

## Day 01: Calorie Counting

Naloge imajo zgodbice. Tule bomo vedno vključili povezavo na izvirno nalogo, namesto opisa pa nalogo *kvečjemu* povzeli.

### Prvi del

V vsaki vrstici datoteke je zapisano celo število. Vrstice so ločene v skupine; med dvema skupinama je prazna vrstica. Sešteti moramo števila v vsaki skupini in na koncu izpisati največjo tako dobljeno vsoto.

#### Preprosta rešitev

Z `open("input.txt").read()` preberemo datoteko v niz. Razcepimo jo glede na prazne vrstice, `split("\n\n")`. Potem se ukvarjamo z vsako skupino posebej: seštejemo in primerjamo vsoto z največjo doslej.

To je nekaj, kar bi znali vsi študenti, ki so poslušali mesec in pol Programiranja 1.

```
naj = 0
for skupina in open("input.txt").read().split("\n\n"):
    vsota = 0
    for vrstica in skupina.splitlines():
        vsota += int(vrstica)
    if vsota > naj:
        naj = vsota
print(naj)
67633
```

#### Izpeljani seznamami - oziroma generatorji

Seštevati se da tudi veliko hitreje. Za začetek: če imamo neko skupino, recimo

```
skupina = """1500
300
200
400
"""
```

lahko preprosto sestavimo seznam vrstic, pretvorjenih v številke.

```
[int(vrstica) for vrstica in skupina.splitlines()]
[1500, 300, 200, 400]
```

Vsota skupine je pač vsota tega seznama,

```
sum([int(vrstica) for vrstica in skupina.splitlines()])
```

2400

Namesto seznama lahko funkciji `sum` damo tudi kar generator - se pravi, izpustimo oglate oklepaje.

Tako dobimo precej krajšo rešitev.

```
naj = 0
for skupina in open("input.txt").read().split("\n\n"):
    vsota = sum(int(vrstica) for vrstica in skupina.splitlines())
    if vsota > naj:
        naj = vsota
print(naj)
```

67633

Zdaj pa pomislimo, da iščemo `max` vsot. Vse skupaj lahko torej zapakiramo v en sam `max`, ki bo šel prek vrstic in računal ... no, pač maksimum vsot.

```
print(
    max(sum(int(vrstica) for vrstica in skupina.splitlines())
        for skupina in open("input.txt").read().split("\n\n")
    )
)
```

67633

### Branje po vrsticah

Zdaj pa naredimo malo drugače, bolj, recimo, C-jevske: brali bomo vrstico za vrstico in seštevali. Ko naletimo na prazno vrstico, vemo, da je konec skupine. Primerjamo vsoto in če je večja od največje doslej, si jo zapomnimo.

```
naj = 0
vsota = 0
for vrstica in open("input.txt"):
    if vrstica.strip(): # vrstica vsebuje vsaj \n, zato strip
        vsota += int(vrstica)
    else:
        if vsota > naj:
            naj = vsota
        vsota = 0
if vsota > naj:
    naj = vsota

print(naj)
```

67633

Pogoj `if vrstica.strip()` preveri, ali je vrstica (po tem, ko odstranimo `\n`) neprazna. Če je, prištejemo številko k trenutni vsoti, sicer postorimo, kar je

potrebno postoriti, ko je konec skupine.

Tisto zoprno preverjanje po zanki je potrebno za primer, da bi imela največjo vsoto ravno zadnja skupina, ki ji ne sledi več prazna vrstica.

Prednost te rešitve je, da nikoli ne preberemo celotne datoteke v pomnilnik. Morda pa niti ne gre za datoteko: takole bi lahko dobivali, na primer, neke podatke z nekega spletnega strežnika, vsako minuto bi prišel nov podatek in mi bi morali v vsakem trenutku vedeti največjo vsoto *doslej*. V takem primeru je možna samo tale rešitev.

### Generator skupin

Naredimo še nekaj zanimivega: napišimo funkcijo, ki kot argument dobi ime datoteke in nato vrača seznane, ki vsebujejo vse številke določene skupine. Ker bo zanimivo.

```
def skupine(ime):
    skupina = []
    for vrstica in open(ime):
        if vrstica.strip():
            skupina.append(int(vrstica))
        else:
            yield skupina
            skupina = []
    if skupina:
        yield skupina
```

Tale funkcija nekoliko spominja na prejšnjo rešitev. Razlika je le v tem, da je prejšnja takrat, ko je naletela na prazno vrstico, izračunala in primerjala vsoto, tale pa vrne (točneje: generira) seznane števil v skupinah.

Uporabimo jo lahko tako:

```
naj = 0
for skupina in skupine("input.txt"):
    vsota = sum(skupina)
    if vsota > naj:
        naj = vsota
print(naj)
```

67633

Ali pa tako.

```
print(max(sum(skupina) for skupina in skupine("input.txt")))
```

67633

Oziroma, podobno ampak še boljše, tako.

```
print(max(map(sum, skupine("input.txt"))))
```

67633

Če bi generator namesto seznamov vračal kar vsote,

```
def skupine(ime):
    skupina = []
    for vrstica in open(ime):
        if vrstica.strip():
            skupina.append(int(vrstica))
        else:
            yield sum(skupina)
            skupina = []
    if skupina:
        yield sum(skupina)
```

pa bi lahko pisali celo kar

```
print(max(skupine("input.txt")))
```

67633

`skupine("input.txt")` namreč generira vsote in `max` bo vrnil največjo zgenerirano stvar.

## Drugi del

Drugi del naloge hoče, da poiščemo največje tri vsote in jih seštejemo.

To najpreprosteje naredimo tako, da zložimo vse vsote v seznam, ga uredimo in seštejemo največje tri.

Rešitve bodo podobne prejšnjim, zato pokažimo le prvo in zadnjo. Vse ostale so pač med njima. :)

```
vsote = []
for skupina in open("input.txt").read().split("\n\n"):
    vsota = 0
    for vrstica in skupina.splitlines():
        vsota += int(vrstica)
    vsote.append(vsota)
print(sum(sorted(vsote)[-3:]))
```

199628

`sorted` vrne urejen seznam. Vzamemo elemente od predpredzadnjega (-3) do konca in seštevamo.

Pa še po vzoru zadnje rešitve:

```
print(sum(sorted(skupine("input.txt"))[-3:]))
```

199628

`sorted(skupine("input.txt"))` uredi vse, kar zgenerira `skupine`. In potem spet vzamemo zadnje tri in seštejemo.

### Seznam treh

Tule preberemo in urejamo vse. Urejanje je v splošnem počasna operacija. Nas zanimajo samo prvi trije.

Lahko naredimo tako: najprej pripravimo seznam vsot. Lahko tako, kot v preprostejši rešitvi drugega dela (se pravi, obdržimo vse, kar je v zanki), lahko pa uporabimo generator iz drugega dela in napišemo kar

```
vsote = list(skupine("input.txt"))
```

Kakorkoli že, `vsote` zdaj vsebuje tri vsote. Nato pobremo tri največje.

```
vsota = 0
for _ in range(3):
    naj = max(vsote)
    vsota += naj
    vsote.remove(naj)
print(vsota)
```

199628

Tole gre šestkrat čez celoten seznam - enkrat zaradi `max`, enkrat zaradi `remove`. Je to boljše, hitrejše od urejanja? Odvisno od tega, koliko skupin imamo. Ker sodobni jeziki uporabljajo hitre algoritme urejanja, bo do nekaj sto skupin hitreje, če urejamo. Od ondod do neskočnosti pa je hitreje takole. (Teorija: čas izvajanja gornjega programa je sorazmeren številu skupin - označimo ga z  $n$ . Za dvakrat več skupin porabi dvakrat več časa. Čas urejanja pa je (običajno, v praksi) sorazmeren  $n \log n$ , torej narašča hitreje kot linearno, zato je od določenega števila skupin naprej gornji program hitrejši.)

Če se želimo izogniti večkratnim prehodom čez vsot, ohranjujemo največje tri.

```
vsote = list(skupine("input.txt"))
```

```
naj3 = []
for vsota in vsote:
    naj3 = sorted(naj3 + [vsota])[-3:]
print(sum(naj3))
```

199628

Tule je `naj3` seznam, ki vsebuje največje tri vsote (oziroma kakšno manj, v začetku). V vsakem koraku dodamo novo vsoto, ga uredimo in obdržimo le večje tri.

Je to hitrejše? V teoriji je, saj `sorted` vedno kličemo le na seznamih s štirimi (ali, v začetku, celo manj) elementi. Torej čas izvajanja `sorted` ni odvisen od

števíla skupin.

V praksi? Dvomim. In: komu mar. :)

Tole bi se dalo še poljubno zaplesti, tako da bi namesto seznama uporabljali kopico. Ampak pustimo.

## Kotlin

Še enkrat pogledjmo rešitev v eni vrstici.

```
print(
    max(sum(int(vrstica) for vrstica in skupina.splitlines())
        for skupina in open("input.txt").read().split("\n\n")
    )
)

67633
```

Čeprav ji lahko priznamo učinkovitost, moramo potožiti, da ni ravno berljiva. Problem je v tem, da jo je potrebno brati ritensko. Preberimo kodo, od leve proti desni. Ta funkcija nekaj izpiše. In sicer maksimum. Maksimum česa? Vsot. Vsot česa? `int`-ov. `int`-ov iz česa? Vrstic skupin. Kakšnih skupin, takšnih, ki jih dobimo, če preberemo datoteko in dobljeni niz razcepimo po `\n\n`.

Najprej se zgodi tisto, kar je napisano nazadnje, in potem se stvari dogajajo, no, od desne proti levi. V resnici razcepimo niz po `\n\n`, nato to razbijemo na vrstice, za vsako vrstico pokličemo `int`, nato zberemo vsote teh `int`ov in izračunamo njihov maksimum. Če poskušate slediti temu stavku po kodi, boste videli, da smo zdaj prebrali kodo ritensko.

Python ima zelo lepo sintakso generatorjev ter izpeljanih seznamov, množic in slovarjev, ko to zlagamo v večje strukture, pa gre vse v napačno smer.

Poglejmo isti program v Kotlinu.

```
import java.io.File

val vsote = File("input.txt")
    .readText()
    .trim()
    .split("\n\n")
    .map {
        it.split("\n")
        .map { it.toInt() }
        .sum()
    }
```

```
println(vsote.maxOrNull()!!)
println(vsote.sorted().takeLast(3).sum())
```

```
import java.io.File
```

```
val chunks = File("input.txt")
    .readText()
    .trim()
    .split("\n\n")
    .map {
        it.split("\n")
        .sumOf { it.toInt() }
    }
```

```
println(chunks.maxOrNull()!!)
println(chunks.sorted().takeLast(3).sum())
```

Vem, da Kotlina (večinoma) ne znamo, ampak koda je tako lepa, da jo lahko vseeno kar preberemo. Predvsem bomo videli, kako različna je od Pythonove.

Odpremo datoteko. Preberemo besedilo. `trim()` je isto kot Pythonov `split`; tule ga potrebujemo, da odbijemo `\n` na koncu datoteke, ki bi sicer povzročal težave kasneje. Dobljeno reč razbujemo glede na `\n\n`. Tako dobimo seznam skupin. Na vsaki skupini naredimo naslednje: razbijemo jo glede na `\n`. Tako dobimo seznam vrstic. Na vsakem elementu pokličemo `toInt`. Dobimo seznam števil. Ta seštejemo.

Tako pridemo do seznama vsot. Sledita `println`, ki izpišeta rešitev prvega in drugega dela naloge.

Malo bolj lepo bi se sicer napisalo tako:

```
import java.io.File
```

```
val chunks = File("input.txt")
    .readText()
    .trim()
    .split("\n\n")
    .map {
        it.lines()
        .sumOf { it.toInt() }
    }
```

```
println(chunks.maxOrNull()!!)
println(chunks.sorted().takeLast(3).sum())
```

In najbrž gre še lepše, ampak jaz nisem ravno veliko poznavalec Kotlina. In

sploh je tule še nekaj tehničnih detajlov, ki jih ne razumemo, ker pač ne znamo Kotlina. (Predvsem: tisto v zavutih oklepajih so lambda-funkcije, ki jih podamo kot argument funkcijama `map` in `sumOf`; in če ima le-ta le en argument, ki ga ne navedemo eksplicitno, mu je ime `it`).

Vendar to ni pomembno, pomembno je, da vidimo, kako to teče lepše od Pythona. Čeprav je ideja programa natančno ista, teče zelo lepo od začetka proti koncu, od zgoraj navzdol, tako kot ga beremo.

Kotlin ni niti približno edini jezik, ki omogoča takšno pisanje programov (če ne drugega, je v tem kar močan tudi JavaScript), je pa eden močnejših. In, za moj okus, lep.