Notatki do wykładu z matematyki dyskretnej

Andrzej Szepietowski

Arytmetyka

0.1 System dziesiętny

Najpowszechniej używanym sposobem przedstawiania liczb naturalnych jest system dziesiętny, gdzie na przykład zapis

$$178$$

przedstawia liczbę składającą się z jednej setki, siedmiu dziesiątek i ośmiu jedności. Możemy to zapisać w postaci

$$178 = 1 \cdot 100 + 7 \cdot 10 + 8 \cdot 1$$

albo inaczej

$$178 = 1 \cdot 10^2 + 7 \cdot 10^1 + 8 \cdot 10^0.$$

Tak więc w systemie dziesiętnym kolejne cyfry oznaczają współczynniki przy kolejnych potęgach liczby dziesięć, zaczynając od największej, a kończąc na najmniejszej potędze. Mówimy, że liczba dziesięć jest podstawą lub bazą systemu dziesiętnego. W systemie dziesiętnym używamy dziesięciu cyfr

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9,$$

a zapis

$$d_r d_{r-1} \ \ldots \ d_1 d_0$$

oznacza liczbę

$$d_r \cdot 10^r + d_{r-1} \cdot 10^{r-1} + \ \ldots \ + d_1 \cdot 10^1 + d_0 \cdot 10^0, \tag{1}$$

którą możemy też zapisać w postaci

$$\sum_{i=0}^{r} d_i \cdot 10^i, \tag{2}$$

gdzie d_i są cyframi należącymi do zbioru $\{0, \ldots, 9\}$.

Liczby można też zapisywać w systemach z inną bazą. Jeżeli za bazę systemu wybierzemy liczbę b, to potrzebujemy b cyfr, a zapis

$$d_r d_{r-1} \ \ldots \ d_1 d_0$$

oznacza w systemie z bazą b liczbę

$$d_r \cdot b^r + d_{r-1} \cdot b^{r-1} + \ \ldots \ + d_1 \cdot b^1 + d_0 \cdot b^0, \tag{3}$$

którą możemy też zapisać w postaci

$$\sum_{i=0}^{r} d_i \cdot b^i. \tag{4}$$

0.2 System dwójkowy

W informatyce bardzo ważnym systemem jest system dwójkowy (binarny), gdzie podstawą jest liczba 2 i gdzie mamy tylko dwie cyfry, 0 i 1 (cyfry w systemie dwójkowym nazywa się *bitami*). Zapis

$$d_r d_{r-1} \ \ldots \ d_1 d_0$$

oznacza liczbę

$$\sum_{i=0}^{r} d_i \cdot 2^i = d_r \cdot 2^r + d_{r-1} \cdot 2^{r-1} + \ \ldots \ + d_1 \cdot 2^1 + d_0 \cdot 2^0.$$

Przykład 0.1 *Zapis* 110 *oznacza w systemie dwójkowym liczbę* $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$*, czemu w systemie dziesiętnym odpowiada* $4 + 2 = 6$.

Podobnie zapis 1101101 *oznacza w systemie dziesiętnym liczbę* $64 + 32 + 8 + 4 + 1 = 109$.

Poniżej w pierwszej tabeli przedstawiono jedenaście pierwszych potęg liczby 2 w postaci dwójkowej i dziesiętnej, a w drugiej – siedemnaście kolejnych liczb w postaci dwójkowej i dziesiętnej.

Potęga	Dwójkowy	Dziesiętny
0	1	1
1	10	2
2	100	4
3	1000	8
4	10000	16
5	100000	32
6	1000000	64
7	10000000	128
8	100000000	256
9	1000000000	512
10	10000000000	1024

Dwójkowy	Dziesiętny
0	0
1	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10
1011	11
1100	12
1101	13
1110	14
1111	15
10000	16

0.3 Zwiększanie liczby o jeden

Algorytm zwiększania liczby o jeden. Aby zwiększyć o jeden liczbę zapisaną w systemie dwójkowym:

- wskaż na ostatni bit,
- **powtarzaj**, co następuje, aż do zakończenia:
  - ◇ **jeżeli** wskazany bit jest zerem, **to**
    zamień go na jedynkę i zakończ algorytm;
  - ◇ **jeżeli** jest on równy jeden, **to**
    zamień go na zero i wskaż następny bit w lewo;
    jeżeli nie ma następnego bitu w lewo,
    to dostaw jedynkę na początku liczby i zakończ algorytm.

Mówiąc inaczej, szukamy pierwszego zera od prawej strony, zamieniamy go na jedynkę, a wszystkie jedynki stojące za nim zamieniamy na zera.

Przykład 0.2 *Liczba o jeden większa od liczby* 100100111 *to* 100101000.

Jeżeli liczba nie zawiera zer, tylko same jedynki, to zamieniamy te jedynki na zera i stawiamy jedynkę na początku.

Przykład 0.3 *Po liczbie* 11111 *jest liczba* 100000.

Zauważmy, że podobnie działa algorytm zwiększania o jeden liczb w systemie dziesiętnym (lub dowolnym innym systemie). Szukamy pierwszej od prawej cyfry różnej od dziewiątki (największej cyfry w systemie), zwiększamy tę cyfrę o jeden, a wszystkie stojące za nią dziewiątki zamieniamy na zera.

0.4 Porównywanie liczb

Algorytm porównywania liczb. Aby stwierdzić, która z dwóch liczb w postaci dwójkowej jest większa, postępuj w następujący sposób:

- **jeżeli** liczby nie są tej samej długości, **to**
  - ◇ większą jest dłuższa liczba;
- **jeżeli** liczby są równej długości, **to**
  - ◇ porównuj bit po bicie od lewej strony do prawej:
    - ▷ **jeżeli** bity są takie same, **to**
      przejdź do następnego bitu w prawo,
    - ▷ **jeżeli** bity są różne, **to**
      zakończ; większa jest liczba z większym bitem na tej pozycji,
  - ◇ **jeżeli** wszystkie bity są takie same, **to** porównywane liczby są równe.

Przykład 0.4 1011010 > 1010101*. Liczby te są tej samej długości, mają takie same trzy pierwsze bity, a różnią się dopiero czwartym bitem – czwarty bit pierwszej liczby jest większy od czwartego bitu drugiej liczby.*

0.5 Operacje arytmetyczne w systemie dwójkowym

Operacje dodawania, mnożenia, odejmowania i dzielenia w systemie dwójkowym można wykonywać podobnie do tak zwanych szkolnych pisemnych działań arytmetycznych w systemie dziesiętnym. Działania w systemie dwójkowym są prostsze, ponieważ mamy tu tylko dwie cyfry, 0 i 1. Zacznijmy od tabliczki dodawania i mnożenia:

+	0	1
0	0	1
1	1	10

*	0	1
0	0	0
1	0	1

Algorytm dodawania.

Dane wejściowe: dwie liczby w postaci binarnej, $d_r \ldots d_0$ oraz $e_r \ldots e_0$. Zakładamy, że liczby są tej samej długości. Jeżeli nie są, to krótszą uzupełniamy na początku zerami.

Dane wyjściowe: bity sumy obu liczb $s_{r+1}s_r \ldots s_0$.

- $s_0 := (d_0 + e_0) \bmod 2$.
- $c_0 := d_0 \cdot e_0$.
- **Dla kolejnych pozycji** i **od** 1 **do** r **oblicz:**
  - ◇ $s_i := (d_i + e_i + c_{i-1}) \bmod 2$,
  - ◇ $c_i := 1$, jeżeli co najmniej dwa spośród bitów d_i, e_i oraz c_{i-1} są jedynkami.
- $s_{r+1} := c_r$.

Przykład 0.5 *Dodawanie liczb* 10101 *i* 111 *wygląda następująco:*

$$\begin{array}{cccccc} & 1 & 0 & 1 & 0 & 1 \\ + & & & 1 & 1 & 1 \\ \hline & 1 & 1 & 1 & 0 & 0 \end{array}$$

Algorytm najpierw oblicza ostatni bit sumy s_0, który jest resztą z dzielenia $(d_0 + e_0)$ przez 2, oraz przeniesienie z ostatniego bitu c_0. Następnie dla każdej pozycji i od 1 do r oblicza bit sumy s_i oraz przeniesienie do następnej pozycji. Na końcu obliczany jest najbardziej znaczący bit sumy $s_{r+1} = c_r$.

Zauważmy, że pomnożenie liczby w postaci dwójkowej przez dwa oznacza dopisanie jednego zera na końcu liczby. Podobnie pomnożenie liczby przez i-tą potęgę dwójki oznacza dopisanie na końcu i zer.

Przykład 0.6 1101101 *pomnożone przez* 1000 *daje wynik* 1101101000.

Jeżeli liczba jest podzielna przez dwa, to ma w postaci binarnej zero na końcu, a jeżeli dzieli się przez i-tą potęgę dwójki, to ma na końcu i zer. Podzielenie liczby przez dwa oznacza skreślenie w jej postaci binarnej jednego zera z końca.

Przykład 0.7 1010011000 *podzielone przez dwa daje* 101001100.

0.6 Zamiana systemu

Zastanówmy się teraz, jak przechodzić od jednego sposobu przedstawiania liczby do drugiego. Ponieważ będziemy używać różnych systemów zapisu liczb, więc zapis w systemie z bazą b oznaczymy przez

$$(d_r d_{r-1} \ldots d_1 d_0)_b.$$

Zrezygnujemy z tego zapisu, jeżeli nie będzie wątpliwości, jakiego systemu używamy. Przedyskutujmy to na przykładach. Aby przedstawić liczbę

$$(110101)_2$$

w postaci dziesiętnej, korzystamy ze wzoru (3)

$$(110101)_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0,$$

i wykonujemy wszystkie rachunki w systemie dziesiętnym

$$(110101)_2 = 1 \cdot 32 + 1 \cdot 16 + 0 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 32 + 16 + 4 + 1 = (53)_{10}.$$

Teraz zamiana z systemu dziesiętnego na dwójkowy. Weźmy liczbę 178. Pierwszy sposób polega na tym, że wyszukujemy największą potęgę liczby 2, która jeszcze jest mniejsza od naszej liczby (w przykładzie $128 = 2^7$), następnie odejmujemy tę potęgę od naszej liczby i z różnicą postępujemy tak samo. Na końcu mamy liczbę w postaci sumy potęg dwójki. W naszym przykładzie wygląda to tak

$$178 = 128 + 50 = 128 + 32 + 18 = 128 + 32 + 16 + 2.$$

Teraz już łatwo zapisać naszą liczbę w postaci dwójkowej

$$(178)_{10} = (10110010)_2.$$

Drugi sposób polega na kolejnym dzieleniu liczby w sposób całkowity przez 2 i zapamiętywaniu reszt z dzielenia. Reszty te zapisane w odwrotnej kolejności tworzą zapis binarny liczby. Na przykład, weźmy znowu liczbę 178. W poniższej tabeli przedstawiono kolejne ilorazy i reszty.

Liczba	Iloraz	Reszta
178	89	0
89	44	1
44	22	0
22	11	0
11	5	1
5	2	1
2	1	0
1	0	1

Reszty zapisane w odwrotnej kolejności

$$10110010,$$

tworzą binarny zapis liczby $(178)_{10}$.

Ostatni sposób można łatwo uogólnić na algorytm zamiany liczb z systemu dziesiętnego na system z inną bazą b. Należy tylko dzielić przez b. Aby przedstawić liczbę 60 w systemie trójkowym, dzielimy ją kolejno przez 3 i spisujemy reszty z dzielenia:

Liczba	Iloraz	Reszta
60	20	0
20	6	2
6	2	0
2	0	2

W wyniku otrzymamy: $(60)_{10} = (2020)_3$.

0.7 Długość liczby

Zastanówmy się, ile cyfr zawiera zapis dziesiętny liczby naturalnej x. Cztery cyfry w systemie dziesiętnym mają liczby od $1000 = 10^3$ do $9999 = 10^4 - 1$, a k cyfr – liczby od 10^{k-1} do $10^k - 1$. Tak więc liczba x ma k cyfr wtedy i tylko wtedy, gdy $k-1 \leqslant \log_{10} x < k$, czyli gdy $\lfloor \log_{10} x \rfloor = k-1$. Mamy więc poniższy lemat.

Lemat 0.8 *Liczba naturalna x ma w systemie dziesiętnym $\lfloor \log_{10} x \rfloor + 1$ (w przybliżeniu $\log_{10} x$) cyfr.*

Podobnie w systemie z podstawą b, liczba x ma k cyfr wtedy i tylko wtedy, gdy $k-1 \leqslant \log_b x < k$.

Lemat 0.9 *W systemie o podstawie b liczba naturalna x ma* $\lfloor \log_b x \rfloor + 1$ *(w przybliżeniu* $\log_b x$*) cyfr.*

Korzystając z tego faktu, możemy ustalać przybliżoną liczbę cyfr potrzebną do zapisu liczb.

Wniosek 0.10 *Liczba, mająca k cyfr w systemie dziesiętnym, ma około* $k \log_2 10 \approx k \cdot \frac{10}{3}$ *bitów w systemie dwójkowym, a liczba, mająca k bitów w postaci dwójkowej, ma około* $k \cdot \frac{3}{10}$ *cyfr w postaci dziesiętnej.*

0.8 Duże liczby

Aby się zorientować, jak duże mogą być liczby przedstawione za pomocą systemu dziesiętnego lub dwójkowego, przypatrzmy się poniższej tabeli. Tabela ta pozwala ocenić algorytmy.

Liczba sekund w roku	$\approx 3 \times 10^7$
Wiek układu słonecznego w latach	$\approx 10^{10}$
Wiek układu słonecznego w sekundach	$\approx 10^{17}$
Liczba cykli w roku (100 MHz)	$\approx 3 \times 10^{15}$
Liczba ciągów 64-bitowych	$2^{64} \approx 1.8 \times 10^{19}$
Liczba ciągów 128-bitowych	$2^{128} \approx 3.4 \times 10^{38}$
Liczba ciągów 256-bitowych	$2^{256} \approx 1.2 \times 10^{77}$
Liczba atomów na Ziemi	$\approx 10^{51}$
Liczba atomów w naszej galaktyce	$\approx 10^{67}$

Dane do tabeli zaczerpnięto z książki A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, oraz z książki B. Schneier, *Applied cryptography.*

Przykład 0.11 *Rozważmy prosty algorytm sprawdzający, czy liczba naturalna x jest pierwsza. Algorytm ten dzieli x przez kolejne liczby od 2 do* $\sqrt{x}$*. Jeżeli x ma 120 bitów, to potrzeba* $2^{60} \approx 10^{18}$ *dzieleń. Jeżeli założymy, że komputer potrafi wykonać* 100 *milionów dzieleń w ciągu sekundy i* 3×10^{15} *w ciągu roku, to będzie liczył około 300 lat (znane są szybsze algorytmy sprawdzające pierwszość liczb).*

0.9 System szesnastkowy

W informatyce używa się też systemu szesnastkowego, gdzie podstawą jest liczba 16. Do systemu szesnastkowego potrzebujemy szesnastu cyfr. Zwykle używa się następujących „cyfr"

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

W poniższej tabeli zestawiono cyfry systemu szesnastkowego z odpowiadającymi im liczbami w systemie dwójkowym i dziesiętnym.

Szesnastkowy	Dwójkowy	Dziesiętny
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

W języku Pascal liczby w systemie szesnastkowym poprzedza się znakiem dolara \$, a w języku C – znakami $0x$.

Przykład 0.12 *Zapis* $\$A1$ *oznacza liczbę w systemie szesnastkowym, która w systemie dziesiętnym ma postać* $\$A1 = 10 \cdot 16 + 1 = (161)_{10}$.

Dzięki temu, że 16 jest potęgą dwójki, prosta jest zamiana postaci liczby z systemu dwójkowego na szesnastkowy, i na odwrót. Przy zamianie z systemu szesnastkowego na dwójkowy wystarczy zamieniać kolejne cyfry przedstawienia na grupy odpowiednich czterech bitów według powyższej tabeli.

Przykład 0.13 *Liczba, która w systemie szesnastkowym wygląda tak:* $\$A91$*, w systemie dwójkowym ma postać:* $1010|1001|0001$.

Przy zamianie z postaci dwójkowej na postać szesnastkową postępujemy odwrotnie. Zastępujemy grupy po cztery bity pojedynczymi cyframi.

Przykład 0.14 $(1110|0011|1011|0000)_2 = \$E3B0$.

Jeżeli liczba cyfr w postaci dwójkowej nie dzieli się przez 4, to uzupełniamy ją zerami na początku.

Przykład 0.15 $(110|1111|0110|0010)_2 = (0110|1111|0110|0010)_2 = \$6F62$.

W ten sposób możemy używać zapisu szesnastkowego do zwięzłego przedstawiania długich ciągów bitów.

0.10 Reprezentacja liczb w komputerze

W wielu językach każda zmienna ma swój typ, który jest deklarowany na początku programu. Sposób przechowywania wartości zmiennej zależy od jej typu.

0.10.1 Integer

Zmienne typu integer przechowywane są zwykle w dwóch bajtach. Jeden *bajt* (ang. *byte*) zawiera osiem bitów, tak więc wartość zmiennej typu integer przechowywana jest w szesnastu bitach. Pierwszy bit oznacza znak. Jeżeli jest on zerem, to liczba jest dodatnia, jeżeli jedynką, to ujemna.

Jeżeli liczba jest dodatnia, to pozostałe piętnaście bitów stanowi binarny zapis tej liczby. Na przykład liczba 15 jest przechowywana jako

$$0000|0000|0000|1111.$$

Największą liczbę dodatnią, jaką można przechować w zmiennej typu integer, jest

$$0111|1111|1111|1111,$$

czyli zero i piętnaście jedynek. Jest to

$$2^{15} - 1 = (32767)_{10}.$$

Liczby ujemne są przechowywane w tak zwanym systemie uzupełnieniowym. Liczba ujemna x o wartości bezwzględnej $|x|$ jest przedstawiana jako liczba

$$2^{16} - |x|$$

w postaci binarnej. Na przykład liczba -1 jest przedstawiona jako

$$1111|1111|1111|1111,$$

czyli szesnaście jedynek. A liczba -3 jako

$$1111|1111|1111|1101.$$

Najmniejsza liczba ujemna, którą można zmieścić do zmiennej typu integer, to

$$1000|0000|0000|0000,$$

czyli jedynka i piętnaście zer, która koduje liczbę

$$-2^{15} = (-32768)_{10}.$$

Często nie ma żadnego zabezpieczenia przed przekroczeniem maksymalnego lub minimalnego zakresu liczb typu integer. Jeżeli, na przykład, do liczby 32767, która jest przechowywana jako

$$0111|1111|1111|1111,$$

dodamy jedynkę, to otrzymamy

$$1000|0000|0000|0000,$$

która koduje liczbę -32768, i komputer nie zakomunikuje tego przekroczenia.

0.10.2 Real

Liczby typu real są zapisywane w dwóch notacjach:

- *stałopozycyjnej*,
- *zmiennopozycyjnej*.

Liczby w notacji stałopozycyjnej to na przykład

$$0.10, \qquad 11.023, \qquad 12.0,$$

czyli notacja dziesiętna. Zwróćmy uwagę, że kropka oddziela część całkowitą liczby od części ułamkowej.

W notacji zmiennopozycyjnej liczba przedstawiona jest w postaci

$$mEc,$$

gdzie m jest *mantysą*, liczbą w postaci dziesiętnej z przedziału $1 \leqslant m < 10$ lub $-10 < m \leqslant -1$, a c, zwana *cechą*, jest liczbą całkowitą. Zapis mEc oznacza liczbę

$$m \cdot 10^c.$$

W poniższej tabeli mamy kilka liczb w postaci stało- i zmiennopozycyjnej.

4837.92	$4.83792E3$
0.034	$3.4E-2$
-12.0	$-1.2E1$

0.11 Poszukiwania binarne (*binary search)*

Znana jest gra w dwadzieścia pytań. W tej grze za pomocą dwudziestu pytań, na które odpowiedzią może być „tak” lub „nie”, należy odgadnąć pomyślaną przez przeciwnika rzecz. Zobaczymy, jak można wykorzystać binarny system zapisu liczb do opracowania strategii wygrywającej w tej grze.

Uprośćmy nieco tę grę. Załóżmy, że możemy zadać tylko trzy pytania i że odgadujemy liczbę naturalną x ze zbioru

$$A = \{0, 1, 2, 3, 4, 5, 6, 7\}.$$

Aby odgadnąć liczbę, należy postępować w następujący sposób. Najpierw dzielimy zbiór na połowy: na liczby mniejsze od 4

$$\{0, 1, 2, 3\}$$

i na liczby większe lub równe 4

$$\{4, 5, 6, 7\},$$

i pytamy, do której połowy należy odgadywana liczba (dokładniej – pytamy, czy liczba należy do górnej połowy). Po uzyskaniu odpowiedzi mamy dwa razy węższy przedział poszukiwań i postępujemy z nim podobnie jak w pierwszej turze, dzielimy go na połowy i pytamy, w której połowie jest szukana liczba. Po drugiej odpowiedzi przedział poszukiwań jest już cztery razy krótszy i zawiera dwie liczby. W trzecim pytaniu znowu dzielimy przedział na połowy i pytamy, w której połowie znajduje się szukana liczba. Po trzeciej odpowiedzi przedział poszukiwań jest już osiem razy krótszy od wyjściowego i zawiera jedną liczbę, która jest tą szukaną.

Prześledźmy ten algorytm w przypadku, gdy szukaną liczbą x jest 5. Po odpowiedzi na pierwsze pytanie wiemy, że x jest w zbiorze $\{4, 5, 6, 7\}$. W drugiej rundzie dzielimy ten zbiór na połowy: $\{4, 5\}$ oraz $\{6, 7\}$ i pytamy, czy x jest w górnej połowie. Po negatywnej odpowiedzi wiemy, że x jest w $\{4, 5\}$. W trzeciej rundzie pytamy, czy $x = 5$.

Zauważmy także, że kolejne pytania dotyczą kolejnych bitów szukanej liczby. W pierwszym pytamy o pierwszy, w drugim o drugi, a w trzecim o trzeci bit szukanej liczby. Oczywiście taka zgodność pomiędzy bitami szukanej liczby i etykietami wierzchołków drzewa nie wystąpi przy innym ponumerowaniu elementów wyjściowego zbioru.

Jak zobaczyliśmy, trzeba trzy razy kolejno dzielić zbiór na połowy, aby z początkowego zbioru 8-elementowego dojść na końcu do zbiorów jednoelementowych. A ile trzeba podziałów, jeżeli na początku mamy $n = 2^k$ elementów? Po pierwszym podziale nasz zbiór będzie miał 2^{k-1} elementów, po drugim 2^{k-2}, a po i-tym 2^{k-i}. Jak widać, potrzeba $k = \log_2 n$ kolejnych podziałów, aby dojść do zbioru jednoelementowego. Tak więc jeżeli mamy do dyspozycji 20 pytań, to możemy odnaleźć jedną spośród 2^{20} liczb całkowitych z przedziału od 0 do $2^{20} - 1 = 1\,048\,575$.

0.11.1 Poszukiwanie pierwiastka

Metodę poszukiwań binarnych można zastosować do stwierdzenia, czy jakaś liczba naturalna n jest kwadratem (lub jakąś inną ustaloną potęgą) innej liczby naturalnej. Inaczej – czy istnieje liczba naturalna k, taka że $k^2 = n$, lub ogólniej, czy istnieje liczba naturalna k, taka że $k^\alpha = n$.

Poniżej przedstawiono taki algorytm. Algorytm ten używa dwóch dodatkowych zmiennych, k_d i k_g; wartości tych zmiennych przybliżają pierwiastek stopnia α z n od dołu i od góry. W trakcie wykonywania algorytmu przybliżenia te są coraz lepsze.

Algorytm szukający pierwiastka.

Dane wejściowe: liczba naturalna n.

Dane wyjściowe: pierwiastek naturalny stopnia α z n lub informacja, że takiego pierwiastka nie ma.

- Podstaw $k_d := 1$; $k_g := n$.
- **Powtarzaj aż do skutku**:

◇ **jeżeli** $k_g - k_d \leqslant 1$, **to**
koniec, n nie jest potęgą;
◇ **w przeciwnym wypadku:**
▷ $j := \lfloor \frac{k_d + k_g}{2} \rfloor$;
▷ **jeżeli** $j^\alpha = n$, **to** koniec, n jest potęgą j,
▷ **jeżeli** $j^\alpha > n$, **to** $k_g := j$,
▷ **jeżeli** $j^\alpha < n$, **to** $k_d := j$.

0.12 Zadania

1. Zwiększ o jeden liczby: a) $(11010011)_2$, b) $(111111)_2$, c) $(2012)_3$, d) $(2013)_4$.

2. Porównaj pary liczb: a) $(110011)_2$, $(110100)_2$, b) $(11010)_2$, $(1110)_2$, c) $(12121)_3$, $(12201)_3$, d) $(33132)_4$, $(33201)_4$.

3. Dodaj następujące pary liczb: a) $(110011)_2$, $(110100)_2$, b) $(11010)_2$, $(1110)_2$, c) $(11101)_2$, $(1111)_2$.

4. Dodaj w postaci trójkowej liczby $(2101)_3$ oraz $(1212)_3$.

5. Liczby $(81)_{10}$, $(126)_{10}$, $(200)_{10}$, $(257)_{10}$, $(258)_{10}$, $(1025)_{10}$, $(1023)_{10}$ przedstaw w postaci dwójkowej i ósemkowej.

6. Jak liczby 4, 20, -4, -20 będą reprezentowane w komputerze jako stałe typu integer?

7. Liczby \$B1, \$FF przedstaw w postaci dziesiętnej, dwójkowej i ósemkowej.

8. Liczby $(80)_{10}$, $(120)_{10}$ przedstaw w postaci trójkowej.

9. Liczby $(10001101)_2$, $(100101)_2$ przedstaw w postaci dziesiętnej, szesnastkowej i ósemkowej.

10. Liczby $(1023)_8$, $(713)_8$ przedstaw w postaci dwójkowej i dziesiętnej.

11. Ile maksymalnie pytań z odpowiedziami tak/nie trzeba zadać, aby odgadnąć liczbę: a) z przedziału od 0 do 100 000, b) z przedziału od 0 do 1 000 000?

12. Zastosuj algorytm wyznaczania pierwiastków do znalezienia następujących pierwiastków: $\sqrt[2]{256}$, $\sqrt[2]{1000}$, $\sqrt[3]{512}$, $\sqrt[4]{256}$.

13. Sprawdź, czy liczby 111, 1111 są potęgami liczb całkowitych.