

## Day 22: Crab Combat

(Povezava na nalogo)

V prvem delu igramo vojno s kartami, pri čemer ima vsaka karta drugačno številko.

V drugem delu igra postane rekurzivna: zmagovalne karte ne določimo glede na njuni števili temveč glede na to, kdo zmaga rekurzivno igro. Če ima en igralec številko 7 in drugi 5, bo prvi vzel naslednjih 7 kart in drugi naslednjih 5. Z njima bosta odigrala igro vojne in karti 7 in 5 dobi tisti igralec, ki zmaga v tej igri. Pri tem igrata s "kopijama kupčkov" - po igri za 5 in 7 ostane tistih naslednjih 7 in 5 kart normalno v igri. Če kateri od igralcev nima dovolj kart za rekurzivno igro (v gornjem primeru, če prvi nima vsaj 7 in drugi vsaj 5 preostalih kart), se karti 7 in 5 primerjata kot v običajni igri (zmaga tisti, ki ima karto z višjo številko).

Da se igra ne bi zaciklala, je potrebno paziti, ali se je točno enaka situacija že kdaj pojavila. V tem primeru takoj zmaga igralec 1. Enakost situacij primerjamo le znotraj posamične igre oz. podigre itd. Igralec 1 zmaga le "rekurzivno" igro in ne vsega skupaj.

Naloga ima kup zoprnih detajlov; če se zataknejo obnje, se splača pogledati primere v izvirnem opisu naloge.

### Branje podatkov

Podatki so v datoteki takšne oblike.

Player 1:

9  
2  
6  
3  
1

Player 2:

5  
8  
4  
7  
10

Razdelimo glede na "\n\n" in poberemo oba dela v p1 in p2. Vsakega razkosamo po 'splitline

```
p1, p2 = open("input.txt").read().split("\n\n")
c1, c2 = ([int(x) for x in p.splitlines()[1:]] for p in (p1, p2))
```

## Prvi del: Vojna kot vojna

Tole smo imeli enkrat celo za domačo nalogo pri Programiranju 1, tako preprosto je. Dokler imata oba igralca kaj kart, pobremo prvi karti iz obeh kupčkov (`pop(0)`) in ju dodamo na konec zmagovalčevega kupčka v vrstnem redu, ki ga predpisuje naloga.

```
while c1 and c2:
    a1, a2 = c1.pop(0), c2.pop(0)
    if a2 > a1:
        c2 += [a2, a1]
    else:
        c1 += [a1, a2]
```

Tu se na vso moč spodobi povedati: `pop(0)` je počasen, ker mora vse elemente seznama prestaviti za eno mesto nazaj. Podatkovni strukturi, pri kateri je učinkovito tako dodajanje kot pobiranje, tako na koncu kot na začetku, pravimo *double ended queue* ali *deque* (izgovori: "dek"). Z njim bi vojno izvedli tako.

```
from collections import deque
```

```
c1, c2 = (deque(int(x) for x in p.splitlines()[1:]) for p in (p1, p2))
```

```
while c1 and c2:
    a1, a2 = c1.popleft(), c2.popleft()
    if a2 > a1:
        c2.extend([a2, a1])
    else:
        c1.extend([a1, a2])
```

Glede na našo velikost problema to niti ni pomembno, predvsem pa z deque ne moremo udobno delati v drugem delu naloge.

Naloga zahteva, da pomnožimo številko na zadnji zmagovalčevi karti z 1, številko na predzadnji z 2 in tako naprej ter izračunamo vsoto.

Tu nima smisla gledati, kdo je zmagovalec: enostavno zložimo vse karte skupaj, `c1 + c2`. Eden od teh dveh kupčkov je prazen, drugi zmagovalčev. To obrnemo (`reversed(c1 + c2)`), in oštevilčimo od 1 naprej `enumerate(reversed(c1 + c2), start=1)`. Čez to spustimo zanko, množimo in seštevamo.

```
print(sum(i * x for i, x in enumerate(reversed(c1 + c2), start=1)))
34255
```

Na en zanimiv način je privlačna tudi spodnja različica (ki sicer uvozi par funkcij in tako naprej, ampak ... je zanimiva). Vzamemo obrnjen kupček in vsa števila od 1 naprej. Čez te pare z `map` spustimo množenje (`mul`) in rezultat seštejemo.

```
from operator import mul
from itertools import count
```

```
print(sum(map(mul, reversed(c1 + c2), count(1))))  
34255
```

To nima nobenih prednosti pred gornjim, nemara je celo počasnejše. Je pa zanimivo.

## Drugi del: rekurzivna vojna

Naloga ni težka, le veliko priložnosti za napake nudi.

Rekurzivno funkcijo bomo napisali tako, da vrne dve stvari: pove, ali je zmagal igralec številka 2, poleg tega pa vrne zmagovalčev kupček kart. Kdo je zmagal, potrebujemo za rekurzivne vojne, kupček kart pa za prvi klic. Da bo funkcija le ena, pa bomo pač vračali oboje, uporabili pa, kar potrebujemo.

V množici **seen** bomo hranili vse že videne situacije.

V zanki najprej sestavimo situacijo kot terko terk **c1** in **c2**. Pri tem se **c1** in **c2** skopirata; to je potrebno zato, ker ju spodaj spreminjamo.

Če je situacija že videna, javimo, da ni zmagal drugi igralec.

Sicer pa dodamo situacijo med videne. Poberemo karti. Če imata oba igralca dovolj preostalih kart, zmagovalca določi rekurzivna igra na začetkih kupčkov, sicer pa številki na kartah.

Na koncu povemo, ali je zmagal drugi igralec (to je, ali ima neprazen kupček kart) in vrnemo kupček zmagovalčevih kart, ki ga dobimo tako, da seštejemo (prazni) poražencev in zmagovalčev kupček.

```
def game(c1, c2):  
    seen = set()  
  
    while c1 and c2:  
        situation = (tuple(c1), tuple(c2))  
        if situation in seen:  
            return False, c1  
  
        seen.add(situation)  
        a1, a2 = c1.pop(0), c2.pop(0)  
        if len(c1) >= a1 and len(c2) >= a2:  
            winner, _ = game(c1[:a1], c2[:a2])  
        else:  
            winner = a2 > a1  
        if winner:  
            c2 += [a2, a1]  
        else:  
            c1 += [a1, a2]
```

```
    return bool(c2), c1 + c2

c1, c2 = ([int(x) for x in p.splitlines()[1:]] for p in (p1, p2))
_, cards = game(c1, c2)
print(sum(i * x for i, x in enumerate(reversed(cards), start=1)))

33369
```