Structure de Données et Algorithmes 2
L2 Informatique - S4 Automne 2022–2023
Projet de développement

Coloriage d'images

Introduction

1 Modalités

Le TP est à commencer à partir du **11 octobre 2022** et à rendre **au plus tard le 13 décembre 2022 à 20h00** via la plateforme *Moodle*. Il est à réaliser **en binôme exclusivement**, durant le temps libre.

Travail à rendre

Il vous est demandé un document synthétique tenant sur **au maximum trois feuilles recto-verso, soit six pages**, où vous ferez figurer les réponses aux questions ainsi que tout élément que vous jugeriez bon de préciser. Vous déposerez sur *Moodle*, dans une archive au format `.tar.gz` :

— le document synthétique au format électronique `.pdf` ;
— les sources **dûment commentées** de votre réalisation en langage C, ce qui inclut notamment un fichier `Makefile` mais pas les fichiers objets `.o`. **Un code qui ne compile pas ne sera pas accepté** ;
— les jeux de tests (les programmes de tests et les fichiers contenant toutes les images testées).

L'ensemble des programmes ayant servi à l'élaboration du TP doit être dûment commenté. Essayez d'attacher une grande importance à la lisibilité du code : il est indispensable de respecter les conventions d'écriture (*coding style*) du langage C et de bien présenter le code (indentation, saut de ligne, etc.) ; les identificateurs (nom de types, variables, fonctions, etc.) doivent avoir des noms cohérents documentant leur rôle ; indiquez les pré-conditions, post-conditions, paramètres, la sortie et un descriptif de chaque fonction. Un commentaire se doit d'être sobre, concis et aussi précis que possible.

Pensez à utiliser des outils comme `gdb`, `valgrind` ou `gprof` pour faciliter votre développement et optimiser votre programme.

Barème indicatif

— Code (programmation et algorithmique) : 8 points
— Code (forme, commentaires) : 2 points
— Tests (évaluation de performance) : 5 points
— Rapport : 5 points

2 Problème

On se propose dans ce projet d'étudier une méthode de *coloriage automatique* d'une image en noir et blanc. Plus précisément, le problème est le suivant.

Soit I_{nb} un tableau bi-dimensionnel de taille $n \times m$, avec $n \geq 1$ et $m \geq 1$. Un élément du tableau $I_{nb}[i, j]$ sera appelé *pixel* et contiendra une information appelée *couleur*, codée sur un tableau de trois octets. Seulement deux couleurs sont possibles dans I_{nb} :

— soit $I[i, j] = [0xFF, 0xFF, 0xFF]$, et dans ce cas on dit que le pixel (i, j) est de couleur *blanche* ;
— soit $I[i, j] = [0x00, 0x00, 0x00]$, et dans ce cas on dit que le pixel (i, j) est de couleur *noire*.

L'objectif du coloriage de I_{nb} de regrouper tous les pixels blancs en régions **connexes** et de leur attribuer une couleur aléatoire (voir figure 1). Pour cela, on va générer un nouveau tableau bi-dimensionnel I_c de même taille que I_{nb}, tel que :

1. si $I_{nb}[i,j] = [0x00, 0x00, 0x00]$, alors $I_c[i,j] = [0x00, 0x00, 0x00]$ également ;
2. si $I_{nb}[i,j] = [0xFF, 0xFF, 0xFF]$, alors $I_c[i,j]$ est de couleur différente de blanche et de noire ;
3. si $I_{nb}[i,j] = [0xFF, 0xFF, 0xFF]$ et $I_{nb}[i \pm 1, j \pm 1] = [0xFF, 0xFF, 0xFF]$, alors $I_c[i,j]$ est de même couleur que $I_c[i \pm 1, j \pm 1]$;
4. il y a un nombre maximum de couleurs différentes dans l'image I_c.

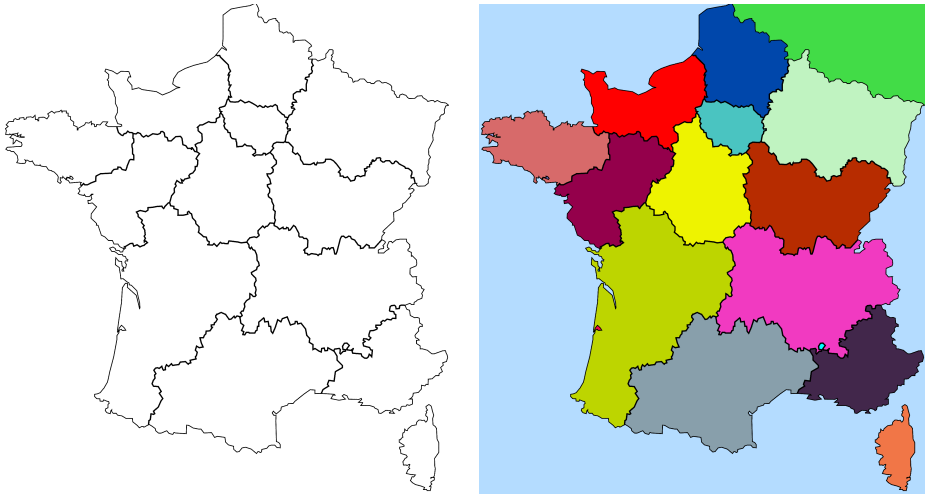

FIGURE 1 – A gauche : une image en noir et blanc. A droite : la même image coloriée.

3 Approche proposée

Afin de résoudre le problème de coloriage automatique d'une image, nous allons utiliser une approche par **ensembles disjoints** (*disjoint-set data structure* en anglais). Une structure d'ensembles disjoints est une collection $\mathcal{S} = \{S_1, \ldots, S_k\}$ d'ensembles dynamiques disjoints. Un ensemble est dit *dynamique* si son contenu évolue au cours de l'algorithme qui l'utilise : des éléments sont ajoutés, d'autres supprimés de l'ensemble. Deux ensembles sont *disjoints* si leur intersection est vide. Chaque ensemble S_i sera identifié par un de ses éléments, appelé *représentant*. Une structure d'ensembles disjoints doit typiquement posséder les trois fonctions suivantes :

— `MakeSet(`x`)` : créé un nouvel ensemble S_x, ne contenant que l'élément x. x est donc le représentant de S_x ;
— `FindSet(`x`)` : retourne un pointeur vers le représentant de l'ensemble contenant l'élément x ;
— `Union(`x`,`y`)` : créé un nouvel ensemble S_{xy}, qui contient à la fois les éléments de l'ensemble S_x dont le représentant est x et les éléments de l'ensemble S_y dont le représentant est y, et supprime les ensembles S_x et S_y. Précondition : les deux ensembles S_x et S_y doivent être disjoints. Le représentant de S_{xy} peut être n'importe quel élément de cet ensemble.

Dans notre cas, les ensembles seront des ensembles de pixels. Le problème de coloriage peut se résoudre de manière très simple avec l'approche proposée :

1. Créer un ensemble pour chaque pixel blanc de l'image en noir et blanc I_{nb}.
2. Pour chaque pixel blanc x de I_{nb} :
   (a) Pour chaque voisin blanc y de x :
      i. Si x et y ne sont pas dans le même ensemble, faire l'union des deux ensembles contenant respectivement x et y.

La figure 2 détaille le fonctionnement de cet algorithme sur un exemple simple.

4 Structures de données

L'objectif de ce projet est de coder en C l'algorithme de coloriage d'images proposé ci-dessus en utilisant deux structures de données différentes pour les ensembles, et de comparer ensuite les

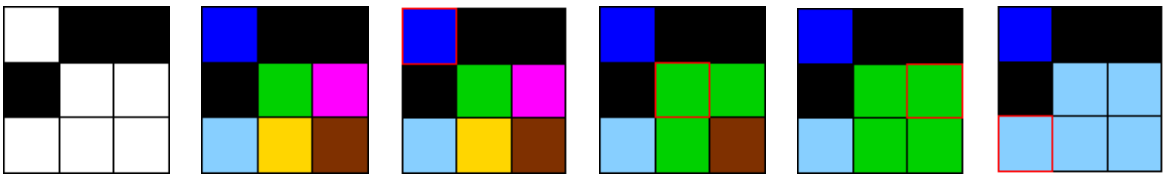

FIGURE 2 – Fonctionnement de l'algorithme de coloriage automatique sur une image de taille 3×3. (a) Image initiale I_{nb} en noir et blanc. (b) Création des ensembles initiaux (représentés par des couleurs). (c — f) Parcours des pixels blancs de l'image initiale et union des ensembles. Le pixel x étudié à chaque étape est entouré de rouge.

performances des deux structures.

La première structure est celle d'une **liste chaînée améliorée**. Plus précisément, un ensemble S_x sera implanté avec les caractéristiques suivantes (voir Figure 3) :

— chaque objet de la liste chaînée contiendra une information de pixel, un pointeur vers l'objet suivant dans la liste, et un pointeur vers le représentant de l'ensemble ;
— le premier objet de la liste servira de représentant ;
— la liste sera définie avec deux champs : un pointeur `head` vers le premier objet de la liste (i.e. le représentant de l'ensemble), et un pointeur `tail` vers le dernier objet de la liste.

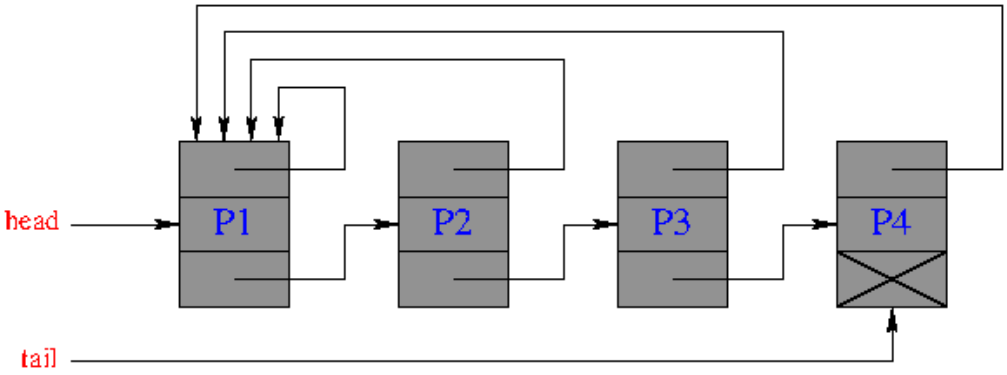

FIGURE 3 – Implantation d'un ensemble sous forme de liste chaînée améliorée.

La seconde structure pouvant implanter un ensemble est celle d'un **arbre**. La racine d'un arbre contiendra le représentant de l'ensemble. Chaque nœud de l'arbre contiendra une information de pixel et un pointeur vers son nœud père, mais pas vers le représentant de l'ensemble, contrairement à la structure de liste précédente. Nous n'avons pas non plus besoin de pointeurs vers les fils : l'arbre sera orienté dans le sens fils $\rightarrow$ père et non père $\rightarrow$ fils comme habituellement (voir Figure 4). Afin d'avoir un algorithme performant d'union de deux arbres, chaque nœud contiendra également un entier appelé **rang**, qui sera une borne supérieure sur la hauteur du nœud dans l'arbre. La *hauteur* d'un nœud correspond au nombre maximum d'arêtes père-fils dans l'arbre entre le nœud et ses descendants. Une feuille est donc de hauteur zéro, et la racine possède la hauteur maximale dans l'arbre.

5 Tests - Formats *PBM* et *PPM*

Cinq images vous sont fournies pour tester la validité de vos codes. Vous pouvez (devez ?) bien sûr utiliser aussi d'autres images. Les images fournies (toutes *courtoisie* Wikimedia) sont au format *PBM* (*Portable Bitmap File Format*) ASCII. Ce format est très simple :

— la première ligne stocke le "nombre magique" `P1` ;
— la deuxième ligne stocke la largeur m puis la hauteur n de l'image (en nombre de pixels) ;
— les n lignes suivantes stockent chacune m entiers : 1 lorsque le pixel correspondant est noir, 0 lorsque le pixel est blanc *(attention, convention inverse de notre énoncé)* ;
— aucune ligne ne peut faire plus de 70 caractères : une ligne trop longue est scindée en plusieurs lignes séparées par un saut de ligne.

Les images résultats devront être au format *PPM* (*Portable Pixmap File Format*) ASCII. Ce format est similaire au format *PBM* ASCII, excepté que le "nombre magique" vaut *P*3, que sur la ligne suivant m et n est stockée la valeur maximale utilisée pour stocker les couleurs (255 dans

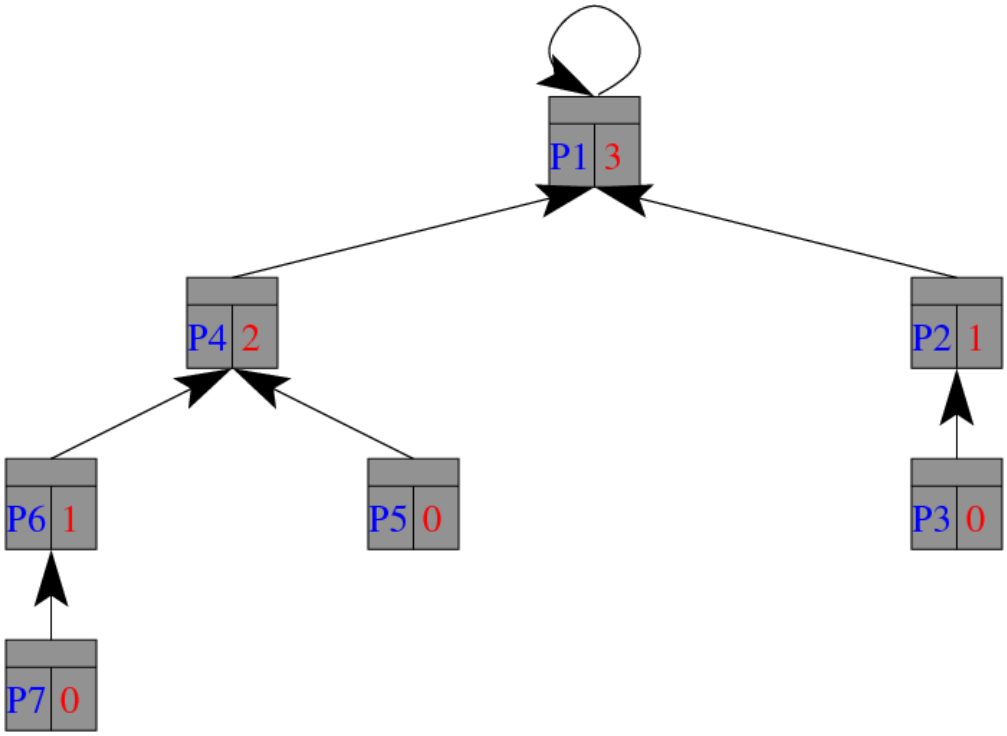

FIGURE 4 – Implantation d'un ensemble sous forme d'arbre.

notre cas), et que les trois composantes d'une couleur (dans notre cas, les trois cases du tableau) sont stockées sous forme d'entiers les unes après les autres, séparées par un espace.

6 Questions

6.1 Lecture/écriture de données

Question 1 Codez une fonction `Read()` qui lit une image au format *PBM* ASCII (voir Section 5) et stocke ses pixels dans un tableau bidimensionnel. La fonction doit traiter comme un cas d'erreur le fait que le fichier n'est pas conforme au format tel que décrit Section 5.

Question 2 Codez une fonction `Write()` qui, à partir d'un tableau bidimensionnel de pixels de couleur, génère une image au format *PPM* ASCII (voir Section 5). La fonction doit traiter comme un cas d'erreur le fait que les structures de données ne permettent pas la génération d'une image conforme au format tel que décrit Section 5.

Question 3 Codez une fonction `Generate()` qui, à partir de deux entiers n et m quelconques, génère une image aléatoire en noir et blanc de taille $n \times m$. Vous pouvez, si vous le souhaitez, raffiner cette fonction pour qu'elle prenne en entrée plus de paramètres.

6.2 Implantation par listes chaînées améliorées

Question 4 Codez les fonctions `MakeSet()` et `FindSet()` de création et recherche d'ensembles implantés sous forme de listes chaînées telles que décrites en Section 4. Testez votre solution en précisant quels tests ont été réalisés.

Question 5 Donnez, en les justifiant, la complexité asymptotique en pire cas et le coût en mémoire des fonctions `MakeSet()` et `FindSet()`.

Question 6 Codez la fonction `Union()` de manière à ce qu'elle soit efficace. On pourra pour cela légèrement modifier la structure de données en y ajoutant un unique entier bien choisi.

Question 7 Donnez, en la justifiant, la complexité asymptotique en pire cas d'une séquence de n créations d'ensembles $S_1, \ldots, S_n$ suivie des unions successives de ces ensembles en un seul ensemble final.

Question 8 Codez l'algorithme de coloriage automatique décrit en Section 3, pour une image au format PBM ASCII. Cet algorithme doit générer une image au format PPM ASCII, et utiliser les fonctions `Read()`, `Write()`, `MakeSet()`, `FindSet()` et `Union()` précédemment codées. Testez votre solution en précisant quels tests ont eté réalisés.

Question 9 Donnez, en les justifiant, la complexité asymptotique en pire cas et le coût en mémoire de l'algorithme.

Question 10 Utilisez la fonction `Generate()` pour tester votre algorithme sur des images aléatoires de tailles croissantes, et comparez à l'aide d'un graphique la complexité asymptotique en pire cas théorique et le coût réel en temps de votre algorithme.

6.3 Implantation par arbres

Question 11 Codez les fonctions `MakeSet()` et `FindSet()` de création et recherche d'ensembles implantés sous forme d'arbres tels que décrits en Section 4. Testez votre solution en précisant quels tests ont été réalisés.

Question 12 Donnez, en les justifiant, la complexité asymptotique en pire cas et le coût en mémoire des fonctions `MakeSet()` et `FindSet()`.

Question 13 Codez la fonction `Union()` de manière à ce qu'elle soit efficace. Pour cela, pensez à utiliser et mettre à jour le rang de nœuds bien choisis.

Question 14 Donnez, en la justifiant, la complexité asymptotique en pire cas d'une séquence de n créations d'ensembles $S_1, \ldots, S_n$ suivie des unions successives de ces ensembles en un seul ensemble final.

Question 15 Codez l'algorithme de coloriage automatique décrit en Section 3, pour une image au format PBM ASCII. Cet algorithme doit générer une image au format PPM ASCII, et utiliser les fonctions `Read()`, `Write()`, `MakeSet()`, `FindSet()` et `Union()` précédemment codées. Testez votre solution en précisant quels tests ont eté réalisés.

Question 16 Donnez, en les justifiant, la complexité asymptotique en pire cas et le coût en mémoire de l'algorithme.

Question 17 Utilisez la fonction `Generate()` pour tester votre algorithme sur des images aléatoires de tailles croissantes, et comparez à l'aide d'un graphique la complexité asymptotique en pire cas théorique et le coût réel en temps de votre algorithme.

Question 18 Comparez les performances des deux structures de données.