resitev

January 28, 2024

1 Day 13: Shuttle Search

(Povezana na nalogo)

1.1 Branje podatkov

Vemo, koliko je ura (v minutah od bogvekdaj) in kateri avtobusi so na razpolago.

```
939
7,13,x,x,59,x,31,19
```

To pravi, da je ura 939, vozijo avtobusi 7, 13, 59, 31 in 19. Vsi avtobusi so odpeljali s postaje ob času 0, nato pa 7 odpelje vsakih 7 minut, 13 vsakih 13 minut, 59 vsakih 59 minut. Ostanek razumete.

Tisti `x` bodo postali pomembni v prvem delu naloge, zato jih bomo razložili takrat. Za zdaj povejmo le, da potrebujemo številke avtobusov in mesta, na katerih se pojavijo, torej `[(7, 0), (13, 1), (59, 4), (31, 6), (19, 7)]`.

[1]:
```
f = open("example.txt")
start = int(f.readline())
buses = [(int(x), i) for i, x in enumerate(f.readline().split(",")) if x != "x"]

buses
```

[1]:
```
[(7, 0), (13, 1), (59, 4), (31, 6), (19, 7)]
```

1.2 Prvi del: prvi večkratnik

Zanima nas, kateri avtobus bo odpeljal naslednji.

Najprej za vsak avtobus izpišimo, kdaj odpelje naslednjič. Za vsak avtobus bi lahko izračunali, za koliko smo ga zamudili (`start % bus`); naslednji bo odpeljal čez `start + (bus - start % bus)` minut … razen, če je `start % bus` ravno 0, se pravi, če avtobus odpelje *zdaj*.

Da ne kompliciramo, se raje lotimo tako: `(start + bus - 1) // bus * bus`: k trenutnemu času prišetejemo `bus - 1`. V času od `start` do (vključno) `start + bus - 1` bo peljal natančno en avtobus. Če ta čas celoštevilsko delimo z `bus` in pomnožimo nazaj z `bus`, bomo dobili prvi večkratnik `bus`-a po `start`.

[2]:
```
for bus, _ in buses:
    print(bus, (start + bus - 1) // bus * bus)
```

```
7 945
13 949
59 944
31 961
19 950
```

Izmed teh je potrebno poiskati tistega, ki odpelje prvi in izračunati produkt njegove številke in časa čakanja.

Prav. Gornje pare bomo obrnili, tako da bo spredaj čas odhoda. Iz teh parov poiščemo minimum, in iz para razberemo številko in čas odhoda.

[3]:
```
departure, bus = min(((start + bus - 1) // bus * bus, bus) for bus, _ in buses)
print(departure, bus, (departure - start) * bus)
```

```
944 59 295
```

1.3 Drugi del: Kitajski ostanki

Zdaj poglejmo celo spremenljivko `buses`.

[4]:
```
buses
```

[4]:
```
[(7, 0), (13, 1), (59, 4), (31, 6), (19, 7)]
```

Zanima nas, za kateri čas velja, da bo avtobus 7 odpeljal točno ob tem času, 13 odpelje eno minuto kasneje, 59 odpelje 4 minute kasneje in tako naprej. Čas, ki ga iščemo, je precej velik, tako da kakšno iskanje po vrsti ne pride v poštev.

Najprej obrnimo: 7 ujememo, 13 smo zamudili za 13 - 1 = 12 minut, 59 smo zamudili za 59 - 4 = 55 minut in tako naprej.

To pomeni: iščemo tak čas `t`, da je ostanek po deljenju `t` s 7 enak 0, ostanek po deljenju `t` s 13 je enak 12, ostanek po deljenju `t` z 59 je enak 55 in tako naprej.

V splošnem, imamo seznam števil n_i in a_i, in naloga je najti x, za katerega velja, da je ostanek po deljenju `x` z vsakim `ni` enak `ai`. Iščemo rešitev problema kitajskih ostankov. Stvar je popolnoma matematična: sprogramirati je potrebno razširjeni Evklidov algoritem in ga uporabljati na parih števil.

[5]:
```
def egcd(a, b):
    sp, tp = 1, 0
    s, t = 0, 1
    while b:
        k = a // b
        a, b = b, a % b
        sp, s = s, sp - k * s
        tp, t = t, tp - k * t
```

```
    return a, sp, tp

def chinese(conds):
    a1, n1 = 0, 1
    for n2, a2 in conds:
        _, m1, m2 = egcd(n1, n2)
        a1 = a1 * m2 * n2 + a2 * m1 * n1
        n1 *= n2
        a1 %= n1
    return a1

f = open("input.txt")
start = int(f.readline())
buses = [(int(bus), int(bus) - delay) for delay, bus in enumerate(f.readline().
 ↪split(",")) if bus != "x"]
print(chinese(buses))
```

```
626670513163231
```