INF4705 – Analyse et conception d’algorithmes

TP2 – Automne 2018

|  |  |
| --- | --- |
| **Nom, prénom, matricule des membres** | Badirou Salim, 1733231  Badirou Bissola, 1770039 |
| **Note finale / 13** | 0 |

# Informations techniques

* Répondez directement dans ce document ODT avec LibreOffice. Veuillez ne pas inclure le texte en italique servant de directive.
* La correction se fait à même le rapport.
* Avant votre cinquième séance de laboratoire (16 novembre 2018 à 12h45), vous devez faire une *remise électronique sur Moodle* en suivant les instructions suivantes :
  + Le dossier remis doit se nommer matricule1\_matricule2\_tp2 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. Notez que le code et les exécutables soumis seront testés sur les ordinateurs de la salle L-4714 et doivent être compatibles avec cet environnement. En d’autres mots, tout doit fonctionner correctement lorsque le correcteur exécute votre script *tp.sh* sur un des ordinateurs de la salle.
* 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

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l'analyse et la conception d'algorithmes développés suivant différents patrons de conception.

Vous venez de décrocher l'emploi de vos rêves à Vancouver. Malheureusement, le prix des logements dans cette ville est prohibitif et vous auriez du mal à boucler votre budget. Néanmoins, en bon polytechnicien instruit que vous êtes, vous tombez sur l'article suivant :

<http://www.cbc.ca/news/canada/british-columbia/bunker-underground-vancouver-1.4345255>

Voilà, il suffit de creuser un trou et d'y habiter! Dans un élan d'inspiration, vous vous rappelez qu'un de vos oncles est dynamiteur. En lui empruntant un peu de dynamite, il sera possible de faire un trou sans même utiliser une pelle. De plus, avec la bonne quantité d'explosif, vous pourrez vous rapprocher des conduites d'eau afin d'avoir l'eau courante... Mais attention, si on dépasse cette quantité, on brise les conduites!

Vous avez à votre disposition *n* bâtons de dynamite de poids respectifs *p1, p2, ..., pn* et vous calculez que vous aurez besoin d'un poids total *P*. Vous devez donc déterminer quelle combinaison de bâtons utiliser pour s'approcher le plus possible du poids total *P* mais sans le dépasser! Vous minimiserez ainsi la quantité de terre qu'il reste à pelleter pour atteindre les conduites d'eau.

# Implantation

Quatre algorithmes seront implantés, mettant en pratique les patrons de conception glouton, probabiliste, programmation dynamique et recuit simuler.

## Algorithme glouton

Vous devez concevoir un algorithme glouton déterministe de votre cru pour résoudre ce problème.

## Algorithme de programmation dynamique

Vous devez également résoudre ce problème à l'aide de la programmation dynamique. Vous devez implanter deux algorithmes qui consistent à adapter les deux versions de “Faire de la monnaie” vues dans les capsules vidéo ainsi que dans le chapitre 6 des diapositives. Pour un exemplaire de *n* bâtons de dynamite avec un poids *P* qu’il ne faut pas dépasser :

* Premier essai : la solution se trouve à *c[n]*
* Deuxième essai : la solution se trouve à *c[n, P]*

## Recuit simulé

Enfin vous devez implanter une méta-heuristique basée sur le recuit simulé. La fonction *voisine* renvoie une solution voisine de la solution passée en argument. Une solution voisine est obtenue en choisissant uniformément au hasard un bâton parmi ceux qui ne sont pas encore choisis et en l’ajoutant à la solution. Il est possible que cela rende le poids total supérieur à *P* : on retirera alors des bâtons un à un (choisis uniformément au hasard) jusqu’à ce que le poids total ne dépasse pas *P*. La fonction *somme* retourne le poids total d’une solution.

*recuit*(*S*0, *T*, *k*max, *P*, α):

*1 S* ← *S*0

*2 S*meilleur ← *S*

*3 θ*1 ← *T*

4 **for** *k* = 1 ... *k*max **do**

5 **for** *j* = 1 ... *P* **do**

6 *S*’ ← *voisin*(S)

7 Δ ← *somme*(*S’*) - *somme*(*S*)

8 **if** Δ ≥ 0 **or** eΔ/θk ≥ *unif*([0,1]) **then**

9 *S* ← *S*’

10 **if** *somme*(*S*) > *somme*(*S*meilleur) **then**

11 *S*meilleur ← *S*

12 **end** **if**

13 **end if**

14 **end for**

15 *θ*k+1 *← θ*k × α

16 **end for**

17retourner *S*meilleur

1 : La solution courante *S* est égale à une certaine solution initiale valide *S*0(par exemple la solution de votre algorithme glouton).

2 : On garde en mémoire la meilleure solution trouvée jusqu’à présent.

3 : La température courante θk est égale à la température initiale *T.*

4 et 5: Pour chacune des *k*maxitérations et pour chacun des *P* paliers de température.

6 : On génère une nouvelle solution *S’* (voisine de la solution *S*)*.*

7 à 9 : Si la nouvelle solution est meilleure que la solution courante ou si la condition de probabilité est rencontrée, on met à jour la solution courante.

10 et 11 : On met à jour la meilleure solution si nécessaire.

15 : On met à jour la température.

16 : En terminant l’algorithme retourne la meilleure solution qu’il a pu trouver.

# Jeu de données

Les jeux de données du problème se trouvent dans le répertoire /jeux\_de\_donnees. Les exemplaires sont regroupés par nombre de bâtons *n* et ordre de grandeur du poids total *m* : ainsi le fichier "WC-n-m-k.txt" décrit le *kième* exemplaire du groupe.

La structure des fichiers d'exemplaires est :

Sur la première ligne : nombre de bâtons *n*

Une ligne pour chaque bâton : *i* <espace> *pi*

Sur la dernière ligne : poids total limite *P*

# Présentation des résultats

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

### Tableau des résultats

*Exécutez chacun des trois algorithmes en notant leur temps d'exécution et le poids total, mais ne rapportez dans un tableau que le temps moyen et l'écart relatif moyen au poids total limite de chacune des séries de dix exemplaires. Pour le recuit simulé, exécutez chaque exemplaire cinq fois et utilisez la médiane de ces exécutions.*

*Glouton*

|  |  |  |
| --- | --- | --- |
| Exemplaires | Temps moyen | Écart relatif moyen |
| 1000-10 | 0.0017108 | 0 |
| 1000-100 | 0 | 0 |
| 1000-1000 | 0.00166221 | 0.1 |
| 1000-4000 | 0.00166178 | 4.4 |
| 100-10 | 0 | 0 |
| 100-100 | 0 | 0.2 |
| 100-400 | 0 | 7.5 |
| 100-1000 | 0 | 4.6 |
| 10-10- | 0 | 0 |
| 10-100 | 0 | 3.3 |
| 10-1000 | 0 | 58.2 |
| 2000-10 | 9.98E-05 | 0 |
| 2000-100 | 0.00019946 | 0 |
| 2000-1000 | 0 | 0 |
| 200-10 | 0 | 0 |
| 200-100 | 0.00019951 | 0 |
| 200-1000 | 0 | 1.4 |
| 20-10 | 0 | 0 |
| 20-100 | 0 | 1.8 |
| 20-1000 | 0 | 27 |
| 500-10 | 0 | 0 |
| 500-100 | 9.98E-05 | 0 |
| 500-1000 | 9.97E-05 | 0.6 |
| 500-2000 | 0 | 2 |
| 50-10 | 0 | 0.1 |
| 50-100 | 0 | 3.3 |
| 50-1000 | 0 | 8.6 |

*Progdyn1*

|  |  |  |
| --- | --- | --- |
| Exemplaires | Temps moyen | Écart relatif moyen |
| 1000-10 | 0.010077024 | 0 |
| 1000-100 | 0.602974534 | 0 |
| 1000-1000 | 57.36136918 | 0 |
| 1000-4000 | 0.825087547 | 0 |
| 100-10 | 0 | 0 |
| 100-100 | 0.004017663 | 0 |
| 100-400 | 0.006659937 | 0 |
| 100-1000 | 0.486755872 | 0 |
| 10-10- | 0 | 0 |
| 10-100 | 0 | 0 |
| 10-1000 | 0.079584408 | 0 |
| 2000-10 | 0.027476954 | 0 |
| 2000-100 | 1.889449859 | 0 |
| 2000-1000 | 179.6382599 | 0 |
| 200-10 | 0 | 0 |
| 200-100 | 0.016022372 | 0 |
| 200-1000 | 1.896989369 | 0 |
| 20-10 | 0.001563025 | 0 |
| 20-100 | 0.001561761 | 0 |
| 20-1000 | 0.084066558 | 0 |
| 500-10 | 0.00312438 | 0 |
| 500-100 | 0.11978333 | 0 |
| 500-1000 | 10.9961952 | 0 |
| 500-2000 |  |  |
| 50-10 | 0 | 0 |
| 50-100 | 0.007804465 | 0 |
| 50-1000 | 0.156425158 | 0 |

*Progdyn2*

|  |  |  |
| --- | --- | --- |
| Exemplaires | Temps moyen | Écart relatif moyen |
| 1000-10 | 0.15752809 | 0 |
| 1000-100 | 1.698575711 | 0 |
| 1000-1000 | 9.954168367 | 0 |
| 1000-4000 | 1.429170322 | 0 |
| 100-10 | 0.000502896 | 0 |
| 100-100 | 0.006253266 | 0 |
| 100-400 | 0.015307641 | 0 |
| 100-1000 | 0.094916677 | 0 |
| 10-10- | 0 | 0 |
| 10-100 | 9.97782E-05 | 2.9 |
| 10-1000 | 0.006248927 | 45.8 |
| 2000-10 | 0.447731137 | 0 |
| 2000-100 | 4.245900536 | 0 |
| 2000-1000 | 39.68679059 | 0 |
| 200-10 | 0.004923606 | 0 |
| 200-100 | 0.036824393 | 0 |
| 200-1000 | 0.382324815 | 0 |
| 20-10 | 0 | 0 |
| 20-100 | 9.97305E-05 | 0 |
| 20-1000 | 0.008869481 | 0.8 |
| 500-10 | 0.024993753 | 0 |
| 500-100 | 0.256491351 | 0 |
| 500-1000 | 2.3878613 | 0 |
| 500-2000 |  |  |
| 50-10 | 9.06424E-05 | 0 |
| 50-100 | 0.004686785 | 0 |
| 50-1000 | 0.027771155 | 0 |

*Recuit*

|  |  |  |
| --- | --- | --- |
| Exemplaires | Temps moyen | Écart relatif moyen |
| 1000-10 | 0,00149643 | 0 |
| 1000-100 | 0,00169365 | 0 |
| 1000-1000 | 0,00189183 | 0,1 |
| 1000-4000 | 0,00139618 | 4,1 |
| 100-10 | 9,9826E-05 | 0 |
| 100-100 | 9,9969E-05 | 0,2 |
| 100-400 | 0,00029867 | 7,5 |
| 100-1000 | 0,00019963 | 3,6 |
| 10-10- | 0 | 0 |
| 10-100 | 9,9754E-05 | 2,7 |
| 10-1000 | 0 | 43,9 |
| 2000-10 | 0,00299265 | 0 |
| 2000-100 | 0,00349388 | 0 |
| 2000-1000 | 0,00329099 | 0 |
| 200-10 | 0,00019939 | 0 |
| 200-100 | 0,00039909 | 0 |
| 200-1000 | 0,00039842 | 1,4 |
| 20-10 | 9,973E-05 | 0 |
| 20-100 | 9,9492E-05 | 1,8 |
| 20-1000 | 0 | 17,9 |
| 500-10 | 0,00099432 | 0 |
| 500-100 | 0,00069828 | 0 |
| 500-1000 | 0,00089755 | 0,6 |
| 50-2000 | 0,0007982 | 2 |
| 50-10 | 9,9707E-05 | 0,1 |
| 50-100 | 9,9731E-05 | 3,2 |
| 50-1000 | 0,00019646 | 8,6 |

### Graphiques pour analyse hybride

*Voir questions plus bas.*

# Analyse et discussion

|  |  |
| --- | --- |
| 0 | / 6 pt |

### Donnez une brève description de votre algorithme glouton.

*Soyez concis, seulement une ou deux phrases suffisent.*

Pour l’algorithme glouton, nous avons décidé de trier les poids par ordre croissant, puis nous additionnons les poids jusqu’à se rapprocher le plus près possible du poids maximal. La combinaison de poids final est donc composée des poids additionnés.

### Discutez du choix de votre combinaison de paramètres (*T, kmax, P, α*) pour le recuit simulé.

*Discutez brièvement des différentes combinaisons que vous avez essayées et de vos hypothèses sur l’impact de ces différents paramètres.*

Pour la combinaison finale de paramètres, nous avons choisi : (95,10,10,0.6). Avant cela nous avions essayé quelque combinaisons avec le T plus grand (100,1000,…) et le alpha plus grand aussi (0.8, 0.9…). Nous avons remarqué qu’après plusieurs exécutions successives, le temps d’exécution augmentait mais que l’écart relatif avec le poids limite réduisait. Cependant avec l’augmentation des paramètres kmax et P, le temps d’exécution augmente et les solutions changent, soit l’écart relatif au poids moyen est plus grand. Certes lorsque l’on réduit la valeur de alpha, le temps d’exécution diminue relativement. Même lorsque l’on prend une valeur trop grande pour T, le temps reste relativement plus court si on réduit la valeur de alpha. Nous pensons alors que des valeurs assez grandes pour kmax, P et alpha augmente considérablement le temps d’exécution. Nous pensons aussi que pour avoir un écart relatif avec le poids limite pas très haut il faut avoir une valeur de alpha moyenne, un T assez grand et avoir une valeur moyenne pour kmax.

### Tentez une analyse asymptotique du temps de calcul pour chaque algorithme.

*Si vous préférez écrire vos équations en Latex, vous pouvez ajouter un pdf à la remise avec la réponse à cette question et le mentionner ici.*

*Glouton :* O(nlgn) - nous avons le tri qui se fait en O(nlgn) et la boucle while se fait en O(lgn) & la lecture du fichier se fait en O(n); en prenant le maximum on se retrouve à O(nlgn)

*Progdyn1 :* O(2n) – genSolution fait deux fois un appel récursif ; le 1er for de la boucle se fait en O(n) et le 2e se fait en O((m+1)2/2) ; en prenant le maximum on se retrouve à O(2n)

*Progdyn2 :* O(n\*m) – 1er for se fait en O(n+1), le 2e for en O(n\*m), le while en O(nlgn) & la lecture du fichier se fait en O(n); en prenant le maximum on se retrouve à O(n\*m)

*Recuit simulé :* O(kmax\*nlgn) – gloutonSolutionVoisine se fait en O(nlgn) donc dans recuit la multiplication des complexités des boucles for nous donne une complexité de O(kmax\*nlgn) et glouton se fait en O(nlgn) ; en prenant le maximum on se retrouve à O(kmax\*nlgn)

### Servez-vous de vos temps d'exécution pour confirmer et/ou préciser l'analyse asymptotique théorique de vos algorithmes avec la méthode hybride de votre choix.

*La méthode peut varier d'un algorithme à l'autre. Justifiez les choix ici et mettez les graphiques dans la section précédente.*

### Discutez des quatre algorithmes en fonction de la qualité respective des solutions obtenues, de la consommation de ressources (temps de calcul, espace mémoire) et de la difficulté d'implantation.

*Glouton* fut l’algorithme le plus facile à implanter et il est d’ailleurs assez efficace et l’un des algorithmes les plus rapides. Il ne consomme pas trop d’espace mémoire, certes il n’est pas le plus optimal quant aux résultats. Ensuite, nous avons les deux algorithmes de programmation dynamique qui sont d’ailleurs les meilleurs du point de vue des solutions obtenues. On atteint toujours le poids total limite donc il est très efficace. Cependant, ils sont aussi les plus lents donc requièrent le plus de temps de calcul et d’espace mémoire. Le *progdyn1* a été le plus difficile à implanter à, car a la base, nous devions tenir compte de la non-répétions des bâtons de dynamiques et par conséquent revoit l’algorithme au complet (trouver et changer la relation de récursivité, redéfinir les valeurs frontières en fonction du point précèdent). Mais tous ces problèmes se sont résolus par l’utilisation de l’algorithme de base (Piece de monnaie qui accepte la répétition de bâtons). Le *progdyn1* assez rapidement, mais perd énormément de temps à cause de la récursivité et du moyen de générer la solution. La récursivité est d’ailleurs très couteuse. *Progdyn2* fut quand même assez simple à implanter, quoique les multiples boucles sont en effet couteuses aussi. Pour finir, *recuit* est aussi très efficace, car il atteint souvent le poids limite sauf quand le poids est très grand. De plus, c’est un algorithme qui est très rapide, dans certain cas plus rapide que glouton malgré sa complexité et ne consomme pas énormément de ressources. En revanche, c’est un algorithme très difficile à implanter puisqu’il est assez aléatoire. Il est difficile de le prédire et nous devons nous même trouver les paramètres. On peut ainsi dire que son efficacité dépend des paramètres qu’on lui donne.

### Indiquez sous quelles conditions vous utiliseriez chacun des algorithmes.

Nous pensons qu’on choisira l’algorithme dépendamment de ce qu’on veut prioriser entre rapidité ou meilleure solution. Lorsque l’on a des poids pas trop grands, on constate que recuit simulé est non seulement rapide, mais sa solution est idéale. Donc pour des poids limites pas trop grands, nous prioriserons le recuit simulé, de plus il reste un bon compromis entre rapidité et optimisation du résultat. Lorsque nous avons des poids très grands et que l’exactitude de la solution est prioritaire, proddyn1 est la meilleure solution. Le temps de calcul peut s’avérer très long, mais son résultat est très optimal. Cependant, même avec des poids assez grands et l’on est prêt à faire une concession sur la qualité de la solution glouton ou progdyn2 sont les algorithmes à prioriser. Glouton n’est pas le moins optimal donc ce serait le dernier choix. Quoiqu’avec des petits poids et dépendamment des valeurs de l’exemplaire, il peut s’avérer utile.

# Autres critères de correction

### Respect de l’interface tp.sh

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

Utilisation :

tp.sh -a [glouton | progdyn1 | progdyn2 | recuit] -e [path\_vers\_exemplaire]

Arguments optionnels :

-p affiche la solution (rangs des bâtons utilisés) sur une ligne, avec chaque bâton séparé par un espace

-t affiche le temps d’exécution en ms, sans unité ni texte superflu

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

### Qualité du code

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

### Présentation générale (concision, qualité du français, etc.)

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

### Pénalités

|  |
| --- |
| 0 |

* Retard : -1 pt / journée de retard, arrondi vers le haut. Les TPs ne sont plus acceptés après 3 jours.
* Autres : Le correcteur peut attribuer d’autres pénalités (par exemple si les exécutables sont manquants, fichier *emails.txt* manquant, etc.)

# Fichiers fournis

Outre le présent rapport que vous devez compléter, les autres fichiers fournis sont les suivants :

### Le répertoire “*jeux\_de\_donnees*”

Ce répertoire contient les jeux de données nécessaires au TP.

### Le répertoire “*exemple”*

Ce répertoire est *fourni à titre d’exemple* et vous êtes libres de vous en servir, de le modifier, ou de vous en inspirer. Il inclut un script *tp.sh* fonctionnel (avec des implantations bidon des algorithmes pour les besoins de l’exemple).

### Le répertoire “*verification”*

Ce répertoire comporte trois exemplaires vous permettant d’avoir une idée si vos algorithmes de *programmation dynamique* fonctionnent correctement. Les solutions optimales pour chacun des exemplaires sont respectivement : 57, 97 et 100 (poids maximal atteint). Notez que ce n’est pas une méthode parfaite de validation : si un de vos algorithmes de programmation dynamique donne une mauvaise réponse pour un des exemplaires vous avez la preuve qu’il a été mal implanté, mais s’il donne une bonne réponse pour chaque exemplaire ce n’est pas nécessairement une preuve qu’il a été bien implanté.

# Notes techniques

* La commande *chmod +x mon\_script.sh* rendra le script *mon\_script.sh* exécutable. Pour l’exécuter il s’agira de faire *./mon\_script.sh*
* Le guide “guide bash” sur le site Moodle du cours n’est pas spécialement adapté au présent TP, mais il vous aidera à mieux comprendre la façon dont les différents scripts fonctionnent.