Mon bô tablô
Projet d'ASD3

A réaliser en Java, et en binôme

Octobre 2024

1 Introduction

Piet Mondrian est un peintre du début du 20ème siècle, pionnier de l'art abstrait. Son oeuvre est caractérisée par des tableaux très géométriques, comme celui présenté en Figure 1[1].

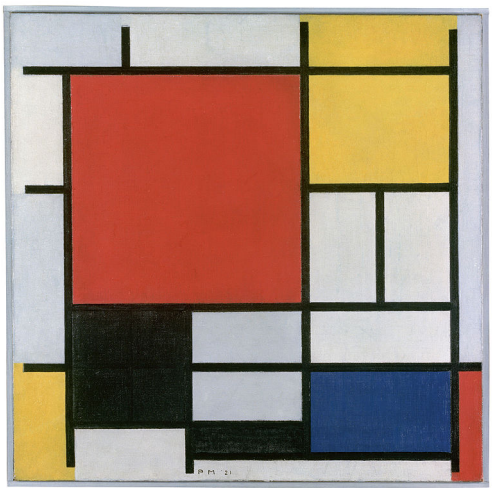

Figure 1: Composition avec grand plan rouge, jaune, noir, gris et bleu

Le but de ce projet est d'utiliser les structures vues en cours pour générer - à partir d'arbres - des images qui ne sont pas tout à fait à la manière des tableaux de Mondrian, mais s'en rapprochent. À la fin du projet vous devrez disposer d'un programme (entièrement écrit par le binôme qui le présente) pour générer des peintures suivant très précisément les consignes.

2 Projet

2.1 Arbre et images

Les arbres que vous allez implémenter et utiliser dans ce projet sont des quadtrees. Un quadtree représente le découpage d'une image. Il est construit petit à petit selon les règles données ci-dessous.
Important. Dans toute la description qui suit on suppose que l'origine des deux axes (horizontale - l'abscisse, et verticale - l'ordonnée) se trouve en bas à gauche du plan (et donc de l'image). Votre programme devra fournir

[1] `https://fr.wikipedia.org/wiki/Composition_avec_grand_plan_rouge,_jaune,_noir,_gris_et_bleu`

des images qui correspondent en tout point à cette description (sans rotation, miroir ou autre opération qui pourrait être induite par un changement d'origine).

- Au départ il y a l'image vide, c'est-à-dire un espace carré, défini par tous les points (x, y) avec $0 \leq x, y \leq n$, où n est une valeur entière qui vous sera fournie.

- Un *centre* est un point de l'image donné sous la forme $P(x, y, c_1, c_2, c_3, c_4)$, où x est son abscisse (valeur entière), y son ordonnée (valeur entière) et c_1, c_2, c_3, c_4 sont des couleurs (pas forcément toutes différentes) parmi 'R' (rouge), 'B' (bleu), 'J' (jaune), 'G' (gris clair) et 'N' (noir). Etant donnée une région rectangulaire ou carrée de l'image, notée $\mathcal{R}$, contenant le centre P, l'opération de *4-division de $\mathcal{R}$ par P* consiste à :

  - diviser $\mathcal{R}$ en quatre régions plus petites, à l'aide des deux segments passant par P, vertical et horizontal, qui relient chacun deux côtés opposés de la région $\mathcal{R}$. Les quatre régions s'appellent NO (nord-ouest, en haut à gauche par rapport au centre P), NE (nord-est, en haut à droite par rapport à P), SO (sud-ouest, en bas à gauche par rapport à P) et SE (en bas à droite par rapport à P).
  - prédéfinir des couleurs pour les quatre régions plus petites, à savoir c_1 pour NO, c_2 pour NE, c_3 pour SE et c_4 pour SO[2].

- Un nombre m de centres $P_1, P_2, \ldots, P_m$ est fourni en entrée. On suppose que toutes les abscisses des centres sont distinctes et toutes les ordonnées sont distinctes (de sorte que deux points ne sont jamais sur la même droite verticale ou la même droite horizontale). L'ordre des centres fournis est très important, puisqu'il permet de construire un arbre unique, et d'en déduire une image unique de la manière suivante.

  - Le point P_1 4-divise l'image vide en quatre régions, chacune ayant une couleur prédéfinie (comme indiqué ci-dessus). Cette opération n'est pas effectuée directement sur l'image, mais est représentée dans un premier temps par le quadtree. Il est donc initialisé. Il contient la racine (représentant l'image entière) et ses quatre fils (chacun représentant une région, dans l'ordre NO, NE, SE, SO[3] de gauche à droite). Chaque région représentée par une feuille est une *région divisible*. Les régions représentées par des noeuds internes ne sont pas divisibles.
  - Le point P_2 appartient à une et une seule des régions divisibles représentées dans le quadtree, région que nous appelons $\mathcal{Q}$. Il s'approprie cette région $\mathcal{Q}$ et la 4-divise en régions divisibles plus petites comme indiqué ci-dessus, chacune avec sa couleur prédéfinie. Donc Q n'est plus divisible et la couleur que P_1 avait prédéfinie pour $\mathcal{Q}$ n'apparaîtra pas dans l'image, puisque P_2 en a décidé autrement. De nouveau, aucune opération n'est effectuée vraiment sur l'image. La 4-division est représentée dans le quadtree: le noeud représentant la région $\mathcal{Q}$ reçoit quatre fils, un pour chaque sous-région, toujours dans l'ordre NO, NE, SE, SO de gauche à droite.
  - Le point P_3 appartient à une et une seule des régions divisibles représentées par le quadtree courant. Similairement à P_1 et P_2, il s'approprie cette région, la divise et impose ses propres couleurs aux nouvelles sous-régions. Dans le quadtree, la division de cette région par P_3 est représentée par les quatre fils, qui sont autant de régions divisibles, que l'on attribue au noeud représentant la région divisée.
  - et ainsi de suite pour tous les centres restants.

  Le quadtree résultant de ces divisions représente le découpage de l'image décidé selon les centres fournis. Chaque région divisible a sa propre couleur prédéfinie.

- Un entier supplémentaire nomme *épaisseur* et noté e est fourni en entrée. Il est impair, et représente l'épaisseur des traits qui seront dessinés sur l'image pour représenter les segments. Un trait doit empiéter

[2] attention, c'est dans l'ordre des aiguilles d'une montre !
[3] c'est le même ordre que celui des couleurs c_1, c_2, c_3, c_4

de la même manière sur chacune des régions qu'il sépare (si, par exemple, il relie les pixels (2,8) et (15,8) et l'épaisseur indiquée est 5, alors sur l'image le trait lui correspondant sera un rectangle défini par mes pixels (2,6), (2,10), (15,6) et (15,10). Les traits sont de couleur noire. **Nous supposons que les abscisses et les ordonnées des centres ont des valeurs assez éloignées pour qu'aucun centre ne soit placé sur un trait défini par un autre centre, et qu'on puisse voir - même un peu - toutes les régions. Plus précisément, pour toute paire de points** (x_1, y_1) **et** (x_2, y_2) **nous allons supposer que** $|x_1 - x_2| \geq 2e$ **et** $|y_1 - y_2| \geq 2e$**.**

Un exemple est donné en Figure 2 et Figure 3. Le quadtree ainsi obtenu et l'image lui correspondant sont appelés **quadtree** et **image de base**.

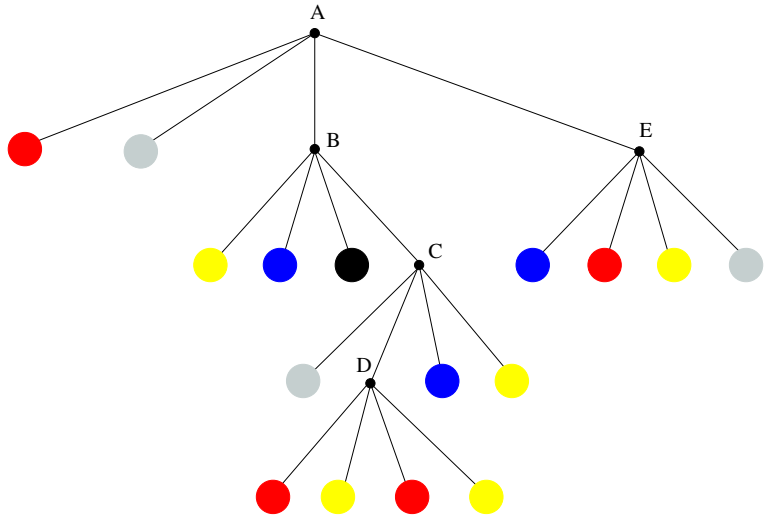

Figure 2: Arbre construit pour une image avec $n = 1000$ et la séquence de centres :
$A(600, 500,' R',' G',' J',' B')$, $B(900, 400,' J',' B',' N',' B')$, $C(800, 300,' G',' R',' B',' J')$,
$D(850, 350,' R',' J',' R',' J')$, $E(540, 120,' B',' R',' J',' G')$. Les régions divisibles sont celles représentées par les feuilles.

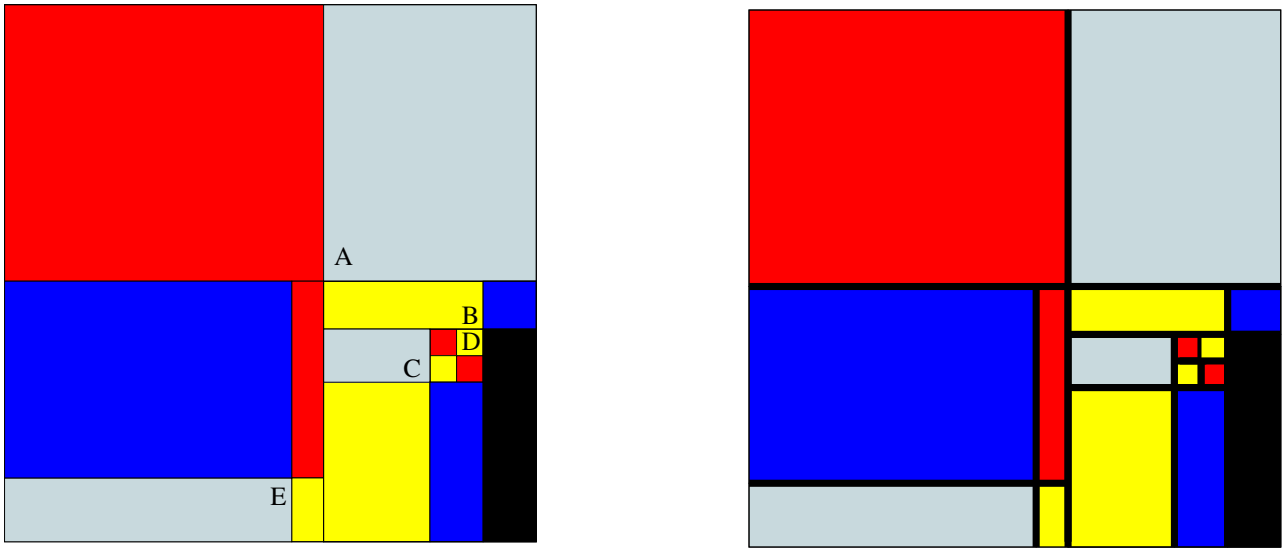

Figure 3: *Gauche:* Découpage et couleurs prédéfinies selon les informations du quadtree. *Droite:* Image colorée, avec segments d'épaisseur 19.

2.2 Amélioration de la stratégie

Pour le projet, vous devrez implémenter dans un premier temps la méthode définie ci-dessus. Cette méthode a plusieurs défauts artistiques[4]. Vous devez proposer une nouvelle méthode, qui utiliserait des arbres *ternaires*, c'est-à-dire où chaque noeud interne aurait exactement trois fils. Vous êtes libres d'inventer le critère de découpage. Si vous voulez avoir des tableaux plus jolis, vous pouvez ajouter des couleurs ou changer d'autres paramètres.

3 Fonctionnalités

3.1 Code

Votre code doit être contenu dans un seul package, celui par défaut. Il doit absolument contenir (avec ces noms) les fonctions :

- `searchQTree` qui, étant donné un centre P et le quadtree en construction, cherche dans le quadtree la région divisible $\mathcal{R}$ à laquelle P appartient (et qui doit être 4-divisée selon le centre P).
- `addQTree` qui, étant donné le résultat de la fonction précédente, modifie le quadtree pour qu'il reflète la 4-division effectuée.
- `buildQTree` qui construit le quadtree entier en utilisant les fonctions précédentes.
- `toImage` qui, à partir d'un quadtree, génère l'image au format PNG du tableau.
- `toText` qui écrit le quadtree de manière parenthésée, à l'image de ce qui a été fait dans l'Exercice 1 du TD2, dans un fichier fourni en sortie : parcours symétrique, avec - pour chaque sommet sauf les feuilles - une parenthèse avant de commencer le parcours du sous-arbre enraciné dans ce sommet et une parenthèse à la fin du parcours du sous-arbre. A la différence de l'exercice du TD, seules les couleurs des feuilles doivent être affichées (ne rien afficher pour les noeuds internes). Le fichier doit contenir une seule ligne et aucun espace.

  Exemple. Pour l'arbre de la Figure 2, la ligne écrite dans le fichier sera:

  (RG(JBN(G(RJRJ)BJ))(BRJG))
- `reColor` qui demande à l'utilisateur un point (situé dans l'une des régions divisibles, mais sans savoir laquelle) et une couleur, et modifie le quadtree pour donner à la région divisible contenant le point la couleur indiquée. Une telle paire (point, couleur) s'appelle *paire de recoloriage*.
- `compressQTree` qui, lors du changement de couleur d'une région $\mathcal{R}$ en utilisant la fonction précédente, détecte si les quatre fils du père $\mathcal{P}$ de $\mathcal{R}$ dans le quadtree représentent tous des régions divisibles de la même couleur, que l'on note C. Dans ce cas, la 4-division enregistrée dans $\mathcal{P}$ est inutile, et doit être annulée. Pour cela, les quatre fils de $\mathcal{P}$ sont effacés de l'arbre et $\mathcal{P}$ devient une feuille de couleur C.

Pour chaque fonction vous devez écrire en commentaire, juste avant la signature, sa complexité (sans la justifier). Vous choisirez également des noms clairs pour vos structures de données, par exemple `Quadtree` pour le quadtree.

Important Pour vous aider avec la gestion des images en Java, nous vous fournissons une classe `Image.java` avec laquelle vous pouvez créer une image, colorier des pixels ou des régions, et sauvegarder au format .png.

Important. Votre programme doit implémenter efficacement chaque étape de la procédure indiquée, en réalisant une *complexité au pire* minimum pour l'ensemble du traitement.

[4]Par exemple, vous pourrez constater que certains types de tableaux ne peuvent pas être construits avec la méthode fournie.

3.2 Entrées

Votre programme doit lire depuis un fichier les informations nécessaires pour la construction de l'image, selon l'énoncé. Un deuxième fichier fournira les informations nécessaires à l'exécution avec votre propre stratégie. Des explications précises sur le format de ce deuxième fichier devront être fournies dans le fichier README joint à votre projet, avec un exemple (comme nous le faisons ci-dessous pour la variante de l'énoncé).

Le fichier pour la stratégie de l'énoncé doit contenir, ligne par ligne, les informations suivantes (la partie qui commence par "//" est juste le commentaire précisant les données; elle ne doit pas apparaître dans le fichier). Ici le fichier contient les informations de l'exemple ci-dessus (Figure 2 et Figure 3), et une suite de paires de recoloriage (point,couleur) sur lesquelles il faut appliquer - paire après paire - les fonctions `reColor` et toute de suite après `compressQTree` (les résultats de ces changements de couleur sont montrés en Figure 4 et Figure 5).

1000	// ligne 1: pour l'image initiale, le côté n (en nombre de pixels). Ici $n = 1000$.
5	// ligne 2: le nombre $m \geq 0$ de centres fournis. Ici $m = 5$.
$600, 500, R, G, J, B$	// ligne 3: le 1er centre.
$900, 400, J, B, N, B$	// ligne 4: le 2ème centre.
$800, 300, G, R, B, J$	// etc.
$850, 350, R, J, R, J$	
$540, 120, B, R, J, G$	// ligne $m + 2$: le m-ème centre.
19	// ligne $m + 3$: l'épaisseur $e \geq 1$ (valeur impaire) du trait. Ici $e = 19$.
4	// ligne $m + 4$: nombre $k \geq 0$ de paires de recoloriage fournies ci-dessous, pour recoloriages successifs.
$400, 300, G$	// ligne $m + 5$: première paire, sous la forme $x, y, couleur$
$570, 250, G$	// ligne $m + 6$: deuxième paire.
$700, 10, R$	// etc.
$580, 12, G$	// ligne $m + 4 + k$: dernière paire.

Les lignes ne contiennent **aucun** espace libre entre deux caractères, ni avant le premier caractère. Des espaces libres peuvent exister après le dernier caractère lisible de chaque ligne. Après la dernière ligne lisible, il y a un dernier retour à la ligne.

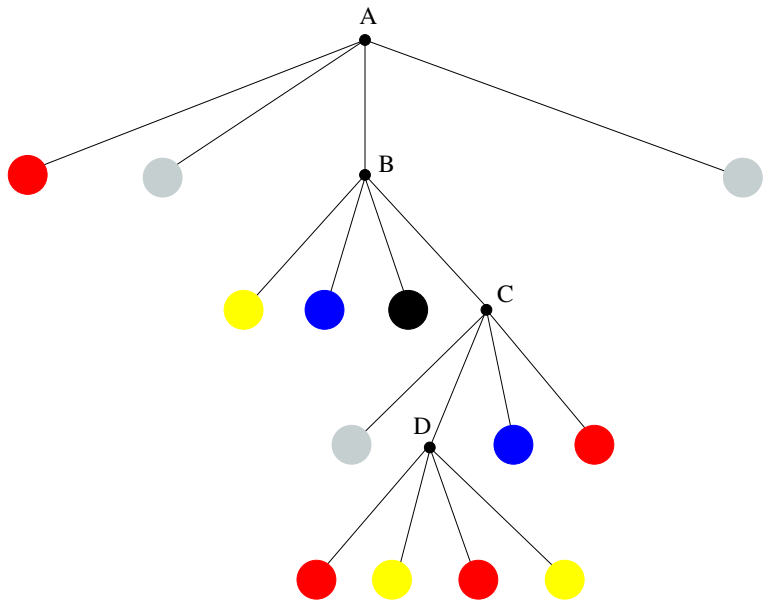

Figure 4: Arbre obtenu du précédent après recoloriage et compression du quadtree, utilisant les points et couleurs: $(400, 300,' G')$,$(570, 250,' G')$, $(700, 10,' R')$, $(580, 12,' G')$.

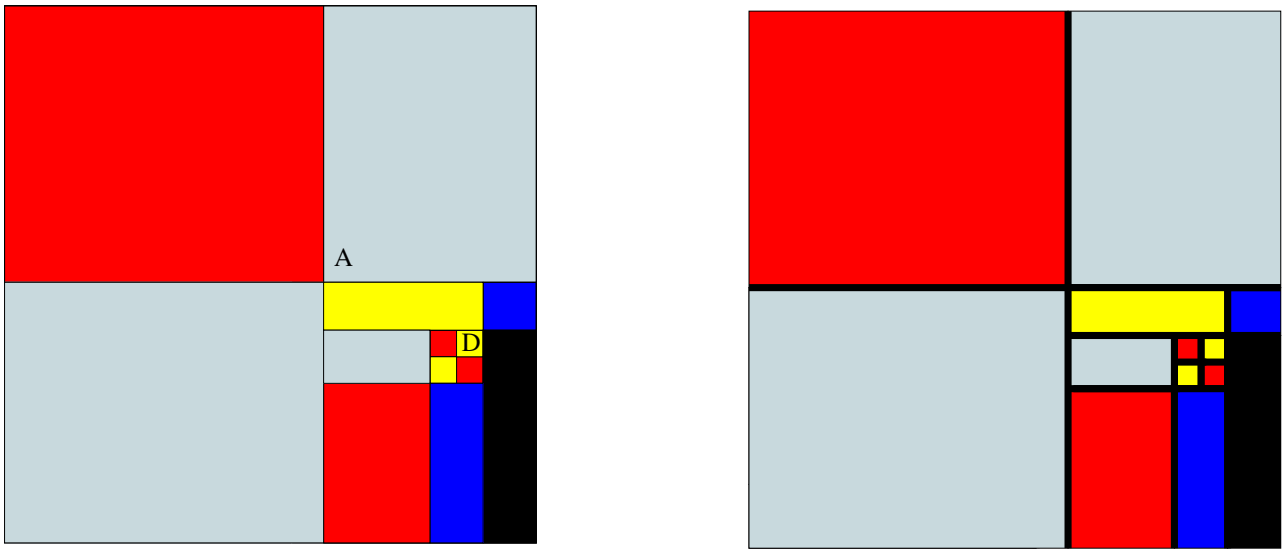

Figure 5: *Gauche:* Découpage et couleurs prédéfinies selon les informations du nouveau quadtree. *Droite:* Image colorée utilisant le nouveau quadtree.

3.3 Archive, Compilation, Exécution, Sorties

Vous déposerez une archive (un seul dépôt par binôme) de type .zip, gzip, .gz ou .tgz qui contiendra un seul répertoire Nom1Nom2 (les deux noms de famille des membres du binôme), dans lequel se trouveront: un repertoire appelé `src` contenant les sources de votre programme, le fichier README et des fichiers de tests (en entrée du programme).

- La compilation doit se faire, une fois l'archive extraite dans le répertoire courant (quel qu'il soit), avec la commande[5]:

  ```
  javac -source 1.7 -target 1.7 -d Nom1Nom2/bin Nom1Nom2/src/*.java
  ```

- L'exécution de la variante du programme dédiée à la stratégie de l'énoncé (appelée variante 1) doit se faire avec la commande:

  ```
  java -classpath Nom1Nom2/bin MonBoTablo 1 chemin_fichier_entree chemin_dossier_sortie
  ```

  Cela signifie que :

  - le nom de votre classe contenant le programme principal (`main`) doit être MonBoTablo.java

  - la variante utilisée est la stratégie de l'énoncé (variante 1)

  - le fichier en entrée est le fichier dont le répertoire et le nom de fichier, avec extension .txt, sont fournis à l'aide d'un chemin relatif depuis le répertoire courant par `chemin_fichier_entree`. Voir l'exemple ci-dessous.

  - les fichiers en sortie doivent être mis dans le répertoire dont le chemin relatif depuis le répertoire courant est donné par `chemin_dossier_sortie`. Il y a quatre fichiers en sortie:

[5]La version Java 1.7 vous refuse des fonctionnalités plus récentes qui vous permettraient d'écrire du code dont vous ne maîtrisez pas le contenu et la complexité. Vous pouvez utiliser d'autres versions de Java pour écrire votre code, mais assurez-vous régulièrement que la compilation avec la version 1.7 fonctionne. Si la version 1.7 venait à disparaître des ordinateurs du CIE (sa disparition est programmée) alors utilisez la version 1.8 pour la compilation mais veillez à **ne pas utiliser** les nouvelles fonctionnalités apportées par la version 8. Voir le 1er post de https://www.quora.com/What-is-the-difference-between-Java-1-8-and-Java-1-7 pour un résumé des différences.

- le fichier .txt contenant la description parenthésée du quadtree de base (telle que fournie par `toText`). Son nom doit être `Nom1Nom2FichierEntree_B.txt`
- le fichier .png qui contient l'image de base. Son nom doit être `Nom1Nom2FichierEntree_B.png`
- le fichier .txt contenant la description parenthésée du quadtree obtenu après tous les recoloriages successifs (telle que fournie par `toText`). Son nom doit être `Nom1Nom2FichierEntree_R.txt`
- le fichier .png qui contient l'image après recoloriages successifs. Son nom doit être `Nom1Nom2FichierEntree_R.png`

Exemple.

`java -classpath DupontMartin/bin MonBoTablo 1 ASD3/MesExemples/MesExemplesPreferes/Exemple18.txt LesResultats/MesResultatsPreferes`

va prendre en entrée le fichier Exemple18.txt du répertoire `~/ASD3/MesExemples/MesExemplesPreferes` et va produire les quatre fichiers suivants:

`~/LesResultats/MesResultatsPreferes/DupontMartinExemple18_B.txt`

`~/LesResultats/MesResultatsPreferes/DupontMartinExemple18_B.PNG`

`~/LesResultats/MesResultatsPreferes/DupontMartinExemple18_R.txt`

`~/LesResultats/MesResultatsPreferes/DupontMartinExemple18_R.PNG`

- De manière similaire, l'exécution de la variante du programme dédiée à votre propre stratégie (appelée variante 2) doit se faire avec la commande:

  `java -classpath Nom1Nom2/bin MonBoTablo 2 chemin_fichier_entree chemin_dossier_sortie`

  Vous donnerez aux fichiers en sortie des noms similaires à ceux demandés pour la variante 1, de sorte que l'on puisse identifier dans le nom du fichier les noms du binôme et le nom du fichier d'entrée.

4 Rendu de Projet

Les programmes déposés doivent fonctionner parfaitement sur les machines du CIE, sous Linux. Chaque programme doit être intégralement écrit par le binôme qui le dépose[6]. L'appel à des structures de données optimisées existantes (du genre `TreeSet` ou `HashMap`, `HashSet` etc.) est interdit : vous devez implémenter vous-mêmes vos structures de données. Seuls les tableaux (ArrayList) et les listes chaînées (LinkedList) peuvent être utilisés tels que fournis par Java. Aucun appel à une méthode mise à disposition dans les bibliothèques Java (par exemple un tri d'une liste) n'est autorisé si vous ne connaissez pas le fonctionnement de la méthode et sa complexité (ce que vous montrerez dans les calculs de complexité qui vous sont demandés).

Important

- Vous devez impérativement suivre toutes les consignes du sujet.
- L'archive du programme (voir la Section 3.3) sera déposée sur Madoc, au plus tard le **vendredi 29 novembre 2024 à 23h55** (Madoc n'acceptera pas de retard).
- La séance double du 2 décembre 2024 est consacrée à un CC de TP (sans connexion Internet) pendant lequel vous devrez répondre, toujours en binôme, à des questions concernant votre programme. Ces questions peuvent vous demander, entre autres, de compléter votre programme (identique à celui fourni en archive)

[6] Toute ressemblance - même mineure - entre deux programmes, ou entre un programme et du code sur Internet, sera considérée comme non-fortuite et entraînera automatiquement la note de 0 sur la partie concernée ainsi que sur toute autre partie utilisant, même très peu, la partie concernée. Une pénalité supplémentaire de 5 points sera également appliquée. Vous devez prendre des mesures pour vous assurer que votre code ne ressemble à aucun autre et n'est pas partagé, avec ou sans votre accord.

avec de nouvelles fonctionnalités. Pour cela, chaque binôme devra garder une copie du programme qu'il a déposé sur au moins l'un des comptes du binôme. **Seulement les réponses aux questions du CCTP seront évaluées pour la note de TP. Mais ces questions peuvent porter sur toute partie de votre programme.**

- Tout est important pour la notation. En particulier, il sera accordé beaucoup d'attention :
  - au respect des consignes
  - à la recherche d'une complexité minimum via la mise en place des structures de données les plus adaptées
  - à la clarté du code.