INF8775 – Analyse et conception d'algorithmes

TP1 – Hiver 2021

Nom, prénom, matricule des membres	NOM, Prénom, 1234567 NOM, Prénom, 1234567
Note finale / 13	0

Informations techniques

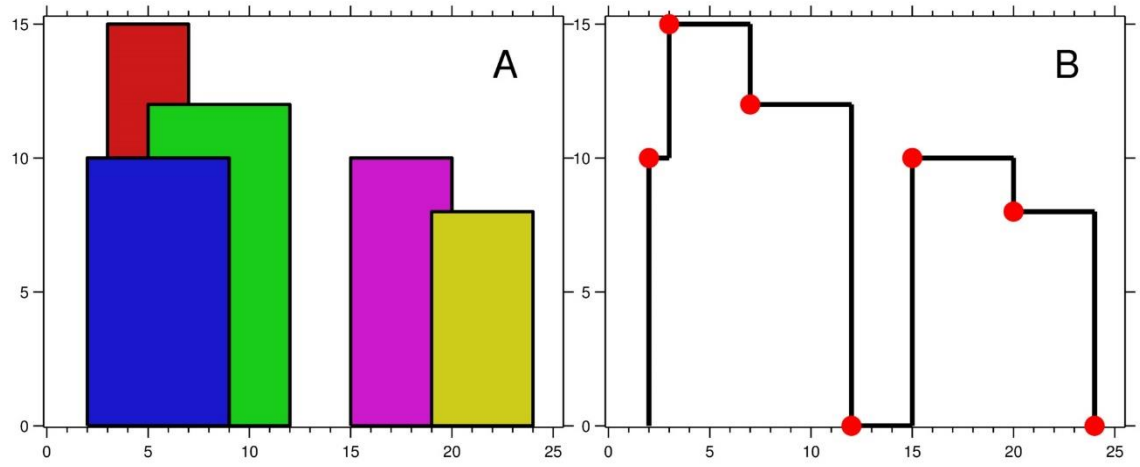
- Répondez directement dans ce document ODT. Veuillez ne pas inclure le texte en italique servant de directive.
- La correction se fait sur ce même rapport.
- Vous devez faire une remise électronique sur Moodle avant le 5 mars à 23h59 en suivant les instructions suivantes :
  - Vos fichiers doivent être remis dans une archive zip à la racine de laquelle on retrouve :
    - 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.
    - 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 compatibles avec les ordinateurs de la salle L-4714.
- Si vous utilisez des extraits de codes (programmes) trouvés sur Internet, vous devez en mentionner la source, sinon vous serez sanctionnés pour plagiat.
- On vous encourage à lire le guide intitulé « guide bash » sur Moodle pour faire vos graphiques. C'est un guide qui a été conçu pour un ancien TP, mais il contient beaucoup d'informations utiles.

Mise en situation

Ce travail pratique se répartit sur deux séances de laboratoire et porte sur l'analyse empirique et hybride des algorithmes. Dans les capsules vidéo de la semaine 3, trois approches d'analyse de l'implantation d'un algorithme sont décrites. Vous les mettrez en pratique pour des algorithmes de résolution d'un problème connu.

Description du problème

On vous demande de résoudre le problème de la ligne d'horizon[1] (*The Skyline Problem*) qui consiste à dessiner la silhouette de bâtiments lorsqu'ils sont vus de loin. Ces bâtiments sont juxtaposés l'un à l'autre et il est possible que l'un en cache un autre. Soit l'exemple suivant avec 5 bâtiments :



La figure B représente la silhouette (et donc la solution) tracée par les bâtiments colorés de la figure A.

Concrètement, chaque bâtiment est défini par le triplet (x_1, x_2, h) avec h la hauteur du bâtiment et x_1 et x_2 les abscisses des murs gauche et droit, respectivement. La solution quant-à-elle représente une suite de couples (x, h) représentant les coordonnées des points définissant la silhouette des bâtiments.

Pour l'exemple ci-dessus :

- Les données du problème sont telles que : (2, 9, 10), (3, 7, 15), (5, 12, 12), (15, 20, 10), (19, 24, 8).
- La solution est telle que : (2, 10), (3, 15), (7, 12), (12, 0), (15, 10), (20, 8), (24, 0).

Algorithmes à implanter

On vous demande de résoudre ce problème de 3 façons différentes :

1. En utilisant un algorithme force brute simple ;
2. En utilisant un algorithme diviser pour régner ;
3. En utilisant un algorithme diviser pour régner avec seuil de récursivité non élémentaire.

Pour l'algorithme 3, vous devez déterminer un seuil de récursivité expérimentalement. Les exemplaires dont la taille est en deçà de ce seuil ne sont plus résolus récursivement mais plutôt directement avec l'algorithme 1.

[1] D'après https://leetcode.com/problems/the-skyline-problem/

Jeu de données

Pour tester les algorithmes, vous devez générer un jeu de données avec au moins 5 exemplaires pour chacune des tailles suivantes : 1000, 5000, 10000, 50000, 100000 et 500000. On vous fournit un script python `inst_gen.py` pour générer vos exemplaires. Son usage est le suivant :

```
$ ./inst_gen.py [-h] -s NB_BATIMENTS [-n NB_EXEMPLAIRES]
```

On suggère d'utiliser le script bash suivant pour automatiser la génération des exemplaires :

```
for n in {"1000","5000","10000","50000","100000","500000"}; do
    ./inst_gen.py -s $n -n 5
done
```

La première ligne de chaque exemplaire représente le nombre de bâtiments N (qui est aussi la taille du problème). Chacune des N lignes subséquentes représente les triplets discutés dans « Description du problème » avec leurs valeurs entières séparées par des espaces. Par ailleurs, on garantit que tous les triplets générés sont triés selon x_1 et que chacun d'eux est tel que $x_2 - x_1 \geq 1$.

Contraintes sur les solutions

Vos algorithmes doivent donner des réponses où les couples (x, h) sont triés de façon non décroissante selon x (cf. exemple plus haut). Par ailleurs, ils ne doivent pas donner de solutions avec couples redondants, i.e. deux couples qui se suivent ne peuvent pas avoir la même hauteur ni la même abscisse.

Présentation des résultats

	/ 4 pts

Tableau des résultats

Pour chacun des trois algorithmes, mesurez le temps d'exécution des exemplaires et rapportez dans un tableau le temps moyen pour chaque taille d'exemplaire.

Lorsque vous calculez les temps d'exécution, vous devez séparer le temps de chargement du jeu de test du temps d'exécution de votre algorithme. Vous devrez donc insérer des sondes temporelles à l'intérieur de votre code.

Tests de puissance

Pour chacun des algorithmes, appliquez le test de puissance et rapportez les graphiques ici.

Test du rapport

Pour chacun des algorithmes, appliquez le test du rapport et rapportez les graphiques ici.

Test des constantes

Pour chacun des algorithmes, appliquez le test des constantes et rapportez les graphiques ici.

Analyse et discussion

	/ 6 pt

Que pouvez-vous déduire du test de puissance ?

Citez la consommation théorique du temps de calcul pour les algorithmes, en notation asymptotique.

Nul besoin de faire une preuve, on demande seulement de citer.

Que pouvez-vous déduire du test du rapport ?

Que pouvez-vous déduire du test des constantes ?

Discutez de l'impact du seuil de récursivité.

Suite à cette analyse, indiquez sous quelles conditions (taille d'exemplaire ou autre) vous utiliseriez chacun de ces algorithmes. Justifiez.

Autres critères de correction

Respect de l'interface `tp.sh`

	/ 1 pt

Utilisation :

```
$ ./tp.sh -a {brute, recursif, seuil} -e CHEMIN_EXEMPLAIRE [-p] [-t]
```

Arguments optionnels :

> `[-p]` affiche, sur chaque ligne, les couples définissant la silhouette de bâtiments, triés selon l'abscisse et sans texte superflu (les deux valeurs d'un couple sont séparées d'un espace) ;
> `[-t]` affiche le temps d'exécution en millisecondes, sans unité ni texte superflu.

Important : l'option `-e` doit pouvoir accepter des chemins absolus.

Qualité du code

	/ 1 pt

Présentation générale

	/ 1 pt

- Concision
- Qualité du français

Pénalité retard

0

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