Rapport de projet Algorithmique
Le compte est bon

MAKHLOUF Mohamed Mehdi
EL AFRIT Mohamed Amine
BUISSON Rémi
KAIDI Sanaa

LaTeX

Table des matières

Chapitre 1

Introduction

1.1 Problème

Le compte est bon est un jeu se déroulant de la façon suivante : un entier x et plusieurs entiers y_1,...,y_l sont choisis. Il s'agit alors d'obtenir x comme résultat d'une expression arithmétique utilisant les entiers y_1,...,y_l. On souhaite écrire un algorithme qui permet d'obtenir une telle expression, si c'est possible, ou bien une meilleure approximation.

1.2 Exemple

Voici un exemple d'experience
x = 26
y_1=1
y_2=4
y_3=5
y_4=3

Le programme donne alors : 3.(5+4)-1 = 26
En fait l'algorithme va effectuer les étapes suivantes :

au départ l'algorithme analyse : 1 , 4 , 5 , 3 (comme entrée)
après l'algorithme analyse : 9 , 1 , 3 (où 9=4+5)
l'algorithme analyse ensuite : 27 , 1 (où 27=9×3)
enfin l'algorithme donne : 26 (où 26=27-1)

Il est plus facile de schématiser les différents combinaisons possibles sous forme d'un arbre(FIG3.1).
Nous n'avons pas utilisé un type arbre mais nous avons utilisé la notion d'arbre car celà nous a aidé énormément dans notre raisonnement.
Dans ce présent travail nous proposons deux méthodes pour résoudre le problème :

- La première méthode consiste à parcourir les possibilités de combinaisons comme si on faisait un parcours en *profondeur* de l'arbre FIG3.1.
- Le seconde méthode consiste à parcourir les possibilités de combinaison comme si on faisait un parcours en *largeur* de l'arbre FIG3.1.

Chapitre 2

Conventions : définitions

2.1 Numérotation des couples formés à partir des éléments d'une séquence

Pour être plus claire on s'appuie sur un exemple de séquence et on énumère les différents couples. Prenons la séquence suivante :

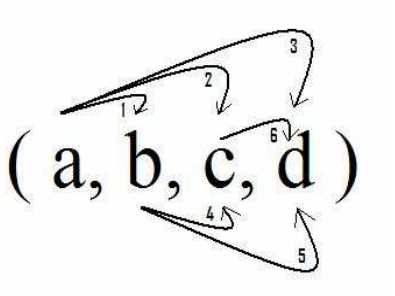

FIG. 2.1 – Méthode de numérotation des couples

On commence par le premier élèment (ici "a"). Les couples sont :

1^{er} couple : (a,b)
2^{me} couple : (a,c)
3^{me} couple : (a,d)
4^{me} couple : (b,c)
5^{me} couple : (b,d)
6^{me} couple : (c,d)

On peut aussi voir la numérotation de la manière suivante aussi.
Ici les numéros correspondent au couple formé par l'entier * et l'entier numéroté. Par exemple le 10^{me} couple est : (c,d), le 11^{me} couple est : (c,e), le 9^{me} couple est : (b,f) etc...

2.2 Convention opérateurs

On se propose de donner un code à chaque opérateur arithmétique pour s'en referer à chaque appel ; pour celà on suivra la convention suivante :

L'opérateur "+" est codé par le chiffre 1
L'opérateur "-" est codé par le chiffre 2
(en effectuant la soustraction on prendra toujours la valeur absolue car on s'interessera à des valeurs positives)

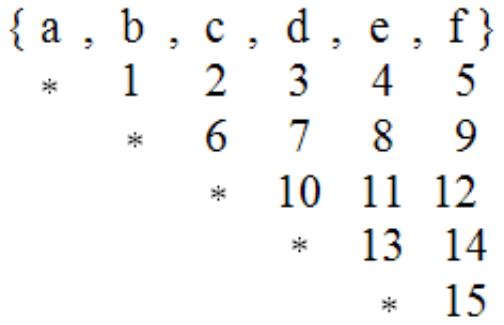

FIG. 2.2 – Méthode de numérotation des couples

L'opérateur "×" est codé par le chiffre 3
L'opérateur "÷" est codé par le chiffre 4

2.3 Expression arithmétique

Une expression arithmetique est une combinaison linéaire d'entiers (au sens mathématique).

2.4 Séquence

Une séquence est une succession ordonnée d'entiers (nous avons besoin de cet ordre pour la fonction "couple" qui construit le i^{me} couple d'une séquence (Cf.5)). On peut la coder comme structure de liste simplement chaînée.
On se permettra de noter le i^{me} élément de la séquence S : S[i].

Chapitre 3

Première approche : parcours en profondeur

Ici on ne va pas mentionner tous les algorithmes, on se contentera uniquement de l'algorithme spécifique à cette première approche à savoir : *La fonction recherche* (qui permet de faire le parcours des expressions).

Les autres algorithmes qui ont été utilisés en commun pour la première version et la seconde version seront mentionnés en 5.

3.1 Principe de résolution algorithmique

- On dipose d'une séquence de n entiers (a_1,a_2,...,a_n) (On peut dégager $C_2^n = \frac{n\times(n-1)}{2}$ couples de cette séquence que nous allons tous parcourir car nous les avons numéroté (Cf. 2.1)) On prend un couple *.
- Pour chaque couple on effectue une opération arithmétique si c'est possible.
- On teste si le résultat de l'opération est bon, sinon on continue.
- On ajoute le résultat de l'opération en tête de séquence.
- On efface les deux éléments du couple * de la séquence précédente.
- On obtient ainsi une nouvelle séquence contenant $n-1$ entiers (donc $C_2^{n-1} = \frac{(n-1)\times(n-2)}{2}$ couples possibles) on exécute les mêmes instructions précédentes à cette nouvelle séquence.

Ceci nous a inspiré l'idée de la récursivité dans cette première méthode.

3.2 Algorithme

3.2.1 Problème

- Problème : recherche
- Entrée : séquence d'entiers, tableau d'opérations
- Sortie : un bouléen

3.2.2 L'algorithme

Cf. Algorithme1

3.2.3 Illustration

Cf. FIG3.1

3.2.4 Correction

Il suffit de considérer les invariants suivants :

Algorithme 1 Algorithme Fonction recherche

ENTRÉES: s : séquence, x : entier, nombre_entiers : entier, proche : entier
SORTIE: booléen

```
  s2 : séquence ;
  i, j : entier ;
  valeur ;
  d :entier ;
  resultat : entier ;
  icouple : couple ;
  difference : entier ;
  atteint ← FAUX : boolén ;
  Tantque i ≤ C^2_{nombres_entiers} et non atteint Faire
    Tantque j ≤ 4 et non atteint Faire
      icouple ← couple(i, nombre_entiers, 0) ;
      resultat ← calcul(icouple, j) ;
      s2 ← remplacer(couple, resultat) ;
      difference ← resultat - x ;
      Si difference < 0 Alors
        atteint ← VRAI
        proche ← 0 ;
      Sinon
        Si difference = proche Alors
          proche ← difference
        Sinon
          atteint ← recherche(s2, nombre_entiers - 1, proche)
        FinSi
      FinSi
    FinTantque
  FinTantque
  retourner atteint
```

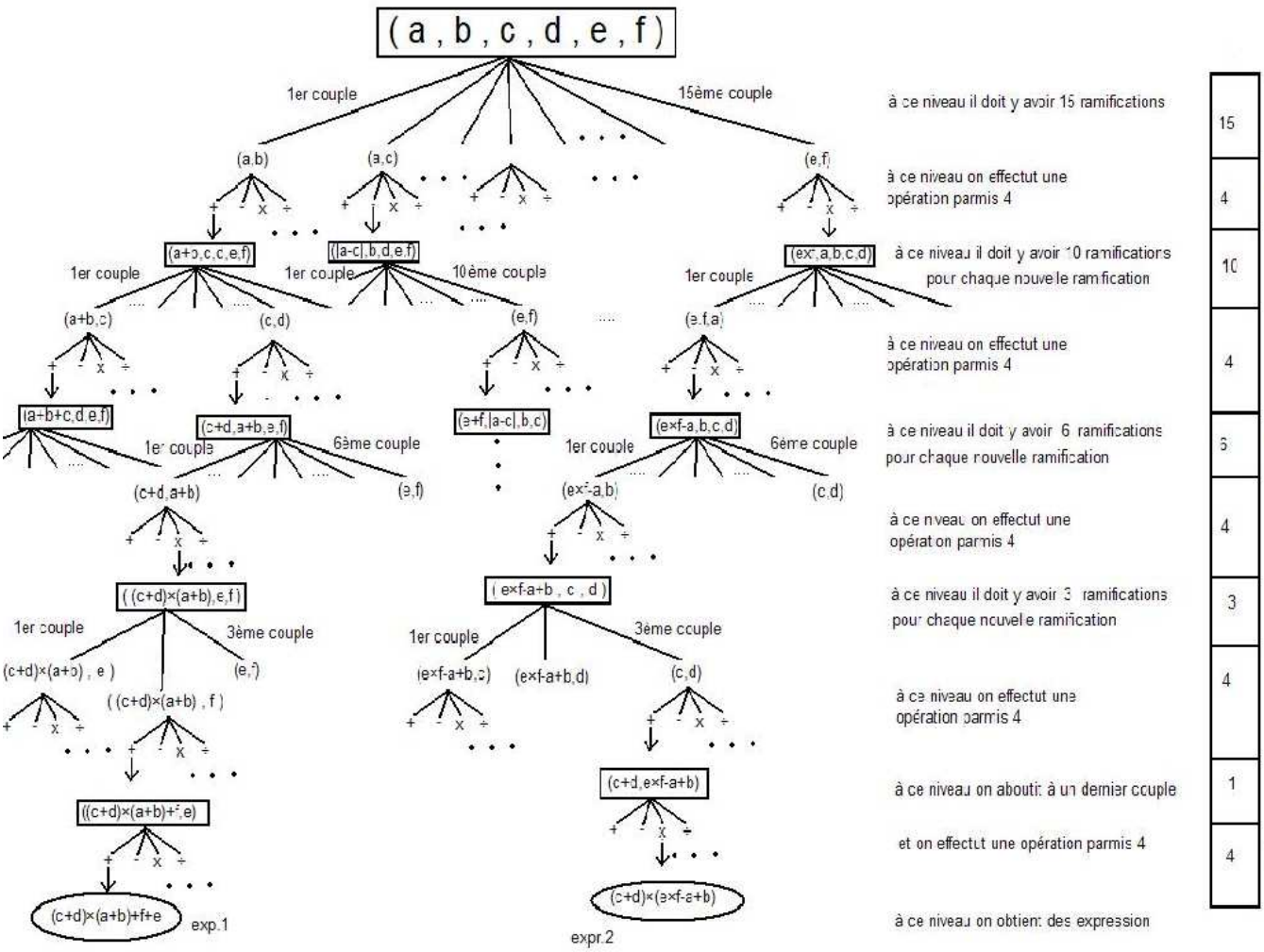

FIG. 3.1 – Les différents combinaisons

- première boucle : $i + 1 \leq C_n^2$,
- deuxième boucle : $j + 1 \leq 4$,

qui assurent :

- première boucle : un maximum de C_n^2 boucles,
- deuxième boucle : 4 boucles.

En observant que dans l'appel récursif, le nombre d'entiers passé en paramètre est $n-1$, ceci assure la terminaison des récursivités et ainsi des boucles.
De plus, en considérant le retour des appels récurisifs, on a bien le résultat attendu (vrai ou faux).

3.2.5 Complexité en temps

La compléxité en temps est donnée par la formule suivante :
$\prod_{i=2}^{n} 4.C_i^2 = 2^{n-1}.n!.(n-1)! \cong (n!)^2 \times 2^n$

(où chaque terme du produit$(4.C_i^2)$ représente le produit du nombre d'opérations arithmétiques (4) par le nombre de couples dans la séquence de longueur i (C_i^2))

3.2.6 Complexité en espace

On a les paramètres suivants : séquence ($nombre\_entiers \times (4+4) = 8n$)(pointeur + entier) et les autres entiers $(4+4+4)$. Ensuite on a la deuxième séquence de longueur $n-1$ donc elle a une complexite de $8 \times (n-1)$.
Dans la suite de l'algorithme on utilises des variables (7×4) et un couple $(4+4=8)$.
On a $n-1$ itérations donc au final, on obtient la complexité en espace de $(8 \times (2n-1)+48) \times (n-1)$.

Chapitre 4

Deuxième approche : parcours en largeur

Ici on ne va pas mentionner tous les algorithmes, on se contentera uniquement de ceux qui sont spécifique à cette deuxieme approche à savoir :

- *La fonction recherche* (qui permet de faire le parcours des expressions).
- *La fonction suivant* (qui permet d'incrémenter le compteur tableau Cf. 4.1).
- *La fonction valeur* (qui permet de calculer la valeur de l'expression codée par le compteur tableau Cf. 4.1).

Les autres algorithmes qui ont été utilisés en commun pour la première version et la seconde version seront mentionnés en 5.

4.1 Principe :"Le Compteur Tableau"

C'est un tableau d'entiers de longueur ($l = 2 \times n - 2$ où n est le nombre d'entiers du jeux). Ce compteur présente à chaque fois un code qui est une combinaison bien déterminée des chiffres du jeux (c'est à dire un code ou plutôt une référence à une certaine expression arithmétique).

Le but est de "décrypter" ce code à chaque incrémentation du compteur et de calculer le résultat de l'expression dont le code est "la valeur" du compteur.

4.1.1 Plage des valeurs du compteur tableau

La construction du compteur tableau est faite de sorte que les cases impaires réfèrent à l'opérateur, les cases paires réfèrent au couple.

Prenons par exemple deux cases successives d'un compteur T :

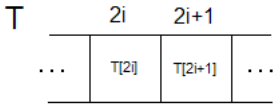

FIG. 4.1 – Plage des valeurs

T[2i] $\in [\![1, C^2_{i+2}]\!]$ (car il y a C^2_{i+2} couples dans la séquence "à analyser" par ces deux cases en effet on est à la case 2i+1 *donc* la séquence contient i+2 entiers Cf 4.1.2)

T[2i+1] $\in [\![1, 4]\!]$ (car il y a 4 opérations arithmétiques)

4.1.2 Longeur du compteur tableau

On appellera "longeur du compteur tableau" l'indice de la case la plus grande du compteur tableau.

La longeur du compteur tableau dépend naturellement du nombre d'entiers du jeux, en effet, plus on a d'entiers plus le nombre d'expression est élevé.

La longeur est ici appelée : MaxT et on a : $MaxT_n = 2 \times n - 3$ où n est le nombre d'entiers. Cette propriéte se démontre par récurrence sur le nombre d'entier "n", en voici la démonstration :

- Pour $n = 2$ (C'est le nombre minimal d'entiers pour jouer).
  Le compteur tableau est à 2 cases, en effet :

i	j

  Où $i = 1$ car il y a un seul couple qui peut être formé par 2 entiers et $j \in [\![1, 4]\!]$ pour parcourir les 4 opérations arithmétiques.
  Il est bien claire que dans ce tableau à deux cases l'indice le plus élevé est : $MaxT_2 = 1 = 2 \times 2 - 3$.
- Supposons que la propriété est vraie pour un certain rang n.
  Soit une séquence S à $n+1$ entiers.
  En choisissant un couple de S et une opération on forme une nouvelle séquence à n entier.
  Le fait de choisir un couple et une opération suggère d'ajouter deux cases au compteur tableau, c'est à dire en terme mathématique :

  $$MaxT_{n+1} = (MaxT_n) + 2 = (2 \times n - 3) + 2 = 2 \times (n + 1) - 3$$

- D'après le principe de raisonnement par récurrence la propriété est vraie pour tout entier n tel que : $2 \leq n$.

4.2 Exemple

Un exemple est très important pour la compréhension de notre algorithme

4.2.1 Illustration

Pour éviter toute confusion : on sous-entend par "k^{me} couple de la liste L_i" : le k^{me} couple construit à partir de la séquence S_i.

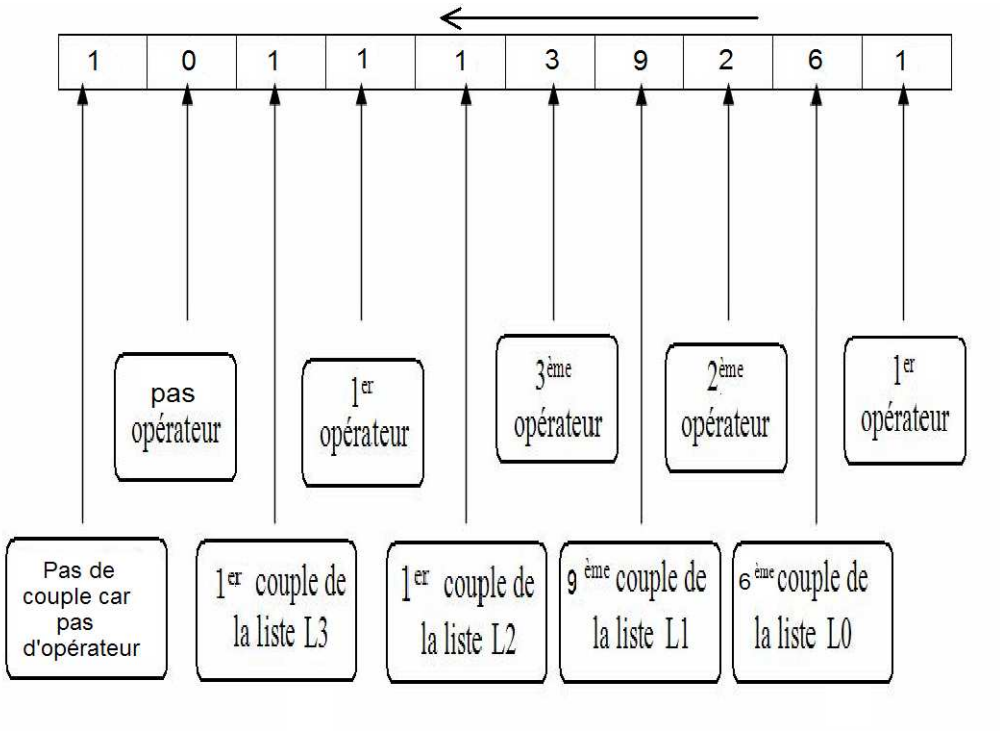

FIG. 4.2 – Un exemple

4.2.2 Analyse de l'exemple

Analyson le tableau "T" précédent de droite à gauche.
En entrée on a la séquence S_0 : (a,b,c,d,e,f).

- *T[8]=6* : on prend le 6^{me} couple de la séquence *(a , b , c , d , e , f)* qui est *(b , c)* (Cf. 2.1)
- *T[9]=1* : on effectue l'opération *"1"* qui est l'addition "+", soit */b+c/* (Cf. 2.2)
- On forme une nouvelle séquence S_1 à partir de la précédente en lui supprimant les éléments du 6^{me} couple S_0 et en lui ajoutant le résultat de la dernière opération. On obtient alors la séquence :

(/b+c/ , a , d , e , f)

- *T[6]=9* : on prend le 9^{me} couple de la dernière séquence formée qui est *(d , f)* (Cf. 2.1)
- *T[7]=2* : on effectue l'opération *"2"* qui est la soustraction "-", soit */d-f/* (Cf. 2.2)
- On forme une nouvelle séquence S_2 à partir de la précédente en lui supprimant les éléments du 9^{me} couple S_1 et en lui ajoutant le résultat de la dernière opération. On obtient alors la séquence :

(/d-f/ , b+c , a , e)

- *T[4]=1* : on prend le 1^{er} couple de la dernière séquence formée qui est *(/d-f/ , b+c)* (Cf. 2.1)
- *T[5]=3* : on effectue l'opération *"3"* qui est la multiplication "×", soit */d-f/×b+c* (Cf. 2.2)
- On forme une nouvelle séquence S_3 à partir de la précédente en lui supprimant les éléments du 1^{er} couple de S_2 et en lui ajoutant le résultat de la dernière opération. On obtient alors la séquence :

(/d-f/×(b+c) , a , e)

- *T[2]=1* : on prend le 1^{er} couple de la dernière séquence formée qui est *(/d-f/×(b+c) , a)* (Cf.

2.1)
- *T[3]=1* : on effectue l'opération *"1"* qui est l'addition *"+"*, soit */d-f/×(b+c)+a* (Cf. 2.2)
- On forme une nouvelle séquence S_4 à partir de la précédente en lui supprimant les éléments du 1^{er} couple de S_3 et en lui ajoutant le résultat de la dernière opération. On obtient alors la séquence :

$$(\ /d\text{-}f/\times(b+c)+a \ , \ e \ )$$

- *T[1]=0* : on arrête notre analyse du tableau car on a rencontré un "0" (On ne parle pas d'opération 0)
- On obtient alors le résultat final de l'expression arithmétique :

$$(\ /d\text{-}f/\times(b+c)+a \ )$$

Ainsi le compteur tableau fait référence à l'expression arithmétique dont le résultat est calculé à la dernière étape.

4.2.3 Plage des valeurs du compteur tableau dans cet exemple

Le compteur tableau analysé précédement a la forme suivante :

q	r	o	p	m	n	k	l	i	j

(j , l , n , p , r) $\in [\![1,4]\!]^5$ (intervalle d'entiers ! !)

$i \in [\![1,15]\!]$ $(15 = C_6^2)$:en effet on peut former C_6^2 couples à partir de la première séquence à 6 éléments

$k \in [\![1,10]\!]$ $(10 = C_5^2)$:en effet on peut former C_5^2 couples à partir d'une séquence à 5 éléments

$i \in [\![1,6]\!]$ $(6 = C_4^2)$:en effet on peut former C_4^2 couples à partir d'une séquence à 4 éléments

$i \in [\![1,3]\!]$ $(3 = C_3^2)$:en effet on peut former C_3^2 couples à partir d'une séquence à 3 éléments

q = 1 : en effet une séquence à 2 élémént est déja un couple $1 = C_2^2$

4.3 Algorithmes

L'algorithme principal est l'algorithme de *la fonction recherche*, en effet, c'est à l'issue de cette fonction qu'on trouve le résultat exacte ou le résultat le plus proche.

4.3.1 La fonction recherche

Problème

- Problème : recherche
- Entrée : séquence d'entiers, compteur tableau
- Sortie : un bouléen

L'algorithme

Algorithme 2 Algorithme Fonction recherche

Entrées: S : séquence ; i,nb_element,a_trouver :entier ; T_proche :tableau d'entiers .
Sortie: Booleen

```
MaxT ← 2×nb_element-3
atteint ← 0
T ← initialiser(T)
T_proche ← initialiser(T_proche)
résultat ← valeur(T,S,nb_element)
différence ← |a_trouver - résultat|
Si différence = 0 Alors
  atteint ← 1 ;
FinSi
Tantque T[MaxT] ≠ 0 et différence ≠ 0 Faire
  suivant(T,nb_element) ;
  résultat ← valeur(T,S,nb_element) ;
  Si différence > |a_trouver - résultat| Alors
    différence ← |a_trouver - résultat| ;
    T_proche ← T
  FinSi
FinTantque
Si différence = 0 Alors
  atteint ← 1 ;
FinSi
return(atteint) ;
```

Correction

C'est une conséquence immédiate des définitions des problèmes de *suivant et résultat.*

Complexité en temps

La complexité en temps de cette fonction est pratiquement la même que la fonction suivant, à savoir :
$\sum_{i=0}^{n-1} \prod_{j=0}^{i} 4.C_{n-j}^{2} = (n!)^2 \times \sum_{i=0}^{n-1} \frac{2^i}{(n-i)!^2}$

Complexité en espace

La complexité en espace est : $8 \times (n + MaxT) + 24$

4.3.2 La fonction suivant

Cette fonction permet d'inrémenter le compteur tableau.

Problème

- Problème : suivant
- Entrée : compteur tableau
- Sortie : compteur tableau

L'algorithme

Algorithme 3 Algorithme Fonction suivant

Entrées: T :tableau d'entiers, nb_element.
Sortie: T :tableau d'entiers.

```
i ← 2×nb_element-3 ;
Si i = 1 Alors
  Si T[i]<4 Alors
    T[i] ← T[i] + 1 ;
  Sinon
    Pour j de 0 à MaxT Faire
      T[j] ← 0 ;
    FinPour
  FinSi
Sinon
  Si T[i]<4 Alors
    T[i] ← T[i] + 1 ;
  Sinon
    T[i] ← 1 ;
    Si T[i-1] < C^2_{(i-1)/2+2} Alors
      T[i-1] ← T[i-1] + 1 ;
    Sinon
      T[i-1] ← 1 ;
      suivant(T,nb_element - 1) ;
    FinSi
  FinSi
FinSi
```

Illustration

On prend par exemple un compteur tableau pour 4 entiers du jeu. Ce compteur va contenir 6 cases (d'après 4.1.2).
L'incrémentation se fait de la manière suivante :

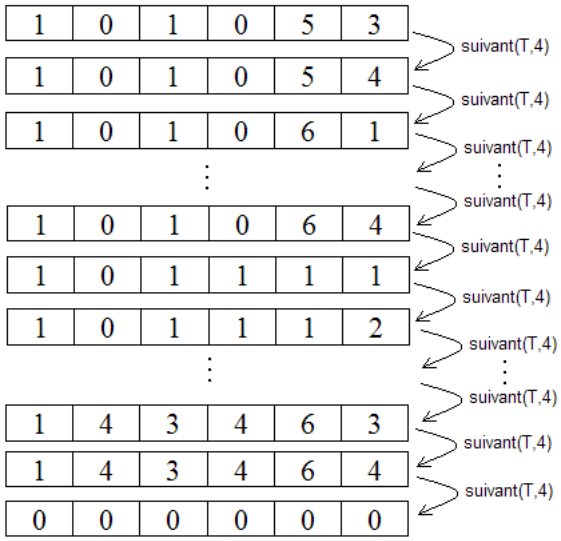

FIG. 4.3 – Incrémentation

Correction

La correction est évidente car la fonction est une suite de tests conditionnels imbriqués. Et elle incrémente bien le compteur tableau.

Complexité en temps

La complexité en temps de cette fonction est C_n donnée par la relation suivante :
$C_{n+1} = 4 \times C_{n+1}^2 \times (1 + C_n)$
qui est aussi donnée par :

$$\sum_{i=0}^{n-1} \prod_{j=0}^{i} 4.C_{n-j}^2 = (n!)^2 \times \sum_{i=0}^{n-1} \frac{2^i}{(n-i)!^2}$$

Complexité en espace

La complexité en espace est : $4 \times n + 8$ (tableau et deux entiers).

4.3.3 La fonction valeur

Cette fonction permet de calculer la valeur de l'expression codé par le compteur tableau.

Problème

- Problème : valeur
- Entrée : compteur tableau
- Sortie : entier

Algorithme 4 Algorithme Fonction valeur

Entrées: T :tableau d'entiers ; S : séquence ; nb_element :entier.
Sortie: resultat :entier.

```
ConstructTab icouple(2)(0)(0) ;
ConstructTab T1(nb_element)(0)(0) ;
i ← 2×nb_element-3 ;
resultat ← 0 ;
Tantque T[i] ≠ 0 Faire
    couple(nb_elements, icouple, T[i-1],0,S) ;
    résultat ← calcul(icouple[0],icouple[1],T[i]) ;
    S ← remplacer(S,T1,icouple[0],icouple[1],T[i])
    i ← i-2 ;
    nb_element ← nb_element-1 ;
FinTantque
return(resultat) ;
```

L'algorithme

Correction

En ce qui concerne la la terminaison on remarque que i decroit strictement jusqu'à être strictement inférieur à 1 ce qui prouve la terminaison de la boucle .
Le resultat est bien celui renvoye par la fonction calcul et on decode bien aussi le tableau en entrée.

Complexité en temps

La complexité en temps pour cette fonction est : $(\lfloor (2n-3)/2 \rfloor + 1) \times \Theta(couple) \times \Theta(remplacer)$

Complexité en espace

On a les paramètres suivants : séquence ($nombre\_entiers \times (4+4) = 8n$)(pointeur + entier) et les autres entiers $(4+4)$. Dans la suite de l'algorithme on utilises un tableau de 2 entiers (4×2) et 2 tableaux de n éléments $(4 \times 2 nombre\_entiers)$.
Au final on a : $20n + 8 + \Theta(couple) + \Theta(remplacer) + \Theta(resultat)$.

Chapitre 5

Les algorithmes en commun pour les deux versions

5.1 La fonction couple

Cette fonction permet de construire le k^{me} couple d'une séquence S en se basant sur la définition 2.1.

5.1.1 Problème

- Problème : couple
- Entrée : tableau vide de deux entiers
- Sortie : tableau remplie de deux entiers

5.1.2 L'algorithme

Algorithme 5 Algorithme Fonction couple

Entrées: s : séquence, nombre_entiers : entier, i :entier, position : entier
Sortie: couple

```
s2 : séquence ;
s2 ← s ;
Si i < 1 ou i > nbCombinaison(nombres_elements) Alors
  retourner new_couple(-1, -1) ;
Sinon
  Tantque s2 ≠ NILL et position ≠ i Faire
    position ← position +1 ;
    s2 ← s2^suivant ;
  FinTantque
  Si s2 ne NILL et i = position Alors
    retourner new_couple(s^entier, s2^suivant) ;
  Sinon
    retourner couple(s^suivant, nombres_entiers, i, position -1) ;
  FinSi
FinSi
```

5.1.3 Correction

`WHILE-PROGRAM:`

```
%precond: s!=NILL/\1<=i/\i<=nbcomb(n)/\position=1
s2:=s;
while (s2!=NILL/\position!=i) do
%invariant: (position+1)<i
position:=position+1;
s2:=suivant(s2)
endwhile
%postcond: position=i/\iemecouple(s)


PROOF:

{s!=NILL/\1<=i/\i<=nbcomb(n)/\position=1}
[0]
{(position+1)<i}
s2:=s
{(position+1)<i}
;
{(position+1)<i}
while (s2!=NILL/\position!=i) do
%invariant: (position+1)<i
[1]
{((position+1)+1)<i}
position:=position+1
{(position+1)<i}
;
{(position+1)<i}
s2:=suivant(s2)
{(position+1)<i}

endwhile
[2]
{position=i/\iemecouple(s)}


PROOF OBLIGATIONS:

s!=NILL/\1<=i/\i<=nbcomb(n)/\position=1
[0] ------------------- implies --------------------
(position+1)<i

(position+1)<i/\(s2!=NILL/\position!=i)
[1] ------------------- implies --------------------
((position+1)+1)<i

(position+1)<i/\!((s2!=NILL/\position!=i))
[2] ------------------- implies --------------------
position=i/\iemecouple(s)
```

5.1.4 Complexité en temps

Au pire des cas, on parcours n fois la première boucle puis on passe une séquence de taille $n-1$ lors de l'appel récurisif ... on a donc $n!$ itérations.

5.1.5 Complexité en espace

On a la complexité de la séquence $(8n)$ ainsi que celle des paramètres (4×3).
Il faut ajouter à ceci la complexité de la séquence de taille $n-1$ qui est $8(n-1)$.
Ceci appliqué au $n-1$ itérations on obtient : $(8 \times (2n-1)) \times (n-1)$.

5.2 La fonction calcul

5.2.1 Problème

- Problème : calcul
- Entrée : tableau rempli de deux entiers, entier entre 1 et 4
- Sortie : entier

5.2.2 L'algorithme

Algorithme 6 Algorithme Fonction Calcul

ENTRÉES: T : tableau de 2 entiers ; j : entier
SORTIE: val :entier

```
Selon j
    Cas '1' :
        val ← (T[1] + T[2] ) ;
        Sortir ;
    Cas '2'
        val ← ( Si T[1] > T[2] Alors T[1] - T[2] Sinon T[2] - T[1] ) ;
        Sortir
    Cas '3'
        val ← ( T[1]×T[2] ) ;
        Sortir
    Cas '4'
        val ← (Si T[2] divise T[1] Alors T[1]/T[2] Sinon( Si T[1] divise T[2] Alors T[2]/T[1] )) ;
        Sortir
        retourner val ;
```

5.2.3 Correction

Evidente (Suite de tests conditionnels)

5.2.4 Complexité en temps

La complexite en temps de cette algorithme est constante .

5.2.5 Complexité en espace

On a les paramètres suivants : un tableau de 2 entiers $(4+4)$ et deux entiers $(4+4=8)$

Chapitre 6

Conclusion

Le projet qui nous a été donné nous a permis de découvrir le travail en équipe et les difficultés qui en découlent.
En effet une des premières difficultés est la communication : faire comprendre ses idées aux autres et les convaincre que c'est la meilleure.
Cette difficulté n'est pas observable lorsqu'on mène un projet tout seul, car il est difficile d'être en désaccord avec soi-même.
Un second obstacle est l'échange des projets.
En effet, même si le rapport intermédiaire est bien fait, se plonger dans le code du projet et se mettre dans son ambiance, est un exercice qui n'est pas évident.
Un autre obstacle est le respect du facteur temps.
En effet, respecter les délais de préparation ou achever une mission pendant une durée plus ou moins courte est une tâche à ne pas négliger par une entreprise pour ne pas perdre ses clients.
Ainsi, pour conclure, nous pouvons dire que ce qui a été avant tout mis en avant dans le projet est la démarche de l'ingénieur dans une entreprise : convaincre, communiquer et motiver le personnel pour avancer.