Devoir1

mickael.ledenmat

December 2020

Contents

1 Introduction **3**

2 Partie théorique **4**

2.1 Que peut-on toujours faire si on a deux objets de poids p1 et p2 tel que P = p1 + p2. 4

2.2 Que peut-on dire si un algorithme remplit deux colis à moins de la moitié de leur capacité ? 4

2.3 Proposer au moins deux algorithmes gloutons qui résolvent le problème du remplissage des lutins (mais pas nécessairement en donnant la solution optimale). 5

2.4 Proposer également un algorithme qui trouve toujours la solution optimale (mais qui peut être lent). 6

2.5 Faîtes tourner vos algorithmes sur l'exemple : (2, 6, 1, 5, 8, 4, 5, 7, 5, 3) et P = 9 et donner la réponse obtenue. 8

2.6 Donner pour chacun de vos algorithmes gloutons une entrée pour laquelle il trouve une solution optimale. 8

2.7 Le facteur d'approximation d'un algorithme est le pire rapport, sur toutes les entrées possibles, entre le nombre de colis trouvé par l'algorithme divisé par le nombre de colis optimal. Pour un algorithme qui trouve la solution optimale, ce facteur sera toujours un. Donner une borne sur le facteur d'approximation d'un de vos algorithmes glouton. 8

3 Partie pratique **9**

3.1 Implémentez les algorithmes de la partie précédente. 9

3.2 Implémentez un algorithme qui génère une entrée aléatoire avec n objets dont le poids est tiré aléatoirement entre 1 et m. 9

3.3 Pour chacun de vos algorithmes, calculer le nombre moyen de colis nécessaires, pour 1000 entrées avec 100 objets de poids entre 1 et 10 et un poids maximal par colis de 10. Vous calculerez également la moyenne sur 1000 entrées avec 100 objets de poids entre 1 et 1000 et un poids maximal par colis de 1000. 9

1 Introduction

Vous travaillez dans une entreprise de logistique pour les fêtes et on vous donne n objets de poids p1, . . . , pn à expédier. Vous avez à votre disposition des colis, qui peuvent chacun contenir un poids au plus P. Votre but est de ranger les objets dans les colis, en utilisant le moins de colis possible et en respectant la contrainte que la somme des poids d'un objet d'un colis est inférieure ou égale à P. On apelle ce problème le problème du remplissage des lutins. Une solution du problème du remplissage des lutins est donc une liste de donc chaque élément est un colis constitué de la liste des objets qui le remplit. Par exemple, si vos colis font un poids de 10 kilos maximum et que les objets que vous devez ranger font (1, 4, 1, 6, 7, 9, 3, 2, 8) alors (1, 1, 2, 6),(3, 7),(4),(8),(9) est une solution utilisant 5 colis. Une solution est dite optimale si elle utilise un nombre minimal de colis parmis toutes les solutions.

2 Partie théorique

2.1 Que peut-on toujours faire si on a deux objets de poids p1 et p2 tel que P = p1 + p2.

Si l'on a deux poids, p1 et p2 tel que p1 + p2 = P, peut donc toujours remplir un seul colis de poids P.

2.2 Que peut-on dire si un algorithme remplit deux colis à moins de la moitié de leur capacité ?

Si l'algorithme remplit deux colis à moins de la moité de leur capacité, la solution sortie par l'algorithme n'est pas optimale. L'algorithme doit normalement finir de remplir un colis avant de commencer un autre. Ici si l'on a p1 + p2 = (1/2)P et p3 + p4 = (1/2)P, la solution optimale et donc un seul colis avec p1, p2, p3, p4.

2.3 Proposer au moins deux algorithmes gloutons qui résolvent le problème du remplissage des lutins (mais pas nécessairement en donnant la solution optimale).

Algorithm 1 Algorithme remplissageSimple(tableau eniter poidsObj, tableau 2D entier poidsColis, entier poidsMax)

$nombreColis \leftarrow 0$
$sommePoids \leftarrow 0$
$listePoids \leftarrow vide$
for poids : poidsObjs **do**
  if $poids > poidsMax$ **then**
    ERREUR : impossible de le mettre dans un colis
    retourner
  end if
  // On regarde si l'on peut mettre le poids dans le colis courant
  if $sommePoids + poids \leq poidsMax$ **then**
    $sommePoids \leftarrow sommePoids + poids$
    listePoids.ajouter(poids)
  else
    // On ajoute le colis dans le tableau et l'on change de colis
    poidsColis.ajouter(listePoids)
    $listePoids \leftarrow vide$
    listePoids.ajouter(poids)
    $sommePoids \leftarrow 0$
    $sommePoids \leftarrow sommePoids + poids$
  end if
end for
// On regarde si l'on a bien ajouté tous les poids dans les colis
if $sommesPoids \neq 0$ **then**
  poidsColis.ajouter(listePoids)
  $listePoids \leftarrow vide$
  $sommePoids \leftarrow 0$
end if

Algorithm 2 Algorithme maximumObjParColis(tableau eniter poidsObj, tableau 2D entier poidsColis, entier poidsMax)

```
trieCroissant(poidsObj);
remplissageSimple(poidsObj, poidsColis, poidsMax);
```

Algorithm 3 Algorithme maximalRapportParColis(tableau eniter poidsObj, tableau 2D eniter poidsColis, entier poidsMax)

```
trieCroissant(poidsObj, poidsMax); // Fait un trie par rapport element dans poidsObj/poidsMax
remplissageSimple(poidsObj, poidsColis, poidsMax);
```

2.4 Proposer également un algorithme qui trouve toujours la solution optimale (mais qui peut être lent).

Algorithm 4 Algorithme remplissageOptimisé(tableau eniter poidsObj, tableau 2D entier poidsColis, entier poidsMax)

```
trieDecroissant(poidsObj)
if poidsObj[0] > poidsMax then
  ERREUR : impossible de le mettre dans un colis
  retourner
end if
tailleTableauObj ← poidsObj.size()
listePoids ← vide
for i de 0 à tailleTableauObj do
  // -1 <=> l'objet est déjà dans un colis
  if poidsObj[i] == −1 then
    continue
  end if
  sommePoids ← 0
  for j de i + 1 à tailleTableauObj do
    // -1 <=> l'objet est déjà dans un colis
    if poidsObj[j] == −1 then
      continuer
    end if
    // Si le colis courant est vide
    if sommePoids == 0 then
      listePoids.ajouter(poidsObj[i])
      sommesPoids ← sommPoids + poidsObj[i]
      poidsObj[i] = −1 // On indique qu'on a mis l'objet dans un colis
    end if
    // On regarde si l'on peut mettre poids dans un colis
    if sommePoids + poidsObj[j] ≤ poidsMax then
      listePoids.ajouter(poidsObj[j])
      sommePoids ← lsommPoids + poidsObj[j]
      poidsObj[j] ← −1
    end if
  end for
  // On ajoute les derniers poids dans le dernier colis
  if !listePoids.empty() then
    poidsColis.ajouter(listePoids)
  end if
  listePoids ← vide
  sommePoids ← 0
end for
// Si l'on a pas mis tous les poids dans les colis
for int poids : poidsObj do
  if poids ≠ −1 then
    poidsColis.ajouter(poids)
  end if
end for
if !listePoids.empty() then
  poidsColis.ajouter(listePoids)
end if
```

2.5 Faîtes tourner vos algorithmes sur l'exemple : (2, 6, 1, 5, 8, 4, 5, 7, 5, 3) et P = 9 et donner la réponse obtenue.

Sortie algorithme replissage simple :
Affichage du tableau ((2, 6, 1,), (5,), (8,), (4, 5,), (7,), (5, 3,),).
Nombre de colis : 6
Sortie algorithme via maximum objet par colis :
Affichage du tableau ((1, 2, 3,), (4, 5,), (5,), (5,), (6,), (7,), (8,),).
Nombre de colis : 7
Sortie algorithme via maximum rapport poidsObj/poidsMax par colis :
Affichage du tableau ((3, 5), (7,), (5, 4), (8,), (5, 1,), (6, 2),).
Nombre de colis : 6
Sortie de l'algorithme optimisé :
Affichage du tableau ((8, 1,), (7, 2,), (6, 3,), (5, 4,), (5,), (5 ,),).
Nombre de colis : 6

2.6 Donner pour chacun de vos algorithmes gloutons une entrée pour laquelle il trouve une solution optimale.

Algorithme replissage simple :
Entree optimale : (8,1,7,2,6,3,5,4,5,5), $P = 9$
Algorithme via maximum objet par colis :
Entree optimal : (1,2,2,3,3,4,4,5) $P = 9$
Algorithme via maximum rapport poidsObj/poidsMax par colis :
Entree optimal : (2,6,1,5,8,4,5,7,5,3), $P = 9$

2.7 Le facteur d'approximation d'un algorithme est le pire rapport, sur toutes les entrées possibles, entre le nombre de colis trouvé par l'algorithme divisé par le nombre de colis optimal. Pour un algorithme qui trouve la solution optimale, ce facteur sera toujours un. Donner une borne sur le facteur d'approximation d'un de vos algorithmes glouton.

Pour la suite, "opti" représente le nombre de colis de la solution optimale, et "algo" représente le nombre de colis donné par un algorithme glouton.
Facteur general : $opti \leq algo \leq 2opti$.
On peut calculer la valeur de "opti" : $opti = \frac{1}{2}n$, ce qui correspond à trouver pour tous les poids leurs complementaires tel que $P = p_x + p_y$.
Et la valeur de "2opti" correspond à un objet par colis donc $2n$.
On peut améliorer un peu cet encadrement en supposant que l'algorithme glouton, ratent des couples complementaires car les algorithmes gloutons appliquent

un remplissage simple après un trie particulier. On peut supposer qu'il rate 1/4 des formations des couples complementaires. Ainsi on augment de 1/4 le nombres de colis, l'encadrement devient :
Algorithme replissage simple : $opti \leq algo \leq 3/4opti$
Algorithme via maximum objet par colis : $opti \leq algo \leq 3/4opti$
Algorithme via maximum rapport poidsObj/poidsMax par colis : $opti \leq algo \leq 3/4opti$

3 Partie pratique

3.1 Implémentez les algorithmes de la partie précédente.

cf fichier dm.cpp

3.2 Implémentez un algorithme qui génère une entrée aléatoire avec n objets dont le poids est tiré aléatoirement entre 1 et m.

cf fichier dm.cpp

3.3 Pour chacun de vos algorithmes, calculer le nombre moyen de colis nécessaires, pour 1000 entrées avec 100 objets de poids entre 1 et 10 et un poids maximal par colis de 10. Vous calculerez également la moyenne sur 1000 entrées avec 100 objets de poids entre 1 et 1000 et un poids maximal par colis de 1000.

cf fichier dm.cpp