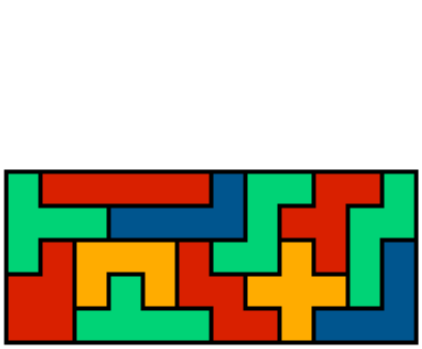
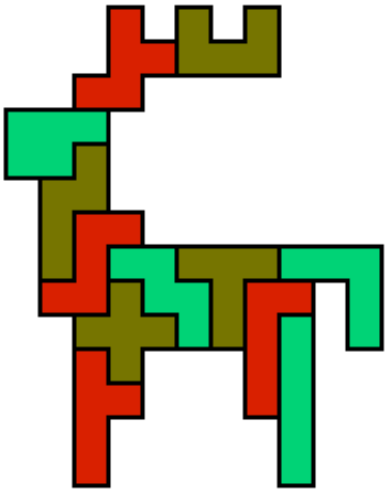
Solving Pentomino Puzzles with Backtracking

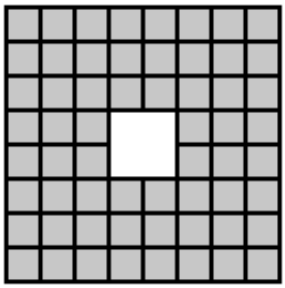
The *pentominoes* are 12 pieces formed by joining five squares along their sides in all possible shapes. These are the 12 pentominoes:

The pentominoes can be used together to form different shapes. These make interesting puzzles. For example, here are the 12 pentominoes forming a 5x12 rectangle and the shape of a deer:





Can the 12 pentominoes be used to form an 8x8 square with a 2x2 hole in the middle? It would be interesting to write a computer program that could solve puzzles like these.



The simplest way of approaching this problem is by using a backtracking, or depth-first traversal, algorithm.

At first thought, one thinks of two possible ways to do a depth-first traversal. One way is to start with the empty board, select the first pentomino, and try all possible ways of placing it on the board. For each of them, take the second pentomino and try placing it in all possible ways. And so on, until one succeeds in placing the twelfth pentomino on the board.

Another way is to consider the first cell of the board, and try all possible ways to cover it with a pentomino. Then consider the next uncovered cell, and try all possible ways to cover it with another pentomino. And so on, until all cells of the board are covered.

Both methods are exhaustive; therefore, they should produce the same solutions. However, the second method is faster than the first one, because at each step, generally speaking, there are fewer possibilities branching out: Normally, there are more ways to place a given pentomino somewhere on the board than to cover a given cell of the board with some pentomino. Therefore, the second method traverses a search tree that has fewer nodes.

Donald E. Knuth in [1] shows that this puzzle -- of forming a given shape with the 12 pentominoes -- is an example of a more general problem, known as the Exact Cover problem. Therefore, we can take a general algorithm that solves the Exact Cover problem and use it to solve the pentomino puzzle. The result is an algorithm that is more efficient than the two methods described above.

First, let's see what the Exact Cover problem is. We are given a matrix of 0's and 1's, and the problem is to find an "exact cover" of the matrix's columns: a subset of its rows such that, put together, they contain exactly one 1 in each column of the matrix. Knuth gives the following example:

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

In this case, the matrix has only one such subset: rows 1, 4, and 5.

An algorithm to solve the Exact Cover Problem is as follows:
1) Choose a column *c* from the matrix.
2) Choose, in turn, each row *r* where *c* contains a 1. For each *r* do the following:
3) Erase all columns where *r* contains a 1, and all rows that contain a 1 in any of those columns.
4) Apply the algorithm recursively to this reduced matrix.

For example, let's apply the algorithm to the above matrix. The starting matrix is:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{pmatrix} & & 1 & & 1 & 1 & \\ 1 & & & 1 & & & 1 \\ & 1 & 1 & & & 1 & \\ 1 & & & 1 & & & \\ & 1 & & & & & 1 \\ & & & 1 & 1 & & 1 \end{pmatrix} \end{array}$$

At step 1, let's choose column A. At step 2, we first try with row 2. Step 3 tells us to eliminate columns A, D, and G, and rows 2, 4, 5, and 6. We're left with:

$$\begin{array}{c} \\ 1 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} B & C & E & F \end{array} \\ \begin{pmatrix} & 1 & 1 & 1 \\ 1 & 1 & & 1 \end{pmatrix} \end{array}$$

Now we choose column B, and row 3. We eliminate columns B, C, and F, and rows 1 and 3, and we are left with a matrix with only column E and no rows. This matrix has no solutions. Therefore we backtrack the choice of row 3 in column B, and since there are no other 1's in B to choose from, we backtrack the choice of row 2 in column A.

Then we choose instead row 4 for column A. Now we eliminate columns A and D, and rows 2, 4, and 6. The result is:

```
   B  C  E  F  G
1 (    1  1  1     )
3 ( 1  1     1     )
5 ( 1              1 )
```

We choose column B, and row 3 (again). We eliminate columns B, C, and F, and rows 1, 3, and 5. Once again, we are left with column E and no rows. Therefore we backtrack the choice of row 3 for column B and try instead row 5. We delete columns B and G, and rows 3 and 5. We are left with:

```
   C  E  F
1 (1  1  1)
```

We select row 1, and we eliminate all the rows and all the columns. That means we succeeded. The solution is the rows we chose: 4, 5, and 1. After announcing this solution, we backtrack the choice of row 1, then row 5, and then row 4. Since there are no other rows that contain a 1 in column A, we are finished. That was the only solution.

In order to minimize the number of branches we have to try at step 2 of the algorithm, when we choose a column at step 1 we should choose the one that contains the fewest number of 1's. This will produce a search tree that has about as few nodes as possible. In order to know which column has the fewest 1's, we need to keep a running track of how many 1's each column contains.

Now we get to the relation between the pentomino puzzle and the Exact Cover problem. Construct a matrix with 72 columns: 12 for the twelve pentominoes and 60 for the cells of the board. Now, for each way a pentomino can be placed on the board, insert a row in the matrix. The row should contain a 1 in one of the first 12 columns, indicating the pentomino, and five 1's among the other 60 rows, indicating the five cells of the board the pentomino occupies; everywhere else the row should contain zeros. This should be done for all possible ways of placing each of the twelve pentominoes on the board. The matrix corresponding to the 8x8 square shown above, for example, has 1568 rows.

Now the problem of finding a solution to the puzzle is equivalent to finding an exact cover of this matrix. The first algorithm we explained before works by selecting the columns sequentially from the first 12. The second algorithm works by selecting columns sequentially from the other 60. The more efficient approach consists of always selecting the column that has the fewest number of 1's. Sometimes the column will be from among the first 12, meaning that the algorithm will try to find a place for a certain pentomino, and sometimes it will be from among the other 60, meaning that it will try to cover a certain cell.

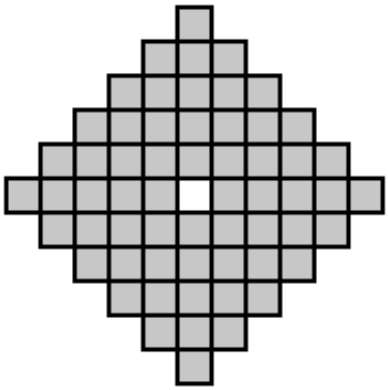
Knuth in [1] explains an implementation of the algorithm with a series of objects that are doubly-linked horizontally and vertically to represent the matrix.

However, to keep things simple, I implemented the algorithm representing the matrix as an array, with some data structures to keep track of which rows and columns are active.

I made the program accept as input an arbitrary 60-cell pattern. The program proceeds to build the matrix corresponding to the pattern, and applies the Exact Cover algorithm, printing out all the solutions it finds.

When I ran the program on the 8x8 pattern shown above, it found 520 solutions, agreeing with Knuth that the pattern has 65 solutions: Since the pattern has eight-fold symmetry, each solution was produced eight times. Knuth in [1] shows a solution to the pattern. Here is a different one:

Afterwards, I decided to try the program on this 60-cell diamond pattern:



I have a real-life set of the pentominoes. The set comes with a little manual that shows several puzzles and solutions to some of them (They did it all by hand, not with a search program; that is the real point of the game). The manual presents the diamond as a "Problem with no solution yet. A challenge to ingenuity!"

I entered the diamond pattern into the program. The program finished after a while, finding no solutions. Now that's a pity! The pattern has no solution!

Then it ocurred to me that the program could be used to write a deductive proof that the pattern has no solution. That would be cool!

Here is the proof that the diamond pattern is impossible to construct: diamondproof.zip (800K; expands to 11 MB -- you need a text editor that can manage well with large files). The proof is eight times as long as it needs to be, because the program didn't take into account the eight-fold symmetry of the pattern.

---

[1] "Dancing Links." Donald E. Knuth, Stanford University. Available as PostScript at http://www-cs-faculty.stanford.edu/~knuth/papers/dancing-color.ps.gz. In this article Knuth uses the Exact Cover algorithm to solve many additional problems, involving polyominoes, polyamonds, polysticks, queens on a chessboard, and more.

Home page

gnivasch@yahoo.com