KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Programavimo kalbų teorija (P175B124)

Laboratorinių darbų ataskaita

Atliko:

IFF-6/11 gr. studentas

Nerijus Dulkė

2019 m. vasario 11 d.

Priėmė:

lekt. **Evaldas Guogis**

KAUNAS 2019

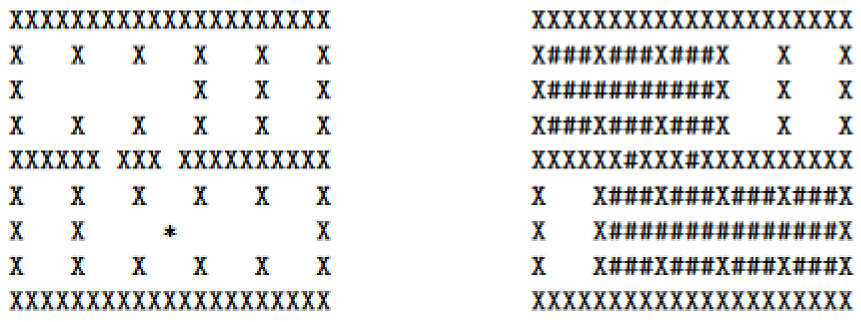
TURINYS

1. Python (L1)

1.1. Darbo užduotis

Nuoroda į užduotį:
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=9&page=show_problem&problem=725

Trumpas aprašymas:
Labirintas sudarytas iš stačiakampių kambarių pavaizduotas plokštumoje, naudojant simbolius. Užduoties tikslas yra pažymėti kambarius, kuriuos galima aplankyti iš nurodytos startinės pozicijos.



```
XXXXXXXXXXXXXXXXXXXXX      XXXXXXXXXXXXXXXXXXXXX
X   X   X   X   X   X      X###X###X###X   X   X
X           X   X   X      X###########X   X   X
X   X   X   X   X   X      X###X###X###X   X   X
XXXXXX XXX XXXXXXXXXX      XXXXXX#XXX#XXXXXXXXXX
X   X   X   X   X   X      X   X###X###X###X###X
X   X     *         X      X   X###############X
X   X   X   X   X   X      X   X###X###X###X###X
XXXXXXXXXXXXXXXXXXXXX      XXXXXXXXXXXXXXXXXXXXX
```

a) Initial maze   b) Painted maze

1.2. Programos tekstas

Nerijus.Dulke.IFF.6.11.Lab.1.py

```python
# IFF-6/11 Nerijus Dulke Lab1
# 
https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=9&page
=show_problem&problem=725

from Maze import Maze

mazes = []

duom = open('duom.txt')

mazeCount = int(duom.readline())

i = 0
iterations = 0
while i < mazeCount:
  maze = Maze()
  line = duom.readline()

  while line and not line.startswith('_'):
    maze.addline(line)
    iterations = iterations + 1
    line = duom.readline()

  mazes.append(maze)
  i = i + 1

duom.close()

rez = open('rez.txt', 'w+')
```

```python
for maze in mazes:
  maze.paint()
  maze.printresult(rez)
  rez.write('_____\n')

rez.close()
```

Maze.py

```python
WALL = 'X'
EMPTY = ' '
MARKED = '#'
START = '*'
directions = ['u', 'd', 'l', 'r', 'ul', 'ur', 'dl', 'dr']

class Maze:
  def __init__(self):
    self.lines = []
    self.startX = -1
    self.startY = -1
    self.maxX = -1
    self.maxY = -1

  def addline(self, line):
    self.lines.append(line)

    if self.startX == -1:
      index = line.find(START)
      if index != -1:
        self.startX = index
        self.startY = self.count() - 1

    self.maxY = self.count() - 1
    maxX = len(line) - 1
    if maxX > self.maxX:
      self.maxX = maxX
    return

  def printlines(self):
    for line in self.lines:
      print line
    return

  def count(self):
    return len(self.lines)

  def printresult(self, file):
    file.writelines(self.lines)
    return

  def getvalue(self, coord):
    return self.lines[coord.y][coord.x]
  def markvalue(self, coord):
    if self.getvalue(coord) is EMPTY:
      self.lines[coord.y] = self.lines[coord.y][:coord.x] + MARKED +
self.lines[coord.y][coord.x + 1:]
      return True
    return False

  def paint(self):
    self.max = Coord(self.maxX, self.maxY)
    current = Coord(self.startX, self.startY)
    coordsToSearch = [current]
    visited = []

    while len(coordsToSearch) > 0:
      current = Coord(coordsToSearch[0].x, coordsToSearch[0].y)
      del coordsToSearch[0]

      for direction in directions:
        neighbour = self.getNeighbour(current, direction)
        isVisited = filter(lambda x: neighbour.equals(x), visited)

        if neighbour is None or len(isVisited) > 0:
          continue

        success = self.markvalue(neighbour)
        if success:
          coordsToSearch.append(neighbour)

      visited.append(current)
    return

  def getNeighbour(self, current, direction):
    coord = Coord(current.x, current.y)
    if not coord.canmove(direction, self.max):
      return None

    if direction is 'u':
      coord.up()
    elif direction is 'd':
      coord.down()
    elif direction is 'l':
      coord.left()
    elif direction is 'r':
      coord.right()
    elif direction is 'ul':
      coord.up()
      coord.left()
    elif direction is 'ur':
      coord.up()
      coord.right()
    elif direction is 'dl':
      coord.down()
      coord.left()
```

```python
    elif direction is 'dr':
      coord.down()
      coord.right()

    return coord


class Coord:
  def __init__(self, x, y):
    self.x = x
    self.y = y

  def canmove(self, direction, maxcoord):
    if direction is 'u':
      return self.y != 0
    elif direction is 'd':
      return self.y != maxcoord.y
    elif direction is 'l':
      return self.x != 0
    elif direction is 'r':
      return self.x != maxcoord.x
    elif direction is 'ul':
      return self.y != 0 and self.x != 0
    elif direction is 'ur':
      return self.y != 0 and self.x != maxcoord.x
    elif direction is 'dl':
      return self.y != maxcoord.y and self.x != 0
    elif direction is 'dr':
      return self.y != maxcoord.y and self.x != maxcoord.x
    return False

  def up(self):
    self.y = self.y - 1

  def down(self):
    self.y = self.y + 1

  def left(self):
    self.x = self.x - 1

  def right(self):
    self.x = self.x + 1

  def equals(self, other):
    return self.x == other.x and self.y == other.y
```

1.3. Pradiniai duomenys ir rezultatai

duom.txt

```
2
XXXXXXXXX
X   X   X
X *     X
X   X   X
XXXXXXXXX
X   X
X   X
X   X
XXXXX
_____
XXXXX
X   X
X * X
X   X
XXXXX
_____
```

rez.txt

```
XXXXXXXXX
X###X###X
X#*#####X
X###X###X
XXXXXXXXX
X   X
X   X
X   X
XXXXX
_____
XXXXX
X###X
X#*#X
X###X
XXXXX
_____
```