resitev

January 28, 2024

0.1 Arboretum

V (skoraj) vseh funkcijah bomo delali s seznami koordinat dreves v takšni obliki:

```
[('smreka', 4, 6), ('bukev', 1, 5), ('bukev', 8, 5), ('javor', -1, 4),
 ('bukev', 3, 4), ('javor', 7, 3), ('bukev', -2, 2), ('bukev', 2, 2),
 ('javor', 4, 2), ('smreka', -3, -1), ('bukev', 3, -1), ('javor', -1, -2),
 ('javor', 4, -3)]
```

Za vsako oceno je potrebno napisati tudi vse funkcije, zahtevane za nižjo oceno.

0.1.1 Ocena 6

Napiši naslednje funkcije.

- `razdalja(x1, y1, x2, y2)` prejme koordinati dveh točk in vrne (Evklidsko) razdaljo med njima.
- `najblizji(x, y, drevesa)` prejme koordinati točke in seznam dreves; vrne vrsto najbližjega drevesa, npr. `"javor"`.
- `najblizji_enak(x, y, vrsta, drevesa)` prejme koordinati točke in vrsto drevesa (npr. `"bukev"`) ter seznam dreves. Vrniti mora koordinato najbližjega drevesa te vrste (npr. najbližje bukve, če iščemo bukve). Če je enako oddaljenih več, sme vrniti koordinati kateregakoli od njih.
- `vse_vrste(drevesa)` vrne množico z imeni vseh vrst dreves, ki nastopajo v seznamu, na primer `{"oreh", "hrast", "bukev"}`.
- `koordinate_tipov(vrsta, drevesa)` prejme ime drevesne vrste (npr. `"smreka"`) in seznam dreves, vrniti pa mora množico koordinat dreves te vrste (npr. smreke).
- `najblizji_po_vrstah(x, y, drevesa)` koordinate neke poljubne točke in seznam dreves; vrniti mora slovar, katerega ključi so vse drevesne vrste v seznamu, pripadajoča vrednost pa koordinati tistega drevesa te vrste, ki je najbližji koordinatam, podanim kot argument. Če je enako oddaljenih več, sme spet vrniti koordinate kateregakoli od njih. Funkcija mora biti napisana tako, da hitro deluje tudi na dolgih seznamih z veliko različnimi vrstami.
- `najpogostejsa_vrsta(drevesa)` vrne ime najpogostejše drevesne vrste v podanem seznamu dreves. Če je enako pogostih vrst več, naj vrne poljubno od njih.

Rešitev Pri razdalji nimamo kaj: samo formula. Funkciji `najblizji` in `najblizji_enak` pa sta tudi nekaj, kar meljemo že od začetka - iskanje največjega elementa po določenem kriteriju.

[ ]:
```
def razdalja(x0, y0, x1, y1):
    return math.sqrt((x0 - x1) ** 2 + (y0 - y1) ** 2)
```

```
def najblizji(x, y, drevesa):
    naj_drevo = None
    for drevo, x0, y0 in drevesa:
        razd = razdalja(x, y, x0, y0)
        if naj_drevo == None or razd < naj_razd:
            naj_razd, naj_drevo = razd, drevo
    return naj_drevo


def najblizji_enak(x, y, tip, drevesa):
    naj_koord = None
    for drevo, x0, y0 in drevesa:
        razd = razdalja(x, y, x0, y0)
        if tip == drevo and (naj_koord == None or razd < naj_razd):
            naj_razd, naj_koord = razd, (x0, y0)
    return naj_koord
```

Preostale tri vrste so preprosta ponovitev zank in pogojev ter množic in slovarjev. Predvsem pa lepo vodijo v izpeljane slovarje in množice pri nalogi za oceno 8.

[1]:
```
def vse_vrste(drevesa):
    vrste = set()
    for drevo, x, y in drevesa:
        vrste.add(drevo)
    return vrste

def koordinate_tipov(tip, drevesa):
    koordinate = set()
    for drevo, x, y in drevesa:
        if drevo == tip:
            koordinate.add((x, y))
    return koordinate

def najblizji_po_vrstah(x, y, drevesa):
    najblizji = {}
    for vrsta in vse_vrste(drevesa):
        najblizji[vrsta] = najblizji_enak(x, y, vrsta, drevesa)
    return najblizji
```

Pri `koordinate_tipov` je potencialen kamen spotike dodajanje v množico: dodati želimo par `(x, y)`, torej ne moremo pisati `add(x, y)`, kar bi bilo videti, kot da dodajamo dve števili, namreč `x` in `y` in seveda ne deluje, temveč `add((x, y))`.

V zadnji funkciji smo uporabili dve izmed prejšnjih funkcij: funkcijo, ki vrne vse drevesne vrste in funkcijo, ki za vsako od njih poišče najbližji primerek.

Tole pravzaprav kar takoj prevedimo v rešitve v eni vrstici, saj so očitne.

[2]:
```
def vse_vrste(drevesa):
    return {drevo for drevo, x, y in drevesa}

def koordinate_tipov(tip, drevesa):
    return {(x, y) for drevo, x, y in drevesa if drevo == tip}

def najblizji_po_vrstah(x, y, drevesa):
    return {vrsta: najblizji_enak(x, y, vrsta, drevesa) for vrsta in␣
 ↪vse_vrste(drevesa)}
```

0.1.2 Ocena 7

Podatke o drevesih pravzaprav dobimo v datoteki. Videti je tako.

```
smreka: 4, 6
bukev: 1, 5; 3, 4
javor: -1, 4; 7, 3; -1, -2; 4, -3
bukev: 8, 5; 2, 2; -2, 2
javor: 4, 2
smreka: -3, -1
bukev: 3, -1
```

Vsaka vrstica se očitno začne z imenom drevesne vrrste, sledi dvopičje in potem koordinate enega ali več dreves te vrste. Drevesa so ločena s podpičji, koordinati z vejico, decimalk ni.

Napiši funkcijo `preberi(ime_datoteke)`, ki prejme ime datoteke s takšno vsebino, in vrne seznam, podoben gornjemu. Vrstni red elementov je lahko poljuben.

Rešitev Rešitev je samo eno dolgo zaporedje `split`-ov. Vsako vrstico razdelimo glede na `:` da dobimo drevesno vrsto in seznam koordinat. Slednjega razbijemo glede na `;`, da dobimo koordinate posameznih dreves. Koordinate razdelimo na x in y z razbitjem po `,`.

[ ]:
```
def preberi(ime_datoteke):
    drevesa = []
    for vrstica in open(ime_datoteke):
        vrsta, koordinate = vrstica.split(":")
        for koordinati in koordinate.split(";"):
            x, y = koordinati.split(",")
            drevesa.append((vrsta, float(x), float(y)))
    return drevesa
```

(Upam, da ne prehuda) vaja iz programiranja je tule, kje je zanka in kje ne. Z zanko gremo po vrsticah datoteke. V razbijanju po `:` ni zanke, saj tu vedno dobimo dve stvari: ime drevesne vrste in koordinate. Razbijanje po `;` da spisek koordinat, ta je lahko ena, lahko pa jih je sto (ali karkoli vmes ali čez), torej spet zanka. Končno, ko razbijamo po `,`, dobimo dve koordinati, torej to spet le shranimo v dve spremenljivki, brez zanke.

0.1.3 Ocena 8

Funkcije `vse_vrste`, `koordinate_tipov` in `najblizji_po_vrstah` naj bodo napisane v eni vrstici.

Rešitev Glej zgoraj.

0.1.4 Ocena 9

Napiši funkcije

- `dolzina_poti(x, y, vrste, drevesa)`, ki prejme neke začetne koordinate in seznam vrst dreves, ki bi jih radi obiskali, na primer `["bukev", "javor", "bukev", "smreka"]`, ter, kot vedno, seznam dreves. Funkcija mora izračunati dolžino poti od začetnih koordinat do (če sledimo gornjemu primeru) najbližje bukve, potem do javorja, ki je najbližji tej bukvi, potem do bukve, ki je najbližja temu javorju (in morda ni ista kot prejšnja bukev!) in potem do smreke, ki je najbližja tej bukvi. Če v katerem izmed korakov obstajata več enako oddaljenih najbližjih dreves, lahko gre k poljubnemu od njih.
- `najblizji_par_enakih(drevesa)` vrne razdaljo med najbližjim parom dreves iste vrste.
- `najmanjsi_krog(x, y, drevesa)` prejme koordinati in seznam dreves. Vrniti mora polmer najmanjšega kroga s središčem v teh koordinatah, ki vsebuje vse različne drevesne vrste.

Rešitev Za `dolzina_poti` potrebujemo spremenljivko, v katero seštevamo dolžino. V vsakem koraku poiščemo drevo, ki je najbližje danim koordinatam, prištejemo razdaljo k dolžini in si zapomnimo nove koordinate.

[ ]:
```
def dolzina_poti(x, y, vrstni_red, drevesa):
    skupno = 0
    for drevo in vrstni_red:
        x0, y0 = najblizji_enak(x, y, drevo, drevesa)
        skupno += razdalja(x, y, x0, y0)
        x, y = x0, y0
    return skupno
```

Za `najblizji_par_enakih` potrebujemo gnezdeno zanko. Če imamo opravka z dvema istovrstnima drevesoma, izračunamo razdaljo med njima; če je ta večja od 0 (torej: ne gre za isto drevo), a manjša od najmanjše (doslej), je to nova najmanjša razdalja (doslej).

[ ]:
```
def najblizji_par_enakih(drevesa):
    naj_razd = None
    for drevo1, x1, y1 in drevesa:
        for drevo2, x2, y2 in drevesa:
            if drevo1 != drevo2:
                continue
            razd = razdalja(x1, y1, x2, y2)
            if razd > 0 and (naj_razd == None or razd < naj_razd):
                naj_razd = razd
    return naj_razd
```

Za vsako vrsto moramo ugotoviti, katero je najbližje drevo te vrste. Krog mora pokriti vsa ta drevesa, torej mora biti njegov polmer enak največji izmed teh razdalij.

To ugotovivši lahko nalogo uženemo kar v eni vrstici. Pokličemo `najblizji_po_vrstah(x, y, drevesa)`; zanimajo nas le koordinate, torej pokličemo metodi `values()`, se pravi `najblizji_po_vrstah(x, y, drevesa).values()`. Tako dobimo koordinate vseh najbližjih dreves posameznih vrst. Izračunamo razdalje do njih: `razdalja(x, y, x1, y1) for x1, y1 in najblizji_po_vrstah(x, y, drevesa).values()`. Funkcija mora vrniti največjo med njimi.

[ ]:
```
def najmanjsi_krog(x, y, drevesa):
    return max(razdalja(x, y, x1, y1) for x1, y1 in najblizji_po_vrstah(x, y,␣
 ↪drevesa).values())
```

0.1.5 Ocena 10

Za oceno 10 ni posebnih nalog. Ocene 10 bodo dobili študenti, ki bodo zelo zgledno rešili vse naloge.