TAP C5: Metoda Backtracking

1. Să se împartă un text fără spații în cuvinte dintr-un dicționar d, în toate modurile posibile.

```
d = {"a", "ab", "ba"}   #set containing the dictionary
w = "ababa"                          #input string
s = [None] * len(w)

def split(k, word_count):
    if (k == len(w)):
        print(s[0:word_count])
    for i in range(k+1, len(w) + 1):
        s[word_count] = w[k:i]
        split(i, word_count + 1 )

def split_d(k, word_count):
    if (k == len(w)):
        print(s[0:word_count])
    for i in range(k+1, len(w) + 1):
        if w[k:i] in d:
            s[word_count] = w[k:i]
            split_d(i, word_count + 1 )

split(0,0)
split_d(0,0)

#output
['a', 'b', 'a', 'b', 'a']
['a', 'b', 'a', 'ba']
['a', 'b', 'ab', 'a']
['a', 'b', 'aba']
['a', 'ba', 'b', 'a']
['a', 'ba', 'ba']
['a', 'bab', 'a']
['a', 'baba']
['ab', 'a', 'b', 'a']
['ab', 'a', 'ba']
['ab', 'ab', 'a']
['ab', 'aba']
['aba', 'b', 'a']
['aba', 'ba']
['abab', 'a']
['ababa']
['a', 'ba', 'ba']
['ab', 'a', 'ba']
['ab', 'ab', 'a']
```

2. Să se genereze toate subșirurile strict crescătoare ale unui șir de numere întregi:

```
v = [4, 2, 7, 9, 12, 10, 11, 15]
s = [None] * len(v)

def print_sol(count):
    print(s[0:count])

def lis(k, count):
    if (k == len(v)):
        print_sol(count)
    for i in range(k, len(v)):
        if v[i] > s[count - 1]:
            s[count] = v[i]
            lis( i+1 , count + 1)


for i in range(len(v)):
    s[0] = v[i]
    lis(i+1, 1)

#output
[4, 7, 9, 12, 15]
[4, 7, 9, 10, 11, 15]
[4, 7, 9, 10, 15]
[4, 7, 9, 11, 15]
[4, 7, 9, 15]
……
[4, 15]
[2, 7, 9, 12, 15]
[2, 7, 9, 10, 11, 15]
….
[2, 7, 10, 11, 15]
[2, 7, 10, 15]
….
[2, 12, 15]
[2, 10, 11, 15]
[2, 10, 15]
[2, 11, 15]
[2, 15]
[7, 9, 12, 15]
…..
[9, 10, 11, 15]
[9, 15]
[12, 15]
[10, 11, 15]
[10, 15]
[11, 15]
[15]
```

TAP C6: Programare dinamica

1. Să se găsească un subșir strict crescător de lungime maximă al unui șir *v* de *n* numere întregi.

   Vom calcula pentru fiecare indice *i = 1 … n* care este lungimea maximă a unui subșir care începe pe poziția *i* și vom memora această valoare în L[i].
   succ[i] va reține poziția celui de-al doilea element din cel mai lung subșir crescător care începe cu *i*.

   Putem număra câte subșiruri strict crescătoare de lungime maximă are șirul v?

```
V = [6, 3, 5, 10 , 12, 2, 9, 15, 14, 7, 4, 8, 13]
L = [4, 5, 4, 3,    2, 4, 2,  1,  1, 3, 3, 2,  1]

un șir de lungime 4 care incepe pe pozitia 5 este:
2, 7, 8, 13 ( următorul element este 7, pornind de la 7 se pot
obține subșiruri strict crescătoare de lungime maxim 3)
```

```
v = [6, 3, 5, 10 , 12, 2, 9, 15, 14, 7, 4, 8, 13]

n = len(v)
L = [1] * len(v)
succ = [n] * len(v)

L[n-1] = 1
poz_max = n - 1
for i in range(n-2, -1, -1):
    for j in range(i + 1, n):
        if v[i] < v[j] and l[j] + 1 > l[i]:
            L[i] = L[j] + 1
            succ[i] = j
    if L[i] > L[poz_max]:
        poz_max = i

k = poz_max
while k < n:
    print(v[k])
    k = succ[k]
```

2. Se dă un șir de caractere *s*, fară spații și un dicționar dict. Să se împartă șirul *s* (să se adauge spații) în număr minim de cuvinte din *dict*.

   Vom memora în L[i] numărul minim de cuvinte în care se împarte s[*1* … *i*].

```
s = " xyzzzzxy"
dict  = set({"xyz", "zzz", "zzzz","x" , "y", "xy"})
prev, L = [],[]

def spaces(s, dict):
    n = len(s)
    global prev, l
    L = [-1] * (n)
    prev = [-1] * (n)
    L[0] = 0
    for i in range(1,n):
        for j in range(i, 0, -1):
            if s[j:i+1] in dict:   #dict.__contains__( s[j:i+1])
                if((L[j-1]+1 < L[i] or L[i]==-1) and L[j-1]!=-1):
                    L[i] = L[j - 1] + 1
                    prev[i] = j - 1
    return L[n - 1]

def print_substr(i):
    if prev[i] > 0:
        print_substr(prev[i])
    print( s[prev[i] + 1 : i + 1] )


print(spaces(s, dict))
print_substr(len(s) - 1)
```