Rapport Algorithmique

Partie 2 : Mastermind

LANUEL Charlotte

SCHWAB Lucas

M1 Informatique

Tout le travail fourni pour ce projet a été réalisé en binôme. Le rapport a été écrit par Charlotte LANUEL.

# Partie 1 : Questions préalables

9) Une combinaison secrète est un arrangement (on tire une couleur et l’ordre a de l’importance, il n’y a pas de remise) de *n* éléments parmi distincts : donc .

10) b est la variable comprise entre 0 et et qui représente le nombre de couleurs bien choisie et bien placées. On ne sait pas quelles sont les couleurs bien placées et quelles sont les couleurs à changer, donc l’ordre n’a pas d’importance ici. On aura donc une combinaison de *(n-b)* couleurs parmi *(k-b)*: .

11) Ici, on a proposé un arrangement de n couleurs parmi k dont est ressorti le couple , b représentant les couleurs bien choisies et bien placées et m représentant les couleurs bien choisies mais mal placées.

Le nombre de combinaisons possible est donc :

* le nombre de combinaisons possible de *b* couleurs parmi *n :* cas où on cherche le nombre de combinaisons secrètes faisable les couleurs qui sont bien choisies :
* multiplié par le nombre de combinaisons possible de *(m)* couleurs parmi *(n-b) :* cas où on cherche le nombre de combinaisons possibles avec les couleurs bien choisies mais mal placées. Il se peut que la combinaison choisie soit encore une fois mal placée. Cependant, l’ordre a plus que tout son importance pour les couleurs qui sont justement mal placées donc ce résultat sera mis en produit avec le nombre d’arrangement de m couleurs parmi les *n-b* restantes qu’on a choisi :
* multiplié par le nombre d’arrangement de *(n-b-m)* couleurs parmi *(k-b-m) donc le nombre d’arrangement de couleurs qu’on a pas encore choisi et qui doivent figurer dans l’arrangement final : elles remplacent possiblement celles qui sont fausse : .*

Le nombre de combinaisons possibles compatibles avec la réponse *(b,m)* est donc majoré par (inférieur ou égal à):

12) Il n’y a pas forcément égalité avec la formule ci-dessus car le joueur 2 peut prendre en compte les combinaisons proposées avant celle-ci. En se servant des résultats précédents, il peut grandement réduire le nombre de combinaisons disponibles à essayer.

# Partie 2 : Programmation dynamique

13) On prend en compte pour chaque couleur sélectionnée si celle juste avant a été bien placée, mal placée, ou absente.

La récurrence basée sur la majoration précédente est :

Si *b = 0* alors la majoration sera :

Si *m = 0* alors :

Sinon :

14) Pour éviter d’avoir à recalculer les combinaisons à chaque fois, on instanciera un tableau à double entrée tab avec les résultats des calculs au fur et à mesure.

Algo

Pour b de 0 à n faire

Pour m de 0 à n-b faire

Si b = 0 alors

Tab[b][m] =

Sinon si m =0 alors

Tab[b][m] =

Sinon

Tab[b][m] = Tab[b-1][m] - Tab[b][m-1] +

FSi

Fpour

Fpour

Fin

15) Pour cette question, l’algorithme a été implémenté dans des classes séparées des parcours de graphe. Cependant, bien que l’algorithme ait été débuggé plusieurs fois, il en ressort une valeur bien au-dessus du majorant : pour Tab[1][2] qui est la valeur qu’on recherche, son majorant est 144. La valeur retournée est 448. Il nous a été impossible de trouver une valeur qui soit approchée du majorant sans fausser tout le problème.

16) La complexité au pire des cas est O(n(n-b)) (on comptabilise deux boucles et le reste des opérations effectuées ne requièrent pas d’opération qui se fait plus d’une fois).

# Partie 3 : Programmation gloutonne

17) Il n’est pas possible de raisonner comme en programmation dynamique. Dans les précédentes questions, nous avons calculé les scores de proche en proche en se servant du score de la couleur placée juste avant, et ainsi de suite. Ici, nous avons besoin du score dans sa globalité, ainsi que le score des combinaisons précédentes. Il ne sera pas possible de faire une programmation dynamique sur un historique multi-facteur complet.

Cette partie du projet a été réalisée intégralement par Charlotte LANUEL.