resitev

January 28, 2024

Imamo seznam tež nekih paketov, ki jih je potrebno odnesti, recimo `[5, 3, 8, 1, 2, 3, 5, 4, 2, 4]`. Vsi nosači so enako močni; znano je, koliko si more vsak naložiti. Recimo `9`.

Prišel bo prvi nosač; vzel bo paketa s težo 5 in 3, preskočil paket s težo 8 (5 + 3 + 8 bi bilo preveč), vzel paket s težo 1 in potem preskočil vse ostale, saj že nosi 9.

Drugi nosač vzame paket s težo 8 … in potem nobenega več, ker bi vsak dodatni paket presegel dovoljeno težo 9.

Tretji nosač pobere pakete s težami 2, 3 in 4; 5, ki je bil vmes, preskoči.

Četrti nosač vzame paketa 5 in 2.

Peti nosač vzame paket s težo 4.

Za drug primer vzemimo pakete `[5, 3, 4, 6]`, nosilnost nosačev je spet 9. Čeprav je skupna teža vseh paketov 18, potrebujemo tri nosače - ker smo nerodni pri obremenjevanju. Prvi pobere paketa 5 in 3, zato ne more pobrati nobenega več. Drugi pobere 4. Tretji pobere 6. Če bi bili pametnejši, bi prvi resda lahko pobral 5 in 4, drugi pa 3 in 6, vendar pač ne delimo po takšnem postopku.

0.0.1 Obvezna naloga

Napiši funkcijo `nosaci(paketi, nosilnost)`, ki prejme seznam tež paketov in največjo možno obremenitev nosača. Funkcija vrne potrebno število nosačev - za gornja primera je to 5 oziroma 3.

0.0.2 Dodatna naloga

Napiši funkcijo `razporedi(paketi, nosilnost)`, ki vrne seznam seznamov paketov, ki jih nosijo nosači. Za prvi primer vrne `[[5, 3, 1], [8], [2, 3, 4], [5, 2], [4]]`, za drugega pa `[[5, 3], [4], [6]]`.

0.0.3 Nasvet

Eden od načinov reševanja naloge je, da si predstavljaš, da vsak nosač bodisi vzame paket, bodisi ga da v nek drug seznam, recimo mu `kup`. Ko konča izbiranje, rečemo, da so `paketi` tisto, kar je ostalo na kupu.

Drug način, ki bo za vas morda težji, je, da iz seznama izbrišete (recimo z `del`) paket, ki ga nosač vzame. Če se naloge lotite na ta način in ne bo delalo, priporočam, da stalno izpisujete teže paketov, za katere se nosač odloča, ali bi ga vzel ali ne. Na ta način vam bo jasno, zakaj funkcija ne deluje, kot bi morala, in jo boste morda znali popraviti.

0.1 Rešitev

Nalogo bomo rešili na oba načina, ki ju nasvetuje nasvet. Najprej naredimo z ločenim kupom.

[6]:
```
def nosaci(paketi, nosilnost):
    nosacev = 0
    while paketi:
        nosacev += 1
        zavrnjeni = []
        nosi = 0
        for paket in paketi:
            if nosi + paket <= nosilnost:
                nosi += paket
            else:
                zavrnjeni.append(paket)
        paketi = zavrnjeni
    return nosacev
```

Šteli bomo nosače (`nosacev = 0`). Vsak krog zanke `while` bo ustrezal nalaganju enega nosača, torej povečamo število nosačev. Pripravimo kup, na katerega bomo nalagali pakete, ki jih bo nosač zavrnil (`zavrnjeni = []`) in spremenljivko, v kateri bo pisalo, koliko bremena že nosi (`nosi = 0`).

Potem gremo z zanko `for` čez pakete. Za vsakega preverimo, ali mu ga smemo dodati (`nosi + paket <= nosilnost`) in ga, če je tako, dodamo (`nosi += paket`). Sicer ga prestavimo na kup zavrnjeni.

Končavši zanko `for`, razglasimo zavrnjene za nove pakete, ki jih bo pregledoval naslednji nosač (`paketi = zavrnjeni`).

Deluje? Mora.

[7]:
```
nosaci([5, 3, 8, 1, 2, 3, 5, 4, 2, 4], 9)
```

[7]: 5

[8]:
```
nosaci([5, 3, 4, 6], 9)
```

[8]: 3

Drugi način je z brisanjem iz seznama.

[15]:
```
def nosaci(paketi, nosilnost):
    nosacev = 0
    paketi = paketi.copy()
    while paketi:
        nosacev += 1
        nosi = 0
        i = 0
        while i < len(paketi):
            if nosi + paketi[i] <= nosilnost:
```

```
                nosi += paketi.pop(i)
            else:
                i += 1
    return nosacev
```

Seznam paketov na začetku skopiramo, saj bomo iz njega brisali in spreminjanje seznama, ki smo ga dobili kot argument, je (običajno) prepovedano.

Notranjo zanko nadomestimo z `while`. Spremenljivka `i` bo indeks, ki bo šel čez pakete. Če paket vzamemo, v `nosi` prištejemo njegovo težo in ga pobrišemo. Tu pride prav metoda `pop(i)`, ki istočasno vrne vrednost `i`-tega elementa in ga odstrani. V tem primeru `i`-ja ne povečamo, saj bo na `i`-to mesto prišel drug paket. Če nosač paketa ne vzame, pa povečamo `i`.

[16]:
```
nosaci([5, 3, 8, 1, 2, 3, 5, 4, 2, 4], 9)
```

[16]: 5

[17]:
```
nosaci([5, 3, 4, 6], 9)
```

[17]: 3

Nekaterim je bilo bolj všeč, da so namesto teže, ki jo trenutno nosi nosač, beležili težo, ki si jo še sme naložiti. (Tehnično gledano je to celo hitrejše, v praksi bo prihranek minimalen.) V tem primeru namesto `nosi = 0` pišemo `zmore = nosilnost`, namesto `nosi + paketi[i] <= nosilnost` imamo `paketi[i] <= zmore` in namesto `nosti += paketi.pop(i)` imamo `zmore -= paketi.pop(i)`. Prihranili smo eno seštevanje. Prav. V bistvu lepo.

Druga izboljšava je bila najprej rešiti dodatno nalogo in potem preprosto vrniti število podseznamov seznama, ki ga vrne ta, dodatna funkcija.

0.1.1 Rešitev dodatne naloge

Dodatna naloga ni bila bistveno težja od obvezne. Mogoče jo je bilo celo lažje reševati, saj je rezultat zgovornejši in boljše vidimo, kaj se dogaja.

Rešujemo jo lahko na oba gornja načina. Če je bila rešitev obvezne naloge takšna:

```
def nosaci(paketi, nosilnost):
    nosacev = 0
    while paketi:
        nosacev += 1
        zavrnjeni = []
        nosi = 0
        for paket in paketi:
            if nosi + paket <= nosilnost:
                nosi += paket
            else:
                zavrnjeni.append(paket)
        paketi = zavrnjeni
    return nosacev
```

je rešitev dodatne le malenkost drugačna.

[21]:
```
def razporedi(paketi, nosilnost):
    razpored = []
    while paketi:
        zavrnjeni = []
        nosi = []
        for paket in paketi:
            if sum(nosi) + paket <= nosilnost:
                nosi.append(paket)
            else:
                zavrnjeni.append(paket)
        paketi = zavrnjeni
        razpored.append(nosi)
    return razpored
```

Namesto da bi šteli nosače in torej začeli z `nosacev = 0`, imamo `razpored = []`. Spremenljivka `nosi` ni več skupna teža paketov temveč seznam njih. V pogoju imamo zato `sum(nosi)` in ko paket naložimo, ne prištejemo teže, temveč z `append` dodamo paket. In po zanki `for` v seznam `razpored` dodamo (pod)seznam s paketi tega nosača.

[22]:
```
razporedi([5, 3, 8, 1, 2, 3, 5, 4, 2, 4], 9)
```

[22]:
```
[[5, 3, 1], [8], [2, 3, 4], [5, 2], [4]]
```

[23]:
```
razporedi([5, 3, 4, 6], 9)
```

[23]:
```
[[5, 3], [4], [6]]
```

Če gremo po drugi poti, z brisanjem, so spremembe podobne.