date: 2025-11-05

1 integer representation and algorithms

we usually use base 10 to represent numbers, and we can break each number down like the following:

$$3528 = 3 \cdot 10^3 + 5 \cdot 10^2 + 2 \cdot 10^1 + 8 * 10^0$$

this process is called **decimal expansion**. other base systems work very similarly; and we can define it formally.

1.1 base b expansion of n

theorem: let b be an integer greater than 1. any $n \in \mathbb{Z}^+$ can be expressed *uniquely* in the form

$$n = a_k b^k + a_{k-1} b^{k-1} + ... + a_1 b^1 + a_0 b^0$$

where $k \in \mathbb{N}$, each $a_i \in \mathbb{N}$ where $a_i < b$ and $a_k \neq 0$.

when $b > 10$, we write each a_i as a single symbol in an extended "alphabet" of digits. for example, for a base 16 system, the digits would be: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

1.2 common base expansions

these base systems are very common in computing:

- base 2 (binary): expansions are bit strings.
  - 1 bit per digit
  - $412 = (110011100)_2$
- base 8 (octal): each digit a_i is $0 \leq a_i < 8$
  - 3 bits per digit
  - $412 = (634)_8$
- base 16 (hexadecimal): each digit $a_i \in \{0, 1, ..., 9, A, B, ..., F\}$
  - 4 bits per digit
  - $412 = (19C)_{16}$

1.3 constructing base b expansions

the algorithm works by repeatedly dividing the number n by the base b.

```
1 procedure base-b-expansion(n: ℕ, b: ℕ, b > 1)
2 |  q := n
3 |  k := 0
4 |  while q ≠ 0
5 |  |  a_k := q mod b
6 |  |  q := q div b
7 |  |  k := k + 1
8 |  return (a_{k-1}, a_{k-2}, ..., a_1, a_0)
```

the complexity of this algorithm is $O(n)$, where n is the number of digits.

1.3.1 aside: base 2^x systems

when the base system is a power of 2 ($b = 2^x$), we can convert it directly between binary and base b by grouping bits into chunks of x, rather than doing the full division algorithm.

for an octal (2^3) system, we group binary digits into chunks of 3; and for an hexadecimal (2^4) system, we can group them into chunks of 4. for example, if we want to convert an octal into a hexadecimal:

$$
\begin{aligned}
&(3720)_8 \\
&= (011\_111\_010\_000)_2 && \text{(convert each digit to decimal)} \\
&= (0111\_1101\_0000)_2 && \text{(regroup in chunks of 4)} \\
&= (7D0)_{16} && \text{(convert to hexadecimal)}
\end{aligned}
$$

1.4 base b algorithms

1.4.1 addition

this is the standard "column addition" we have learned in grade school. we add digits column by column from right to left; and if the sum (t) is bigger than the base b, we carry the overflow ($\lfloor t/b \rfloor$) to the next column.

```
1 procedure add(x: ℕ, y: ℕ, b: ℕ, b > 1)
2 |  c := 0
3 |  for j := 0 to n - 1
4 |  |  t := x_j + y_j + c
5 |  |  c := ⌊t/b⌋
6 |  |  s_j := t - bc
7 |  s_n := c
8 |  return (s_n, s_{n-1}, ..., s_1, s_0)
```

its complexity is $O(n)$ (linear), where n is the number of digits.

1.4.2 multiplication

this algorithm uses the "shift and add" method. we first multiply the entire number x by each single digit of y, then shift the product by j places, and finally find the sum.

```
1 procedure multiply(x: ℕ, y: ℕ, b: ℕ, b > 1)
2 |  p := 0
3 |  for j := 0 to n - 1
```

```
    c := 0
    for i := 0 to n - 1
        t := x_i · y_j + c
        c := ⌊t/b⌋
        r_i := t - bc
    r_n := c
    r := r * b^j
    p := add(p, r)
return (p_2n, p_2n−1, ..., p_1, p_0)
```

the complexity of this algorithm is $O(n^2)$, where n is the number of digits.