Rapport TP algorithmique et programmation

Coloriage d’images

# Réponses

## Question 1

Pour vérifier le format du fichier, nous pouvons vérifier que la première ligne correspond bien au « nombre magique » P1. Ensuite nous vérifions que chaque caractère de chaque ligne du contenu du fichier soit un 0 ou un 1 ou un caractère de séparation, si ce n’est pas le cas on lève une exception. On lève aussi une exception si la taille de fichier indiquée en début de fichier ne correspond pas au nombre de caractères lus.

## Question 2

Le cas d’erreur à traiter est une erreur lors de la création ou l’écrasement du fichier.

Pour tester cette fonction nous avons implémenté une fonction testEcriture() dans testWriteFic.cpp. Cette fonction crée un tableau de pixel de quatre cases et un objet WriteFic avec le nombre de ligne du futur fichier à 2 et le nombre de colonne à 2. On lance la fonction writeThePPMFic qui transcrit ce tableau dans un fichier ppm avec une couleur par pixel qui sera nommé result.ppm. Il n’y a qu’une couleur par pixel car ceci n’appartient pas à un ensemble. Pour vérifier les résultats on a juste à regarder le fichier result.ppm qui doit être constitué de la manière suivante : deux ligne de chacune un pixel qui ont chacun une couleur différente.

## Question 3

Pour tester la fonction generate, nous avons fait la fonction testGenerate qui exécute le generate puis lis le fichier généré pour vérifier si celui-ci est conforme.

## Question 4

Pour makeSet(x) qui correspond au constructeur de la classe « Ensemble » fut testé ainsi : on construit un ensemble et ensuite on vérifie que le head de l’ensemble correspond bien au pixel qu’on a passé en paramètre.

Le findset(x) correspond à la fonction getHead() de la classe Pixel. Pour la tester on fait un makeset(x) puis on vérifie que findset(x) renvoi bien x.

## Question 5

MakeSet(x) correspond au constructeur de la classe « Ensemble ». La complexité asymptotique relative au nombre d’affectation est de 3 ce qui correspond à Θ(1). Le coût en mémoire est de 3 car il y a trois affectations.

FindSet(x) correspond à la fonction getHead() de la classe Pixel. Sa complexité asymptotique relative au nombre d’affectation est de Θ(1) car il n’y a qu’un « return ». Son coût en mémoire est de zéro.

## Question 6

Soit X et Y deux ensemble avec X le plus grand des deux ensembles. On commence par faire pointer tous les pixels de y vers le représentant de X. C’est-à-dire que chaque pixel de Y aura pour attribut « \_head » un pointeur vers le « \_head » de X, qui est le représentant de l’ensemble X.

Ensuite on stocke la taille du « nouvel » ensemble. On rattache la tête de y à la queue de x. La queue de x devient la queue de y.

Enfin on détruit le contenant y qui ne sert plus, le contenant x étant devenu l’ensemble xy.

Dans le pire des cas, n’étant la taille de y la complexité asymptotique dans le pire des cas est de O(n). Le coût en mémoire correspondrait à la taille des deux ensembles.

## Question 7

On définit |S| comme étant le nombre d’élément de l’ensemble S.

La création d’un ensemble de pixel S1 a une complexité asymptotique (en pire cas) de |S1|. Et l’union de S1 et S2 (avec |S1| >= |S2|) a une complexité asymptotique en pire cas de |S2|.

En supposant que les ensembles créés ici contiennent un nombre **m** de pixels, la complexité asymptotique en pire cas de la création de **n** ensembles contenant **m** pixels est

(**n** -1) **m**

## Question 8

## Question 11

## Question 12

Commençons par le coût mémoire :

Lors de la lecture du fichier, l’objet *LecteurImage* alloue n\*m\*16 octets. n et m étant la hauteur et la largeur de l’image. 16 octets représente la taille d’un objet *Pixel*.

Exemple : Pour une image de 2567 de largeur et 1091 de hauteur. Le cout mémoire sera de : 44,8Mo

Complexité asymptotique en pire cas :

* De l’algorithme d’union des ensembles

On applique au n pixels la fonction d’union de deux ensembles addEnsemble\_inTail