F L O W

Revision to consider : 73

TABLE DES MATIÈRES

TABLE DES FIGURES

INTRODUCTION

Flow est originellement un jeu de reflexion. Le principe consiste à remplir une grille dans sa totalité en reliant les points de même couleur.

On comprend déjà que notre algorithme de résolution va chercher à trouver des parcours qui vont à la fois chercher à relier les couleurs correspondantes et couvrir le plus d'espace possible sans se géner entre eux.

Nous allons donc poser dans un premier temps les problèmes à résoudre ainsi que le vocabulaire que nous allons employer pour caractériser le jeu. Nous verrons ensuite comment nous avons implémenté les types abstraits de données qui apparaissent dans les problèmes évoqués ci-dessus puis nous terminerons par présenter l'algorithme de résolution et l'ensemble des tests réalisés pour vérifier sa validité.

Capacité du programme réalisé

Notre programme permet d'afficher, de jouer et de résoudre une grille carrée donnée en paramètre. La résolution ne gère pas les grilles comportant des ponts. Le programme résout des grilles de taille variable (mais carrées). Il n'y a pas de limite ormis que le programme prendra plus de temps pour des grilles de grande taille.

Le résolution ne vérifie pas si toutes les cases sont coloriées. Par conséquent, le programme propose parfois en résolution, une grille ou tous les chemins rejoignent bien les cases de même couleur mais où certaines cases sont vides.

1 ALGORITHMIQUE DU JEU

1.1 DÉFINITIONS

Les mots que nous allons utiliser pour décrire un objet ou un concept précis mis en oeuvre dans la résolution du problème sont des mots pouvant donner lieu à de nombreuses lectures et approches différentes qui peuvent fausser les démonstrations qui s'appuient dessus. Pour fixer les idées, nous nous devons d'imposer des bases, seul appui pour nos raisonnements ultérieurs.

Une *case* c sera considéré comme une élément admettant une unique couleur (que l'on pourra étiqueter par un entier positif) dans l'ensemble Ω. On admettra pour simplifier que o correspond à une abscence de couleur (on dira que la case n'est pas coloriée) et on pourra assimiler cet ensemble à $\mathbb{R}_+$.

Une *grille* $\mathcal{G}$ est une matrice de cases, à laquelle est associée une fonction $\eta_{\mathcal{G}} : C \to \Omega$, où C est l'ensemble des cases de la grille et $\forall c \in C, \exists \omega \in \Omega : \eta_{\mathcal{G}}(c) = \omega$, où ω est la couleur de la case c.

Une case *primaire* est une case qui possède une couleur fixée dans la grille *vide* correspondant à $\mathcal{G}$.

Une case sera dite *remplie* si son état est dans Ω^*.

On définie la notion de *chemin de couleur* ω comme étant une succession de cases de couleur ω de la grille formant un ensemble 4-connexe. De fait, deux chemins sur la grille ne peuvent pas s'intersecter. Un chemin est *valide* s'il débute dans une case primaire et aboutit dans une autre case primaire.

On appelle cases *adjacentes* deux cases liées par une arrête et qui partagent une couleur commune (deux cases successives sur un chemin associé à une couleur).

1.2 PROBLÈMES

1.2.1 Présentation du jeu

Le jeu Flow consiste à déterminer les chemins valides de chaque couleur dans la grille, et idéalement, à fixer la couleur de chacune des cases. Pour cela, nous devons passer par des étapes de lecture d'une fichier de grille, puis de résolution et enfin d'affichage. Dans cette partie, nous chercherons à présenter les algorithmes principaux utilisées.

1.2.2 Spécifications

Le premier problème auquel nous pouvons nous intéresser est un problème de validation d'une grille remplie (après que l'utilisateur ait joué au jeu par exemple). Le problème est spécifié de la façon suivante :

```
Problème : Validation
Entrée : Une grille G remplie.
Sortie : Un booléen indiquant si la grille est valide ou non.
```

Pour la résolution de ce problème, on doit vérifier les points suivants :

— pour chaque couleur, l'ensemble des cases de cette couleur doit être connexe,

— les cases d'une même couleur forment un chemin sans boucles,

— ce chemin débute dans une case primaire pour aboutir dans une autre case primaire.

Comme à chaque case est associée une unique couleur, on interdit ici toute intersection non-vide entre les ensemble de différentes couleurs.

Le second problème que nous avons traité est le problème d'existence ou non d'une solution, spécifié de la façon suivante :

```
Problème : Resolution
Entrée : Une grille G.
Sortie : Une grille R et un booléen r tel que :
    r = FAUX s'il n'existe pas de solutions de G,
    r = VRAI si R est une solution de G.
```

1.2.3 Algorithmes

Validation

La fonction principale du moteur de jeu est la vérification d'une grille résolue. En effet le moteur graphique gère la jouablilité d'un coup, ce qui enlève des possibilités de coups et réduit la complexité de la vérification.

Par exemple, une grille où des sommets colorés non primaires non relié à des sommets primaires n'est pas une grille que nous pouvons obtenir avec notre moteur graphique.

L'algorithme fonctionne de la façon suivante :

Data: Graphe représentant la grille
Result: Booléen Vrai si la grille est bien remplie, Faux sinon
while *Toutes les couleurs n'ont pas été testées* **do**
    Choisir nouvelle couleur;
    Chercher sommet primaire correspondant;
    Stocker l'indice de ce sommet;
    while *Sommet courant n'est pas primaire de clé différente de celle stockée* **do**
        if *Sommet courant est primaire de même clé* **then**
            retourner Faux;
        else
            Passer au sommet voisin;
        end
    end
    retourner Vrai;
end

Algorithm 1: Vérification de grille

Dans notre implémentation, le degré de chaque sommet est limité à deux. Lorsque deux sommets sont adjacents sur la grille, de la même couleur mais non liés, le parcours se fera correctement.

De plus, chaque sommet coloré étant lié à un sommet primaire, les seuls cas où l'algorithme retournera faux sont lorsque la grille n'est pas complétement remplie, ou lorsque qu'un chemin n'atteint pas un sommet primaire distinct. Nos listes d'adjacences se remplissent en tête, lorsqu'un sommet est au bout d'un chemin, il n'a qu'un seul voisin. En fait, l'algorithme retournera en arrière et parcourera le même chemin une seconde fois. Il se stoppera en arrivant sur un sommet primaire de même clé que celui de départ.

L'algorithme de validation a été construit avec l'interface graphique utilisant la librairie *ltk*. L'utilisateur lorsqu'il joue au jeu ne peux pas effectuer certaines actions, comme relier deux cases de couleurs différentes, intersecter des chemins, etc. Par

conséquent, la difficulté de notre fonction de validation se retrouve transférée à la gestion de l'interface graphique elle-même, et nous avons ainsi une fonction de valisation qui n'effectue que des vérifications de base pour valider un remplissage.

Résolution

Notre algorithme de résolution s'inspire de la recherche en profondeur dans un graphe (DFS). Son principe consiste à construire récursivement les chemins reliant les couleurs de la grille en appelant à chaque étape une recherche en profondeur qui tient compte des modifications apportés.

Considérons la grille de jeu représentée sur la Figure 1a page suivante. Avant de décrire le déroulement de l'algorithme sur cette grille nous allons mettre au point un certain nombre de conventions utiles pour la suite. Premièrement, à chaque couleur nous attribuons les entiers suivants (Rouge :1, Orange :2, Jaune :3, Bleu :4, Vert :5).

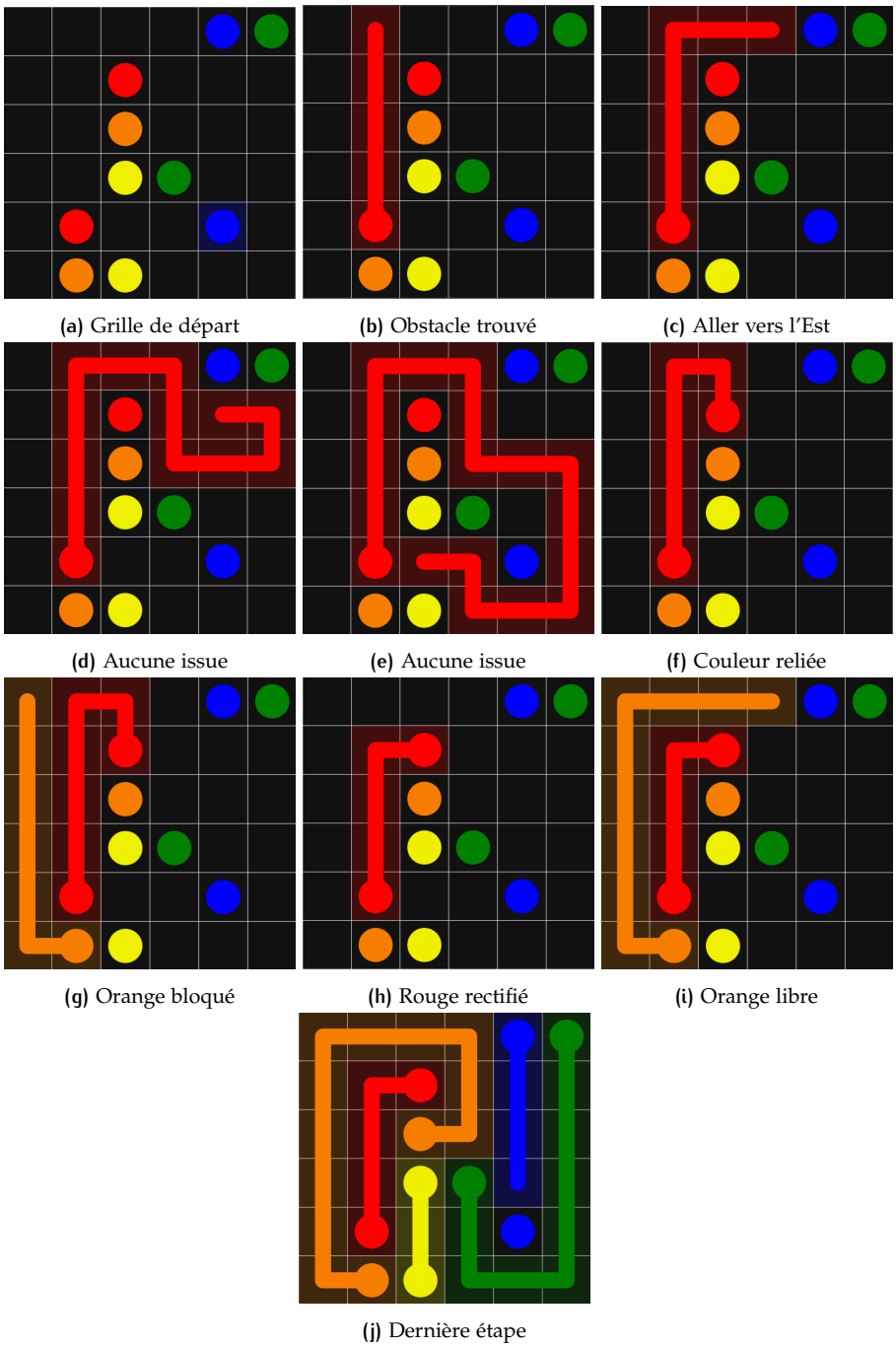
Deuxièmement, si l'algorithme est amené à choisir une direction où se déplacer, il devra la choisir selon l'ordre suivant : nord, est, sud, ouest. Ces conventions sont totalement arbitraires, néanmoins elles servent à prendre des décisions pendant l'éxéution des instructions. Finalement, nous appellerons obstacle les bords de la grille ou un noeud de couleur différentes de la couleur recherchée.

A l'état initial, le solver choisit la première couleur à relier, celle correspondant à l'entier le plus faible (Rouge). Ensuite, il en détermine une position dans la grille et choisit une direction disponible pour se déplacer (nord). Puis, il se déplace en ligne droite suivant cette direction jusqu'à rencontrer un obstacle (bords de la grille) (cf. Figure 1b page suivante).

Arrivé à cet étape, le solver doit encore décider d'une direction vers laquelle il va tourner (est) (cf. Figure 1c page suivante). Si aucune direction n'est disponible, le solver retourne en arrière d'une case jusqu'à en trouver une. Ce processus est répeté jusqu'à ce que le solver tombe sur un noeud primaire ayant la même couleur à relier et différent du noeud de départ. Les Figures 1d page suivante et 1e page suivante correspondent à des états intermédiares, on y voit notamment le solveur changer de direction lorsqu'il ne trouve aucune issue. La Figue 1f page suivante correspond à la fin de la recherche pour la couleur rouge.

La prochaine étape va consister à relier la deuxième couleur d'entier le plus faible (Orange). Toujours en suivant les même règles de recherche, nous constatons qu'au bout de quelques déplacements le chemin rouge bloque le chemin orange (cf. Figure 1g page suivante), le solveur fait donc des retours en arrière, ce qui le conduit en fin de compte à retracer un nouveau chemin pour la couleur rouge (cf. Figure 1h page suivante). Cette fois-çi la voie est libre pour le chemin orange qui peut se déplacer vers l'est (cf. Figure 1i page suivante). Le solveur va continuer ainsi à appliquer les mêmes règles jusqu'à relier toutes les couleurs dans l'ordre croissant des entiers qui leurs sont attribués (cf. Figure 1j page suivante).

A partir de cette description, nous pouvons constater que l'algorithme est amené à plusieurs reprises à prendre des décisions influancant ses déplacements. Ses décisions concernent notamment, le choix de la couleur, le choix de la direction, le déplacement en ligne droite et le retour en arrière. L'algorithme doit aussi identifier quelques situations particulières : aucun déplacement possible, obstacle atteint, couleur reliée.



(a) Grille de départ **(b)** Obstacle trouvé **(c)** Aller vers l'Est

(d) Aucune issue **(e)** Aucune issue **(f)** Couleur reliée

(g) Orange bloqué **(h)** Rouge rectifié **(i)** Orange libre

(j) Dernière étape

FIGURE 1 – Étapes de résolution

2 RÉALISATION DU PROGRAMME

2.1 STRUCTURE DES DONNÉES

2.1.1 Structure grille

La grille fournie en entrée est un fichier texte contenant les dimensions de la grille ainsi qu'une matrice symbolisant la grille tel que :

— Un # signifie une case vide.

— Un entier de signifie une case colorée primaire.

— Un + signifie un pont.

```
4 4
#1#3
#3#2
1#+#
2#44
```

FIGURE 1 – Exemple de grille 4×4

Pour stocker les informations de ce fichier, nous avons décidé d'utiliser une liste de n sous-liste où n est le nombre de cases de la grille.
Ces sous-listes contiennent un quadruplet et une liste d'adjacence.

Le quadruplet contient :

— Un entier correspondant au numéro de la case.

— Un entier correspondant à la couleur de la case.

— Un entier correspondant à la couleur du pont ou "nil" si la case n'est pas un pont.

— Un booléen indiquant si la case est primaire.

La liste d'adjacence contient la liste des cases adjacentes à la case dont il est question.

Cette transformation des données est effectuée par une fonction de parcours. Elle parcourt la structure et construit de manière récursive notre structure de données.

2.1.2 Structure de remplissage

Nous avons implémenté une deuxième structure afin de faciliter l'affichage et la validation de la grille dans l'interface graphique.
Cette structure est une liste de m sous-listes où m est le nombre de couleurs de la grille.
Chaque sous-liste contient :

— Un entier correspondant à la couleur du chemin décrit.

— Des couples d'entier correspondant aux coordonnées d'une case.

Les sous-listes représentent les chemins de chacune des couleurs.
Le premier couples de coordonnées correspont à une case primaire. Les couples suivants correspondent au chemin tracé de la couleur.

2.2 MISE EN OEUVRE DE L'ALGORITHME DE RÉSOLUTION

2.2.1 Tests et complexité

La résolution fonctionne pour des grilles de toute les tailles. Cependant certaines configurations de grilles mettent plus de temps que les autres :

— Dans le cas de grilles dépassant 8×8

— Dans le cas ou le nombre de couleur est faible par rapport aux dimensions de la grille. Par exemple une grille 8×8 avec 4 couleurs.

La résolution est de complexité $\Theta(C(n+m))$ où n est le nombre de cases, m le nombre d'arrêtes (liaison entre deux cases adjacentes) et C un facteur qui dépend du nombre de recherches effectué par l'algorithme.

Le nombre de recherches effectué par l'algorithme est dans le pire des cas celui d'un algorithme de force brute.

3 REMARQUES ET AMÉLIORATIONS POSSIBLES

GESTION DES PONTS

L'idée qui peut être implémentée pour la résolution des grilles avec des ponts serait d'ignorer l'occupation de ces cases.

En effet, avant d'avancer dans une case, on vérifie que la case n'est pas déjà occupée. Comme il n'est possible d'accéder qu'une fois aux cases qui l'entourent, il n'est possible d'accéder que deux fois à la case pont. (Elle est entourée de quatre cases.)

Il suffit donc d'ignorer l'occupation d'une case pont car on ne rencontrera jamais la situation de troisième passage de la case.

AMÉLIORATION DE LA COMPLEXITÉ DE LA RÉSOLUTION

Il pourrait être intéressant de trouver une moyen d'améliorer l'algorithme de résolution afin d'éviter qu'il ne fasse un trop grand nombre de calculs.

Cependant, étant donnée la nature de cet algorithme, son optimisation ne nous a pas semblée rentable en terme de temps passé pour éviter une quantité de calculs dont on ne sait si elle serait significative.