rešitev

January 28, 2024

1 Datotečni dnevnik

Nek računalnik hrani arhiv vseh sprememb datotek: vsakič, ko shranimo datoteko, zabeleži trenutni datum, ime datoteke in njeno dolžino. Podatki so lahko, recimo, takšni:

```
arhiv = ["10/14/2020 ime dat.avi 314236",
         "12/31/2018   ime dat.avi 21532",
         "10/16/2020 some other.avi 314236",
         "1/1/1970    file   with spaces.avi 42",
         "5/16/2020   ime dat.avi 21532",
         "10/18/2020 another file.avi 351352",
         "10/16/2020 ime dat.avi   312353",
         "10/18/2018 another file.avi 314236"]
```

- Na začetku vsake vrstice je datum shranjevanja (oz. spreminjanja) datoteke v obliki mesec/dan/leto. Ne pred njim ne znotraj njega ni presledkov.
- Sledi poljubno število presledkov.
- Ko se ti presledki končajo, imamo ime datoteke. To lahko vsebuje presledke. Celo zaporedne presledke. Pač pa se ime datoteke nikoli ne konča ali začne s presledkom; vsi presledki na začetku ali koncu so odveč.
- Na koncu vrstice je vedno številka ki predstavlja dolžino datoteke. Pred njo je lahko poljubno število presledkov, za njo jih gotovo ni.

Na disku, ki ga opisuje gornji seznam, so trenutno samo štiri datoteke ("ime dat.avi", "some other.avi", "file with spaces.avi" in "another file.avi"). Datoteka "ime dat.avi" je bila shranjena štirikrat, in sicer 14. oktobra 2020, 31. decembra 2018, 16. maja 2020 in 16. oktobra 2020. Tudi njena dolžina se je spreminjala. Trenutno je na disku le zadnja različica, z 16. oktobra letos.

Vrstice niso urejene po datumu ali čemerkoli drugem.

1.1 Testi

testi.py

1.2 Obvezna naloga

Prve štiri funkcije, ki jih moraš napisati, prejmejo ime vrstice, kot je gornja. Funkcije se lahko seveda kličejo med sabo. Katera kliče katero, je stvar tvoje odločitve; nalogo je mogoče rešiti na različne načine.

- `datum(vrstica)` vrne datum kot terko (leto, mesec, dan). Klic `datum("5/15/1970  file with spaces.avi  42")` vrne `(1970, 5, 15)`.
- `ime(vrstica)` vrne ime datoteke. Klic `ime("5/15/1970  file  with spaces.avi 42")` vrne niz `"file  with spaces.avi"`.
- `dolzina(vrstica)` vrne dolžino datoteke. Klic `datum("5/15/1970  file  with spaces.avi  42")` vrne 42.
- `podatki(vrstica)` vrne terko z gornjimi podatki. Klic `podatki("5/15/1970  file with spaces.avi  42")` vrne `((1970, 5, 15), file  with spaces.avi", 42)`.

Poleg teh napiši še naslednje štiri funkcije.

- `je_novejsa(s1, s2)` prejme dve vrstici in vrne `True`, če ima `s1` novejši (kasnejši) datum kot `s2`, in `False`, če ne. Klic `je_novejsa("11/16/2020 ime.txt 316", "11/15/2015 foo.txt 314")` vrne `True`.

- `najnovejsa(ime_datoteke, arhiv)` prejme ime datoteke in seznam, kot je na začetku naloge. Vrniti mora podatke o datoteki v času zadnje spremembe. Klic `najnovejsa("ime dat.avi", arhiv)` (pri čemer je `arhiv` takšen, kot je definiran zgoraj) vrne `((2020, 10, 16), "ime_dat.avi", 314236)`, saj so to podatki o datoteki, kot je bila shranjena na zadnjega izmed štirih datumov, ko smo spreminjali to datoteko.

- `datumi(ime_datoteke, arhiv)` vrne datume sprememb podane datoteke. Datumi morajo biti urejeni od kasnejših proti starejšim. Klic `datumi("ime dat.avi", arhiv)` vrne `[(2020, 10, 16), (2020, 10, 14), (2020, 5, 16), (2018, 12, 31)]`. Če datoteke s podanim imenom sploh ni, pač vrne prazen seznam.

- `odstrani(ime_datoteke, arhiv)` iz podanega seznama `arhiv` odstrani vse vrstice, ki se nanašajo na datoteko s podanim imenom. Klic `odstrani("ime dat.avi", arhiv)` spremeni gornji seznam v

  ```
  arhiv = ["10/16/2020 some other.avi 314236",
           "1/1/1970   file   with spaces.avi 42",
           "10/18/2020 another file.avi 351352",
           "10/18/2018 another file.avi 314236"]
  ```

  Funkcija spreminja podani seznam in ne vrača ničesar.

1.2.1 Rešitev

Najprejprostejša je dolžina, pa začnimo z njo. Vrstico razsekamo s `split()`, pograbimo zadnjo stvar, pretvorimo v `int` in vrnemo.

[1]:
```
def dolzina(s):
    return int(vrstica.split()[-1])
```

Z datumom je malo več dela. Pokličemo `vrstica.split()` in poberemo prvo stvar. To razkosamo še na s `split("/")`, pa dobimo tri stvari: mesec, dan in leto. Te vrnemo spremenjene v `int` in zložene v pravi vrstni red.

[2]:
```
def datum(vrstica):
    mesec, dan, leto = vrstica.split()[0].split("/")
    return int(leto), int(mesec), int(dan)
```

Zdaj pa ime. Ime se nahaja med prvim in zadnjim presledkom; indeks prvega dobimo z `index(" ")` in indeks zadnjega z `rindex(" ")`. To, kar je vmes, ima lahko sicer še kakšne odvečne presledke na začetku in koncu. Znebimo se jih s `strip()`.

[3]:
```
def ime(vrstica):
    return vrstica[vrstica.index(" "):vrstica.rindex(" ")].strip()
```

Zadnja funkcija, `podatki`, le kliče gornje tri funkcije.

[4]:
```
def podatki(s):
    return datum(s), ime(s), dolzina(s)
```

Seveda bi se dalo obrniti tudi drugače: lahko bi vse delo opravila zadnja funkcija, prve tri pa klicale njo. Pomembno je le, da si dela ne podvajamo in se te funkcije kličejo med seboj.

`je_novejsa` mora le primerjati datume. Terki sta že oblikovani tako, da imamo najprej leto, nato mesec in dan. Ko Python primerja terke, najprej primerja prvi element; če sta enaka, primerja drugega in če sta enaka tudi tadva, še tretjega. Točno to, kar potrebujemo.

[5]:
```
def je_novejsa(s1, s2):
    return datum(s1) > datum(s2)
```

Pri tej funkciji sicer ni bilo pričakovati hujših zapletov, vendar se je izkazala za eno najbolj zanimivih. Seveda so mnogi pisali

[6]:
```
def je_novejsa(s1, s2):
    if datum(s1) > datum(s2):
        return True
    else:
        return False
```

kar ni najbolj smiselno, vendar deluje.

Bolj zanimivo - in včasih usodno - je bilo, da se nekateri niso spomnili, da lahko primerjajo terke in so ročno primerjali leta, mesece in dneve.

[7]:
```
# To ne deluje pravilno!

def je_novejsa(s1, s2):
    leto1, mesec1, dan1 = datum(s1)
    leto2, mesec2, dan2 = datum(s2)

    if leto1 > leto2:
        return True
    if mesec1 > mesec2:
        return True
    if dan1 > dan2:
        return True
    return False
```

Ta funkcija ne deluje takrat, ko je `leto1` manjše od `leta2`. V tem primeru bi morala funkcija namreč takoj (recimo: takoj za prvim `if`-om vrniti `False`, ne pa, da potem primerja mesece in dneve.

Druga variacija istega je

[8]:
```
# To ne deluje pravilno!

def je_novejsa(s1, s2):
    leto1, mesec1, dan1 = datum(s1)
    leto2, mesec2, dan2 = datum(s2)

    if leto1 < leto2:
        return False
    if mesec1 < mesec2:
        return False
    if dan1 < dan2:
        return False
    return True
```

Tudi to ne deluje, samo na ravno obraten način. Če je `leto1` večje kot `leto2`, funkcija nima kaj primerjati mesecev.

Pravilno je

[9]:
```
# To je brez potrebe zapleteno

def je_novejsa(s1, s2):
    leto1, mesec1, dan1 = datum(s1)
    leto2, mesec2, dan2 = datum(s2)

    if leto1 < leto2:
        return False
    if leto1 == leto2 and mesec1 < mesec2:
        return False
    if leto1 == leto2 and mesec1 == mesec2 and dan1 < dan2:
        return False
    return True
```

Funkcija `je_novejsa` pa je nekaj, kar prežvekujemo že ves čas: iskanje največjega elementa po določenem kriteriju - tokrat datumu.

[10]:
```
def najnovejsa(ime, arhiv):
    naj_pod = None
    naj_dat = None
    for vrstica in arhiv:
        pod = dat, ime2, _2 = podatki(vrstica)
        if ime == ime2 and (naj_dat == None or dat > naj_dat):
            naj_pod = pod
```

```
            naj_dat = dat
    return naj_pod
```

Seznam datumov je preprosto dobiti: pripravimo prazen seznam, gremo čez arhiv in za vrstice, ki se nanašajo na podano datoteko, dodamo datum v seznam. Na koncu ga vrnemo padajoče urejenega.

[11]:
```
def datumi(ime_datoteke, arhiv):
    dats = []
    for vrstica in arhiv:
        dat, ime, _ = podatki(vrstica)
        if ime == ime_datoteke:
            dats.append(dat)
    return sorted(dats, reverse=True)
```

Zadnjo vrstico lahko zamenjamo tudi z

```
    dats = sorted(dats, reverse=True)
    return dats
```

ali

```
    dats.sort(reverse=True)
    return dats
```

ne pa z

```
    sorted(dats, reverse=True)
    return dats
```

ali

```
    dats = dats.sort(reverse=True)
    return dats
```

Kot sem opozoril na predavanjih, funkcija `sorted(s)` vrne nov seznam, starega pa pusti pri miru - zato zadnje ne deluje, saj `sorted(dats, reverse=True)` ne uredi seznama, temveč le vrne urejenega - rezultat pa vržemo stran. Nasprotno od tega pa metoda `sort` uredi seznam in ne vrne ničesar. Zato v zadnjem primeru priredimo seznamu `dats` `None` in funkcija torej vrne `None`.

Funkcija `odstrani` zahteva, da znamo brisati seznam, prek katerega gremo z zanko. To je najlažje narediti z zanko `while`.

[12]:
```
def odstrani(ime_datoteke, arhiv):
    i = 0
    while i < len(arhiv):
        if ime(arhiv[i]) == ime_datoteke:
            del arhiv[i]
        else:
            i += 1
```

Kot smo videli na predavanjih: v vsakem koraku bodisi pobrišemo element bodisi povečamo `i`. Če pobrišemo element namreč ne smemo povečati `i` (indeksa elementa, ki ga opazujemo), ker se nam bo en element v tem primeru "izmaknil".

Nekateri so po pričakovanjih poskušali

[13]:
```
def odstrani(ime_datoteke, arhiv):
    for vrstica in arhiv:
        if ime(vrstica) == ime_datoteke:
            arhiv.remove(vrstica)
```

To ne dela zaradi "izpodmikanja": `for` zanko koraka prek seznama. Najprej pogleda ničti element, nato prvega, nato drugega… Če pobrišemo, recimo, ničti element, se prvi pomakne na mesto ničtega, zanka pa vseeno napreduje na prvega (ki je bil prej drugi). Na ta način se nam element, ki je bil prvotno prvi, izmakne.

Drugi so po pričakovanju poskusili tole.

[14]:
```
def odstrani(ime_datoteke, arhiv):
    t = []
    for vrstica in arhiv:
        if ime(vrstica) != ime_datoteke:
            t.append(vrstica)
    arhiv = t
```

Ideja je sicer lepa: v nov seznam zberemo vse vrstice, ki morajo ostati in potem rečemo, naj se ime `arhiv` nanaša na ta, nov seznam. Vendar to ne deluje, ker smo s tem spremenili le lokalno spremenljivko, ne pa objekta (seznama), ki ga je funkcija dobila kot argument. Več o tem se bomo učili čez nekaj tednov.

Spet, po pričakovanjih so nekateri nagooglali tole.

[15]:
```
def odstrani(ime_datoteke, arhiv):
    t = []
    for vrstica in arhiv:
        if ime(vrstica) != ime_datoteke:
            t.append(vrstica)
    arhiv[:] = t
```

To pa deluje, vendar … če to rešitev uporabi nekdo, ki ne razume, kaj je naredil, se s tem ni ničesar naučil.

1.3 Dodatna naloga

Napiši funkcijo `skupna_dolzina(arhiv)`, ki vrne skupno dolžino vseh datotek v arhivu. Klic

```
skupna_dolzina([
    "10/14/2020 ime dat.avi   1",
    "5/16/2020   ime dat.avi 2",
    "10/16/2020 ime dat.avi 4",
    "12/31/2018   ime dat.avi 8",
    "10/16/2020 some other.avi 16",
    "10/18/2020 another file.avi 32",
```

```
    "10/18/2018 another file.avi 64",
]
```

vrne `52` (4 + 16 + 32). Upoštevati mora, očitno, le zadnje različice datotek.

1.3.1 Rešitev

Možnih variant je seveda veliko. Ena je tale.

[16]:
```
def skupna_dolzina(arhiv):
    obdelano = []
    dolzina = 0
    for vrstica in arhiv:
        dat, im, dol = podatki(vrstica)
        if im not in obdelano:
            dolzina += najnovejsa(im, arhiv)[-1]
            obdelano.append(im)
    return dolzina
```

Gremo čez vse vrstice. Za vsako ime, ki ga vidimo, poiščemo najnovejšo datoteko, prištejemo njeno dolžino in si zapišemo, da smo to ime že obdelali, tako da ga bomo naslednjič preskočili.