Project 2, TCSS 480 Autumn 2014
Due Dec. 2, 2014, 5 p.m.
Hard deadline Dec. 3, 2014, 8 a.m.

OBJECTIVE

The objective of this project is to apply your newly learned ML knowledge.

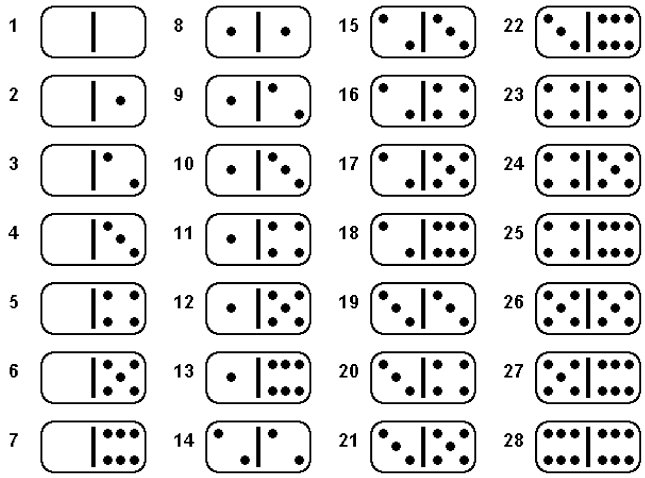
ASSIGNMENT SUBMISSION

You will need to submit your finished sml file (85 points), the test file (if modified), and an executive summary (15 points) through Canvas. Your code needs to include comments and has to be compatible with SML N/J.

ASSIGNMENT DESCRIPTION

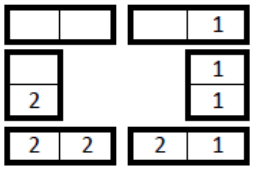
For this assignment, you are to write a program that finds a domino pattern for a "double-N" domino set that forms a loop / a ring using all available tiles.

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. 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) - no pair of numbers occurs more than once:



(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 dominos is a loop (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 domino loop that uses all the tiles from that set:



You are to write a program that finds a solution for any number N, if such a solution exists.

ASSIGNMENT SPECS

Program

There are many possible ways to write your program. Perhaps the simplest is to generate all permutations of a list of the tiles in the domino set, and check to see which form the loops. 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. So, it is much better to generate just one permutation at a time and check whether that permutation forms a loop – this is the basic requirement for this program.

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, e.g. `dominoes(2)` returns some variation of ((2 , 2), (2 , 1), (2 , 0), (1 , 1), (1 , 0) (0 , 0)) – order does not matter; any helper function that you use should be an inner function
- `permutation(L)` – given a list of tuples, returns a reordered list of tuples, e.g. given a list from the bullet above `permutation(L)` returns some variation of `L`, e. g. ((2 , 1), (2 , 0), (1 , 1), (2 , 2), (1 , 0) (0 , 0)) – order does not matter but it is important that your function returns a different permutation each time, unless the permutation is a loop (meaning, you used heuristics that result in the right pattern when a function is called just once instead of trying some random permutation of the set);
- `isLoop(L)` – given a list of tuples, returns `true` if tuples form a ring, as described in the problem definition above or `false` otherwise, e.g. `isLoop(L)` given a list from the bullet above returns false; note that the first and the last dominoes need to match as well. Bear in mind that the sequence of dominoes such as ((0 , 0), (1 , 0), (1 , 1), (1 , 2) ,(2 , 2), (0 , 2)) should be considered as forming a loop since it is enough to flip a couple of tile to get the loop
- `flip(L)` – given a list of tuples that form a loop, returns a list with tiles that are flipped where appropriate to make the loop self-evident, e.g. if flip is applied to ((0 , 0), (1 , 0), (1 , 1), (1 , 2) ,(2 , 2), (0 , 2)), it results in ((0 , 0), (0 , 1), (1 , 1), (1 , 2) ,(2 , 2), (2 , 0))
- `solution(N)` – given an integer, generates a solution – ties all the functions given above in a functional manner; note that there are no domino loops when N is odd and you should return an empty list in such a case
- `listAsString(L)` – given a list of tuples, returns a string representing that list
- `driver(F1, F2) N` – given `listAsString` and `solution` functions, as well as 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 problem solution back

In addition, your program must fulfill the following requirements:

- it is to be written in a functional manner, i.e. no loop iterations – only recursion, minimal side effects, if any, etc.
- any helper function should be nested inside the function it is supposed to be helping
- include comments – the header with your name, project name, date; function headers; code comments that describe the high level logic
- your program is to be contained within one file called `domino.sml`

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. The sample test file is provided with the project description. It tests the program as a whole and it tests each function separately. If you do not get all the functionality in your program to work correctly, comment out appropriate test file portions and submit it through Canvas as well.

Executive summary

You are to 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. You can share your personal experiences, things that particularly frustrated you about the assignment, things that particularly interested you about the assignment, etc. It is especially important that you document any difficulties you had with ML, the libraries, etc.