INFORMATICS PROJECT REPORT POLYOMINO TILINGS AND EXACT COVER

Quang Anh NGUYEN, Anh Duc HA

30 january 2020

Part 1
POLYOMINOES

1.1 Manipulate polyominoes

Task 1

A polyomino is represented by a `LinkedList` of `Square`, which includes two integer coordinates, along with neighborhood and inclusion relations.

Initially, we hesitated to represent a polyomino using a boolean matrix corresponding to squares that belong to the polyomino. However, this methode does not distinguish translation classes, therefore causes difficulties in tiling afterwards. The method chosen here facilitate isometric transformations on polyominoes, in terms of geometrical coordinates.

This method allow direct generation from a string of coordinates, and furthermore, adding squares to draw polyominoes should be immediate.

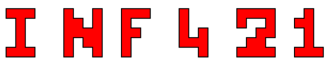


Figure 1: Demonstration of polyomino generation by text file

1.2 Generate polyominoes

Task 2

A first naive approch for generation of a polyomino of area n is recursively adding neighboring squares to a polyomino of area n. In each cases (fixed, one-side, free), each newly obtained polyomino is checked correspondingly to eliminate repeatition.

As can be seen below, this algorithm demands enormally in terms of execution time. We could not push the enumeration beyond $n = 9$.

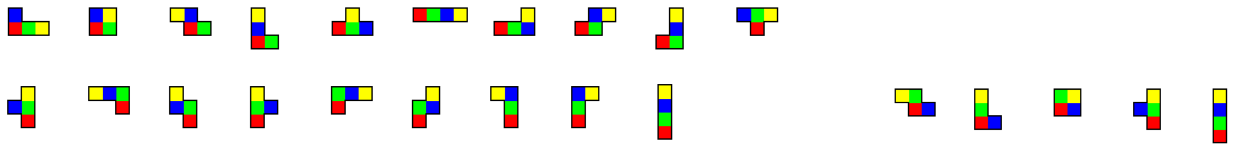
Task 3

It is clear that Redelmeiers' algorithm surpass, comparing time complexity. Enumeration limit is pushed further, specially with fixed polyominoes.

These polyominoes are organized in a tree of `PolyNode`, where each child polyomino is an extension of it parent, using Redelmeiers' algorithm. The two fields `tried` and `untried` serve as potential square to add to the parent and squares already added, and are passed on by inheritence. The aim is to add only squares that do not belong to the child's bigger brothers or its ancestors' bigger brothers.

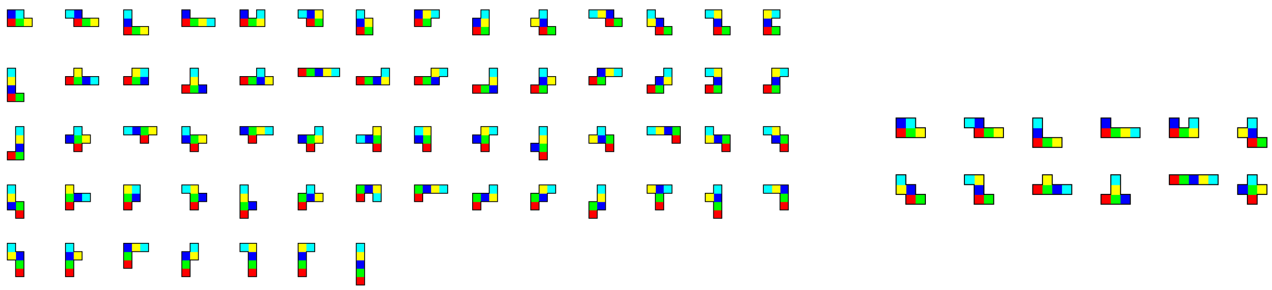
However, for generating free polyominoes, we have not found a more optimsed solution than removing all transformations of a polyomino. For each variant of a free polyominoes, its position in the returned list is search by going down the tree presented above, which can be slightly faster than linear searching. Nevertheless, when n increases, memory could be exhausted.

n	Number of polyominoes		Execution time (ms)			
			Naive		Redelmeier	
	Fixed	Free	Fixed	Free	Fixed	Free
3	6	2	1	4	1	2
4	19	5	4	5	2	4
5	63	12	7	10	3	7
6	216	35	13	28	6	12
7	760	108	31	97	10	22
8	2725	369	222	663	25	58
9	9910	1285	3849	7361	38	304
10	36446	4655	102373	106955	91	1457
11	135270	17073			778	16442
12	505861	63600			2540	354814



(a) (b)

Figure 2: Fixed 2a and free 2b tetra-ominoes



(a) (b)

Figure 3: Fixed 3a and free 3b penta-ominoes

Part 2
POLYOMINO TILINGS AND THE EXACT COVER PROBLEM

2.1 The exact cover problem

Task 4

We implement the class `ExactCover` to represent the exact cover problem. The fields neccessary include:

- `groundSet`: the set X of elements need covering.
- `collection`: the set $\mathcal{C}$ of subsets of X.
- `solution`: the set of coverings of X, each is a subset of $\mathcal{C}$ that contains disjoint subsets of X whose union equals X.

The naive approch through backtracking is tested on $X = \{1, 2, ..., n\}$ and $\mathcal{C} = \mathcal{P}(X)$. The size of X could be pushed to $n = 9$, the reults is shown in comparaision to the next algorithm below. Another test conducted is to consider combinations of k among n elements of X.

In order to examinate starting covering from the element $x \in X$ contained in the least subsets, we randomly chose half of the subsets in $\mathcal{C}$. Consequently, reducing the number of branches at each step does improve time complexity.

2.2 D. Knuth's dancing links algorithm

Task 5

The class `DancingLinks` extending `ExactCover` permit transforming an exact cover problem to its corresponding dancing links data structure. The details of this implementation has been included in articles cited by the assignment.

To initiate this structure, given by X and $\mathcal{C}$, we first iterate through elements of X to create column objects `colObj` representing X. Then each subset contained in $\mathcal{C}$ is added to by creating new links and updating fields `U, D, L, R, N, C, S`. Each one is represented by a data object `dataObj` and its horizontal neighbor.

Task 6

We apply the same instances of the exact cover of the previous part to D. Knuth's dancing links algorithm to test for its performance. The result can be seen from the table below, as we try to cover $X = \{1, 2, ..., n\}$ by a collection of its 2^n subsets.

n	Number of partitions	Execution time (ms)	
		Naive	DancingLinks
1	1	0	0
2	2	1	0
3	5	1	0
4	15	3	0
5	52	11	1
6	203	44	5
7	877	262	7
8	4140	1603	16
9	21147	18629	93
10	115975		339
11	678570		2428
12	4213597		25565

2.3 From polyomino tilings to exact cover

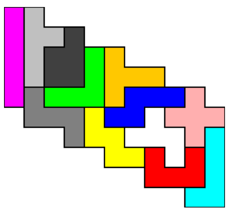
Task 7

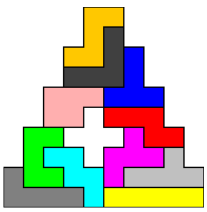
A problem of a polyomino P by smaller polyomioes can be considered as an exact cover problem. The ground set is the set of all squares of P. Meanwhile, a subset is represented by a positioning of a small polyomio at a certain position.

In order to include the condition that a polyomio from the collection S exactly once, one further step can be taken. We put all polyomioes of S into X, and each subset contains the corresponding one as well.

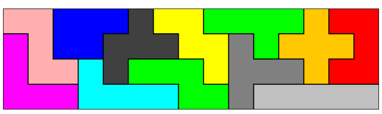
Task 8

- Tiling polyominoes given in Figure 5 of the assigment by all free pentaominoes, refering to `test1(which)` in class `Test`
  - left: 404 tilings
  - middle: 374 tilings
  - right: none

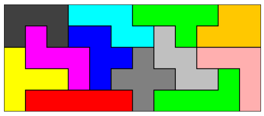




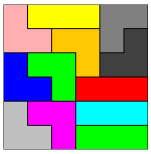
- Tiling a rectangle: refering to `Test2()` in class `Test`
  - Tiling 4×15 by all free polyominoes, rotation and flip allowed: 1472 tilings



– Tiling 12×5 by all free polyominoes, rotation and flip allowed: 4040 tilings

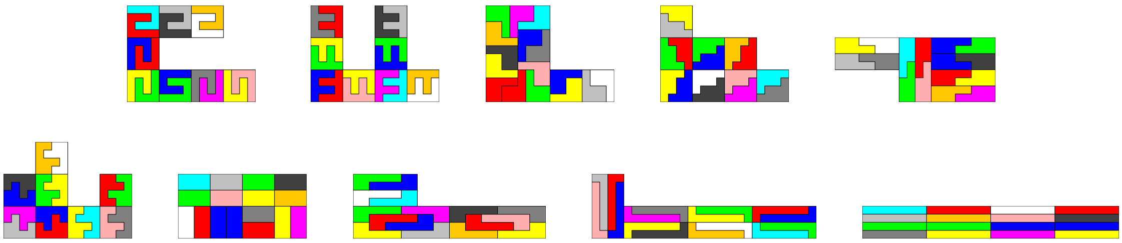


– Tiling 6×6 by fixed polyominoes, neither rotation or flip, repetition allowed: 80092 tilings



- Covering one's dilation: refering to `Test3(n, k)` in class `Test`

  For $(n, k) = (8, 4)$, we found 10 free polyominoes satisfying this property.



Part 3
EXTENSIONS

3.1 Higher dimension

To further apply what we have accomplished with flat square polyomino into three dimensional polycube or other lattices, a new abstract class `Cell` was createda and inherited by `Cube`, `Triangle` and `Hexagon`. This class includes three coordinates and abstract methods adapted to each situation.

The result of enumeration is displayed below.

n	PolyCube			Triangulaminoes			Hexagonaminoes		
	Fixed	One-sided	Free	Fixed	One-sided	Free	Fixed	One-sided	Free
1	1	1	1	2	1	1	1	1	1
2	3	1	1	3	1	1	3	1	1
3	15	2	2	6	1	1	11	3	3
4	86	8	7	14	4	3	44	10	7
5	534	29	23	36	6	4	186	33	22
6	3481	166	112	94	19	12	814	147	82
7	23502	1023	607	250	43	24	3652	620	333
8	162913	6922	3811	675	120	66	16689	2821	1448
9	1152870			1838	307	160	77359	12942	6572
10			5053	866	448	362671			
11			14016	2336	1186				
12			39169	6588	3334				
13			110194	18373	9235				
14			311751						

Task 9

Naturally, two neighboring cubes share two coordinates and the another differing by 1. Moreover, transformation with 3D polycube can be expressed using signed permutation corresponding to the rearrangement of its orthogonal coordinate axes.

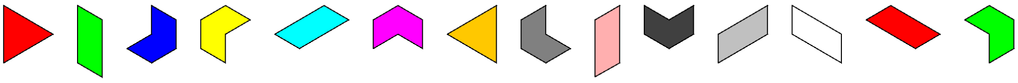
As result, for example, there are 128 tilings of the $2 \times 2 \times 7$ using 7 polycube of volume 4. Unfortunately, we have not had enough time to build presentation methods for 3D polycubes.

3.2 Other lattices

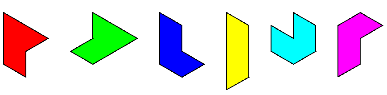
The neighboring relations, as well as isometric transformation on triangular and hexagonal lattices are much more delicate and difficult to express. Here we keep 3 coordinates eventhough these lattices exist in the plane, so as to define three direction in the plan, allowing moving from one cell to one of its three or six neighbors.

We have been able to reproduce some tiling of parallelograms similar to those given. However as the number of tilings became enorme and exceeded the memory limit, it was obligated to restrain the branchement of each step while bactracking, in order to be capable to return some complete solutions.

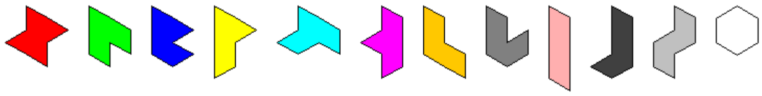
Task 10



(a) Fixed triangulaminoes of area 4

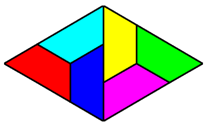


(b) Oneside triangulaminoes of area 5

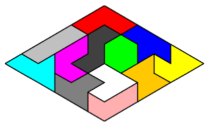


(c) Free triangulaminoes of area 6

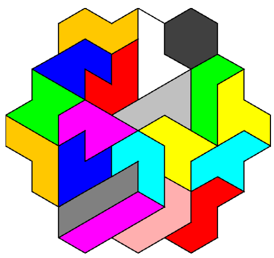
Figure 4: Generation of triangulaminoes



(a) 4 - 6 fixed triangulaminoes of area 3



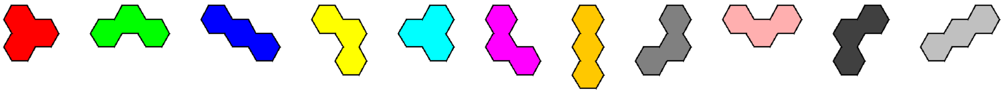
(b) 624 - 12 free triangulaminoes of area 6



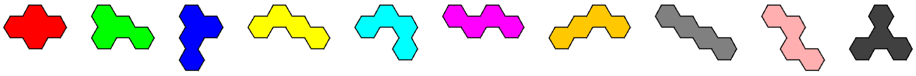
(c) 528 - 19 one-sided triangulaminoes of area 6

Figure 5: Some tilings with triangulaminoes

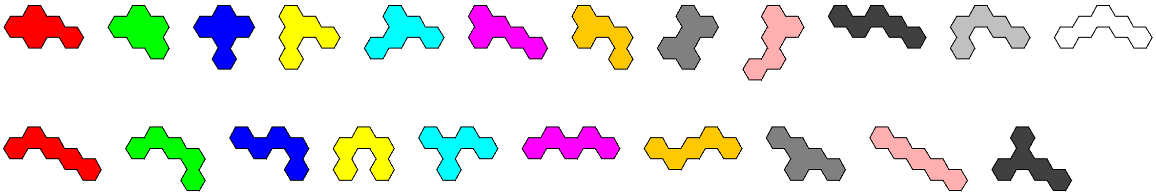
For tiling with hexagonaminoes, the number of solutions in each case exceeded over 10000. In the implementation of `DancingLinks.exactCover()` a ceil is introduced to limit the branchement at each step while backtracking.



(a) Fixed hexagonaminoes of area 3

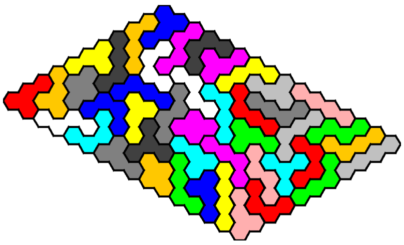


(b) Oneside hexagonaminoes of area 4

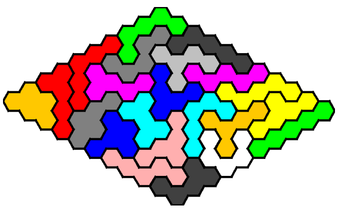


(c) Free hexagonaminoes of area 6

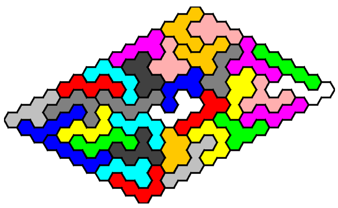
Figure 6: Generation of hexagonaminoes



(a) 44 fixed hexagonaminos of area 4



(b) 22 free hexagonaminoes of area 5



(c) 33 one-sided hexagonaminoes of area 5

Figure 7: Some tilings with hexagonaminoes

3.3 Sudoku

A Sudoku problem can be transformed into an exact cover problem. Each square contains a number from 1 to 9 so that each row, each column and each box 3×3 contains all numbers from 1 to 9.

Each possibility that the intersection of row r and column c containes the number x is considered as an element of $\mathcal{C}$. This possibility contains three elements:

- The row r must contain the number x.
- The column c must contain the number x.
- The box containing the square (r, c) mus contain the number x.

  These constaints are used to construct our ground set X, that needs covering.

Task 11

Here we represent a table of sudoku as a string of numbers, since it includes only number from 1 to 9. Notice that the squared unfilled contains initially 0. This program allows to determines all solutions possible for a given sudoku problem usually seen.

For example:

0	0	4	3	0	0	2	0	9
0	0	5	0	0	9	0	0	1
0	7	0	0	6	0	0	4	3
0	0	6	0	0	2	0	8	7
1	9	0	0	0	7	4	0	0
0	5	0	0	8	3	0	0	0
6	0	0	0	0	0	1	0	5
0	0	3	5	0	8	6	9	0
0	4	2	9	1	0	3	0	0

8	6	4	3	7	1	2	5	9
3	2	5	8	4	9	7	6	1
9	7	1	2	6	5	8	4	3
4	3	6	1	9	2	5	8	7
1	9	8	6	5	7	4	3	2
2	5	7	4	8	3	9	1	6
6	8	9	7	3	4	1	2	5
7	1	3	5	2	8	6	9	4
5	4	2	9	1	6	3	7	8