TAP C6: Programare dinamica

1. Se dau două șiruri de caractere *a, b*. Să se calculeze d(a,b) = distanța de editare dintre a și b (distanța Levenshtein). https://en.wikipedia.org/wiki/Levenshtein_distance

   Vom memora în M[I, j] = d(a[*1* … *i*], b[*1* … *j*]).

```
a = " kitten"
b = " sitting"
n, m = len(a), len(b)

M = [ [0 for j in range(m)] for i in range(n)]
M[0] = [ j for j in range(m) ]

for i in range(1, n):     #for(int i = 1; i < n; i++)
    M[i][0] = i
    for j in range(1, m): #for(int j = 1; i < m; j++)
        if a[i] == b[j]:
            M[i][j] = M[i-1][j-1]
        else:
            M[i][j] = 1 + min(M[i-1][j-1], M[i-1][j], M[i][j-1])

edit = [ None ] * M[n-1][m-1] #array with M[n][m] edits
i, j, k = n - 1, m - 1, M[n-1][m-1] - 1
while k >= 0 and i > 0 and j > 0:
    if a[i] == b[j]:
        i, j = i - 1, j - 1
    else:
        if M[i][j] == 1 + M[i - 1][j - 1]:
             edit[ k ] = "replace " + a[i] + " --> " + b[j]
             i, j, k = i - 1, j - 1, k - 1

        elif M[i][j] == 1 + M[i][j - 1]:
            edit[k] = "add " + b[j]
            j, k = j - 1, k - 1
        elif M[i][j] == 1 + M[i-1][j]:
            edit[k] = "delete " + a[i]
            i, k = i - 1, k - 1

while k >= 0 and i > 0:
    if M[i][j] == 1 + M[i-1][j]:
            edit[k] = "delete " + a[i]
            i, k = i - 1, k - 1
```

```
while k >= 0 and j > 0:
    if M[i][j] == 1 + M[i][j - 1]:
        edit[k] = "add " + b[j]
        j, k = j - 1, k - 1


print(edit)
```

2. Sa se scrie relatiile de recurenta care permit rezolvarea urmatoarelor probleme:

a. Sa se gaseasca cel mai lung subsir comun al doua siruri de caractere a, b

b. Se dau doua siruri de caractere t si s. Sa se numere de cate ori apare t ca subsir al sirului s.

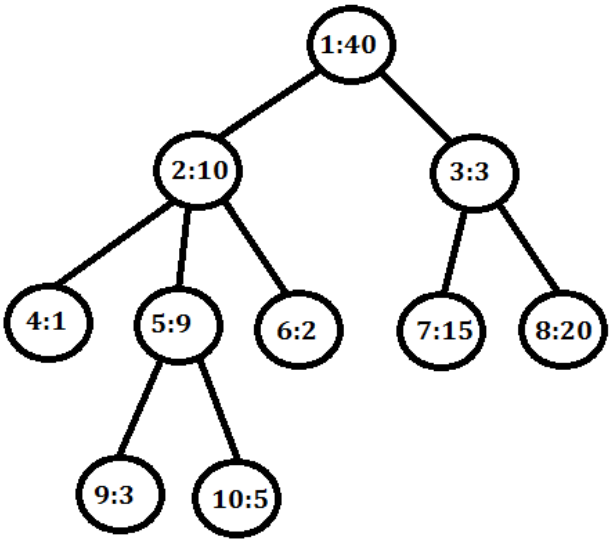
c. Se dau doua siruri s si p. Sirul s contine litere mici ale alfabetului, sirul p contine literele mici ale alfabetului si simbolurile * si _ . Sa se testeze daca p este un pattern care recunoaste s. Simbolul * inlocuieste orice sir (inclusiv sirul vid), simbolul _ inlocuieste o singura litera.

d. Se dau trei siruri a, b, c. Sa se testeze daca c este rezultatul interclasarii sirurilor a, b.

e. Se dau doua numere naturale n si k. Sa se afiseze numarul de permutari de n elemente care au exact k inversiuni.

3. Se consideră un arbore în care muchiile sunt etichetate cu ponderi. Să se găsească o mulțime independentă de noduri cu suma ponderilor maximă.

   Pentru graful din imagine o submultime independentă de noduri cu suma ponderilor maximă este formată din {1, 4, 5, 6, 7, 8} cu suma 87.



Vom memora pentru fiecare nod v:
bestw[v, 0] = suma maxima a ponderilor unei submultime independente de noduri din subarborele care are rădăcina v și care **nu include** nodul v

bestw[v, 1] = suma maxima a ponderilor unei submultime independente de noduri din subarborele care are rădăcina v și care **include** nodul v

```
nrnodes = int(input())
bestw = [None] * nrnodes
visited = [False] * nrnodes
neighbours = [None] * nrnodes
weight = [0] * nrnodes


for i in range(nrnodes):
    weight[i] = int(input())
    neighbours[i] = []
    bestw[i] = [0,0]

for i in range(nrnodes - 1):
    edge = input().split()
    i = int(edge[0]) - 1
    j = int(edge[1]) - 1
    neighbours[i].append(j)
    neighbours[j].append(i)
```

```
def df(v):
    visited[v] = True
    bestw[v][1] = weight[v]
    bestw[v][0] = 0
    for i in neighbours[v]:
        if visited[i] == False:
            df(i)
            bestw[v][1] += bestw[i][0]
            bestw[v][0] += max(bestw[i][1], bestw[i][0])

df(0)
print(  max(bestw[0][0],bestw[0][1]) )
```