

Rapport Flood-It

SD

L2 DANT

LAYAIDA RADJA

RIZWAN SALIME

Ce projet a été fait en utilisant l’aide mis en ligne pour coder sur windows, plus précisément, nous avons utilisé Mingw et installer la bibliothèque SDL grâce aux nombreux tutoriels. Plusieurs tests ont été effectué en fonction de la dimension et de la difficulté du jeu pour permettre de calculer le temps d’exécution de chaque fonction et ainsi comparer leur rapidité. Ils sont d’abord testé avec le main du fichier Flood-It\_Partie1 et puis testé sur leur temps d’exécution avec le fichier temps.c.

Partie A

Exo 1

1.3- En insérant les deux fonctions et en effectuant différents jeux de test, on constate que le temps d’exécution dépend de la difficulté et de la dimension. Plus la taille est petite, plus il y aura moins de couleur, le jeu se résoudra plus vite et le temps d’exécution sera plus rapide. Il en est de même pour la difficulté. Si on augmente la difficulté, plus de tests seront nécessaire pour trouver la bonne couleur, le temps d’exécution sera lent.

Exo 2

2.2- Afin que l’utilisateur puisse choisir entre la version récursive et la version impérative, on a mis un scanf dans le int main, pour que l’utilisateur puisse choisir la fonction. Effectuant de nombreux jeux de test, on trouve que la fonction impérative est plus rapide que la fonction récursive.

Exo 3

3.4- Après plusieurs jeux de tests, on trouve que la version acyclique est plus rapide que la version impérative. En effet, dans les fonctions précédentes la recherche d’une case dans une zone avait une complexité de O(n). Dans cette version acyclique, la recherche se fait en O(1).

3.5- En comparant les courbes obtenues des différents jeux de test, on peut constater que en fonction de la taille et de la difficulté, la version acyclique est la plus rapide par rapport à la version impérative, qui elle-même plus rapide que la version récursive.

Comme une erreur s’était glissée dans liste\_case.c, on n’avait pas réussi à tracer les graphes. Après correction de l’erreur, on peut voir sur les graphes que le temps d’exécution est presque la même pour la fonction récursive et la fonction impérative, peut-être que la fonction récursive est légèrement plus rapide.

Partie B

4- La fonction créer\_graphe\_zone permet de créer le graphe tout entier. Pour créer le graphe, on doit initialiser un sommet vide et déterminer la zone de même couleur correspondante. Pour cela, on va parcourir toutes les cases et si on trouve que la case mat[i][j] n’est associé à aucun sommet, on initialise un sommet vide puis on trouve la zone de même couleur que cette case grâce à la fonction trouver\_zone\_rec. Pour créer les voisins, on va créer le sommet temporaire s2 et on va parcourir les deux sommets s1 et s2 en même temps. Si les sommets sont différents et qu'ils ne sont pas déjà adjacents, on l’ajoute avec la fonction ajoute\_voisin.

5- La stratégie de maxBordure sera de toujours choisir la couleur qui est la plus représentée dans la bordure-graphe. Pour cela, on va utiliser une fonction intermédiaire : la fonction mise\_a\_jour\_bg. A partir de la zone supérieure gauche, on parcourt ses sommets adjacents et on cherche la couleur la plus présente. Quand on trouve la couleur la plus présente, on fait appel à la fonction nouvelle\_couleur qui elle va changer la couleur de la Zsg et parcourir toutes les cases de la Zsg et leur donner une nouvelle couleur. Puis on va supprimer toutes les zones de la bordure et ajouter à la zsg les cases des zones supprimées et leurs adjacences. Finalement dans maxBordure on aura plus qu’à créer le graphe entier à l’aide de l’appel de graphe\_zone et puis faire la mise à jour avec la fonction mise\_a\_jour\_bg qu’on vient d’écrire.