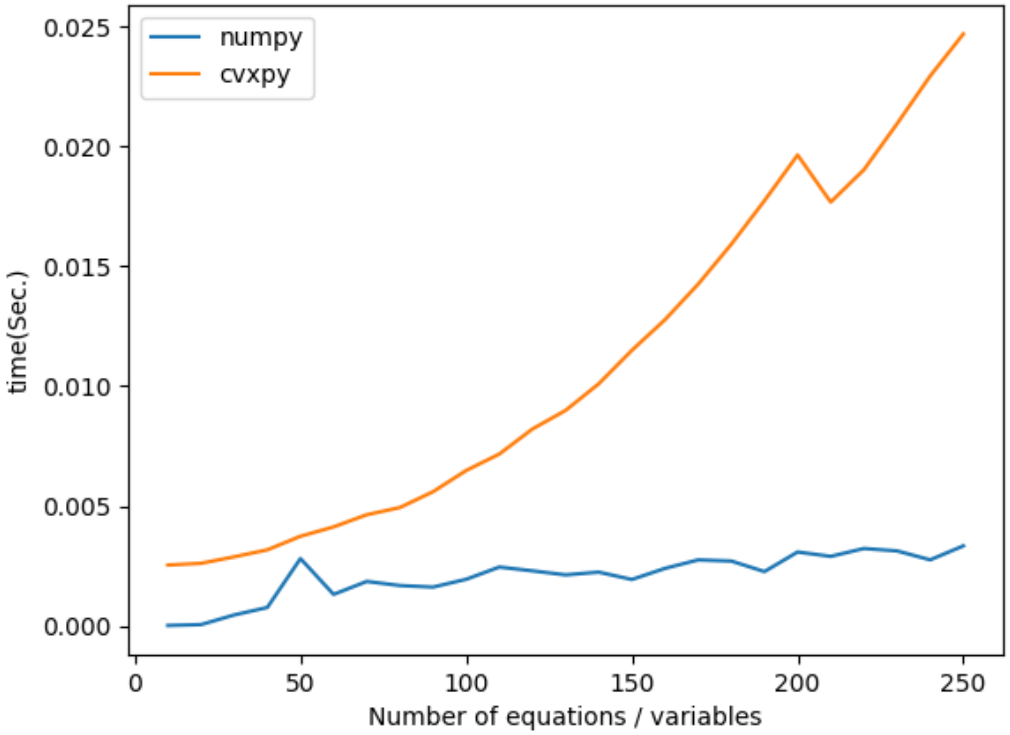
תכנות אלגוריתמים מחקריים - מטלה 7

ניר סון - 323918599

שאלה 1:

קישור לגיטהב (שאלות 1 ו2 באותו הקובץ):
https://github.com/nirson123/researchAlgoCourse/blob/master/task%207/main.py

גרף התוצאות:



ניתן לראות ש numpy מהירה יותר בהרבה - ככל שכמות המשוואות והמשתנים גדלה, הפער גדל. אני מניח שהקפיצות בגרף (ב50 וב 200) זו פשוט שונות (תהליכים אחרים במחשב שדרשו חלק מכוח החישוב, או הגרלה "טובה" או "רעה" של מקדמים במשוואות), ואין לייחס להן משמעות מיוחדת.

שאלה 2:

קישור לגיטהב (שאלות 1 ו2 באותו הקובץ):
https://github.com/nirson123/researchAlgoCourse/blob/master/task%207/main.py

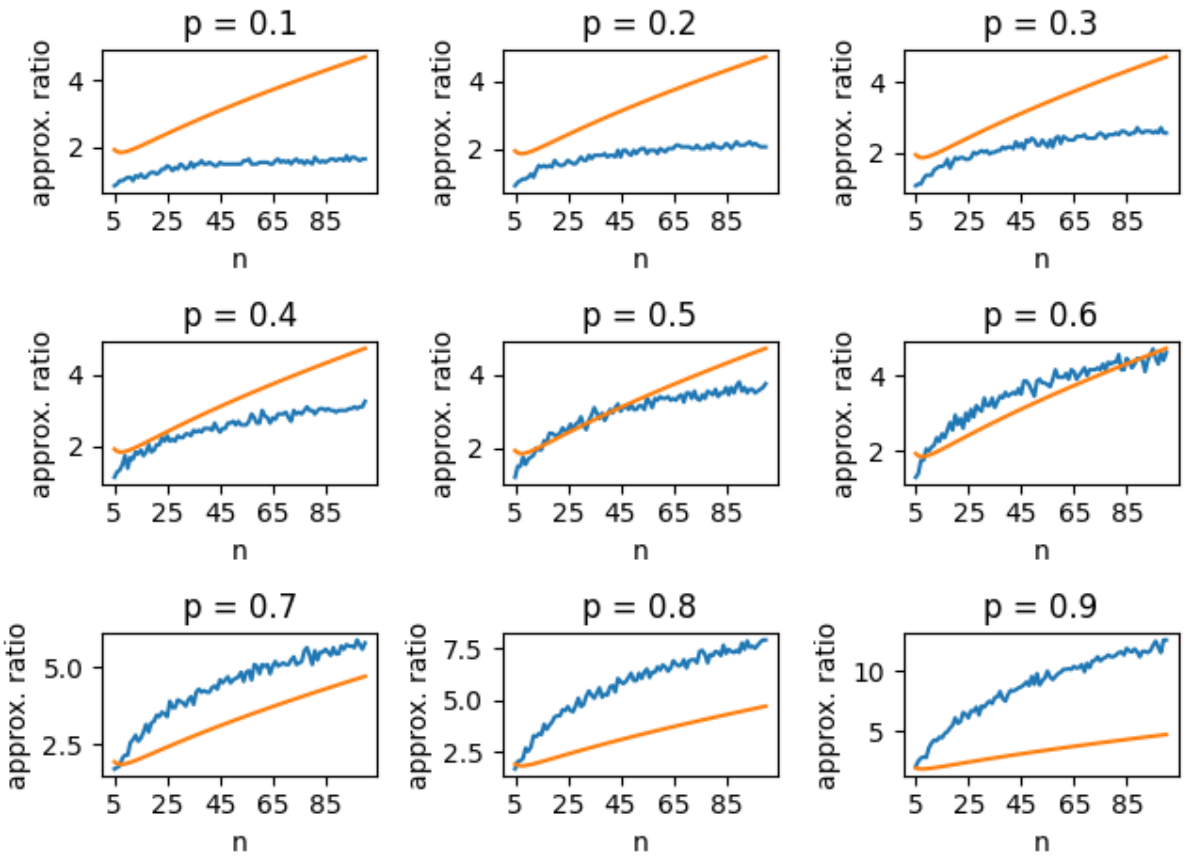
בחרתי את בעיית מציאת קליקה מקסימלית בגרף. יחס הקירוב התיאורטי של האלגוריתם הממומש ב networkx הוא $\frac{n}{log^2(n)}$, כאשר n זו כמות הקודקודים בגרף.

בדקתי תשעה ערכים שונים של p:

$0.1, 0.2, ..., 0.8, 0.9$

כל אחד מהם בדקתי עם כל כמות קודקודים בין 5 ל 100, כאשר כל זוג (p,n) נבדק עשר פעמים, כדי להפחית את השונות (התוצאות הן הממוצע של 10 הפעמים).

גרף התוצאות:



הגרפים הכחולים הם התוצאות שיצאו לי. הגרפים הכתומים הם יחס הקירוב התיאורטי.
ניתן לראות שעבור ערכי p גדולים (גרפים צפופים), לוקח זמן עד שהאלגוריתם מגיע באמת ליחס הקירוב שלו, ובגרפים קטנים הוא לא מתפקד טוב כל כך.

<u>שאלה 3:</u>

קישור:

https://www.codingame.com/training/expert/unfolding-paper/solution?id=27616262

הפתרון שלי משתמש באלגוריתם למציאת רכיבי קשירות בגרף, שמימשתי בעצמי, כיוון שהאתר לא תומך בספרייה networkx (או לכל הפחות לא הצלחתי לייבא אותה באתר).

צילום מסך של הפתרון:

```python
import numpy as np

# input
n = int(input())
w, h = [int(i) for i in input().split()]
table = []
for i in range(h):
    table.append([c for c in input()])
table = np.array(table)

# find islands
visited = np.zeros(table.shape)
m, l, r, u, d, lr, ud, lu, ld, ru, rd, lur, urd, rdl, dlu, lurd = (0 for _ in range(16))

for i in range(table.shape[0]):
    for j in range(table.shape[1]):

        if visited[i ,j] == 0 and table[i,j] == '#':

            left, right, up, down = False, False, False, False  # the sides the island touch

            # DFS
            nodes = [(i ,j)]
            while nodes:
                k, p = nodes.pop()
                visited[k, p] = 1

                # mark sides
                if k == 0:
                    up = True
                if p == 0:
                  left = True
                if k == table.shape[0] - 1:
                    down = True
                if p == table.shape[1] - 1:
                    right = True
```

```python

                # visit neighbors
                if k != 0 and visited[k - 1, p] == 0 and table[k - 1, p] == '#':
                    nodes.append((k - 1, p))
                if k != table.shape[0] - 1 and visited[k + 1, p] == 0 and table[k + 1, p] == '#':
                    nodes.append((k + 1, p))
                if p != 0 and visited[k, p - 1] == 0 and table[k, p - 1] == '#':
                    nodes.append((k, p - 1))
                if p != table.shape[1] - 1 and visited[k, p + 1] == 0 and table[k, p + 1] == '#':
                    nodes.append((k, p + 1))

            # classife into class
            if sum((left, right, up, down)) == 0:       # no side (middle)
                m += 1
            elif sum((left, right, up, down)) == 4:     # all sides
                lurd += 1
            elif sum((left, right, up, down)) == 1:     # one side
                if left:
                    l += 1
                elif right:
                    r += 1
                elif up:
                    u += 1
                else:
                    d += 1
            elif sum((left, right, up, down)) == 2:     # two sides
                if left and right:
                    lr += 1
                elif left and up:
                    lu += 1
                elif left and down:
                    ld += 1
                elif right and up:
                    ru += 1
                elif right and down:
```

```python
                rd += 1
            else:
                ud += 1
        else:                                       # three sides
            if left and up and right:
                lur += 1
            elif up and right and down:
                urd += 1
            elif right and down and left:
                rdl += 1
            else:
                dlu += 1

# calculate results
# instead of accualy calculating the un-folded talbe (a lot of memory) just calaulate nuber of islands of each type after one un-fold
for _ in range(n):
    m1 = 4*m + 2*l + 2*u + lu

    l1 = 2*r + ru
    r1 = 2*r + ru
    d1 = 2*d + ld
    u1 = 2*d + ld

    lurd1 = lurd

    lur1 = rdl
    rdl1 = rdl
    dlu1 = urd
    urd1 = urd

    lr1 = lur + 2*lr
    ud1 = dlu + 2*ud
    ld1 = rd
    lu1 = rd
    rd1 = rd
```

```python
    ru1 = rd

    # updete the counts
    m, l, r, u, d, lr, ud, lu, ld, ru, rd, lur, urd, rdl, dlu, lurd = \
    m1, l1, r1, u1, d1, lr1, ud1, lu1, ld1, ru1, rd1, lur1, urd1, rdl1, dlu1, lurd1

# calculate the total number of islands
result = sum([m, l, r, u, d, lr, ud, lu, ld, ru, rd, lur, urd, rdl, dlu, lurd])
print(result)
```