

Zadanie Duże liczby – LOGIA 20 (2019/20), etap 3

Treść zadania

Ola bawi się liczbami – zarówno małymi jak i takimi, które mają kilkaset cyfr. Zapisuje wybraną liczbę na kartce i szuka największej liczby całkowitej, której kwadrat nie przekracza zapisanej liczby. Szybko znalazła algorytm dający poprawny wynik. Teraz zastanawia się, jak szukać takich liczb, które podniesione do innej jednocyfrowej potęgi nie przekraczają wybranej liczby. Pomóż Oli uporać się z problemem. Napisz funkcję **liczba(a, n)**. Pierwszy parametr **a** jest liczbą złożoną maksymalnie z 1000 cyfr, drugi **n** to liczba z zakresu od 2 do 7 oznaczająca potęgę, do której podnosimy poszukiwaną wartość. Postaraj się, by na wynik nie trzeba było czekać zbyt długo.

Przykłady:

Wynikiem liczba(1024, 2) jest 32, ponieważ $32^2 = 1024$

Wynikiem **liczba(899, 2)** jest 29, ponieważ $29^2 = 841 < 899$, a $30^2 = 900 > 899$.

Wynikiem **liczba(11000, 3)** jest 22, ponieważ $22^3 = 10648 < 11000$, a $23^3 = 12167 > 11000$.

Wynikiem liczba(90972061672647417382949994702882855264900000, 2) jest

9537927535510396054172.

Omówienie rozwiązania

Pierwsze podejście

Można próbować wyznaczać wynik, korzystając ze wzoru:

Niestety, otrzymany wynik po zaokrągleniu w dół do najbliższej liczby całkowitej da poprawny wynik dla niewielu liczb. Wraz ze wzrostem liczby cyfr wynik jest obarczony coraz większym błędem. Na rezultat ma też wpływ wynik operacji 1 / n (zestawienie w tabeli poniżej).

| a | n | a ^(1/n) | int(a ** (1 / n)) | poprawny wynik |
|--|---|-----------------------|------------------------|------------------------|
| 1024 | 2 | 32.0 | 32 | 32 |
| 11000 | 3 | 22.23980090569315 | 22 | 22 |
| 117649 | 3 | 48.9999999999999 | 48 | 49 |
| 90972061672647417382949994702882855264900000 | 2 | 9.537927535510396e+21 | 9537927535510396338176 | 9537927535510396054172 |

Ponadto powyższe operacje są możliwe tylko dla liczb składających się z maksymalnie 308 cyfr – dla większych pojawia się błąd.





Drugie podejście

Można zacząć od 0 i sprawdzać kolejne liczby. Taki algorytm zapiszemy w następujący sposób:

```
liczba = 0
dopóki liczba<sup>n</sup> <= a
    liczba = liczba + 1
wynik liczba - 1</pre>
```

Niestety taka strategia zawodzi – dla liczb składających się z trzynastu cyfr na wynik trzeba czekać około sekundy. W tabeli poniżej znajdują się informacje dotyczące czasu działania algorytmu.

| liczba | czas |
|------------------|---------------------|
| 1012 | 0.6 s |
| 10 ¹⁶ | 53 s (ok. 1 minuta) |
| 10 ³⁰ | ponad 1 rok |

Trzecie podejście

Można wykorzystać wyszukiwanie binarne, gdzie w każdym kroku odrzucamy połowę kandydatów na poszukiwaną liczbę.

| liczba | maksymalna liczba kroków prowadzących do uzyskania wyniku |
|--------------------|--|
| 10 ¹⁰ | 34 |
| 10 ¹⁰⁰ | 333 |
| 10 ¹⁰⁰⁰ | 2322 |

Algorytm korzystający z wyszukiwania binarnego:

Ta strategia prowadzi do sukcesu wystarczająco szybko. Dla osób dociekliwych polecamy sprawdzenie, jaki wpływ na czas działania programu ma operacja potęgowania dla dużych liczb (srodekⁿ).





Rozwiązanie w języku Python

Funkcja realizująca opisany algorytm, zapisana w języku Python może wyglądać tak:

```
1. def liczba(a, n):
2. if a == 1:
3.
            return 1
4.
        p = 1
5.
        k = a
       while p < k:
7.
            s = (p + k) // 2
            if s ** n <= a:
8.
9.
               p = s + 1
10.
            else:
11.
12.
        return p - 1
```

Testy

Pierwsze dwie grupy testów przeznaczone są dla rozwiązania iteracyjnego, w którym sprawdzane są kolejne liczby począwszy od 0. Zastosowanie wzoru skutkuje błędnym wynikiem, przy czym rezultat różni się co najwyżej o 1 (wyjątek stanowi pierwszy test). Trzecia grupa testów przeznaczona jest dla rozwiązań, które liczą wynik ze wzoru, a następnie iteracyjnie sprawdzają kolejno liczby mniejsze (pierwszy test z grupy III) i większe (drugi test z grupy III). Czwarta i piąta grupa testów przeznaczona jest dla rozwiązań korzystających z metody połowienia. Tym razem zastosowanie strategii skutecznej w przypadku trzeciej grupy nie daje poprawnego wyniku w skończonym czasie (czwarta grupa) lub generuje błąd (piąta grupa).

| Grupa testów | Test | Wynik |
|-----------------|---|------------------------------------|
| I | liczba(121104, 2) | 348 |
| | liczba(117649, 3) | 49 |
| | liczba(97952412517413330078126000000000 | 192500 |
| п | 00000, 7) | |
| " | liczba(3542463423148271506515549004969, | 123467 |
| | 6) | |
| III | liczba(8769999999999000000000000000000000000000 | 189483487950327 |
| | 000000000000000000000000000000000000000 | |
| | 000000000000000000000000000000000000000 | |
| | liczba(90972061672647417382949994702882 | 301615751698493720069441 |
| | 855264900000123, 2) | |
| | liczba(999999999999999999999999999999999999 | 999999999999999999999999999999 |
| IV | 9999999999999997037037036999999999 | 9999999999990123456789999999999999 |
| | 99999999999999999999999999999 | 99999999999999999999 |
| | 173369913123000000000000000000000000000000000 | |
| | 0000004962507597306504390891481839000 | |
| | 000000000000000000007361682663008687 | |
| | 699999999999999999999999999999 | |
| | 370369999999999999999999999999999999 | |
| | 00000000, 3) | |





| | I' - L - /00000000000000000000000000000000 | 000000000000000000000000000000000000000 |
|---|---|---|
| | liczba(80999999999999999999999999999999999999 | 899999999999999999999999999999999999999 |
| | 9999999998623000000000000000000000 | 9999992349999999999999999999999999999 |
| | 00000000000000000585224999983800000 | 9999999999999999999 |
| | 000000000000000000000000000000000000000 | |
| | 234000000000000000000000000000000000000 | |
| | 0000000000000000081, 2) | |
| | liczba(999999999999999999999999999999999999 | 999999999999999999999999999999999999999 |
| | 9999999999999999999999999999999 | 9999999999999999999999999999999999999 |
| | 9999999999999999999999999999999 | 9999999999999999999999999999999999999 |
| | 9999999999999999999999999999 | 9999999999999999999999999867967 |
| | 3983925026106885059440165021960937482 | 8500522137701188803300439218749641437 |
| | 0718978686198816140645567378955261091 | 957372397632281 |
| | 4801711467821924680028662658248384063 | |
| | 4708272274800934100875272068773301480 | |
| | 8703291181720446260597851966010742864 | |
| | 5681360258985972138325135462213860926 | |
| | 0154959538722448198841635399619402412 | |
| | 5681044248770472166311451859484212842 | |
| | 6809154061244637526330629550641413157 | |
| | 4294758610537585111152182603591873559 | |
| | 0226001594379445564679697904666504347 | |
| | 8911544945969593984710853056985106035 | |
| | 1126058092522820164162691070501614305 | |
| | 1489098354377201822400562770793104852 | |
| | 1100078668157464973040376891429456789 | |
| | 0709370295899065123847164477093330453 | |
| | 6450115551456795324840815416628886645 | |
| V | 2394279215590196587752330646656088822 | |
| | 1727034913458412909110228406373672321 | |
| | 9803024539748275909301363449012433491 | |
| | 7268737941727770231509666494131171358 | |
| | 8698554580231613616473443368255184958 | |
| | 4202131906386246576436558601424438321 | |
| | 428608, 5) | |
| | liczba(999999999999999999999999999999999999 | 99999999999999999999999999999999 |
| | 9999999999999999999999999999999999 | 9999999999999999999999999999999 |
| | 9999999999999999999999999999999 | 99999999999999999999999999999 |
| | 999999999999999999999999999999 | 999999999999999999999999999999 |
| | 9999999999999999999745980305452990 | 99999999999999993649507636324759432 |
| | 3772907195055793504211810361217618761 | 2679876394837605295259030440469042281 |
| | 6912765618489580605630781336189080800 | 9140462239515140769533404727 |
| | 3134342165585177093963432045109495457 | 31,0702233313170703333704727 |
| | 9605359332378630795330321025017114751 | |
| | 3875120512529050992802057781085407236 | |
| | 6647190448070701596965590544263302880 | |
| | 9776113489924328599500892398206783700 | |
| | 4128762842068803891272831467856350912 | |
| | | |
| | 7613647024479807568036486334478781270 | |





000000, 4)

