Distanciranje, 1. del

↓↓↓
Nino Bašić

→ **Naloga, ki se je lotevamo tukaj, je poenostavljena različica naloge *Measures*, ki se je pojavila na srednjeevropski računalniški olimpijadi leta 2022. Ta je potekala med 24. in 30. julijem v Varaždinu na Hrvaškem [1].**

Na ravni dolgi cesti stoji n ljudi. Ker so razdalje med njimi relativno velike, lahko cesto predstavimo s številsko premico, ljudi na njej pa s točkami. Naj bodo $x_1, x_2, \ldots, x_n$ koordinate točk, kjer ti ljudje stojijo. Brez škode lahko predpostavimo, da velja $x_1 \leq x_2 \leq \cdots \leq x_n$. Velja tudi, da so $x_1, x_2, \ldots, x_n$ cela števila.

Zaradi smrtonosne pandemije je treba poskrbeti, da razdalja med nobenima dvema osebama ne bo manjša kot D, pri čemer je D neko predpisano celo število, ki ga izbere ekspertna skupina. Vsi ljudje se hkrati odpravijo na svoje nove položaje y_1, y_2, …, y_n, tako da bo veljalo $|y_i - y_j| \geq D$ za $i \neq j$, tj., da bo razdalja med vsakima dvema vsaj D. Pri tem želimo, da bo razdalja, ki jo prehodi tisti človek, ki prehodi najdaljšo pot, čim manjša. Razdalja, ki jo prehodi i-ta oseba, je $|x_i - y_i|$. Drugače povedano, položaje $y_1, y_2, \ldots, y_n$ je treba izbrati tako, da bo vrednost

- $$\max_{1 \leq i \leq n} |x_i - y_i| \qquad (1)$$

čim manjša.

Oglejmo si preprost primer, kjer je $n = 5$, $D = 3$ ter

- $x_1 = 1$, $x_2 = 2$, $x_3 = 4$, $x_4 = 6$ in $x_5 = 7$.

Denimo, da prva oseba ostane na svojem mestu, tj. $y_1 = x_1 = 1$. Pogoj, da bo razdalja med vsakima vsaj $D = 3$, lahko dosežemo na primer z naslednjim izborom končnih položajev:

- $y_2 = 4$, $y_3 = 7$, $y_4 = 11$ in $y_5 = 14$.

Poglejmo, kolikšna je dolžina poti, ki jo morajo prehoditi posamezne osebe:

- $|1 - 1| = 0$, $|4 - 2| = 2$, $|7 - 4| = 3$, $|11 - 6| = 5$ in $|14 - 7| = 7$.

Najdaljšo pot je prehodil tisti, ki je bil v začetnem položaju x_5. Njegova pot je bila dolžine 7. To seveda ni najmanjša mogoča vrednost izraza (1). Minimum znaša namreč 3, ustrezne končne položaje $y_1, y_2, \ldots, y_5$ pa za vajo poiščite sami. (V določenih primerih lahko obstaja več različnih izborov vrednosti $y_1, \ldots, y_n$, ki minimizirajo izraz (1).)

Program, ki reši nalogo, bomo napisali v programskem jeziku Python. Še preden pa se lotimo reševanja, si naredimo funkcijo `testni_primer(a, n)`, ki bo naključno zgenerirala testni primer. Pri tem je seveda `n` število oseb, argument `a` pa omeji možen izbor števil $x_1, x_2, \ldots, x_n$, tako da bo veljalo $|x_i| \leq$ `a` za vse $1 \leq i \leq n$. Ker imamo opravka z naključnimi števili, bomo uporabili modul `random`, ki je del standardne knjižnice jezika Python.

```
import random

def testni_primer(a, n):
    return sorted([random.randint(-a, a)
    for i in range(n)])
```

Ni se težko prepričati, da med optimalnimi rešitvami obstaja tudi takšna, kjer velja $y_1 \leq y_2 \leq \cdots \leq y_n$. Denimo, da osebi, ki svojo pot začneta na x_i in x_{i+1}, končata na y_i in y_{i+1}, kjer je $y_i > y_{i+1}$. To pomeni, da se nekje na svoji poti srečata. Če bi se dogovorili za izmenjavo svojih končnih položajev, se pot nobeni od njiju zaradi tega ne bi podaljšala.

Označimo vrednost izraza (1) s T. Če ne vemo, kako bi poiskali optimalni T^* (tj. najmanjši možni T, za katerega še obstajajo $y_1, y_2, \ldots, y_n$, da velja $|y_i - y_j| \geq D$ za $i \neq j$), si lahko zastavimo nekoliko drugačno vprašanje: *Ali obstaja razporeditev, ki upošteva pravilo distanciranja, če predpišemo največjo dovoljeno razdaljo T, ki jo posamezna oseba sme prehoditi?*

Na to vprašanje pa ni tako težko odgovoriti. Seveda je lahko pri optimalni rešitvi veliko ljudi, pri katerih je prehojena razdalja manjša od T. (Lahko imamo tudi kakšno nepremično osebo, ki ostane na svojem mestu.) Seveda pa iskana rešitev ne bo nič slabša, če poleg tistega nesrečneža, ki »mora« prehoditi razdaljo T, obstajajo še drugi ljudje, ki tudi prehodijo pot največje dovoljene dolžine.

Začnimo pri osebi, ki stoji skrajno levo, tj. na položaju x_1. Za y_1 mora veljati $x_1 - T \leq y_1 \leq x_1 + T$. Nič ne bo narobe, če izberemo $y_1 = x_1 - T$, saj se na tak način razdalja med y_1 in y_2 lahko samo poveča. Tistim, ki stojijo desno od prve osebe, slednja pusti samo še več manevrskega prostora, če se sprehodi na položaj $x_1 - T$ in se tako še bolj oddalji od preostanka skupine.

Podobno, za y_2 mora veljati $x_2 - T \leq y_2 \leq x_2 + T$. Zaradi distanciranja pa mora hkrati veljati še $y_1 + D \leq y_2$. Ker želimo tistim, ki so desno od druge osebe, pustiti kar se da veliko manevrskega prostora, se bo druga oseba sprehodila čim dlje proti levi. Hkrati pa moramo paziti, da se ne približa preveč prvi osebi. Zato postavimo $y_2 = \max\{x_2 - T, y_1 + D\}$, da zadostimo obema pogojema. Če pa je druga oseba že preblizu prve, se bo sprehodila na desno in sicer ravno dovolj, da bosta na medsebojni razdalji D. Pri tem se nam lahko zgodi, da je $y_1 + D > x_2 + T$. To pa pomeni, da se druga oseba nikakor ne more dovolj oddaljiti od prve osebe. Takoj lahko zaključimo, da rešitev pri izbrani vrednosti T ne obstaja.

Za vse nadaljnje osebe lahko razmišljamo na podoben način. Ker velja $y_1 \leq y_2 \leq \cdots \leq y_i$, je dovolj, če pri izbiri y_{i+1} upoštevamo le y_i (ne pa tudi končnega položaja vseh predhodnih oseb). Če je $y_i + D > x_{i+1} + T$, rešitev ne obstaja, sicer pa postavimo $y_{i+1} = \max\{x_{i+1} - T, y_i + D\}$.

Glede na zgornji razmislek lahko napišemo funkcijo `preveri(x, D, T)`, ki kot prvi argument dobi seznam `x` s položaji vseh oseb, kot drugi in tretji argument pa vrednosti D in T iz naloge. Funkcija preveri, ali sploh obstaja rešitev, kjer je prehojena razdalja pri posamezni osebi kvečjemu T. Če rešitev ne obstaja, funkcija vrne `None`, sicer pa seznam končnih položajev.

```
def preveri(x, D, T):
    y = [None for i in range(len(x))]
    y[0] = x[0] - T
    for i in range(1, len(x)):
        if y[i - 1] + D > x[i] + T:
            return None
        y[i] = max(x[i] - T, y[i - 1] + D)
    return y
```

Zdaj lahko preverimo, ali obstaja rešitev za naš prvi zgled, kjer je denimo $T = 2$ ali pa $T = 5$.

```
>>> zgled_1 = [1, 2, 4, 6, 7]
>>> preveri(zgled_1, 3, 2)
None
>>> preveri(zgled_1, 3, 5)
[-4, -1, 2, 5, 8]
```

Naj bo T^* najmanjši možen T, za katerega obstaja rešitev. Kako pa poiščemo T^*? Ni se težko prepričati, da če rešitev ni mogoča za nek T, potem tudi ni mogoča za nobeno manjšo vrednost, saj je dovoljen premik posamezne osebe manjši, tako da imajo na voljo še manj manevrskega prostora. Če rešitev obstaja za neki T, potem lahko to isto rešitev uporabimo za vse večje vrednosti parametra T. To pa pomeni, da lahko T^* poiščemo z *binarnim iskanjem*. Če vemo, da je $T_a \leq T^* \leq T_b$, za neka T_a in T_b, lahko preverimo, ali obstaja rešitev za $(T_a + T_b)/2$. Če obstaja, potem je $(T_a + T_b)/2 \leq T^* \leq T_b$; sicer pa vemo, da je $T_a \leq T^* \leq (T_a + T_b)/2$. Postopek ponavljamo toliko časa, dokler je razlika med T_a in T_b dovolj velika.

Potrebujemo še primerni začetni vrednosti za T_a in T_b. Za T^* velja $0 \leq T^* \leq D(n-1)$. V najboljšem primeru ljudje na cesti že na samem začetku stojijo dovolj daleč, zato se nikomur ni treba premikati (od tod $0 \leq T^*$). V najslabšem primeru so vsi na istem mestu, torej $x_1 = x_2 = \cdots = x_n$. V tem primeru se lahko razporedijo takole: $y_1 = x_1$, $y_2 = x_2 + D$, $y_3 = x_3 + 2D$, …, $y_n = x_n + (n-1)D$. Največjo razdaljo prehodi najbolj desna oseba, in sicer $(n-1)D$ (od tod $T^* \leq D(n-1)$). Sami se prepričajte, da je to res najbolj neugoden scenarij.

Zdaj lahko napišemo funkcijo `distanciranje(x, D)`, ki kot argumenta dobi seznam `x` s položaji vseh oseb in vrednost D. Funkcija vrne iskano vrednost T^*.

```
def distanciranje(x, D):
    T_a = 0
    T_b = D * (len(x) - 1)
    while T_b - T_a > 1e-12:
        T_c = (T_a + T_b) / 2
        if preveri(x, D, T_c):
            T_b = T_c
        else:
            T_a = T_c
    return T_b
```

Zdaj pa funkcijo preizkusimo še na našem začetnem zgledu:

```
>>> zgled_1 = [1, 2, 4, 6, 7]
>>> distanciranje(zgled_1, 3)
3.0
```

Za vajo lahko funkcijo prilagodite tako, da bo poleg T^* vrnila še ustrezen seznam vrednosti y_1, y_2, …, y_n. Preizkusite funkcijo še na kakšnem drugem primeru; te si lahko enostavno naredite s pomočjo funkcije `testni_primer`. V uvodnem odstavku smo zahtevali, da so vrednost $x_1, \ldots, x_n$ in D cela števila. Naša rešitev te predpostavke nikjer ne uporabi in še vedno pravilno deluje, tudi če zahtevo po celoštevilskosti opustimo. Testni seznami, ki jih naredimo s pomočjo funkcije `testni_primer`, pa vsebujejo samo celo števila. Kaj pri tem opazite? Je rešitev T^* vedno celo število? Ali to ni nujno res?

Povejmo še nekaj besed o časovni zahtevnosti rešitve. Funkcija `preveri` ima zanko `for`, ki naredi $O(n)$ iteracij, vse operacije pa so elementarne, torej je časovna zahtevnost te funkcije $O(n)$. Funkcija `distanciranje` kliče funkcijo `preveri` v svoji zanki `while`. Koliko klicev naredimo? Začetni interval pri binarnem iskanju je $[T_a, T_b] = [0, D(n-1)]$. Tega razpolavljamo toliko časa, dokler ne pridemo do $T_b - T_a < 10^{-12}$. Torej naredimo $O(\log_2(nD))$ iteracij. Časovna zahtevnost funkcije `distanciranje` je zatorej $O(n \log_2(nD))$.

V drugem delu prispevka bomo rešitev še izboljšali. Nato si bomo ogledali še nekoliko zahtevnejšo različico naloge, pri kateri lahko na cesto dodamo še dodatnih M oseb, po vsakem dodajanju pa moramo izpisati T^*. Vso izvorno kodo iz pričujočega članka lahko najdete na GitHub repozitoriju [2].

Literatura

[1] Central European Olympiad in Informatics (CEOI 2022), Varaždin, Hrvaška, 24.–30. julij 2022, Naloge z rešitvami, dostopno na `https://ceoi.hsin.hr/competition/`, ogled 11. 10. 2022.

[2] N. Bašić, Distanciranje (GitHub), dostopno na `https://github.com/nbasic/presek-distanciranje`, ogled 11. 10. 2022.

Križne vsote

→ Naloga reševalca je, da izpolni bele kvadratke s številkami od 1 do 9 tako, da bo vsota števk v zaporednih belih kvadratkih po vrsticah in po stolpcih enaka številu, ki je zapisano v sivem kvadratku na začetku vrstice (stolpca) nad (pod) diagonalo. Pri tem morajo biti vse števke v posamezni vrstici (stolpcu) različne.