Školsko natjecanje iz informatike

Srednja škola

Druga podskupina (3. i 4. razred)

18. siječnja 2018.

RJEŠENJA ZADATAKA

Ministarstvo znanosti i obrazovanja

Agencija za odgoj i obrazovanje

HRVATSKA ZAJEDNICA TEHNIČKE KULTURE

HRVATSKI SAVEZ INFORMATIČARA

Iako su igrači poredani u krug, mi ćemo njihova imena držati u nizu stringova, imajući na umu da su prvi i posljednji igrač u nizu zapravo susjedni. Zadatak se svodi na implementaciju sljedećih operacija:

- brisanje elementa iz niza,
- pronalazak pozicije zadanog elementa u nizu,
- traženje udaljenosti između dviju pozicija X i Y u „kružnom“ nizu, što rješavamo uzimajući manji od kružnih lukova čije su duljine $|X - Y|$ i $N - |X - Y|$.

```
#!/usr/bin/python3

n = int(input())
a = []
for i in range(n):
    a.append(input())
k = int(input())
for it in range(k):
    upit = input().split()
    if len(upit) == 2:
        a.remove(upit[1])
        continue
    i = a.index(upit[1])
    j = a.index(upit[2])
    d = abs(i - j)
    print(min(d, len(a) - d))
```

Zadatak se sastoji od dvaju manjih zadataka:

- **Pronalazak dimenzija tablice** vršimo ispitivanjem svih mogućnosti, tj. svih mogućih rastava N = R x S pri čemu je N duljina zadane riječi. Za svaki mogući broj redaka R (koje biramo for petljom) provjeravamo je li N djeljiv sa R i ako jest, računamo broj stupaca S i razliku S – R te u pomoćnoj varijabli pamtimo dimenzije (R, S) koje imaju najmanju pronađenu razliku.
- **Ispis riječi u tablicu** vršimo prolaskom po znakovima dane riječi, određujući polje u koje treba upisati trenutačni znak. U tu svrhu u pomoćnim varijablama pamtimo trenutačno polje i smjer (0, 1, 2, 3) u kojem se krećemo po tablici. Ako se u tom smjeru možemo pomaknuti za još jedno polje (bez iskakanja iz tablice ili ulaska u već posjećeno polje), to i činimo, a inače mijenjamo smjer kretanja. U novo polje upisujemo trenutačni znak.

```
#!/usr/bin/python3
def rs(n):
    best = (1, n)
    for r in range(2, n):
        if n % r == 0:
            s = n // r
            if s >= r and best[1] - best[0] > s - r:
                best = (r, s)
    return best

rijec = input()
r, s = rs(len(rijec))
a = [['.' for j in range(s)] for i in range(r)]
i, j, smjer = 0, 0, 0
for znak in rijec:
    a[i][j] = znak
    for it in range(4):
        x = i + [0, 1, 0, -1][smjer]
        y = j + [1, 0, -1, 0][smjer]
        if min(x, y) < 0 or x >= r or y >= s or a[x][y] != '.':
            smjer = (smjer + 1) % 4
            continue
        i, j = x, y
        break
for i in range(r):
    print(''.join(a[i]))
```

Ključ rješenja ovog zadatka krije se u ograničenju $N \leq 8$ koje nam sugerira da ne moramo smisliti naročito brzo rješenje u ovisnosti o N. Štoviše, budući da broj mogućih N-permutacija nije veći od 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1 = 40 320, možemo iskušavati razne nizove transformacija koje nas različitim putevima vode do novih permutacija. "Putovanje po permutacijama koristeći transformacije" zvuči kao "putovanje po vrhovima koristeći bridove" i sada se prirodno nameće graf čiji su vrhovi N-permutacije, a bridovi transformacije kojima iz jedne permutacije dolazimo u drugu. Ono što u takvom grafu tražimo jest najkraći put od permutacije A do permutacije B.

Ovaj graf nije potrebno eksplicitno konstruirati, tj. pohraniti sve njegove bridove, što ne bi bilo ni lako učiniti (morali bismo za početak nekako numerirati permutacije od prve do posljednje). Dovoljno je pustiti pretraživanje u širinu (BFS) iz permutacije A, posjećene permutacije spremati npr. u set da ih ne bismo posjetili više puta, a duljine dobivenih putova pamtiti zajedno s pripadnim vrhovima u redu (Queue) koji će, dakle, sadržavati parove oblika *(permutacija, duljina najkraćeg puta do te permutacije)*.

```
#!/usr/bin/python3
from queue import *

n = int(input())
a = tuple(map(int, input().split()))
b = tuple(map(int, input().split()))
q = Queue()
s = set()
q.put((a, 0))
s.add(a)
while not q.empty():
    a, brojac = q.get()
    if a == b:
        print(brojac)
        break
    for k in range(2, n + 2):  # sve moguce transformacije
        c = [(x < k) * (k - x) + (x >= k) * x for x in a]
        c = tuple(c)
        if c not in s:
            s.add(c)
            q.put((c, brojac + 1))
```