Scheme Assignment

Boğaziçi University
CmpE 260 Principles of Programming Languages
2020 Spring

Instructor: Fatma Başak Aydemir
TA: Alper Ahmetoğlu
SA: Burak Çetin

1 Introduction

For this project your goal is implementing an automated solver for the *Tents* puzzle in *Scheme*. Here is an online version where you can play around. Even though the puzzle rules will be given here, it is highly recommended that you try solving a few yourself as well.

2 Tents Rules

Place tents in the empty squares in such a way that:

1. No two tents are adjacent, even diagonally.
2. The number of tents in each row and column matches the numbers around the edge of the grid.
3. It is possible to match tents to trees so that each tree is horizontally or vertically adjacent to its own tent (but may also be adjacent to other tents that are matched with other trees).

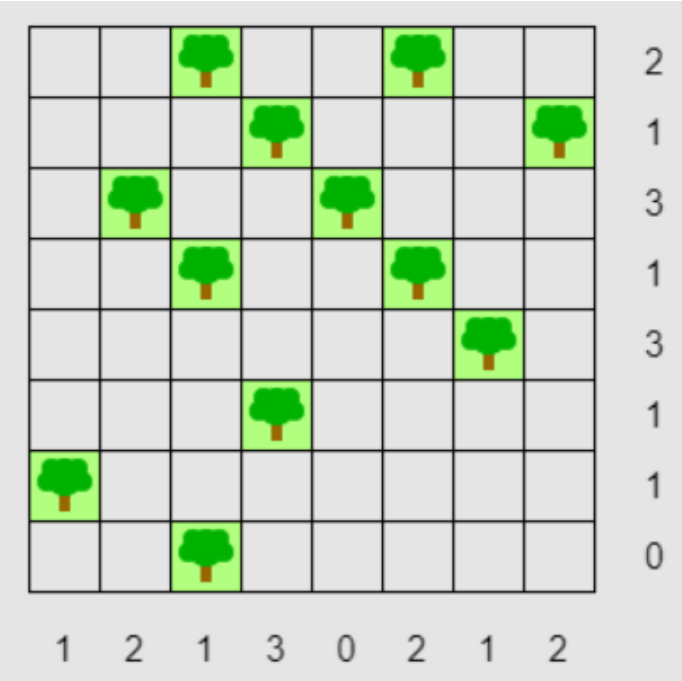

Figure 1: An example 8x8 tents puzzle

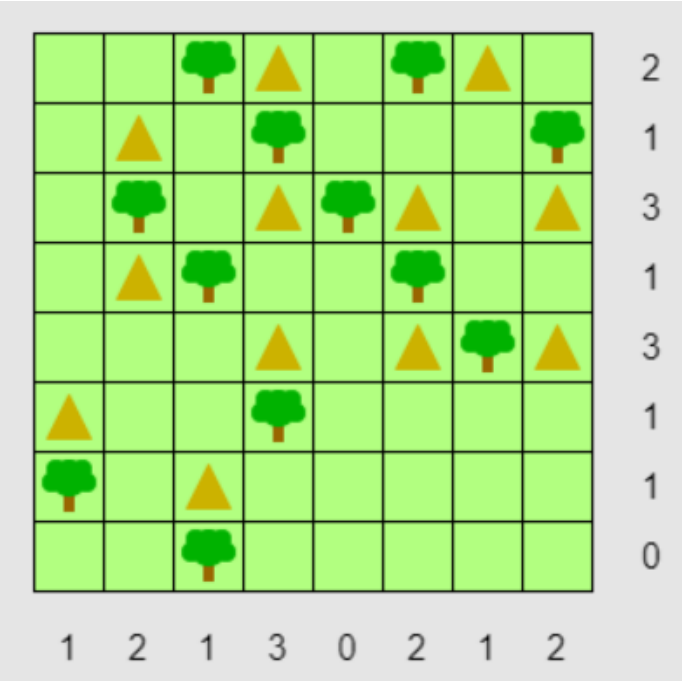

Figure 2: A solution for the example puzzle

3 Input and Output Format

In Scheme the input puzzle will be in the following form:

```
( (<tent-count-row1> ... <tent-count-rowN>)
(<tent-count-column1> ... <tent-count-columnM>)
( (<tree1-x> <tree1-y>) ... (<treeT-x> <treeT-y>) ) )
```

For example the puzzle in the figure 1 would be written down like this:

```
'( (2 1 3 1 3 1 1 0)
(1 2 1 3 0 2 1 2)
( (1 3) (1 6) (2 4) (2 8) (3 2) (3 5) (4 3) (4 6) (5 7) (6 4) (7 1) (8 3) ) )
```

In Scheme the answer for a puzzle will be in the following form:

```
( (<tents1-x> <tents1-y>) ... (<tentsT-x> <tentsT-y>) )
```

An example answer for this puzzle as shown on figure 2 would be outputted like this in Scheme:

```
((7 3) (6 1) (5 4) (5 8) (5 6) (4 2) (3 6) (2 2) (3 8) (3 4) (1 7) (1 4))
```

Row and column count for a puzzle will be no bigger than 10 and at least 1. Positions of the trees and number of tents on rows and columns will be consistent with the row and column sizes.

4 Functions to Implement

Mainly you are asked to implement a function named **TENTS-SOLUTION** that provides a solution for a given tents puzzle or returns #f (false in Scheme) if there are no solutions. In case there are multiple valid solutions for the given puzzle your function can return any of them. Make sure you follow the given input and output format properly. Here is an example as it would be seen in the DrRacket terminal:

```
> (TENTS-SOLUTION '((1 0 1) (0 2) ((2 2) (3 1)) ) )
((3 2) (1 2))
```

You can also implement the following helper functions for partial credit. You may or may not use these in your actual solver function but they can be useful. *It is not required to implement these functions to get a full grade if your solver function is working properly. See the grading rules in the next section for details.*

RETURN-FIRST-NOT-FALSE (15 Points): This function takes 2 parameters. First parameter will be a function and the second parameter will be a list. This function would apply the given function to each element of the list in order. If the given function returns something that is not false, then this function should return the return value as well. If the given function returns false for each and every element of the list then this function should return false as well.

```
> (RETURN-FIRST-NOT-FALSE
(lambda (x) (if (> x 5) (* x x) #f)) '(3 4 8 9))
64
```

ADJACENT (5 Points): This function takes 2 parameters. Both parameters will be lists with 2 integers representing a position on a grid. This function should return true if these positions are adjacent, including diagonals, or the same position. Otherwise it should return false.

```
> (ADJACENT '(1 3) '(2 4) )
#t
> (ADJACENT '(2 3) '(2 3) )
#t
> (ADJACENT '(2 3) '(5 5) )
#f
```

NEIGHBOR-LIST (5 Points): This function takes a single parameter. It will be a list with 2 integers representing a position on a grid. This function should return a list of neighboring positions, horizontally and vertically. The function can return the neighbors in any order. *Even if a position is out of bounds in the context of the tents puzzle this function must put that position in the list that this function returns.*

```
> (NEIGHBOR-LIST '(5 6) )
((5 7) (6 6) (5 5) (4 6))
> (NEIGHBOR-LIST '(1 5) )
((1 6) (2 5) (1 4) (0 5))
```

ADJACENT-WITH-LIST (5 Points): This function takes 2 parameters. A list containing 2 integers and a list of lists that are containing 2

integers. The first parameter represents a position on a grid and each element on the second list also represents a position. This function should return true if the position represented by the first parameter is adjacent to at least one of the positions in the given list (using the same definition for adjacency from the ADJACENT function). Otherwise it should return false.

```
> (ADJACENT-WITH-LIST '(5 2) '((3 5) (4 6)))
#f
> (ADJACENT-WITH-LIST '(4 3) '((7 5) (4 6) (5 3)))
#t
```

REPLACE-NTH (10 Points): This function takes 3 parameters. A list, an integer indicating a position on this list and a value. It should return a list that is identical to the given list, except the value in the given position should be replaced with the given value. The given position will always be positive and not bigger than the length of the given list.

```
> (REPLACE-NTH '(5 4 3 2 1) 4 42)
(5 4 3 42 1)
```

5 Grading

Grading of this project is based on the success of your code in test cases as well as your implementations of the helper functions. The following language constructs are explicitly prohibited. **You will not get any points if you use them**:

- Any function or language element that ends with an !.
- Any of these constructs: begin, begin0, when, unless, for, for*, do, loop, set!-values, let.
- Any language construct that starts with for/ or for*/.
- Any construct that causes any form of mutation (impurity).
- In short, you **must follow pure functional programming style**.

Your score for the TENTS-SOLUTION function will be the sum of collected points from different input puzzles. Each correctly solved puzzle will provide equal points. Execution time limit is 3 seconds. **If your code runs**

more than 3 seconds on a puzzle you will get 0 points from that puzzle. Total score is 100.

Your score for the helper functions will be the sum of points for each correctly implemented helper function.

Your final score for the project will be the maximum of these two scores. As a result if you have a correct implementation for the TENTS-SOLUTION function and it gets full score, it does not matter if you have implemented the helper functions or not.

Since your projects will be graded automatically, you must use the exact function names and argument lists as specified. Additionally the first two lines of your program should be:

```
#lang scheme
; student-id
```

You should only submit a single file and it should be named as:

```
tents_solution.rkt
```

If any kind of minor manual change had to be applied to your project by the assistants before it could be graded automatically, **20 points will be subtracted from your final score**. What kind of changes will be considered as minor changes is up to the assistants.