Matematyka dyskretna dla informatyków. Wykład 7, wersja 2.0
INDUKCJA

1 Indukcja

1.1 Podstawowe pojęcia

Indukcja matematyczna pełni ważną rolę w programowaniu. Nie tylko jako metoda dowodzenia — z tym zastosowaniem zapoznaje uczniów program liceum — ale przede wszystkim jako metoda definiowania i rozwiązywania problemów algorytmicznych.

Podstawowa zasada jest zazwyczaj zrozumiała: dla zadanego ciągu zdań $P(0), P(1), \ldots$ ustalamy prawdziwość każdego z nich pokazując prawdziwość dla jednego (lub w razie potrzeby kilku) zdań bazowych (najczęściej dla jednego zdania $P(0)$), a następnie żądamy, aby udowodnione zdania zaświadczały o prawdziwości kolejnych. Najczęściej jest to wynikanie $P(n) \rightarrow P(n+1)$ dla każdego n. Taka zasada wnioskowania o prawdziwości wszystkich zdań z podanego ciągu nazywa się *zasadą indukcji prostej*. W odróżnieniu od niej stosuje się zasadę *indukcji zupełnej*, która do udowodnienia zdania $P(n+1)$ wymaga założenia o prawdziwości wszystkich zdań poczynając od bazowych aż do $P(n)$ włącznie. Nie ma istotnej różnicy między tymi metodami. Indukcja zupełna jest wygodniejsza w stosowaniu i w informatyce częściej spotykana. Dlatego mówiąc o indukcji będziemy zawsze rozumieli następującą zasadę indukcji zupełnej:

Zasada indukcji zupełnej
Dany jest ciąg zdań $P(0), P(1), \ldots$. Jeżeli prawdziwe są zdania $P(b), P(b+1), \ldots, P(b+k)$ dla pewnych $b, k \geq 0$ oraz jeżeli dla każdego $n \geq b+k$ z prawdziwości zdań $P(b), \ldots, P(n)$ wynika prawdziwość zdania $P(n+1)$, to zdanie $P(n)$ jest prawdziwe dla każdego $n \geq b$.

Zdania $P(b), \ldots, P(b+k)$ nazywamy *bazą indukcji*, wynikanie $(P(b), \ldots, P(n)) \rightarrow P(n+1)$ *krokiem indukcyjnym*. Liczba $k \geq 0$ określa liczbę zdań bazowych: jest ich $k+1$.

Przykładowo pokażemy, że $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$. Tutaj dowód będzie najbardziej klasyczny z możliwych. Bazą indukcji będzie jedno zdanie $P(0)$, czyli w naszej notacji k będzie równe 0. Zdanie to wystarcza do założenia bazy indukcji. Udowodnienie bazy indukcji polega tu na pokazaniu, że $\sum_{k=0}^{0} 2^k = 2^{0+1} - 1$, co oczywiście jest prawdą: obie strony są równe jeden.

Do pokazania kroku indukcyjnego wystarczy tak przekształcić lewą stroną równości, żeby skorzystać z założenia. Jest to proste, jak zresztą w przypadku wszystkich sum. Mamy bowiem ogólnie dla każdego ciągu a_k: $\sum_{k=0}^{n+1} a_k = \sum_{k=0}^{n} a_k + a_{n+1}$. W szczególności dla $a_k = 2^k$ mamy $\sum_{k=0}^{n+1} 2^k = \sum_{k=0}^{n} 2^k + 2^{n+1} = 2^{n+1} - 1 + 2^{n+1} = 2^{n+2} - 1$. Skorzystaliśmy tu z założenia indukcyjnego, że $\sum_{k=0}^{n} 2^k = 2^{n+1} - 1$.

Kolejny przykład będzie dotyczył sytuacji, gdy baza musi zawierać więcej niż jedno zdanie. Zdefiniujmy ciąg liczb Fibonacciego w następujący sposób: $F_0 = 0, F_1 = 1, F_n = F_{n-2} +_{n-1}$ dla $n > 1$. Kolejne liczby Fibonacciego, pełniące bardzo ważną rolę w informatyce, to $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$. Zostały wprowadzone przez Leonardo z Pizy zwanego Fibonaccim w jego dziele *Liber abaci*, pochodzącym z roku 1202 pierwszym średniowiecznym europejskim podręczniku matematyki.

Pokażemy teraz przez indukcję następujący wzór przypisywany dziewiętnastowiecznemu matematykowi Jacquesowi Binetowi[1]:

Twierdzenie 4.1.

$$F_n = \frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^n - (\frac{1-\sqrt{5}}{2})^n).$$

W naszej notacji dla każdego n zdanie mówiące o równości liczb po lewej i prawej stronie znaku równości nazwiemy $P(n)$. Dowód zaczniemy od pokazania prawdziwości dwóch zdań: $P(0)$ i $P(1)$.

Wiemy, że z definicji $F_0 = 0$. Po prawej stronie mamy $\frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^0 - (\frac{1-\sqrt{5}}{2})^0) = \frac{1}{\sqrt{5}}(1-1) = 0$.

Zbadajmy teraz prawdziwość zdania $P(1)$. Prawa strona równości wynosi $\frac{1}{\sqrt{5}}((\frac{1+\sqrt{5}}{2})^1 - (\frac{1-\sqrt{5}}{2})^1) = \frac{1}{\sqrt{5}}(\frac{1+\sqrt{5}-1+\sqrt{5}}{2}) = \frac{1}{\sqrt{5}}(\frac{2\sqrt{5}}{2}) = 1$. Zdanie $P(1)$ jest więc również prawdziwe.

Teraz pora na krok indukcyjny. Wprowadźmy pewne oznaczenia Niech $\varphi = \frac{1+\sqrt{5}}{2}$, a $\hat{\varphi} = \frac{1-\sqrt{5}}{2}$ W sprawnym wykonaniu tego kroku pomoże nam następujący lemat:

Lemat 4.2 Liczby φ oraz $\hat{\varphi}$ są pierwiastkami równania $x^2 = x + 1$.

Sprawdzenie, że tak rzeczywiście jest pozostawiamy Czytelnikowi. załóżmy, że teza twierdzenia 4.1 zachodzi dla wszystkich $k = 0, \ldots, n-1$, gdzie $n > 1$. Pokażemy, że zachodzi ona również dla n. Mamy więc udowodnić, że $F_n = \frac{1}{\sqrt{5}}(\varphi^n - \hat{\varphi}^n)$. Ze względu na to, że $F_n = F_{n-1} + F_{n-2}$ możemy zastosować krok indukcyjny: $F_n = \frac{1}{\sqrt{5}}(\varphi^{n-1} - \hat{\varphi}^{n-1}) + \frac{1}{\sqrt{5}}(\varphi^{n-2} - \hat{\varphi}^{n-2}) = \frac{1}{\sqrt{5}}(\varphi^{n-1} + \varphi_{n-2} - (\hat{\varphi}^{n-1} + \hat{\varphi}^{n-2})) = \frac{1}{\sqrt{5}}(\varphi^{n-2}(\varphi+1) - (\hat{\varphi}^{n-2}(\hat{\varphi}+1))) \stackrel{4.2}{=} \frac{1}{\sqrt{5}}(\varphi^{n-2}(\varphi^2) - (\hat{\varphi}^{n-2}(\hat{\varphi}^2))) = \frac{1}{\sqrt{5}}(\varphi^n - \hat{\varphi}^n)$.

Co było do pokazania.

Stosowanie indukcji często wiąże się z błędami. Oto przykłady błędnych rozumowań indukcyjnych:

1. Pokążemy, że wszyscy ludzie mają niebieskie oczy. Zrobimy to przez udowodnienie, że w dowolnym $n-$elementowym zbiorze każdy człowiek ma niebieskie oczy. Baza indukcji: w pustym zbiorze ludzi daje dobry wynik. Faktycznie każdy człowiek należący do zbioru pustego ludzi ma niebieskie oczy (podobnie, jak trąbę, świńskie uszy i ogon). Załóżmy teraz, że w każdym zbiorze ludzi o liczbie elementów mniejszej od n wszyscy mają niebieskie oczy. ,,Pokażemy", że również w każdym $n-$elementowym zbiorze wszyscy mają niebieskie oczy. Weźmy dowolny $n-$elementowy zbiór ludzi $l_1, l_2, \ldots, l_n$ dla $n \geq 1$. Na mocy założenia indukcyjnego w podzbiorze $l_1, \ldots, l_{n-1}$ wszyscy mają niebieskie oczy, bo jest to zbiór $n-1$-elementowy. Podobnie w podzbiorze $l_2, \ldots, l_n$, z tego samego powodu. W związku z tym wszyscy $l_1, \ldots, l_n$ mają niebieskie oczy, czego należało dowieść.

2. Pokażemy, że $2^n \geq n^2$. Baza: sprawdzamy, że faktycznie $2^0 \geq 0^2$. Pokażemy zatem, że z tego że nieroówność ta jest prawdziwa dla $n \geq 1$ wynika, że jest prawdziwa dla $n+1$. Czyli, że z założenia iż $2^n \geq n^2$, albo równoważnie $2^n - n^2 \geq 0$ wynika, że $2^{n+1} - (n+1)^2 \geq 0$. Przekształćmy lewą stronę tej nierówności: $2^{n+1} - (n+1)^2 = 2^n + 2^n - n^2 - 2n - 1 \geq 2^n - 2n - 1$. Ostatnia nierówność pochodzi z

[1] w rzeczywistości był on znany juz Eulerowi mniej więcej 100 lat wcześniej

zastosowania kroku indukcyjnego. Wystarczy więc pokazać, że dla każdego $n \geq 0$ zachodzi $2^n - 2n - 1 \geq 0$. Zrobimy to znowu indukcyjnie. Baza dla $n = 0$ wychodzi: $2^0 - 2 * 0 - 1 = 0 \geq 0$. Krok drugiej indukcji: zakładamy, że druga nierówność zachodzi dla $n \geq 0$. Pokażemy, że zachodzi również dla $n + 1$, czyli że $2^{n+1} - 2(n + 1) - 1 \geq 0$. Podobnie jak poprzedno przekształcamy lewą stronę: $2^{n+1} - 2(n + 1) - 1 = 2^n + 2^n - 2n - 1 - 2 \geq 2^n - 2$. Ta ostatnia nierówność zachodzi dla wszystkich $n \geq 1$, więc zadowoleni uznajemy, że ponieważ przypadek $n = 0$ pokazaliśmy jako bazę, to nierówność ta nas satysfakcjonuje i twierdzenie jest prawdziwe dla każdego n.

Guzik prawda! Co łatwo na palcach sprawdzić: druga nierówność jest fałszywa dla $n = 1$ oraz dla $n = 2$. Prawdziwa jest dla $n = 3$ (i wszystkich wartości większych od trzech, co można już pokazać bez błędu przez analogiczne rozumowanie z bazą $n = 3$!) Ale i to nie wystarcza do tego, aby odtrąbić prawdziwość pierwszej nierówności dla $n \geq 3$. Faktycznie jest zupełnie na odwrót: $2^3 < 3^2$. Nierówność ta zachodzi dopiero począwszy od wartości $n = 4$. Błąd nasz polegał na tym, że w trakcie dowodu kroku indukcyjnego robiliśmy dodatkowe założenia na n.

W informatyce mamy często do czynienia z jeszcze jedną postacią indukcji, tzw. indukcją noetherowską, od nazwiska wybitbej matematyczki niemieckiej, Emmy Noether, uczennicy Davida Hilberta. Emma Noether zauważyła, że tak naprawdę indukcję można przeprowadzać nie tylko w zbiorze liczb naturalnych, a nawet nie tylko w zbiorach uporządkowanych (dziwne: w końcu wydaje się, że trudnbo jest mówić o indukcji, gdy nie wiemy, co jest mniejsze od czego), a wystarczy założenie, że nasz zbiór ma określoną relację dobrze ufundowaną.

Zasada indukcji noetherowskiej

Niech X będzie dowolnym zbiorem, a r relacją dobrze ufundowaną określoną na X. Rozważmy formułę zdaniową $P(x)$ określoną na elementach zbioru X. Jeżeli dla każdego $x \in X$ jesteśmy w stanie pokazać, że z prawdziwości formuły $\forall y : ((x, y) \in R \to P(y))$ wynika prawdziwość formuły $P(x)$, to formuła $P(x)$ jest prawdziwa dla każdego $x \in X$.

Zauważmy, że zasada indukcji zupełnej jest szczególnym przypadkiem indukcji noetherowskiej: relacja $>$ jest dobrze ufundowana. I rzeczywiście, jeśli pokażemy, że dla każdego x z prawdziwości formuły dla wszystkich y takich, że $x > y$ wynika prawdziwość formuły dla x, to formuła nasza jest prawdziwa dla każdego x.

Zauważmy też, że w sformułowaniu indukcji noetherowskiej nie potrzebujemy w ogóle mówić o bazie indukcji. Wynika to stąd, że założenia indukcji noetherowskiej implikują, że formuła $P(x)$ musi być spełniona dla wszystkich elementów nie mających nastęnika y takiego, że $(x, y) \in R$. Formuła $\forall y : ((y, x) \in R \to P(y))$ est bowiem prawdziwa w trywialny sposób: poprzednik implikacji pod kwantyfikatorem jest fałszywy. Zatem z prawdziwej formuły musi wynikać $P(x)$, czyli $P(x)$ musi być prawdziwe. Założenie o bazie jest ukryte więc w ogólniejszym sformułowaniu założenia twierdzenia o indukcji noetherowskiej.

Dowód twierdzenia o indukcji noetherowskiej. Załóżmy, że z prawdziwości formuły $\forall y : ((x, y) \in R \to P(y))$ wynika prawdziwość formuły $P(x)$ dla każdego x, czyli że z prawdziwości formuły dla następników x w relacji r wynika prawdziwość formuły dla x, a jednocześnie formuła P jest fałszywa dla pewnego x_0. Z założenia wynika, że musiałby istnieć taki element x_1, że $(x_0, x_1) \in r$ oraz $P(x_1)$ byłoby zdaniem fałszywym.

Skoro tak, to musiałby istnieć element x_2 będący następnikiem x_1 (czyli $(x_1, x_2) \in r$), dla którego $P(x_2)$ byłoby zdaniem fałszywym. I tak dalej. Utworzylibyśmy nieskończony ciąg $x_0 r x_1 r x_2 r \ldots$ elementów, dla których formuła P przyjęłaby wartość fałsz. Pal licho ten fałsz! Ciekawsze jest to, że stworzyliśmy nieskończony łańcuch elementów będących ze sobą w relacji! Sprzeczność, bo zakładaliśmy dobre ufundowanie relacji.

Zasadę indukcji noetherowskiej stosuje się przy dowodzeniu własności stopu pętli programów. Chodzi o to, aby wykazać, że interesująca nas pętla w programie zakończy swoje działanie dla wszystkich danych. Aby taką własność udowodnić wystarczy określić pewną funkcję na wektorze wartości zmiennych występujących w pętli o wartościach w zbiorze dobrze ufundowanym tak, aby w kolejnych obrotach pętli dostawać wartości tej funkcji będące ze sobą w rozważanej relacji.

Przykład Rozważmy algorytm Euklidesa w wersji z dzieleniem modulo. Niech $n \geq m \geq 0$ będą liczbami naturalnymi takimi, że $n + m > 0$.

```
// Algorytm wyznaczania NWD(n,m)
int r;
while (m>0)
  {
  r=m;
  m=n mod m;
  n=r;
  }
wypisz(n);
```

Ponieważ zawsze `n mod m < m`, a zaczynaliśmy od `n>m`, więc szukaną funkcją może być funkcja $f(n, m, r) = [n, m]$ o wartościach w zbiorze N^2 z dobrze ufundowaną relacją r określoną następująco: $[n, m]r[n', m']$ wttw gdy $n \geq n'$ oraz $m \geq m'$ i $[n, m] \neq [n', m']$. Zauważmy teraz, że jeżeli w dwóch kolejnych obrotach pętli wartości zmiennych n i m są odpowiednio równe n_1, n_2 i m_1, m_2, to $n_1 > n_2$ i $m_1 > m_2$, Zatem na mocy definicji relacji r zachodzi $[n_1, m_1]r[n_2, m_2]$. Teraz wystarczy przyjąć za $P(x, y)$ formułę taką: program zatrzyma się dla $n = x$ i $m = y$.

2 Indukcyjne dowodzenie poprawności pętli

Bardzo często projektując algorytm chcemy upewnić się, czy jest on poprawny. Poprawny, czyli czy robi rzeczywiście to, czego się od niego spodziewamy. Musimy zacząć od określenia, czego się po algorytmie spodziewamy. Dokładniej: będziemy chcieli ustalić takie dwie formuły logiczne In i Out, żeby dla wszystkich danych spełniających początkowy warunek In wykonanie napisanego przez nas algorytmu doprowadziło do zagwarantowania prawdziwości formuły Out. Parę (In, Out) określającą to, czego oczekujemy nazywamy *specyfikacją problemu algorytmicznego*.

Oto przykłady specyfikacji problemów w tym stylu:

- **NWD** $In = (n_0 \geq m_0 \geq 0 \wedge n_0 + m_0 > 0)$, $Out = (n = NWD(n_0, m_0))$
- **sort** $In = (n > 0 \wedge \forall 0 \leq i < n : T[i] = a_i)$, $Out = (\forall 0 \leq i < n - 1 : T[i] \leq T[i + 1] \wedge \exists \pi \in \{1, \ldots, n\} \to \{1, \ldots, n\} : \pi \text{jest 1-1 i na taka, że} \quad T[\pi[i]] = a_i)$.

Warunek Out mówi, że w tablicy T są wszystkie elementy ciągu $a_1, \ldots, a_n$ i uporządkowane tak, że ich wartości nie maleją wraz ze wzrostem indeksów tablicy.

- **mnożenie** $In = (n = n_0 \wedge m = m_0)$, $Out = (z = m_0 n_0)$.
- **potęgowanie** $In = (n = n_0 \wedge m = m_0)$, $Out = (z = m_0^{n_0})$.

2.1 Algorytm mnożenia chłopów rosyjskich

Zajmiemy się teraz trzecim problemem. Opracujemy szybki algorytm mnożenia używający jedynie dodawania i działania dzielenia całkowitego przez 2. Algorytm ten jest nazywany algorytmem mnożenia chłopów rosyjskich. Ponoć na terenach Rosji były wioski, w których używano tego algorytmu do mnożenia liczb naturalnych (oczywiście sami chłopi mnożyli się w zupełnie inny sposób).

```
/* n0>=0 */
z=0; m=m0; n=n0;
while (n!=0) {
  if (n%2)
        z+=m;
  m+=m;
  n/=2;
}
/* z=m0*n0 */
```

To wcale nie jest oczywiste, że algorytm ten jest poprawny względem podanych warunków, czyli że po wyjściu z pętli zmienna `z` będzie rzeczywiście miała wartość równą iloczynowi `m0` i `n0`.

Aby to pokazać użyjemy metody zwanej metodą Floyda-Hoare'a, albo inaczej metodą niezmienników pętli. Polega ona na znalezieniu logicznego warunku N, który będzie spełniony ilekroć będziemy na początku pętli. Warunek ten powinien możliwie dokładnie opisywać zależności między zmiennymi w pętli. Jeżeli określimy go wystarczająco precyzyjnie (mocno), to za jego pomocą będziemy w stanie pokazać poprawność programu.

Wprowadźmy notację

$$In\{P\}Out$$

która dla dwóch formuł logicznych In i Out oraz pętli `P = (while (B) do I;` będzie oznaczała następujące stwierdzenie:

Jeżeli dane początkowe spełniają warunek `In`*, a pętla* `P` *dla tych danych zakończy swoje działanie, to dane końcowe będą spełniały warunek* `Out`

Mówimy wtedy, że pętla `P` jest *częściowo poprawna* względem warunków `In` i `Out`.

Niezmiennik, N który tworzymy (zazwyczaj konstruujemy pętlę myśląc od razu o niezmienniku), powinien spełniać następujące warunki:

1. $In \Rightarrow N$,
2. $B \wedge N\{I\}N$

3. $\neg B \wedge N \Rightarrow Out$

Zachodzi bowiem następujące twierdzenie. *Jeżeli istnieje niezmiennik N, który spełnia powyższe warunki, to pętla* `P` *jest częściowo poprawna względem warunków In i Out.*

Pokażemy najpierw, że warunek N jest spełniony na początku pętli przez cały czas jej wykonywania. Dowód tego faktu jest indukcyjny względem liczby n obrotów pętli. Baza zachodzi dla $n = 0$, gdyż na mocy (1) warunek N wynika bezpośrednio z warunku In.

Załóżmy teraz indukcyjnie, że po $n-1$ obrotach pętli warunek N jest spełniony na jej początku. Jeżeli teraz warunek B nie jest spełniony („nie puszcza”), to pętla zakończy swój żywot i N będzie spełniony. Jeśli natomiast B jest spełniony, to na mocy (2), po wykonaniu jednego obrotu pętli dla danych spełniających N (założenie indukcyjne) i B, warunek N odtworzy się.

To dowodzi faktu, że N jest prawdą również po n obrotach pętli — o ile do nich dojdzie — dla każdego n, ilekroć znajdujemy się w naszych obliczeniach na początku pętli. I to niezależnie od tego, czy właśnie z niej wychodzimy, czy też zamierzamy wykonać kolejny obrót.

Zatem w momencie wyjścia z pętli (o ile to wyjście w ogóle nastąpi!) będziemy mieli sytuację, w której dane spełniają N i $\neg B$. A to na mocy (3) gwarantuje prawdziwość warunku Out. Co było do pokazania.

Wróćmy do naszego przykładu. Pokażemy częściową poprawność naszej pętli używając jako niezmiennika warunku

$$N(m, n, z) = (m_0 n_0 = z + mn)$$

Sprawdźmy, że są spełnione założenia naszego twierdzenia. Zauważmy, że po początkowych przypisaniach mamy $In = (z = 0 \wedge m = m_0 \wedge n = n_0)$. Zatem w oczywisty sposób z formuły In wynika $m_0 n_0 = z + mn$. Bazę indukcji mamy gotową.

Teraz krok indukcyjny, czyli (2). Musimy pokazać, że $N(m, n, z)$ jest rzeczywiście niezmiennikiem.

```
/* n0>=0 */
z=0; m=m0; n=n0;
while (n!=0) {           //   m0*n0=z+m*n
  if (n%2)               //   m0*n0=z+m*n  oraz n jest nieparzyste
        z+=m;            //   m0*n0=z-m+m*n oraz n jest nieparzyste
                       // lub m0*n0=z+m*n oraz n jest parzyste
  n/=2;                //     m0*n0=z+m*(2*n)
  m+=m;                //     m0*n0=z+m/2*(2*n),
                       // ... a to jest po prostu N(n,m,z)
}
/* z=m0*n0 */
```

Czyli faktycznie po wykonaniu tych instrukcji niezmiennik $N(m, n, z)$ odtworzył się. Warto nadmienić, że nie potrzebowaliśmy tu wcale korzystać z warunku B.

Teraz ostatni krok: pokażemy, że $\neg B \wedge N(n, m, z) \Rightarrow Out$. Faktycznie $n = 0 \wedge n_0 m_0 = z + nm$ implikuje $z = n_0 m_0$. Algorytm nasz jest więc częściowo poprawny i rzeczywiście mnoży liczby. Zauważmy też, że nigdzie w dowodzie nie korzystaliśmy z faktu, że $n \geq 0$. Czyżby ten warunek wstępny był niepotrzebny? Spróbujmy uruchomić ten algorytm dla ujemnego n_0 i m_0. Widać, że wynik nie może wyjść poprawny: do z dodajemy jedynie wielokrotności m, a te są ujemne. Wynik zaś powinien wyjść dodatni. Bzdura!

Problem tkwi w tym, że zapomnieliśmy o zagwarantowaniu tego, że algorytm ma się zakończyć. W częściowej poprawności powiedziane jest wyraźnie: **jeżeli pętla się zakończy**, to wyniki będą spełniały Out. A jak się nie zakończy? To wtedy wszystko jest możliwe.

Dlatego przy dowodzeniu poprawności algorytmów korzystamy z pojęcia *poprawności całkowitej.*

Definicja Program jest poprawny całkowicie względem warunków In, Out wtedy i tylko wtedy, gdy

- Jest częściowo poprawny względem In, Out.
- Dla każdych danych spełniających In zakończy działanie

Zastanówmy się zatem, czy nasz algorytm jest całkowicie poprawny względem warunków $In = (n_0 \geq 0)$ i $Out = (z = m_0 n_0)$. Częściową poprawność pokazaliśmy już. Pozostaje wykazać to, że jeśli $n \geq 0$, to pętla zakończy swoje działanie w skończonej liczbie kroków.

W tym celu użyjemy metody omawianej w poprzednim rozdziale. Zdefiniujmy funkcję $f(n, m, z) = n$ i pokażmy, że po każdym obrocie pętli wartości tej funkcji są nieujemne i maleją. Wynika to z prostego faktu, że jeśli $n > 0$, to $0 \leq n/2 < n$. A ponieważ wchodzimy do pętli tylko dla $n > 0$, a zmienna n przyjmuje wewnątrz pętli wartość $n/2$, więc rzeczywiście tak jest. *Uwaga; $n/2$ w tym kontekście należy rozumieć, jako dzielenie całkowite z obcięciem. Tym niemniej wspomniane nierówności zachodzą również w dziedzinie całkowitej.*

Zauważmy, że gdybyśmy dopuścili ujemne wartości n, to funckja $f(n, m, z) = n$ nie byłaby już ani nieujemna ani malejąca. Można by się ratować określając ją jako $f'(n, m, z) = |n|$, co rokuje nieźle poza jednym wyjątkiem $n = -1$, dla którego mamy $(-1)/2 = -1$. I nie ma ostrej mniejszości! Jest to jedyna przyczyna, dla której dowód skończoności pętli się załamuje. Gorąco polecam zrobić symulację wykonania tej pętli dla danych ujemnych.

Założenie $n \geq 0$ było więc konieczne do pokazania całkowitej poprawności tego algorytmu.

Zauważmy, że podany algorytm ma bardzo sympatyczną złożoność. Liczba obrotów pętli jest równa logarytmowi przy podstawie 2 z n (zaokrąglonemu w górę), a w każdym obrocie pętli wykonujemy 3 proste działania: jedno dodawanie, jedno mnożenie przez 2 i jedno dzielenie przez 2. Ostatnie dwa działania są bardzo łatwe do implementacji w systemie binarnym.

2.2 Potęgowanie logarytmiczne

Wprowadźmy następujące zmiany w poprzednim programie. Inicjalizację z zmieńmy na 1, a dodawanie zastąpmy mnożeniem. Ponieważ mnożenie tak się ma do dodawania, jak potęgowanie do mnożenia, otrzymaliśmy szybki sposób obliczania potęgi $m_0^{n_0}$.

```
/* n0>=0 */
z=1; m=m0; n=n0;
```

```
while (n!=0) {
  if (n%2)
        z*=m;
  m*=m;
  n/=2;
}
/* z=m0^n0 */
```

Dowód tego faktu przebiega analogicznie do poprzedniego przypadku. Algorytm ten nosi angielską nazwę *binpower* i jest algorytmem o złożoności logarytmicznej ze względu na wielkość n_0.