Županijsko natjecanje iz informatike

Srednja škola

Prva podskupina (1. i 2. razred)

9. veljače 2018.

RJEŠENJA ZADATAKA

Napomena: kodovi za većinu opisanih algoritama dani su u Pythonu radi jednostavnosti i lakše čitljivosti. Zbog prirode jezika, rješenja u Pythonu sporija su od ekvivalentnih rješenja u C++u pa natjecateljima preporučujemo da u slučaju potrebe za bržim rješenjem koriste C++.

Ako su odabrani dani s rednim brojevima A, B i C, na kraju ćemo imati

10 / cijena[A] * cijena[B] / cijena[C]

bitcoina.

Isprobavajući sve moguće trojke A, B, C (koristeći tri for petlje jednu unutar druge) dobivamo složenost $O(N^3)$, što je presporo za veće test podatke.

Brže rješenje u $O(N^2)$ možemo dobiti isprobavanjem svih odabira A i B (dvije for petlje) ako znamo brzo naći element s najmanjom vrijednošću cijena[C] desno od odabranog B, jer se očito isplati podijeliti sa što manjim brojem. To možemo provesti ako, prije isprobavanja odabira, unaprijed izračunamo tražene minimume za svaki B. Preciznije, prolazeći nizom s desna na lijevo pamtimo dosadašnji minimum i spremamo ga u pomoćni niz na trenutačnu poziciju B.

Konačno, uočimo da isto možemo učiniti i za elemente lijevo od B. Naime, za dan B koji odabiremo na sve moguće načine, isplati se uzeti najmanji element cijena[A] lijevo od B, kao i najmanji element cijena[C] desno od B. Ako smo te minimume lijevo i desno za svaki mogući B unaprijed izračunali dvama prolascima po nizu, dobivamo rješenje u složenosti O(N).

```
#!/usr/bin/python3
n = int(input())
a = [float(input()) for i in range(n)]

min_lijevo = [a[0]] + [0 for i in range(n - 1)]
for i in range(1, n):
    min_lijevo[i] = min(min_lijevo[i - 1], a[i])

min_desno = [0 for i in range(n - 1)] + [a[n - 1]]
for i in range(n - 2, -1, -1):
    min_desno[i] = min(min_desno[i + 1], a[i])

print(10 * max(a[i] / min_lijevo[i - 1] / min_desno[i + 1]
    for i in range(1, n - 1)))
```

Ovo je zadatak u kojemu je prilično jasno što treba činiti, ali treba razmisliti kako to jednostavno i što elegantnije implementirati. Očekivano rješenje isprobava sve stupce, za svaki od njih pronalazi najniže slobodno polje, i potom se iz tog polja širi u svih osam smjerova, brojeći uzastopne žetone boje 1.

Za elegantno širenje u osam smjerova koristimo dva pomoćna niza od 8 elemenata koji sadrže pomake po retku i stupcu (0, -1 ili 1) za pojedini smjer.

```
#!/usr/bin/python3
m = []
for i in range(6):
    m.append(input())
m = [[c for c in m[i]] for i in range(5, -1, -1)]
sol = False
for s in range(7):
    r = -1
    for i in range(6):
        if m[i][s] == '0':
            r = i
            m[r][s] = '1'
            break
    if r == -1:
        continue
    zetona = [0 for i in range(8)]
    for smjer in range(8):
        x, y = r, s
        while 0 <= x < 6 and 0 <= y < 7 and m[x][y] == '1':
            zetona[smjer] += 1
            x += [0, 1, 1, 1, 0, -1, -1, -1][smjer]
            y += [-1, -1, 0, 1, 1, 1, 0, -1][smjer]
    if max(zetona[0] + zetona[4], zetona[1] + zetona[5],
            zetona[2] + zetona[6], zetona[3] + zetona[7]) >= 5:
        sol = True
        print(s + 1)
    m[r][s] = '.'
if not sol:
    print(-1)
```

Najprije valja izdvojiti sve vremenske intervale u kojima je prskalica mogla biti ispravna, tj. naći komplement intervala zadanih u ulaznim podatcima. To činimo tako da zadane intervale kvara prskalice interpretiramo kao događaje koji mogu biti „kvar počinje“ ili „kvar završava“. Te događaje potom sortiramo po vremenu i prolazimo po njima, održavajući broj trenutačno nezavršenih „kvarova“. Na taj način možemo detektirati vremenske intervale u kojima nema kvara.

Za svaki od dobivenih „ispravnih“ intervala znamo početnu i završnu sekundu i pitamo se koja su djeca i koliko puta bila poprskana u tom vremenskom intervalu. Dijeljenjem trajanja intervala s brojem N možemo saznati broj punih krugova vrtuljka u kojima su sva djeca bila poprskana čokoladom. Taj broj održavamo u pomoćnoj varijabli i na kraju svaki element rješenja (za svako dijete) treba uvećati za taj broj.

U slučaju da se interval prskanja ne sastoji samo od punih krugova, formulama valja pronaći prvo i zadnje poprskano dijete u tom intervalu. Za svu djecu koja se u krugu nalaze između to dvoje treba povećati rješenje za 1, jer ona su bila poprskana jedan put više. Da ne bismo premašili vremensko ograničenje, ne prolazimo po svima petljom, nego u pomoćnom nizu označimo početno i završno dijete koje je bilo jedanput više poprskano, npr. povećavajući odgovarajuće elemente za +1 i -1. Na kraju ćemo jednom proći po tom pomoćnom nizu i, održavajući trenutačni zbroj njegovih elemenata („broj otvorenih intervala prskanja“), saznati traženi ukupan broj prskanja za pojedino dijete.

Posebnu pažnju valja posvetiti činjenici da je niz kružni, što malo komplicira gornju ideju ažuriranja intervala koristeći pomoćni niz.

```
#!/usr/bin/python3
import sys
n, m = map(int, input().split())
k = int(input())
events = []
for i in range(k):
    x, y = map(int, input().split())
    events.append((x - 1, 'poc'))
    events.append((y - 1, 'zav'))
events.append((m, 'poc'))
a = [0 for i in range(n + 1)]

svi = 0
def prskanje(lo, hi):
    global svi
    if lo == hi: return
    svi += (hi - lo) // n
    if (hi - lo) % n == 0:
        return
```

```
    prvi = lo % n
    zadnji = (hi - 1) % n
    if prvi <= zadnji:
        a[prvi] += 1
        a[zadnji + 1] -= 1
    if zadnji < prvi:
        a[0] += 1
        a[zadnji + 1] -= 1
        a[prvi] += 1

zadnji_kraj = -1
otvorenih_intervala = 0
for t, tip in sorted(events):
    if tip == 'poc':
        if otvorenih_intervala == 0:
            prskanje(zadnji_kraj + 1, t)
        otvorenih_intervala += 1
    else:
        zadnji_kraj = max(zadnji_kraj, t)
        otvorenih_intervala -= 1

for i in range(n):
    if i > 0:
        a[i] += a[i - 1]
    print(a[i] + svi)
```