Project 2, TCSS 380 Spring 2019

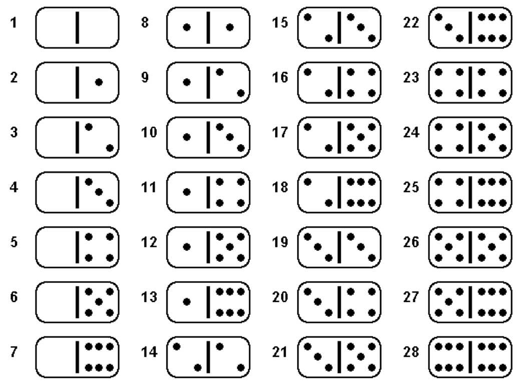
OBJECTIVE

The objective of this project is to apply your newly learned Erlang knowledge to solve an interesting puzzle.

ASSIGNMENT DESCRIPTION

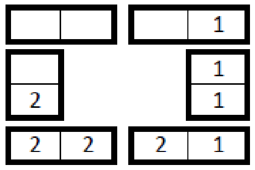
For this assignment, you are to write a program that finds a circular train constructed from all domino tiles from the given set of dominoes using a list of tuples to represent dominoes. You will choose one of two approaches to the problem described below.

Dominoes are small rectangular game tiles divided into two squares and embossed with dots on the top surface of the tile. You can think of dots as integers. The traditional set of dominoes contains one unique piece for each possible combination of two numbers – such sets are called double-N domino sets. A double-N domino set consists of (N+1)(N+2)/2 tiles, e.g. a standard double-six domino set has 28 tiles (T): one for each possible pair of values from (0, 0) to (6, 6):



(image from http://www.diy-enthusiasts.com/diy-gifts/diy-kids-games-dominoes-pebbles/)

Dominoes are used to play a variety of games involving patterns. One possible pattern to make with dominoes is a circular train (or a ring), in which all the tiles are laid in a circle, end-to-end, with identical numbers on all adjacent ends. In a double-two domino set, with six tiles, (0 , 0) (0 , 1) (1 , 1) (1 , 2) (2 , 2) (2 , 0) is an example of a ring that uses all the tiles from that set:

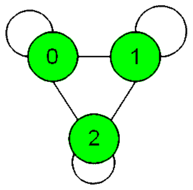


Circular Train Approaches

There are many possible ways to write the program. Perhaps the simplest way is to generate all possible tile permutations, and to check if any one of them forms a circular train. Unfortunately, you will run out of memory very quickly since the number of permutations is a factorial of the number of tiles (T), e.g. if N=6, then T = 28, and 28! is … well, it is too big of a data structure to be held in memory. How can we improve on this? Perhaps we only generate one permutation at a time and check whether that permutation forms a ring – we will not run out of memory for sure, but using this approach we will most likely run out of patience, as the program will run forever before the match is found.

Another approach, the one you will follow if you pick an option we will call OPTION R, is to try to form a ring randomly using random walk strategy. Random walk simply means that you pick a random tile to start with and for the second tile, you can randomly match either of the values appearing on the first tile. However, for each subsequent tile you randomly pick a tile from the set of tiles that match the preceding one on the "second" value. For example, if { 2 , 1 } is the random initial starting node, then the next random tile may be either of the tiles that contain 2 or 1, e.g. { 2 , 0 }, as long as they are still available. If { 2 , 0 } is chosen, however, because of the value 2 matching the prior tile, then the next random choice must be on all the remaining tiles containing a zero. If you select to encode the functions for OPTION R, then you need to run your program multiple times and count how many times you had to run it to find a solution and report the result of your trials in executive summary.

For OPTION A, you will use Fleury's algorithm or Hierholzer's algorithm (you will need to research these algorithms) but in order to understand what they are attempting to do, some background is necessary. Our ring problem can be modeled using a graph, and as such, the problem was solved by Leonhard Euler in the 18th century. The problem is known as Bridges of Königsberg problem and the question that we are really asking is whether there is a Eulerian circuit in a double-N domino set. Dominoes could be represented as a graph in the following fashion: dots are used as nodes, whereas tiles are used as edges between them. A double tile is shown as an arc that starts and finishes at the same node. The graphs for all double-N domino sets will have the same number of edges as there are dominoes in the set they model. For example, the following graph models a double-2 domino set:



As shown by Euler, for a circuit path to exist, a graph must be connected, and each node must have an even number of edges (not counting self-referential edges). It follows that the only double-N domino sets that will have a solution, are the sets where N is even (i.e. there will be no solution for double-1, double-3, double-5, etc). Moreover, there have been a number of algorithms already written for walking such a graph – this is where Fleury and Hierholzer come in – they give you a recipe for finding the ring.

Program Specs

Your program is to include the following functions (follow the naming conventions if you want your program to be graded):

- **dominoes(N)** – given an integer, returns a list of tuples containing all the tiles in the set of double-N dominoes, e.g. dominoes(2) returns some variation of [{2 , 2}, {2 , 1}, {2 , 0}, {1 , 1}, {1 , 0}, {0 , 0}] – order does not matter
- OPTION A (more difficult)
  - **Eulers(MyList)** – given a list of tuples, returns a reordered list of tuples that forms one of possible circular trains, e.g. given a list from the bullet above, the function may return [{2 , 1}, {1 , 1}, {1 , 0}, {0 , 0}, {2 , 0}, {2 , 2}] (this is the function in which you will use Fleury's or Hierholzer's algorithm) – this is the most challenging part of this assignment and you may want to postpone it until you develop other functions
- OR OPTION R (less difficult but requires running the code through multiple trials)
  - **giveRandom (MyList)** – given a list of tuples, returns randomly assembled list of tuples where the algorithm picks a random tuple to begin the list and then picks randomly another tuple that matches the preceding one in the fashion described before; the function returns when it can find no more matches; the random number generator must be seeded properly so that each time the function is called, it results in a different outcome
  - **isRing (MyList)** – given a list of tuples, returns true if tuples form a ring and all tiles from a double-N domino set have been used, or false otherwise; note that the first and the last dominoes need to match as well. Bear in mind that the sequence of dominoes such as [{2 , 1}, {1 , 1}, {1 , 0}, {0 , 0}, {2 , 0}, {2 , 2}]should be considered as forming a ring since it is enough to flip a tile to get the ring
- **flip(MyList)** – given a list of tuples that form a circular train, returns a list with tiles that are flipped where appropriate to make the loop self-evident, e.g. if flip is applied to [{2 , 1}, {1 , 1}, {1 , 0}, {0 , 0}, {2 , 0}, {2 , 2}], it results in [{2 , 1}, {1 , 1}, {1 , 0}, {0 , 0}, {0 , 2}, {2 , 2}]

- **solution(N)** – given an integer, generates a solution – ties all the functions given above in a functional manner; note that there are no domino rings when N is odd and you should return an empty list in such a case
- **listAsString(MyList)** – given a list of tuples, returns a string representing that list that follows the format "[{2, 1}, {1, 1}, {1, 0}, {0, 0}, {0, 2}, {2, 2}]" (blanks after all commas, after opening square bracket and before closing square bracket)
- **driver(F1, F2, N)** – given **listAsString** and **solution** functions, as well as **N** - the initial highest number of dots for the set, returns a string representing a solution to the program, i.e. it should be enough to call this function to get the answer to the circular train question.

In addition, your program must fulfill the following requirements:
- it is to be compatible with the Erlang version used this quarter
- it is to be written in a functional manner
- helper functions should NOT be listed in an export statement
- include comments – the header with your name, program name, date; function headers that describe the purpose of each function; code comments that describe the high-level logic
- your program is to be contained within one file called *pr2_A.erl* if you followed option A or *pr2_R.erl* if you followed option R
- you are to provide Erlang unit tests for each of the main functions in a file called *pr2_tests.erl*

Your program will be graded by running it through a test file, so it is extremely important that you meet all the specifications and naming conventions given above.

You are to also write an executive summary similar to the ones you used to write in TCSS 305. It should be written as a txt file. In the file, describe what you have done to complete the assignment in 200-500 words. The word count is not strict, so do not worry about going slightly over; however, summaries that do not meet the minimum length requirement or are trivial in nature (representing little thought or effort) will not get full credit.

EXTRA CREDIT

Extend the program in an interesting way but make sure that you do not modify basic functions described above (i.e. add extra functions that extend functionality)**.** At the top of *pr2.erl* explain what your extra credit is and in a test file show the tests for your functions.

SUBMISSION AND GRADING NOTES

You are to submit your finished program through *Canvas*. This is an individual assignment – you are NOT allowed to work on it with any other student or share your code with others. The code will be graded based on its correctness by running it on instructor/grader designed test cases. Then, the code will be looked over for anything that violates good coding practices or the assignment specs (e.g. non-meaningful variable names, lack of code modularization). Note I will accept no submissions past the assignment deadline. NO submissions will be accepted even if you submit one minute after the deadline's timestamp. The link will not be available after the due date.

Grading points will be distributed in the following fashion:

• All functions (per given option)	30 pts
• Test file	7.5 pts
• Coding style and comments	7.5 pts
• Executive summary	5 pts
• Extra credit	5 pts