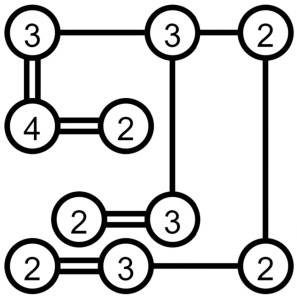
26/04/2024

Mohamad
Jad
Anthony
Sam

Hashiwokakero



Règles du jeu

Il faut connecter les îles avec autant de ponts (horizontaux ou verticaux) que le numéro de l'île.
Il ne peut y avoir plus de deux ponts entre deux îles.
Les ponts ne peuvent pas traverser des îles ou d'autres ponts.
Les ponts forment un lien continu entre toutes les îles.

Modélisation en logique du premier ordre

Domaine

$\forall i \Leftrightarrow \forall i \in colonnes\ comportant\ des\ îles$
$\forall j \Leftrightarrow \forall j \in lignes\ comportant\ des\ îles$

Prédicat

$P((i_1, j_1), (i_2, j_2))$: un pont allant de l'île de coordonnées i_1, j_1 à l'île de coordonnées i_2, j_2
L'expression $P((i_1, j_1), (i_2, j_2)) \wedge P((i_2, j_2), (i_1, j_1))$ désigne un pont double entre ces deux îles.

Relation

$croisement\ ((i_1, j_1), (i_2, j_2), (i_3, j_3)(i_4, j_4))$: retourne $True$ si les ponts entre les îles (i_1, j_1) et (i_2, j_2) et les ponts entre les îles (i_3, j_3) et (i_4, j_4) peuvent se croiser.

Fonctions

- $combinaisons\ (i, j)$: génère toutes les combinaisons de ponts qui peuvent être connectés à une île située aux coordonnées $i,\ j$, pour satisfaire son degré, c'est-à-dire le nombre de ponts requis pour cette île.
  Exemple 1 de $combinaison(i, j)$ avec degré de (i, j) = **1** :
  $(P((i, j), (i_2, j_2)) \wedge \overline{P((i_2, j_2), (i, j))} \wedge \ldots) \vee (P((i_2, j_2), (i, j)) \wedge \overline{P((i, j), (i_2, j_2))} \wedge \ldots\ )\ \ldots$
  Exemple 2 de $combinaison(i, j)$ avec degré de (i, j) = **2** :
  $(P((i, j), (i_2, j_2)) \wedge\ P((i_2, j_2), (i, j)) \wedge \overline{P((i, j), (i_2, j_2))} \wedge\ \overline{P((i, j), (i_3, j_3))} \wedge \ldots)$
  $\vee\ (P((i, j), (i_2, j_2)) \wedge\ P((i, j), (i_3, j_3)) \wedge \overline{P((i, j), (i_2, j_2))} \wedge\ \overline{P((i_2, j_2), (i, j))} \wedge \ldots)$
- $chemins\ ((i_1, j_1))$: retourne une forme normale disjonctive de tous les chemins (conjonction de ponts) qui passe par tous les îles qui commence par l'île (i_1, j_1).
  Exemple $chemins((i_1, j_1))$ avec 4 îles à (i_1, j_1), (i_2, j_2) , (i_3, j_3) et (i_4, j_4) :
  $P((i_1, j_1), (i_2, j_2)) \wedge P((i_2, j_2), (i_3, j_3)) \wedge P((i_3, j_3), (i_4, j_4)) \vee \ldots$

Contraintes du problème

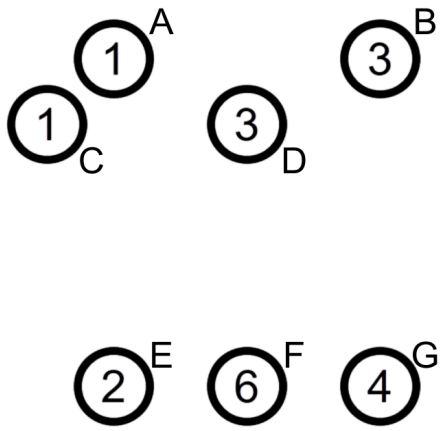
1. Chaque île ne peut être reliée qu'à une autre île sur la même ligne et colonne :
   $\forall(i_1, j_1)\ \forall(i_2, j_2)\ [\neg((i_1 = i_2) \oplus (j_1 = j_2)) \Rightarrow \neg P(i_1, j_1, i_2, j_2)]$

2. Les ponts ne peuvent pas traverser d'autres ponts :
   $\forall(i_1, j_1)\ \forall(i_2, j_2)\ \forall(i_3, j_3)\ \forall(i_4, j_4)\ [croisement((i_1, j_1), (i_2, j_2), (i_3, j_3)(i_4, j_4))$
   $\Rightarrow P((i_1, j_1), (i_2, j_2)) \oplus P((i_3, j_3)(i_4, j_4))]$
3. Les ponts ne peuvent pas traverser des îles :
   $\forall(i_1, j_1)\ \forall(i_2, j_2)\ \exists\ (i_3, j_3)\ [(i_1 = i_2 = i_3) \wedge (\ j_1 >\ j_3 > j_2) \vee (j_1 = j_2 = j_3) \wedge (\ i_1 >\ i_3 > i_2)$
   $\Rightarrow \neg P((i_1, j_1), (i_2, j_2))]$
4. Il faut connecter les îles avec autant de ponts que le numéro de l'île :
   $\forall(i, j)\ [combinaisons(i, j)]$

5. Les ponts forment un lien continu entre toutes les îles (graphe connexe) :
   $chemins((i_1, j_1))$

Modélisation en forme normale conjonctive

La variable booléenne IJ représente le fait qu'il y a un pont entre l'île I et l'île J. Pour illustrer cela, prenons comme exemple l'ensemble d'îles $\{A,\ B,\ C,\ D,\ E,\ F,\ G\}$ dans le cas ci-dessous :



Notre programme Python lit le fichier suivant :

```
6 6
010003
100300
000000
000000
000000
020604
```

Les deux premiers '6' indiquent les dimensions de la grille de jeu. Ensuite, chaque île est désignée par un chiffre compris entre 1 et 8 inclus, tandis que les '0' représentent des cases vides sans île.

Il génère ensuite le dictionnaire suivant en Python :

```
{'A': [1, 0, 1, ['E', 'B']], 'B': [5, 0, 3, ['G', 'A']], 'C': [0, 1, 1,
['D']], 'D': [3, 1, 3, ['F', 'C']], 'E': [1, 5, 2, ['A', 'F']], 'F': [3, 5,
6, ['D', 'E', 'G']], 'G': [5, 5, 4, ['B', 'F']]}
```

Dans ce dictionnaire, chaque clé représente le nom de l'île. Les valeurs associées à chaque clé sont une liste composée de quatre éléments :

1. Le premier entier représente la colonne de l'île dans le graphe.
2. Le deuxième entier représente la ligne de l'île dans le graphe.
3. Le troisième entier représente le numéro (ou le degré) de l'île.
4. Dans cette partie de la liste, les îles qui peuvent se connecter à l'île clé sont énumérées, respectant ainsi les contraintes 1 et 3 (laissées au programme Python pour simplifier la formule) : chaque île ne peut être reliée qu'à une autre île sur la même ligne ou colonne, et les ponts ne peuvent pas traverser d'autres îles.

Nous fournissons au SAT-solveur les contraintes suivantes, exprimées en forme normale conjonctive, qui englobent les contraintes 2, 4 et 5 du problème.

Contrainte 2 : Les ponts ne peuvent pas traverser d'autres ponts

```
(-AE+-CD) . (-AE+-DC) . (AE+EA+CD+DC) . (-EA+-CD) . (-EA+-DC)
```

Contrainte 4 : Il faut connecter les îles avec autant de ponts que le numéro de l'île

```
(-AE) . (-EA) . (-AB+-BA) . (AB+BA) . (BG) . (GB) . (-CD+-DC) . (CD+DC) .
(DF) . (FD) . (EF) . (FE) . (FG) . (GF)
```

Contrainte 5 : Les ponts forment un lien continu entre toutes les îles

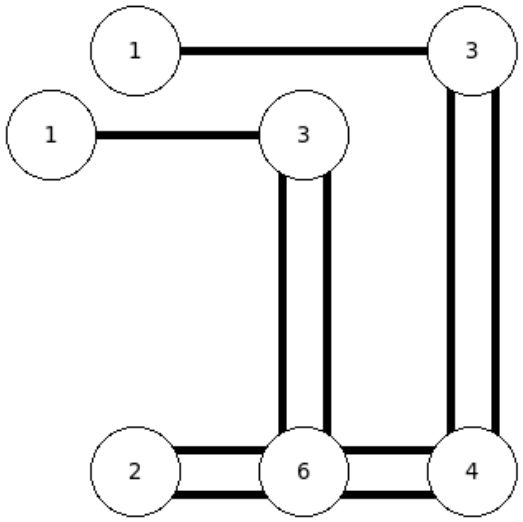
```
(AE+EA+FE+EF) . (AE+EA+BA+AB) . (AE+EA+GB+BG) . (AE+EA+GF+FG) .
(FE+EF+BA+AB) . (FE+EF+GB+BG) . (FE+EF+GF+FG) . (DF+FD) . (CD+DC) .
(BA+AB+GB+BG) . (BA+AB+GF+FG) . (GB+BG+GF+FG)
```

Solution

SAT-solveur

```
-AE -EA AB -BA BG GB -CD DC DF FD EF FE FG GF
```

Graphe



Implémentation

On utilise le dictionnaire généré en appliquant plusieurs fonctions sur ce dictionnaire.

La fonction `func_combinaison` permet de déterminer les conjonctions de modèles possibles pour satisfaire chaque point, en utilisant la forme normale conjonctive (FNC) via un subprocess. De même, la fonction `croisement` génère les clauses pour les croisements, basées sur les conjonctions des clauses avec des opérations XOR entre deux ponts possiblement croisés. Encore une fois, la transformation en FNC est effectuée via un subprocess.

Pour les chemins connexes, la fonction `chemin` génère la disjonction de tous les chemins menant à des graphes connexes, également transformée en FNC via un subprocess.

Pour obtenir la formule finale, nous concaténant les trois formules précédentes et les transformons en fichiers DIMACS pour les résoudre à l'aide de MiniSat. Si une solution est trouvée, nous affichons le graphe résolu. Dans le cas contraire, où aucune solution n'est trouvée, nous affichons simplement la grille initiale avec un message "INSATISFIABLE!!!" dans le terminal.

Merci