Politechnika Gdańska

Wydział Elektroniki,
Telekomunikacji i Informatyki

Katedra: Architektury Systemów Komputerowych

Imię i nazwisko dyplomanta: Wojciech Pasternak

Nr albumu: 137361

Forma i poziom studiów: Stacjonarne jednolite studia magisterskie

Kierunek studiów: Informatyka

Praca dyplomowa magisterska

Temat pracy:
Obliczanie zer wielomianów

Kierujący pracą:
dr hab. inż. Robert Janczewski

Zakres pracy:
Obliczanie pierwiastków wielomianów i porównanie służących do tego struktur

Gdańsk, 2016

Spis treści

Rozdział 1

Wstęp

Rozdział 2

Przegląd literatury

2.1 Wielomiany

2.1.1 Definicja

Definicja 1 *Wielomianem zmiennej rzeczywistej x nazywamy wyrażenie:*

$$\begin{aligned} &W(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + ... + a_{n-1}x + a_n, \\ &gdzie\ a_0, a_1, a_2, ..., a_{n-1}, a_n \in R, n \in N \end{aligned} \tag{2.1}$$

Liczby $a_0, a_1, a_2, ..., 1_{n-1}, a_n$ nazywamy współczynnikami wielomianu, natomiast n nazywamy stopniem wielomianu.

Szczególnym przypadkiem wielomianu jest jednomian.

Definicja 2 *Jednomianem zmiennej rzeczywistej x nazywamy wielomian, który posiada co najwyżej jeden wyraz niezerowy i określamy wzorem:*

$$W(x) = ax^n, gdzie\ a \in R, n \in N \tag{2.2}$$

Można, więc rozumieć wielomian jako skończoną sumę jednomianów. Jednomianem stopnia zerowego jest stała, pojedyncza liczba rzeczywista, która w szczególności może być zerem.

Definicja 3 *Wielomianem zerowym nazywamy, wielomian wyrażony wzorem:*

$$W(x) = 0 \tag{2.3}$$

W dalszej części, jeżeli nie zaznaczymy inaczej, mówiąc wielomian, będziemy mieli na myśli pewien wielomian, nie będący wielomianem zerowym.

2.1.2 Podstawowe działania na wielomianach

Na wielomianach, tak jak na liczbach możemy wykonywać podstawowe działania. Należą do nich: porównywanie, dodawanie, odejmowanie, mnożenie, dzielenie, a także obliczanie reszty z dzielenia

oraz NWD (największego wspólnego dzielnika). Jako, że wielomian zmiennej x możemy traktować jak funkcję jednej zmiennej, możemy także policzyć z niego pochodne.

Porównywanie wielomianów

Porównywanie należy do najbardziej elementarnych działań na wielomianach. Wymaga ono zwykłego porównania kolejnych współczynników, a jego długość trwania, zależy od ich liczby. Zapoznajmy się z twierdzeniem dotyczącym operacji porównywania wielomianów.

Twierdzenie 1 *Dwa wielomiany uważamy za równe wtedy i tylko wtedy, gdy są tego samego stopnia, a ich kolejne współczynniki są równe.*

Powyższe twierdzenie nie jest złożone, nie mniej w celu pełnego zrozumienia, zilustrujmy je przykładem.

Przykład 1 *Mamy dane wielomian W_1 oraz wielomian W_2.*

$$\begin{aligned} W_1(x) &= a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \\ W_2(x) &= b_0x^n + b_1x^{n-1} + ... + b_{n-1}x + b_n \end{aligned} \tag{2.4}$$

Wielomiany W_1 oraz W_2 są równe wtedy i tylko wtedy gdy $\forall i \in N \ a_i = b_i$.

Można zauważyć, potencalny wpływ reprezentacji wielomianu na szybkość operacji porównania. Gdy mamy do czynienia z wielomianem, w którym uwzględniamy każdy współczynnik, także gdy jest on zerowy, złożoność czasowa porównania jest liniowa względem stopnia wielomianu. Natomiast w przypadku, gdy pomijamy wszystkie zerowe współczynniki wielomianu, złożoność również jest liniowa, ale tym razem względem liczby niezerowych współczynników wielomianów. Jak widać, w sytuacji, gdy stopień wielomianu jest znacznie większy od liczby zerowych współczynników, reprezentacja wielomianu ma niebagatelne znaczenie.
Dodatkowo, podobnie jak w przypadku porównywania liczb i sprawdzania kolejnych bitów, operacja porównania kończy się w momencie stwierdzenie, że porównywane współczynniki są różne lub porównaliśmy ze sobą już wszystkie współczynniki. Wynika z tego, że zakładając stały czas porównywania dwóch liczb, będącymi współczynnikami wielomianów, operacja porównania różnych wielomianów nigdy nie jest dłuższa od stwierdzenia, że porównywane wielomiany są równe.

Suma wielomianów

Dodawanie to kolejne elementarne działanie na wielomianach, które nie wymaga wykonywania skomplikowanych obliczeń.

Twierdzenie 2 *Aby dodać dwa wielomiany, należy dodać ich wyrazy podobne.*

Podobnie jak w przypadku porównywania czas dodawania wielomianów jest liniony, a ich reprezentacja ma zasadniczy wpływ na liczbę operacji dodawania. Pokażmy zastosowanie powyższego twierdzenia na przykładzie.

Przykład 2 *Mamy dane wielomian W_1 oraz wielomian W_2.*

$$\begin{aligned} W_1(x) &= a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \\ W_2(x) &= b_0x^n + b_1x^{n-1} + ... + b_{n-1}x + b_n \end{aligned} \tag{2.5}$$

Zdefiniujmy trzeci wielomian: $W_3(x) = W_1(x) + W_2(x)$. Wówczas:

$$W_3(x) = (a_0 + b_0)x^n + (a_1 + b_1)x^{n-1} + ... + (a_{n-1} + b_{n-1})x + a_n + b_n \tag{2.6}$$

Na powyższym przykładzie łatwo zaobserwować, że stopień sumy dwóch wielomianów nie może być większy od większego ze stopni dodawanych wielomianów. Znajduje to potwierdzenie w twierdzeniu, dotyczącym stopnia sumy wielomianów.

Twierdzenie 3

$$\deg(W_1 + W_2) \leq max(deg(W_1),\ deg(W_2)) \tag{2.7}$$

W przedstawionym twierdzeniu, należy uwagę na operator mniejsze równe. W przypadku gdy oba te wielomiany są tego samego stopnia, o przeciwnym współczynniku przy najwyższej potędze, to stopień ten będzie mniejszy.

Różnica wielomianów

Odejmowanie to operacja bliźniacza do dodowania, nie tylko w przypadku liczb, ale także w przypadku wielomianów. By pokazać olbrzymie podobieństwo tych operacji, zacznijmy od zapoznania się z definicją wielomianu przeciwnego.

Definicja 4 *Wielomianem przeciwnym nazywamy wielomian, którego wszystkie współczynniki są przeciwne do danych.*

Spójrzmy na poniższy przykład, pokazujący, że dla każdego wielomianu można bardzo prosto zdefiniować wielomian przeciwny, zmieniając znak wszystkich jego współczynników.

Przykład 3 *Mamy dany wielomian W_1.*

$$W_1(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \tag{2.8}$$

Zdefiniujmy drugi wielomian: $W_2(x) = -W_1(x)$. Wówczas:

$$W_1(x) = -a_0x^n + (-a_1)x^{n-1} + ... + (-a_{n-1})x + (-a_n) \tag{2.9}$$

Wiemy już, czym jest wielomian przeciwny. Przedstawmy teraz twierdzenie mówiące jak odejmować od siebie wielomiany.

Twierdzenie 4 *Aby obliczyć różnicę wielomianów W_1 i W_2, należy dodać ze sobą wielomiany W_1 i $-W_2$, czyli wielomian przeciwny do wielomianu W_2.*

Jak widać, przedstawione twierdzenie potwierdza analogię obliczania róznicy i sumy wielomianów. Spójrzmy na przykład, pokazujący jak obliczać różnicę wielomianów, potrafiając już je do siebie dodawać.

Przykład 4 *Mamy dane wielomian W_1 oraz wielomian W_2.*

$$\begin{aligned} W_1(x) &= a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \\ W_2(x) &= b_0x^n + b_1x^{n-1} + ... + b_{n-1}x + b_n \end{aligned} \tag{2.10}$$

Zdefiniujmy trzeci wielomian: $W_3(x) = W_1(x) - W_2(x)$. Wówczas:

$$W_3(x) = (a_0 - b_0)x^n + (a_1 - b_1)x^{n-1} + ... + (a_{n-1} - b_{n-1})x + a_n - b_n \tag{2.11}$$

Warto zauważyć, że wielomianem neutralnym ze względu na dodawanie i odejmowanie jest wielomian $W(x) = 0$. Oznacza to, że po dodaniu lub odjęciu wielomianu neutralnego, dostaniemy wynik, będący danym wielomianem.

Iloczyn wielomianów

Mnożenie to kolejne operacja zaliczająca się do podstawowych działań na wielomianach. Jego zasady przypominają nieco zwykłe mnożenie. Dokonujemy przemnożenia odpowiednich wyrazów, z tą różnicą, że w tym przypadku po prostu dodajemy wartości potęg dla odpowiednich współczynników. Zapoznajmy się z twierdzeniem, mówiącym dokładnie jak należy obliczać iloczyn wielomianów.

Twierdzenie 5 *Aby pomnożyć dwa wielomiany, należy wymnożyć przez siebie wyrazy obu wielomianów, a następnie dodać do siebie wyrazy podobne.*

Jak wynika z przedstawionej definicji poza mnożeniem dwóch liczb, mnożenie wielomianów w części polega na redukcji wyrazów podobnych, czyli operacji bazującej na dodawaniu. Spójrzmy na przykład, pokazujący jak definiuje się wielomian, będacy iloczynem dwóch wielomianów.

Przykład 5 *Mamy dane wielomian W_1 oraz wielomian W_2.*

$$\begin{aligned} W_1(x) &= a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \\ W_2(x) &= b_0x^m + b_1x^{m-1} + ... + b_{m-1}x + b_m \end{aligned} \tag{2.12}$$

Zdefiniujmy trzeci wielomian: $W_3(x) = W_1(x) * W_2(x)$. Wówczas:

$$\begin{aligned} W_3(x) =& (a_0b_0)x^{n+m} + (a_0b_1 + a_1b_0)x^{n+m-1} + (a_0b_2 + a_1b_1 + a_2b_0)x^{b+m-1} + ... \\ &+ (a_{n-2}b_m + a_{n-1}b_{m-1} + a_nb_m)x^2 + (a_{n-1}b_m + a_nb_{m-1})x + a_nb_m \end{aligned} \tag{2.13}$$

Można zauważyć, że po wymnożeniu wszystkich współczynników wielomianu liczba wyrazów iloczynu wynosi $(n+1)(m+1)$. Po dokonaniu redukcji wyrazów podobnych liczba ta ulega zmniejszeniu do wartości $n + m + 2$. Oznacza to zmianę liczby wyrazów z wartości kwadratowej, do

wartości liniowej względem stopni wielomianów. Liczba wyrazów podobnych, po przemnożeniu dwóch wielomianów jest symetryczna względem wykładników potęg poszczególnych współczynników. Można zauważyć, że skrajne wyrazy posiadają tylko po jednym wyrazie potęgi, a zbliżając się do współczynników o środkowych indeksach, liczba ta wzrasta, aż do wartości równej połowie stopnia otrzymanego wielomianu.

Widzimy, że czas operacji mnożenia wielomianów jest kwadratowy, wzgędem stopni mnożonych przez siebie czynników. Należy zauważyć, że jeżeli użyjemy reprezentacji wielomianu, w których posiadamy informację tylko o jego niezerowych współczynników, to czas operacji mnożenie będzie nadal kwadratowy, jednak względem liczby tych współczynników. Dla wielomianów wysokich stopni, w których zaledwie kilka współczynników jest niezerowych różnica ta może być niegatelna i w skrajnych przypadkach czas operacji może zmniejszyć się z kwadratowego, do czasu stałego.

Na podstawie powyższego przykładu, możemy także zaobserwować, że stopień wielomianu, będącego iloczynem dwóch wielomianów niezerowych, jest standardowo równy sumie stopni tych wielomianów. Wyjątkiem jest sytuacja gdy jeden z czynników jest wielomianem zerowym. Wówczas wynik takiej operacji również będzie wielomianem zerowym. Fakt ten znajduje potwierdzenie w poniższym twierdzeniu.

Twierdzenie 6

$$\begin{aligned} &deg(W_1 * W_2) = deg(W_1) + deg(W_2),\ dla\ W_1(x)! = 0, W_2(x)! = 0 \\ &W_3(x) = W_1(x) * W_2(x) = 0,\ w\ pozostałych\ przypadkach \end{aligned} \tag{2.14}$$

Z powyższego twierdzenia można zauważyć, że stopień otrzymanego wielomianu nigdy nie będzie wyższy od dwukrotności większego ze stopni mnożonych wielomianów.

Iloraz wielomianów

Dzielenie to zdecydowanie najtrudniejsza z elementarnych operacji na wielomianach. Aby dobrze zrozumieć jego zasady zapoznajmy się z definicją podzielności wielomianów oraz dzielnika wielomianu.

Definicja 5 *Wielomian W(x) nazywamy podzielnym przez niezerowy wielomian P(x) wtedy i tylko wtedy, gdy istnieje taki wielomian Q(x), że spełniony jest warunek* $W(x) = P(x) * Q(x)$. *Wówczas: wielomian Q(x) nazywamy ilorazem wielomianu W(x) przez P(x), zaś wielomian P(x) nazywamy dzielnikiem wielomianu W(x).*

Bardzo ważnym aspektem obliczania ilorazu wielomianów jest reszta z dzielenia. Spójrzmy na poniższą definicję.

Definicja 6 *Wielomian R(x) nazywamy resztą z dzielenia wielomianu W(x) przez niezerowy wielomian P(x) wtedy i tylko wtedy, gdy istnieje taki wielomian Q(x), że spełniony jest warunek* $W(x) = P(x) * Q(x) + R(x)$.

Łatwo zauważyć analogię w wyżej przedstawionych wzorach. Różnią się one właśnie wielomianem $R(x)$, czyli resztą z dzielenia. Gdy jest ona wielomianem zerowym, to znaczy, że mamy do czynienia z dzieleniem bez reszty i mówimy o podzielności dwóch wielomianów. Spórzmy na przykład, w którym zdefiniowane zostały dwa wielomiany, będące ilorazem i resztą z dzielenia dwóch wielomianów.

Przykład 6 *Mamy dane wielomian W oraz wielomian P.*

$$\begin{aligned} W(x) &= a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n \\ P(x) &= b_0x^m + b_1x^{m-1} + ... + b_{m-1}x + b_m \end{aligned} \tag{2.15}$$

Zdefiniujmy wielomian: $Q(x) = \frac{W(x)}{P(x)}$ *oraz* $Q(x) = W(x)\ mod\ P(x)$. *Wówczas:*

$$\begin{aligned} Q(x) &= c_0x^{n-m} + c_1x^{n-m-1} + ... + c_{n-m-1}x + c_{n-m},\ gdzie\ c_0 \neq 0 \\ R(x) &= d_0x^{n-m-1} + d_1x^{n-m-2} + ... + d_{n-m-2}x + d_{n-m-1} \end{aligned} \tag{2.16}$$

Zwróćmy uwagę na potęgi stojące przy najwyższych potęgach wielomianów $Q(x)$ oraz $R(x)$. Widzimy, że stopień ilorazu wielomianów jest zawsze równy różnicy stopni wielomianów, będących dzielną i dzielnikiem. Najważniejszym aspektem jest jednak fakt, że stopień reszty z dzielenia wielomianów jest zawsze mniejszy od stopnia ilorazu. Nie można natomiast ustalić jego wartości, bez dokładnej znajomości wielomianów W i P. W przykładzie podkreślony został fakt, że współczynnik stojący przy x, o potędze $n - m - 1$ może być zerem. To samo tyczy się także kolejnych współczynników. Gdy wszystkie one są zerami, to znaczy, że mamy do czynienia z resztą, będąco wielomianem zerowym. Oznacza to wówczas, że wielomian W jest podzielny przez wielomian P. Poniżej znajduje się twierdzenie, o stopniach wielomianów, będących ilorazem i resztą z dzielenia.

Twierdzenie 7

$$deg(W_1\ mod\ W_2) < deg(W_1/W_2) = deg(W_1) - deg(W_2) \tag{2.17}$$

Jak widać, twierdzenie potwierdza nasze obserwacje i wnioski dotyczące stopni obu wielomianów.

Pochodna wielomianu

Wielomiany jako przykład funkcji ciągłej, pozwalają na obliczanie pochodnych. By przekonać się, że jest to przykład jednej z prostszych operacji na wielomianach, zapoznajmy się z definicją.

Definicja 7 *Dany jest wielomian W, określony wzorem* $W(x) = a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n$. *Pochodną wielomianu W nazywamy wielomian W' i wyrażamy wzorem:*

$$W(x) = na_0x^{n-1} + (n-1)a_1x^{n-2} + ... + 2a_{n-2}x + a_{n-1} \tag{2.18}$$

Widzimy, że powstały wielomian powstał poprzez pomnożenie wartości każdego z współczynników przez stojącą przy danym wyrazie potęgę, a następnie obniżenie jej wartości o jeden. W ten

sposób potęgi wszystkich wyrazów wielomianu obniżają się. Wyjątkiem jest tutaj potęga o wartości zero, czyli stała. Pochodna z funkcji stałej jest zawsze równa zero, dlatego została pominięta w powyższym wzorze. O stopniu pochodnej wielomianu mówi poniższe twierdzenie.

Twierdzenie 8

$$\begin{aligned} &deg(W') = deg(W) - 1,\ dla\ deg(W) > 0 \\ &W(x) = 0,\ w\ pozostałych\ przypadkach \end{aligned} \tag{2.19}$$

Jak widać dla wszystkich wielomianów, nie będących stałą liczbową, stopień ich pochodnej ulega zmniejszeniu o jeden. Czas operacji obliczania pochodnej wielomianu jest porównywalny, z obliczaniem sumy wielomianów, gdyż wystarczy, że dokonamy jednokrotnego obliczania każdego z współczynników.

2.1.3 Dodatkowe twierdzenia dotyczące wielomianów

Istnieje mnóstwo twierdzień dotyczących wielomianów. Zapoznajmy się z tymi, które ułatwią nam znajdowanie pierwiastków wielomianów. Zapoznajmy się z pierwszym twierdzeniem, mówiącym o możliwości zamienienia dowolnego wielomianu o współczynnikach wymiernych, w wielomian o współczynnikach całkowitych.

Twierdzenie 9 *Dowolny wielomian* $W_1(x) = \frac{a_n}{b_n}x^n + \frac{a_{n-1}}{b_{n-1}}x^{n-1} + ... + \frac{a_1}{b_1}x + \frac{a_0}{b_0}$*, o współczynnikach wymiernych, można przekształcić w wielomian* $W_2(x) = k * W_1(x)$*, o współczynnikach całkowitych i tych samych pierwiastkach, co wielomian W. Wówczas:*

$$k = m * NWW(b_0, b_1, ..., b_{n-1}, b_n),\ gdzie\ m \in Z \tag{2.20}$$

Twierdzenie to oznacza, że przy dysponując wyłącznie współczynnikami, będącymi liczbami całkowitymi, jesteśmy w stanie przedstawić dowolny wielomian o współczynnikach wymiernych. Przejdźmy teraz do twierdzenia, mówiącego o wartości wielomianu w danym punkcie.

Twierdzenie 10 *Jeżeli wielomian W(x) podzelimy przez dwumian* $x - x_0$*, to reszta z tego dzielenia jest równa wartości tego wielomianu dla* $x = x_0$*.*

W szczególnym przypadku reszta ta może być równa zero. Oznacza to, że liczba x_0 jest pierwiastkiem tego wielomianu. Wynika z tego bezpośrednio kolejne twierdzenie, znane jako twierdzenie Bezout.

Twierdzenie 11 (Bezout) *Liczba* x_0 *jest pierwiastkiem wielomianu W(x) wtedy i tylko wtedy, gdy wielomian jest podzielny przez dwumian* $x - x_0$*.*

Dodatkowo zapoznajmy się jeszcze z krótkim dowodem poprawności powyższego twierdzenia.

Dowód Jeżeli liczba x_0 jest pierwiastkiem wielomianu W, to wielomian ten możemy wyrazić jako iloczyn dwumianu $x - x_0$ oraz pewnego wielomianu Q: $W(x) = (x - x_0) * Q(x)$. Wyznaczając z

tego wyrażenia wielomian Q, otrzymujemy $Q(x) = \frac{W(x)}{x-x_0}$. Widzimy zatem, że dzieląc wielomian W(x) przez dwumian $x - x_0$, otrzymujemy bez reszty, wielomian Q(x).

Przejdźmy teraz, to twierdzenia mówiącego o pierwiastkach wielokrotnych, bazującego na twierdzeniu Bezout.

Twierdzenie 12 *Liczba x_0 jest pierwiastkiem k-krotnym wielomianu W(x) wtedy i tylko wtedy, gdy wielomian jest podzielny przez $(x - x_0)^k$ i nie jest podzielny przez $(x - x_0)^{k+1}$.*

Dowód poprawności jest analogiczny, jak dla twierdzenia Bezout. Jedyną różnicą jest to, że wielomian W przedstawiamy jako: $W(x) = (x - x_0)^k * Q(x)$. Zapoznajmy się z twierdzeniem, mówiących o możliwej wartości pierwiastków, o ile są one całkowite.

Twierdzenie 13 *Dany jest wielomian $W(x) = x^n + a_{n-1}x^{n-1} + ... + a_1x + a_0$, o współczynnikach całkowitych. Jeżeli wielomian W posiada pierwiastki całkowite, to są one dzielnikami wyrazu wolnego a_0.*

Przejdźmy teraz do twierdzeń mówiących o możliwości rozłożenia wielomianu na czynniki.

Twierdzenie 14 *Każdy wielomian W(x) nie będący wielomianem zerowym jest iloczynem czynników stopnia co najwyżej drugiego.*

Oznacza to, że każdy wielomian stopnia co najmniej trzeciego jest rozkładalny na czynniki. Z powyższego twierdzenia wynika kolejne, mówiące, że rozkład ten jest zawsze jednoznaczny.

Twierdzenie 15 *Niezerowy wielomian, o współczynnikach rzeczywistych, jest jednoznacznie rozkładalny na czynniki liniowe lub nierozkładalne czynniki kwadratowe, o współczynnikach rzeczywistych.*

Oznacza to, że nie da się rozłożyć jednego wielomianu na czynniki, na dwa różne sposoby, tzn. tak, by istniał chociaż jeden czynnik (lub jego proporcjonalny odpowiednik), nie występujący w drugim rozkładzie. Ważnym aspektem, wynikającym z powyższego twierdzenia jest mowa, o tym że nie wszystkie wielomiany da się rozłożyć na czynniki liniowe, o współczynnikach całkowitych. Spójrzmy na pokazujący to przykład.

Przykład 7 *Mamy dany wielomian $W(x) = x^3 - 1$. Rózłóżmy go na czynniki. Wiemy, że pierwiastkiem tego wielomianu jest $x_0 = 1$, zatem możemy przedstawić wielomian W jako iloczyn wielomianu $x - 1$ oraz drugiego wielomianu.*

$$\begin{aligned} W(x) =& x^3 - 1 = x^3 - x^2 + x^2 - x + x - 1 = (x-1)x^2 + (x-1)x + x - 1 = \\ &= (x-1)(x2 + x + 1) \\ &\Delta = 1^2 - 4 * 1 * 1 = 1 - 4 = -3 < 0 - brak\ pierwiastków\ rzeczywistych \end{aligned} \tag{2.21}$$

Jak widać drugi z czynników jest właśnie przykładem nierozkładalnego czynnika kwadratowego, o współczynnikach rzeczywistych.

Powyższy czynnik da się rozłożyć na dwa czynniki liniowe, o współczynnikach zespolonych. W pracy tej będziemy jednak mówić wyłącznie o współczynnikach rzeczywistych, najczęściej zawężając jeszcze zbiór potencjalnych współczynników, do liczb wymiernych. Przejdźmy do kolejnego twierdzenia, wynikającego bezpośrednio z dwóch poprzednich.

Twierdzenie 16 *Każdy wielomian stopnia nieparzystego, ma przynajmniej jeden pierwiastek rzeczywisty.*

Oznacza to, że każdy wielomian stopnia nieparzystego jesteśmy w stanie przedstawić jako iloczyn dwóch czynników, z których jeden jest czynnikiem liniowym, a drugi czynnikiem stopnia parzystego, który z kolei być może da się dalej rozłożyć, na wielomiany, o mniejszych stopniach.

2.2 Eliminacja pierwiastków wielokrotnych

Ważnym aspektem obliczanie zer wielomianów jest eliminacja pierwiastków wielokrotnych. Jest ona niezbędna, by móc skorzystać z metody Sturmy. Zapoznajmy się z twierdzeniem dotyczącym krotności pierwiastków pochodnej wielomianu.

Twierdzenie 17 *Jeżeli liczba jest pierwiastkiem k-krotnym wielomianu W, to jest pierwiastkiem (k-1)-krotnym pochodnej tego wielomianu.*

By lepiej zrozumieć twierdzenie, spójrzmy na poniższy przykład.

Przykład 8 *Mamy dany wielomian* $W(x) = x^3 + 2x^2 + x$. *Obliczmy teraz kolejne pochodne wielomianu W.*

$$\begin{aligned} W'(x) &= 3x^2 + 2 * 2x + 1 = 3x^2 + 4x + 1 \\ W^{(2)}(x) &= 2 * 3x + 4 = 6x + 4 \\ W^{(3)}(x) &= 6 \end{aligned} \tag{2.22}$$

Obliczmy teraz pierwiastki wielomianu W i jego kolejnych pochodnych.

$$\begin{aligned} &W(x) = x^3 + 2x^2 + x = x(x^2 + 2x + 1) = x(x+1)^2 \\ &x_1 = 0,\ k_1 = 1,\ x_2 = -1,\ k_2 = 2 \\ &W'(x) = 3x^2 + 4x + 1 \\ &\Delta = 4^2 - 4 * 3 * 1 = 16 - 12 = 4 \\ &\sqrt{\Delta} = 2 \\ &x_1 = \frac{-4-2}{2*3} = \frac{-6}{6} = -1,\ k_1 = 1,\ x_2 = \frac{-4+2}{2*3} = \frac{-2}{6} = -\frac{1}{3},\ k_2 = 1 \\ &W^{(2)}(x) = 6x + 4 = 6(x + \frac{2}{3}) \\ &x_1 = -\frac{2}{3},\ k_1 = 1 \\ &W^{(3)}(x) = 6 \ - \ \textit{brak pierwiastków} \end{aligned} \tag{2.23}$$

Jak widać powyższy przykład potwierdza zastosowanie przedstawionego twierdzenia. Widzimy, że krotność wszystkich pierwiastków ulega zmniejszeniu o jeden, w kolejnej pochodnej. Dodatkowo możemy zauważyć, że pochodna może zawierać także pierwiastki, których nie miał dany wielomian. Ma to miejsce w przypadku, gdy wielomian, posiada przynajmniej dwa różne pierwiastki. Potwierdza to poniższe twierdzenie.

Twierdzenie 18 *Liczba nowych pierwiastków pochodnej wielomianu W' (takich których nie posiadał wielomian W) jest równa liczbie różnych pierwiastków wielomianu pomniejszonej o jeden.*

By dowieść powyższego twierdzenia przeprowadźmy rozumowanie dowodzące jego poprawności.

Dowód Zdefiniujmy wielomian: $W(x) = (x-x_1)^{k_1} * (x-x_2)^{k_2} * ... * (x-x_{m-1})^{k_{m-1}} * (x-x_m)^{k_m}$. Przyjmijmy, że wielomian W jest stopnia n i posiada m różnych pierwiastków, gdzie $1 \leq m \leq n$, $k_1+k_2+...+k_{m-1}+k_m = n$. Zdefiniujmy wielomian $P(x) = (x-x_1)*(x-x_2)*...*(x-x_{m-1})*(x-x_m)$, w którym każdy z pierwiastków wielomianu W występuję dokładnie jeden raz. Pierwiastków jest m, zatem wielomian P jest stopnia m. Dzieląc wielomian W przez wielomian P, otrzymujemy bez reszty wielomian $Q(x) = (x-x_1)^{k_1-1} * (x-x_2)^{k_2-1} * ... * (x-x_{m-1})^{k_{m-1}-1} * (x-x_m)^{k_m-1}$, który posiada każdy z pierwiastków wielomianu W, krotności pomniejszonej o jeden. Skoro krotność każdego z m pierwiastków uległa zmniejszeniu o jeden, to stopień wielomianu Q w stosunku do wielomianu W zmniejszył się o m. Stopień wielomianu Q jest wieć równy n-m. Na mocy twierdzenia wiemy także, że wielomian Q(x) jest podzielny również przez wielomian W', stopnia n-1. Zatem wielomian W' możemy przedstawić jako iloczyn wielomianu Q oraz innego wielomianu: $W'(x) = Q(x) * V(x)$. Wielomian V, zawiera wszystkie pierwiastki W', których nie posiadał wielomian W. Łatwo policzyć, że stopień wielomianu V jest równy: $\deg(V) = deg(W') - deg(Q) = (n-1) - (n-m) = n-1-n+m = m-1$. Jak widać, właśnie udowodniliśmy, że stopień ten jest równy liczbie różnych pierwiastków wielomianu W pomniejszonej o jeden.

Spróbujmy teraz skorzystać z przedstawionego twierdzenia w przykładzie dokonującym eliminacji pierwiastków wielokrotnych.

Przykład 9 *Dany jest wielomian W, określony wzorem:* $W(x) = x^6 - 6x^4 - 4x^3 + 9x^2 + 12x + 4$.
Dokonajmy eliminacji pierwiastków wielokrotnych dla wielomianu W.
Obliczamy pochodną wielomianu.

$$\begin{aligned} W'(x) &= 6 * x^5 - 4 * 6x^3 - 3 * 4x^2 + 2 * 9x + 12 = \\ &= 6x^5 - 24x^3 - 12x^2 + 18x + 12 = 6(x^5 - 4x^3 - 2x^2 + 3x + 2) \end{aligned} \tag{2.24}$$

Obliczmy teraz resztę z dzielenia wielomianu W przez wielomian W'.

$$\begin{array}{l} x \\ \hline x^6 - 6x^4 - 4x^3 + 9x^2 + 12x + 4 : (x^5 - 4x^3 - 2x^2 + 3x + 2) \\ \underline{- x^6 + 4x^4 + 2x^3 - 3x^2 - 2x} \\ - 2x^4 - 2x^3 + 6x^2 + 10x \end{array} \tag{2.25}$$

Kluczowa jest wartość reszty z dzielenia. Gdyby otrzymana reszta z dzielenia była wielomianem zerowym, to pochodna wielomianu było równocześnie NWD(W, W'). W tym przypadku tak jednak nie jest, więc wykonujemy analogiczną operację z tą różnicą, że nową dzielną jest dotychczasowy dzielnik, a reszta z wielomianu jest nowym dzielnikiem. Tę operację wykonujemy tak długo, jak otrzymana reszta jest niezerowego stopnia. W przypadku gdy jest ona jednocześnie wielomianem zerowym, to podobnie jak wyżej, aktualny dzielnik, jest naszym NWD(W, W'). W przypadku gdy otrzymana reszta jest niezerowym wielomianem stopnia zerowego, to największym wspólnym dzielnikiem wielomianów jest pewna niezerowa stała.

2.3 Twierdzenie Sturma

Przeanalizujmy na początek sposób konstruowania ciągu Sturma dla wielomianu W. Pierwszym wyrazem ciagu jest sam wielomian W. Z kolei drugim wyrazem jest pochodna wielomianu W. Kolejne wyrazy ciągu Sturma wyznaczamy obliczając resztę z ilorazu dwóch poprzednich wyrazów. Dzieje się to do uzyskania pierwszej reszty wielomianu, będącą wielomianem stopnia zerowego. Za każdym razem dzieląc wielomian stopnia n, przez wielomian stopnia m, gdzie $m < n$, mamy gwarancję, że wielomian będacy resztą tego ilorazu będzie stopnia mniejszego niż m. Korzystając z tego faktu, wiemy, że liczba wyrazów ciągu Sturma dla wielomianu W jest niewiększa od $\deg(W)+1$.

Definicja 8 *Ciągiem Sturma dla wielomianu W nazywamy ciąg wielomianów: $X_0, X_1, X_2, ...,$, takich że $X_0 = W$, $X_1 = W'$, a kolejne wyrazy definiuje się jako wielomian przeciwny to reszty z dzielenia dwóch poprzednich wyrazów, przy czym ostatnim wyrazem ciągu Sturma jest pierwszy wielomian stopnia zerowego.*

Wiemy już jak definiować ciąg Sturma. Przeanalizujmy teraz jaki wpływ ma uzyskany ciąg Sturma na obliczanie pierwiastków wielomianu. By to zrobić, zapoznajmy się z kolejną definicją, mówiącą o liczbie zmian znaków ciagu Sturma.

Definicja 9 *Liczbą zmian znaków ciągu Sturma dla wielomianu W(x) w punkcie x, obliczamy zliczając liczbę zmian pomiędzy kolejnymi wyrazami, pomijac te o wartości równej zero w punkcie x. Liczbę zmian znaku w punkcie $x = x0$ definiujemy jako wartość funkcji $Z(x_0)$.*

Wiemy już jak obliczać liczbę zmian ciagu Sturma. Sprawdźmy zatem, jak przekłada się ona na liczbę pierwiastków w danym przedziale.

Twierdzenie 19 *Jeżeli wielomian W(x) nie ma pierwiastków wielokrotnych, to liczba pierwiastków rzeczywistych w przedziale $a < x \leq y$), jest równa $Z(a) - Z(b)$.*

W ogólności twierdzenie Sturma można zastosować dla przedziału $(-\infty, +\infty)$, dopuszczając w ciągu Sturma wartości niewłaściwe $+\infty$ oraz $-\infty$. Wówczas $Z(-\infty)$ będzie oznaczać liczbę zmian znaków w ciągu $W(-\infty), W_1(-\infty), W_2(-\infty), ..., W_m(-\infty)$, zaś $Z(+\infty)$ liczbę zmian w ciągu $W(+\infty), W_1(+\infty), W_2(+\infty), ..., W_m(+\infty)$.

Twierdzenie 20 *Liczba różnych pierwiastków rzeczywistych wielomianu W(x) jest równa* $Z(-\infty)-Z(+\infty)$.

Stosując twierdzenie Sturma dla coraz mniejszych przedziałów możliwe jest wyznaczenie pierwiastków wielomianu z dowolną dokładnością. Sposób ten określony jest mianem metody Sturma. Twierdzenie Sturma jest bardzo mocnym środkowiem używanym do znajdowania pierwiastków w określonym przedziale. Może być używana nie tylko dla wielomianów, ale dowolnej różniczkowalnej funkcji ciągłej. Zapoznajmy się z twierdzeniem, mówiącym o ograniczeniach dotyczących maksymalnej liczby zmian znaków dla Ciąg Sturma.

Twierdzenie 21 *Liczba zmian znaków ciągu Sturma dla wielomianu W(x) jest mniejsza od liczby wyrazów tego ciągu i niewiększa od stopnia wielomianu W(x).*

Można, więc zauważyć, że warunkiem wystarczającym i jednocześnie koniecznym do tego by wielomian W(x) stopnia n, posiadający wyłącznie pierwiastki jednokrotne, posiadał n pierwiastków rzeczywistych jest to by wartość ciągu Sturma wynosiła n dla $x = -\infty$ oraz 0 dla $x = +\infty$.

Warto zauwazyć, że dla liczb dostatecznie dużych, co do wartości bezwzględnej, z uwzględnieniem wartości niewłaściwych $+\infty$ oraz $-\infty$ liczba zmian znaków zależy wyłącznie od współczynnika stojącego przy najwyżej potędze wielomianu. Znajduje to potwierdzenie w poniższym twierdzeniu.

Twierdzenie 22 *Jeżeli współczynnik stojący przy najwyższej potędze wielomianu jest większy od 0, to wartość tego wielomianu jest większa od zera w punkcie* $x = +\infty$ *oraz mniejsza od zera w punkcie* $x = -\infty$. *Z kolei, gdy współczynnik stojący przy najwyższej potędze wielomianu jest mniejszy od 0, to wartość tego wielomianu jest mniejsza od zera w punkcie* $x = +\infty$ *oraz większa od zera w punkcie* $x = -\infty$.

Na podstawie powyższego twierdzenia można zauważyć, że nie ma potrzeby wyliczania wartości wyrazów ciągu sturma dla wartości niewłaściwych, tzn. $x = -\infty$ oraz $x = +\infty$. Dzieje się tak dlatego, że wartość wielomianu nie jest ważna, a istotny jest jedynie znak. Fakt ten ma duży wpływ na optymalizacje obliczeń, gdyż dla wielomianów wysokich stopni ograniczamy się jedynie do sprawdzenia znaku przy najwyższej potędze, pomijąc wykonywanie potęgowania, mnożenia i sumowania kolejnych wyrazów wielomianu. Jeżeli zależy nam, na obliczeniu wartości wielomianu, dla odpowiednio dużych x, będącego ilorazem dwóch wielomianów, możemy skorzystać z poniższego twierdzenia.

Twierdzenie 23 *Dany jest wielomian* $W_1(x) = a_0x^n+a_1x^{n-1}+a_2x^{n-2}+...+a_{n-1}x+a_n$, *gdzie* $a_0 \neq 0$, *stopnia n oraz wielomian* $W_2(x) = b_0x^{n-1} + b_1x^{n-2} + b_2x^{n-3} + ... + b_{n-2}x + b_{n-1}$, *gdzie* $b_0 \neq 0$, *stopnia n-1. Wówczas* $\lim_{x\to+\infty} \frac{W_1(x)}{W_2(x)} = \frac{a_0}{b_0}$ *oraz* $\lim_{x\to-\infty} \frac{W_1(x)}{W_2(x)} = -\frac{a_0}{b_0}$.

Jak widać jesteśmy w stanie ustalić znak, a nawet dokładną wartość, największego współczynnika wielomianu, będącego ilorazem dwóch innych wielomianów, wykonując proste dzielenie dwóch liczb, będących odpowiednio najwyższymi współczynnikami wielomianów - dzielnej i dzielnika.

2.4 Biblioteka Mpir

2.4.1 Podstawowe informacje

Wstęp

Mpir jest przykładem biblioteki napisanej w języku C, pozwalającą operować na liczby o dowolnej wielkości i dokładności. Jej celem było dostarczenie narzędzia dla arytmetyki na liczbach, które nie są wspierane przez podstawowe typy wbudowane w języku C.

Kluczowy jest fakt, że biblioteka ta została zoptymalizowana w taki sposób, by w zależności od potrzeb używać odmiennych algorytmów. Stało się tak dlatego, że algorytm działania na małych liczbach z niewielką precyzją jest prosty i znacznie różni się od wymyślnej metody, używanej w przypadku wielkich liczb, z setkami cyfr po przecinku.

Warto zauważyć, że optymalizacja kodu odbywa się także w zależności od rodzaju procesora, na którym są wykonywane obliczenia. Biblioteka przeznaczona dla procesorów Intel i7 dostarcza inny kod niż dla procesora Pentium 4 lub Athlon. Dzięki temu, z pewnością, działa w sposób bardziej wydajny, kosztem wygody programistów, którzy jej używają. Nie są oni w łatwy sposób zapewnić optymalnego działania programu, niezależnie od sprzętu na którym jest on uruchamiany, co może znacznie utrudnić proces powstawania nowego oprogramowania, bazującego na bibliotece mpir.

Licencja

Biblioteka mpir jest przykładem wolnego oprogramowania, charakteryzującego się kilkoma podstawowymi założeniami. Pierwszą z nich jest wolność pozwalająca użytkownikowi na uruchamianie programu, w dowolnym celu. Drugą jest jego analiza oraz dostosowywania go do swoich potrzeb. Kolejną cechą jest możliwość dowolnego rozpowszechnianie kopii programu. Ostatnią zaś jest udoskonalanie programu i publiczne rozprowadzanie własnych ulepszeń, z których każdy będzie mógł skorzystać.

Nazewnictwo i typy

Biblioteka mpir posiada odpowiedniki dla wszystkich podstawowych typów liczbowych, które są charakterystyczne dla języka C. Odpowiednikiem typów całkowitych jest typ mpz_t, zaś odpowiednikiem liczb zmiennoprzecinkowych, typu float, jest mpq_t. Dla liczb całkowitych możemy łatwo zdecydować, czy interesuje nas liczba ze znakiem, czy bez niego. Oba przypadki posiadają prawie identycznie nazwane funkcje. Jedyną różnicą jest to, że dla zmiennych ze znakiem występuje przyrostek si, od wyrazu angielskiego wyrazu signed, a dla zmiennych bez znaku przyrostek ui, od wyrazu unsigned. Typ mpq_t daje użytkownikowi możliwość zdefiniowana dowolnej liczby, dającej się przedstawić w postaci ułamka, czyli dowolnej liczby wymiernej. Ograniczenie to spowodowane jest faktem, iż liczba taka składa się z dwóch liczb typu mpz_t. Pierwsza z nich jest licznikiem, a druga mianownikiem.

Funkcje

W bibliotece przyjęto konwencję charakterystyczną, dla języka C, polegającą na tym, że zarówno argumenty wejściowe jak i wyjściowe funkcji są przekazywane za pomocą referencji do funkcji, która zazwyczaj jest typu void, czyli nie zwraca żadnej wartości. Warto zauważyć, że argumenty wyjściowe są podawane przed wejściowymi. Stało się tak poprzez zachowanie analogii do operatora przypisania, w którym to zmienna wynikowa znajduje się po lewej stronie. W bibliotece mpir zmienna po zadeklarowaniu nie jest natychmiastowo gotowa do użycia, ponieważ wymaga inicjalizacji, przydzielającej jej miejsce w pamięci. Możemy to zrobić poprzez wywołanie odpowiedniej funkcji init, w argumencie podając odpowiednią zmienną. Analogicznie, aby zwolnić zajmowany, przez zmienną, obszar pamięci należy wywołać funkcję clear. Istnieją także funkcje realokujące, czyli zwiększające lub zmniejszające, zajmowany obszar. Użycie ich jest jednak opcjonalne, gdyż zmiana rozmiaru jest automatycznie wykrywana. Raz zainicjalizowaną zmienną, można używać dowolną liczbę razy. Twórcy biblioteki zalecają unikanie nadmiernego korzystania z funkcji init i clear, ponieważ są to operacje czasochłonne i zbyt duża liczba ich wywołań może mieć negatywny wpływ na wydajność naszego programu.

Zarządzanie pamięcią

Biblioteka mpir, alokując w pamięci nowe obiekty, stara się minimalizować wielkość, zajmowaną przez nie, miejsca. Dodatkowa pamięć jest przyznawana dopiero, kiedy jej aktualna wielkość jest niewystarczająca. W przypadku zmiennych typów mpz_t i mpq_t, raz zwiększony rozmiar danej zmiennej nigdy nie zostanie zmniejszony. Zazwyczaj jest to najlepsza praktyka, ponieważ częsta alokacja pamięci ma niekorzystny wpływ na wydajność programu. Jeżeli aplikacja potrzebuje zmniejszyć zajmowane przez daną zmienną miejsce, można dokonać realokacji albo całkowicie zwolnić, zajmowane przez nią miejsce. Dodatkowo kolejne zwiększania rozmiaru przeznaczonego na poszczególne zmienne może powodować zauważalny spadek wydajności lub fragmentację danych. Nie da się uniknąć, jeżeli chcemy zaalokować więcej miejsca dla danej zmiennej, w momencie, gdy w jej sąsiedztwie znajdują się już inne zmienne. Wówczas zależnie od implementacji mamy do czynienia z jednej z dwóch wariantów. W pierwszym przypadku, alokujemy nowe miejsce w pamięci, uzupełniając odpowiednio jego wartość, a następnie zwalniając wcześniej zajmowane miejsce. Alternatywą jest pozostawienie zajmowanej już pamięci i przydzielenie dodatkowej w innym miejscu. W takim przypadku jesteśmy zmuszeni do odpowiedniego zarządzania nieciągłymi fragmentami pamięci, co również może mieć bardzo negatywny wpływ na wydajność aplikacji. Z kolei, zmienne typu mpf_t mają niezmienny rozmiar, zależny od wybranej precyzji.
Warto zauważyć, że biblioteka standardowo do alokowania pamięci domyślnie używa funkcji malloc, ale możliwe jest także korzystanie z funkcji alloca. Pierwszy alokator zajmuje się tylko przydzieleniem pamięci, a użytkownik musi sam pamiętać, o jego zwolnieniu, w odpowiednim momencie. Z kolei, drugi z nich zdejmuje tę odpowiedzialność ze strony programisty i sam dba o zwolnienie pamięci, gdy jest ona już nie używana.

Kompatybilność

Poza kilkoma wyjątkami, biblioteka mpir jest kompatybilna z odpowiednimi wersjami biblioteki gmp. Dodatkowo, jej twórcy starają się nie usuwać istniejących już funkcji. W przypadku, gdy któraś przestanie być rekomendowana, zostaje po prostu zaznaczona jako przestarzawała, ale jej implementacja nie przestaje istnieć w kolejnych wydaniach biblioteki. Dzięki temu program, bazujący na starszej wersji biblioteki, zadziała też na nowszej edycji.

Wydajność

Dla małych liczb, narzut na korzystanie z biblioteki może być znaczący w porównaniu do typów prostych. Jest to nieuniknione, ale celem biblioteki jest próba znalezienia złotego środka pomiędzy wysoką wydajność, zarówno dla małych, jak i dużych liczb.

Operacje w miejscu

Operacje obliczania wartości bezwzględnej i negacji danej liczby są bardzo szybkie, gdy obliczane są w miejscu, tzn. wtedy, gdy zmienna wejściowa jest również zmienną wyjściową. Wówczas nie ma potrzeby alokacji i zwalnia pamięci, a cała funkcja sprowadza się do ustawienia odpowiedniego bitu, mówiącego o znaku liczby. Według specyfikacji biblioteki mpir, zauważalny powinien być także zysk, w przypadku operacji dodawania, odejmowania i mnożenie w miejscu. W momencie, gdy drugim argumentem jest nieduża liczba całkowite operację te są nieskomplikowane i bardzo szybkie.

2.4.2 Instalacja

Instalacja biblioteki mpir jest różna w zależności od systemu operacyjnego. W systemach unixowych instalacja polega na zbudowaniu i instalacji, korzystając ze źródeł. Pod windowsem jest ona równie łatwa i przebiega analogicznie, o ile używamy cygwina lub mingw. Są to narzędzia, które zapewniają, programom działającym pod systemem Windows, funkcjonalność przypominającą system Linux. Bardziej skomplikowane będzie użycie biblioteki bez wyżej wymienionych narzędzi, ale warto zaznaczyć, że biblioteka ta może być budowana z użyciem Microsoft Visual Studio, począwszy od wersji 2010, korzystając z programu o nazwie yasm asembler. Dodatkowo korzystanie z biblioteki różni się w zależności od tego czy potrzebujemy jej statyczną (mpir/lib) czy dynamiczną (mpir/dll) wersję.

Aby używać biblioteki mpir w naszym programie musimy do naszego programu dodać następującą linię:

```
#include <mpir.h>
```

Pozwoli ona nam na używanie wszystkich funkcji i typów, które udostępnia dla nas biblioteka. Dodatkowo wymagana jest kompilacja z dołączeniem naszej biblioteki, poprzez dodanie opcji -lmpir, np.:

```
gcc myprogram.c -lmpir
```

Jeżeli chcemy skorzystać z biblioteki, wspierającej język C++ dodatkowo musimy dodać także opcję -mpirxx, np.:

```
g++ myprogram.cc -lmpirxx -lmpir
```

2.4.3 Operacje na liczbach całkowitych

W niniejszym podrozdziale, omówię podstawowe funkcje liczbowe, które mają zastosowanie, zarówno dla liczb całkowitych typu mpz_t, jak i rzeczywistych typu mpq_t.

Funkcje inicjalizujące

```
void mpz_init (mpz_t integer)
```

Alokuje w pamięci miejsce na zmienną integer i ustawia jej wartość na 0.

```
void mpz_clear (mpz_t integer)
```

Zwalnia miejsce zajmowaną przez zmienną. Funkcja ta powinna być używana dla każdej zmiennej, w momencie, gdy nie ma już potrzeby, by z niej korzystać.

```
void mpz_realloc2 (mpz_t integer, mp bitcnt_t n)
```

Zmienia rozmiar zajmowanego przez zmienną miejsca. Funkcja jest używana z wartością n, większą od aktualnej, by zagwarantować zmiennej określone miejsce w pamięci. Z drugiej strony, zostaje wywołana z wartością mniejszą, jeżeli chcemy zmniejszyć liczbę zajmowanego przez zmienną miejsca. Gdy nowy rozmiar jest wystarczający by pomieścić aktualną wartość zmiennej, to zostaje ona zachowana. W przeciwnym razie zostanie ona ustawiona na 0. Istnieje również przestarzała funkcja mpz_realloc, ale jej użycie jest niezalecane. Nie została usunięta, by zachować kompatybilność wsteczną.

Funkcje przypisania

```
void mpz_set (mpz_t rop, mpz_t op)
```

Pozwala na przypisanie wartości tych samów typów. Wartość zmiennej op jest ustawiana jako rop. Jest to równoważnik operatora przypisania.

```
void mpz_set_sx (mpz_t rop, intmax_t op)
```

Pozwala na przypisanie zmiennej typu całkowitego do wartości typu mpz_t.

```
void mpz_set_d (mpz_t rop, double op)
```

Zapisuje wqrtość typu double do zmiennej typu mpz_t.

```
void mpz_set_q (mpz_t rop, mpq_t op)
```

Pozwala na zrzutowanie liczby rzeczywistej, typu mpq_t do zmiennej mpz_t. Gdy liczba mpq_t nie jest liczbą całkowitą, zostaje ona zaokrąglona w dół, poprzez obcięcie jej części ułamkowej.

```
int mpz_set_str (mpz_t rop, char *str, int base)
```

Konstruuje liczbę typu mpz_t, na podstawie podanego łańcucha znaków, reprezentującego daną liczbę. Wartość zmiennej base mówi o podstawie podanej liczby. Zmienna znakowa dopuszcza występowanie białych znaków, które są ignorowane. Z kolei zmienna base dopuszcza 0 oraz wartości z zakresu (2,61). Gdy jest ona równa 0, podstawowa zostaje ustalona, bazując na początkowych znakach. Dla prefixów 0x oraz 0X reprezentacja liczby zostaje ustalona jako szesnastkowa. Gdy jest ona równa 0b lub 0X to liczba ta jest binarna, a gdy rozpoczyna się od zera, a drugi znak jest inny od wymienionych, to zostaje uznana za liczbę oktalną. W pozostałych wypadkach jest to liczba dziesiętna. Kolejne wartości dziesiętne od 10 do 35 są reprezentowane jako litery od a do z, przy czym nie ma rozróżnienia ze względu na wielkość liter. W przypadku wyższych wartości podstawy, wielkość liter ma znaczenia, przy czym liczby od 10 do 35 reprezentowane są przez duże litery, a liczby od 36 do 61 przez litery małe. Funkcja dokonuje weryfikacji, czy podane ciąg znaków w całości reprezentuje poprawną wartość liczbą. Jeżeli tak, to zwraca ona wartość 0. W przeciwnym wypadku jest to wartość 1.

```
void mpz_swap (mpz_t rop1, mpz_t rop2)
```

Używana jest do zamiany wartości pomiędzy dwoma zmiennymi typu mpz_t. Jej użycie jest rekomendowane, ze względów wydajnościowych. W przypadku braku tej funkcji, potrzeba by zmiennej tymczasowej. W tym celu musiała by ona na początku zostać zaalokowana w pamięci, a na końcu zwolniona. Obie operacje są czasochłonne i zaleca się minimalizować ich użycie, więc użycie zoptymalizowanej funkcji służącej do zamiany wartości dużych liczb wydaje się zdecydowanie najlepszą i najszybszą opcją.

Funkcje konwersji

```
intmax_t mpz_get_sx (mpz_t op)
```

Zwraca zmienną typu int, a znak liczby zostaje przepisany. Jeżeli wartość jest poza zakresem liczb typu int, to rzutowanie następuje poprzez pozostawienie najmniej znaczącej części. Powoduje to, że funkcja w wielu przypadkach może okazać się bezużyteczna.

```
double mpz_get_d (mpz_t op)
```

Pozwala zrzutować typ mpz_t na zmienną typu double. Jeżeli jest to konieczne, stosowane jest zaokrąglenie. Jeżeli eksponent jest za duży, zwrócony wynik jest zależny od danego systemu. Jeżeli jest dostępna, zwrócona może być wartość nieskończoności.

```
char * mpz_get_str (char *str, int base, mpz_t op)
```

Zwraca ciąg znaków reprezentujących liczbę op., o podstawie danej w paremetrze o nazwie base. Funkcja ta jest analogiczna do mpz_set_str. Jeżeli parametrem str jest NULL, to ciąg znaków zostaje zwrócony przez funkcję. W przeciwnym razie funkcja umieszcza go pod adresem, na który

wskazuje zmienna str. Należy zadbać o to, by bufor, do zapisania rezultatu funkcji, był wystarczający. Powinien on być 2 bajty dłuższy niż długość zwróconej liczby w danym systemie liczbowym. Pierwszy bajt to służy na wpisanie ewentualnego znaków minus, natomiast drugi na znak ‘ 0’ kończący łańcuch znaków. Długość zasadniczej części liczbowej można pobrać używając funkcji mpz_sizeinbase (op, base), w której podajemy odpowiednio daną liczbę oraz podstawę.

Funkcje arytmetyczne

```
void mpz_add (mpz_t rop, mpz_t op1, mpz_t op2)
```

Ustawia wartość rop jako sumę op1 i op2.

```
void mpz_sub (mpz_t rop, mpz_t op1, mpz_t op2)
```

Ustawia wartość rop jako różnicę op1 i op2.

```
void mpz_mul (mpz_t rop, mpz_t op1, mpz_t op2)
```

Ustawia wartość rop jako iloczyn op1 i op2.

```
void mpz_addmul (mpz_t rop, mpz_t op1, mpz_t op2)
```

Oblicza wartość iloczynu czynników op1 i op2, a następnie zwiększa wartość rop o otrzymany wynik. Tożsame z wykonaniem działania rop = rop + op1*op2.

```
void mpz_submul (mpz_t rop, mpz_t op1, mpz_t op2)
```

Oblicza wartość iloczynu czynników op1 i op2, a następnie zmniejsza wartość rop o otrzymany wynik. Tożsame z wykonaniem działania rop = rop - op1*op2.

```
void mpz_neg (mpz_t rop, mpz_t op)
```

Jako rezultat ustawia liczbę przeciwną do danej. Gdy rop i op są tą samą zmienną to mamy do czynienia z przykładem operacji w miejscu. Wówczas całe funkcja jest bardzo szybka, gdyż opiera się tylko na zmianie bitu, mówiącym o znaku danej liczby.

```
void mpz_abs (mpz_t rop, mpz_t op)
```

Funkcja w swoim działaniu bardzo podobna do funkcji mpz_neg. Jedyną róznicą jest to, że bit znaku nie zostaje zanegowany, a ustawiony tak, by wskazywał na wartość nieujemną.

Funkcje dzielenia

Uzywając niżej wymienionych funkcji należy pamiętać, ze dzielenie przez 0, zarówno w przypadku obliczania wartości ilorazu, jak i reszty z dzielenia jest nielegalne. W przypadku, gdy wystąpi taka sytuacja, biblioteka zachowa się jak w przypadku dzielenia przez 0, dla liczb typu int, tzn. skończy działanie funkcji, rzucając odpowiedni wyjątek.

```
void mpz_tdiv_q (mpz_t q, mpz_t n, mpz_t d)
```

Funkcja oblicza iloraz z liczb n i d. Litera przed ‘div’ w nazwie funkcji mówi, o jej zachowaniu w przypadku, gdy rezultat nie jest liczbą całkowitą. Symbol ‘t’ (truncate), oznacza obcięcie części ułamkowej. Zalecanym użyciem jest przypadek, gdy nie ma znaczenia wartość reszty z dzielenia.

```
void mpz_tdiv_r (mpz_t r, mpz_t n, mpz_t d)
```

Oblicza wartość rezsty z dzielenia liczb n i d. Podobnie jak w fukcji mpz_tdiv_q, mamy do czynienia z przypadkiem obcięcia ewentualnej części ułamkowej ilorazu. Oznacza to, że zwracana reszta zawsze będzie tego samego znaku co zmienna n. Rekomendowana, gdy wartość ilorazu z dzielenia nie jest istotna.

```
void mpz_tdiv_qr (mpz_t q, mpz_t r, mpz_t n, mpz_t d)
```

Łączy w sumie dwie powyższe funkcje. Ze względów wygody i wydajności zaleca się jej użycie wówczas, gdy potrzebujemy obliczyć zarówno iloraz, jak i resztę z dzielenia. Ważne jest by pamiętać, że przekazane argumenty q i r, muszą być referencjami do różnych zmiennych. W przeciwnym razie, działanie funkcji będzie nie poprawne, a wynik najprawdopodobniej błędny.

Oprócz wyżej wymienionych funkcji, istnieją także ich odpowiedniki, o odmiennym zachowaniu w kwestii zaokrąglania części ułamkowej ilorazu. Pierwszym z nich jest grupa funkcji, posiadająca w nazwie „cdiv", gdzie litera 'c' (ceil) oznacza sufit. Oznacza to, że w razie potrzeby iloraz zostanie zaokrąglony w górę. Wówczas otrzymana reszta r będzie przeciwnego znaku do liczby d. Funkcjami z tej grupy są: mpz_cdiv_q, mpz_cdiv_r oraz mpz_cdiv_qr. Drugim odpowiednik jest grupa funkcji, posiadająca w nazwie „fdiv", gdzie 'f' (floor) oznacza podłogę. Oznacza to, że w razie potrzeby wynik zostanie zaokrąglony w dół. Reszta r będzie w tym przypadku tego samegu znaku co liczba d. Przykładami funkcji z tej grupy są: mpz_fdiv_q, mpz_fdiv_r oraz mpz_fdiv_qr. Zauważmy, że we wszystkich trzech przypadkach zostaje spełnione równanie n = q * d + r, gdzie 0 <= |r| < |d|.

```
int mpz_divisible_p (mpz_t n, mpz_t d)
```

Funkcja sprawdza, czy d dzieli liczbę n. Jeżeli nie, to zwracana 0, w przeciwnym razie zwraca wartość różną od zera.

```
void mpz_divexact (mpz_t q, mpz_t n, mpz_t d)
```

Oblicza iloraz z dzielenia liczb n przez d. Działa poprawnie tylko w przypadku, gdy n jest podzielna przez d. By to sprawdzić można uzyć funkcji mpz_divisable_p. Jest znacznie szybsza niż pozostałe funkcje, umożliwiające obliczanie ilorazu z dzielenia. Dlatego zawsze, gdy spełniony jest powyższy warunek, użycie tej funkcji jest wysoko rekomendowane.

Funkcje porównania

```
int mpz_cmp (mpz_t op1, mpz_t op2)
```

Porównuje wartości dwóch liczb. Zwraca wartość dodatnią, gdy op1 > op2, ujemną, gdy op1 < op2, oraz 0, gdy obie wartości są równe.

```
int mpz_cmpabs (mpz_t op1, mpz_t op2)
```

Porównuje wartość bezwzględną dwóch liczb. Zwraca wartość dodatnią, gdy |op1| > |op2|, ujemną, gdy |op1| < |op2|, oraz 0, gdy obie wartości są równe lub przeciwne.

```
int mpz_sgn (mpz_t op)
```

Określa znak danej liczby. Zwraca 1 dla wartości dodatnich, -1 dla wartości ujemnych oraz 0 w przypadku zera. Należy pamiętać, ze w obecnej implementacji mpz_sgn jest makrem i rozpatruje dany argument wielokrotnie.

Pozostałe funkcje

```
void mpz_pow_ui (mpz_t rop, mpz_t base, mpir_ui exp)
```

Podnosi liczbę base do potęgi exp.

```
void mpz_sqrt (mpz_t rop, mpz_t op)
```

Oblicza pierwiastek całkowity dla danej liczby. W przypadku, gdy wynik nie jest liczbą całkowitą, część ułamkowa zostaje obcięta.

```
void mpz_sqrtrem (mpz_t rop1, mpz_t rop2, mpz_t op)
```

Jest bardzo podobna do funkcji mpz_sqrt. Poza obliczeniem wartości pierwiastka z danej liczby, zwraca także różnicę pomiędzy daną liczbą a kwadratem tego pierwiastka. Rop1 = sqrt(op) Rop2 = op – sqrt(rop1)$\hat{}$2. Zauważmy, że jeżeli rop2 = 0, to rop1 jest pierwiastkiem kwadratotowym z op.

Operacje na liczbach rzeczywistych

W tym podrozdziale, rozpatrzę funkcje dla zmiennych typu mpq_t. Skupię się w nim na funkcjach, które nie występują dla liczb całkowitych typu mpz_t, opisanych w poprzednim podrozdziale lub ich implementacja znacząco się różni, od już przedstawionej. Na początku warto zaznaczyć, że zmienne mpq_t reprezentują wyłącznie liczby wymierne. Jest to spowodowane ich implementacją. Każda liczba typu mpq_t składa się z dwóch liczb całkowitych typu mpz_t, reprezentujących licznik i mianownik.

```
void mpq_init (mpq_t dest_rational)
```

Funkcja alokuje miejsce w pamięci i inicjuje wartość danej liczby, ustawiając licznik na 0 i mianownik na 1.

```
void mpq_canonicalize (mpq_t op)
```

Znajduje wspólne dzielniki licznika i mianownika, skracając ułamek. Dodatkowo zapewnia, że mianownik jest liczbą dodatnią, w razie potrzeby, mnożąc licznik i mianownik przez liczbę -1.

```
int mpq_set_str (mpq_t rop, char *str, int base)
```

Tworzy liczbę na podstawie wartości podanej w łańcuchu znaków. Jej wartość podawana jest w postaci najpierw licznik, a następnie mianownik, na zasadach podobnym jak w funkcji mpz_set_str. Obie liczby oddziela znak operatora dzielenia '/'. Funkcja zwraca wartość 0 w przypadku gdy cały łańcuch wejściowy jest poprawny oraz -1 w innym przypadku. Jeżeli argument base jest równy 0, to format liczbowy, dla licznika i mianownika, jest ustalany osobno. Oznacza to, że

możemy podać liczby w dwóch różnych formatach, np. „0xFF/256” . Warto zauważyć, że funkcja mpq_canonicalize nie jest wołana automatycznie, więc jeżeli podaliśmy ułamek, który chcemy skrócić, to musimy pamiętać o jej wywołaniu.

```
char * mpq_get_str (char *str, int base, mpq_t op)
```

Funkcja zwracająca liczbę w postaci ułamka, tzn. licznik i mianownik, oddzielony znakiem kreski ułamkowej. Jest analogiczna do funkcji mpz_get_str. Bufor wyjściowy, w którym chcemy umieścić rezultat funkcji powinien być, wystarczający i wynosić mpz_sizeinbase(mpq_numref(op), base) + mpz_sizeinbase (mpq_denref(op), base) + 3. Jeden dodatkowy bajt w porównaniu do funkcji mpz_get_str, spowodowany jest koniecznością umieszczenia w buforze kreski ułamkowej.

.

.

Rozdział 3

Opis Rozwiązania

3.1 Podział na moduły

3.2 Główne klasy

3.3 Główne funkcje

Parser Klasa umożliwiająca użytkownikom podanie wielomianów poprzez standardowe wejście. Składa się tylko z dwóch metod. Zadaniem pierwszej z nich jest unifikacja wielomianu podanego na wejściu. Funkcja jako argument przyjmuje pojedynczy obiekt klasy string. Po dokonaniu odpowiednich operacji, jako rezultat funkcji, zwraca również typ string. Użycie funkcji ma na celu ułatwienie przetwarzania w kolejnym etapie, w którym na podstawie podanego ciągu znaków, tworzony będzie obiekt wielomianu. Głównym zadaniem funkcji, jest weryfikacja, czy ciąg znaków podanych na wejściu reprezentuje poprawny składniowo wielomian. Weryfikowana jest liczba nawias otwierających i zamykających oraz fakt, czy w każdym miejscu wyrażenia liczba otwartych nawiasów jest nie mniejsza niż liczba nawiasów zamkniętych. Dodatkowo sprawdzane jest, czy na sąsiednich miejscach nie występują dwa operatory. Dodatkowo poprawne wyrażenie nigdy nie kończy się operatorem, a zaczynać może się tylko minusem, literałem, cyfrą lub nawiasem otwierającym. W czasie unifikacji wyrażenia, ignorowane są wszystkie występujące w nim białe znaki. Warto zaznaczyć, że dopuszczalny jest tylko jeden znak, reprezentujący zmienną wielomianu. Nie ma ograniczeń co do wartości tego znaku, może być to znak ‘a’, ‘x’, lub jakikolwiek chcemy, ale powinniśmy zadbać, by w całym wyrażeniu występował on w tej samej postaci. Sporym ułatwieniem w interfejsie jest brak konieczności wpisywania operatorów mnożenia (*) i potęgowania ($\hat{}$) w oczywistych miejscach. Z punktu widzenia aplikacji wyrażenia 4x oraz 4*x są identyczne. Podobnie jest w przypadku x3 oraz x$\hat{3}$. Funkcja przeanalizuje wyrażenie i zwróci je w odpowiedniej postaci. Dla przykładu wyrażenia x3+2x2+3x+1, x$\hat{3}$ +2x$\hat{2}$ + 3x+1 zostaną zamienione w x$\hat{3}$+2*x$\hat{2}$+3*x+1.

3.4 Zewnętrze biblioteki

3.5 Instrukcja programu

Rozdział 4

Przeprowadzone testy

4.1 Testy jednostkowe

4.2 Testy interfejsu użytkownika

4.3 Testy wydajności czasowej

4.4 Testy wydajności pamięciowej

Rozdział 5

Podsumowanie

Bibliografia

[1] E.J. Barbeau. *Polynomials*. Problem Books in Mathematics. Springer New York, 2003.

[2] D. Buell. *Algorithmic Number Theory: 6th International Symposium, ANTS-VI, Burlington, VT, USA, June 13-18, 2004, Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004.

[3] R.L. Burden, J.D. Faires, and A.M. Burden. *Numerical Analysis*. Cengage Learning, 2015.

[4] L.N. Childs. *A Concrete Introduction to Higher Algebra*. Undergraduate Texts in Mathematics. Springer New York, 2012.

[5] T. Granlund and G.D. Team. *Gnu MP 6.0 Multiple Precision Arithmetic Library*. Samurai Media Limited, 2015.

[6] W. Kryszewski. *Wykład analizy matematycznej, cz. 1: Funkcje jednej zmiennej:*. Number pkt 1. Wydawnictwo Naukowe UMK, 2014.

[7] D.S. Malik. *Data Structures Using C++*. Cengage Learning, 2009.

[8] Polskie Towarzystwo Matematyczne. *Wiadomości matematyczne*. Number t. 10-11. Państwowe Wydawn. Naukowe., 1968.

[9] J.M. McNamee. *Numerical Methods for Roots of Polynomials -*. Number pkt 1 in Studies in Computational Mathematics. Elsevier Science, 2007.

[10] T. Mora. *Solving Polynomial Equation Systems I: The Kronecker-Duval Philosophy*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2003.

[11] V. Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Birkhäuser Boston, 2012.

[12] W. Sierpiński and A. Mostowski. *Zasady algebry wyższej, z przypisem Andrzeja Mostowskiego Zarys teorii Galois*. Monografie Matematyczne. Nakładem Polskiego Tow. Matematycznego, 1951.

[13] Mieczysław (1918-2007) Warmus and Józef (1927-) Łukaszewicz. Metody numeryczne i graficzne. cz. 1.