Školsko natjecanje iz informatike

Srednja škola

Druga podskupina (3. i 4. razred)

25. siječnja 2019.

RJEŠENJA ZADATAKA

Ministarstvo znanosti i obrazovanja

Agencija za odgoj i obrazovanje

HRVATSKA ZAJEDNICA TEHNIČKE KULTURE

HRVATSKI SAVEZ INFORMATIČARA

Riječ-kandidata u zadanoj riječi tražimo tako da na sve načine isprobamo krenuti od nekog **i**-tog slova, odabrati neki razmak i uzimati slovo po slovo s odabranim razmacima, uspoređujući ih sa slovima kandidata.

```
#!/usr/bin/python3
a = input()
n = len(a)
k = int(input())
for i in range(k):
    kandidat = input()
    nasao = False
    for i in range(n):
        for razmak in range(1, n):
            ok = True
            for k, slovo in enumerate(kandidat):
                if i + k * razmak >= n or a[i + k * razmak] != slovo:
                    ok = False
            nasao = nasao or ok
    if nasao:
        print('DA')
    else:
        print('NE')
```

I gornji i donji dio ekrana pretvorit ćemo u niz visina odgovarajućih stupaca, pretvarajući tako matricu u dva niza brojeva. Potom računamo visinu špilje nakon urušavanja – to će biti najveći zbroj visina nekog stupca gornjeg dijela i pripadnog stupca donjeg dijela špilje. Znajući tu visinu, lako je izračunati gdje će se točno naći stupci gornjeg dijela špilje nakon urušavanja. „Crtanje" konačne špilje možemo činiti tako da u matricu koja je na početku ispunjena točkama dodajemo stupce donjeg i stupce gornjeg dijela špilje na temelju izračunatih nizova i visine.

```
#!/usr/bin/python3
r, s = map(int, input().split())
a = []
for i in range(r):
    a.append(input())
gore, dolje = [0 for j in range(s)], [0 for j in range(s)]
for j in range(s):
    punim = gore
    for i in range(r):
        if a[i][j] == '#':
            punim[j] += 1
        else:
            punim = dolje
visina = max(gore[j] + dolje[j] for j in range(s))
prazno = r - visina
b = [['.' for j in range(s)] for i in range(r)]
for j in range(s):
    for i in range(gore[j]):
        b[prazno + i][j] = '#'
    for i in range(dolje[j]):
        b[r - 1 - i][j] = '#'
for i in range(r):
    print(''.join(b[i]))
```

Možemo iskušavati razne nizove pretakanja koje nas različitim putevima vode do novih količina otopine u epruvetama. Dobivene količine otopine u epruvetama nakon nekog pretakanja možemo nazvati stanjem. Ovdje se prirodno nameće graf čiji su vrhovi stanja, a bridovi pretakanja kojima iz jednog stanja prelazimo u drugo. Ovaj graf nije potrebno eksplicitno konstruirati, tj. pohraniti sve njegove bridove. Dovoljno je pustiti pretraživanje u širinu (BFS) iz početnog stanja, pri čemu za svako stanje iz kojeg se širimo stavljamo u red susjedna stanja isprobavajući sva moguća pretakanja. Duljinu dobivenog puta za svako stanje pamtimo u rječniku (mapi) koji će služiti i za provjeru jesmo li neko stanje već posjetili. Za BFS koristimo strukturu reda koju u Pythonu podržava npr. klasa deque.

```
#!/usr/bin/python3
from collections import deque
n = int(input())
ukupno = list(map(int, input().split()))
pocetno = list(map(int, input().split()))
udaljenost = dict()
q = deque()

def obradi(stanje, br_poteza):
    stanje = tuple(stanje)  # jer lista ne može biti ključ rječnika
    if stanje in udaljenost:
        return
    if n in stanje:
        print(br_poteza)
        exit(0)
    udaljenost[stanje] = br_poteza
    q.append(stanje)

obradi(pocetno, 0)
while len(q):
    stanje = q.popleft()
    br_poteza = udaljenost[stanje] + 1
    for i in range(3):
        novo_stanje = list(stanje)
        novo_stanje[i] = 0
        obradi(novo_stanje, br_poteza)
    for i in range(3):
        for j in range(3):
```

```
        if i == j: continue
        u, v = stanje[i], stanje[j]
        novo_stanje = list(stanje)
        novo_stanje[i] = min(ukupno[i], u + v)
        novo_stanje[j] = max(0, v - (ukupno[i] - u))
        obradi(novo_stanje, br_poteza)
```