Міністерство освіти і науки України

НТУУ «Київський політехнічний інститут»

Фізико-технічний інститут

# Програмування 4

# Лабораторна робота №10

«Індивідуальне завдання»

**Виконав:**

Студент II курсу ФТІ групи ФЕ-81

Безуглий Ростислав Сергійович

2020

1. Завдання лабораторної роботи

Написати програму для пошуку виходу з лабіринту, що задається користувачем (порядок до 50 елементів).

1. Аналіз умови задачі.

Для взаємодії з користувачем було обрано зчитування з файлу \*.txt, що надає користувач. Інструкція щодо оформлення такого файлу надається на початку роботи програми. Результат роботи доступний одразу та у окремому файлі результату для наданої схеми. Для пошуку найкоротшого з можливих шляхів було обрано алгоритм Лі (Lee algorithm).

* Файловий ввід, перевірка коректності вводу.
* Алгоритм Лі.
* Файловий вивід.

1. Код реалізації

|  |
| --- |
| Python\_lab10v2.py |
| description = "This program is based on Lee algorithm and\nrequires a .txt file with square text matrix\nwith size limit of 50, were:\n----------------------------------------------\nSPACE will be read as FREE SPACE\nS will be read as START position\nF will be read as FINISH position\nANY OTHER CHARACTERS will be read as OBSTACLES\n----------------------------------------------\n(See example.txt)"  print(description)  class LineLenError(Exception):  pass  class KeyPointError(Exception):  pass  class SizeLimitError(Exception):  pass  def wave(labyrinth, row, col, level=0):  labyrinth[row][col] = level  if row!=0:  if ((labyrinth[row-1][col] != -1 and labyrinth[row-1][col] > level) or labyrinth[row-1][col] == 0):  wave(labyrinth, row-1, col, level+1)  if col!=n-1:  if ((labyrinth[row][col+1] != -1 and labyrinth[row][col+1] > level) or labyrinth[row][col+1] == 0):  wave(labyrinth, row, col+1, level+1)  if row!=n-1:  if ((labyrinth[row+1][col] != -1 and labyrinth[row+1][col] > level) or labyrinth[row+1][col] == 0):  wave(labyrinth, row+1, col, level+1)  if col!=0:  if ((labyrinth[row][col-1] != -1 and labyrinth[row][col-1] > level) or labyrinth[row][col-1] == 0):  wave(labyrinth, row, col-1, level+1)  def build\_path(labyrinth, path, row, col, level):  path.append([row, col])  if level == 0:  return path  if row!=0:  if level-labyrinth[row-1][col] == 1:  return build\_path(labyrinth, path, row-1, col, level-1)  if col!=n-1:  if level-labyrinth[row][col+1] == 1:  return build\_path(labyrinth, path, row, col+1, level-1)  if row!=n-1:  if level-labyrinth[row+1][col] == 1:  return build\_path(labyrinth, path, row+1, col, level-1)  if col!=0:  if level-labyrinth[row][col-1] == 1:  return build\_path(labyrinth, path, row, col-1, level-1)  labyrinth = []  original = []  pointF = [0,0]  pointS = [0,0]  while(1):  labyrinth.clear()  original.clear()  foundF = 0  foundS = 0  try:  filename = str (input("Enter file name: "))  file = open(filename, 'r')  except Exception:  print("Unable to find given path.")  continue  try:  pre = str (file.readline())  pre = pre.rstrip('\n')  n = len(pre)  if n>50:  raise SizeLimitError  file.seek(0)  for i in range(0,n):  pre = str (file.readline())  pre = pre.rstrip('\n')  if len(pre)!=n:  raise LineLenError  labyrinth.append(list (pre))  original.append(list (pre))  if pre.find('S')!=-1:  foundS+=1  pointS[0] = i  pointS[1] = pre.find('S')  if pre.find('F')!=-1:  foundF+=1  pointF[0] = i  pointF[1] = pre.find('F')  if foundF!=1 or foundS!=1:  raise KeyPointError  except SizeLimitError:  print("Size limit is exceeded. Length of first line is {}. (Expected length <50)".format(n))  file.close()  continue  except LineLenError:  print("Wrong length for line {}. (Expected length - {} is based on first line)".format(i+1, n))  file.close()  continue  except KeyPointError:  print("Wrong number of key points. (There are {} Start and {} Finish points)".format(foundS, foundF))  file.close()  continue  file.close()  break  for i in range(n):  for j in range(n):  if labyrinth[i][j]==' ' or labyrinth[i][j]=='S' or labyrinth[i][j]=='F':  labyrinth[i][j] = 0  else:  labyrinth[i][j] = -1  wave(labyrinth, pointS[0], pointS[1])  labyrinth[pointS[0]][pointS[1]] = 0  path = []  build\_path(labyrinth, path, pointF[0], pointF[1], labyrinth[pointF[0]][pointF[1]])  file = open("result\_"+filename, 'w')  for cell in path:  if cell != pointS and cell != pointF:  original[cell[0]][cell[1]] = '•'  for i in range(n):  pre = ''  for j in range(n):  pre+=str (original[i][j])  pre = pre.strip('[]')  file.writelines(pre+'\n')  print(pre)  if path.count(pointS)==0:  file.writelines("There is no path from Start to Finish :(")  print("There is no path from Start to Finish :(")  file.close()  print("Result was written to:", "result\_"+filename)  input("Press Enter to exit...") |

1. Виконання програми

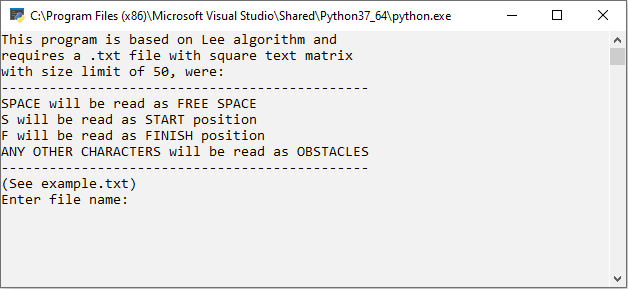


Рис.1.1. Опис роботи програми та вимог до вхідних даних.

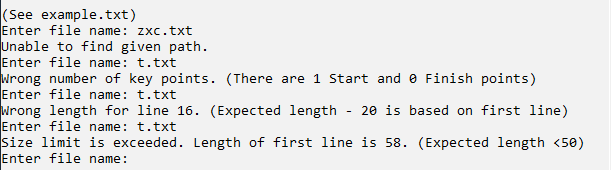


Рис.1.2. Перевірка коректності вхідних даних.

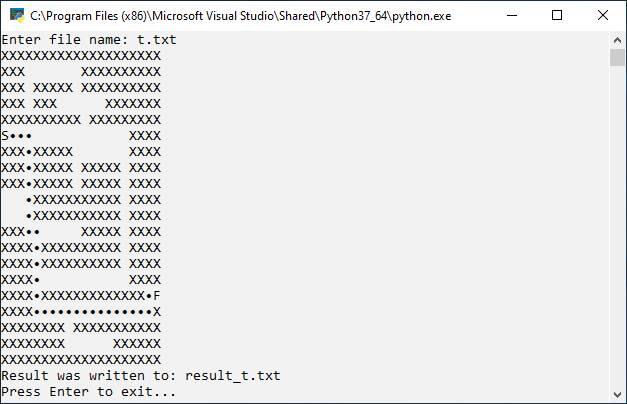


Рис.1.3. Результат роботи програми в діалоговому вікні.

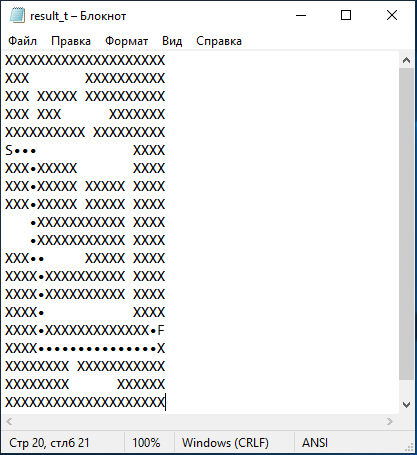


Рис.1.4. Результат роботи програми в вихідному текстовому файлі.