Grid path problem

mame mriezku m x n
po mriezke sa pohybuju auta - iba do prava alebo dole
auto zacina v bode [0, 0]
kolkymi sposobmi sa vie dostat na miesto [m, n]

1. podproblem
    a. kolkymi sposobmi sa vieme dostat na [i-1, j] a [i, j-1]
2. vyriesime podproblem pomocou inych podproblemov
    a. OPT(i, j) = OPT(i-1, j) + OPT(i, j-1)
3. bazove podproblemy
    a. OPT(0, j) = OPT(i, 0) = 1
4. poradie
    a. rekurzia + memoizacia
    b. po riadkoch/stlpcoch/diagonalach

kod - exponencialna verzia:

```
n, m = 10, 10

def get_num_paths(i, j):
    if i == 0 or j == 0:
        return 1

    return get_num_paths(i-1, j) + get_num_paths(i, j-1)

print(get_num_paths(n, m))
```

kod - memoizacia:

```
import numpy as np

n, m = 30, 30
memo = np.zeros((n+1, m+1), dtype=np.int64)

def get_num_paths(i, j):
    if i == 0 or j == 0:
        return 1

    if memo[i, j] == 0:
        memo[i, j] = get_num_paths(i-1, j) + get_num_paths(i, j-1)

    return memo[i, j]

print(get_num_paths(n, m))
```

Wine selling problem

mame n flasiek vin
s pociatocnymi cenami $c = [c_1, c_2, \ldots, c_n]$
mozme predavat flase zo zaciatku alebo konca
cena kazdeho vina kazdy rok rastie - v roku y predame vino v_i za cenu v_i*y
maximalizujte profit

specific example
n = 5
c = [2, 4, 6, 2, 5]

1. urcime podproblem
    a. o jedna kratsi rad
2. vyriesime podproblem za pomoci inych podproblemov
    a. `OPT(i, j) = max{c[i] * y + OPT(i + 1, j), c[j] * y + OPT(i, j-1)}`
3. bazove podproblemy
    a. `OPT(i, j) = c[i] * n`
4. vyberieme poradie
    a. od uhlopriecky

vysledna tabulka:

10	28	52	56	64
0	20	46	52	62
0	0	30	38	53
0	0	0	10	33
0	0	0	0	25

m[4, 4] = 25
m[3, 3] = 10
m[3, 4] = max(8 + 25.0, 20 + 10.0) = 33.0
m[2, 2] = 30
m[2, 3] = max(24 + 10.0, 8 + 30.0) = 38.0
m[2, 4] = max(18 + 33.0, 15 + 38.0) = 53.0
m[1, 1] = 20
m[1, 2] = max(16 + 30.0, 24 + 20.0) = 46.0
m[1, 3] = max(12 + 38.0, 6 + 46.0) = 52.0
m[1, 4] = max(8 + 53.0, 10 + 52.0) = 62.0
m[0, 0] = 10
m[0, 1] = max(8 + 20.0, 16 + 10.0) = 28.0
m[0, 2] = max(6 + 46.0, 18 + 28.0) = 52.0
m[0, 3] = max(4 + 52.0, 4 + 52.0) = 56.0
m[0, 4] = max(2 + 62.0, 5 + 56.0) = 64.0

kod:

```
import numpy as np

c = [2, 4, 6, 2, 5]
n = len(c)
m = np.zeros((n, n))

for i in range(n-1, 0-1, -1):
    for j in range(i, n):
        if i == j:
            m[i, j] = c[i] * n
        else:
            y = n - (j - i)
            m[i, j] = max(c[i]*y + m[i+1, j], c[j]*y + m[i, j-1])
```

casova zlozitost: O(n^2)

Maximum size square sub-matrix with all 1s

mame binarnu (0/1) maticu rozmerov m*n
mame najst velkost najvacsej matice plnej 1

1. urcime podproblem
    a. stvorec o jedna mensi hore, nalavo a diagonalne
2. vyriesime podproblem za pomoci inych podproblemov
    a. `if OPT(i, j) = 1`
        i. `OPT(i, j) = min(OPT(i-1, j), OPT(i, j-1), OPT(i-1, j-1)) + 1`
    b. `else 0`
3. bazove podproblemy
    a. lavy a horny okraj, spravat sa ako keby tam bola 0
4. vyberieme poradie
    a. riadky, stlpce, diagonala - vsetky mozne

casova zlozitost: O(n*m)