

Algoritmické úlohy

Poznámky k téme

Táto téma spája všetko, čo už vieš. Pri skutočných úlohách – na písomke, v súťaži alebo na maturite – nestačí poznať príkazy; treba vedieť **postupovať**: pochopiť zadanie, rozložiť ho na menšie časti, napísať program, ktorý funguje aj pre nezvyčajné vstupy, a nebyť zbytočne pomalý. Naučíš sa postup riešenia, simuláciu krok po kroku, základy efektívnosti a testovanie hraničných prípadov.

Čo sa naučíš

- postupovať pri riešení úlohy od zadania po otestovaný program,
- rozložiť úlohu na podúlohy a tie zapísať ako funkcie,
- simulovať dej krok po kroku,
- odhadnúť, ako rýchlo program rastie s veľkosťou vstupu, a zrýchliť ho,
- nájsť a otestovať hraničné prípady.

1 Postup riešenia úlohy

1. **Pochop zadanie.** Čo je na vstupe (koľko údajov, akého typu, v akom tvare)? Čo má byť na výstupe – a v akom presnom tvare? Aké sú obmedzenia (rozsah čísel, veľkosť vstupu)?
2. **Vyrieš príklad ručne.** Na papieri prejdí príklad zo zadania. Ak ho nevieš vyriešiť ručne, nenaprogramuješ ho.
3. **Rozlož úlohu** na menšie podúlohy (načítanie, výpočet, výpis; alebo jednotlivé kroky výpočtu).
4. **Napíš plán** – slovami alebo pseudokódom. Ujasníš si, aké premenné a štruktúry budeš potrebovať.
5. **Programuj po častiach** a každú časť hneď vyskúšaj (výpisy medzivýsledkov).
6. **Otestuj** – na príklade zo zadania aj na **hraničných prípadoch**.
7. **Zamysli sa nad rýchlosťou**, ak je vstup veľký.

1.1 Ukážka postupu: najdlhšia séria

Zadanie: Hádzali sme kockou. Nájdi najdlhšiu sériu rovnakých hodov idúcich po sebe a vypíš jej dĺžku a hodnotu.

1. *Pochopenie:* vstup je zoznam čísel 1–6, výstup dve čísla. Čo ak je zoznam prázdny? Čo ak je viac rovnako dlhých sérií? (Rozhodneme sa: vypíšeme prvú.)

2. *Ručne:* pre [3, 5, 5, 2, 2, 2, 6, 2] je najdlhšia séria 2, 2, 2 – dĺžka 3, hodnota 2.

3.–4. *Plán:* prechádzame zoznam a počítame dĺžku aktuálnej série. Ak je prvok rovnaký ako predchádzajúci, séria rastie; inak začína nová s dĺžkou 1. Popritom si pamätáme najdlhšiu sériu (hľadanie maxima).

5. Program:

```
def najdlhsia_seria(hody):
    if not hody:
        return 0, None
    naj_dlzka, naj_hodnota = 1, hody[0]
    dlzka = 1
    for i in range(1, len(hody)):
        if hody[i] == hody[i - 1]:
            dlzka += 1
        else:
            dlzka = 1
        if dlzka > naj_dlzka:
            naj_dlzka, naj_hodnota = dlzka, hody[i]
    return naj_dlzka, naj_hodnota

print(najdlhsia_seria([3, 5, 5, 2, 2, 2, 6, 2]))
```

Výstup:

(3, 2)

6. Testy – aj hraničné prípady (podrobnejšie ďalej):

```
def najdlhsia_seria(hody):
    if not hody:
        return 0, None
    naj_dlzka, naj_hodnota = 1, hody[0]
    dlzka = 1
    for i in range(1, len(hody)):
        if hody[i] == hody[i - 1]:
            dlzka += 1
        else:
            dlzka = 1
        if dlzka > naj_dlzka:
            naj_dlzka, naj_hodnota = dlzka, hody[i]
    return naj_dlzka, naj_hodnota

testy = [
    ([3, 5, 5, 2, 2, 2, 6, 2], (3, 2)), # príklad zo zadania
    ([], (0, None)), # prázdny vstup
    ([4], (1, 4)), # jeden prvok
    ([6, 6, 6, 6], (4, 6)), # všetko rovnaké
    ([1, 1, 2, 2], (2, 1)), # dve rovnako dlhé – prvá
    ([1, 2, 3, 3], (2, 3)), # séria na konci
]

for vstup, ockavane in testy:
    vysledok = najdlhsia_seria(vstup)
    stav = "OK" if vysledok == ockavane else "CHYBA"
    print(f"{stav:5} {vstup} → {vysledok}")
```

Výstup:

```
OK    [3, 5, 5, 2, 2, 2, 6, 2] → (3, 2)
OK    [] → (0, None)
OK    [4] → (1, 4)
OK    [6, 6, 6, 6] → (4, 6)
OK    [1, 1, 2, 2] → (2, 1)
OK    [1, 2, 3, 3] → (2, 3)
```

2 Rozklad úlohy

Zložitejšiu úlohu rozdelíme na podúlohy, z ktorých každá je jednoduchá. Každú podúlohu zapíšeme ako funkciu (téma Modulárny program).

Úloha: *Skoky na lyžiach*. Každý skok dostane 5 známok. Najvyššia a najnižšia známka sa škrtnú, ostatné tri sa sčítajú. Vypíš poradie podľa súčtu.

Podúlohy: (1) vypočítať hodnotenie jedného skoku, (2) vypočítať hodnotenie všetkých skokov, (3) zoradiť a vypísať.

```
def hodnotenie(znamky):
    """Súčet známok bez najvyššej a najnižšej."""
    return sum(znamky) - max(znamky) - min(znamky)

def vysledky(skoky):
    return [(meno, hodnotenie(znamky)) for meno, znamky in skoky.items()]

def vypis_poradie(zoznam):
    zoradene = sorted(zoznam, key=lambda d: d[1], reverse=True)
    for poradie, (meno, body) in enumerate(zoradene, start=1):
        print(f"{poradie}. {meno:<8} {body:5.1f}")

skoky = {
    # štartové číslo a krajina: päť známok
    "12 SVK": [18.5, 19.0, 18.5, 17.5, 19.5],
    "27 NOR": [19.0, 19.5, 19.0, 19.0, 18.0],
    "31 AUT": [18.0, 18.5, 19.5, 18.0, 18.5],
}

vypis_poradie(vysledky(skoky))
```

Výstup:

```
1. 27 NOR    57.0
2. 12 SVK    56.0
3. 31 AUT    55.0
```

Každá funkcia sa dá otestovať samostatne – napríklad `hodnotenie([10, 10, 10, 10, 10])` musí byť 30.

3 Simulácia

Simulácia napodobňuje dej krok po kroku: deň po dni, ťah po ťahu, zastávku po zastávke. Program si v premenných pamätá **stav** (koľko je peňazí, kde je figúrka, koľko ľudí je vo vlaku) a v cykle ho podľa pravidiel mení.

3.1 Cestujúci vo vlaku

Zadanie: Vlak prechádza zastávkami. Na každej zastávke najprv vystúpia ľudia, potom nastúpia. Pre zadané počty vystupujúcich a nastupujúcich vypíš, koľko ľudí je vo vlaku po každej zastávke a aké najväčšie obsadenie vlak mal.

```
zastavky = [  
    ("Bratislava", 0, 120),  
    ("Trnava", 30, 45),  
    ("Trenčín", 60, 20),  
    ("Žilina", 70, 80),  
    ("Poprad", 100, 10),  
]  
vo_vlaku = 0  
maximum = 0  
for mesto, vystupi, nastupi in zastavky:  
    vo_vlaku = vo_vlaku - vystupi + nastupi  
    maximum = max(maximum, vo_vlaku)  
    print(f"{mesto:<11} -{vystupi:<3} +{nastupi:<3} → {vo_vlaku}")  
print("Najväčšie obsadenie:", maximum)
```

Výstup:

```
Bratislava  -0   +120 → 120  
Trnava      -30  +45  → 135  
Trenčín     -60  +20  → 95  
Žilina      -70  +80  → 105  
Poprad      -100 +10  → 15  
Najväčšie obsadenie: 135
```

3.2 Sporenie s úrokom

Zadanie: Na účte je 1000 €. Každý rok sa pripíše úrok 3 % a na konci roka sa vloží ďalších 200 €. Po koľkých rokoch prvýkrát presiahne suma 5000 €?

Riešenie: Počet rokov dopredu nepoznáme – simulujeme rok po roku cyklom `while`, kým suma nepresiahne cieľ.

```
suma = 1000
rok = 0
while suma <= 5000:
    rok += 1
    suma = suma * 1.03 + 200
    print(f"rok {rok:>2}: {suma:8.2f} €")
print("Cieľ dosiahnutý po", rok, "rokoch.")
```

Výstup:

```
rok  1: 1230.00 €
rok  2: 1466.90 €
rok  3: 1710.91 €
rok  4: 1962.23 €
rok  5: 2221.10 €
rok  6: 2487.73 €
rok  7: 2762.37 €
rok  8: 3045.24 €
rok  9: 3336.59 €
rok 10: 3636.69 €
rok 11: 3945.79 €
rok 12: 4264.17 €
rok 13: 4592.09 €
rok 14: 4929.85 €
rok 15: 5277.75 €
Cieľ dosiahnutý po 15 rokoch.
```

Pri simuláciách sa oplatí vypisovať stav po každom kroku – hneď vidno, či sa pravidlá uplatňujú správne. Po overení môžeš priebežný výpis vypnúť.

4 Efektívnosť

Dva správne programy sa môžu výrazne líšiť v rýchlosti. Pri malých vstupoch to nepostrehneš, ale pri veľkých (desiatky tisíc až milióny údajov) rozhoduje, **ako rastie počet krokov** s veľkosťou vstupu n .

Rast	Príklad	$n = 1\,000$	$n = 1\,000\,000$
logaritmický ($\approx \log_2 n$)	binárne vyhľadávanie	10	20
lineárny ($\approx n$)	jeden prechod zoznamom, súčet, maximum	1 000	1 000 000
$n \cdot \log n$	vstavané <code>sorted</code>	10 000	20 000 000
kvadratický ($\approx n^2$)	dva vnorené cykly cez celý zoznam	1 000 000	1 000 000 000 000

Python zvládne rádovo desiatky miliónov jednoduchých krokov za sekundu. Lineárny program pre milión údajov preto skončí okamžite, kvadratický by bežal hodiny.

4.1 Príklad: sú v zozname dve rovnaké čísla?

Kvadratické riešenie porovná každú dvojicu:

```
def ma_duplikat_pomaly(z):
    for i in range(len(z)):
        for j in range(i + 1, len(z)):
            if z[i] == z[j]:
                return True
    return False
```

Lineárne riešenie si pamätá už videné čísla v množine (získovanie `in` v množine je takmer okamžité):

```
import time

def ma_duplikat_pomaly(z):
    for i in range(len(z)):
        for j in range(i + 1, len(z)):
            if z[i] == z[j]:
                return True
    return False

def ma_duplikat_rychly(z):
    videne = set()
    for x in z:
        if x in videne:
            return True
        videne.add(x)
    return False

z = list(range(5000))          # bez duplikátov – najhorší prípad
for f in [ma_duplikat_pomaly, ma_duplikat_rychly]:
    t = time.time()
    vysledok = f(z)
    rychle = time.time() - t < 0.1
    print(f"{f.__name__}: duplikát {vysledok}, hotovo do 0,1 s: {rychle}")
```

Výstup:

```
ma_duplikat_pomaly: duplikát False, hotovo do 0,1 s: False
ma_duplikat_rychly: duplikát False, hotovo do 0,1 s: True
```

Pre 5 000 čísel urobí pomalá verzia asi 12,5 milióna porovnaní, rýchla 5 000 krokov. (Ešte kratšie:

```
len(set(z)) < len(z) .)
```

4.2 Typické zrýchlenia

- **Množina namiesto zoznamu** pri opakovanom `in`.
- **Nepočítaj to isté opakované.** `z.count(x)` v cykle cez celý zoznam je kvadratické – spočítaj výskyty raz do slovníka.
- **Zorad' raz**, potom využi zoradenie (binárne vyhľadávanie, susedné prvky).
- **Skonči hneď**, keď poznáš odpoveď (`break`, `return`).
- **Matematika pomáha:** súčet $1 + 2 + \dots + n$ je $n \cdot (n + 1) / 2$ – netreba cyklus; delitele stačí skúšať po $\sqrt{n}$.

4.3 Dvojica so zadaným súčtom

Zadanie: Zisti, či v zozname existujú dve čísla (na rôznych pozíciách) so súčtom `k`.

Riešenie: Pre každé číslo x hľadáme jeho „doplnok“ $k - x$ medzi číslami, ktoré sme už videli. Vďaka množine je to jeden prechod.

```
def dvojica_so_suctom(z, k):
    videne = set()
    for x in z:
        if k - x in videne:
            return (k - x, x)
        videne.add(x)
    return None

print(dvojica_so_suctom([8, 3, 11, 5, 7], 12))
print(dvojica_so_suctom([8, 3, 11, 5, 7], 4))
print(dvojica_so_suctom([6, 1, 6], 12))
```

Výstup:

```
(5, 7)
None
(6, 6)
```

5 Hraničné prípady

Hraničné prípady sú vstupy na „okraji“ toho, čo zadanie pripúšťa. Práve v nich býva najviac chýb. Pred odovzdaním programu si vždy polož tieto otázky:

Prípad	Na čo si dať pozor
prázdny vstup	delenie dĺžkou 0, <code>max([], z[0])</code>
jeden prvok	cyklus <code>for i in range(1, len(z))</code> sa nevykoná ani raz
všetky prvky rovnaké	„druhé najväčšie“ nemusí existovať
nula a záporné čísla	začiatočné maximum 0, delenie nulou, <code>%</code> so zápornými číslami
prvý a posledný prvok	chyba „o jednu“ v <code>range</code> , indexy mimo zoznamu
rovnaké výsledky (remíza)	ktorý vypísať? prvý, posledný, všetky?
veľmi veľký vstup	je program dost' rýchly?

```
>>> max([])
ValueError: max() arg is an empty sequence
>>> sum([]) / len([])
ZeroDivisionError: division by zero
>>> sum([])
0
```

Program, ktorý pri hraničnom prípade spadne, sa dá opraviť buď **podmienkou** (`if not z: return None`), alebo jasnou **výnimkou** s vysvetlením (téma Výnimky). Rozhodnutie zapíšať do komentára – aby bolo jasné, že ide o zámer, nie o náhodu.

5.1 Testovanie pomocou assert

Príkaz `assert podmienka` nerobí nič, ak podmienka platí; ak neplatí, program skončí chybou `AssertionError`. Hodí sa na rýchle automatické testy funkcií:

```
def priemer(z):  
    if not z:  
        return 0  
    return sum(z) / len(z)  
  
assert priemer([2, 4, 6]) == 4  
assert priemer([5]) == 5  
assert priemer([]) == 0  
assert priemer([-1, 1]) == 0  
print("Všetky testy prešli.")
```

Výstup:

```
Všetky testy prešli.
```

6 Riešené úlohy

6.1 Druhé najväčšie číslo – s hraničnými prípadmi

Zadanie: Napiš funkciu, ktorá vráti druhú najväčšiu **rôznu** hodnotu zoznamu, alebo `None`, ak neexistuje.

```
def druhe_najvacsie(z):  
    rozne = set(z)  
    if len(rozne) < 2:  
        return None  
    rozne.remove(max(rozne))  
    return max(rozne)  
  
for z in [[4, 9, 2, 9, 7], [5, 5, 5], [], [3, -1], [-8, -2, -5]]:  
    print(z, "→", druhe_najvacsie(z))
```

Výstup:

```
[4, 9, 2, 9, 7] → 7  
[5, 5, 5] → None  
[] → None  
[3, -1] → -1  
[-8, -2, -5] → -5
```

6.2 Najbližšia dvojica

Zadanie: Nájdí dve čísla v zozname, ktorých rozdiel je najmenší.

Riešenie: Porovnanie každej dvojice by bolo kvadratické. Po zoradení však stačí porovnať iba **susedné** čísla – najbližšie čísla budú v zoradenom zozname vedľa seba.

```
def najblizsia_dvojica(z):
    s = sorted(z)
    najlepsia = (s[0], s[1])
    for i in range(1, len(s) - 1):
        if s[i + 1] - s[i] < najlepsia[1] - najlepsia[0]:
            najlepsia = (s[i], s[i + 1])
    return najlepsia

print(najblizsia_dvojica([40, 12, 31, 7, 28, 55]))
```

Výstup:

```
(28, 31)
```

(Funkcia predpokladá aspoň dve čísla – pri jednom by `s[1]` spôsobilo chybu. Zadanie by to malo povedať; ak nepovie, ošetri to.)

7 Tipy na písomku a maturitu

- **Čítaj zadanie dvakrát.** Podčiarkni, čo je vstup, čo výstup a v akom tvare.
- **Dodrž presný tvar výstupu** – medzery, veľké písmená, počet desatinných miest.
- **Najprv funkčné riešenie, potom pekné.** Jednoduchý správny program je lepší ako zložitý nedokončený.
- **Spúšťaj často** – po každých pár riadkoch. Chybu nájdeš ľahšie, keď vieš, čo sa práve zmenilo.
- **Pomenúvaj premenné zrozumiteľne** a zložitejšie miesta okomentuj.
- **Otestuj hraničné prípady** z tabuľky vyššie.

Zhrnutie

- Postup: pochop zadanie → vyrieš príklad ručne → rozlož → naplánuj → programuj po častiach → testuj → zväž rýchlosť.
- Rozklad: každá podúloha je funkcia, ktorú možno otestovať samostatne.
- Simulácia: stav v premenných, cyklus podľa pravidiel, priebežný výpis pri ladení.
- Efektívnosť: lineárne verzus kvadratické riešenie; množina pre `in`, nepočítaj opakovane, zorad' raz.
- Hraničné prípady: prázdny vstup, jeden prvok, rovnaké hodnoty, nula a záporné čísla, okraje, remízy.
- `assert` na rýchle testy funkcií.

Úlohy na precvičenie

1. V zozname teplôt nájdi najdlhšie obdobie, keď teplota každý deň stúpala.
2. Simuluj hru: figúrka začína na políčku 0, v každom ťahu hodí kockou a posunie sa; na políčkach 5, 13 a 20 sa vracia na začiatok. Koľko ťahov potrebuje na dosiahnutie políčka 30?
3. Zisti, ktoré číslo v zozname sa nachádza najviac ráz – lineárne pomocou slovníka.
4. Napíš funkciu, ktorá vráti medián zoznamu (stredná hodnota po zoradení; pri párnom počte priemer dvoch stredných). Otestuj ju aspoň na piatich vstupoch vrátane hraničných.
5. Pre zoznam 10 000 náhodných čísel porovnaj čas kvadratického a lineárneho hľadania duplikátov.

Úlohy k této téme nájdeš na learnwithmente.com v části Informatika → Python.