

Algoritmy

Poznámky k téme

Algoritmus je presný postup, ktorý v konečnom počte krokov vyrieši úlohu. V tejto téme prejdeme klasické algoritmy, ktoré by mal poznať každý programátor: hľadanie najmenšieho a najväčšieho prvku, vyhľadávanie, triedenie a základné číselné algoritmy. Python má na mnohé z nich hotové funkcie (`max`, `sort`, `in`) – no pochopiť, ako fungujú, je dôležité: rovnaké myšlienky použiješ v zložitejších úlohách, kde hotová funkcia nestačí.

Čo sa naučíš

- nájsť minimum a maximum (aj ich pozíciu) bez vstavaných funkcií,
- vyhľadávať prvok lineárne aj binárne a vedieť, kedy ktoré použiť,
- triediť bublinkovým triedením a triedením výberom,
- programovať číselné algoritmy: NSD, prvočísla, delitele, prevody sústav.

1 Čo je algoritmus

Algoritmus musí byť:

- **jednoznačný** – v každom kroku je jasné, čo sa má urobiť,
- **konečný** – po konečnom počte krokov skončí,
- **všeobecný** – rieši celú skupinu úloh (napríklad súčet *ľubovoľných* čísel, nie len $3 + 5$),
- **rezultatívny** – dáva výsledok.

Algoritmus sa dá zapísať slovami, **pseudokód**om (zjednodušeným zápisom podobným programu) alebo vývojovým diagramom. Program je algoritmus zapísaný v programovacom jazyku.

Pri porovnávaní algoritmov nás zaujíma aj **efektívnosť** – koľko krokov potrebujú pre veľké vstupy. Pre zoznam s miliónom prvkov je obrovský rozdiel, či algoritmus potrebuje 20, milión alebo milión krokov.

2 Minimum a maximum

Myšlienka: prvý prvok vyhlásime za doteraz najväčší. Potom prechádzame ostatné prvky; ak je niektorý väčší, stane sa novým maximom.

```
def maximum(z):
    naj = z[0]
    for x in z[1:]:
        if x > naj:
            naj = x
    return naj

print(maximum([7, 3, 12, 5, 12, 1]))
print(maximum([-8, -3, -5]))
```

Výstup:

```
12
-3
```

Pozor: Nezačínať s `naj = 0`. Pre zoznam záporných čísel by výsledok bol 0, hoci 0 v zozname vôbec nie je. Vždy začni prvým prvkom zoznamu.

2.1 Pozícia maxima

Často nás zaujíma, **kde** je maximum. Pamätáme si index namiesto hodnoty:

```
def index_maxima(z):
    naj_i = 0
    for i in range(1, len(z)):
        if z[i] > z[naj_i]:
            naj_i = i
    return naj_i

z = [7, 3, 12, 5, 12, 1]
i = index_maxima(z)
print("maximum", z[i], "na indexe", i)
```

Výstup:

```
maximum 12 na indexe 2
```

Pri rovnakých hodnotách nájde `>` **prvý** výskyt maxima; `>=` by našiel posledný.

2.2 Všetky výskyty minima

```
z = [4, 1, 7, 1, 9, 1]
najmensie = min(z)
pozicie = []
for i in range(len(z)):
    if z[i] == najmensie:
        pozicie.append(i)
print("minimum:", najmensie, "počet:", len(pozicie), "pozície:", pozicie)
```

Výstup:

```
minimum: 1 počet: 3 pozície: [1, 3, 5]
```

3 Lineárne vyhľadávanie

Úloha: zistiť, či (a kde) sa hodnota nachádza v zozname. **Lineárne vyhľadávanie** prechádza prvky jeden po druhom, kým hodnotu nenájde. Ak prejde celý zoznam bez úspechu, hodnota v ňom nie je – zvykne sa vrátiť `-1`.

```
def najdi(z, hodnota):
    for i in range(len(z)):
        if z[i] == hodnota:
            return i
    return -1

z = [8, 3, 15, 4, 15]
print(najdi(z, 15))
print(najdi(z, 7))
```

Výstup:

```
2
-1
```

`return i` ukončí funkciu pri prvom nájdení. Až **po** cykle (keď sme prešli všetko) vraciame `-1`.

V najhoršom prípade (hodnota je na konci alebo chýba) treba porovnať **všetkých n prvkov**. Pre zoznam s miliónom prvkov je to milión porovnaní. Vstavane `hodnota in z` a `z.index(hodnota)` robia to isté – iba rýchlejšie, lebo sú naprogramované v jazyku C.

4 Triedenie

Triedenie usporiada prvky podľa veľkosti. V praxi použiješ `z.sort()` alebo `sorted(z)`, ale klasické algoritmy triedenia sú výborným tréningom práce s indexmi.

4.1 Bublínkové triedenie

Myšlienka: prechádzame zoznam a porovnávame **susedné** dvojice. Ak sú v zlom poradí, vymeníme ich. Po prvom prechode „vypublie“ najväčší prvok na koniec, po druhom druhý najväčší na predposledné miesto atď.

```
z = [5, 1, 4, 2, 8]
for i in range(len(z) - 1):
    if z[i] > z[i + 1]:
        z[i], z[i + 1] = z[i + 1], z[i]
print("po 1. prechode:", z)
```

Výstup:

```
po 1. prechode: [1, 4, 2, 5, 8]
```

Prvý prechod podrobne:

porovnanie	stav pred	výmena?	stav po
5 a 1	[5, 1, 4, 2, 8]	áno	[1, 5, 4, 2, 8]
5 a 4	[1, 5, 4, 2, 8]	áno	[1, 4, 5, 2, 8]
5 a 2	[1, 4, 5, 2, 8]	áno	[1, 4, 2, 5, 8]
5 a 8	[1, 4, 2, 5, 8]	nie	[1, 4, 2, 5, 8]

Celé triedenie opakuje prechody. Po každom prechode je na konci o jeden zoradený prvok viac, takže ďalší prechod môže byť o jedno porovnanie kratší. Ak počas prechodu nenastala žiadna výmena, zoznam je zoradený a môžeme skončiť:

```
def bublinkove(z):
    n = len(z)
    for koniec in range(n - 1, 0, -1):
        vymena = False
        for i in range(koniec):
            if z[i] > z[i + 1]:
                z[i], z[i + 1] = z[i + 1], z[i]
                vymena = True
        if not vymena:
            break

cisla = [5, 1, 4, 2, 8, 3]
bublinkove(cisla)
print(cisla)
```

Výstup:

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

Funkcia nič nevracia – zoraduje priamo zoznam, ktorý dostala (zoznam je meniteľný).

4.2 Triedenie výberom

Myšlienka: nájdeme najmenší prvok a vymeníme ho s prvým. Potom nájdeme najmenší zo zvyšku a vymeníme ho s druhým atď.

```
def vyberom(z):
    for i in range(len(z) - 1):
        naj = i
        for j in range(i + 1, len(z)):
            if z[j] < z[naj]:
                naj = j
        z[i], z[naj] = z[naj], z[i]

cisla = [29, 10, 14, 37, 13]
vyberom(cisla)
print(cisla)
```

Výstup:

```
[10, 13, 14, 29, 37]
```

4.3 Ako rýchle je triedenie

Bublínkové triedenie aj triedenie výberom porovnávajú v najhoršom prípade približne $n^2/2$ dvojíc. Pre 1 000 prvkov je to asi 500 000 porovnaní, pre milión prvkov 500 miliárd – to by trvalo veľmi dlho. Vstavané `sort` používa šikovnejší algoritmus, ktorý potrebuje rádovo $n \cdot \log_2 n$ porovnaní (pre milión prvkov asi 20 miliónov) – preto ho v skutočných programoch používaj.

```
>>> mena = ["Sofia", "ema", "Ján", "Adam"]
>>> sorted(mena)
['Adam', 'Ján', 'Sofia', 'ema']
>>> sorted(mena, key=str.lower)
['Adam', 'ema', 'Ján', 'Sofia']
>>> sorted(mena, key=len)
['ema', 'Ján', 'Adam', 'Sofia']
```

Parameter `key` určuje, podľa čoho sa triedi: `str.lower` triedi bez ohľadu na veľké písmená, `len` podľa dĺžky.

5 Binárne vyhľadávanie

Ak je zoznam **zoradený**, dá sa hľadať oveľa rýchlejšie. Poznáš to z hry „Myslím si číslo od 1 do 100“: najlepšie je tipnúť si 50 a podľa odpovede „viac/menej“ zahodiť polovicu možností.

Myšlienka: pozrieme sa na **stredný** prvok. Ak je to hľadaná hodnota, máme ju. Ak je hľadaná hodnota menšia, môže byť iba v ľavej polovici, inak iba v pravej. Hľadanie opakujeme v polovici, ktorá ostala.

```
def binarne(z, hodnota):
    lavy, pravy = 0, len(z) - 1
    while lavy <= pravy:
        stred = (lavy + pravy) // 2
        if z[stred] == hodnota:
            return stred
        if z[stred] < hodnota:
            lavy = stred + 1
        else:
            pravy = stred - 1
    return -1

z = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
print(binarne(z, 23))
print(binarne(z, 7))
```

Výstup:

```
5
-1
```

Priebeh hľadania hodnoty 23:

krok	lavy	pravy	stred	z[stred]	rozhodnutie
1	0	9	4	16	16 < 23 → hľadáme vpravo
2	5	9	7	56	56 > 23 → hľadáme vľavo
3	5	6	5	23	nájdené na indexe 5

5.1 Prečo je binárne vyhľadávanie také rýchle

Každý krok zmenší oblasť hľadania na polovicu. Pre n prvkov treba najviac približne $\log_2 n$ krokov:

počet prvkov	lineárne (najhorší prípad)	binárne (najviac)
10	10	4
1 000	1 000	10
1 000 000	1 000 000	20

Pozor: Binárne vyhľadávanie funguje **iba v zoradenom** zozname. V nezoradenom zozname by zahadzovalo polovicu, v ktorej hľadaná hodnota môže byť.

6 Číselné algoritmy

6.1 Najväčší spoločný deliteľ (Euklidov algoritmus)

Najväčší spoločný deliteľ (NSD) dvoch čísel sa dá hľadať skúšaním všetkých deliteľov, ale Euklidov algoritmus je oveľa rýchlejší. Využíva, že $\text{NSD}(a, b) = \text{NSD}(b, a \% b)$ a $\text{NSD}(a, 0) = a$:

```
def nsd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

def nsn(a, b):
    return a * b // nsd(a, b)

print(nsd(48, 18), nsn(4, 6))
```

Výstup:

6 12

a	b	a % b
48	18	12
18	12	6
12	6	0
6	0	výsledok je 6

Najmenší spoločný násobok (NSN) dostaneme zo vzťahu $\text{NSN}(a, b) = a \cdot b / \text{NSD}(a, b)$. Python má na NSD aj hotovú funkciu `math.gcd`.

6.2 Prvočísla

Číslo $n > 1$ je prvočíslo, ak nemá iného deliteľa ako 1 a n . Netreba skúšať všetky delitele do n : ak má n deliteľa väčšieho ako $\sqrt{n}$, musí mať aj deliteľa menšieho ako $\sqrt{n}$. Stačí preto skúšať delitele d , pre ktoré platí $d * d \leq n$:

```
def je_prvocislo(n):
    if n < 2:
        return False
    d = 2
    while d * d <= n:
        if n % d == 0:
            return False
        d += 1
    return True

print([n for n in range(50) if je_prvocislo(n)])
print(je_prvocislo(1_000_003))
```

Výstup:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
True
```

Pre číslo okolo milióna stačí vyskúšať asi 1 000 deliteľov namiesto milióna. (Podčiarkovníky v 1_000_003 iba sprehľadujú zápis čísla.)

Eratostenovo sito nájde všetky prvočísla do n naraz: vypíšeme čísla 2 až n a postupne preškrtavame násobky každého ešte nepreškrtnutého čísla.

```
n = 50
je_prvo = [True] * (n + 1)
je_prvo[0] = je_prvo[1] = False
for i in range(2, n + 1):
    if je_prvo[i]:
        for nasobok in range(i * i, n + 1, i):
            je_prvo[nasobok] = False
print([i for i in range(n + 1) if je_prvo[i]])
```

Výstup:

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

6.3 Delitele a dokonalé čísla

```
def delitele(n):
    return [d for d in range(1, n + 1) if n % d == 0]

print(delitele(28))
for n in range(2, 1000):
    if sum(delitele(n)) - n == n:
        print(n, "je dokonalé číslo")
```

Výstup:

```
[1, 2, 4, 7, 14, 28]
6 je dokonalé číslo
28 je dokonalé číslo
496 je dokonalé číslo
```

Dokonalé číslo sa rovná súčtu svojich deliteľov menších ako ono samo: $28 = 1 + 2 + 4 + 7 + 14$. Ďalšie dokonalé číslo je až 8128.

6.4 Prevody medzi číselnými sústavami

Z desiatkovej do dvojkovej sústavy opakovane delíme dvoma a zvyšky čítame odzadu (téma Cykly). Opačne: každá cifra zľava vynásobí doterajší výsledok dvoma a pripočíta sa:

```
def z_dvojkovej(text):
    vysledok = 0
    for cifra in text:
        vysledok = vysledok * 2 + int(cifra)
    return vysledok

print(z_dvojkovej("1101"), z_dvojkovej("1111111"))
print(int("1101", 2), bin(13), hex(255))
```

Výstup:

```
13 255
13 0b1101 0xff
```

Vstavané funkcie: `int(text, základ)` prevedie z ľubovoľnej sústavy, `bin` a `hex` do dvojkovej a šestnástkovej.

7 Riešené úlohy

7.1 Najdlhšia rastúca súvislá časť

Zadanie: V zozname nájdí dĺžku najdlhšej súvislej časti, v ktorej každé číslo je väčšie ako predchádzajúce.

Riešenie: Prechádzame zoznam a počítame dĺžku aktuálnej rastúcej časti. Keď rast skončí, začínáme počítať od 1. Popri tom si pamätáme maximum – je to vlastne hľadanie maxima „za behu“.

```
z = [3, 5, 7, 2, 4, 6, 8, 9, 1, 3]
aktualna = najdlhsia = 1
for i in range(1, len(z)):
    if z[i] > z[i - 1]:
        aktualna += 1
        najdlhsia = max(najdlhsia, aktualna)
    else:
        aktualna = 1
print(najdlhsia)
```

Výstup:

```
5
```

7.2 Zlúčenie dvoch zoradených zoznamov

Zadanie: Dva zoradené zoznamy spoj do jedného zoradeného bez použitia `sort`.

Riešenie: Dva „ukazovatele“ `i` a `j` ukazujú na najmenší ešte nepoužitý prvok každého zoznamu. Menší z nich pridáme do výsledku a posunieme príslušný ukazovateľ. Keď sa jeden zoznam minie, pridáme zvyšok druhého.

```
a = [1, 4, 9, 12]
b = [2, 3, 10, 15, 20]
i = j = 0
vysledok = []
while i < len(a) and j < len(b):
    if a[i] <= b[j]:
        vysledok.append(a[i])
        i += 1
    else:
        vysledok.append(b[j])
        j += 1
vysledok += a[i:] + b[j:]
print(vysledok)
```

Výstup:

```
[1, 2, 3, 4, 9, 10, 12, 15, 20]
```

Toto zlučovanie je základom rýchleho triedenia zlučovaním (*merge sort*).

Zhrnutie

- Algoritmus je jednoznačný, konečný, všeobecný a dáva výsledok; zaujíma nás aj jeho efektívnosť.
- Maximum: začni prvým prvkom (nie nulou), pamätaj si hodnotu alebo index.
- Lineárne vyhľadávanie: až n porovnaní; `-1` vráť až po prejdení celého zoznamu.
- Bublínkové triedenie vymieňa susedov, triedenie výberom presúva minimum zvyšku; obe $\approx n^2/2$ porovnaní. V praxi `sort/sorted`.
- Binárne vyhľadávanie: iba v zoradenom zozname, $\approx \log_2 n$ krokov.
- NSD Euklidovým algoritmom (`a, b = b, a % b`), prvočísla testom deliteľov do $\sqrt{n}$ alebo Eratostenovým sitom.

Úlohy na precvičenie

1. Napíš funkciu, ktorá vráti minimum aj maximum zoznamu jedným prechodom.
2. Uprav bublinkové triedenie tak, aby triedilo zostupne, a spočítaj, koľko výmen urobilo.
3. Napíš funkciu, ktorá vráti zoznam indexov všetkých výskytov hodnoty v zozname.
4. Pomocou binárneho vyhľadávania zisti, koľko krokov treba na nájdenie čísla 1 v zozname `list(range(1, 1025))`.
5. Nájdi všetky dvojice prvočísel menších ako 100, ktoré sa líšia o 2 (prvočíselné dvojčatá).
6. Napíš funkciu, ktorá prevedie číslo z desiatkovej sústavy do ľubovoľnej sústavy so základom 2 až 9.

Úlohy k tejto téme nájdeš na learnwithmente.com v časti Informatika → Python.