**1. Letala z razgledom**

Kot smo vajeni, so letala tisti krogi, katerih notranjih krogi ne vsebujejo nobenega kroga. Letalo mora imeti enega ali več kot dva kroga. Testom sta priloženi funkciji vsebovanost in letala, skopirani iz rešitve domačih nalogah. Smeš ju spreminjati (morda jo boš celo moral), kopirati in uporabljati, kakor želiš.

def vsebovanost(krogi):

vsebuje = defaultdict(list)

notranji = set()

for krog0 in krogi:

for krog1 in krogi:

x0, y0, r0 = krog0

x1, y1, r1 = krog1

if r0 > r1 and (x1 - x0) \*\* 2 + (y1 - y0) \*\* 2 < r0 \*\* 2:

vsebuje[krog0].append(krog1)

notranji.add(krog1)

return vsebuje, notranji

def letala(krogi):

vsebuje, notranji = vsebovanost(krogi)

letala = set()

for krog, vkrogu in vsebuje.items():

if krog in notranji or len(vkrogu) == 2:

continue

for vkrog in vkrogu:

if vkrog in vsebuje:

break

else:

letala.add(krog[:2])

return letala

Napiši funkcijo najvec\_oken(krogi), ki prejme seznam krogov, predstavljenih s trojkami (x, y, r). Funkcija mora vrniti koordinati središča letala z največ okni. Če letal ni ali pa je letal z največ okni več, funkcija vrne None.

**Rešitev**

Funkcijo letala spremenimo tako, da ne vrača koordinat središč temveč cele kroge - vrstico letala.add(krog[:2]) popravimo v letala.add(krog). To je potrebno narediti zato, da bo mogoče te kroge uporabljati kot ključe v slovar vsebuje.

Rešitev je potem takšna.

def najvec\_oken(krogi):

vsebuje, \_, = vsebovanost(krogi)

vsa\_letala = letala(krogi)

if not vsa\_letala:

return None

naj\_oken = max(len(vsebuje[krog]) for krog in vsa\_letala)

naj\_letala = [krog for krog in vsa\_letala if len(vsebuje[krog]) == naj\_oken]

if len(naj\_letala) != 1:

return None

return naj\_letala[0][:2]

Najprej sestavimo seznam vseh letal. Če jih ni, vrnemo None.

Nato poiščemo največje število oken. To najpreprosteje naredimo z max(len(vsebuje[krog]) for krog in vsa\_letala). Nato pripravimo seznam vseh letal, ki imajo toliko oken: [krog for krog in vsa\_letala if len(vsebuje[krog]) == naj\_oken].

Če ta seznam ne vsebuje natančno enega letala, vrnemo None. Sicer pa vrnemo središče tega letala.

Če ne obvladamo izpeljanih seznamov oziroma, v tem primeru, generatorjev, se iskanje največjega števila oken in seznama letal s toliko okni nekoliko razpihne.

def najvec\_oken(krogi):

vsebuje, \_, = vsebovanost(krogi)

vsa\_letala = letala(krogi)

if not vsa\_letala:

return None

naj\_oken = 0

for krog in vsa\_letala:

if len(vsebuje[krog]) > naj\_oken:

naj\_oken = len(vsebuje[krog])

naj\_letala = []

for krog in vsa\_letala:

if len(vsebuje[krog]) == naj\_oken:

naj\_letala.append(krog)

if len(naj\_letala) != 1:

return None

return naj\_letala[0][:2]

Tole pa je klasična rešitev, ki bi jo napisali v vsakem "normalnem" jeziku. Brez generatorjev in kar hitra.

def najvec\_oken(krogi):

vsebuje, \_, = vsebovanost(krogi)

vsa\_letala = letala(krogi)

naj\_letala = []

naj\_oken = 0

for krog in vsa\_letala:

ta\_oken = len(vsebuje[krog])

if ta\_oken > naj\_oken:

naj\_letala = []

naj\_oken = ta\_oken

if ta\_oken == naj\_oken:

naj\_letala.append(krog)

if len(naj\_letala) != 1:

return None

return naj\_letala[0][:2]

Tu je naj\_oken spet največje število oken, na katerega smo naleteli doslej, naj\_letala pa so vsa (doslej videna) letala s takšnim številom oken. Če naletimo na letalo z večjim številom oken, pobrišem seznam (vsa letala, ki smo jih nabrali doslej, imajo manj oken od tega, ki ga gledamo zdaj). Obenem si zapomnimo, da je največje število oken prav to število oken. In če vidimo, da ima to letalo naj\_oken, ga dodamo v seznam naj\_letala. Konec je pa takšen kot prej.

**2. Število oken**

Tudi marsovske ladje imajo okna. In ptiči imajo okna (ki se jim reče oči). Vse ima okna.

Napiši funkcijo stevilo\_oken(krog, hierarhija). Ta kot argument prejme krog in slovar, katerega ključi so krogi, ki vsebujejo vsaj en notranji krog, pripadajoče vrednosti pa seznami neposredno vsebovanih notranjih krogov (ne pa tudi krogov znotraj teh krogov). Če krog ne vsebuje nobenega kroga, ne nastopa kot ključ v slovarju.

Funkcija mora vrniti število krogov, ki so posredno ali neposredno znotraj podanega kroga, ki ne vsebujejo nobenega kroga. **Dodatek:** Če podani krog ne vsebuje nobenih krogov, naj funkcija vrne 1.

**Rešitev**

Klasična rekurzivna funkcija. V bistvu je enaka funkciji brez\_potomcev.

def stevilo\_oken(krog, hierarhija):

if krog not in hierarhija:

return 1

return sum(stevilo\_oken(podkrog, hierarhija) for podkrog in hierarhija[krog])

**3. Pari**

Napiši funkcijo pari(krogi, razdalje). Ta prejme seznam središč krogov (par koordinat, brez polmerov!) in seznam razdalj med vsemi pari krogov. (Z razdaljo tu mislimo razdaljo med krožnicama, ne med središčema.) Ta seznam torej vsebuje toliko elementov, kolikor je parov krogov. Vsakemu paru ustreza terka (razdalja, {(x0, y0), (x1, y1)}), ki pomeni, da je razdalja med krogoma s središčema v (x0, y0) in (x1, y1) enaka razdalja. Seznam je že urejen naraščajoče po razdaljah.

Funkcija mora vrniti seznam parov najbližjih krogov. Vsak par naj bo predstavljen z množico, ki vsebuje koordinati središč. Funkcija naj deluje preprosto tako, da vzame najbližji par in ga da v seznam, nato doda naslednji najbližji par (seveda brez teh dveh) in tako naprej. Če je število krogov liho, naj zadnji, preostali krog ignorira. Za formalnejšo razlago argumentov in rezultata poglej v teste.

Pri reševanju lahko morda ignorirate ta ali oni argument. Štelo pa se vam bo v dobro, če funkcija ne bo brez potrebe stalno računala enih in istih stvari.

**Rešitev**

Tole je predvsem naloga iz množic.

En način je tak:

def pari(krogi, razdalje):

krogi = set(krogi)

pari = []

for r, par in razdalje:

if par <= krogi:

pari.append(par)

krogi -= par

return pari

Seznam krogov spremenimo v množico krogov, ker bo tako veliko hitreje in udobneje. Množica krogi bo vsebovale vse kroge, ki jih še nismo uporabili.

Gremo čez vse pare in razdalje - pri čemer razdalij niti ne potrebujemo. Za vsak par pogledamo, ali je podmnožica še ne uporabljenih krogov. Če je tako, ga dodamo v seznam parov, kroga pa odstranimo iz množice.

Drugi način ignorira argument krogi in raje sestavlja seznam uporabljenih krogov.

def pari(krogi, razdalje):

uporabljeni = set()

pari = []

for r, par in razdalje:

if not par & uporabljeni:

pari.append(par)

uporabljeni |= par

return pari

Gremo čez vse pare. Za vsakega preverimo njegov presek z množico uporabljenih krogov. Biti mora prazen - saj smo v nasprotnem primeru že uporabili enega (ali oba) kroga iz para. Če je torej prazen, dodamo par v seznam parov, kroga pa "priunijamo" v množico uporabljenih krogov.

Rešitve, ki bi ignorirale razdalje in vedno znova iskale najbližji par, pa bi bile slabe, ker bi bile počasne pri velikem številu krogov. Na to je se je nanašalo navodilo, ki je odsvetovalo, da vedno znova računate eno in isto.

**4. Datoteke s krogi**

Recimo, da so vse koordinate središč in polmeri cela števila med 0 in 999. V tem primeru bi jih lahko zapisali v datoteko v takšni obliki.

150 23 38

512 418 12

1 0 123

123 11 5

Trenutni format datoteke pa je takšen, da je vsako število zapisano s tremi mesti (z vodilnimi ničlami; 42 je zapisano kot 042) in stlačeno v eno vrstico. Tisto zgoraj je torej zapisano kot 150023038512418012001000123123011005.

Napiši funkcijo prepisi\_koordinate(vhodna, izhodna), ki dobi imeni dveh datotek. Datoteka vhoda vsebuje podatke v drugi, stisnjeni obliki, v eni sami vrstici. Funkcija naj v datoteko z imenom izhodna zapiše podatke v drugi obliki.

**Rešitev**

Tole je mogoče rešiti na kup načinov. Meni je najbolj všeč, narediti dve zanki, eno s korakom 9 in eno s korakom 3. Ta po 9 se nanaša na vrstice, ona po 3 na številke znotraj teh vrstic. Znotraj zunanje zanke pripravimo prazno vrstico. V notranji zanki jemljemo po tri števke in jih spremenimo v int ter ustrezno oblikovane dodamo v vrstico, za številko pa dodamo še presledek. Po notranji zanki odstranimo zadnji presledek in dodamo "\n". To zapišemo v datoteko.

def prepisi\_koordinate(vhodna, izhodna):

podatki = open(vhodna).read().strip()

izhod = open(izhodna, "wt")

for vrsta in range(0, len(podatki), 9):

vrstica = ""

for i in range(vrsta, vrsta + 9, 3):

stevilka = int(podatki[i:i + 3])

vrstica += f"{stevilka:3} "

vrstica = vrstica[:-1] + "\n"

izhod.write(vrstica)

Če hočemo reč malo skrajšati, potlačimo zadnjo zanko v en sam join, ki med številke postavi presledek.

def prepisi\_koordinate(vhodna, izhodna):

podatki = open(vhodna).read().strip()

izhod = open(izhodna, "wt")

for vrsta in range(0, len(podatki), 9):

izhod.write(" ".join(f"{int(podatki[i:i+3]):3}"

for i in range(vrsta, vrsta + 9, 3)) + "\n")

Še bolj stlačena rešitev je

def prepisi\_koordinate(vhodna, izhodna):

podatki = open(vhodna).read().strip()

open(izhodna, "wt").write("\n".join(

" ".join(f"{int(podatki[i:i+3]):3}"

for i in range(vrsta, vrsta + 9, 3))

for vrsta in range(0, len(podatki), 9)))

Katera je najboljša? Najbrž prva. :)

**5. Strelec**

Napiši razred Strelec z naslednjimi metodami.

* Konstruktor ne sprejema argumentov in naredi, kar je potrebno.
* dodeli(krog): dodeli strelcu ladjo, opisano s terko, ki vsebuje koordinati in polmer. Strelcu je lahko dodeljenih več ladij. Ladje se lahko "prekrivajo", ker se nahajajo ena za drugo.
* strel(x, y): uniči vse ladje, ki vsebujejo to koordinato (isti strel lahko prestreli več ladij, naši laserji so močni). Funkcija vrne True, če je strel zadel kakšno ladjo in False, če je ni.
* preostalih() vrne število ladij, ki so dodeljene strelcu in jih le-ta še ni uničil.

**Rešitev**

* Očitno bo potrebno shranjevati seznam ali množico dodeljenih ladij. Konstruktor torej pripravi prazno množico.
* dodeli preprosto postavi nov krog v to množico.
* strel sestavi novo množico, ki vsebuje vse kroge, ki preživijo strel. To je lažje od brisanja saj, vemo, ni zdravo iti z zanko prek neke stvari, iz katere brišemo elemente.
* preostalih vrne velikost množice.

Torej tako:

class Strelec:

def \_\_init\_\_(self):

self.dodeljeno = set()

def dodeli(self, krog):

self.dodeljeno.add(krog)

def strel(self, x, y):

prej = self.preostalih()

self.dodeljeno = {(x0, y0, r0) for x0, y0, r0 in self.dodeljeno

if (x - x0) \*\* 2 + (y - y0) \*\* 2 > r0 \*\* 2}

return self.preostalih() != prej

def preostalih(self):

return len(self.dodeljeno)

Pri strel si najprej zapomnimo, koliko krogov nas še čaka. Funkcija potem vrne, ali se je število čakajočih krogov spremenilo.

Metodo strel bi lahko napisali tudi tako:

def strel(self, x, y):

zadeti = {(x0, y0, r0) for x0, y0, r0 in self.dodeljeno

if (x - x0) \*\* 2 + (y - y0) \*\* 2 <= r0 \*\* 2}

self.dodeljeno -= zadeti

return zadeti != set()

Pravi programerji v Pythonu pa vedo, da se zadnja vrstica napiše

return bool(zadeti)