**Štrāzena algoritms**

**2.izcilības (desmitnieka) uzdevums**

Sastādīt programmu, kas realizē matricu ātro reizināšanas (Štrāzena) algoritmu.

**Kods:**

# Programmas nosaukums: Štrāzena algoritms

# 2.izcilības (desmitnieka) uzdevums

# Uzdevuma formulējums: Sastādīt programmu, kas realizē matricu ātro reizināšanas (Štrāzena) algoritmu.

# Programmas autors: Vladislavs Babaņins

# Versija 1.0

"""

Kods realizē Štrāzena algoritmu kvadrātisku matricu reizināšanai.

Šis algoritms sadala matricas reizināšanu mazākās matricās un izmanto rekursiju.

"""

import numpy as np

def pad\_matrix(M):

# Funkcija aprēķina ar kādiem izmēriem jābut jaunai matricai, lai izmērs būtu pakāpē 2.

# Atgriež sakotnēju matricu kreisā augšēja stūri un pēdejas rindas/kolonnas varētu būt nulles, ja bija nepieciešāms matricu pielāgot.

(a, b) = M.shape

# Uzzinām kurš ir garāks. Matricas garums vai platums ir garāks.

size = a

if b > size:

size = b

# Pārbaudam, vai izmērs ir pakāpē 2 (vai to var sadalīt, vai būs nepieciešāms likt liekas nulles, lai algoritms strādātu).

is\_power\_of\_two = (size - 1)

if size == 0 and is\_power\_of\_two == 0:

is\_power\_of\_two = True

else:

is\_power\_of\_two = False

# Ja nepieciešams, tad izrēķinām cik ir nepieciešām likt liekas rindiņas/kolonnas (size), lai izmērs ir pakāpē 2.

if not is\_power\_of\_two:

power\_of\_two = 1

while power\_of\_two < size:

power\_of\_two = power\_of\_two \* 2

size = power\_of\_two

# Izveidojam matricu ar nepieciešamam liekam nullem, lai matricas izmērs ir pakāpē 2.

padded = np.zeros((size, size)) # Izveidojam jaunu kvadrātveida matricu ar nullēm ar nepieciešamiem izmēriem (izmērs kura pakāpē ir kāda 2).

padded[:a, :b] = M # Ievietojam ievadīto sakotnēju matricu M jaunās matricas augšējā kreisajā stūrī.

return padded

def strassen(A, B):

# Funkcija, lai veiktu matricas reizināšanu, izmantojot Štrāzena algoritmu.

# Atgriež matricu C = A\*B pēc Štrāzena algoritma. Varētu rasties liekas rindas/kolonnas, tāpēc pēc tam noņemam tos, ja ir vajadzīgi.

# Pievienojam nulles matricam, ja ir vajadzīgi, lai matricas izmērs ir pakāpē 2 (izmērs kura pakāpē ir kāda 2).

A = pad\_matrix(A)

B = pad\_matrix(B)

# Ja matricas izmērs ir 1x1, atgriežam reizinājumu (rekursijas visdziļāka iespējama situācija).

# Rekursijas tā sauksim "pamatgadījums".

n = A.shape[0]

if n == 1:

return A \* B

half = n // 2

# Sadalam matricu "kvadrantos".

# Matricai A

# Paņemam pusi "pirmās" rindas un pusi "pirmās" kolonnas.

a = A[:half, :half]

# Paņemam pusi "pirmās" rindas un pusi "otrās" kolonnas.

b = A[:half, half:]

# Paņemam pusi "otrās" rindas un pusi "pirmās" kolonnas.

c = A[half:, :half]

# Paņemam pusi "otrās" rindas un pusi "otras" kolonnas.

d = A[half:, half:]

# Matricai B

# Paņemam pusi "pirmās" rindas un pusi "pirmās" kolonnas.

e = B[:half, :half]

# Paņemam pusi "pirmās" rindas un pusi "otrās" kolonnas.

f = B[:half, half:]

# Paņemam pusi "otrās" rindas un pusi "pirmās" kolonnas.

g = B[half:, :half]

# Paņemam pusi "otrās" rindas un pusi "otras" kolonnas.

h = B[half:, half:]

# Veicam Štrāzena reizināšanu uz mazākām matricām rekursīvi.

p1 = strassen(a, f - h)

p2 = strassen(a + b, h)

p3 = strassen(c + d, e)

p4 = strassen(d, g - e)

p5 = strassen(a + d, e + h)

p6 = strassen(b - d, g + h)

p7 = strassen(a - c, e + f)

r = p5 + p4 - p2 + p6

s = p1 + p2

t = p3 + p4

u = p5 + p1 - p3 - p7

C = np.empty((n, n))

C[:half, :half] = r

C[:half, half:] = s

C[half:, :half] = t

C[half:, half:] = u

return C # Atgriež matricu C = A\*B pēc Strasena algoritma. Varētu rasties liekas rindas/kolonnas, tāpēc pēc tam noņemam tos, ja ir vajadzīgi.

def unpad\_matrix(M, row\_count, col\_count):

# Atgriež matricu M bet ar nodzēstam pēdējam rindam row\_count skaitā un ar nodzēstam pēdējam kolonnam col\_count skaitā.

# M - matrica, kurai gribam noņemt row\_count rindas skaitu un col\_count kolonnas skaitu.

# row\_count - cik rindas gribam noņemt no matricas M (pēdējas rindas).

# col\_count - cik kolonnas gribam noņemt no matricas M (pēdējas kolonnas).

if row\_count == 0 and col\_count == 0: # Ja rindas un kolonnu skaits, kuru gribam nodzēst ir 0, tad atgriežām sakotnēju matricu.

return M

# Nosakam pēdējo rindu un kolonnu skaitu, cik vajadzīgi "unpad" (nodzēst liekas nulles rindas/kolonnas), pamatojoties uz ievadītiem skaitliem row\_count un col\_count.

last\_row = M.shape[0] - row\_count # Nosakam pēdējo rindu skaitu, atņemot ievadītu rindu skaitu (nonēmam tik, cik tika ievadīts row\_count'ā).

last\_col = M.shape[1] - col\_count # Nosakam pēdējo kolonnu skaitu, atņemot ievadītu kolonnu skaitu (nonēmam tik, cik tika ievadīts col\_count'ā).

# Noņemam norādīto rindu un kolonnu skaitu

unpadded = M[:last\_row, :last\_col]

return unpadded

def convert\_matrix\_elements\_to\_real(matrix):

# Konvertējam matricas elementus reālos skaitļos (decimal ar punktu).

# Atgriež matricu, kurai visi elementi ir float.

# matrix - matrica, kuras visas vērtības būs konvērtētas float skaitļos.

return np.array(matrix, dtype=float)

def convert\_matrix\_elements\_to\_int(matrix):

# Konvertējam matricas elementus veselos skaitļos (int).

# Atgriež matricu, kurai visi elementi ir int.

# matrix - matrica, kuras visas vērtības būs konvērtētas int skaitļos.

return np.array(matrix, dtype=int)

def create\_matrix():

# Metode, kas prasa lietotājam ievadīt kvadrātiskas matricas izmēru un prasa ievadīt katru matricas elementu.

# Atgriež kvadrātisku matricu ar visām vertībam, kuru ievadīja lietotājs.

# Prasa lietotājam ievadīt kvadrātiskas matricas izmēru.

n = int(input("Ievadiet kvadrātiskas matricas izmēru ==> "))

# jo vajadzīgi pielāgot ar nulles rindam/kolonnam un pēc tam viņus noņemt, tātad liekas darbības būs.

# Arī nav ieteicams izmantot algoritmu ja matricas ir vajadzīgi pielagot (labāk uzreiz ievadīt, kā pielāgotus).

# Jo tas varētu aizņēmt laiku ļoti lielām matricam.

matrix = np.zeros((n, n)) # Tukša matrica ar ievadītiem izmēriem

# Prasa lietotājam ievadīt katru vērtību matricā.

for i in range(n):

for j in range(m):

matrix[i][j] = float(input(f"Ievadiet skaitlisko vērtību elementam ({i+1},{j+1}) ==> ")) # varam arī prasīt int ievādīt.

return matrix

def row\_number(matrix):

# Atgriež kopējo rindu skaitu matrica matrix.

# matrix - divdimensijas numpy masīvs.

# Izveidojam mainīgo, lai saskaitītu rindas skaitu.

row\_count = 0

for row in matrix: # Eterejam cauri rindam un skaitam tos.

row\_count += 1

# Atgriež kopējo rindu skaitu matricā.

return row\_count

def column\_number(matrix):

# Atgriež kopējo kolonnu skaitu matrica matrix.

# matrix - divdimensijas numpy masīvs.

# Izveidojam mainīgo, lai saskaitītu kolonnu skaitu.

column\_count = 0

# Izvēlāmies pirmo rindu

row = matrix[0]

# Ejam cauri katras rindas elementam

for column in row:

# Katrs rindas elements apzīmē vienu kolonnu, tāpēc palielinam skaitu par vienu

column\_count += 1

# Atgriež kopējo kolonnu skaitu matricā.

return column\_count

def result\_with\_padding(A, B, rinda\_sk\_A, kolonnu\_sk\_A, rinda\_sk\_B, kolonnu\_sk\_B):

# Atgriež matricu bez liekam rindam un kolonnam, kura tika izveidota ar Štrāzena algoritmu reizinot A ar B.

# Atgriež arī, ka visi elementi ir int, to var noņemt pēc vajadzības. (!)

# rinda\_sk\_A - rindas skaits matrica A

# kolonnu\_sk\_A - kolonnu skaits matrica A

# rinda\_sk\_B - rindas skaits matrica B

# kolonnu\_sk\_B - kolonnu skaits matrica B

# A - matrica A

# B - matrica B

# Atgriež matricu C = A\*B (pēc Štrāzena algoritma) bez liekam nullem.

if kolonnu\_sk\_A == rinda\_sk\_B: # Tikai tad varam sareizināt divas matricas (ja rindas sk 1. matrica ir vienāds ar kolonnu sk 2. matrica).

strasen = strassen(A, B) # Izmantojam Štrāzena algoritmu.

rinda\_sk\_strasen = row\_number(strasen) # Noteicam cik rindas ir matricai, kas bija iegūta ar Štrāzena algoritmu (tur var rastīties liekas rindas, lai varētu izmantot Štrāzena algoritmu).

kolonnu\_sk\_strasen = column\_number(strasen) # Noteicam cik kolonnas ir matricai, kas bija iegūta ar Štrāzena algoritmu (tur var rastīties liekas kolonnas, lai varētu izmantot Štrāzena algoritmu).

pad\_row = 0 # par cik vienībam vajag unpad (nodzēst liekas) rindas

pad\_column = 0 # par cik vienībam vajag unpad (nodzēst liekas) kolonnas

if rinda\_sk\_strasen != rinda\_sk\_A: # Ja nesakrīt matricas izmērs (rindu skaits) ar to, kuru būtu nepieciešāms dabūt, pēc matricas reizināšanas likumiem, tad nosakam cik ir tā starpība.

pad\_row = rinda\_sk\_strasen - rinda\_sk\_A # Ja rindas skaits pēc Štrāzena algoritma palielinājies, tad noņēmam tik rindas.

if kolonnu\_sk\_strasen != kolonnu\_sk\_B: # Ja nesakrīt matricas izmērs (kolonnu skaits) ar to, kuru būtu nepieciešāms dabūt, pēc matricas reizināšanas likumiem, tad nosakam cik ir tā starpība.

pad\_column = kolonnu\_sk\_strasen - kolonnu\_sk\_B # Ja kolonnu skaits pēc Štrāzena algoritma palielinājies, tad noņēmam tik kolonnas.

res\_strasen = unpad\_matrix(strasen, pad\_row, pad\_column) # Noņēmam liekas rindas pad\_row skaitā un noņēmam liekas kolonnas pad\_column skaitā.

res\_strasen = convert\_matrix\_elements\_to\_int(res\_strasen) # (!) Lai butu visi skaitļi veseli. Vajag noņemt, ja gribam float vērtības.

return res\_strasen # Atgriež matricu C = A\*B (pēc Štrāzena algoritma) bez liekam nullem.

else:

return None

#"Nevar sareizināt! Matricas A kolonnu skaits nav vienāds ar matricas B rindu skaitu!"

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

# Galvenā programmas daļa

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

# jo vajadzīgi pielāgot ar nulles rindam/kolonnam un pēc tam viņus noņemt, tātad liekas darbības.

print("Pirmās kvadrātiskas matricas izveidošana.")

A = create\_matrix()

print("\nOtrās kvadrātiskas matricas izveidošana.")

B = create\_matrix()

rinda\_sk\_A = row\_number(A) # Skaitam rindas skaitu matricā A

kolonnu\_sk\_A = column\_number(A) # Skaitam kolonnas skaitu matricā A

rinda\_sk\_B = row\_number(B) # Skaitam rindas skaitu matricā B

kolonnu\_sk\_B = column\_number(B) # Skaitam kolonnas skaitu matricā B

# Izmantojam Štrāzena algoritmu un pielagojam matricu un atgriežām bez liekām nullem.

strasen\_result = result\_with\_padding(A, B, rinda\_sk\_A, kolonnu\_sk\_A, rinda\_sk\_B, kolonnu\_sk\_B)

if strasen\_result is None: # Ja nevaram sareizināt matricas.

print("Matricas A rindu skaitam jāsakrīt ar matricas B kolonnu skaitu, lai varētu sareizināt matricas!")

else:

print()

print(A)

print("\*")

print(B)

print("=")

print(strasen\_result) # Parāda matricas reizinājuma rezultātu.

# Visi elementi tiek pārverti par int! Tāpēc ja gribat reizināt arī float skaitļus, tad vajag noņemt")

# no result\_with\_padding(rinda\_sk\_A, kolonnu\_sk\_A, rinda\_sk\_B, kolonnu\_sk\_B) vajag noņemt res\_strasen = convert\_matrix\_elements\_to\_int(res\_strasen)

# šeit tas ir realizēts, lai testa piemēros būtu vieglāk pārbaudīt.

'''

# TESTĒŠANAI (ērtak un ātrak šeit vērtības rakstīt).

#A = np.array([[1, 2], [3, 4]])

#B = np.array([[5, 6], [7, 8]])

A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

B = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

rinda\_sk\_A = row\_number(A)

kolonnu\_sk\_A = column\_number(A)

rinda\_sk\_B = row\_number(B)

kolonnu\_sk\_B = column\_number(B)

strasen\_result = result\_with\_padding(rinda\_sk\_A, kolonnu\_sk\_A, rinda\_sk\_B, kolonnu\_sk\_B)

if strasen\_result is None: # == False

print("Matricas A rindu skaitam jāsakrīt ar matricas B kolonnu skaitu, lai varētu sareizināt matricas!")

else:

print()

print(A)

print("\*")

print(B)

print("=")

print(strasen\_result)

# Visi elementi tiek pārverti par int! Tāpēc ja gribat reizināt arī float skaitļus, tad vajag noņemt")

# no result\_with\_padding(rinda\_sk\_A, kolonnu\_sk\_A, rinda\_sk\_B, kolonnu\_sk\_B) vajag noņemt res\_strasen = convert\_matrix\_elements\_to\_int(res\_strasen)

# šeit tas ir realizēts, lai testa piemēros būtu vieglāk pārbaudīt.

'''

'''

# TESTĒŠANAI

# Izveidojam divas nejaušas kvadrātveida matricas ar veseliem skaitļiem (var arī ar float, tikai būtu grūtak salidzināt, jo float nav precīzs, tāpēc būtu vajadzīgi ņemt kādu precizitāti).

# bet funkcija strāda arī ar float.

matrix\_a = np.random.randint(1, 10, (10, 10)) # Matricas izmēri ir 10x10.

matrix\_b = np.random.randint(1, 10, (10, 10)) # Matricas izmēri ir 10x10.

rinda\_sk\_a = row\_number(matrix\_a)

kolonnu\_sk\_a = column\_number(matrix\_a)

rinda\_sk\_b = row\_number(matrix\_b)

kolonnu\_sk\_b = column\_number(matrix\_b)

print(matrix\_a)

print("\*")

print(matrix\_b)

# Veicam matricas reizināšanu, izmantojot Strassena algoritmu.

result\_strassen = strassen(matrix\_a, matrix\_b) # Izmantojot Štrāzena algoritmu

stras = result\_with\_padding(matrix\_a, matrix\_b, rinda\_sk\_a, kolonnu\_sk\_a, rinda\_sk\_b, kolonnu\_sk\_b)

# stras = unpad\_matrix(result\_strassen, ) # Noņemam liekas rindiņas un kolonnas ar 0, kuri varētu izveidoties.

# stras = convert\_matrix\_elements\_to\_int(stras) # Konvertējam matricas visus elementus float -> int

print("\nRezultāts izmantojot Štrāzena algoritmu:")

print(stras)

print("\nRezultāts izmantojot iebūvētās numpy funkcijas:")

# Veicam matricas reizināšanu, izmantojot np.matmul (lai pārbaudītu Štrāzena funkciju, vai pareizi strāda).

result\_matmul = np.matmul(matrix\_a, matrix\_b) # Iebuvēta reizināšana pārbaudei.

result\_matmul = convert\_matrix\_elements\_to\_int(result\_matmul) # Konvertējam matricas visus elementus float -> int

# (To daram, lai salidzinātu matricas, jo float elementus nevar salidzināt precīzi (tur vajag ņemt kādu precizitāti, tāpēc pārbaudam tikai int skaitļiem)

# Bet algoritms strāda ari kad float.

print(result\_matmul)

# Pārbauda,, vai matricas ir vienādas.

if np.array\_equal(stras, result\_matmul): # Varētu izveidot pašu definētu funkciju, bet tas ir tikai testēšanai, tāpēc izmantoju gatavu no numpy bibliotēkas

print("\nMatricas ir vienādas.")

else:

print("\nMatricas nav vienādas")

'''

**Testa piemēri:**

1) Matrica 2x2 reizinot ar matricu 2x2 (lietotājs ievada)

A picture containing text, screenshot, font

Description automatically generated

2) Matrica 10x10 reizinot ar matricu 10x10

A screenshot of a computer

Description automatically generated

3) Matrica 6x6 reizinot ar matricu 6x6

A picture containing text, screenshot, font, design

Description automatically generated

4) Matrica 100x100 reizinot ar matricu 100x100

A screenshot of a computer

Description automatically generated

5) Matrica 100x100 reizinot ar matricu 100x100

A screenshot of a computer

Description automatically generated