Compte Rendu 3I003 Confitures

Rémi Dadvisard et Clément Castellon

December 5, 2018

1 Partie Théorique

1.1 Algorithme 1: Recherche Exhaustive

Question 1 Montrer par récurrence forte sur S la validité et la terminaison de cet algorithme lorsqu'il est appelé par l'appel initial RechercheExhaustive(k, V, S).

Terminaison:

RechercheExhaustive(k, V, S) est une fonction récursive, et ne fait des appels que sur S, ou S-V[i].

- *Sur S, si S<V[i], alors on cherche a comparer avec V[i-1](<V[i]), et on termine quand on atteint V[0].*
- *Sur S-V[i], si S-V[i] est négatif, alors l'algo termine.*

L'algorithe se termine si s≤0, récursivement, la valeur de S est réduite de V[i] ≥0, à chaque appel, on a une boucle strictement décroissante, donc il existe un appel ou s sera négatif ou nul. ET donc si un appel récursif termine, tous les appels récursifs termineront.

Validité:

P(i) *= « NbCont égal au nombre minimum pour remplir S=i » P(0) =0 donc valide P(1) = 1 Valide (V[1] = 1 étant obligatoire)*

x *prends la plus petite valeur de RechercheExhaustive(k, V, S-V[k]) à Recherche-Exhaustive(k, V, S-V[1]). Qui sont valides par HR. Or, pour chacune de ses valeurs; il a besoin d'un seul bocal en plus, donc le nombre minimum sera égal a min(RechercheExhaustive(k, V, S-x) avec x appartenant à V) + 1. NBCont(i) renvoie ce nombre, donc:*

RechercheExhaustive(k, *V, S) est valide.*

Conclusion: *donc l'algorithme est valide et se termine*

Question 2 Supposons que les seuls bocaux disponibles sont des bocaux de capacités 1dg et 2dg. On note a(s) le nombre d'appels récursifs effectués par RechercheExhaustive(2, [1, 2], s).

a)Exprimer a(s) sous forme d'une suite récursive.

$$a(s)=\begin{cases}0 & si\ s=0\\ 2 & si\ s=1\\ a(s-1)+a(s-2)+2 & si\ s\geq 2\end{cases}.$$

Soit b(s) la suite définie par:

$$b\,(s)=\begin{cases}0 & si\ s=0\\ 2 & si\ s=1\\ 2b(s-2)+2 & si\ s\geq 2\end{cases}.$$

Soit c(s) la suite définie par:

$$c\,(s)=\begin{cases}0 & si\ s=0\\ 2c(s-1)+2 & si\ s\geq 1\end{cases}.$$

b)Montrer que pour tout entier $s\geq 0$, on a $b(s)\leq a(s)\leq c(s)$.

La suite est strictement croissante et positive, on a donc $a(s-1)+a(s-2)+2\geq a(s-1)$ *et* $a(s-1)\geq a(s-2)$ *on a donc* $a(s-1)+a(s-2)\leq 2a(s-1)$ *et* $2a(s-2)\leq a(s-1)+a(s-2)$ *on a ainsi* $2a(s-2)+2\leq a(s-1)+a(s-2)+2\leq 2(s-1)$ *or les suites a, b et c ont les mêmes premiers termes, on peut donc remplacer les termes de l'inégalité. on a ainsi* $b(s)\leq a(s)\leq c(s)$*pour tout entier* $s\geq 0$

c)Quel est le terme général de la suite c(s)? Justifiez.

Initialisation:

$c_0=0$

$c_1=2$

$c_2=6$

$c_3=14$

On décèle que $c_n=2^{n+1}-2$

Supposons $P_n=\ c_n=2^{n+1}-2$ *pour un n donné, montrons* P_{n+1}*:* $c_{n+1}=2^{n+2}-2$

Induction:

$c_{n+1} = 2c(n+1-1)+2$

$c_{n+1} = 2*(2^{n+1}-2)+2$ *Par H.R*

$c_{n+1} = 2^{n+2}-4+2$

$c_{n+1} = 2^{n+2}-2$

Hérédité confirmée. On en déduit donc que $c_n = 2^{n+1}-2$ *pour tout* $n \geq 0$.

d)En déduire que $b(s) = c(\lceil \frac{s}{2} \rceil)$.

Posons P:" $b(k) = c(\lceil \frac{k}{2} \rceil)$ *avec* $k = \{0, ..., n\}$*" pour un n donné, montrons* P_{n+1}

Induction:

$b(n+1) = 2b(n-1)+2$

$b(n+1) = 2c(\lceil \frac{n-1}{2} \rceil)+2$ *Par H.R*

$b(n+1) = 2(2^{\lceil \frac{n-1}{2} \rceil+1}-2)+2$

$b(n+1) = 2^{\lceil \frac{n-1}{2} \rceil+2}-2$ *or* $\lceil \frac{n-1}{2} \rceil + 2 = \lceil \frac{n+1}{2} \rceil + 1$*par définition de la partie entière*

$b(s+1) = 2^{\lceil \frac{n+1}{2} \rceil+1}-2$

$b(n+1) = c(\lceil \frac{n+1}{2} \rceil)$

Hérédité confirmée par le théorème de la recurrence forte.

e)Donner le terme général de la suite b(s). En déduire un encadrement du nombre d'appels récursifs réalisés par RechercheExhaustive(2, [1, 2], s). Que peut-on en conclure quant à la complexité temporelle de l'algorithme RechercheExhaustive?

On déduit de la question précédente que $b(s) = 2^{\lceil \frac{n}{2} \rceil+1}-2$.

On peut ainsi encadre la complexité $2^{\lceil \frac{n}{2} \rceil+1}-2 \leq a(s) \leq 2^{s+1}-2$

On peut ainsi en déduire que RechercheExhaustive est minorée par $2^{n/2}$*et surtout majorée* 2^n *par et donc de complexité* $\mathcal{O}(2^n)$

1.2 Algorithme II : Programmation dynamique

Pour ce deuxième algorithme, nous allons exploiter la structure du problème pour éviter une exploration exhaustive de toutes les solutions.

On note m(S) le nombre minimum de bocaux pour S décigrammes de confiture et un tableau de capacités V. On définit une famille de problèmes intermédiaires de la façon suivante: étant donnés un entier s et un entier $\forall i \in \{1, \ldots, k\}$, on note m(s, i) le nombre minimum de bocaux nécessaires pour une quantité totale s en ne choisissant des bocaux que dans le système de capacités V [1], V [2] , . . . , V[i].

Pour initialiser la récursion, on pose:

- $m(0,i) = 0 \qquad \forall i \in \{1,\ldots,k\}$: il est possible de réaliser la capacité totale 0 avec 0 bocal.
- $m(s,0) = +\infty \qquad \forall s \geq 1$: il n'est pas possible de réaliser la capacité s ≥ 1 sans utiliser au moins un bocal.
- $m(s,i) = +\infty \qquad \forall i \in \{1,\ldots,k\}\,,\ \forall s > 0$: il n'est pas possible de réaliser une capacité négative.

Question 3 a)Quelle est la valeur de m(S) en fonction des valeurs m(s,i) définies dans la section précédente?

$m(S) = m(S,k)$

b)Montrer la relation de récurrence suivante pour tout $\boldsymbol{i \in \{1,\ldots,k\}}$

$$m(s,i) = \begin{cases} 0 & si\ s = 0 \\ \min\{m(s,i-1), m(s-V[i],i)+1\} & sinon \end{cases}$$

Initialisation:

$m(0,i) = 0$ *voir énoncé.*

Supposons la relation de l'énoncé vraie pour les valeurs de m(s',i') avec $s' \epsilon \{0,\ldots,s-1\}$ *et* $i' \epsilon \{0,\ldots,i\}$ *ainsi que la valeur* $m(s,i-1)$.

alors, soit m(s, i) utilise un bocal de la plus grande contenance et est égal à m(s-V[i],i)+1, soit il on ne l'utilise pas et m(s,i) = (s,i-1).

or on veut minimiser la valeur de m(s,i) donc on a min(m(s,i-1),m(s-v[i],i)+1) si $s \neq 0$

Question 4 Pour résoudre le problème de manière plus efficace que l'algorithme RechercheExhaustive, on va utiliser l'approche dite de programmation dynamique .Le calcul des valeurs m(s, i), grâce à la relation de récurrence décrite dans la question précédente, nous permettra de trouver m(S).

L'idée maîtresse est de ne pas calculer une même valeur m(s, i) plusieurs fois: pour cela on stocke ces valeurs dans un tableau M doublement indicé. Dans chaque case du tableau M[s, i] (avec s $\in\{$ 0 , . . . , S $\}$ et i $\in\{$ 0 , . . . , k $\}$), on stocke la valeur m(s,i), c'est-à-dire soit une valeur ∞ , soit le nombre de bocaux utilisés.

a)Décrire (et justifier) dans quel ordre les cases du tableau M peuvent être remplies en utilisant la formule de récurrence.

Algorithm 1 AlgoProgDyn(S,V,k):

Creer un tableau m(S,I)
Pour tout s appartenant à {0,...,S}
$m(s,0) = +\infty$
fin pour
Pour tout i appartenant à {0,...,k}
$m(0,i) = 0$
fin pour
pour tout s appartenant à {1,...S}:
pour tout i appartenant à {1,...,k}:
m(s,i)=min{m(s,i-1), m(s-V[i],i)+1}
fin pour
fin pour
retourner m(s,i)

On remplira le tableau à l'aide d'une boucle sur s d'abord, puis sur i à l'interieur de celle-ci: On remplit pour chaque s les m(s,i) avec i croissant avant de passer à la valeur de s suivante

b)En déduire un algorithme AlgoProgDyn (en pseudo-code) qui détermine le nombre de bocaux nécessaires pour une quantité de confiture S.

c)Analyser la Complexité temporelle et spatiale de cet algorithme. On entend par Complexité spatiale l'espace mémoire requis en fonction de k et S.

- Complexité temporelle:
  - L'algorithme effectue un parcours de chaque case du tableau en fonction des valeurs trouvées pour des s inférieurs, et termine lorsqu'il est à la case V[S][k]. Il effectue donc un traitement pour chaque case, et il y a $S \times k$cases dans la matrice utilisée, donc cet algorithme a une Complexité de $\Theta(S \times k)$.
- Complexité spatiale:
  - L'algorithme va nécessiter un stockage d'un tableau de $S \times k$cases, donc il a une complexité spatiale de $\Theta(S \times k)$.

Question 5

On désire à présent permettre à l'algorithme de retourner un tableau A indiquant le nombre de bocaux pris pour chaque type de capacités.

a) Une première idée est de placer dans chaque case du tableau M[s,i], un tableau A indiquant les bocaux pris pour la capacité totale s et des capacités

Algorithm 2 AlgoProgDynTab(S,V,k):

Soit m[S,I] un tableau de tuple(entier, tableau[k])
Pour tout s appartenant à {0,...,S}
$m(s,0)\,[1] = +\infty$
fin pour
Pour tout i appartenant à {0,...,k}
$m(0,i)\,[1] = 0$
fin pour
pour tout s appartenant à {1,...S}:
pour tout i appartenant à {1,...,k}:
m(s,i)[1] =min{m(s,i-1)[1], m(s-V[i],i)[1]+1}
si m(s,i)[1]≠m(s,i-1)[1]:
m(s,i)[2][i]+=1
fin si
fin pour
fin pour
retourner m(s,i)

Algorithm 3 AlgoProgDyn_retour(S,V,k,m):

(avec m le tableau de l'algorythme dynamique)
Soit A[k] un tableau
tant que s>0:
si m(s,k)=m(s-V[k],k)+1
A[k]= A[k]+1
S=S-V[k]
fin si
sinon k = k-1
fin tant que
retourner A

de bocaux prises parmi V [1], . . . , V [i]. Indiquer les modifications à effectuer dans l'algorithme précédent. Indiquer la complexité spatiale de cet algorithme

b) On souhaite éviter de copier dans chaque case du tableau M, les tableaux A des bocaux utilisés. Pour ce faire, on conserve l'algorithme initial et on va reconstituer a posteriori le tableau A de la solution optimale : ce second algorithme est souvent appelé "algorithme- retour" (ou "backward"). L'idée est basée sur la remarque suivante : en remarquant que M[s,i] est égal soit à M[s−V[i],i]+1; soit à M[s,i−1], on sera en mesure de savoir si un bocal de capacité V [i] a été choisi ou non, et quelle case du tableau il faut ensuite examiner pour connaître les autres bocaux choisis. Décrire ce algorithme-retour (en pseudo-code) et indiquer la complexité temporelle et spatiale globale de cet algorithme

Algorithm 4 AlgoGlouton(S,V,k):

```
res = 0
tant queS > 0:
si S < v[k] :
k= k- 1
fin si
si S ≥ v[k]:
res += S // v[k]
k= k-1
S = S - (S // v[k]) * v[k]
fin si
fin tant que
retourner res
```

la compléxité temporelle dans le pire cas est en $\Theta(s+k)$ iterations pour l'algortithme retour. On a donc une complexité totale est en $\Theta(s*k+s+k)$

La complexité spatiale est en $\Theta(s*k+k)$

Question 6

Peut-on dire que cet algorithme est polynomial ?

La complexité, $\Theta(s*k+k)$ est un polynôme de degré 1. l'algorithme est donc polynomial

1.3 Algorithme III : Cas particulier et algorithme glouton

Quetion 7 *Sa complexité temporelle est dans le pire cas (uniquement des bocaux de 1) $\mathcal{O}(s*k)$*

Question 8

Montrer qu'il existe des systèmes de capacités qui ne sont pas glouton-compatibles. Justifiez votre réponse. *Supposons S=160g et V={1,80,100}*

l'algorithme glouton rendra 61 (1 bocal de 100 et 60 de 1)

alors qu'il existe la solution 2 (deux bocaux de 80)

Question 9 (Bonus)

a) Montrer qu'il existe un plus grand indice j pris dans {1,...,k} tel que oj < gj. *la solution o est optimale et différente de g. le nombre de bocaux total utilisé par o est donc inferieur ou égal a celui de g. il existe donc forcément*

un indice j tel que $o_j < g_j$(si tous les indices de o sont supperieurs, o utilise plus de bocaux, et si ils sont égaux o=g)

b) Montrer que $\sum V[i]g_i = \sum V[i]o_i$ pour i allant de 1 à j Si j=k, on a $\sum V[i]gi = \sum V[i]oi$ = S.

c) En déduire que $\sum$V [i]oi $\geq$ V [j] *on peut en déduire que la solution optimale o ne passe pas par l'algorithme glouton. car il reste assez de confiture dans les pots V[1] à V[J-1] dans la solution o pour remplir au moins un pot de taille V[j] (ce que l'algorithme glouton ne laisse pas passer)*

Question 10 Montrer que tout système de capacité V avec k = 2 est glouton-compatible (on rappelle qu'on a toujours V [1] = 1).

Demonstration par l'absurde:

Si S$\leq$V[2] alors, on a pour seule solution de n'utiliser que des bocaux de taille 1, donc l'algorithme glouton sera une solution optimale.

*Si S$\geq$V[2], soit y la valeur rendue par l'algorithme glouton, avec $y = a + b$ et a*V[2]+b=S, la division euclidienne de S par V[2].*

Supposons x une solution optimale qui n'est pas issue de l'algorithme glouton, avec un V de capacité 2 et une quantité S, et x<y.

*On a par définition x=a'+b', a' la quantité de bocaux V[2] et b' la quantité b'=S-a'*V[2] (nombre de bocaux de taille 1 utilisés) supposons a'$\neq$a et b'$\neq$b. (par définition du système de taille 2 si a'=a, b'=b).*

*supposons a'>a, par définition de la division euclidienne, a'*V[2]>S donc impossible.*

*On a donc a'<a. On a alors b'=b+(a-a')*V[2] puisqu'on doit remplir le reste de la confiture avec des bocaux de 1. donc on a x=a'+b+v[2]a-V[2]a'*

x=a+b +(V[2]-1)a-(V[2]-1)a'

x=a+b +(V[2]-1)a-a'

x= y + (V[2]-1)a-a'

or, a'<a donc, et V[2]>1 donc (V[2]-1)a-a'>0 et y<x

Contradiction.

On a donc montré l'absurde qu'un système de capacité de taille 2 est toujours glouton-compatible.

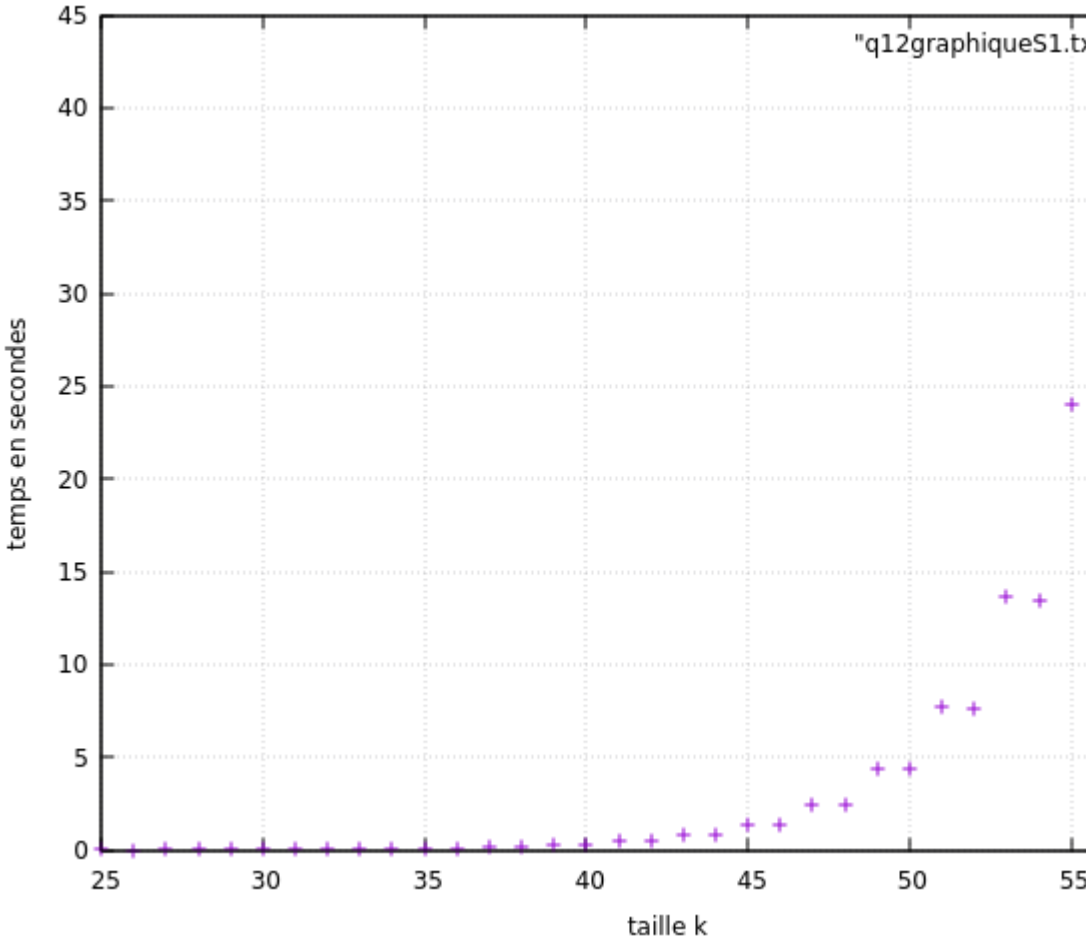
Question 11 Prouver que cet algorithme de test est polynomial en donnant sa complexité.

On peut facilement estimer la première boucle par $\Theta(V[k])$ (elle fait exactement V[k]+V[k-1]-V[3]+1 tours)

La deuxième boucle est effectuée k fois, et fait appel deux fois a AlgoGlouton (de complexité $\mathcal{O}(s*k)$) et fait une comparaison. on a donc k tours en $\mathcal{O}(s*k+1)$

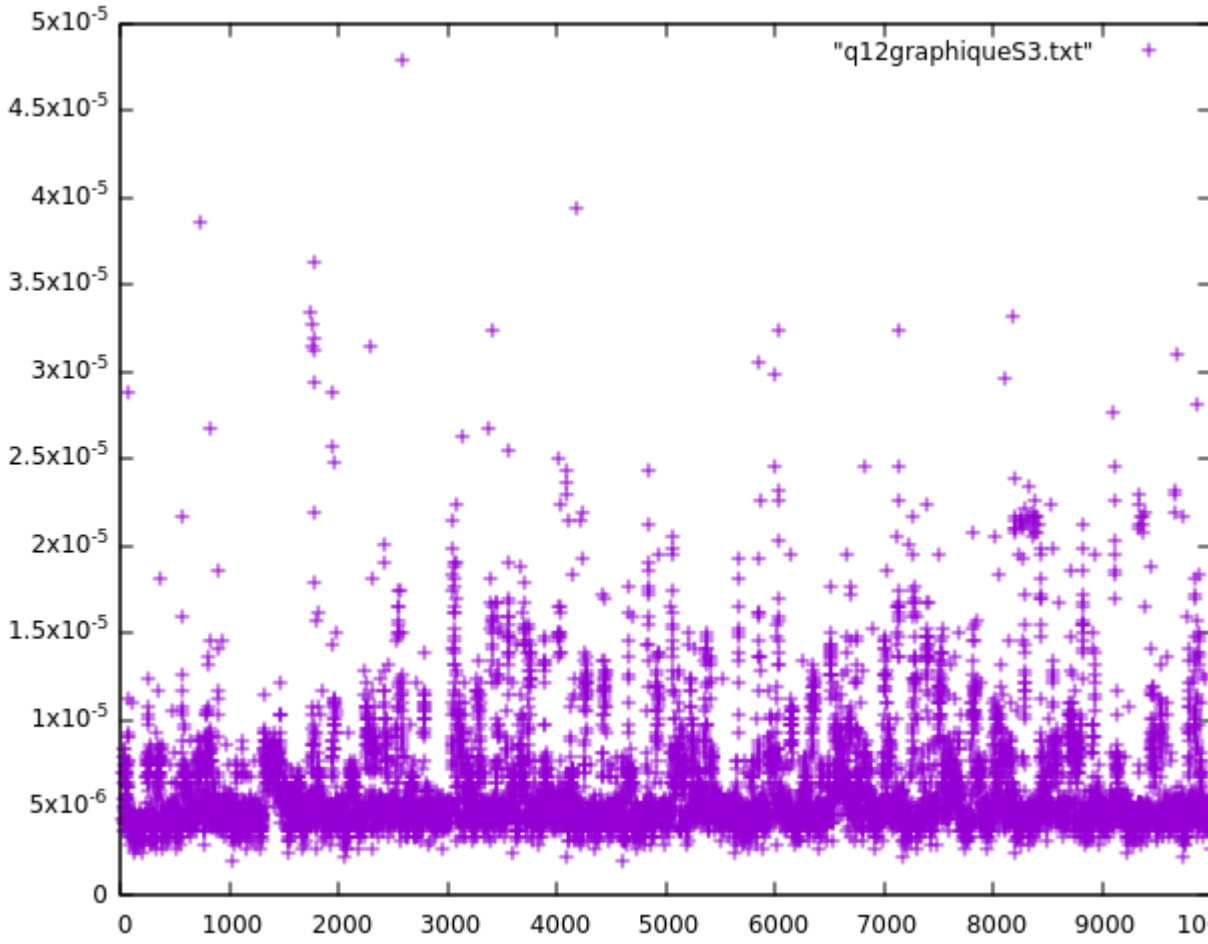
on se retrouve donc avec une boucle en $\mathcal{O}(s*k^2+k)$ et donc une complexité en $\Theta(V[k]*(s*k^2+1))$ qui est donc bien polynomiale

2 Mise en œuvre



Question 12

Temps d'éxécution de l'algorithme exhaustif en fonction de la valeur k (courbe analogue pour S)



Temps d'éxécution pour l'algorithme Glouton, en fonction de la valeur S(courbe analogue pour k), temps linéaire et éxécution très rapide.