

Řešení domácích úkolů - 2. týden

Převod do dvojkové soustavy

Napište program, který přečte ze vstupu nezáporné celé číslo a vytiskne jeho zápis ve dvojkové soustavě (bez úvodních nul).

Řešení

Analýza Načítáme jediné číslo, které máme převést do binárního tvaru, tedy na posloupnost nul a jedniček. Vyhýbáme se použití seznamů ani řetězců. Nejjednodušeji uskutečníme převod "zprava", tedy budováním výsledné binární reprezentace od nejméně významných bitů. Přirozené by bylo budovat výstup jako řetězec, ale stejně dobře můžeme ukládat binární číslice do celého desítkového čísla.

- nesmíme zapomenout na nulu
- různé způsoby konverze zabudované v Pythonu považujeme za zakázány.

Vzorové řešení

Odečítáme binární číslice zprava kombinací operací modulo a celočíselného dělení, a ukládáme je do *desítkového* celého čísla `binary`.

```
n = int(input())
binary = 0
power10 = 1

while n > 0:
    binary += (n % 2) * power10
    power10 *= 10
    n //= 2

print(binary)
```

Samozřejmě stejně dobře můžeme ukládat binární číslice

- do seznamu a spojit je operací `join`. Komplikace je, že máme seznam v obráceném pořadí a jeho položky potřebujeme převést na stringy. Řešení by vypadalo nějak takto:

```
n = int(input())
binary = []
while n > 0:
    binary.append(n % 2)
    n //= 2
if len(binary) == 0:
    binary = [0]
print("".join(map(str, reversed(binary))))
```

Rozebereme si poslední příkaz `print`:

Obrácení řetězce: `reverse(seznam)` vytvoří obrácený seznam jako nový objekt v paměti.

`reversed(seznam)` je *iterátor*, tedy jenom obálka kolem seznamu, která způsobuje, že se indexuje od konce a ne od začátku. Nevytváří se nový objekt a to je velká výhoda, pokud máme velký seznam.

Konverze položek na `str`: Máme dvě možnosti: (meta/-)funkce `map(fun, kontejner)` aplikuje funkci *fun* na každý prvek kontejneru *v momentě přístupu*, tedy nevytvorí nový objekt, ale modifikuje jenom ty položky, které si vyžádáme.

Druhá možnost je použití *list comprehension*: `[str(b) for b in binary]` dělá to co říká: vytvoří nový seznam z položek konvertovaných na `str`. Všimněme si podstatný rozdíl: tady se v paměti vytvoří duplikát seznamu `binary` s konvertovanými položkami.

Můžeme také celou konverzi nechat na funkci `print` pomocí hvězdičkové notace:

```
print(*reversed(binary), sep="")
```

- do stringu: Toto je podstatně jednodušší a vlastně i hezčí řešení, má ale důležitou nevýhodu: nebude dobře fungovat pro velice dlouhá čísla. Pro praktické použití to nemusí být podstatný problém, protože rozsah čísel, které bychom chtěli převádět do dvojkové soustavy je určitě ohraničený.

```
n = int(input())
binary = ""
while n > 0:
    binary = str(n % 2) + binary
    n //= 2
if len(binary) == 0:
    binary = "0"
print(binary)
```

Alternativní řešení

Postup zleva: Je možný, ale je to komplikovanější algoritmus. Musíme nejdříve najít nejvyšší mocninu 2, menší nebo rovnou vstupnímu číslu, a pak kombinujeme operace celočíselného dělení a modulo, abychom získali binární číslice. Ty ukládáme do desítkového celého čísla `binary`.

```
n = int(input())
binary = 0
power2 = 1

while 2*power2 <= n:
    power2 *= 2

while power2 > 0:
    binary = 10 * binary + (n // power2)
    n %= power2
    power2 //= 2

print(binary)
```

Rekurzivní řešení: Vytvoříme funkci, která vypočte jedinou binární číslici jako `n % 2`, pokud `n` ještě obsahuje další binární číslice, zavolá sama sebe pro `n // 2`, a vypíše vypočtenou binární číslici. To, že vypisujeme binární číslice před návratem z funkce nám zaručuje, že se vypíší ve správném pořadí.

```
def to_binary(n:int) -> int:
    digit = n % 2
    if n > 1:
        to_binary(n // 2)
    print(digit, end="")

n = int(input())
to_binary(n)
print()
```

Obvyklé problémy v řešeních

Porucha v 0

Nejčastější problém. Pojdme si zkusmo ukázat, jak jednotlivá výše uvedená řešení fungují pro vstup 0:

Základní řešení:

```
n = int(input())
binary = 0
power10 = 1

while n > 0:
    binary += (n % 2) * power10
    power10 *= 10
    n //= 2

print(binary)
```

Načteme 0, `binary` se nastaví na nulu a cyklus `while` se úplně přeskočí. Vytiskne se `binary`, tedy 0.

Postup zleva:

Na začátku `binary` je 0, `power2` je 1. První cyklus se přeskočí, ve druhém se `binary` nastaví opět na 0, a cyklus skončí po první iteraci, protože `power2` se nastaví na `1 // 2`, tedy na 0. Vytiskne se `binary`, tedy 0.

Rekurzivní řešení:

`digit` je při prvním volání `to_binary` 0, k rekurzivnímu volání funkce nedojde, a vytiskne se pouze 0, čímž program skončí.

Použití zabudované konverze

Tato řešení jsem nepřipouštěl, ale tady je rozumné je uvést pro informaci, jako provést binární konverzi, když ji potřebujete mimo této úlohy.

Funkce `bin`

Tato funkce vrací řetězec s binární reprezentací argumentu. Tato reprezentace má tvar `0bxxxxx`, takže první dva znaky vynecháváme:

```
n = int(input())
print(bin(n)[2:])
```

f-string

Pomocí formátového specifikátoru `:b` vytisknete číslo binárně.

```
print(f"{int(input()):b}")
```