Rapport SDA-2

Sultanli Aykhan, Hasanli Ayhan

Question 1:

Cette fonction lit un fichier PBM (Portable BitMap) et enregistre son contenu dans un tableau à deux dimensions de pixels. Les fichiers PBM sont un type de fichier image qui utilise un format de texte simple pour stocker des images en noir et blanc.

La fonction commence par ouvrir le fichier spécifié par le paramètre ‘fichier’ à l'aide de la fonction ‘fopen’. Elle lit ensuite les trois premiers caractères du fichier, qui devraient être "P1" pour indiquer que le fichier est au format PBM. Si le fichier n'est pas au format PBM, la fonction affiche un message d'erreur et renvoie ‘NULL’.

La fonction lit ensuite la largeur et la hauteur de l'image dans le fichier et les stocke dans les variables ‘largeur’ et ‘hauteur’, respectivement. Ces valeurs sont également stockées dans les variables ‘l’ et ‘h’, qui sont passées en tant que pointeurs.

Ensuite, la fonction alloue de la mémoire pour un tableau à deux dimensions de pixels à l'aide de la fonction ‘malloc’. Le tableau a hauteur lignes et largeur colonnes.

La fonction lit ensuite les données de pixel du fichier, un pixel à la fois. Chaque pixel est représenté par un seul caractère dans le fichier, qui peut être soit un "0" soit un "1". La fonction convertit ce caractère en pixel en créant une structure ‘Pixel’ avec les valeurs de couleur correspondantes. Le pixel résultant est alors ajouté au tableau à deux dimensions de pixels. Enfin, la fonction ferme le fichier et renvoie le tableau à deux dimensions de pixels.

Question 2 :

Cette fonction écrit un tableau à deux dimensions de pixels dans un fichier PPM (Portable PixMap). Les fichiers PPM sont un type de fichier image qui utilise un format de texte simple pour stocker des images couleur.

La fonction commence par ouvrir le fichier spécifié par le paramètre ‘fichier’ à l'aide de la fonction ‘fopen’. Si le fichier ne peut pas être ouvert, la fonction affiche un message d'erreur et retourne.

La fonction écrit les trois premiers caractères "P3" dans le fichier pour indiquer que le fichier est au format PPM. Elle écrit également la largeur et la hauteur de l'image ainsi que la valeur maximale de couleur (255) dans le fichier.

Ensuite, la fonction parcourt le tableau à deux dimensions de pixels et écrit chaque pixel dans le fichier. Chaque pixel est représenté par trois valeurs entières séparées par des espaces, correspondant aux valeurs de rouge, de vert et de bleu du pixel. La fonction limite chaque ligne de pixels à 70 caractères en ajoutant des retours à la ligne lorsque nécessaire.

Finalement, la fonction ferme le fichier.

Question 3 :

Cette fonction crée un tableau à deux dimensions de pixels, qui ont une largeur et une hauteur spécifiées en entrée. Elle utilise la fonction malloc pour allouer de l'espace mémoire pour chaque ligne du tableau. Ensuite, elle remplit chaque pixel du tableau avec une couleur aléatoire en utilisant la fonction rand. La couleur est définie comme étant la même pour les composantes rouge, verte et bleue du pixel. Enfin, la fonction renvoie le tableau de pixels rempli.

Question 4 :

La fonction MakeSet crée un nouveau nœud (un élément) dans un ensemble d'ensembles disjoints. Elle alloue de l'espace mémoire pour ce nœud et l'initialise avec certaines valeurs, comme ses coordonnées x et y, sa taille (qui est initialisée à 1 car il s'agit d'un nœud unique) et son représentant (qui est lui-même au début). Ensuite, la fonction renvoie ce nouveau nœud.

La fonction FindSet renvoie le représentant d'un nœud donné dans un ensemble d'ensembles disjoints. Cela signifie qu'elle renvoie le nœud qui représente tout l'ensemble auquel appartient le nœud donné.

Question 5 :

La complexité asymptotique en pire cas de la fonction MakeSet est O(1), car elle effectue un nombre constant d'opérations indépendamment de la taille de l'ensemble d'ensembles disjoints.

Le coût en mémoire de la fonction MakeSet est également O(1), car elle alloue de l'espace mémoire pour un seul nœud à chaque fois qu'elle est appelée.

La Complexite de la fonction FindSet est O(n*m) (n = largeur de l'image, m = hauteur de l'image).

Question 6 :

La fonction Union prend en entrée deux nœuds (node1 et node2) et les fusionne dans un ensemble d'ensembles disjoints. Elle commence par trouver les représentants de ces deux nœuds en appelant la fonction FindSet sur chacun d'entre eux. Si les deux nœuds ont déjà le même représentant, cela signifie qu'ils sont déjà dans le même ensemble et il n'y a rien à faire. Sinon, la fonction ajoute la liste chaînée la plus petite à la plus grande en mettant à jour les champs de données des nœuds (taille, représentant, etc.) et en enlevant la liste chaînée de la liste des listes chaînées. La fonction ne renvoie rien, car elle modifie directement les nœuds et les ensembles d'ensembles disjoints.

Question 7 :

- Complexite de la fonction Union: O(min(n,m)) (n = taille de la liste chaînée de node1, m = taille de la liste chaînée de node2)

- Complexite de n creation d'ensembles: O(n)

- Complexite de n unions: O(n^2)

- Complexite totale: O(n^2)

Question 8 :

La fonction Colorie lit un fichier d'image et colore chaque pixel de l'image en fonction de sa proximité avec d'autres pixels de la même couleur. Pour ce faire, elle parcourt chaque pixel de l'image et vérifie si ses pixels voisins ont la même couleur. Si c'est le cas, elle utilise la fonction Union pour fusionner ces pixels dans un même ensemble d'ensembles disjoints. Elle utilise également la fonction MakeSet pour créer de nouveaux nœuds pour les pixels qui n'ont pas encore de représentant dans l'ensemble d'ensembles disjoints. Enfin, la fonction modifie l'image en remplissant chaque pixel de la couleur de son représentant dans l'ensemble d'ensembles disjoints.

Question 9 :

La complexité asymptotique en pire cas de cet algorithme est O(n^2), où n est le nombre de pixels dans l'image. Cela est dû au fait que l'algorithme parcourt chaque pixel de l'image et vérifie ses pixels voisins, ce qui prend du temps proportionnel au nombre de pixels dans l'image.

Le coût en mémoire de cet algorithme est également O(n^2), car il stocke chaque pixel de l'image dans un tableau à deux dimensions de pixels. De plus, il stocke chaque nœud dans un tableau de nœuds, ce qui prend également de l'espace mémoire proportionnel au nombre de pixels dans l'image. Enfin, il utilise la fonction malloc pour allouer de l'espace mémoire pour chaque nœud créé, ce qui ajoute également au coût en mémoire de l'algorithme.

Question 11 :

La fonction ‘MakeSet’ crée et initialise un nouveau noeud de l'arbre d'ensemble de disjoint. Cet arbre est utilisé pour implémenter un ensemble de disjoint, qui est un type de structure de données utilisé pour maintenir un ensemble de sous-ensembles disjoints. Le noeud est créé en allouant de la mémoire pour le noeud à l'aide de ‘malloc’ et en initialisant ses champs de données.

La fonction ‘FindSet’ trouve la racine de l'arbre d'ensemble de disjoint auquel appartient le noeud donné en parcourant les parents du noeud jusqu'à ce qu'on atteigne la racine. Cette opération prend environ O(log n) temps, car l'arbre d'ensemble de disjoint est généralement construit de manière à ce que sa hauteur soit logarithmique en fonction du nombre total de noeuds.

Question 12 :

La complexité asymptotique en pire cas de la fonction ‘MakeSet’ est O(1), car elle effectue un nombre constant d'opérations indépendamment du nombre de noeuds dans l'arbre d'ensemble de disjoint. En particulier, elle alloue de la mémoire pour un seul noeud, initialise ses champs de données et retourne le pointeur vers le noeud créé.

Le coût en mémoire de la fonction ‘MakeSet’ dépend de la taille de la structure Node et de la quantité de mémoire allouée par ‘malloc’. En général, on peut considérer que le coût en mémoire de ‘MakeSet’ est O(1), car il n'y a pas de dépendance entre la taille de la structure ‘Node’ et le nombre total de noeuds dans l'arbre d'ensemble de disjoint.

La complexité asymptotique en pire cas de la fonction ‘FindSet’ est O(log n), car elle parcourt l'arbre d'ensemble de disjoint jusqu'à ce qu'elle atteigne la racine. En général, l'arbre d'ensemble de disjoint est construit de manière à ce que sa hauteur soit logarithmique en fonction du nombre total de noeuds, de sorte que la fonction ‘FindSet’ prendra environ O(log n) temps pour trouver la racine de l'arbre.

Le coût en mémoire de la fonction ‘FindSet’ est O(1), car elle ne nécessite pas d'allouer de nouvelle mémoire ou de stocker des données supplémentaires. Elle parcourt simplement l'arbre d'ensemble de disjoint en suivant les pointeurs de parenté, ce qui ne nécessite pas de mémoire supplémentaire.

Question 13 :

La fonction ‘Union’ prend en entrée deux noeuds de l'arbre d'ensemble de disjoint et fusionne leurs ensembles en ajoutant l'un de ces ensembles à l'autre. Pour ce faire, la fonction utilise la fonction ‘FindSet’ pour trouver les racines des arbres auxquels appartiennent les noeuds donnés. Si les noeuds sont déjà dans le même arbre, la fonction ne fait rien et retourne simplement. Sinon, elle ajoute l'arbre de l'un des noeuds à l'autre en définissant la racine de l'arbre ajouté comme le parent de la racine de l'arbre cible. La fonction met également à jour le rang des noeuds de l'arbre ajouté en leur donnant une valeur supérieure à celle de leur parent. Enfin, la fonction retire la racine de l'arbre ajouté de la liste des racines d'arbres distincts.

Question 14 :

La complexité asymptotique en pire cas d'une séquence de n créations d'ensembles suivie des unions successives de ces ensembles en un seul ensemble final est de l'ordre de O(n * log n). Cela signifie que le temps d'exécution de cette séquence d'opérations augmentera de manière

approximativement linéaire avec le nombre d'ensembles créés, mais avec un coefficient multiplicatif logarithmique en fonction du nombre total d'ensembles. En d'autres termes, si le nombre d'ensembles augmente de manière significative, le temps d'exécution augmentera moins rapidement que si le coefficient multiplicatif était constant. Cette complexité découle du fait que chaque opération de fusion prend environ O(log n) temps et qu'il y a environ n opérations de fusion pour fusionner n ensembles distincts en un seul ensemble final. Si les arbres d'ensemble de disjoint sont très déséquilibrés, la complexité peut être encore plus grande.

Question 15 :

La fonction ‘Colorie’ lit un fichier image et colorie chaque pixel en fonction de ses voisins. Pour ce faire, elle parcourt chaque pixel de l'image et crée un nouvel ensemble de disjoint pour chaque pixel de couleur non nulle. Si un pixel a un voisin de la même couleur, la fonction utilise la fonction ‘Union’ pour fusionner les ensembles de disjoint correspondants. La fonction parcourt chaque pixel de l'image et vérifie ses voisins immédiats (haut, bas, gauche, droite, diagonaux) pour voir s'ils ont la même couleur. Si c'est le cas, elle crée un nouveau noeud pour le pixel voisin s'il n'en a pas déjà un et utilise la fonction Union pour fusionner les ensembles de disjoint correspondants.

Question 16 :

La complexité asymptotique en pire cas de cette fonction est O(n^2), où n est le nombre de pixels dans l'image. Cela découle du fait que la fonction parcourt chaque pixel de l'image et vérifie ses voisins, ce qui prend environ O(n) temps. Si un pixel a un voisin de la même couleur, la fonction utilise la fonction ‘Union’, qui a une complexité asymptotique en pire cas de O(log n). Comme il y a environ n pixels dans l'image, la complexité totale de la fonction sera de l'ordre de O(n * log n). Cependant, comme la fonction parcourt chaque pixel de l'image et vérifie ses voisins, la complexité totale sera en réalité de l'ordre de O(n^2), car la fonction ‘Union’ sera appelée environ n fois. Cela signifie que le temps d'exécution de la fonction ‘Colorie’ augmentera de manière quadratique avec le nombre de pixels dans l'image.

Il est important de noter que cette complexité asymptotique en pire cas ne tient pas compte de la complexité de la fonction ‘Read’, qui lit le fichier image et crée un tableau de pixels à partir de celui-ci. Si la fonction ‘Read’ a une complexité supérieure à O(n), alors la complexité totale de la fonction ‘Colorie’ sera également supérieure à O(n^2).