**13. praktiskais darbs. 2. semestris**

**PU1. uzdevums 1.līmenis.**

Sastādīt programmu, kas uzģenerē korekti aizpildītu Sudoku spēles lapiņu, ja ir zināms cik skaitļu ir atvērti.

1.līmenis - lapiņa sākotnēji uzģenerēta (aizpildīta) korekti ar visiem skaitļiem un tad izvēlas atklātos skaitļus un izdzēš pārējos.

2.līmenis - uzģenerē atvērtos skaitļus (visus uzreiz vai secīgi pa vienam) un pēc tam pārbauda, vai šis komplekts ir korekti atrisināms, bet ja nē, tad atkārto procesu, kamēr iegūst korekti aizpildāmu komplektu.

**Kods:**

# Programmas nosaukums: Sudoku spēlē ģenerācija (1.LĪMENIS)

# PU1. (1MPR13\_Vladislavs\_Babaņins)

# Uzdevuma formulējums: Sastādīt programmu, kas uzģenerē korekti aizpildītu Sudoku spēles lapiņu, ja ir zināms cik skaitļu ir atvērti.

# 1.līmenis - lapiņa sākotnēji uzģenerēta (aizpildīta) korekti ar visiem skaitļiem un tad izvēlas atklātos skaitļus un izdzēš pārējos.

# 2.līmenis - uzģenerē atvērtos skaitļus (visus uzreiz vai secīgi pa vienam) un pēc tam pārbauda, vai šis komplekts ir korekti atrisināms, bet ja nē, tad atkārto procesu,

# kamēr iegūst korekti aizpildāmu komplektu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

"""

1.LĪMENIS TIKA IZPILDĪTS AR TĀ SAUCĀMO "BACKTRACKING" ALGORITMU.

"""

import random

def check\_array\_rows\_and\_columns(a):

# Pārbauda vai matricā katra rindā un kolonnā visi skaitļi ir dažādi, izmantojot kopas.

# Atgriež True, ja visi skaitļi visas rindas un kolonnas ir dažādi.

# Atgriež False, ja ir kādi divi skaitļi kāda rinda vai kolonna kuri sakrīt.

# a - divdimensijas masīvs (matrica).

for i in range(9):

# Pārbauda, vai katrā rindā nav skaitļu dublikātu (nav vienādu skaitļu).

# Izmantojam kopas. Ja kopā nav ar gārumu 9, tad ir cipari kas atkartojas.

if len(set(a[i])) != 9:

print(str(i + 1) + ". rindā ir cipari, kas atkārtojas.") # Izvadam, kur tika atrāsta kļūda.

return False

for j in range(9):

# Pārbauda, vai katrā kolonnā nav skaitļu dublikātu (nav vienādu skaitļu).

# Izmantojam kopas. Ja kopā nav ar gārumu 9, tad ir cipari kas atkartojas.

col\_numbers = [a[i][j] for i in range(9)]

if len(set(col\_numbers)) != 9:

print(str(j + 1) + ". kolonnā ir cipari, kas atkārtojas.")

return False

return True

def check\_submatrix(matrix, i, j):

# Atgriež True ja 3x3 apakšmatricā (apakšmatricas ir paradītas zemāk) visi skaitļi ir dažādi.

# Atgriež False ja 3x3 apakšmatricā kādi divi skaitļi ir vienādi.

# Sudoku 3x3 apakšmatricas.

# matrix - pilnā 9x9 matrica (divdimensijas masīvs).

# i - no kuras rīndas sāksim (int).

# j - no kuras kolonnas sāksim (int).

submatrix = []

for row in range(i, i + 3):

for col in range(j, j + 3):

submatrix.append(matrix[row][col])

return len(set(submatrix)) == 9

def check\_3x3\_submatrixes(a):

# Pārbaudam katru 3x3 apakšmatricu un paziņojam lietotājam kāda apakšmatrica skaitļi ir dažādi un kurā ir vienādi.

# Izsauc check\_submatrix(a, i, j) un paziņo lietotājam "Visi skaitļi apakšmatricā [{i//3 + 1}][{j//3 + 1}] ir atšķirīgi.",

# vai "Ne visi skaitļi apakšmatricā [{i//3 + 1}][{j//3 + 1}] ir atšķirīgi."

# Atgriež True, ja nav nevienas apakšmatricas, kurai iekšā ir divi vienādi skaitļi.

# Atgriež False, ja ir kaut viena apakšmatrica, kurai iekšā ir divi vienādi skaitļi.

# a - divdimensiju masīvs.

# Sudoku deviņas apakšmatricas:

# [0][0] [0][1] [0][2] [0][3] [0][4] [0][5] [0][6] [0][7] [0][8]

# [1][0] [1][1] [1][2] [1][3] [1][4] [1][5] [1][6] [1][7] [1][8]

# [2][0] [2][1] [2][2] [2][3] [2][4] [2][5] [2][6] [2][7] [2][8]

# [3][0] [3][1] [3][2] [3][3] [3][4] [3][5] [3][6] [3][7] [3][8]

# [4][0] [4][1] [4][2] [4][3] [4][4] [4][5] [4][6] [4][7] [4][8]

# [5][0] [5][1] [5][2] [5][3] [5][4] [5][5] [5][6] [5][7] [5][8]

# [6][0] [6][1] [6][2] [6][3] [6][4] [6][5] [6][6] [6][7] [6][8]

# [7][0] [7][1] [7][2] [7][3] [7][4] [7][5] [7][6] [7][7] [7][8]

# [8][0] [8][1] [8][2] [8][3] [8][4] [8][5] [8][6] [8][7] [8][8]

for i in range(0, 9, 3):

for j in range(0, 9, 3):

if check\_submatrix(a, i, j):

pass

#print(f"Visi skaitļi apakšmatricā [{i//3 + 1}][{j//3 + 1}] ir atšķirīgi.")

else:

#print(f"Kļūda! Apakšmatricā [{i//3 + 1}][{j//3 + 1}] ir divi vienādi skaitļi!")

return False

return True

def generate\_sudoku():

# Izveidojam 9x9 nulles matricu.

matrix = []

for i in range(9):

row = [0] \* 9

matrix.append(row)

# Aizpildam matricu, sākot no augšējā kreisā stūra.

fill\_matrix(matrix, 0, 0)

return matrix

def fill\_matrix(matrix, i, j):

# Aizpildām nulles matricu ar (backtracking algorithm) algoritmu tā, lai visas rindas būtu atšķirīgi skaitli.

if i == 9:

return True

# Aprēķina nākamo "šūnu" indeksus.

next\_i = i + (j + 1) // 9

next\_j = (j + 1) % 9

# Izveidot sarakstu ar veseliem skaitļiem no 1 līdz 9.

list\_random = list(range(1, 10))

# Sajaucam sarakstā vērtības.

random.shuffle(list\_random)

# Mēgīnām ielikt pēc kārtas katru ciparu no nejauši sajaukta saraksta.

for number in list\_random:

# Pārbaudam, vai skaitlis ir derīgs (ar karogu "valid").

valid = True

# Pārbaudam rindas.

for k in range(9):

if matrix[i][k] == number:

valid = False # Ja sakrīt kāds skaitlīs rindā ar izvelēto skaitli, tad karogs ir False, vajag paņemt citu skaitli no list\_random (ņēm pēc kārtas).

break

# Pārbaudam kolonnas.

for k in range(9):

if matrix[k][j] == number:

valid = False # Ja sakrīt kāds skaitlīs kolonna ar izvelēto skaitli, tad karogs ir False, vajag paņemt citu skaitli no list\_random (ņēm pēc kārtas).

break

# Pārbaudam 3x3 noteiktas apakšmatricas.

sub\_i = i // 3 \* 3 # "apakš\_i"

sub\_j = j // 3 \* 3 # "apakš\_j"

for k in range(sub\_i, sub\_i + 3):

for l in range(sub\_j, sub\_j + 3):

if matrix[k][l] == number:

valid = False

break

# Ja skaitlis ir derīgs, aizpildam to "šūnu" un rekursīvi sākam aizpildīt nākamo "šūnu".

if valid:

matrix[i][j] = number

if fill\_matrix(matrix, next\_i, next\_j):

return True

# Ja esam izmēģinājuši visus skaitļus un neviens nedēr, tad jāatkāpjas par vienu soli atpakāļ, un jau citus skaitļus likt. (backtracking algorithm)

matrix[i][j] = 0

return False

def delete\_cells(matrix, delete\_count):

# Izveidojam matricu, kas piepildīta ar nullēm

result = []

for i in range(len(matrix)):

row = [0] \* len(matrix[0])

result.append(row)

# Aizpildam noteiktu skaitu "šūnu" ar skaitļiem no 1 līdz 9

fill\_count = len(matrix) \* len(matrix[0]) - delete\_count

for k in range(fill\_count):

i = random.randint(0, 8)

j = random.randint(0, 8)

while result[i][j] != 0:

i = random.randint(0, 8)

j = random.randint(0, 8)

result[i][j] = matrix[i][j]

return result

def matrix\_to\_string\_float\_3x3(matrix):

# Atgriež matricas virknes attēlojumu, kur katra rinda ir atdalīta ar \n un izlīdzināta atbilstoši maksimālās vērtības garumam.

# Ja vērtība ir vesels skaitlis, tā tiek parādīta bez komata. Pretējā gadījumā tas tiek parādīts ar decimālzīmi.

# Funkcija arī atrod maksimālo vērtību garumu matricā un aizpilda nepieciešamās atstarpes " ", lai tās glīti izlīdzinātu (glītas atkāpes).

# Ja matricā ir 0, tas tiek aizstāts ar simbolu ∙.

# matrix - matrica (divdimensiju masīvs ar izmēriem n x m).

# Piemērs, kāda veida tiek atgriezta simbolu virkne:

# 1 6 3 9 3 4 3 6 6

# 4 9 9 2 7 3 9 9 5

# 3 5 9 5 2 7 9 7 4

#

# 4 6 6 3 3 8 2 5 3

# 1 5 6 8 9 2 4 8 3

# 9 3 9 6 8 7 2 8 2

#

# 7 4 9 3 9 3 7 1 1

# 1 3 5 2 6 3 1 3 1

# 6 5 3 8 9 7 7 1 8

rindas = len(matrix)

kolonnas = len(matrix[0])

max\_len = 0

for i in range(rindas): # atrod max\_len, lai noteiktu nepieciešamo atkāpi.

for j in range(kolonnas):

number = matrix[i][j]

if number == int(number):

value\_len = len(str(int(number)))

else:

value\_len = len(str(float(number)))

if value\_len > max\_len:

max\_len = value\_len

# Izveido matricas virknes attēlojumu, kur katra rinda tiek atdalīta ar \n un izlīdzināta atbilstoši maksimālās vērtības garumam

sv = ""

for i in range(rindas):

for j in range(kolonnas):

number = matrix[i][j]

if number == 0:

number = "∙"

elif number == int(number):

number = int(number)

else:

number = str(float(number))

atkape = " " \* (max\_len - len(str(number)))

sv += atkape + str(number)

if j < kolonnas - 1 and (j + 1) % 3 != 0:

sv = sv + " "

elif j < kolonnas - 1 and (j + 1) % 3 == 0:

sv = sv + " "

sv = sv + "\n"

if (i + 1) % 3 == 0 and i < rindas - 1:

sv = sv + "\n"

return sv

def is\_natural\_or\_zero(n):

# Pārbauda vai simbolu virkne ir naturāls skaitlis vai nulle, vai nav

# Ja ir naturāls skaitlis vai nulle, tad True. Ja nav tad False.

# n - simbolu virkne, kuru pārbauda.

if str(n).isdigit() and float(n) == int(n) and int(n) >= 0:

return True

else:

return False

# ---------------------------------------------------------

# Galvenā programmas daļa

# ---------------------------------------------------------

b = input("Ievadiet cik skaitļu ir atvērti Sudoku lapiņai ==> ")

while not is\_natural\_or\_zero(b) or int(b) > 81:

b = input("Kļūda! Ievadiet veselu skaitli no 0 līdz 81. Ievadiet cik skaitļu ir atvērti Sudoku lapiņai ==> ")

b = int(b)

izd = 81 - b

a = generate\_sudoku()

if check\_array\_rows\_and\_columns(a) and check\_3x3\_submatrixes(a):

print("Korekti aizpildīta Sudoku spēlēs lapiņa:")

print()

else:

print("Nav korekti aizpildīta Sudoku spēlēs lapiņa:")

print(matrix\_to\_string\_float\_3x3(a))

print("-------------------\n")

c = delete\_cells(a, izd)

print("Sudoku lapiņa ar nejauši izdzēstiem skaitļiem:\n")

print(matrix\_to\_string\_float\_3x3(c))

**Testa piemēri:**

1)

A picture containing text, screenshot

Description automatically generated

2)

A screenshot of a computer

Description automatically generated

3)

A screenshot of a computer

Description automatically generated

4)

A picture containing text, screenshot

Description automatically generated

5)

A screenshot of a computer

Description automatically generated

6)

A screenshot of a computer screen

Description automatically generated with medium confidence

7)

A screenshot of a computer screen

Description automatically generated with medium confidence