Projet d'algorithmique
Confitures

LU3IN003

Licence d'informatique L3

Sorbonne Université

Année universitaire 2024-2025

1 Introduction

Sur un forum de cuisine, on pourrait lire la question suivante :

> "Je viens de cuisiner une grande quantité de confiture et j'ai des bocaux de tailles diverses prêts pour la verser dedans. Mais je me pose la question suivante : combien de pots, tous bien remplis, me faut-il au minimum pour y préserver ma confiture ?"

Le but de ce projet est de répondre à cette question. Lors de la mise en bocal, la confiture est encore liquide et il faut rapidement la verser dans des bocaux comme ceux de la photographie ci-dessous. Il est important de remplir chaque bocal complètement, c'est-à-dire exactement à sa capacité. Dans le cas contraire, l'air contenu dans le bocal limite le temps de conservation de la confiture. Mais il existe des bocaux de capacités bien différentes ! Votre but sera de concevoir des algorithmes qui s'adaptent aux différentes capacités : étant donné les capacités des pots, la quantité de confiture, l'objectif est d'utiliser le moins de bocaux possible.

- *Définition du problème*

On dispose de S décigrammes (notation dg) de confiture que l'on doit verser dans des bocaux vides.

On dispose de bocaux de plusieurs capacités que l'on classe en k types de bocaux différents : chaque type de bocal correspondant à une capacité distincte. On appellera *système de capacités* l'ensemble des capacités dont on dipose. On note ces k capacités par un tableau V de taille k numéroté par ordre croissant, *i.e.*

$$V[1] < V[2] < \cdots < V[k]$$

Une capacité $V[i]$ est exprimée par la quantité en décigrammes que l'on peut mettre dans un bocal. Par exemple, on peut avoir $k = 8$, avec $V = [1, 2, 5, 10, 20, 50, 100, 200]$.

On fait les hypothèses suivantes :

- La quantité S de confiture est un nombre entier de décigrammes, par exemple $S = 200$ décigrammes (2 kg).
- Le plus petit des bocaux est de capacité 1 décigramme, *i.e.* $V[1] = 1$.
- On dispose d'une très grande quantité (supposée ainsi illimitée) de bocaux de chacune des capacités.

Remarquez que pour respecter la première hypothèse, il suffit d'un peu de gourmandise. Ces trois hypothèses permettent de dire qu'il y a bien toujours une solution à notre problème : en effet, il y a toujours une solution en remplissant uniquement des petits pots de 1 décigramme (10 grammes).

L'objectif est de remplir le moins possible de bocaux **sachant que chaque bocal doit être rempli exactement à sa capacité maximale**. Ainsi, étant donnés :

- un système de k capacités $V[i] \in \mathbb{N}$, $i \in \{1, \ldots, k\}$ avec $V[1] = 1$,
- et une quantité totale $S \in \mathbb{N}$ de confiture,

le but est de déterminer le nombre minimum de bocaux tels que la somme de leurs capacités est exactement égale à S. On cherche à retourner un couple (n, A) où n est le nombre de bocaux utilisés et A est un tableau de taille k tel que $A[i]$ représente le nombre de bocaux de capacité $V[i]$ à remplir "à fond". Notez qu'ainsi $n = \sum_{i=1}^{k} A[i]$.

Pour le tableau de capacité $V = [1, 2, 5, 10, 20, 50, 100, 200]$ et une quantité $S = 748$ décigrammes de confiture par exemple, il faudra remplir au minimum 9 bocaux : 3 de 200dg, 1 de 100dg, 2 de 20dg, 1 de 5dg, 1 de 2dg et 1 de 1dg. Le résultat rendu par un algorithme résolvant ce problème sera donc le couple (9, | 1 | 1 | 1 | 0 | 2 | 0 | 1 | 3 |).

• *Objectifs du projet*

Ce projet vise à l'analyse et la mise en œuvre de plusieurs algorithmes résolvant le problème décrit ci-dessus. Le travail se divise en une **partie théorique** et une **partie expérimentale**. La partie théorique formalise le problème et permet d'établir et d'analyser plusieurs algorithmes ainsi que leurs complexités respectives. La partie expérimentale porte sur la mise en œuvre de ces algorithmes et la vérification expérimentale de leurs complexités respectives.

2 Partie théorique

Dans cette partie, nous étudions trois algorithmes pour résoudre le problème de choix de bocaux.

2.1 Algorithme I

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 $i \in \{1, \ldots, k\}$, 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 $V[1], V[2], \ldots, V[i]$.

Pour initialiser la récursion, on pose :

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

Question 1
a) Quelle est la valeur de $m(S)$ en fonction des valeurs $m(s,i)$ définies dans la section précédente ?

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

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

Question 2
Le calcul des valeurs $m(s,i)$, grâce à la relation de récurrence décrite dans la question précédente, permet de trouver $m(S)$. Ecrire (en pseudo-code) un algorithme récursif basé sur cette relation de récurrence et calculant $m(S)$, sans oublier d'indiquer quel est l'appel initial de l'algorithme. Cet algorithme n'utilisera *pas* de tableau auxiliaire pour sauvegarder d'éventuelles valeurs déjà calculées.

Question 3
Représenter l'arbre des appels récursifs de votre algorithme pour $S = 5$ et $k = 2$ avec $V[1] = 1$ et $V[2] = 2$.

Question 4
Dans l'arbre représenté à la question précédente, combien de fois la valeur $m(1,1)$ est-elle calculée ? Plus généralement, si $k = 2$ avec $V[1] = 1$ et $V[2] = 2$, combien de fois la valeur $m(1,1)$ est-elle calculée en fonction de S (disons que S est impair) ?

2.2 Algorithme II

Nous allons maintenant améliorer l'algorithme précédent en ne calculant pas une même valeur $m(s,i)$ plusieurs fois.

Question 5
Pour cela, nous allons stocker ces valeurs dans un tableau M doublement indicé : dans la case $M[s,i]$ (avec $s \in \{0,\ldots,S\}$ et $i \in \{0,\ldots,k\}$), on stocke la valeur $m(s,i)$, qui sera soit la valeur ∞, soit le nombre de bocaux utilisés.

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

b) En déduire un algorithme itératif **AlgoOptimisé** (en pseudo-code) qui détermine le nombre minimum 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.

Question 6
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 de bocaux prises parmi $V[1], \ldots, V[i]$. 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.
Donner le pseudo-code de cet algorithme retour, et indiquer les complexités temporelles et spatiales obtenues en enchaînant l'algorithme initial et l'algorithme retour.

Question 7
Peut-on dire que l'algorithme II ainsi obtenu est de complexité polynomiale ?

2.3 Algorithme III : Cas particulier et algorithme glouton

On se propose d'écrire un algorithme glouton, c'est-à-dire un algorithme qui prend à chaque étape une décision (un bocal à remplir) qui ne sera pas remise en cause dans les étapes suivantes de l'algorithme. Le principe de l'algorithme est le suivant : parmi les bocaux de capacités au plus S, on choisit autant de fois que possible le bocal de plus grande capacité. On poursuit avec cette même stratégie pour la quantité restante, jusqu'à ce que cette capacité restante soit nulle.

Question 8
Écrire l'algorithme **AlgoGlouton** en pseudo-code. Quelle est la complexité temporelle de cet algorithme ?

Un système de capacité est dit **glouton-compatible** si, pour ce système de capacité, l'algorithme glouton produit la solution optimale quelle que soit la quantité totale S.

Question 9
Montrer qu'il existe des systèmes de capacités qui ne sont pas glouton-compatibles. Justifiez votre réponse.

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

En l'état actuel des connaissances, on ne sait pas caractériser mathématiquement ce qu'est un système de capacités glouton-compatible. En revanche, on sait tester si un

système de capacités est ou non glouton-compatible par l'algorithme **TestGloutonCompatible** ci-dessous. La validité de cet algorithme est assez difficile à démontrer, et cette preuve n'est donc pas demandée.

---

algorithme TestGloutonCompatible(k : entier, V : **t**ableau de k entiers) : booléen

S, j, : entiers
si $k \geq 3$ **alors**
  pour S de $(V[3]+2)$ à $(V[k-1]+V[k]-1)$ **faire**
    pour j de 1 à k **faire**
      si $(V[j] < S)$ et (AlgoGlouton(S)>1+AlgoGlouton($S - V[j]$)) **alors**
        Retourner Faux
      fin si
    fin pour
  fin pour
fin si
Retourner Vrai

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

On peut remarquer qu'enchaîner l'algorithme de test et, en cas de réponse positive, l'algorithme glouton est de complexité polynomiale. On a ainsi une méthode très efficace pour répondre à notre problème... mais seulement pour certaines instances. En cas de réponse négative à l'algorithme de test, on doit utiliser l'algorithme de programmation dynamique.
Dans la partie Mise en œuvre, on va étudier expérimentalement la probabilité qu'un système de capacité soit glouton-compatible ou non.

3 Mise en œuvre

Un première partie propose une campagne de tests numériques permettant de vérifier expérimentalement la complexité des algorithmes I, II et III (section 3.2). Une deuxième partie étudie si l'algorithme glouton peut être utilisé ou non en pratique (section 3.3).

3.1 Implémentation

Dans un premier temps, le travail d'implémentation attendu est le suivant :

Entrées-Sorties
Donner à vos programmes une fonctionnalité permettant de lire en entrée **un fichier texte** contenant les données du problèmes, c'est-à-dire la quantité totale S, le nombre k de types de bocaux et le tableaux **trié** V des capacités. La première ligne du fichier sera

S, la seconde k, et les suivantes les k capacités des pots. Par exemple, si $S = 151$ dl, et s'il y a 3 types de pots de cotenances 20 dl, 5 dl, et 1 dl, le fichier sera le suivant :

```
151
3
1
5
20
```

Trois programmes
Créer trois programmes correspondant aux trois algorithmes I, II et III.

Tests de fonctionnement
Sur de petits exemples, testez si vos programmes fournissent bien la solution attendue.

3.2 Analyse de complexité expérimentale

Pour l'analyse de complexité expérimentale, on va s'intéresser au système de capacités suivant, nommé **système Expo**, qui est construit à partir d'une valeur entière $d \geq 2$:

$$V[1] = 1 \quad V[2] = d \quad V[3] = d^2 \quad \ldots \quad V[k] = d^{k-1}$$

Pour tout entier $d \geq 2$, ce système de capacités Expo est glouton-compatible (admis).

On considérera ce système pour des valeurs $d = 2$, $d = 3$ et $d = 4$. Pour chaque système de capacités ainsi considéré, vous veillerez à faire varier les valeurs des paramètres des instances (*i.e.* la quantité S et la valeur k) afin d'observer la complexité de vos algorithmes.

Pour mesurer le temps d'exécution d'un algorithme, il faudra pour chaque système de capacités, si les temps obtenus sont significatifs, tracer des courbes de temps d'exécution en fonction de S et k. Il s'agira de générer des fichiers de points qui seront utilisés pour tracer des courbes à l'aide d'un tableur ou d'un outil graphique. A titre d'exemple, vous trouverez sur moodle une documentation sur les outils `xgraphic` ou `gnuplot`.

Question 12
Ces tests sont à mener pour les algorithmes II et III. Pour l'algorithme I, vous vous arrêterez dès que le temps de calcul est supérieur à 1 minute. Pour cela, pour chaque programme, prévoir une mesure du temps CPU de son exécution (attention cette valeur peut être différente du temps d'exécution).
Analysez de façon critique les comportements expérimentaux obtenus en fonction des complexités théoriques, et comparez les performances des algorithmes.

3.3 Utilisation de l'algorithme glouton

On s'intéresse à déterminer si l'algorithme glouton peut être utilisé dans un cadre pratique. En effet, il est rapide mais il ne donne pas toujours la meilleure solution.
On veut ainsi tester expérimentalement plusieurs aspects : la fréquence d'apparition de systèmes de capacités glouton-compatibles ; et aussi évaluer, pour un système qui n'est pas glouton-compatible, l'écart relatif (le rapport) entre la valeur de la solution optimale et la valeur de la solution fournie par l'algorithme glouton III.

Génération de systèmes de capacités
Créer aléatoirement des systèmes de capacités en faisant varier le nombre k de types de bocaux et en utilisant une génération aléatoire pour ces capacités (attention, le système de capacités doit contenir la capacité 1 et être trié par ordre croissant). Les capacités des bocaux seront choisies uniformément entre 2 et une valeur maximum p_{max} que vous choisirez.

Question 13
Utiliser l'algorithme TestGloutonCompatible de la section 2.3 pour déterminer si ce système de capacités est ou non glouton-compatible. Ceci vous permettra de donner, pour un ensemble important de tirages de système de capacités, la proportion de systèmes glouton-compatibles parmi l'ensemble des systèmes possibles.

Question 14
Pour des tailles p_{max} et f raisonnables, à chaque fois que vous détectez un système qui n'est pas glouton-compatible, utiliser les algorithmes II et III sur toute les valeurs S entre p_{max} et $f * p_{max}$ où f est suffisamment grand : vous indiquerez alors le pire écart obtenu et l'écart moyen moyen obtenu entre les deux solutions. Vous donnerez ensuite des statistiques sur l'ensemble des systèmes qui sont détectés ou non glouton-compatibles.

Cahier des charges

- Le projet est à faire **en binôme du même groupe**.
- Pour la mise en œuvre algorithmique, vous êtes libres de choisir le langage de programmation que vous utiliserez. Seuls les aspects purement algorithmiques seront abordés avec vos enseignants.
- Chaque binôme doit rédiger sur traitement de texte un rapport. Ce rapport doit contenir l'analyse théorique (dans laquelle vous prendrez soin de justifier toutes vos réponses en suivant l'ordre des questions), ainsi que l'analyse expérimentale des algorithmes (réponses aux questions 12 à 14). Vous veillerez à inclure dans le rapport les courbes que vous aurez obtenues, et à les commenter. Ce rapport doit être constitué d'**un seul fichier, au format pdf**.
- Vous devez créer un répertoire ayant pour nom `nomBinome1_nomBinome2`, en remplaçant `nomBinome1` et `nomBinome2` par vos noms de famille. Ce répertoire contiendra votre rapport au format `pdf` ainsi que les fichiers sources de vos programmes (et ne contient pas les exécutables !). Compressez ce répertoire sous forme d'un fichier `zip` ou `tgz` (par `tar cvzf nomRepertoire.tgz nomRepertoire`). Ce fichier sera soumis sur moodle au plus tard le **mercredi 20 novembre 2024 à 23h59**, dans le dépôt correspondant à votre groupe de TD. Les soutenances auront lieu la semaine suivante en salle machine à la place des séances de TD (semaine du 25 novembre).