resitev

January 28, 2024

1 Specops

Izkazalo se je, da Marsovci včasih tudi koga ugrabijo. V tej nalogi bomo vadili reševanje.

Najprej celotna slika: čeprav so marsovske ladje okrogle, je smrdljivi kevder, v katerega zapirajo ugrabljene, pravokotne oblike. Vsaka soba (razen robnih) ima vrata v sosednje štiri. Vrata se vsakih deset sekund odprejo za trenutek, torej prehod skozi sobo vedno traja 10 sekund. (Ugrabljenci so omamljeni, zato kljub odpiranju vrat ne morejo nikamor.)

Vsako polno uro se marsovski stražarji teleportirajo v določene sobe in prehodijo določeno poti. Vse je vedno enako. Točne podatke bomo dobili od MOSSAD-a, primer pa je na sliki.

Kje so ugrabljenci, ne vemo zagotovo, prepričani pa smo, da v eni od sob, ki jih stražarji največkrat obiščejo. Na sliki sta dve takšni sobi, `(6, 4)` in `(4, 5)`; obiskani sta po trikrat.

Reševalno operacijo bodo izvedli slovenski specialci. V istem trenutku, ko se v klet teleportirajo marsovci, se bodo tudi specialci teleportirali v sobo `(0, 0)`. Vsak bo šel po predvideni poti.

- Če specialec sreča marsovca, ga prime za ušesa (ali antene, bogve, kaj imajo) in ga zadrži v tej sobi.
- Če v sobi hkrati naletita na marsovca dva specialca, z njim ostane tisti specialec, ki je prej v seznamu. Ker vojska in hierarhija.
- Na koncu (ko so vsi specialci bodisi zaposleni z marsovci bodisi so končali svoje poti) morajo biti vsi marsovci prijeti, poleg tega pa mora biti po en specialec v vsaki sobi, kjer bi lahko bil ugrabljenec. Morebitni dodatni specialci so nepomembni.

Po koncu operacije teleportiramo specialce skupaj z ugrabljenci nazaj domov, pa še marsovce mimogrede ugrabimo. Zanalašč.

Primer uspešno načrtovane akcije je na sliki.

Simulacijo boste napisali za oceno 9. Za ostale ocene boste pisali funkcije, delno povezane s scenarijem; nekatere vam lahko pomagajo, nekatere morda tudi samo zato, da si znate kaj izpisati in izrisati.

Poti v nalogah so opisane v obliki `"vv^^^^>>v<"`, pri čemer posamični znaki pomenijo premike. Kot kaže slika, koordinata y narašča v smeri `v` in pada v smeri `^`. Poti marsovcev na sliki so torej takšne:

```
poti_s_slike = [
    (0, 6, ">^^>^>>vv>^>>v>"),  # rdeči
    (2, 4, ">>vv<<^"),  # zeleni
    (5, 3, ">vv<v<^<vv>>"),  # modri
```

```
    (8, 8, "^^^^<<^^<<<<<"),  # oranžni
]
```

1.1 Za oceno 6

- Napiši funkcijo `koraki(x, y, pot)`, ki prejme začetni koordinati in pot ter vrne zaporedje koordinat polj, ki jih ta, ki gre po tej poti, obišče.

  Klic `koraki(10, 80, "vv>>>^<")` vrne `[(10, 80), (10, 81), (10, 82), (11, 82), (12, 82), (13, 82), (13, 81), (12, 81)]`.

- Napiši funkcijo `cilj(x, y, pot)`, ki vrne polje, na katerem se pot konča.

  Klic `cilj(3, 6, "<<v")` vrne `(1, 7)`

- Napiši funkcijo `cilji(opisi)`, ki prejme seznam terk `(x, y, pot)` in vrne seznam končnih lokacij.

  Klic `cilji([(3, 6, "<<v"), (2, 1, "vv"), (5, 5, "")])` vrne `[(1, 7), (2, 3), (5, 5)]`.

- Napiši funkcijo `najveckrat_obiskana(opisi)`, ki prejme podobne argumente kot `cilji` in vrne množico največkrat obiskanih polj. Če isti stražar obišče isto polje večkrat, se to šteje za več obiskov.

  Klic `najveckrat_obiskana(poti_s_slike)`, kjer so `poti_s_slike` poti s slike, vrne `{(4, 5), (6, 4)}`.

1.1.1 Rešitev

koraki

[1]:
```
def koraki(x, y, pot):
    polja = [(x, y)]
    for c in pot:
        if c == ">":
            x += 1
        elif c == "<":
            x -= 1
        elif c == "v":
            y += 1
        elif c == "^":
            y -= 1
        polja.append((x, y))
    return polja
```

Bolj zanimivo je tako:

[2]:
```
def koraki(x, y, pot):
    polja = [(x, y)]
    for c in pot:
        x, y = {">": (x + 1, y), "<": (x - 1, y),
                "^": (x, y - 1), "v": (x, y + 1)}[c]
```

```
        polja.append((x, y))
    return polja
```

cilj Funkcija `cilj` zgolj vrne zadnji element, ki ga pridela `korak`.

[3]:
```
def cilj(x, y, pot):
    return koraki(x, y, pot)[-1]
```

cilji Funkcija `cilji` le pokliče `cilj` za vsak opis poti.

[4]:
```
def cilji(opisi):
    return [cilj(x, y, pot) for x, y, pot in opisi]
```

najveckrat_obiskana Pri tej funkciji se moramo izogniti pasti, da bi uporabili funkcijo `obiskana`. Ta je neuporabna, ker navodila pravijo: "Če isti stražar obišče isto polje večkrat, se to šteje za več obiskov." V množici pa se večkratni obiski izgubijo.

Tu povadimo stopnjevanje pridevnikov (emm, prislovov?). Uporabiti slovar bi bilo *praktično*. Uporabiti `defaultdict` bi bilo *bolj praktično*. Uporabiti `Counter` je najbolj *praktično*.

[5]:
```
def najveckrat_obiskana(opisi):
    c = Counter()
    for x, y, pot in opisi:
        c.update(koraki(x, y, pot))
    pogostosti = c.most_common()
    naj = pogostosti[0][1]
    return {polje for polje, f in pogostosti if f == naj}
```

1.2 Ocena 7

Specialce označimo z velikimi črkami angleške abecede (A, B, C, D, E, F, G, H, I, J) in marsovce z malimi. Tako enih kot drugih je največ deset.

- Napiši funkcijo `situacija(specialci, marsovci, sirina, visina)`, ki prejme seznam koordinat specialcev, seznam koordinat marsovcev. Vrniti mora seznam seznamov množic ter širino in višino zemljevida. Če se seznam recimo imenuje `s`, bo `s[y][x]` množica vseh, ki se nahajajo v sobi s koordinatama (x, y).

  Klic

  ```
  situacija([(1, 0), (0, 2), (3, 1), (0, 2)],
            [(2, 2), (3, 1), (3, 1), (1, 1)], 4, 3)
  ```

  vrne

  ```
  [[set(),       {'A'},  set(),     set()          ],
   [set(),       {'d'},  set(),     {'C', 'c', 'b'}],
   [{'D', 'B'},  set(),  {'a'},     set()          ]]
  ```

- Funkcija `znak(m)` prejme množico črk.

  - Če je množica prazna, funkcija vrne `"."`.
  - Če vsebuje en element, vrne ta element.
  - Če vsebuje več kot en element, vrne niz s številom elementov. Če je velikost množice 3, vrne `"3"`. Predpostaviti smeš, da velikost ne bo večja od 9.

- Funkcija `izris(polozaj)` prejme seznam seznamov množic, kakršnega vrne funkcija `situacija` in vrne niz z izpisom v naslednji obliki:

  ```
  .A..
  .d.3
  2.a.
  ```

  Dejanski niz, ki ga vrne funkcija, je seveda, `".A..\n.d.3\n2.a."`, tole zgoraj je že njegov izpis.

1.2.1 Rešitev

`situacija` Ta funkcija se je izkazala za kar zoprno. Recimo, da imamo:

[6]:
```
sirina = 5
visina = 3
```

Prvi problem je bil, da takale inicializacija seznama:

[7]:
```
polje = [[set()] * sirina] * visina
```

izgleda pravilna

[8]:
```
polje
```

[8]:
```
[[set(), set(), set(), set(), set()],
 [set(), set(), set(), set(), set()],
 [set(), set(), set(), set(), set()]]
```

vendar ni:

[9]:
```
polje[2][1].add("A")

polje
```

[9]:
```
[[{'A'}, {'A'}, {'A'}, {'A'}, {'A'}],
 [{'A'}, {'A'}, {'A'}, {'A'}, {'A'}],
 [{'A'}, {'A'}, {'A'}, {'A'}, {'A'}]]
```

V vseh elementih `polje` se nahaja ena in ista množica. Temu smo posvetili večji del predavanj o imenskih prostorih (zapiski).

Polje je potrebno inicializirati z

[10]:
```
polje = []
for y in range(visina):
```

```
    vrstica = []
    for x in range(sirina):
        vrstica.append(set())
    polje.append(vrstica)
```

[11]:
```
polje
```

[11]:
```
[[set(), set(), set(), set(), set()],
 [set(), set(), set(), set(), set()],
 [set(), set(), set(), set(), set()]]
```

[12]:
```
polje[2][1].add("A")

polje
```

[12]:
```
[[set(), set(), set(), set(), set()],
 [set(), set(), set(), set(), set()],
 [set(), {'A'}, set(), set(), set()]]
```

Ali s kakšno krajšo različico, kot je

[13]:
```
polje = [[set() for _ in range(sirina)] for _ in range(visina)]
```

Bistveno je, da `set()` ne pokličemo le enkrat, temveč tolikokrat, kolikor množic potrebujemo.

Rešitev je potem

[14]:
```
def situacija(specialci, marsovci, sirina, visina):
    polje = [[set() for _ in range(sirina)] for _ in range(visina)]
    for znak, (x, y) in zip("ABCDEFGHIJ", specialci):
        polje[y][x].add(znak)
    for znak, (x, y) in zip("abcdefghij", marsovci):
        polje[y][x].add(znak)
    return polje
```

Ne spreglejte, kako smo z `zip` poparili specialce in marsovce s črkami.

Precej študentov je namesto tega sestavljalo množice za vsako sobo posebej in potem iskalo, kdo spada vanjo:

[15]:
```
def situacija(specialci, marsovci, sirina, visina):
    polje = []
    for y in range(visina):
        vrstica = []
        for x in range(sirina):
            soba = set()
            for znak, (x0, y0) in zip("ABCDEFGHIJ", specialci):
                if x0 == x and y0 == y:
                    soba.add(znak)
```

```
            for znak, (x0, y0) in zip("abcdefghij", marsovci):
                if x0 == x and y0 == y:
                    soba.add(znak)
            vrstica.append(soba)
        polje.append(vrstica)
    return polje
```

To je nerodnejše, daljše in počasnejše. Če se ne spomnimo na `zip`, da vsak marsovec in specialec dobi pravo črko, pa je potrebno še veliko akrobacij, da določimo pravo črko za vsakega.

`znak` Ta funkcija ni nič posebnega; potrebujemo jo le, da bi naslednje stekle gladkeje.

[16]:
```
def znak(s):
    if not s:
        return "."
    if len(s) == 1:
        return next(iter(s))
    else:
        return str(len(s))
```

`izris` Tudi `izris` ni nič posebnega. Da ne telovadimo z nizi, pokažimo kar, kako se to naredi z generatorji in `join`-om:

[17]:
```
def izris(polozaj):
    return "\n".join("".join(znak(s) for s in vrstica) for vrstica in polozaj)
```

1.3 Za oceno 8

- `prvo_srecanje(specialec, marsovec)` prejme dve trojki z začetnima koordinatama in potjo specialca in marsovca. Vrniti mora prvo polje, na katerem se srečata. Če se ne srečata nikoli, vrne `None`.

  Klic `prvo_srecanje((2, 1, ">>"), (1, 1, ">>>>>>"))` vrne `(4, 1)`: specialec se premika pred marsovcev, potem pa ga pričaka v zasedi v `(4, 1)`.

1.3.1 Rešitev

Ni tako težko, če se je **lotimo sistematično**.

Najprej sestavimo seznam polj, po katerih bosta hodila marsovec in specialec. Potem gremo po parih istoležnih elementov in če sta polje kdaj enaki, vrnemo koordinati.

Nato je potrebno poskrbeti še za primer, ko sta poti različno dolgi. Če se prvi ustavi marsovec, se moramo vprašati, ali so njegove koordinate kje na ostanku specialčeve poti - preveriti moramo `korakis` od `len(korakim)` naprej. Če se prvi ustavi specialec, preverimo, ali bo marsovec naletel nanj na ostanku svoje poti. Katera pot je v resnici krajša, nam ni potrebno preverjati: ena od rezin bo pač vedno prazna.

[18]:
```
def prvo_srecanje(specialec, marsovec):
    korakis = koraki(specialec[0], specialec[1], specialec[2])
    korakim = koraki(marsovec[0], marsovec[1], marsovec[2])
    for koords, koordm in zip(korakis, korakim):
        if koords == koordm:
            return koords
    if koordm in korakis[len(korakim):]:
        return koordm
    if koords in korakim[len(korakis):]:
        return koords
    return None
```

Ker nam bo prišlo prav kasneje, omenimo, da bi lahko prvi dve vrstici napisali tudi tako:

```
korakis = koraki(*specialec)
korakim = koraki(*marsovec)
```

Zvezdica pred imenom pomeni, da želimo elemente tega seznama oziroma terke uporabiti kot argumente. Se pravi: kot argument ne podamo terke `specialec`, temveč elemente terke `specialec`.

Druga možnost je, da najprej rešimo nalogo za oceno 9 in v nalogi za oceno 8 le pokličemo to funkcijo.

[19]:
```
def prvo_srecanje(specialec, marsovec):
    return bingo([specialec], marsovec)
```

1.4 Za oceno 9

- `bingo(specialci, marsovec)` prejme podobne argumente kot prejšnja funkcija, le da namesto opisa enega specialca prejme opise več specialcev. Funkcija vrne koordinate polja, kjer bo nek specialec ujel marsovca, oziroma `None`, če se bo mali zeleni izmaknil.

  Klic

  ```
  bingo([(8, 12, ">>>>>>>>>"), (10, 14, ">>>>>>>>>"), (9, 13, ">>>>>>>>>")],
        (12, 16, "^>^>^>^>^>"))
  ```

  vrne `(13, 14)`: marsovec bo po treh korakih na `(13, 14)` in ravno takrat se bo tam znašel specialec, ki začne na `(10, 14)` in se pomika desno.

1.4.1 Rešitev

Tudi ta ni tako težka, če se je **lotimo res sistematično**. Recimo takole:

[20]:
```
def bingo(specialci, marsovec):
    korakiss = [koraki(*specialec) for specialec in specialci]
    korakim = koraki(*marsovec)

    for i, koordm in enumerate(korakim):
        for korakis in korakiss:
            if korakis[min(i, len(korakis) - 1)] == koordm:
```

```
                return koordm
    for korakis in korakiss:
        if koordm in korakis[i:]:
            return koordm
    return None
```

Najprej si pripravimo seznam seznamov, ki vsebuje vse korake specialcev; poimenovali smo ga `korakiss`, kot množino `korakis` (slongleščina). `korakiss` je torej seznam senzamov, kakršen je bil `korakis`.

Potem korakamo po marsovčevi poti in vemo tudi, na katerem koraku smo. Za vsak korak gremo čez vse `korakis`-e iz `korakiss` (se pravi čez vse specialce) in preverimo, če je specialec v tem, `i`-tem trenutku slučajno v isti sobi kot marsovec. Če, potem je ujet.

Pomemben trik tule je, kako odkrijemo specialčevo polje: `korakis[min(i, len(korakis) - 1)]`. Zanima nas, kje je specialec v trenutku `i`. Če je `i` manjši od dolžine specialčeve poti, nas zanima element s tem indeksom, sicer pa zadnji element, saj je specialec že zaključil pot in preži tam.

Če se ta zanke izteče neuspešno, je marsovec nekje obstal. Če bo kje ujet, bo ujet na tem polju. Gremo torej čez vse specialce in pogledamo, ali je v ostanku seznama katerega izmed specialcev (`korakis[i:]`, torej po `i`-tem koraku!) to marsovčevo polje. Če je, smo ga ujeli.

Če ne, se je izmuznil.

1.5 Ocena 10

Napiši funkcijo `simulacija(specialci, marsovci)`, ki prejme seznam poti specialcev (le nize, saj vsi specialci začnejo na `(0, 0)`!) in sezname s trojkami, ki opisujejo gibanje marsovcev. Vrniti mora `True`, če je reševalna akcija uspešna in `False`, če ni. Kako poteka akcija in kdaj je uspešna, je opisano v uvodu.

1.5.1 Rešitev

Da ta ne bi bila težka, pa se je moramo lotiti **res res res sistematično.**

Prejšnji dve funkciji sta nas naučili, da je potrebno sestaviti sezname korakov. Seznama `korakiss` in `korakims` bosta vsebovala vse poti vseh specialcev in marsovcev, ki so še aktivni. Ob ujetju marsovca bomo preprosto pobrisali pot specialca in marsovca iz seznama.

V zanki bomo izvedli toliko korakov, kolikor je dolga najdaljša pot kateregakoli. Šli bomo prek vseh specialcev in znotraj tega prek vseh marsovcev. Naletivši na dva z enakimi koordinatami bomo pobrisali pripadajoča elementa `korakiss` in `korakims`. Za to brisanje bomo potrebovali indekse obeh. Indeksa marsovca (v notranji zanki) bomo pridobili z `enumerate`. Specialci bodo zahtevali zanko `while` - bomo videli, zakaj.

[21]:
```
def simulacija(specialci, marsovci):
    korakiss = [koraki(0, 0, specialec) for specialec in specialci]
    korakims = [koraki(*marsovec) for marsovec in marsovci]
    for i in range(max(len(x) for x in korakiss + korakims)):
        speci = 0
        while speci < len(korakiss):
```

```
            korakis = korakiss[speci]
            for marsi, korakim in enumerate(korakims):
                if korakis[min(i, len(korakis) - 1)] == korakim[min(i,␣
↪len(korakim) - 1)]:
                    del korakims[marsi]
                    del korakiss[speci]
                    break
            else:
                speci += 1
    return not korakims and najveckrat_obiskana(marsovci) <= {korakis[-1] for␣
↪korakis in korakiss}
```

Brisanje marsovca je preprosto: `del korakims[marsi]`. S specialcem je podobno. Zanko prek marsovcev prekinemo z `break`. Znotraj ranks `for ... in korakims` preverjamo, ali bo specialec z indeksom `speci` ujel kakega marsovca. Če ga ujame, ne smemo pregledovati, ali bo morda še katerega drugega: ne bo, ker je že tega.

Navidez smo se ujeli v past: znotraj zanke `for` prek nekega seznama ne smemo brisati elementov tega seznama, saj bo zanka, kot smo že nekoč spoznali, v tem primeru preskakovala elemente. V resnici tu te nevarnosti ni, saj smo zanko po brisanju tako ali tako prekinili.

Pač pa moramo biti na to pozorni v zanki prek specialcev. To ne sme biti zanka `for`, ker bi po brisanju enega specialca zanka preskočila naslednjega. Zato tu uporabimo `while`: `speci` je indeks specialca. Zanka teče, dokler je indeks manjši od števila specialcev. Indeks se poveča le, če *ne pobrišemo* specialca - torej v primeru, da se zanka prek marsovcev ni končala z `break`.

Akcija je uspešna, če na koncu ni več nobenega aktivnega marsovca in če je množica največkrat obiskanih polj podmnožica zadnjih koordinat vseh specialcev, ki so še aktivni.

Ta rešitev je tako kratka po zaslugi nekaterih spretnosti. Najprej, prejšnje funkcije so nas naučile, da se splača delati sezname korakov. Odkrili smo trik, s katerim pridemo do `i`-tega elementa (indeksiranje z `min`). Predvsem pa je koristilo, da smo uporabili pravi zanki in se spomnili na `else` po zanki `for`. Brez teh trikov … bi se funkcija pač zapletla.