dynamique) et résolvant notre problème. (3 points)
   L'algorithme est présenté figure 3.
   (b) Quelle est la complexité de cet algorithme ? (1 point)
   Avec les notations définies à la question précédente, la complexité de cet algorithme est :

$$T(n, d) = \Theta(n \times d^2).$$

   En effet, l'initialisation coûte $\Theta(n \times d^2)$. Comme l'algorithme RECENSEMENT-CALCUL *calcule au plus une fois chaque case du tableau « durée », la complexité de l'appel* RECENSEMENT-CALCUL*(n, d, d) est en $O(\Theta(n \times d^2))$.*
   (c) La solution par programmation dynamique « pure » et celle par recensement sont-elles de complexités comparables ? Un des deux algorithmes effectue-t-il plus de calculs que l'autre (sans tenir compte des initialisations) ? (1 point)
   Nous avons montré aux questions précédentes que ces deux solutions étaient de même complexité. Si l'on ne tient pas compte des initialisations, on remarque que la solution par programmation dynamique « pure » calcule ***toutes*** *les cases du tableau « durée », alors que la solution par recensement ne calcule que celles qui apparaissent dans la résolution de* RECENSEMENT-CALCUL*(n, d, d). Par exemple, si n, d et les durées de tous les morceaux sont des entiers pairs, la solution par programmation dynamique pure calculera les valeurs de durée(i, t_1, t_2) où t_1 ou t_2 est un entier impair, ce que ne calculera pas la solution par recensement. La solution par programmation dynamique pure effectue donc plus de calculs — si l'on ne tient pas compte des initialisations — même si la différence n'est pas significative.*

5. **Sélection des morceaux.** Jusqu'à présent, tout ce que l'on a fait, c'est de calculer la durée totale maximale de musique enregistrable sous les conditions de l'énoncé. Nous n'avons pas encore explicité comment sélectionner les morceaux qui permettent d'atteindre ce maximum. Modifiez un de vos algorithmes pour qu'il génère une telle liste des morceaux (une liste par face). (4 points)

```
RECENSEMENT(n, d)
  pour i ← 1 à n faire
    pour t1 ← 1 à d faire
      pour t2 ← 1 à d faire
        durée(i, t1, t2) ← −∞
  RECENSEMENT-CALCUL(n, d, d)
  renvoyer durée(n, d, d)

RECENSEMENT-CALCUL(i, t1, t2)
  si durée(i, t1, t2) > −∞
    alors renvoyer durée(i, t1, t2)
    sinon
      si d_i > t1
        alors δ1 = 0
        sinon δ1 = d_i + RECENSEMENT-CALCUL(i − 1, t1 − d_i, t2)
      si d_i > t2
        alors δ2 = 0
        sinon δ2 = d_i + RECENSEMENT-CALCUL(i − 1, t1, t2 − d_i)
      δ3 = RECENSEMENT-CALCUL(i − 1, t1, t2).
      durée(i, t1, t2) ← max{δ1, δ2, δ3}
      renvoyer durée(i, t1, t2)
```

FIG. 3 – Calcul de la durée maximale enregistrable par recensement.

– *Solution par programmation dynamique « pure ».*

```
PROGRAMMATION-DYNAMIQUE(n)
  pour t1 ← 1 à d faire
    pour t2 ← 1 à d faire
      durée(0, t1, t2) ← 0
      face1(0, t1, t2) ← ∅
      face2(0, t1, t2) ← ∅
  pour i ← 1 à n faire
    pour t1 ← 1 à 30 faire
      pour t2 ← 1 à 30 faire
        si d_i > t1
          alors δ1 = 0
          sinon δ1 = d_i + durée(i − 1, t1 − d_i, t2)
        si d_i > t2
          alors δ2 = 0
          sinon δ2 = d_i + durée(i − 1, t1, t2 − d_i)
        δ3 = durée(i − 1, t1, t2).
        si max{δ1, δ2, δ3} = 0
          alors
            durée(i, t1, t2) ← 0
            face1(i, t1, t2) ← ∅
            face2(i, t1, t2) ← ∅
          sinon
            si δ1 = max{δ1, δ2, δ3}
              alors
                durée(i, t1, t2) ← δ1
                face1(i, t1, t2) ← face1(i − 1, t1 − d_i, t2) ∪ {i}
                face2(i, t1, t2) ← face2(i − 1, t1 − d_i, t2)
```

sinon
si $\delta_2 = \max\{\delta_1, \delta_2, \delta_3\}$
alors
$durée(i,\ t_1,\ t_2) \leftarrow \delta_2$
$face_1(i, t_1, t_2) \leftarrow face_1(i-1, t_1, t_2 - d_i)$
$face_2(i, t_1, t_2) \leftarrow face_2(i-1, t_1, t_2 - d_i) \cup \{i\}$
sinon
$durée(i,\ t_1,\ t_2) \leftarrow \delta_3$
$face_1(i, t_1, t_2) \leftarrow face_1(i-1, t_1, t_2)$
$face_2(i, t_1, t_2) \leftarrow face_2(i-1, t_1, t_2)$
renvoyer $(durée(n,\ d,\ d), face_1(n,\ d,\ d), face_2(n,\ d,\ d))$

– *Solution par recensement.*

RECENSEMENT$(n,\ d)$
pour $i \leftarrow 1$ à n **faire**
pour $t_1 \leftarrow 1$ à d **faire**
pour $t_2 \leftarrow 1$ à d **faire**
$durée(i,\ t_1,\ t_2) \leftarrow -\infty$
$face_1(i, t_1, t_2) \leftarrow \emptyset$
$face_2(i, t_1, t_2) \leftarrow \emptyset$
RECENSEMENT-CALCUL$(n,\ d,\ d)$
renvoyer $(durée(n,\ d,\ d), face_1(n,\ d,\ d), face_2(n,\ d,\ d))$

RECENSEMENT-CALCUL$(i,\ t_1,\ t_2)$
si $durée(i,\ t_1,\ t_2) > -\infty$
alors renvoyer $durée(i,\ t_1,\ t_2)$
sinon
si $d_i > t_1$
alors $\delta_1 = 0$
sinon $\delta_1 = d_i +$ RECENSEMENT-CALCUL$(i-1, t_1 - d_i, t_2)$
si $d_i > t_2$
alors $\delta_2 = 0$
sinon $\delta_2 = d_i +$ RECENSEMENT-CALCUL$(i-1, t_1, t_2 - d_i)$
$\delta_3 =$ RECENSEMENT-CALCUL$(i-1, t_1, t_2)$.
si $\max\{\delta_1, \delta_2, \delta_3\} = 0$
alors
$durée(i,\ t_1,\ t_2) \leftarrow 0$
$face_1(i, t_1, t_2) \leftarrow \emptyset$
$face_2(i, t_1, t_2) \leftarrow \emptyset$
sinon
si $\delta_1 = \max\{\delta_1, \delta_2, \delta_3\}$
alors
$durée(i,\ t_1,\ t_2) \leftarrow \delta_1$
$face_1(i, t_1, t_2) \leftarrow face_1(i-1, t_1 - d_i, t_2) \cup \{i\}$
$face_2(i, t_1, t_2) \leftarrow face_2(i-1, t_1 - d_i, t_2)$
sinon
si $\delta_2 = \max\{\delta_1, \delta_2, \delta_3\}$
alors
$durée(i,\ t_1,\ t_2) \leftarrow \delta_2$
$face_1(i, t_1, t_2) \leftarrow face_1(i-1, t_1, t_2 - d_i)$
$face_2(i, t_1, t_2) \leftarrow face_2(i-1, t_1, t_2 - d_i) \cup \{i\}$
sinon
$durée(i,\ t_1,\ t_2) \leftarrow \delta_3$
$face_1(i, t_1, t_2) \leftarrow face_1(i-1, t_1, t_2)$

$face_2(i, t_1, t_2) \leftarrow face_2(i-1, t_1, t_2)$
renvoyer *durée*$(i,\ t_1,\ t_2)$

2 Élections présidentielles aux États-Unis (9 points)

Le décompte des voix après une élection présidentielle aux États-Unis est un problème excessivement sensible et qui doit être résolu le plus rapidement possible. Ce problème se trouve compliqué par le fait que n'importe qui peut recevoir des voix et être élu, les électeurs pouvant tout simplement écrire sur leur bulletin la personne pour laquelle ils votent [1]. Dans le pire des cas le nombre de personnes ayant reçu au moins une voix est égal au nombre de bulletins !

Notations. Nous notons n le nombre des votes. Ces votes sont stockés dans le tableau *Votes*. Pour faciliter l'écriture des algorithmes, nous supposerons que n est une puissance de deux (il existe donc un entier p tel que $n = 2^p$). Nous supposons également que la seule opération à notre disposition nous permet de vérifier si deux éléments sont ou non égaux.

Problème. Nous nous intéresserons ici uniquement au problème de savoir si quelqu'un a obtenu la majorité absolue des voix, c'est-à-dire si quelqu'un a obtenu strictement plus de $n/2$ voix.

1. **Algorithme naïf.** (3 points)
   (a) Écrivez un algorithme qui calcule le nombre de voix reçues par le citoyen x parmi les votes stockés entre les indices i et j du tableau *Votes*. (0,5 point)

   OCCURRENCES(x, *Votes*, i, j)
   compteur $\leftarrow 0$
   pour $k \leftarrow i$ **à** j **faire**
   si *Votes*$[k] = x$ **alors** *compteur* $\leftarrow$ *compteur* $+ 1$
   renvoyer *compteur*

   (b) Quelle est la complexité de cet algorithme ? (1 point)
   La boucle exécute $j - i + 1$ itérations. La complexité de cet algorithme est donc en $\Theta(j - i)$.

   (c) Au moyen de l'algorithme précédent, écrivez un algorithme MAJORITÉ-ABSOLUE qui vérifie si, parmi les citoyens qui ont reçu des voix, il en existe un qui a remporté la majorité absolue des votes enregistrés dans le tableau *Votes*. (0,5 point)

   MAJORITÉ-ABSOLUE(*Votes*)
   pour $i \leftarrow 1$ **à** *longueur*(*Votes*)/2 **faire**
   si OCCURRENCES(*Votes*$[i]$, *Votes*, i, *longueur*(*Votes*)) $>$ *longueur*(*Votes*)/2 **alors renvoyer** VRAI
   renvoyer FAUX

   (d) Quelle est la complexité de cet algorithme ? (1 point)
   Dans le pire cas, la boucle effectue $n/2$ itérations, chacune de ces itérations effectuant un appel à OCCURRENCES *sur un tableau de taille $n - i$ (i variant de 1 à n) donc de coût $\Theta(n - i)$. Le coût total de l'algorithme est donc en $\Theta(n^2)$.*

2. **Algorithme « diviser pour régner ».** (6 points)
   (a) Proposez un algorithme MAJORITÉ-ABSOLUE-DIV construit suivant le paradigme « diviser pour régner ». Cet algorithme divisera en deux le tableau *Votes* sur lequel il travaille. Cet algorithme renverra le couple (VRAI, x) s'il existe un citoyen (noté x) ayant reçu la majorité absolue des votes stockés dans le tableau *Votes* et renverra le couple (FAUX, 0) sinon. (4 points)

   MAJORITÉ-ABSOLUE-DIV(*Votes*, i, j)
   si $i = j$ **alors renvoyer** (VRAI, *Votes*$[i]$)
   $(r_x,\ x) \leftarrow$ MAJORITÉ-ABSOLUE-DIV(*Votes*, i, $\frac{i+j-1}{2}$)
   $(r_y,\ y) \leftarrow$ MAJORITÉ-ABSOLUE-DIV(*Votes*, $\frac{i+j+1}{2}$, j)

[1] Cette information, aussi loufoque qu'elle paraisse, est véridique.

si r_x = FAUX et r_y = FAUX **alors renvoyer** (FAUX, 0)
si r_x = VRAI et r_y = VRAI
  alors si $x = y$
    alors renvoyer (VRAI, x)
    sinon
      $c_x \leftarrow$ OCCURRENCES(x, *Votes*, i, j)
      $c_y \leftarrow$ OCCURRENCES(y, *Votes*, i, j)
      si $c_x > \frac{j-i+1}{2}$
        alors renvoyer (VRAI, x)
        sinon si $c_y > \frac{j-i+1}{2}$
          alors renvoyer (VRAI, y)
          sinon renvoyer (FAUX, 0)
  sinon si r_x = VRAI
    alors si OCCURRENCES(x, *Votes*, i, j) $> \frac{j-i+1}{2}$
      alors renvoyer (VRAI, x)
      sinon renvoyer (FAUX, 0)
    sinon si OCCURRENCES(y, *Votes*, i, j) $> \frac{j-i+1}{2}$
      alors renvoyer (VRAI, y)
      sinon renvoyer (FAUX, 0)

Justifications. *Les deux seuls cas qui ne sont peut-être pas immédiats sont les suivants :*

i. r_x = FAUX *et* r_y = FAUX *: dans ce cas il n'y a pas d'élément qui soit majoritaire dans la première moitié du tableau, ni d'élément qui soit majoritaire dans la deuxième moitié du tableau. Si le tableau contient n éléments, le nombre d'occurrences d'un élément quelconque dans la première moitié du tableau est donc inférieur ou égal à $\frac{\frac{n}{2}}{2}$ —la première moitié ayant $\frac{n}{2}$ éléments— et il en va de même pour le deuxième moitié. Donc le nombre d'occurrences d'un élément quelconque dans le tableau est inférieur à $\frac{n}{2}$ et le tableau ne contient pas d'élément majoritaire.*

ii. r_x = VRAI *et* r_y = VRAI *avec $x = y$: dans ce cas x est présent au moins $1 + \frac{n}{4}$ fois dans chacune des deux parties —qui sont de taille $\frac{n}{2}$— et donc au moins $2 + \frac{n}{2}$ fois dans le tableau.*

(b) Quelle est la complexité de cet algorithme ? (2 points)

La complexité de cet algorithme est définie par la relation de récurrence :

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

En effet, la phase de combinaison nécessite, dans le pire des cas, la recherche du nombre d'occurrences de deux éléments dans le tableau, ce qui a un coût de n, toutes les autres opérations étant de coût constant ($\Theta(1)$).

Nous avons donc ici : $a = 2$, $b = 2$ et $f(n) = \Theta(n) == \Theta(n^{\log_2 2})$. Nous sommes donc dans le cas 2 du théorème et donc :

$$T(n) = \Theta(n \log n).$$