



Université de Bourgogne, UFR Sciences et Techniques, Département I.E.M.

Projet « Algorithmique et complexité »

Groupe
Elio GENCE
Guillaume BELDILMI

Sujet n°4 : Eternity II

Contents

1.1	Présentation du projet	4
1.2	Organisation	4
1.3	Explication de l'algorithme utilisé	4
1.3.1	Analyse de la complexité	5
1.3.2	Explication détaillés (code sans Rotation)	5
1.4	Résultats et analyses	9
1.4.1	Temps	9
1.4.2	Affichage	10
1.5	Conclusion	12
1.6	Annexe	12

1.1 Présentation du projet

Eternity II est un jeu de puzzle inventé par Christopher Monckton en 2007. Le but du jeu est de placer des pièces de puzzle de manière à ce que les couleurs des bords des pièces soient identiques aux couleurs des bords adjacents. Les pièces sont carrées et chaque côté est coloré. Les pièces sont placées sur une grille de taille $n \times n$.

Notre projet était de créer un solveur capable de résoudre des puzzles d'Eternity II de taille 12 par 12 avec une dizaine de couleurs. Nos seules contraintes étaient que notre programme devait être écrit en Ocaml et que la résolution devait se faire par une méthode de recherche arborescente avec retour en arrière (backtrack).

Pour cela, nous avons divisé le projet en plusieurs étapes : tout d'abord, nous avons créé une structure de données pour représenter les pièces et la grille. Ensuite, nous avons créé un ensemble de fonctions permettant de générer un puzzle aléatoire, de mélanger les pièces du puzzle, et enfin de résoudre le puzzle. A cela s'ajoute une autre fonction permettant d'afficher le puzzle afin d'avoir un retour sur l'exécution du programme.

1.2 Organisation

Premièrement, nous avons réalisé un prototype conformément à l'ISO-1664 dans un langage plus abordable, Python. Ce prototype nous a permis de mieux comprendre le problème et de définir l'architecture de notre programme.

Ensuite, nous avons transcrit ce prototype en Ocaml, en corrigeant quelques erreurs et en ajoutant des fonctionnalités précédemment omises après avoir pris en compte le retour du corps enseignant.

Nous avons ensuite testé notre programme sur des puzzles progressivement de plus en plus grands.

Enfin, nous avons peaufiné notre programme en simplifiant le code et en corrigeant l'affichage du puzzle afin de rendre le résultat plus facile à manipuler.

1.3 Explication de l'algorithme utilisé

Nous avons utilisé un algorithme naïf de backtracking pour résoudre le puzzle. L'idée est de placer chaque pièce sur la grille en progressant selon le sens d'écriture. Si une pièce ne peut pas être placée, on revient à la pièce précédemment placée et on essaie une autre pièce. Si aucune pièce ne peut être placée, on revient à la pièce précédente, et ainsi de suite jusqu'à ce que le puzzle soit résolu. Une fois le puzzle résolu, on affiche le résultat et puis le programme poursuit son exécution afin de trouver d'autres solutions.

D'autres algorithmes plus performants existent, optimisant notamment l'ordre de placement des pièces (commençant par les pièces de bords puis progressant vers le centre en zigzag ou en spirale), mais nous avons choisi de rester sur un algorithme simple pour des raisons de simplicité et de temps.

Nous avons pour notre faire 2 versions différentes : une avec rotation de pièce et une autre sans.

1.3.1 Analyse de la complexité

Nous allons faire l'analyse de la complexité sur le code sans Rotation. Lors d'une itération, le nombre d'opération à faire est minimale (quelques tests et opérations). Nous allons donc simplifier le tout et pour le bien de nos calculs, nous allons supposer qu'une itération est égale à $O(1)$.

Partons d'une situation impossible : toutes les pièces sont exactement similaires (en ignorant également le concept de bordure). Ceci est notre pire scénario ! L'algorithme va donc tester toutes les pièces à toutes les emplacements possibles $N = \text{Taille du puzzle largeur/longueur } N*N = \text{Nombre de pièces}$. En d'autres termes, l'algorithme va parcourir $N*N$ fois la première case puis $(N*N)*(N*N)$ la seconde case, etc. Cela nous donne donc une complexité de $O(N!)$.

Maintenant le meilleur scénario : toutes les pièces sont uniques et une seule solution est possible. Alors, dans cette situation, l'algorithme va être bien plus rapide : il va parcourir $N*N$ la première case, $N*N$ la seconde case (car seulement une seule pièce valide à la première case), etc. Nous donnons donc une complexité de $O(N)$.

Cela veut donc dire que la complexité va varier de $O(N)$ à $O(N!)$. Le facteur qui va décider de la complexité de l'algorithme est le nombre de couleurs, plus celui-ci sera élevé et plus les pièces seront uniques. En d'autres termes : moins nous avons de couleurs, plus de tests sont nécessaires à être effectués pour vérifier toutes les solutions et plus le code sera coûteux/long à exécuter.

Pour le code avec Rotation, il faut quadrupler chaque itération, ce qui donne une complexité variable entre $O(4N)$ et $O(4N!)$.

1.3.2 Explication détaillée (code sans Rotation)

Pour notre code, nous pouvons le diviser en 5 parties : - définition d'une pièce 1.1 - création des structures et initialisation 1.2 - test de pièce lors du backtrack 1.3 - backtrack 1.3 - affichage du résultat 1.3

Listing 1.1: Déclaration pièce (sources/tile.ml)

```

1 type tile = {
2   id: int; (* permet de savoir l'emplacement d'origine *)
3   top: int;
4   right: int;
5   bottom: int;
6   left: int;
7 } ;;

```

Nous utilisons dans notre code 2 structures : board et tiles. Board est un tableau de tableau de pièce, il représente le plateau de jeu sur lequel on va venir placer les pièces pour essayer de résoudre le puzzle. Tiles quant à lui est un tableau de pièce, il va servir à stocker la liste de pièces encore non placées sur le board. (à noter que si une pièce identifiante est égale à -1, alors on considère l'emplacement comme vide).

L'initialisation du board se fait avec un parcours de haut en bas et de gauche à droite / droite à gauche une ligne sur deux (afin de simuler un parcours en zigzag dans les algorithmes de backtrack optimal). Lors de la création d'une pièce, nous avons donc besoin de vérifier seulement 3 paramètres : - s'il est adjacent à une bordure (si c'est le cas, les cotés adjacents prennent

la valeur 0) - la valeur du coté inférieur de la pièce supérieur - la valeur du coté gauche de la pièce à droite (ou inverse pour un parcours de droite à gauche) - le reste des coté prendront une valeur aléatoire parmi celle disponible (défini au tout début du code pour une modification rapide)

Listing 1.2: Initialisation du board (sources/init_board.ml)

```

1 let init_board size =
2   let board = create_board size in
3   for i = 0 to size - 1 do
4     if i mod 2 = 0 then
5       for j = 0 to size-1 do
6         board.(i).(j) <-
7           {
8             top = if i = 0 then 0 else board.(i-1).(j).bottom;
9             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
10            left = if j = 0 then 0 else board.(i).(j - 1).right;
11            right = if j = size - 1 then 0 else Random.int nbColor + 1;
12            id = i*size+j;
13            flag=true;
14          };
15       done
16     else
17       for j = size-1 downto 0 do
18         board.(i).(j) <-
19           {
20             top = board.(i-1).(j).bottom;
21             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
22             left = if j = 0 then 0 else Random.int nbColor + 1;
23             right = if j = size - 1 then 0 else board.(i).(j + 1).left;
24             id = i*size+j;
25             flag=true;
26           };
27       done;
28   done;
29   board;;
30
31 (* Création de gauche à droite puis droite à gauche une ligne sur deux pour simuler le
   parcours optimal non fait pour le backtrack *)

```

Pour ce qui est du test des tuiles lors du backtrack voici le code et ce qui est testé

Listing 1.3: Test de compatibilité des pièces (fonction imbriqués)(sources/test_tile.ml)

```

1 let test_tile t i j =
2   let res = ref true in
3
4   (* Piece à gauche existante *)
5   if j != 0 && board.(i).(j - 1).right != tiles.(t).left then
6     res := false
7   (* Bordure gauche impossible en intérieur *)
8   else if j != 0 && tiles.(t).left = 0 then
9     res := false
10  (* Bordure droite impossible en intérieur *)
11  else if j != (size - 1) && tiles.(t).right = 0 then

```

```

12     res := false
13     (* Bordure gauche *)
14     else if j = 0 && (tiles.(t).left != 0) then
15         res := false
16         (* Bordure droite *)
17         else if j = (size - 1) && (tiles.(t).right != 0) then
18             res := false
19
20         (* Piece supérieur existante *)
21         else if i != 0 && board.(i - 1).(j).bottom != tiles.(t).top then
22             res := false
23             (* Bordure supérieur impossible en intérieur *)
24             else if i != 0 && tiles.(t).top = 0 then
25                 res := false
26                 (* Bordure inférieur impossible en intérieur *)
27                 else if i != (size - 1) && tiles.(t).bottom = 0 then
28                     res := false
29                     (* Bordure supérieur *)
30                     else if i = 0 && (tiles.(t).top != 0) then
31                         res := false
32                         (* Bordure inférieur *)
33                         else if i = (size - 1) && (tiles.(t).bottom != 0) then
34                             res := false;
35
36         (* Parcours de droite à gauche, de haut en bas donc non nécessaire
37         de vérifier les pièces en dessous et à droite
38         mais nécessaire de vérifier les bordures *)
39
40     !res;
41     in

```

Le backtrack la partie la plus important est également la partie la plus simple, le test s'occupant du principal tout ce qui reste à faire maintenant est de gérer les boucles de recherches et les appels récursifs.

Listing 1.4: Backtrack (fonction principale) (sources/backtrack.ml)

```

1  let rec solve_backtrack board tiles i j = (* i et j représentent la position du board que l'
    on va tester *)
2  if board.(i).(j).id = -1 then (* Vérification qu'aucune pièce n'es déjà placé *)
3  for t = 0 to (Array.length tiles) - 1 do
4  if tiles.(t).id != -1 then (* Vérification que la pièce est valide *)
5  if test_tile t i j then begin (* Test de compatibilité de la pièce *)
6  (* Placement de la pièce et retrait de la liste des pièces disponibles *)
7  board.(i).(j) <- tiles.(t);
8  tiles.(t) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0; flag=true };
9  (* backtrack *)
10 if j < size - 1 then
11     solve_backtrack board tiles i (j+1)
12 else if i < size - 1 then
13     solve_backtrack board tiles (i+1) 0
14 else
15     print_board board;
16     (* On retire la pièce et ajout dans la liste des

```

```

17      pièce disponible pour chercher les prochaines solutions *)
18      tiles.(t) <- board.(i).(j);
19      board.(i).(j) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0; flag=true };
20      end
21  done
22  (* failsafe pour le backtrack *)
23  else if j < size - 1 then
24      solve_backtrack board tiles i (j+1)
25  else if i < size - 1 then
26      solve_backtrack board tiles (i+1) 0
27  else
28      print_board board;
29  ;;

```

Enfin il ne nous reste plus que l’affichage qui est composé de plusieurs boucles imbriquées afin d’obtenir un affichage facilement lisible

Listing 1.5: Affichage du résultat (board)(sources/print.ml)

```

1  let print_board = fun board ->
2      Array.iter (fun row ->
3          Array.iter (fun tile ->
4              Printf.printf "+-----";
5              ) row;
6              Printf.printf "+\n";
7              Array.iter (fun tile ->
8                  Printf.printf "| %02d " tile.top;
9                  ) row;
10             Printf.printf "|\n";
11             Array.iter (fun tile ->
12                 Printf.printf "|%02d %04d %02d" tile.left tile.id tile.right;
13                 ) row;
14             Printf.printf "|\n";
15             Array.iter (fun tile ->
16                 Printf.printf "| %02d " tile.bottom;
17                 ) row;
18             Printf.printf "|\n";
19             ) board;
20      Printf.printf "+";
21      Array.iter (fun _ ->
22          Printf.printf "-----+";
23          ) board;
24      Printf.printf "\n" ;;

```

Explication détaillés (code avec Rotation)

L’algorithme avec rotation est très similaire cependant dans le backtrack 1.4 et dans le test de pièce 1.3 on rajoute un nouveau paramètre (que l’on nomme k) variant entre 0 et 3 afin de pouvoir tester chaque rotation et ainsi obtenir de nouvelle solution.

Listing 1.6: Paramètre k pour savoir le sens de rotation de la pièce (sources/rotation.ml)

```

1  if k=0 then

```


		Taille*Taille					
Couleurs		4	6	8	10	11	12
	4	0,050~0,054	00,050~0,074	3,2~45s	+2m	/	/
	8	0,050~0,054	0,058~0,062	0,060~0,065	0,060~0,065	1~5s	+2m
	10	0,050~0,054	0,050~0,054	0,050~0,054	0,052~0,055	0,053~0,066	0,057~0,087

Figure 1.1: Temps sans rotation

```

2   board.(i).(j) <- tiles.(t)
3   else if k = 1 then
4     board.(i).(j) <- rotate_tile_left tiles.(t)
5   else if k = 2 then
6     board.(i).(j) <- rotate_tile_180 tiles.(t)
7   else
8     board.(i).(j) <- rotate_tile_right tiles.(t);

```

Ainsi toutes les solutions possibles seront obtenus mais cela entraine 2 problème : La durée -> le code est bien plus long car bien plus de solution à tester Les solutions -> Le nombre de solution obtenus peut en réalité être divisé par 4 car nous trouvons des solutions "doublons" que l'on peut retrouver simplement en tournant le plateau de puzzle. Une solution possible à ce problème serait de fixer un coin et de faire le backtrack à partir de ce moment là.

1.4 Résultats et analyses

1.4.1 Temps

Voici les temps observé pour un code sans rotation :

Comme on peut le constater la durée moyenne varie selon le nombre de couleur et la taille (logique). Cependant le temps varie de façon différente. Tandis que le temps se rallonge lorsque l'on augmente la taille, il diminue au contraire quand le nombre de couleur augmente. C'est logique car le nombre de cas à tester et de solution possible diminue lorsque le nombre de couleurs augmentent, si toute les pièces sont très atypiques alors il deviens plus dur de les substituer à l'inverse si les pièces se ressemblent (aka le nombre de couleur est faible) alors le programme va devoir tester bien plus de possibilité de résolution car les pièces vont s'imbriquer plus facilement les unes dans les autres.

Voici les temps observé pour un code avec rotation :

		Taille*Taille					
		4	6	8	10	11	12
Couleurs	4	0,063~0,090	++6m	/	/	/	/
	8	0,062~0,075	0,070~0,090	++2m	/	/	/
	10	0,062~0,075	0,070~0,090	0,3-1s	++2m	/	/
	15	0,062~0,075	0,070~0,090	0,07~0,130	5-10s	++2m	/
	20	0,062~0,075	0,070~0,090	0,070~0,095	0,2~0,8s	0,4~1s	1m4

Nous observons ici le même phénomène qu’avec le code sans rotation cependant les temps sont bien plus car, contrairement au précédent code, chaque pièce doit être tester 4 fois (une dans chaque sens) rendant la recherche plus fastidieuse mais également en augmentant le nombre de solution possible !

1.4.2 Affichage

Afin de simplifier l’exploitation des fichiers de résultats après l’exécution de notre programme (comptage, suppression d’éventuels doublons, ou encore d’autres traitements à base de grep ou autre...), nous affichons systématiquement la balise <deb> au début de chaque grille et sautons une ligne vide une fois cette dernière terminée.

Voici un exemple de puzzle de taille 5 par 5 avec 5 couleurs résolu par notre programme sans rotation :

Listing 1.7: Résultat sans Rotation (sources/resNR_5x5.txt)

```

1 Initial board
2 <deb>
3 +-----+-----+-----+-----+-----+
4 | 00 | 00 | 00 | 00 | 00 |
5 |00 0000 05|05 0001 04|04 0002 03|03 0003 02|02 0004 00|
6 | 01 | 01 | 02 | 05 | 04 |
7 +-----+-----+-----+-----+-----+
8 | 01 | 01 | 02 | 05 | 04 |
9 |00 0005 05|05 0006 05|05 0007 05|05 0008 01|01 0009 00|
10 | 02 | 01 | 01 | 01 | 02 |
11 +-----+-----+-----+-----+-----+
12 | 02 | 01 | 01 | 01 | 02 |
13 |00 0010 02|02 0011 02|02 0012 02|02 0013 05|05 0014 00|
14 | 01 | 05 | 05 | 02 | 03 |
15 +-----+-----+-----+-----+-----+
16 | 01 | 05 | 05 | 02 | 03 |
17 |00 0015 03|03 0016 01|01 0017 05|05 0018 02|02 0019 00|
18 | 03 | 01 | 04 | 05 | 01 |
19 +-----+-----+-----+-----+-----+
20 | 03 | 01 | 04 | 05 | 01 |
21 |00 0020 01|01 0021 04|04 0022 04|04 0023 05|05 0024 00|

```

```

22 | 00 | 00 | 00 | 00 | 00 |
23 +-----+-----+-----+-----+-----+
24
25
26
27 Shuffled board
28 <deb>
29 +-----+-----+-----+-----+-----+
30 | 03 | 01 | 05 | 02 | 01 |
31 |02 0019 00|00 0005 05|04 0023 05|05 0018 02|00 0015 03|
32 | 01 | 02 | 00 | 05 | 03 |
33 +-----+-----+-----+-----+-----+
34 | 04 | 01 | 04 | 00 | 00 |
35 |01 0009 00|01 0021 04|04 0022 04|00 0000 05|04 0002 03|
36 | 02 | 00 | 00 | 01 | 02 |
37 +-----+-----+-----+-----+-----+
38 | 01 | 01 | 01 | 00 | 05 |
39 |02 0011 02|02 0013 05|05 0006 05|02 0004 00|05 0008 01|
40 | 05 | 02 | 01 | 04 | 01 |
41 +-----+-----+-----+-----+-----+
42 | 02 | 05 | 03 | 05 | 01 |
43 |05 0014 00|03 0016 01|00 0020 01|01 0017 05|02 0012 02|
44 | 03 | 01 | 00 | 04 | 05 |
45 +-----+-----+-----+-----+-----+
46 | 01 | 02 | 02 | 00 | 00 |
47 |05 0024 00|05 0007 05|00 0010 02|03 0003 02|05 0001 04|
48 | 00 | 01 | 01 | 05 | 01 |
49 +-----+-----+-----+-----+-----+
50
51
52
53 Solving: may the time be with you
54 <deb>
55 +-----+-----+-----+-----+-----+
56 | 00 | 00 | 00 | 00 | 00 |
57 |00 0000 05|05 0001 04|04 0002 03|03 0003 02|02 0004 00|
58 | 01 | 01 | 02 | 05 | 04 |
59 +-----+-----+-----+-----+-----+
60 | 01 | 01 | 02 | 05 | 04 |
61 |00 0005 05|05 0006 05|05 0007 05|05 0008 01|01 0009 00|
62 | 02 | 01 | 01 | 01 | 02 |
63 +-----+-----+-----+-----+-----+
64 | 02 | 01 | 01 | 01 | 02 |
65 |00 0010 02|02 0011 02|02 0012 02|02 0013 05|05 0014 00|
66 | 01 | 05 | 05 | 02 | 03 |
67 +-----+-----+-----+-----+-----+
68 | 01 | 05 | 05 | 02 | 03 |
69 |00 0015 03|03 0016 01|01 0017 05|05 0018 02|02 0019 00|
70 | 03 | 01 | 04 | 05 | 01 |
71 +-----+-----+-----+-----+-----+
72 | 03 | 01 | 04 | 05 | 01 |
73 |00 0020 01|01 0021 04|04 0022 04|04 0023 05|05 0024 00|
74 | 00 | 00 | 00 | 00 | 00 |
75 +-----+-----+-----+-----+-----+

```

```

76
77 <deb>
78 +-----+-----+-----+-----+
79 | 00 | 00 | 00 | 00 | 00 |
80 |00 0000 05|05 0001 04|04 0002 03|03 0003 02|02 0004 00|
81 | 01 | 01 | 02 | 05 | 04 |
82 +-----+-----+-----+-----+
83 | 01 | 01 | 02 | 05 | 04 |
84 |00 0005 05|05 0006 05|05 0007 05|05 0008 01|01 0009 00|
85 | 02 | 01 | 01 | 01 | 02 |
86 +-----+-----+-----+-----+
87 | 02 | 01 | 01 | 01 | 02 |
88 |00 0010 02|02 0012 02|02 0011 02|02 0013 05|05 0014 00|
89 | 01 | 05 | 05 | 02 | 03 |
90 +-----+-----+-----+-----+
91 | 01 | 05 | 05 | 02 | 03 |
92 |00 0015 03|03 0016 01|01 0017 05|05 0018 02|02 0019 00|
93 | 03 | 01 | 04 | 05 | 01 |
94 +-----+-----+-----+-----+
95 | 03 | 01 | 04 | 05 | 01 |
96 |00 0020 01|01 0021 04|04 0022 04|04 0023 05|05 0024 00|
97 | 00 | 00 | 00 | 00 | 00 |
98 +-----+-----+-----+-----+
99
100
101
102 Solving: end of time

```

Voici un exemple de puzzle de taille 5 par 5 avec 5 couleurs résolu par notre programme avec rotation : 1.11 Comme dis lors de l'explication, nous obtenons tous les résultats dans 4 rotations possibles donc il faut diviser par 4 le nombre de résultat obtenus pour avoir le nombre réel de solutions possibles (un code prenant en compte ce détail sera ajouter en annexe 1.10

1.5 Conclusion

Finalement, et pour conclure, nous avons réussi à implémenter ce solveur. Nous pensons avoir compris en profondeur les subtilités algorithmique derrière ce dernier et leur importance, et ce, malgré les améliorations possibles au résultat de notre travail. Nous restons donc relativement satisfait de notre projet et remercions les profs de TP (Boris Bordeaux et Diarra Ibrahim) qui sont resté disponible pour répondre à toutes nos questions (notamment sur comment pour effectuer le parcours de façon optimal par exemple).

1.6 Annexe

Listing 1.8: Code sans Rotation (sources/code.ml)

```

1 Random.self_init ();;
2
3 let size = 12 ;;

```

```

4  let nbColor = 10 ;;
5
6  type tile = {
7    id: int;
8    top: int;
9    right: int;
10   bottom: int;
11   left: int;
12 } ;;
13
14 let create_board size =
15   Array.init size (fun _ ->
16     Array.init size (fun _ ->
17       { id=(-1); top=0; right=0; left=0; bottom=0}
18     )
19   ) ;;
20
21 let init_board size =
22   let board = create_board size in
23   for i = 0 to size - 1 do
24     if i mod 2 = 0 then
25       for j = 0 to size-1 do
26         board.(i).(j) <-
27           {
28             top = if i = 0 then 0 else board.(i-1).(j).bottom;
29             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
30             left = if j = 0 then 0 else board.(i).(j - 1).right;
31             right = if j = size - 1 then 0 else Random.int nbColor + 1;
32             id = i*size+j;
33           };
34       done
35     else
36       for j = size-1 downto 0 do
37         board.(i).(j) <-
38           {
39             top = board.(i-1).(j).bottom;
40             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
41             left = if j = 0 then 0 else Random.int nbColor + 1;
42             right = if j = size - 1 then 0 else board.(i).(j + 1).left;
43             id = i*size+j;
44           };
45       done;
46   done;
47   board;;
48
49 let print_board = fun board ->
50   Array.iter (fun row ->
51     Array.iter (fun tile ->
52       Printf.printf "+-----";
53     ) row;
54     Printf.printf "+\n";
55     Array.iter (fun tile ->
56       Printf.printf "| %02d " tile.top;
57     ) row;

```

```

58     Printf.printf "|\\n";
59     Array.iter (fun tile ->
60         Printf.printf "|%02d %04d %02d" tile.left tile.id tile.right;
61     ) row;
62     Printf.printf "|\\n";
63     Array.iter (fun tile ->
64         Printf.printf "| %02d " tile.bottom;
65     ) row;
66     Printf.printf "|\\n";
67 ) board;
68 Printf.printf "+";
69 Array.iter (fun _ ->
70     Printf.printf "-----+";
71 ) board;
72 Printf.printf "\\n" ;;
73
74
75 let shuffle_board board =
76     for i = 0 to size - 1 do
77         for j = 0 to size - 1 do
78             let random_i = Random.int size in
79             let random_j = Random.int size in
80             let temp_tile = board.(i).(j) in
81             board.(i).(j) <- board.(random_i).(random_j);
82             board.(random_i).(random_j) <- temp_tile;
83         done;
84     done;
85     board ;;
86
87 let init_tiles board=
88     let tiles = Array.make (size*size) ({ id=(-1); top=0; right=0; left=0; bottom=0}) in
89     for i = 0 to size - 1 do
90         for j = 0 to size - 1 do
91             tiles.((i)*size+j) <- board.(i).(j);
92         done;
93     done;
94     tiles ;;
95
96 let rec solve_backtrack board tiles i j =
97
98     let test_tile t i j =
99         let res = ref true in
100         if j != 0 && board.(i).(j - 1).right != tiles.(t).left then
101             res := false
102         else if j != 0 && tiles.(t).left = 0 then
103             res := false
104         else if j != (size - 1) && tiles.(t).right = 0 then
105             res := false
106         else if i != 0 && board.(i - 1).(j).bottom != tiles.(t).top then
107             res := false
108         else if i != 0 && tiles.(t).top = 0 then
109             res := false
110         else if i != (size - 1) && tiles.(t).bottom = 0 then
111             res := false

```

```

112     else if i = 0 && (tiles.(t).top != 0) then
113         res := false
114     else if i = (size - 1) && (tiles.(t).bottom != 0) then
115         res := false
116     else if j = 0 && (tiles.(t).left != 0) then
117         res := false
118     else if j = (size - 1) && (tiles.(t).right != 0) then
119         res := false;
120     !res;
121 in
122
123 if board.(i).(j).id = -1 then
124     for t = 0 to (Array.length tiles) - 1 do
125         if tiles.(t).id != -1 then
126             if test_tile t i j then begin
127                 board.(i).(j) <- tiles.(t);
128                 tiles.(t) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0};
129                 if i < size - 1 then
130                     solve_backtrack board tiles (i+1) j
131                 else if j < size - 1 then
132                     solve_backtrack board tiles 0 (j+1)
133                 else
134                     print_board board;
135                     tiles.(t) <- board.(i).(j);
136                     board.(i).(j) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0};
137                 end
138             end
139         else if i < size - 1 then
140             solve_backtrack board tiles (i+1) j
141         else if j < size - 1 then
142             solve_backtrack board tiles 0 (j+1)
143         else
144             print_board board;
145     ;;
146
147 (* main *)
148
149 let b = init_board size ;;
150 Printf.printf "Initial board\n" ;;
151 print_board b ;;
152
153 Printf.printf "\n\nShuffled board\n" ;;
154 let b = shuffle_board b ;;
155 print_board b ;;
156
157
158 Printf.printf "\n\nSolving: may the time be with you\n";;
159 let tiles = init_tiles b;;
160 let board = create_board size;;
161 solve_backtrack board tiles 0 0;
162 Printf.printf "\n\nSolving: end of time\n" ;

```

Listing 1.9: Code avec Rotation (sources/codeRotation.ml)

```
1 Random.self_init ();;
2
3 let size = 4 ;;
4 let nbColor = 3 ;;
5
6 type tile = {
7   id: int;
8   top: int;
9   right: int;
10  bottom: int;
11  left: int;
12 } ;;
13
14 let rotate_tile_right t:tile =
15 {
16   id=t.id;
17   top = t.left;
18   bottom = t.right;
19   left = t.bottom;
20   right = t.top;
21 } ;;
22
23 let rotate_tile_180 t:tile =
24 {
25   id=t.id;
26   top = t.bottom;
27   bottom = t.top;
28   left = t.right;
29   right = t.left;
30 } ;;
31
32
33 let rotate_tile_left t:tile =
34 {
35   id=t.id;
36   right = t.bottom;
37   top = t.right;
38   bottom = t.left;
39   left = t.top;
40 } ;;
41
42 let create_random_tile () tile =
43 {
44   id=0;
45   top = Random.int nbColor;
46   bottom = Random.int nbColor;
47   left = Random.int nbColor;
48   right = Random.int nbColor;
49 } ;;
50
51
52 let create_board size =
53   Array.init size (fun _ ->
54     Array.init size (fun _ ->
```



```

55     { id=(-1); top=0; right=0; left=0; bottom=0}
56     )
57     ) ;;
58
59 let init_board size =
60   let board = create_board size in
61   for i = 0 to size - 1 do
62     if i mod 2 = 0 then
63       for j = 0 to size-1 do
64         board.(i).(j) <-
65           {
66             top = if i = 0 then 0 else board.(i-1).(j).bottom;
67             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
68             left = if j = 0 then 0 else board.(i).(j - 1).right;
69             right = if j = size - 1 then 0 else Random.int nbColor + 1;
70             id = i*size+j;
71           };
72       done
73     else
74       for j = size-1 downto 0 do
75         board.(i).(j) <-
76           {
77             top = board.(i-1).(j).bottom;
78             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
79             left = if j = 0 then 0 else Random.int nbColor + 1;
80             right = if j = size - 1 then 0 else board.(i).(j + 1).left;
81             id = i*size+j;
82           };
83       done;
84     done;
85   board;;
86
87 let print_board = fun board ->
88   Array.iter (fun row ->
89     Array.iter (fun tile ->
90       Printf.printf "+-----";
91     ) row;
92     Printf.printf "+\n";
93     Array.iter (fun tile ->
94       Printf.printf "| %02d " tile.top;
95     ) row;
96     Printf.printf "|\n";
97     Array.iter (fun tile ->
98       Printf.printf "|%02d %04d %02d" tile.left tile.id tile.right;
99     ) row;
100    Printf.printf "|\n";
101    Array.iter (fun tile ->
102      Printf.printf "| %02d " tile.bottom;
103    ) row;
104    Printf.printf "|\n";
105  ) board;
106  Printf.printf "+";
107  Array.iter (fun _ ->
108    Printf.printf "-----+";

```

```

109   ) board;
110   Printf.printf "\n" ;;
111
112
113   let shuffle_board board =
114     for i = 0 to size - 1 do
115       for j = 0 to size - 1 do
116         let random_i = Random.int size in
117         let random_j = Random.int size in
118         let temp_tile = board.(i).(j) in
119         board.(i).(j) <- board.(random_i).(random_j);
120         board.(random_i).(random_j) <- temp_tile;
121       done;
122     done;
123     board ;;
124
125   let init_tiles board=
126     let tiles = Array.make (size*size) ({ id=(-1); top=0; right=0; left=0; bottom=0}) in
127     for i = 0 to size - 1 do
128       for j = 0 to size - 1 do
129         tiles.((i)*size+j) <- board.(i).(j);
130       done;
131     done;
132     tiles ;;
133
134   let rec solve_backtrack board tiles i j =
135
136     let test_tile t i j k =
137       let res = ref true in
138
139       let test_tile = ref tiles.(t) in
140
141       if k = 1 then
142         test_tile := rotate_tile_left !test_tile
143       else if k = 2 then
144         test_tile := rotate_tile_180 !test_tile
145       else if k = 3 then
146         test_tile := rotate_tile_right !test_tile;
147
148       if j != 0 && board.(i).(j - 1).right != !test_tile.left then
149         res := false
150       else if j != 0 && !test_tile.left = 0 then
151         res := false
152       else if j != (size - 1) && !test_tile.right = 0 then
153         res := false
154       else if i != 0 && board.(i - 1).(j).bottom != !test_tile.top then
155         res := false
156       else if i != 0 && !test_tile.top = 0 then
157         res := false
158       else if i != (size - 1) && !test_tile.bottom = 0 then
159         res := false
160       else if i = 0 && (!test_tile.top != 0) then
161         res := false
162       else if i = (size - 1) && (!test_tile.bottom != 0) then

```

```

163     res := false
164     else if j = 0 && (!test_tile.left != 0) then
165         res := false
166     else if j = (size - 1) && (!test_tile.right != 0) then
167         res := false;
168     !res;
169 in
170
171 if board.(i).(j).id = -1 then
172     for t = 0 to (Array.length tiles) - 1 do
173         if tiles.(t).id != -1 then
174             for k = 0 to 3 do
175                 if test_tile t i j k then
176                     begin
177                         if k=0 then
178                             board.(i).(j) <- tiles.(t)
179                         else if k = 1 then
180                             board.(i).(j) <- rotate_tile_left tiles.(t)
181                         else if k = 2 then
182                             board.(i).(j) <- rotate_tile_180 tiles.(t)
183                         else
184                             board.(i).(j) <- rotate_tile_right tiles.(t);
185
186                     tiles.(t) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0 };
187                     if j < size - 1 then
188                         solve_backtrack board tiles i (j+1)
189                     else if i < size - 1 then
190                         solve_backtrack board tiles (i+1) 0
191                     else
192                         print_board board;
193
194                     if k=0 then
195                         tiles.(t) <- board.(i).(j)
196                     else if k = 1 then
197                         tiles.(t) <- rotate_tile_right board.(i).(j)
198                     else if k = 2 then
199                         tiles.(t) <- rotate_tile_180 board.(i).(j)
200                     else
201                         tiles.(t) <- rotate_tile_left board.(i).(j);
202                     board.(i).(j) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0 };
203                 end
204             done
205         done
206     else if j < size - 1 then
207         solve_backtrack board tiles i (j+1)
208     else if i < size - 1 then
209         solve_backtrack board tiles (i+1) 0
210     else
211         print_board board;
212 ;;
213
214 (* main *)
215
216

```

```

217 let b = init_board size ;;
218 Printf.printf "Initial board\n" ;;
219 print_board b ;;
220
221 Printf.printf "\n\nShuffled board\n" ;;
222 let b = shuffle_board b ;;
223 print_board b ;;
224
225 Printf.printf "\n\nSolving: may the time be with you\n";;
226 let tiles = init_tiles b;;
227 let board = create_board size;;
228 solve_backtrack board tiles 0 0;
229 Printf.printf "\n\nSolving: end of time\n" ;

```

Listing 1.10: Code avec Rotation et première tuile Locked (sources/codeRotation1stTileLocked.ml)

```

1 Random.self_init ();;
2
3 let size = 4 ;;
4 let nbColor = 5 ;;
5
6 type tile = {
7   id: int;
8   top: int;
9   right: int;
10  bottom: int;
11  left: int;
12 } ;;
13
14 let rotate_tile_right t:tile =
15   {
16     id=t.id;
17     top = t.left;
18     bottom = t.right;
19     left = t.bottom;
20     right = t.top;
21   } ;;
22
23 let rotate_tile_180 t:tile =
24   {
25     id=t.id;
26     top = t.bottom;
27     bottom = t.top;
28     left = t.right;
29     right = t.left;
30   } ;;
31
32 let rotate_tile_left t:tile =
33   {
34     id=t.id;
35     right = t.bottom;
36     top = t.right;
37     bottom = t.left;

```

```

38     left = t.top;
39 } ;;
40
41 let create_board size =
42   Array.init size (fun _ ->
43     Array.init size (fun _ ->
44       { id=(-1); top=0; right=0; left=0; bottom=0}
45     )
46   ) ;;
47
48 let init_board size =
49   let board = create_board size in
50   for i = 0 to size - 1 do
51     if i mod 2 = 0 then
52       for j = 0 to size-1 do
53         board.(i).(j) <-
54           {
55             top = if i = 0 then 0 else board.(i-1).(j).bottom;
56             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
57             left = if j = 0 then 0 else board.(i).(j - 1).right;
58             right = if j = size - 1 then 0 else Random.int nbColor + 1;
59             id = i*size+j;
60           };
61       done
62     else
63       for j = size-1 downto 0 do
64         board.(i).(j) <-
65           {
66             top = board.(i-1).(j).bottom;
67             bottom = if i = size-1 then 0 else Random.int nbColor + 1;
68             left = if j = 0 then 0 else Random.int nbColor + 1;
69             right = if j = size - 1 then 0 else board.(i).(j + 1).left;
70             id = i*size+j;
71           };
72       done;
73   done;
74   board;;
75
76 let print_board = fun board ->
77   Array.iter (fun row ->
78     Array.iter (fun tile ->
79       Printf.printf "+-----";
80     ) row;
81     Printf.printf "+\n";
82     Array.iter (fun tile ->
83       Printf.printf "| %02d " tile.top;
84     ) row;
85     Printf.printf "|\n";
86     Array.iter (fun tile ->
87       Printf.printf "|%02d %04d %02d" tile.left tile.id tile.right;
88     ) row;
89     Printf.printf "|\n";
90     Array.iter (fun tile ->
91       Printf.printf "| %02d " tile.bottom;

```

```

92     ) row;
93     Printf.printf "|\\n";
94 ) board;
95 Printf.printf "+";
96 Array.iter (fun _ ->
97     Printf.printf "-----+";
98 ) board;
99 Printf.printf "\\n" ;;
100
101
102 let shuffle_board board =
103     for i = 0 to size - 1 do
104         for j = 0 to size - 1 do
105             if not (i = 0 && j = 0) then
106                 begin
107                     let random_i = (Random.int (size-1)) + 1 in
108                     let random_j = (Random.int (size-1)) + 1 in
109                     let temp_tile = board.(i).(j) in
110                     board.(i).(j) <- board.(random_i).(random_j);
111                     board.(random_i).(random_j) <- temp_tile;
112                 end;
113             done;
114         done;
115     board ;;
116
117 let init_tiles board =
118     let tiles = Array.make ((size*size)) ({ id=(-1); top=0; right=0; left=0; bottom=0}) in
119     for i = 0 to size - 1 do
120         for j = 0 to size - 1 do
121             tiles.((i)*size+j) <- board.(i).(j);
122         done;
123     done;
124     tiles ;;
125
126 let rec solve_backtrack board tiles i j =
127
128     let test_tile t i j k =
129         let res = ref true in
130
131         let test_tile = ref tiles.(t) in
132
133         if k = 1 then
134             test_tile := rotate_tile_left !test_tile
135         else if k = 2 then
136             test_tile := rotate_tile_180 !test_tile
137         else if k = 3 then
138             test_tile := rotate_tile_right !test_tile;
139
140         if j != 0 && board.(i).(j - 1).right != !test_tile.left then
141             res := false
142         else if j != 0 && !test_tile.left = 0 then
143             res := false
144         else if j != (size - 1) && !test_tile.right = 0 then
145             res := false

```

```

146     else if i != 0 && board.(i - 1).(j).bottom != !test_tile.top then
147         res := false
148     else if i != 0 && !test_tile.top = 0 then
149         res := false
150     else if i != (size - 1) && !test_tile.bottom = 0 then
151         res := false
152     else if i = 0 && (!test_tile.top != 0) then
153         res := false
154     else if i = (size - 1) && (!test_tile.bottom != 0) then
155         res := false
156     else if j = 0 && (!test_tile.left != 0) then
157         res := false
158     else if j = (size - 1) && (!test_tile.right != 0) then
159         res := false;
160     !res;
161 in
162
163 if board.(i).(j).id = -1 then
164     for t = 1 to (Array.length tiles) - 1 do
165         if tiles.(t).id != -1 then
166             for k = 0 to 3 do
167                 if test_tile t i j k then
168                     begin
169                         if k=0 then
170                             board.(i).(j) <- tiles.(t)
171                         else if k = 1 then
172                             board.(i).(j) <- rotate_tile_left tiles.(t)
173                         else if k = 2 then
174                             board.(i).(j) <- rotate_tile_180 tiles.(t)
175                         else
176                             board.(i).(j) <- rotate_tile_right tiles.(t);
177
178                     tiles.(t) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0 };
179                     if j < size - 1 then
180                         solve_backtrack board tiles i (j+1)
181                     else if i < size - 1 then
182                         solve_backtrack board tiles (i+1) 0
183                     else
184                         print_board board;
185
186                     if k=0 then
187                         tiles.(t) <- board.(i).(j)
188                     else if k = 1 then
189                         tiles.(t) <- rotate_tile_right board.(i).(j)
190                     else if k = 2 then
191                         tiles.(t) <- rotate_tile_180 board.(i).(j)
192                     else
193                         tiles.(t) <- rotate_tile_left board.(i).(j);
194                     board.(i).(j) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0 };
195                 end
196             done
197         done
198     else if j < size - 1 then
199         solve_backtrack board tiles i (j+1)

```

```

200     else if i < size - 1 then
201         solve_backtrack board tiles (i+1) 0
202     else
203         print_board board;
204 ;;
205
206
207 (* main *)
208
209 let b = init_board size ;;
210 Printf.printf "Initial board\n" ;;
211 print_board b ;;
212
213 Printf.printf "\n\nShuffled board\n" ;;
214 let b = shuffle_board b ;;
215 print_board b ;;
216
217 Printf.printf "\n\nSolving: may the time be with you\n";;
218 let tiles = init_tiles b;;
219 let board = create_board size;;
220 board.(0).(0) <- tiles.(0);;
221 tiles.(0) <- { id = -1; top = 0; right = 0; bottom = 0; left = 0 };;
222 solve_backtrack board tiles 0 0;;
223 Printf.printf "\n\nSolving: end of time\n" ;;

```

Listing 1.11: Résultat avec Rotation (sources/resR_5x5.txt)

```

1 Initial board
2 <deb>
3 +-----+-----+-----+-----+-----+
4 | 00 | 00 | 00 | 00 | 00 |
5 |00 0000 05|05 0001 03|03 0002 02|02 0003 04|04 0004 00|
6 | 02 | 05 | 04 | 02 | 02 |
7 +-----+-----+-----+-----+-----+
8 | 02 | 05 | 04 | 02 | 02 |
9 |00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
10 | 04 | 05 | 02 | 05 | 05 |
11 +-----+-----+-----+-----+-----+
12 | 04 | 05 | 02 | 05 | 05 |
13 |00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0014 00|
14 | 03 | 04 | 02 | 01 | 03 |
15 +-----+-----+-----+-----+-----+
16 | 03 | 04 | 02 | 01 | 03 |
17 |00 0015 01|01 0016 01|01 0017 04|04 0018 01|01 0019 00|
18 | 05 | 02 | 04 | 01 | 01 |
19 +-----+-----+-----+-----+-----+
20 | 05 | 02 | 04 | 01 | 01 |
21 |00 0020 04|04 0021 03|03 0022 03|03 0023 02|02 0024 00|
22 | 00 | 00 | 00 | 00 | 00 |
23 +-----+-----+-----+-----+-----+
24
25
26
27 Shuffled board

```



```

28 <deb>
29 +-----+-----+-----+-----+-----+
30 | 05 | 02 | 04 | 05 | 02 |
31 |00 0020 04|05 0009 00|00 0010 01|01 0011 05|00 0005 04|
32 | 00 | 05 | 03 | 04 | 04 |
33 +-----+-----+-----+-----+-----+
34 | 05 | 01 | 00 | 02 | 04 |
35 |02 0013 01|03 0023 02|02 0003 04|01 0017 04|05 0007 05|
36 | 01 | 00 | 02 | 04 | 02 |
37 +-----+-----+-----+-----+-----+
38 | 00 | 04 | 03 | 02 | 00 |
39 |03 0002 02|03 0022 03|00 0015 01|05 0008 05|04 0004 00|
40 | 04 | 00 | 05 | 05 | 02 |
41 +-----+-----+-----+-----+-----+
42 | 00 | 01 | 00 | 02 | 05 |
43 |00 0000 05|02 0024 00|05 0001 03|04 0021 03|04 0006 05|
44 | 02 | 00 | 05 | 00 | 05 |
45 +-----+-----+-----+-----+-----+
46 | 02 | 04 | 03 | 05 | 01 |
47 |05 0012 02|01 0016 01|01 0019 00|01 0014 00|04 0018 01|
48 | 02 | 02 | 01 | 03 | 01 |
49 +-----+-----+-----+-----+-----+
50
51
52
53 Solving: may the time be with you
54 <deb>
55 +-----+-----+-----+-----+-----+
56 | 00 | 00 | 00 | 00 | 00 |
57 |00 0020 05|05 0015 03|03 0010 04|04 0005 02|02 0000 00|
58 | 04 | 01 | 01 | 04 | 05 |
59 +-----+-----+-----+-----+-----+
60 | 04 | 01 | 01 | 04 | 05 |
61 |00 0003 02|02 0016 04|04 0011 05|05 0006 05|05 0001 00|
62 | 02 | 01 | 05 | 05 | 03 |
63 +-----+-----+-----+-----+-----+
64 | 02 | 01 | 05 | 05 | 03 |
65 |00 0002 04|04 0017 02|02 0012 02|02 0007 04|04 0022 00|
66 | 03 | 04 | 02 | 05 | 03 |
67 +-----+-----+-----+-----+-----+
68 | 03 | 04 | 02 | 05 | 03 |
69 |00 0023 01|01 0018 01|01 0013 05|05 0008 02|02 0021 00|
70 | 02 | 01 | 01 | 05 | 04 |
71 +-----+-----+-----+-----+-----+
72 | 02 | 01 | 01 | 05 | 04 |
73 |00 0024 01|01 0019 03|03 0014 05|05 0009 02|02 0004 00|
74 | 00 | 00 | 00 | 00 | 00 |
75 +-----+-----+-----+-----+-----+
76
77 <deb>
78 +-----+-----+-----+-----+-----+
79 | 00 | 00 | 00 | 00 | 00 |
80 |00 0020 05|05 0015 03|03 0010 04|04 0005 02|02 0000 00|
81 | 04 | 01 | 01 | 04 | 05 |

```

```

82 | +-----+-----+-----+-----+-----+
83 | | 04 | 01 | 01 | 04 | 05 |
84 | |00 0021 02|02 0016 04|04 0011 05|05 0006 05|05 0001 00|
85 | | 03 | 01 | 05 | 05 | 03 |
86 | +-----+-----+-----+-----+-----+
87 | | 03 | 01 | 05 | 05 | 03 |
88 | |00 0022 04|04 0017 02|02 0012 02|02 0007 04|04 0002 00|
89 | | 03 | 04 | 02 | 05 | 02 |
90 | +-----+-----+-----+-----+-----+
91 | | 03 | 04 | 02 | 05 | 02 |
92 | |00 0023 01|01 0018 01|01 0013 05|05 0008 02|02 0003 00|
93 | | 02 | 01 | 01 | 05 | 04 |
94 | +-----+-----+-----+-----+-----+
95 | | 02 | 01 | 01 | 05 | 04 |
96 | |00 0024 01|01 0019 03|03 0014 05|05 0009 02|02 0004 00|
97 | | 00 | 00 | 00 | 00 | 00 |
98 | +-----+-----+-----+-----+-----+
99
100 | <deb>
101 | +-----+-----+-----+-----+-----+
102 | | 00 | 00 | 00 | 00 | 00 |
103 | |00 0020 05|05 0015 03|03 0022 03|03 0019 01|01 0024 00|
104 | | 04 | 01 | 04 | 01 | 02 |
105 | +-----+-----+-----+-----+-----+
106 | | 04 | 01 | 04 | 01 | 02 |
107 | |00 0021 02|02 0016 04|04 0017 01|01 0018 01|01 0023 00|
108 | | 03 | 01 | 02 | 04 | 03 |
109 | +-----+-----+-----+-----+-----+
110 | | 03 | 01 | 02 | 04 | 03 |
111 | |00 0014 01|01 0013 02|02 0012 05|05 0011 01|01 0010 00|
112 | | 05 | 05 | 02 | 05 | 04 |
113 | +-----+-----+-----+-----+-----+
114 | | 05 | 05 | 02 | 05 | 04 |
115 | |00 0009 05|05 0008 05|05 0007 05|05 0006 04|04 0005 00|
116 | | 02 | 02 | 04 | 05 | 02 |
117 | +-----+-----+-----+-----+-----+
118 | | 02 | 02 | 04 | 05 | 02 |
119 | |00 0004 04|04 0003 02|02 0002 03|03 0001 05|05 0000 00|
120 | | 00 | 00 | 00 | 00 | 00 |
121 | +-----+-----+-----+-----+-----+
122
123 | <deb>
124 | +-----+-----+-----+-----+-----+
125 | | 00 | 00 | 00 | 00 | 00 |
126 | |00 0020 05|05 0014 03|03 0010 04|04 0005 02|02 0000 00|
127 | | 04 | 01 | 01 | 04 | 05 |
128 | +-----+-----+-----+-----+-----+
129 | | 04 | 01 | 01 | 04 | 05 |
130 | |00 0003 02|02 0016 04|04 0011 05|05 0006 05|05 0001 00|
131 | | 02 | 01 | 05 | 05 | 03 |
132 | +-----+-----+-----+-----+-----+
133 | | 02 | 01 | 05 | 05 | 03 |
134 | |00 0002 04|04 0017 02|02 0012 02|02 0007 04|04 0022 00|
135 | | 03 | 04 | 02 | 05 | 03 |

```

```

136 +-----+-----+-----+-----+-----+
137 | 03 | 04 | 02 | 05 | 03 |
138 |00 0023 01|01 0018 01|01 0013 05|05 0008 02|02 0021 00|
139 | 02 | 01 | 01 | 05 | 04 |
140 +-----+-----+-----+-----+-----+
141 | 02 | 01 | 01 | 05 | 04 |
142 |00 0024 01|01 0019 03|03 0015 05|05 0009 02|02 0004 00|
143 | 00 | 00 | 00 | 00 | 00 |
144 +-----+-----+-----+-----+-----+
145
146 <deb>
147 +-----+-----+-----+-----+-----+
148 | 00 | 00 | 00 | 00 | 00 |
149 |00 0020 05|05 0014 03|03 0010 04|04 0005 02|02 0000 00|
150 | 04 | 01 | 01 | 04 | 05 |
151 +-----+-----+-----+-----+-----+
152 | 04 | 01 | 01 | 04 | 05 |
153 |00 0021 02|02 0016 04|04 0011 05|05 0006 05|05 0001 00|
154 | 03 | 01 | 05 | 05 | 03 |
155 +-----+-----+-----+-----+-----+
156 | 03 | 01 | 05 | 05 | 03 |
157 |00 0022 04|04 0017 02|02 0012 02|02 0007 04|04 0002 00|
158 | 03 | 04 | 02 | 05 | 02 |
159 +-----+-----+-----+-----+-----+
160 | 03 | 04 | 02 | 05 | 02 |
161 |00 0023 01|01 0018 01|01 0013 05|05 0008 02|02 0003 00|
162 | 02 | 01 | 01 | 05 | 04 |
163 +-----+-----+-----+-----+-----+
164 | 02 | 01 | 01 | 05 | 04 |
165 |00 0024 01|01 0019 03|03 0015 05|05 0009 02|02 0004 00|
166 | 00 | 00 | 00 | 00 | 00 |
167 +-----+-----+-----+-----+-----+
168
169 <deb>
170 +-----+-----+-----+-----+-----+
171 | 00 | 00 | 00 | 00 | 00 |
172 |00 0020 05|05 0014 03|03 0022 03|03 0019 01|01 0024 00|
173 | 04 | 01 | 04 | 01 | 02 |
174 +-----+-----+-----+-----+-----+
175 | 04 | 01 | 04 | 01 | 02 |
176 |00 0021 02|02 0016 04|04 0017 01|01 0018 01|01 0023 00|
177 | 03 | 01 | 02 | 04 | 03 |
178 +-----+-----+-----+-----+-----+
179 | 03 | 01 | 02 | 04 | 03 |
180 |00 0015 01|01 0013 02|02 0012 05|05 0011 01|01 0010 00|
181 | 05 | 05 | 02 | 05 | 04 |
182 +-----+-----+-----+-----+-----+
183 | 05 | 05 | 02 | 05 | 04 |
184 |00 0009 05|05 0008 05|05 0007 05|05 0006 04|04 0005 00|
185 | 02 | 02 | 04 | 05 | 02 |
186 +-----+-----+-----+-----+-----+
187 | 02 | 02 | 04 | 05 | 02 |
188 |00 0004 04|04 0003 02|02 0002 03|03 0001 05|05 0000 00|
189 | 00 | 00 | 00 | 00 | 00 |

```

```

190 +-----+-----+-----+-----+-----+
191
192 <deb>
193 +-----+-----+-----+-----+-----+
194 | 00 | 00 | 00 | 00 | 00 |
195 |00 0004 02|02 0009 05|05 0015 03|03 0021 04|04 0020 00|
196 | 04 | 05 | 01 | 02 | 05 |
197 +-----+-----+-----+-----+-----+
198 | 04 | 05 | 01 | 02 | 05 |
199 |00 0003 02|02 0008 05|05 0013 01|01 0016 01|01 0014 00|
200 | 02 | 05 | 02 | 04 | 03 |
201 +-----+-----+-----+-----+-----+
202 | 02 | 05 | 02 | 04 | 03 |
203 |00 0002 04|04 0007 02|02 0012 02|02 0017 04|04 0022 00|
204 | 03 | 05 | 05 | 01 | 03 |
205 +-----+-----+-----+-----+-----+
206 | 03 | 05 | 05 | 01 | 03 |
207 |00 0001 05|05 0006 05|05 0011 04|04 0018 01|01 0019 00|
208 | 05 | 04 | 01 | 01 | 01 |
209 +-----+-----+-----+-----+-----+
210 | 05 | 04 | 01 | 01 | 01 |
211 |00 0000 02|02 0005 04|04 0010 03|03 0023 02|02 0024 00|
212 | 00 | 00 | 00 | 00 | 00 |
213 +-----+-----+-----+-----+-----+
214
215 <deb>
216 +-----+-----+-----+-----+-----+
217 | 00 | 00 | 00 | 00 | 00 |
218 |00 0004 02|02 0009 05|05 0015 03|03 0019 01|01 0024 00|
219 | 04 | 05 | 01 | 01 | 02 |
220 +-----+-----+-----+-----+-----+
221 | 04 | 05 | 01 | 01 | 02 |
222 |00 0003 02|02 0008 05|05 0013 01|01 0018 01|01 0023 00|
223 | 02 | 05 | 02 | 04 | 03 |
224 +-----+-----+-----+-----+-----+
225 | 02 | 05 | 02 | 04 | 03 |
226 |00 0002 04|04 0007 02|02 0012 02|02 0017 04|04 0022 00|
227 | 03 | 05 | 05 | 01 | 03 |
228 +-----+-----+-----+-----+-----+
229 | 03 | 05 | 05 | 01 | 03 |
230 |00 0001 05|05 0006 05|05 0011 04|04 0016 02|02 0021 00|
231 | 05 | 04 | 01 | 01 | 04 |
232 +-----+-----+-----+-----+-----+
233 | 05 | 04 | 01 | 01 | 04 |
234 |00 0000 02|02 0005 04|04 0010 03|03 0014 05|05 0020 00|
235 | 00 | 00 | 00 | 00 | 00 |
236 +-----+-----+-----+-----+-----+
237
238 <deb>
239 +-----+-----+-----+-----+-----+
240 | 00 | 00 | 00 | 00 | 00 |
241 |00 0004 02|02 0009 05|05 0015 03|03 0019 01|01 0024 00|
242 | 04 | 05 | 01 | 01 | 02 |
243 +-----+-----+-----+-----+-----+

```

```

244 | 04 | 05 | 01 | 01 | 02 |
245 |00 0021 02|02 0008 05|05 0013 01|01 0018 01|01 0023 00|
246 | 03 | 05 | 02 | 04 | 03 |
247 +-----+-----+-----+-----+
248 | 03 | 05 | 02 | 04 | 03 |
249 |00 0022 04|04 0007 02|02 0012 02|02 0017 04|04 0002 00|
250 | 03 | 05 | 05 | 01 | 02 |
251 +-----+-----+-----+-----+
252 | 03 | 05 | 05 | 01 | 02 |
253 |00 0001 05|05 0006 05|05 0011 04|04 0016 02|02 0003 00|
254 | 05 | 04 | 01 | 01 | 04 |
255 +-----+-----+-----+-----+
256 | 05 | 04 | 01 | 01 | 04 |
257 |00 0000 02|02 0005 04|04 0010 03|03 0014 05|05 0020 00|
258 | 00 | 00 | 00 | 00 | 00 |
259 +-----+-----+-----+-----+
260
261 <deb>
262 +-----+-----+-----+-----+
263 | 00 | 00 | 00 | 00 | 00 |
264 |00 0004 02|02 0009 05|05 0014 03|03 0021 04|04 0020 00|
265 | 04 | 05 | 01 | 02 | 05 |
266 +-----+-----+-----+-----+
267 | 04 | 05 | 01 | 02 | 05 |
268 |00 0003 02|02 0008 05|05 0013 01|01 0016 01|01 0015 00|
269 | 02 | 05 | 02 | 04 | 03 |
270 +-----+-----+-----+-----+
271 | 02 | 05 | 02 | 04 | 03 |
272 |00 0002 04|04 0007 02|02 0012 02|02 0017 04|04 0022 00|
273 | 03 | 05 | 05 | 01 | 03 |
274 +-----+-----+-----+-----+
275 | 03 | 05 | 05 | 01 | 03 |
276 |00 0001 05|05 0006 05|05 0011 04|04 0018 01|01 0019 00|
277 | 05 | 04 | 01 | 01 | 01 |
278 +-----+-----+-----+-----+
279 | 05 | 04 | 01 | 01 | 01 |
280 |00 0000 02|02 0005 04|04 0010 03|03 0023 02|02 0024 00|
281 | 00 | 00 | 00 | 00 | 00 |
282 +-----+-----+-----+-----+
283
284 <deb>
285 +-----+-----+-----+-----+
286 | 00 | 00 | 00 | 00 | 00 |
287 |00 0004 02|02 0009 05|05 0014 03|03 0019 01|01 0024 00|
288 | 04 | 05 | 01 | 01 | 02 |
289 +-----+-----+-----+-----+
290 | 04 | 05 | 01 | 01 | 02 |
291 |00 0003 02|02 0008 05|05 0013 01|01 0018 01|01 0023 00|
292 | 02 | 05 | 02 | 04 | 03 |
293 +-----+-----+-----+-----+
294 | 02 | 05 | 02 | 04 | 03 |
295 |00 0002 04|04 0007 02|02 0012 02|02 0017 04|04 0022 00|
296 | 03 | 05 | 05 | 01 | 03 |
297 +-----+-----+-----+-----+

```

```

298 | 03 | 05 | 05 | 01 | 03 |
299 |00 0001 05|05 0006 05|05 0011 04|04 0016 02|02 0021 00|
300 | 05 | 04 | 01 | 01 | 04 |
301 +-----+-----+-----+-----+-----+
302 | 05 | 04 | 01 | 01 | 04 |
303 |00 0000 02|02 0005 04|04 0010 03|03 0015 05|05 0020 00|
304 | 00 | 00 | 00 | 00 | 00 |
305 +-----+-----+-----+-----+-----+
306
307 <deb>
308 +-----+-----+-----+-----+-----+
309 | 00 | 00 | 00 | 00 | 00 |
310 |00 0004 02|02 0009 05|05 0014 03|03 0019 01|01 0024 00|
311 | 04 | 05 | 01 | 01 | 02 |
312 +-----+-----+-----+-----+-----+
313 | 04 | 05 | 01 | 01 | 02 |
314 |00 0021 02|02 0008 05|05 0013 01|01 0018 01|01 0023 00|
315 | 03 | 05 | 02 | 04 | 03 |
316 +-----+-----+-----+-----+-----+
317 | 03 | 05 | 02 | 04 | 03 |
318 |00 0022 04|04 0007 02|02 0012 02|02 0017 04|04 0002 00|
319 | 03 | 05 | 05 | 01 | 02 |
320 +-----+-----+-----+-----+-----+
321 | 03 | 05 | 05 | 01 | 02 |
322 |00 0001 05|05 0006 05|05 0011 04|04 0016 02|02 0003 00|
323 | 05 | 04 | 01 | 01 | 04 |
324 +-----+-----+-----+-----+-----+
325 | 05 | 04 | 01 | 01 | 04 |
326 |00 0000 02|02 0005 04|04 0010 03|03 0015 05|05 0020 00|
327 | 00 | 00 | 00 | 00 | 00 |
328 +-----+-----+-----+-----+-----+
329
330 <deb>
331 +-----+-----+-----+-----+-----+
332 | 00 | 00 | 00 | 00 | 00 |
333 |00 0000 05|05 0001 03|03 0002 02|02 0003 04|04 0004 00|
334 | 02 | 05 | 04 | 02 | 02 |
335 +-----+-----+-----+-----+-----+
336 | 02 | 05 | 04 | 02 | 02 |
337 |00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
338 | 04 | 05 | 02 | 05 | 05 |
339 +-----+-----+-----+-----+-----+
340 | 04 | 05 | 02 | 05 | 05 |
341 |00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0015 00|
342 | 03 | 04 | 02 | 01 | 03 |
343 +-----+-----+-----+-----+-----+
344 | 03 | 04 | 02 | 01 | 03 |
345 |00 0023 01|01 0018 01|01 0017 04|04 0016 02|02 0021 00|
346 | 02 | 01 | 04 | 01 | 04 |
347 +-----+-----+-----+-----+-----+
348 | 02 | 01 | 04 | 01 | 04 |
349 |00 0024 01|01 0019 03|03 0022 03|03 0014 05|05 0020 00|
350 | 00 | 00 | 00 | 00 | 00 |
351 +-----+-----+-----+-----+-----+

```

```

352
353 <deb>
354 +-----+-----+-----+-----+
355 | 00 | 00 | 00 | 00 | 00 |
356 |00 0000 05|05 0001 03|03 0002 02|02 0003 04|04 0004 00|
357 | 02 | 05 | 04 | 02 | 02 |
358 +-----+-----+-----+-----+
359 | 02 | 05 | 04 | 02 | 02 |
360 |00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
361 | 04 | 05 | 02 | 05 | 05 |
362 +-----+-----+-----+-----+
363 | 04 | 05 | 02 | 05 | 05 |
364 |00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0015 00|
365 | 03 | 04 | 02 | 01 | 03 |
366 +-----+-----+-----+-----+
367 | 03 | 04 | 02 | 01 | 03 |
368 |00 0014 01|01 0016 01|01 0017 04|04 0018 01|01 0019 00|
369 | 05 | 02 | 04 | 01 | 01 |
370 +-----+-----+-----+-----+
371 | 05 | 02 | 04 | 01 | 01 |
372 |00 0020 04|04 0021 03|03 0022 03|03 0023 02|02 0024 00|
373 | 00 | 00 | 00 | 00 | 00 |
374 +-----+-----+-----+-----+
375
376 <deb>
377 +-----+-----+-----+-----+
378 | 00 | 00 | 00 | 00 | 00 |
379 |00 0000 05|05 0001 03|03 0002 02|02 0003 04|04 0004 00|
380 | 02 | 05 | 04 | 02 | 02 |
381 +-----+-----+-----+-----+
382 | 02 | 05 | 04 | 02 | 02 |
383 |00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
384 | 04 | 05 | 02 | 05 | 05 |
385 +-----+-----+-----+-----+
386 | 04 | 05 | 02 | 05 | 05 |
387 |00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0014 00|
388 | 03 | 04 | 02 | 01 | 03 |
389 +-----+-----+-----+-----+
390 | 03 | 04 | 02 | 01 | 03 |
391 |00 0023 01|01 0018 01|01 0017 04|04 0016 02|02 0021 00|
392 | 02 | 01 | 04 | 01 | 04 |
393 +-----+-----+-----+-----+
394 | 02 | 01 | 04 | 01 | 04 |
395 |00 0024 01|01 0019 03|03 0022 03|03 0015 05|05 0020 00|
396 | 00 | 00 | 00 | 00 | 00 |
397 +-----+-----+-----+-----+
398
399 <deb>
400 +-----+-----+-----+-----+
401 | 00 | 00 | 00 | 00 | 00 |
402 |00 0000 05|05 0001 03|03 0002 02|02 0003 04|04 0004 00|
403 | 02 | 05 | 04 | 02 | 02 |
404 +-----+-----+-----+-----+
405 | 02 | 05 | 04 | 02 | 02 |

```

```

406 | 00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
407 | 04 | 05 | 02 | 05 | 05 |
408 +-----+-----+-----+-----+-----+
409 | 04 | 05 | 02 | 05 | 05 |
410 | 00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0014 00|
411 | 03 | 04 | 02 | 01 | 03 |
412 +-----+-----+-----+-----+-----+
413 | 03 | 04 | 02 | 01 | 03 |
414 | 00 0015 01|01 0016 01|01 0017 04|04 0018 01|01 0019 00|
415 | 05 | 02 | 04 | 01 | 01 |
416 +-----+-----+-----+-----+-----+
417 | 05 | 02 | 04 | 01 | 01 |
418 | 00 0020 04|04 0021 03|03 0022 03|03 0023 02|02 0024 00|
419 | 00 | 00 | 00 | 00 | 00 |
420 +-----+-----+-----+-----+-----+
421
422 <deb>
423 +-----+-----+-----+-----+-----+
424 | 00 | 00 | 00 | 00 | 00 |
425 | 00 0000 05|05 0001 03|03 0022 03|03 0021 04|04 0004 00|
426 | 02 | 05 | 04 | 02 | 02 |
427 +-----+-----+-----+-----+-----+
428 | 02 | 05 | 04 | 02 | 02 |
429 | 00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
430 | 04 | 05 | 02 | 05 | 05 |
431 +-----+-----+-----+-----+-----+
432 | 04 | 05 | 02 | 05 | 05 |
433 | 00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0015 00|
434 | 03 | 04 | 02 | 01 | 03 |
435 +-----+-----+-----+-----+-----+
436 | 03 | 04 | 02 | 01 | 03 |
437 | 00 0014 01|01 0016 01|01 0017 04|04 0018 01|01 0019 00|
438 | 05 | 02 | 04 | 01 | 01 |
439 +-----+-----+-----+-----+-----+
440 | 05 | 02 | 04 | 01 | 01 |
441 | 00 0020 04|04 0003 02|02 0002 03|03 0023 02|02 0024 00|
442 | 00 | 00 | 00 | 00 | 00 |
443 +-----+-----+-----+-----+-----+
444
445 <deb>
446 +-----+-----+-----+-----+-----+
447 | 00 | 00 | 00 | 00 | 00 |
448 | 00 0000 05|05 0001 03|03 0022 03|03 0021 04|04 0004 00|
449 | 02 | 05 | 04 | 02 | 02 |
450 +-----+-----+-----+-----+-----+
451 | 02 | 05 | 04 | 02 | 02 |
452 | 00 0005 04|04 0006 05|05 0007 05|05 0008 05|05 0009 00|
453 | 04 | 05 | 02 | 05 | 05 |
454 +-----+-----+-----+-----+-----+
455 | 04 | 05 | 02 | 05 | 05 |
456 | 00 0010 01|01 0011 05|05 0012 02|02 0013 01|01 0014 00|
457 | 03 | 04 | 02 | 01 | 03 |
458 +-----+-----+-----+-----+-----+
459 | 03 | 04 | 02 | 01 | 03 |

```



```

460 |00 0015 01|01 0016 01|01 0017 04|04 0018 01|01 0019 00|
461 | 05 | 02 | 04 | 01 | 01 |
462 +-----+-----+-----+-----+
463 | 05 | 02 | 04 | 01 | 01 |
464 |00 0020 04|04 0003 02|02 0002 03|03 0023 02|02 0024 00|
465 | 00 | 00 | 00 | 00 | 00 |
466 +-----+-----+-----+-----+
467
468 <deb>
469 +-----+-----+-----+-----+
470 | 00 | 00 | 00 | 00 | 00 |
471 |00 0024 02|02 0023 03|03 0010 04|04 0005 02|02 0000 00|
472 | 01 | 01 | 01 | 04 | 05 |
473 +-----+-----+-----+-----+
474 | 01 | 01 | 01 | 04 | 05 |
475 |00 0019 01|01 0018 04|04 0011 05|05 0006 05|05 0001 00|
476 | 03 | 01 | 05 | 05 | 03 |
477 +-----+-----+-----+-----+
478 | 03 | 01 | 05 | 05 | 03 |
479 |00 0022 04|04 0017 02|02 0012 02|02 0007 04|04 0002 00|
480 | 03 | 04 | 02 | 05 | 02 |
481 +-----+-----+-----+-----+
482 | 03 | 04 | 02 | 05 | 02 |
483 |00 0015 01|01 0016 01|01 0013 05|05 0008 02|02 0003 00|
484 | 05 | 02 | 01 | 05 | 04 |
485 +-----+-----+-----+-----+
486 | 05 | 02 | 01 | 05 | 04 |
487 |00 0020 04|04 0021 03|03 0014 05|05 0009 02|02 0004 00|
488 | 00 | 00 | 00 | 00 | 00 |
489 +-----+-----+-----+-----+
490
491 <deb>
492 +-----+-----+-----+-----+
493 | 00 | 00 | 00 | 00 | 00 |
494 |00 0024 02|02 0023 03|03 0010 04|04 0005 02|02 0000 00|
495 | 01 | 01 | 01 | 04 | 05 |
496 +-----+-----+-----+-----+
497 | 01 | 01 | 01 | 04 | 05 |
498 |00 0019 01|01 0018 04|04 0011 05|05 0006 05|05 0001 00|
499 | 03 | 01 | 05 | 05 | 03 |
500 +-----+-----+-----+-----+
501 | 03 | 01 | 05 | 05 | 03 |
502 |00 0022 04|04 0017 02|02 0012 02|02 0007 04|04 0002 00|
503 | 03 | 04 | 02 | 05 | 02 |
504 +-----+-----+-----+-----+
505 | 03 | 04 | 02 | 05 | 02 |
506 |00 0014 01|01 0016 01|01 0013 05|05 0008 02|02 0003 00|
507 | 05 | 02 | 01 | 05 | 04 |
508 +-----+-----+-----+-----+
509 | 05 | 02 | 01 | 05 | 04 |
510 |00 0020 04|04 0021 03|03 0015 05|05 0009 02|02 0004 00|
511 | 00 | 00 | 00 | 00 | 00 |
512 +-----+-----+-----+-----+
513

```

```

514 <deb>
515 +-----+-----+-----+-----+-----+
516 | 00 | 00 | 00 | 00 | 00 |
517 |00 0024 02|02 0023 03|03 0002 02|02 0003 04|04 0020 00|
518 | 01 | 01 | 04 | 02 | 05 |
519 +-----+-----+-----+-----+-----+
520 | 01 | 01 | 04 | 02 | 05 |
521 |00 0019 01|01 0018 04|04 0017 01|01 0016 01|01 0015 00|
522 | 03 | 01 | 02 | 04 | 03 |
523 +-----+-----+-----+-----+-----+
524 | 03 | 01 | 02 | 04 | 03 |
525 |00 0014 01|01 0013 02|02 0012 05|05 0011 01|01 0010 00|
526 | 05 | 05 | 02 | 05 | 04 |
527 +-----+-----+-----+-----+-----+
528 | 05 | 05 | 02 | 05 | 04 |
529 |00 0009 05|05 0008 05|05 0007 05|05 0006 04|04 0005 00|
530 | 02 | 02 | 04 | 05 | 02 |
531 +-----+-----+-----+-----+-----+
532 | 02 | 02 | 04 | 05 | 02 |
533 |00 0004 04|04 0021 03|03 0022 03|03 0001 05|05 0000 00|
534 | 00 | 00 | 00 | 00 | 00 |
535 +-----+-----+-----+-----+-----+
536
537 <deb>
538 +-----+-----+-----+-----+-----+
539 | 00 | 00 | 00 | 00 | 00 |
540 |00 0024 02|02 0023 03|03 0002 02|02 0003 04|04 0020 00|
541 | 01 | 01 | 04 | 02 | 05 |
542 +-----+-----+-----+-----+-----+
543 | 01 | 01 | 04 | 02 | 05 |
544 |00 0019 01|01 0018 04|04 0017 01|01 0016 01|01 0014 00|
545 | 03 | 01 | 02 | 04 | 03 |
546 +-----+-----+-----+-----+-----+
547 | 03 | 01 | 02 | 04 | 03 |
548 |00 0015 01|01 0013 02|02 0012 05|05 0011 01|01 0010 00|
549 | 05 | 05 | 02 | 05 | 04 |
550 +-----+-----+-----+-----+-----+
551 | 05 | 05 | 02 | 05 | 04 |
552 |00 0009 05|05 0008 05|05 0007 05|05 0006 04|04 0005 00|
553 | 02 | 02 | 04 | 05 | 02 |
554 +-----+-----+-----+-----+-----+
555 | 02 | 02 | 04 | 05 | 02 |
556 |00 0004 04|04 0021 03|03 0022 03|03 0001 05|05 0000 00|
557 | 00 | 00 | 00 | 00 | 00 |
558 +-----+-----+-----+-----+-----+
559
560 <deb>
561 +-----+-----+-----+-----+-----+
562 | 00 | 00 | 00 | 00 | 00 |
563 |00 0024 02|02 0023 03|03 0022 03|03 0021 04|04 0020 00|
564 | 01 | 01 | 04 | 02 | 05 |
565 +-----+-----+-----+-----+-----+
566 | 01 | 01 | 04 | 02 | 05 |
567 |00 0019 01|01 0018 04|04 0017 01|01 0016 01|01 0015 00|

```

```

568 | 03 | 01 | 02 | 04 | 03 |
569 +-----+-----+-----+-----+
570 | 03 | 01 | 02 | 04 | 03 |
571 |00 0014 01|01 0013 02|02 0012 05|05 0011 01|01 0010 00|
572 | 05 | 05 | 02 | 05 | 04 |
573 +-----+-----+-----+-----+
574 | 05 | 05 | 02 | 05 | 04 |
575 |00 0009 05|05 0008 05|05 0007 05|05 0006 04|04 0005 00|
576 | 02 | 02 | 04 | 05 | 02 |
577 +-----+-----+-----+-----+
578 | 02 | 02 | 04 | 05 | 02 |
579 |00 0004 04|04 0003 02|02 0002 03|03 0001 05|05 0000 00|
580 | 00 | 00 | 00 | 00 | 00 |
581 +-----+-----+-----+-----+
582
583 <deb>
584 +-----+-----+-----+-----+
585 | 00 | 00 | 00 | 00 | 00 |
586 |00 0024 02|02 0023 03|03 0022 03|03 0021 04|04 0020 00|
587 | 01 | 01 | 04 | 02 | 05 |
588 +-----+-----+-----+-----+
589 | 01 | 01 | 04 | 02 | 05 |
590 |00 0019 01|01 0018 04|04 0017 01|01 0016 01|01 0014 00|
591 | 03 | 01 | 02 | 04 | 03 |
592 +-----+-----+-----+-----+
593 | 03 | 01 | 02 | 04 | 03 |
594 |00 0015 01|01 0013 02|02 0012 05|05 0011 01|01 0010 00|
595 | 05 | 05 | 02 | 05 | 04 |
596 +-----+-----+-----+-----+
597 | 05 | 05 | 02 | 05 | 04 |
598 |00 0009 05|05 0008 05|05 0007 05|05 0006 04|04 0005 00|
599 | 02 | 02 | 04 | 05 | 02 |
600 +-----+-----+-----+-----+
601 | 02 | 02 | 04 | 05 | 02 |
602 |00 0004 04|04 0003 02|02 0002 03|03 0001 05|05 0000 00|
603 | 00 | 00 | 00 | 00 | 00 |
604 +-----+-----+-----+-----+
605
606
607
608 Solving: end of time

```