resitev

January 28, 2024

Marsovci še kar najedajo. Zdaj so se parkirali pred Zemljo in imajo očitno slabe namene. Na srečo je Slovenska vojska pripravljena na vse: na strateških mestih ima parkirane ČNVL-je (Čisto nevidne vesoljske ladje), ki bodo uničile marsovsko floto. Manjka jim le še strateški načrt. Ta je predmet te domače naloge.

Za razumevanje naloge si najprej oglejte animacijo. Marsovske ladje so predstavljene s krogci, hrabri ČNVL pa s kvadratki.

0.1 O testih

V testih so funkcije že zastavljene: imajo nekaj dokumentacije v enem od običajnih oblik, poleg tega pa so označeni tipi argumentov in rezultatov. Če tega ne razumeš, ignoriraj. Če razumeš, ti lahko služi kot dodatna dokumentacija.

Ker je vrstni red elementov seznamov, ki jih vračajo tvoje funkcije včasih poljuben, bodo testne funkcije včasih uredile rezultat tvoje funkcije in ga primerjale z urejenim pričakovanim rezultatom. Naj te to ne zmede: funkciji ni potrebno ničesar urejati.

Če tvoja funkcija vrača seznam, kjer testi pričakujejo terko, se bodo (najbrž) pritožili. Bodi pozoren na oklepaje v izpisu.

`(3)` ni terka temveč samo `3` v oklepaju. `(3, )` je terka.

Če test pričakuje, da bo funkcija vrnila `[(4 + 6) / 2, (3 + 9) / 3]`, seveda pričakuje `[5, 4]`. V tej obliki je zapisano le, da lažje razberemo, kaj mora funkcija računati.

Test zadnje funkcije deluje tako, da razpostavi naključne množice s po 20 točkami v bližini treh vnaprej izbranih točk. Nato izračuna središče vsake od teh množic. Te točke predstavljajo skupine marsovcev, ki jih mora odkriti funkcija. Pričakovati je, da bo v 100 poskusih gotovo vsaj enkrat (najbrž pa 90-krat) našla pravi razpored.

0.2 Za oceno 6

- Napiši funkcijo `razdalja(x, y)`, ki prejme terki s koordinatami dveh točk in vrne evklidsko razdaljo med njimi. Da bomo pripravljeni tudi na transdimenzionalna bitja, je dimenzija točk lahko poljubna (od 1 do kolikor je treba). Razdaljo v večdimenzionalnem prostoru računamo podobno kot v dvo- ali tridimenzionalnem

$$d(x, y) = \sqrt{\sum_{i=0}^{n-1} (x_i - y_i)^2}$$

  - Klic `razdalja((5, 12), (2, 16))` vrne 5 (to je, $\sqrt{(5-2)^2+(12-16)^2} = \sqrt{3^2+4^2}$).
  - Klic `razdalja((5, ), (8, )` (razdalja na premici) vrne 3 (to je $\sqrt{(5-8)^2}$).
  - Klic `razdalja((1, 7, 4, 2, 6, -7, 4), (-1, 10, 5, 2, 6, -8, 3)` (razdalja v 7-dimenzionalnem prostoru) vrne 4.

- Napiši funkcijo `najblizja(marsovec, ladje)`, ki dobi terko s koordinatami marsovske ladje in seznam terk s položaji naših ladij, ČNVL-jev. Vrniti mora koordinate najbližjega ČNVL-a. (Ta bo zadolžen za napad na to marsovsko ladjo.)

  - Klic `najblizja((1, 1), [(0, 0), (5, 2), (-8, 3)]` vrne `(0, 0)`, saj je marsovski ladji na koordinatah `(1, 1)` najbližji ČNVL na koordinatah `(0, 0)`.
  - Klic `najblizja((-7, 2, 5, 1), [(0, 0, 5, 2), (5, 2, 5, 2), (-8, 3, 5, 2)])` vrne `(-8, 3, 5, 2)`.

- Napiši funkcijo `dodeli_ladje(marsovci, ladje)`, ki prejme seznam marsovskih in naših ladij. Vrniti mora seznam terk, ki je enako dolg kot seznam `marsovci`: i-ti element rezultata vsebuje koordinate ČNVL-ja, ki je najbližji i-ti marsovski ladji.

  - Klic `dodeli_ladje([(1, 1), (0, 0), (-7, 2), (5, 1)], [(0, 0), (5, 2), (-8, 3)])` vrne `[(0, 0), (0, 0), (-8, 3), (5, 2)]`, saj je prvima dvema marsovcema najbližji ČNVL na koordinatah `(0, 0)`, tretjemu ČNVL na `(-8, 3)`, četrtemu pa tisti na `(5, 2)`.

- Napiši funkcijo `skupine_marsovcev(marsovci, ladje)`, ki vrne seznam množic terk s koordinatami marsovskih ladij. Bistvo te funkcije je grupiranje marsovskih ladij. Če bodo marsovke ladje uničevali trije ČNVL-je, se vsaka množica nanaša na enega od njih in vsebuje tiste ladje, ki jih bo uničil ta, pripadajoči ČNVL.

  Vrstni red množic je lahko poljuben. Predpostaviti smeš, da bo vsak ČNVL uničil vsaj eno ladjo (oziroma: s tem vprašanjem se ne ukvarjaj.)

  Klic

```
skupine_marsovcev(
    [(0, 1), (2, 2), (51, 52), (100, 100),
     (1, 3), (4, 5), (45, 50), (103, 98)],

    [(0, 0), (50, 50), (100, 100)]))
```

  vrne

```
[{(0, 1), (2, 2), (1, 3), (4, 5)},
 {(100, 100), (103, 98)},
 {(51, 52), (45, 50)}]
```

  Prva množica vsebuje koordinate marsovskih ladij, ki jih uniči NČVL na koordinatah `(0, 0)`, druga ladje, ki jih uniči NČVL na koordinatah `(100, 100)` in tretja tiste, ki jih uniči `(50, 50)`.

  Funkcija v bistvu vrne množice koordinat ladij, ki so iste barve.

  Namig: čeprav ni videti tako, ti bodo tu prišli prav slovarji.

0.2.1 Rešitev

razdalja Ker je število dimenzij poljubno, bomo kvadrate razlik seštevali z zanko.

[1]:
```
from math import sqrt

def razdalja(x, y):
    r = 0
    for xi, yi in zip(x, y):
        r += (xi - yi) ** 2
    return sqrt(r)
```

Pravzprav pa znamo tudi preprosteje. (Ali bolj zapleteno, kakor za koga.)

[2]:
```
def razdalja(x, y):
    return sqrt(sum((xi - yi) ** 2 for xi, yi in zip(x, y)))
```

najblizja Iskanje "najboljšega" elementa po določenem kriteriju je stara naloga.

[3]:
```
def najblizja(marsovec, ladje):
    naj_ladja = naj_razdalja = None
    for ladja in ladje:
        razd = razdalja(marsovec, ladja)
        if naj_razdalja is None or razd < naj_razdalja:
            naj_razdalja = razd
            naj_ladja = ladja
    return naj_ladja
```

Zanimivo je, da je precej študentov pozabljalo vrstico `naj_razdalja = razd`, zato so vrnili zadnjo ladjo, ki je bila bližja od prve ladje. Nerodno je, da tega niso odkrili testi za to funkcijo temveč šele testi za eno naslednjih. Pri pisanju testov je pač težko pomisliti na vse možne napake.

Kdor zna nekaj več, pa je naredil tole:

[4]:
```
def najblizja(marsovec, ladje):
    return min(ladje, key=lambda ladja: razdalja(marsovec, ladja))
```

dodeli_ladje Gremo po seznamu marsovcev in v nov seznam zlagamo pripadajoče najbližje ladje.

[5]:
```
def dodeli_ladje(marsovci, ladje):
    dodelitev = []
    for marsovec in marsovci:
        dodelitev.append(najblizja(marsovec, ladje))
    return dodelitev
```

Lahko pa tudi tako.

[6]:
```
def dodeli_ladje(marsovci, ladje):
    return [najblizja(marsovec, ladje) for marsovec in marsovci]
```

skupine_marsovcev Tu se je najbolj splačalo sestaviti slovar, katerega ključi so naše ladje, pripadajoči seznami pa marsovske ladje, ki jih bo uničila ta naša ladja. Funkcija potem vrne seznam vrednosti v tem slovarju.

[7]:
```
from collections import defaultdict


def skupine_marsovcev(marsovci, ladje):
    dodelitve = defaultdict(set)
    for marsovec, ladje in zip(marsovci, dodeli_ladje(marsovci, ladje)):
        dodelitve[ladje].add(marsovec)
    return list(dodelitve.values())
```

0.3 Za oceno 7

- Napiši funkcijo `sredisce(s)`, ki prejme seznam ali množico koordinat točk in vrne terko s koordinatami njihovega središča. Koordinate te točke so izračunane kot poprečne koordinate točk v `s` po vsaki dimenziji posebej.

  Število točk v `s` je poljubno. Točke so poljubno-dimenzionalne.

  – Klic `sredisce([(2, 8), (6, 10)` vrne `(4, 9)`.

- Napiši funkcij `razpostavi_ladje(skupine)`, ki poišče optimalno razpostavitev ladij. Funkcija prejme neko razdelitev marsovskih ladij v skupine; skupine so opisane s seznamom množic koordinat (torej v obliki, ki jo vrne funcija `skupine_marsovcev`, ki je opisana spodaj). Funkcija mora vrniti seznam koordinat središč skupin.

  – Klic

  ```
  razpostavi_ladje([{(0, 1), (2, 2), (1, 3), (4, 5)},
                    {(100, 101), (103, 98)},
                    {(51, 52), (45, 50)}])
  ```

  vrne seznam, ki vsebuje

  ```
  [((0 + 2 + 1 + 4) / 4, (1 + 2 + 3 + 5) / 4),
   ((100 + 103) / 2, (101 + 98) / 2),
   ((51 + 45) / 2, (52 + 50) / 2)]
  ```

  (Seveda ne vsebuje teh izrazov, temveč ustrezne vrednosti.)

0.3.1 Rešitev

sredisce Nalogo je možno rešiti na kup načinov. Tu kar takoj pokažimo enega elegantnejših.

[8]:
```
def sredisce(s):
    p = [0] * len(s[0])
    for x in s:
        p = [pi + xi for pi, xi in zip(p, x)]
    return tuple(pi / len(s) for pi in p)
```

V `p` bomo sešteli vse elemente `s` po koordinatah. Najprej vanj vpišemo toliko ničel, kolikor je dimenzij. Nato gremo prek vseh točk v `s` in zamenjamo `p` s seznamom, ki vsebuje vsoto istoležnih elementov `p`-ja in `s`-ja. Na koncu vrnemo terko, ki vsebuje elemente `p`-ja deljene s številom ladij.

Nekoliko splošnejša rešitev je tale:

[9]:
```
def sredisce(s):
    p = None
    for x in s:
        if p == None:
            p = x
        else:
            p = [pi + xi for pi, xi in zip(p, x)]
    return tuple(pi / len(s) for pi in p)
```

Namesto da `p` inicializiramo s samimi ničlami, ga v začetku nastavimo na `None` in potem zanj uporabimo prvi element. Ker ta rešitev ne zahteva elementa z indeksom 0, deluje tudi, če je `s` generator. Naloga tega ni zahtevala, nič pa ni narobe, če se spomnimo, kako se to naredi.

`razpostavi_ladje` Tu ni kaj pametovati: vrniti moramo seznam središč skupin, torej:

[10]:
```
def razpostavi_ladje(skupine):
    return [sredisce(v) for v in skupine]
```

0.4 Za oceno 8

- Napiši funkcijo `optimiraj_ladje(marsovci, ladje)`, ki prejme položaje marsovcev in neko začetno pozicijo naših ladij. Funkcija naj izmenično ponavlja naslednja koraka:
  - določi skupine, ki pripadajo posamezni ladji, ko to počne funkcija `skupine_marsovcev`;
  - na podlagi teh skupin, naj izračuna boljše položaje ladij, se pravi, prestavi vsako ladjo v središče skupine, ki ji je dodeljena;
  - ker so se ladje premaknile, naj izračuna novo dodelitev marsovcev;
  - ker to spremeni skupine, naj premakne ladje v nova središča skupin;
  - ker so se ladje premaknile …

  Prej ko slej se bodo ladje nehale premikati in skupine nehale spreminjati. Takrat naj se postopek ustavi. Funkcija naj vrne končne koordinate ladij.

  Funkcija v bistvu izvaja optimizacijo, ki jo kaže video. Video, konkretno, predstavlja štiri klice funkcije.

  Za pomoč je v testu za oceno 9 shranjen potek položajev ladij v zadnjem testu. Če ga funkcija izpisuje, mora biti (do določenega števila decimalnih mest) enak.

0.4.1 Rešitev

Zapomnili si bomo položaje ladij in zanko izvajali, dokler je novi položaj drugačen kot stari. V zanki pa določimo (nove) skupine marsovcev in razpostavimo ladje.

[11]:
```
def optimiraj_ladje(marsovci, ladje):
    ladje_prej = None
    while ladje != ladje_prej:
        ladje_prej = ladje
        ladje = razpostavi_ladje(skupine_marsovcev(marsovci, ladje))
    return set(ladje)
```

0.5 Za oceno 9

- Za čim večjo učinkovitost napada morajo biti razdalje med ČNVL in marsovci čim manjše. Napiši funkcijo `kvaliteta(marsovci, ladje)`, ki prejme seznam položajev marsovskih ladij in množico položajev ČNVL-jev. Vrne naj vsoto vseh razdalij med marsovskimi ladjami in najbližjim ČNVLjem za vsako ladjo. Z drugimi besedami, vrne skupno dolžino črt na sliki.
- Napiši funkcijo `nakljucna(marsovci, ladij)`, ki vrne nek naključen razpored podanega števila ladij. Funkcija mora vrniti seznam, ki vsebuje `ladij` terk koordinat. Koordinate so izžrebane iz koordinat marsovskih ladij. Za `i`-to koordinato izberemo koordinato `i` neke naključne marsovske ladje.

  Klic

  ```
  nakljucna([(5, 8),
             (2, 17),
             (1, 33),
             (4, 9)], 3)
  ```

  bi lahko vrnil, recimo `[(1, 8), (4, 17), (2, 8)]`. Vsaka prva koordinata je žrebana iz prvih koordinata in druga iz drugih.

0.5.1 Rešitev

`kvaliteta(marsovci, ladje)` Vrniti je potrebno vsoto razdalij med marsovci in ladjami, ki jim jih dodelimo, torej:

[12]:
```
def kvaliteta(marsovci, ladje):
    return sum(razdalja(marsovec, ladja)
               for marsovec, ladja in zip(marsovci, dodeli_ladje(marsovci,␣
 ↪ladje)))
```

Pravzaprav pa je celo krajše, če ne kličemo `dodeli_ladje` in se izognemo `zip`-u.

[13]:
```
def kvaliteta(marsovci, ladje):
    return sum(razdalja(marsovec, najblizja(marsovec, ladje)) for marsovec in␣
 ↪marsovci)
```

`nakljucna` Je bila to najtežja funkcija? Mogoče. Rešitev brez hujših trikov je takšna.

[14]:
```
def nakljucna(marsovci, ladij):
    moznosti = [[] for _ in marsovci[0]]
    for marsovec in marsovci:
```

```
        for i, x in enumerate(marsovec):
            moznosti[i].append(x)

    ladje = []
    for _ in range(ladij):
        nova = []
        for izbor in moznosti:
            nova.append(random.choice(izbor))
        ladje.append(tuple(nova))
    return ladje
```

V prvem delu funkcije sestavimo sezname seznamov: `i`-ti element vsebuje seznam vseh možnih `i`-tih koordinat.

V drugem delu v `nova` sestavljamo koordinate ladje tako, da gremo prek vseh dimenzij in iz vsake naključno izberemo eno.

Prvemu delu se lahko izognemo tako, da gremo prek dimenzij in za vsako dimenzijo izžrebamo nekega marsovca ter v `nova` dodamo ustrezno koordinato.

[15]:
```
def nakljucna(marsovci, ladij):
    dim = len(marsovci[0])
    ladje = []
    for _ in range(ladij):
        nova = []
        for i in range(dim):
            nova.append(random.choice(marsovci)[i])
        ladje.append(tuple(nova))
    return ladje
```

Tole gre očitno v eno vrstico.

[16]:
```
def nakljucna(marsovci, ladij):
    return [tuple(random.choice(marsovci)[i] for i in range(len(marsovci[0])))
            for _ in range(ladij)]
```

Lepše pa je s tem trikom.

[17]:
```
def nakljucna(marsovci, ladij):
    moznosti = list(zip(*marsovci))
    return [tuple(random.choice(izbor) for izbor in moznosti) for _ in␣
 ↪range(ladij)]
```

To je po zgledu prve funkcije. Prva vrstica ustreza prvemu delu, druga drugemu.

0.6 Za oceno 10

- Napiši funkcijo `optimiraj_ladje_2(marsovci, ladje)`, ki je enaka funkciji `optimiraj_ladje`, le da namesto pozicij vrne par `(kvaliteta, pozicije)`, pri čemer je kvaliteta izračunana s funkcijo za oceno 9.

- Recimo, da začnemo optimizacijo z nekim naključnim položajem. Kot vidimo iz tretjega primera v videu, se lahko včasih zgodi, da imamo smolo in končamo s kakim slabim končnim razporedom. Napiši funkcijo `planiraj(marsovci, ladij)`, ki si stokrat izmisli nek naključen razpored (s funkcijo `nakljucni`) in izvede celoten postopek optimiranja za podano število `ladij`. Dobivala bo različno kvalitetne razporede. Vrniti mora najboljši končni razpored, na katerega je naletela.

0.6.1 Rešitev

`optimiraj_ladje_2` Tule le pokličemo funkciji, ki ju že imamo.

[18]:
```
def optimiraj_ladje_2(marsovci, ladje):
    pozicije = optimiraj_ladje(marsovci, ladje)
    return kvaliteta(marsovci, pozicije), pozicije
```

`planiraj` To pa je repriza funkcije `najblizja_ladja`, le da ne iščemo najbližje ladje temveč najboljši razpored.

Ker gre za nalogo za oceno 10, jo naredimo le po najhitrejši različici - sploh zato, ker naj je `optimiraj_ladje_2` namignila, dala terke, med katerimi je potrebno poiskati najmanjšo in vrniti njen element z indeksom `1`.

[19]:
```
def planiraj(marsovci, ladij):
    return min(optimiraj_ladje_2(marsovci, nakljucna(marsovci, ladij)) for _ in␣
 ↪range(100))[1]
```