Minule cvika - fractional knapsack problem (greedy algorithm)
Prednaska - 0/1 knapsack problem (dynamic programming)

Prednaska 3 - activity selection problem

1. urcime podproblem
2. vyriesime podproblem za pomoci inych podproblemov
3. bazove podproblemy
4. vyberieme poradie

Change-making problem (general)

k druhov minci s hodnotami h = {h_1, h_2, … , h_k}, kazdy druh neobmedzeny pocet
chceme vydat cenu n co najmensim poctom minci m

specific example:
h = {1, 3, 4}
n = 18

1. urcime podproblem
    a. mensie n-ka
2. vyriesime podproblem za pomoci inych podproblemov
    a. OPT(i) = min(OPT(i-1) + 1, OPT(i-3) + 1, OPT(i-4) + 1)
3. bazove podproblemy
    a. OPT(0) - OPT(4) vyriesit rucne
4. vyberieme poradie
    a. i od 5 po n, reportujeme OPT(n)

i	0	1	2	3	4	5	6	7	8	9
m	0	1	2	1	1	2	2	2	2	3
i	10	11	12	13	14	15	16	17	18	19
m	3	3	3	4	4	4	4	5	5	5

Casova zlozitost: O(n) - trivialne

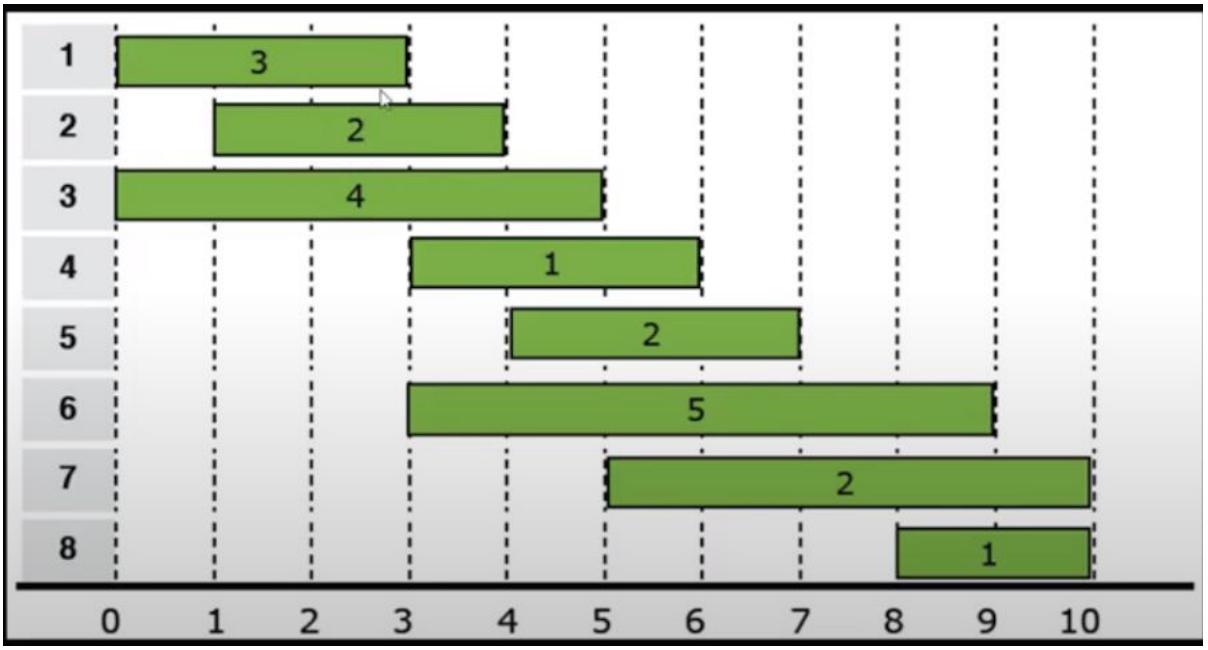
Weighted activity selection problem

#activities n
activity = {start, finish, weight}

task: select non-overlapping activities such that the sum of their weight is maximal

specific example:

- `n = 8`
- `s = [0, 1, 0, 3, 4, 3,  5,  8]`
- `f = [3, 4, 5, 6, 7, 9, 10, 10]`
- `w = [3, 2, 4, 1, 2, 5,  2,  1]`



1. urcime podproblem
    a. neprekryvajuce sa aktivity $p(i) = max(j, f_j <= s_i)$
2. vyriesime podproblem za pomoci inych podproblemov
    a. $OPT(i) = max(w_i + OPT(p(i)), OPT(i-1))$
3. bazove podproblemy OPT(0) = 0, p(0) = 0
4. vyberieme poradie naplnania (->)

sledujme i, w_i, $p(i)$, $OPT(p(i))$, $OPT(i)$

i	1	2	3	4	5	6	7	8
w_i	3	2	4	1	2	5	2	1
p(i)	0	0	0	1	2	1	3	5
OPT(p(i))	0	0	0	3	3	3	4	5
OPT(i)	3	3	4	4	5	8	8	8

casova zlozitost:

- sort by finish - O(nlog(n))
- find p(i) for i=1..n - O(nlog(n)) - binary search
- find OPT(N) - O(n)
- backtrack - O(n)

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)