INF4705 – Analyse et conception d’algorithmes

TP3 – Hiver 2018

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Tremblay, David, 1748125  Souli, Taleb, 1770491 |
| **Note finale / 14** | 0 |

# Informations techniques

* Répondez directement dans ce document ODT avec LibreOffice. Veuillez ne pas inclure le texte en italique servant de directive. Utilisez LibreOffice et non Word, sinon ce document sera corrompu.
* La correction se fait à même le rapport.
* Avant le 17 avril à 23h59, vous devez faire une remise électronique en suivant les instructions suivantes:
  + Le dossier remis doit se nommer matricule1\_matricule2\_tp3 et doit être compressé sous format zip.
  + À la racine de ce dernier, on doit retrouver :
    - Ce rapport sous format ODT.
    - Un script nommé *tp.sh* servant à exécuter les différents algorithmes du TP. L’interface du script est décrite à la fin du rapport.
    - Un fichier texte nommé *emails.txt* contenant le courriel de chaque membre de l’équipe
    - Le code source et les exécutables
* Vous avez le choix du langage de programmation utilisé mais vous devrez utiliser les mêmes langage, compilateur et ordinateur pour toutes vos implantations. Le code et les exécutables soumis devront être compatible avec les ordinateurs de la salle L-4714. Conseil pour le TP3: Compilez avec -O3 si vous utilisez C++.
* Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.

# Mise en situation

Le dernier travail pratique se fera dans le cadre du concours du meilleur algorithme pour la session d'hiver 2018. Le travail demandé consiste à concevoir et implanter un algorithme de votre cru pour résoudre un problème combinatoire. Le classement des équipes déterminera votre note pour la qualité de l'algorithme. Votre algorithme sera exécuté sur 3 exemplaires de notre choix pendant 3 minutes.

Le problème à résoudre est une variante de celui du TP2. Vous disposez d'un ensemble de blocs de dimensions variées. Ici, on vous demande de construire plusieurs tours au lieu d'une seule en utilisant tous les blocs de l'exemplaire du problème. En plus des contraintes d'équilibre du TP2, les tours ne doivent pas dépasser une hauteur maximum *H*. L'objectif est de minimiser le nombre de tours construites tout en respectant les contraintes décrites précédemment. Il est interdit de faire une rotation des blocs.

Le rapport pour ce dernier travail pratique est assez succinct. Nous vous encourageons à terminer ce travail assez tôt afin de ne pas compromettre votre préparation à vos examens finaux.

# Jeu de données

Pour tester les algorithmes, vous disposez d’un jeu de données de 10 exemplaires (disponible sur moodle). La première ligne contient le nombre de blocs *n* suivit de la hauteur maximale *H* des tours sur la seconde ligne. Finalement, une liste de *n* triplets (*hauteur*,*largeur*,*profondeur*) représente les blocs.

*n*

*H*

a b c

d e f

h i j

...

### Description du sujet et des objectifs de travail

|  |  |
| --- | --- |
| 0 | / 1 pt |

Pour ce dernier travail pratique, le travail demandé consiste à implémenter un algorithme qui permet de trouver une solution à un problème similaire au dernier travail pratique. Toutefois, dans le cadre de ce travail, on cherche à construire un nombre minimal de tours toutes de grandeur inférieure ou égale à une valeur *H* en utilisant tous les blocs d’un ensemble *N* sans rotation de blocs possibles plutôt qu’une tour de grandeur maximale. L’ordonnancement des blocs dans la tour doit suivre les mêmes contraintes qu’au TP précédent par contre. C’est-à-dire que les blocs ne peuvent être empilés que si le bloc en dessous à une largeur et une longueur qui est strictement supérieure au bloc qu’on tente d’insérer. Il n’y a aucune contrainte sur la manière de concevoir notre algorithme et les solutions des équipes seront mises en compétition. Il y a toutefois une contrainte de temps (maximum 3 minutes) pour générer une solution et l’affichage de la solution doit aussi se conformer à un certain affichage.  
  
Les objectifs de ce travail sont donc de tout d’abord de penser à une manière de résoudre le problème dans les délais et avec une solution prometteuse (en s’inspirant des algorithmes vu dans le cours, ailleurs ou tout simplement d’en inventer une) et ensuite de l’implémenter et d’afficher les résultats selon les règles qui nous sont fournies.

### Présentation de votre algorithme

|  |  |
| --- | --- |
| 0 | / 2 pt |

Create a structure Bloc with int height, int width and int depth

Initialize int numberOfBlocs

Initialize int maxHeight

Initialize vector of Bloc finalTowers

**Function winner**(vector of bloc blocs)

sort the blocs by increasing area size

**While** blocs is not empty

Initialize vector of ints optimizedValues

**For** i from 1 to size of blocs minus 1 with step of 1

**For** j from 0 to value of i minus 1 with step of 1

**If** bloc at index i of blocs has smaller width and depth than bloc at index j of blocs and int at index i of optimizedValue is smaller than int at index j of optimizedValue

Set int at index i of optimizedValue is equal to int of optimizedValue at index j plus the height of bloc at index i of blocs

**End If**

**End For**

**End For**

Set int max to -1

Set int indexMaxHeight to 0

**For** i from 0 to size of blocs minus 1 with step of 1

**If** max is smaller than int at index i of optimizedValues

Set max to value of optimizedValue at index j

**End If**

**End For**

Initialize vector of bloc solution

Initialize vector of int indexToBeRemoved

Set int currentHeight to max minus height of bloc at index indexMaxHeight of blocs

Add bloc at index indexMaxHeight of blocs to solution

Add indexMaxHeight to indexToBeRemoved

**Fo**r i from indexMaxHeight minus 1 to i greater or equal to 0 with step of -1

**If** currentHeight is equal to optimizedValue at index i

Add bloc at index i of blocs to solution

Set currentHeight to currentHeight minus height of bloc at index i of blocs

Add i to indexToBeRemoved

**End If**

**End For**

Call function splitTowers with argument solution

**End While**

sort finalTowers by height

Call function optimiseSolution

**End Function**

**Function splitTowers**( vector of bloc solution)

Initialize vector of Bloc tower

Set int height to 0

Set int push to 0

**For** i from 0 to size of solution minus 1 with step of 1

Set height to height plus height of solution at index i

**If** height is smaller or equal to maxHeight

Add solution at index i to tower

**End If**

**Else**

Add tower to finalTowers

Set height to height of solution at index i

clear the content of tower

Add solution at index i to tower

**End Else**

**End For**

**If** size of tower is greater than 0

Add tower to finalTowers

**End If**

**Enf Function**

**Function optimiseSolution()**  
 **For** i from finalTowers size to 1  
 **For** j from finalTowers at i size to 1  
 bool isBlocInserted = false;  
 **For** k from 0 to i {   
 **If** current block height plus tower height is less or equal maxHeigh

try to insert the block in the tower  
 **If** block was inserted

ercase the bloc from the the tower where it was

**End If**

**End If**

**End For**

If current tower become empty

erace the tower from the final towers vector

**End If**

**End For**

**End Function**

### Justification de l’originalité de votre algorithme

|  |  |
| --- | --- |
| 0 | / 3 pt |

Notre algorithme a été inspiré de la programmation du travail pratique précédent. L’algorithme vorace, bien que très rapide à implémenter, ne garantissait pas une solution assez prometteuse même avec l’ajout d’une probabilité et notre équipe a eu beaucoup de difficulté à implémenter l’algorithme de recherche tabou. Nous étions donc confiant que malgré une contrainte de temps de 3 minutes, notre algorithme nous donnerait une meilleure réponse dans les délais en s’inspirant de cet algorithme déterministe.

Le principe est le suivant, nous exécutons l’algorithme probabiliste du TP2 mais nous arrêtons la construction de la tour dès que l’ajout du prochain bloc de la solution mènerait à une tour plus grande que la grandeur maximale. Nous sauvegardons cette tour et retirons ces blocs des blocs disponibles et itérons jusqu’à ce qu’il ne reste plus de blocs disponibles. Une fois toutes ces tours obtenues, nous améliorons la solution de la manière suivante: nous trions les tours en ordre de hauteur puis nous prenons un bloc à la fois (en partant du bloc inférieur de la plus petite tour) et tentons de le placer dans une autre tour (en partant de la tour la plus haute). Ainsi, les tours les plus petites au départ ne contenant qu’un ou deux blocs sont éliminées afin d’obtenir une solution encore meilleure.

**Analyse Asymptotique:**

Cet algorithme contient tout d’abord la méthode std::sort de complexité **O(nlog(n))**. Nous avons ensuite une boucle  **for** sur l’ensemble des blocs et donc **O(n)** avec une complexité de **O(1)** à l’intérieur. Nous avons par la suite 2 boucles  **for** imbriquées. La première est encore pour tous les blocs et donc **O(n)** et la seconde est aussi en **O(n)** et l’intérieur des boucles est en **O(1)**. Nous obtenons donc une complexité de **O(*n2*)** pour ces boucles. Une autre boucle for sur l’ensemble des blocs de complexité **O(n)** suit ces boucles. Ensuite nous avons une dernière boucle for sur une partie des blocs disponibles (disons m, *où* m ≤ n) et donc de complexité **O(m).** Finalement nous appelons une méthode pour optimiser la solution déjà trouvée, la fonction contient trois boucle imbriquée, ce qui nous donne une complexité de **O(nf.nf.log(nf))**, soit nf le nombre de tours final trouvé avant l’appel à la fonction d’optimisation. Notre algorithme aura donc une complexité de **max(O(nlog(n), O(n), O(*n2*), O(1), O(m), O(nfxnfxlog(nf)))**  et donc **O(n2)**.

# Autres critères de correction

### Respect de l’interface tp.sh

|  |  |
| --- | --- |
| 0 | / 1 pt |

Utilisation

tp.sh -e [path\_vers\_exemplaire]

Lorsque exécuté sans le paramètre -p, le programme affiche uniquement le nombre de tours chaque fois qu’une meilleure solution est trouvée.

Arguments optionnels

-p chaque fois qu’une meilleure solution est trouvée, le programme doit afficher, pour chaque tour, le nombre de blocs suivi de triplets (*hauteur*,*largeur*,*profondeur*) pour indiquer les blocs utilisés chacun dans une ligne séparée dans l'ordre décroissant de surface. L’affichage d’une solution se termine avec “fin”.

Important: l’option -e doit accepter des fichiers avec des paths absolus.

Voici un exemple d’affichage pour 2 tours de 3 et 4 blocs (les lettres de l’alphabet représentent les dimensions):

3

a b c

d e f

g h i

4

j k l

m n o

p q r

s t u

fin

Il est d'une grande importance de respecter le format de sortie, puisque l'évaluation de la qualité de votre algorithme passera d'abord par le test de l'intégrité de la solution. Ce dernier échouera si votre format de sortie n'est pas respecté et votre programme serait considéré non fiable ce qui entraîne une note 0 sur 4 pour la qualité de l'algorithme. A cet effet, on vous suggère fortement de commencer tout d'abord par implanter un petit programme qui vérifie l'intégrité de vos solutions, ainsi au fur et à mesure que vous apportiez des modifications à votre algorithme, votre programme fait le test pour s'assurer que les solutions sont valides.

### Qualité de l’algorithme

|  |  |
| --- | --- |
| 0 | / 4 pt |

### Qualité du code

|  |  |
| --- | --- |
| 0 | / 1 pt |

### Présentation générale

|  |  |
| --- | --- |
| 0 | / 2 pt |

* Concision
* Qualité du français

### Pénalité retard et autres

|  |
| --- |
| 0 |

* -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.