Izpit iz predmeta Programiranje 1, 6. februar 2013 ob 10.00. **Čas pisanja: 90 minut.**

Rešitve shranite **v eno samo datoteko s končnico .py** in jo oddajte prek Učilnice. Imena funkcij naj bodo takšna, kot jih predpisuje naloga. Rešitev preverite s testnimi primeri na Učilinici. Za rešitev naloge lahko dobite določeno število točk, tudi če ne prestane vseh testov. Funkcija, ki prestane vse teste, še ni nujno pravilna.

Normalno sestavljene rešitve prvih treh nalog imajo po 11 vrstic, zadnjih dveh pa 6. Čeprav ne zahtevamo takšne učinkovitosti, pa bomo posebej okorno napisane rešitve kaznovali z nekaj odbitimi točkami.

Dovoljena je vsa literatura na poljubnih medijih in ves material, ki je objavljen na Učilnici. Študenti s predolgimi vratovi in podobnimi hibami bodo morali (najmanj!) zapustiti izpit, katerega opravljanje se bo štelo kot neuspešno.

1. Tovornjaki

Fakulteta se seli v novo stavbo. Pred staro so v dolgi vrsti postavljene omare. Te so izjemno težke, zato problem ni prostor na tovornjakih, temveč njihova nosilnost. Vsak tovornjak lahko nosi dve toni (na srečo nobena omara ni težja od dveh ton). Omare nalagajo po vrsti. Na vsak tovornjak nakladajo, dokler smejo; ko bi bila naslednja omara pretežka, tovornjak odpelje in začnejo nakladati novega. Napišite funkcijo `tovornjaki(omare)`, ki dobi seznam tež omar (v kilogramih) in vrne potrebno število voženj.

Primeri

```
>>> tovornjaki([1500, 1000, 200, 900])
3    # ena na prvi tovornjak, dve na
drugi, zadnja na tretji
>>> tovornjaki([1500, 500])
1    # obe omari gresta na isti tovornjak
>>> tovornjaki([1500, 600])
2    # vsaka omara gre na svoj tovornjak
>>> tovornjaki([2000])
1    # to gre na enega
>>> tovornjaki([1000])
1    # tudi to
>>> tovornjaki([2000, 2000, 2000])
3    # trije, očitno
>>> tovornjaki([])
0    # ni omar, ni tovornjakov
```

2. Pajkova mreža

Na koordinatah `(1, 1)` stoji pajek. Ker razume Python, mu lahko podamo seznam premikov, opisanih s pari števil. Seznam `[(2, 4), (-1, 8)]` pomeni premik najprej za 2 desno in 4 gor, nato pa za 1 dol in 8 desno. Pajek za seboj vleče nit in jo po vsakem premiku pripne na tla. Ker je vraževeren, mu nikoli ne bomo dali takšnih navodil, da bi se po katerem od premikov znašel ravno na eni ali drugi kordinatni osi (ali obeh).

Primeri

```
>>> pajek([(-2, 0)])
1
>>> pajek([(2, 0), (-1, -2), (1, -1), (-5, 5)])
3
>>> pajek([(1, 5), (2, 1), (5, 1), (1, 3)])
0
>>> pajek([])
0
```

Napišite funkcijo `pajek(s)`, ki prejme seznam premikov in vrne, kolikokrat bo nit prečkala vodoravno in navpično koordinatno os.

Namig: ne da bi moral kaj risati, vem, da nit, ki je napeta med točkama (2, -5) in (-6, -8) prečka navpično os, nit med točkama (2, -5) in (-6, 8) pa obe. Bodite pozorni: seznam ne vsebuje koordinat točk, temveč premike.

3. Soimenjaki

V direktoriju imamo datoteke s podatki o študentih. Vsakemu ustreza ena datoteka, katere ime je enako vpisni številki, prva vrstica vsebuje ime in priimek, ostale pa druge podatke, ki nas ne zanimajo. V direktoriju ni nobenih drugih datotek.

Primeri

```
>>> soimenjaki("studenti1")
{'Janez Novak': ['63152568', '63155912'],
'Matija Kralj': ['63153650', '63153962',
'63154189']}
>>> soimenjaki("studenti2")
{'Janez Novak': ['63152730', '63154624']}
>>> soimenjaki("studenti3")
{}
```

Napišite funkcijo `soimenjaki(dir)`, ki dobi ime direktorija in poišče študente z enakimi imeni ter njihove vpisne številke. Rezultat vrne v obliki slovarja, katerega ključi so imena študentov, vrednosti pa seznami vpisnih številk študentov s takšnim imenom. V podanem direktoriju "studenti1", recimo, sta dva Janeza Novaka in trije Matiji Kralji, v "studenti2" dva Janeza Novaka, v "studenti3" pa ni nobenih ponovitev.

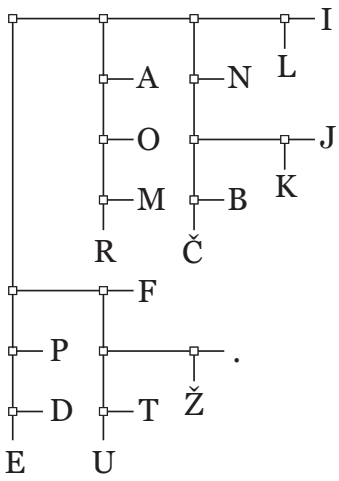
Namig: najprej sestavite slovar (lahko tudi `collections.defaultdict`) z vsemi študenti in seznami vpisnih številk. Ko končate prebiranje, prepišite v nov slovar vsa imena, pri katerih ste zabeležili več vpisnih številk.

4. Branje

V tej in naslednji nalogi se bomo ukvarjali z zemljevidi, kot jih kaže slika na desni. Iz vsake točke vodita poti na vzhod *in* na jug – ali pa nikamor. V točkah na koncu slepih ulic je zapisana črka.

Za sprehajanje po zemljevidu bomo uporabljali objekte razreda `Tocka`; razred je definiran v testnih primerih ter vam ga ni potrebno (in ga ne smete) definirati, temveč le uporabljati. Objekt vrste `Tocka` ima tri metode:

- `soseda(kam)`, ki vrne točko, ki je vzhodno ali južno od te; argument `kam` je lahko `"V"` ali `"J"`;
- `konec()`, ki pove ali je točka na koncu slepe ulice (`True` ali `False`);
- `znak()`, ki pove znak, zapisan v točki.



Če pokličemo `soseda(kam)` v točki, ki je že na koncu, dobimo napako. Če pokličemo `znak()`, ko še nismo na koncu, dobimo nekaj čudnega.

Če se bi začetna točka imenovala `t` in bi napisali

```
>>> t2 = t.soseda("J")
```

bi bila `t2` točka, ki je južno od izhodiščne točke `t`. Če nadaljujemo

```
>>> t3 = t2.soseda("V")
>>> t4 = t3.soseda("V")
```

smo prišli do točke, kjer je napisana črka F. Zdaj imamo

```
>>> t4.konec()
True
>>> t4.znak()
"F"
>>> t3.konec()
False
```

Vse to je torej že sprogramirano.

Vaša naloga je napisati dve funkciji. Prva naj se imenuje `crka(tocka, niz)`; prvi argument predstavlja levo zgornjo točko, drugi pa niz. Ta niz bo vedno dovolj dolg, da nas bo pripeljal do neke črke, navadno pa bo celo predolg. Funkcija naj vrne par (terko), ki vsebuje črko, do katere nas je pripeljal niz, in neuporabljeni del niza.

Druga funkcija, `preberi(tocka, niz)`, mora prebrati besedo, sestavljeno iz več črk. Bere jo tako, da začne levo zgoraj. Ko pride do konca poti, si zapomni dobljeno črko, s preostankom niza pa ponovno začne pri prvi točki. Pri tem mora za branje uporabljati prvo funkcijo. Na koncu naj vrne niz, ki vsebuje prebrane znake.

Primeri

```
>>> crka(t, "JVVVJJJJVVVV")
("F", "VJJJJVVVV")
   # pridemo do F, VJJJJVVVV je neuprabljeno
>>> crka(t, "VJJJJVVVV")
("R", "VVVV")
   # pridemo do I, VVVV ostane
>>> preberi(t, "JVVVJJJJVVVV")
"FRI" # JVV da F, VJJJJ da R, VVVV da I
```

5. Pisanje

Napišite funkcijo `huffman(tocka)`, ki kot argument prejme točko, kot v prejšnji nalogi, kot rezultat pa vrne slovar, katerega ključi so črke, vrednosti pa poti, ki vodijo do vsake črke. Za primer z gornje slike mora vrniti `{'A': 'VJV', 'F': 'JVV', 'B': 'VVJJJV', 'Č': 'VVJJJJ', 'D': 'JJJV', 'I': 'VVVV', 'K': 'VVJJVJ', 'J': 'VVJJVV', 'M': 'VJJJV', 'L': 'VVVJ', 'O': 'VJJV', 'N': 'VVJV', 'P': 'JJV', 'R': 'VJJJJ', 'U': 'JVJJJ', 'T': 'JVJJV', 'E': 'JJJJ', '.': 'JVJVV', 'Ž': 'JVJVJ'}`.

Funkcija naj deluje za poljubne zemljevide, ne le za zgornjega. Na koncih poti so lahko poljubni znaki, vključno z ヤポンスコ カタカノ！！

Namig: rešitev (skoraj) zahteva rekurzivno funkcijo. Če želite, ima lahko vaša funkcija tudi dodatne argumente, ki pa morajo imeti privzete vrednosti (na primer `huffman(tocka, pot="")`). Lahko pa si pomagate z dodatno funkcijo.