|  |
| --- |
| PI |
| Dokumentacja techniczna |
| [Wpisz nazwisko autora] |

Spis treści

[1. Wstęp 4](#_Toc194319920)

[2. Funkcjonalność programu 4](#_Toc194319921)

[3. Algorytmy 4](#_Toc194319922)

[3.1. Algorytmy operacji na dużych liczbach 4](#_Toc194319923)

[3.2. Algorytm generacji π 6](#_Toc194319924)

[3.3. Algorytm kontynuacji obliczania π 6](#_Toc194319925)

[3.4. Algorytm generacji funkcji 8](#_Toc194319926)

[3.4.1. Algorytm wyszukiwania pojedynczego wzorca 8](#_Toc194319927)

[3.4.2. Algorytm wyszukiwania grupy wzorców 11](#_Toc194319928)

[4. Interfejs użytkownika 13](#_Toc194319929)

[4.1. Interfejs użytkownika części programu generującego liczbę 14](#_Toc194319930)

[4.2. Interfejs użytkownika części programu generującego funkcję 16](#_Toc194319931)

[4.3. Interfejs użytkownika kalkulatora dużych liczb 20](#_Toc194319932)

Dane Dokumentu

|  |  |  |  |
| --- | --- | --- | --- |
| Nazwa Projektu: | PI – analiza techniczna | | |
| Opracowany Przez: | Piotr Witosławski | Nr Wersji: | 1.0.0 |
| Rola: | Koordynator | Data Wersji: | 25-03-2008 |
| Sprawdzony Przez: | Piotr Witosławski | Data Przeglądu: | 25-03-2008 |

Historia Zmian

| Nr Wersji | Data Wersji | Zmiany Wprowadził(a) | Opis Zmian | Recenzent | Status |
| --- | --- | --- | --- | --- | --- |
| 0.1.0 | 18-03-2008 | Piotr Witosławski | Utworzenie dokumentu |  |  |
| 0.2.0 | 21-03-2008 | Adam Nowacki | Dodanie sekcji „Algorytmy operacji na dużych liczbach” |  |  |
| 0.3.0 | 23-03-2008 | Adam Nowacki | Dodanie sekcji „Interfejs użytkownika” |  |  |
| 0.4.0 | 25-03-2008 | Piotr Witosławski | Dodanie algorytmów przeszukiwania |  |  |
| 0.5.0 | 25-03-2008 | Paweł Rokoszny | Algorytmy generowania pi |  |  |
| 1.0.0 | 25-03-2008 | Piotr Witosławski | Drobne poprawki | Piotr Witosławski | Poprawny |

1. Wstęp

Dokument stanowi analizę techniczną projektu PI realizowanego w ramach przedmiotu Teoria Algorytmów i Obliczeń. Oprócz opisu podstawowych algorytmów zastosowanych w projekcie zawiera on uzupełnienie zakresu funkcjonalności programu oraz opisu interfejsu użytkownika w stosunku do wersji zaprezentowanej w analizie biznesowej.

1. Funkcjonalność programu
   1. **Obliczenia programu**

Zarówno w części zajmującej się generacją liczby jak i w części programu poświęconej znajdowaniu argumentów funkcji, będzie istnieć możliwość rozpoczęcia obliczeń bez warunków ich zatrzymania. Zakończenie obliczeń umożliwi wciśnięcie przycisku STOP.

Algorytm znajdowania wartości funkcji wzbogacony zostanie o możliwość degenerowania liczby , gdy zajdzie taka potrzeba, (tzn. gdy poszukiwany argument funkcji nie został odnaleziony w aktualnym zakresie). Przed rozpoczęcie procesu dogenerowywania rozwinięcia program zapyta użytkownika o zgodę.

Wszystkie obliczenia w programie wykorzystują w wysokim stopniu pamięć operacyjną komputera, dlatego zostanie położony szczególny nacisk na uniknięcie błędów jej przepełnienia (*OutOfMemoryException*). Gdy wystąpi taki błąd, program wyświetli stosowny komunikat i zakończy działanie (zwalniając zasoby).

Format pliku, do którego zapisywana będzie liczba będzie zgodny z ustalonym przez inne zespoły.

* 1. **Kalkulator**

Realizowana aplikacja zawierać będzie także opcje kalkulatora. Kalkulator będzie umożliwiał załadowanie dwóch bardzo dużych liczb z plików lub ich wpisanie ręczne. Do funkcji należeć będzie:

* Porównywanie liczb
* Operacje na liczbach całkowitych: +, -, \*, \

Istnieć będzie możliwość zapisania wyniku obliczeń do pliku.

Wygląd i interfejs użytkownika kalkulatora zostanie zaprezentowany w paragrafie 4.3

# Algorytmy

## Algorytmy operacji na dużych liczbach

W obliczeniach związanych, z generowanie liczny π oraz wszelkich innych operacjach na dużych liczbach w projekcie skorzystamy z darmowej biblioteki **GMP** (GNU Multiple Precision Arithmetic) o otwartych źródłach, opartej na licencji GPL (GNU General Public License). GMP jest powszechnie stosowaną biblioteką do obliczeń na liczbach naturalnych, wymiernych i rzeczywistych (zmiennoprzecinkowych). GMP jest najszybszą[[1]](#footnote-2) z obecnie dostępnych, biblioteką do przeprowadzania obliczeń na dużych liczbach. Szybkość działania biblioteki została uzyskana poprzez użycie odpowiednich algorytmów oraz optymalizację kodu dla różnych typów procesorów. Jedynym ograniczeniem, dotyczącym precyzji obliczeń, jest ilość dostępnej pamięci na komputerze, na którym zostanie ona użyta (patrz 2.1).

Poniżej, aby przybliżyć ogólne idee tej biblioteki zostaną opisane tylko wybrane algorytmy znajdujące zastosowanie w generacji liczby π. Pozostałe informacje, w tym m.in. pełny opis konstrukcji zastosowanych w niej algorytmów można znaleźć na stronie <http://gmplib.org/manual/>.

W projekcie, w trakcie obliczeń będziemy operować na liczbach całkowitych oraz zmiennoprzecinkowych. Każda zmienna typu zmiennoprzecinkowego[[2]](#footnote-3) ma swoją indywidualną precyzją, która poprzez odpowiednie funkcje może być zwiększana lub zmniejszana w czasie wykonywania programu. Jej rozmiar ustawiony w danej chwili jest jednak wartością minimalną - w trakcie obliczeń możliwe jest automatyczne zwiększenie precyzji, w celu przeprowadzenia dokładniejszego wyliczenia.

Wykładnik liczby zmiennoprzecinkowej posiada natomiast stale ustaloną precyzję – na większości komputerów jest to jedno słowo maszynowe, oznacza to, że np. na komputerach 32-bitowych jest on z zakresu w przybliżeniu od -268719476768 do 268719476736 (odpowiednio więcej na komputerach 64-bitowych).

Każda zmienna przechowuje rozmiar swojej aktualnie używanej mantysy. Dzięki temu, jeśli liczba zmiennoprzecinkowa jest reprezentowana tylko przez kilka bitów, to tylko te kilka bitów zostanie użytych w obliczeniach, nawet przy wysoko ustawionej precyzji. Mantysa liczby przechowywana jest w pamięci, (co wydaje się naturalnym rozwiązaniem) w postaci binarnej, wynika to między innymi z faktu, że precyzja ustalana dla danej liczby również wyrażana jest w postaci bitowej.

Często w użytych do generacji π algorytmach będzie wykorzystywane mnożenie liczba całkowitych. Biblioteka GMP w zależności od ustawień oraz zastosowanych metod optymalizacji stosuje jedną z poniższych metod mnożenia liczba całkowitych (jedna z nich została uszczegółowiona[[3]](#footnote-4) dla lepszego przybliżenia cech i sposobu działania użytej biblioteki, szczegóły dotyczące pozostałych można znaleźć w oficjalnej dokumentacji[[4]](#footnote-5)):

* Algorytm Karatsuba - do pomnożenia dwóch *n*-cyfrowych liczb *x* i *y* przy podstawie *B*, gdzie *n=2m* (jeśli *n* jest nieparzyste, albo *x* ma różną liczbę cyfr niż *y*, można to naprawić dodając zera po lewej stronie tych liczb), rozpisujemy je jako:



*x* = *x*1 *Bm* + *x*0

*y* = *y*1 *Bm* + *y*0

*xy* = (*x*1*Bm* + *x*0)(*y*1*Bm* + *y*0) = *x*1*y*1*B*2*m* + (*x*1*y*0 + *x*0*y*1)*Bm* + *x*0*y*0

Standardową metodą byłoby pomnożenie czterech czynników osobno i dodanie ich po odpowiednim przesunięciu. Daje to algorytm działający w czasie O(n2). Karatsuba zauważył jednak, że można ten sam wynik uzyskać wykonując tylko trzy mnożenia:

**

*X* = *x*1*y*1

*Y* = *x*0*y*0

*Z* = (*x*1 + *x*0)(*y*1 + *y*0) - X - Y

Dostajemy wtedy

*Z* = (*x*1*y*1 + *x*1*y*0 + *x*0*y*1 + *x*0*y*0) - *x*1*y*1 - *x*0*y*0 = *x*1*y*0 + *x*0*y*1

A zatem *xy* = *X* *B*2m + *Y* + *Z* *B*m, tym samym kosztem kilku dodawań i odejmowań można zmniejszyć liczbę mnożeń z czterech do trzech.

* Algorytm Toom 3
* Algorytm FFT, oparty na szybkiej transformacie Fouriera

## Algorytm generacji π

Algorytm, który może być zastosowany w takim wymagającym obliczeniowo zadaniu musi spełniać wiele różnych warunków. Przede wszystkim musi wyznaczać kolejne przybliżenia liczby z dużą precyzja. Nie może to być wyznaczanie liczby z przybliżeniem, jak np. 22/7.

Jego złożoność obliczeniowa i pamięciowa musza być jak najmniejsze. Optymalnie byłoby, gdyby złożoność obliczeniowa była rzędu *O(n)*, a pamięciowa – *O(1)*, gdzie *n* oznacza precyzję obliczeń w cyfrach po przecinku. Jednak takiego algorytmu (jeszcze) nie ma. Po szeroko zakrojonych poszukiwaniach zdecydowano się zastosować algorytm wyznaczania liczby opracowany przez braci Chudnowsky’ich. Jest to podstawowy algorytm wykorzystywany w profesjonalnych aplikacjach matematycznych, m.in. w *Mathematice*.

Idea algorytmu jest bardzo prosta. W 1987 roku bracia Chudnovsky zauważyli, ze „ jest bardzo bliskie zeru”. Po skomplikowanych obliczeniach w dziedzinie zespolonej doszli do następującej równości:

|  |  |  |  |
| --- | --- | --- | --- |
| 1/pi | = | 12sum_(n=0)^(infty)((-1)^n(6n)!(13591409+545140134n))/((n!)^3(3n)!(640320^3)^(n+1/2)) |  |
| http://mathworld.wolfram.com/images/equations/PiFormulas/Inline220.gif | = | (163·8·27·7·11·19·127)/(640320^(3/2))sum_(n=0)^(infty)((13591409)/(163·2·9·7·11·19·127)+n)((6n)!)/((3n)!(n!)^3)((-1)^n)/(640320^(3n)) |  |

Po odwróceniu powyższego równania otrzymujemy „przepis” na . „Przepis”, gdyż trudno mówić o tym równaniu jako o algorytmie obliczeń. Jest to szereg, w którym dodając kolejne wyrazy można przybliżać liczbę o 14 następnych cyfr rozwinięcia dziesiętnego. Tak się też dzieje zarówno w profesjonalnych programach w dziedzinie matematyki, jak i w konstruowanej przez nas aplikacji.

Należy też powiedzieć, że aby optymalnie wykorzystać zasoby sprzętowe oraz algorytmy, które są implementowane przez profesjonalne biblioteki dużych liczb, nie należy liczyć każdej iteracji jedna po drugiej, ale wykonywać operacje seryjnie, np. dodawanie, mnożenie, dzielenie, potęgowanie. Takie podejście pozwala między innymi na wykorzystanie pełnych możliwości współczesnych procesorów, jak np. wykorzystanie techniki ssl, wieloprocesowości czy też strumieniowej obróbki danych.

Jednocześnie podejście takie w sposób oczywisty wyklucza możliwość zapamiętania np. ostatniej iteracji naszego sumowanego szeregu i późniejsze „dogenerowywanie” kolejnych cyfr.

## Algorytm kontynuacji obliczania π

Aplikacja przez nas implementowana, jako jedno z podstawowych założeń, ma zakładaną funkcjonalność „dogenerowywania” rozwinięcia liczby . Przy rozpatrywaniu tego zagadnienia skupimy się na dwóch alternatywnych podejściach, z których do dalszej implementacji wybierzemy to, które będzie oferowało efektywniejsze obliczenia.

1. „Dogenerowywanie” n-tej cyfry liczby .

Jedyny znany algorytm obliczania n-tej cyfry liczby bez znajomości cyfr poprzednich, działający w systemie dziesiętnym, został opracowany przez Simona Plouffe’a, który też brał udział w opracowaniu (podobnego) algorytmu BBP w systemie hexadecymalnym. Algorytm ten ma asymptotyczna złożoność , skutkiem czego jest bardzo wolny. My zaimplementujemy jego modyfikacje, zaprezentowana przez Fabrice’a Belarda, o złożoności obliczeniowej .

Zauważył on, ze liczba może być zapisana jako suma szeregu:

Mając teraz liczbę pierwsza *p* można zapisać:

,

gdzie jest mnożeniem p przez n.

Wynika to z następującej relacji:

Stad, jeśli chcemy obliczyć n-tą cyfrę liczby w bazie B, musimy użyć następującego algorytmu:

* , gdzie - jest mala liczba naturalna użyta, aby zapewnić żądaną precyzje
* Dla każdej liczby pierwszej : należy wykonać:
* Dla *k* od 1 do *2N*:
* Jeśli to:
* , wtedy jeśli zaniedbamy błędy zaokrągleń, to . *q* jest liczbą poprawnie obliczonych cyfr i zależy ściśle od .

Algorytm ten działa w czasie ponieważ jest liczb pierwszych pomiędzy *2* i *2N*. Złożoność pamięciowa rzędu *O(1)*.

1. Istnieje też możliwość, ze bardziej opłacalne, jeśli chodzi o złożoność czasową, będzie sumowanie po kolejnych wyrazach omówionego wcześniej szereg, który został opracowany przez braci Chudnovsky’ich. Wtedy należy zrezygnować z optymalizacji kolejności wykonania operacji i wykonywać je tak, jak to wynika z kolejności wykonywania działań dla opracowywanego szeregu. Wtedy, jeśli zachowana zostanie wystarczająca precyzja ze strony biblioteki dużych liczb, możliwe powinno być dodanie kolejnych wyrazów szeregu tak, aby w efekcie otrzymać kolejne, znacznie dokładniejsze przybliżenie .

To, który ze sposobów na „dogenerowywanie” rozwinięcia dziesiętnego liczby będzie użyty, będzie uzależnione od ich wydajności i nie rozstrzygnięte do czasu przeprowadzenia testów w trakcie implementacji.

## Algorytm generacji funkcji

Algorytm generacji funkcji:

Można sprowadzić do problemu wyszukiwania wzorca w tekście, gdzie tekst stanowią kolejne cyfry rozwinięcia , natomiast wyszukiwane wzorce to kolejne cyfry liczb z przedziału [a; b]. Algorytmy wyszukiwania wzorca dzielimy na takie, które w efektywny sposób szukają pojedynczego wzorca oraz na takie, które szukają zbioru wzorców.

### Algorytm wyszukiwania pojedynczego wzorca

#### Algorytm naiwny

Podstawowym algorytmem wyszukiwania wzorca jest algorytm naiwny. Dane wejściowe to wzorzec x o długości m, oraz przeszukiwany tekst t o długości n.

1. Przyjmij i := 0, jest to aktualna pozycja w przeszukiwanym tekście
2. Dopóki i < n – m + 1 wykonuj:
   1. Jeśli x[0..m-1] = y[i..i+m-1] to znaleziono wzorzec na pozycji i (zakończ algorytm)
   2. i = i + 1

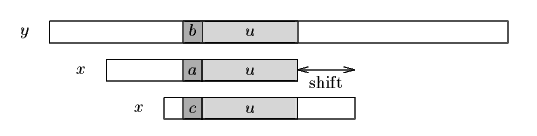
Algorytm ten jest rzędu θ(n\*m), gdyż każdą z liter tekstu porównujemy m razy (z każdą z liter wzorca)

#### Algorytm Turbo Boyera – Moore’a

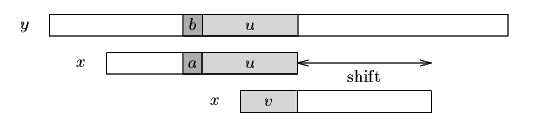
Algorytm Boyera – Moore’a jest modyfikacją algorytmu naiwnego: różni się sposobem porównywania oraz sposobem przesuwania aktualnego indeksu w tekście. Algorytm przeszukuje znaki wzorca od prawej do lewej począwszy od znaku położonego najbardziej po prawej. W przypadku niedopasowania korzysta z dwóch funkcji przesunięcia okna poszukiwań w prawo – te funkcji nazywane są: good-suffix shift oraz bad character shift.

Przypuśćmy, że niedopasowanie pojawia się między znakiem x[i]=a wzorca i znakiem y[i+j]=b tekstu w czasie próby dopasowania na pozycji j. Wtedy *x*[*i*+1 .. *m*-1]=*y*[*i*+*j*+1 .. *j*+*m*-1]=u and

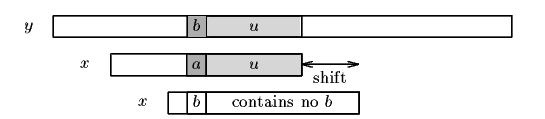
*x*[*i*] ≠ y[*i*+*j*]. Funkcja good-suffix shift polega wtedy na wyrównaniu segmentu u do jego położonego najbardziej po prawej w x wystąpienia poprzedzonego znakiem różnym od x[i].



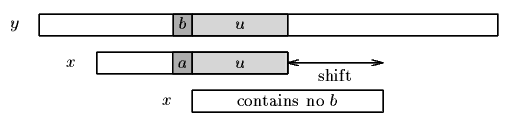
Jeśli taki podciąg nie istnieje, przesuniecie polega na wyrównaniu najdłuższego sufiksu v z *y*[*i*+*j*+1 .. *j*+*m*-1] do pasującego prefiksu z x.



Bad-character shift polega na wyrównaniu znaku tekstu y[i+j] do jego położonego najbardziej na prawo wystąpienia w x[0..m-2].



Jeśli znak y[i+j] nie występuje we wzorcu x, wtedy lewy brzeg okna porównania jest wyrównywany ze znakiem y[i+j+1]



Wartości pomagające w obliczeniu good-suffix shift przechowywane są w specjalnej tablicy (bmGS) rozmiaru m+1. Aby określić jej zawartość zdefiniujmy dwa warunki:

* + 1. Cs(i,s): dla każdego k, takiego, że i < k < m, s >= k x[k-s]=x[k] (warunek identyczności sufiksu)
    2. Co(i,s): jeśli s < i wtedy x[i-s] != x[i] (różny znak w odległości s)

Wtedy dla 0 <= i < m bmGs[i+1] = min{s > 0: Cs(i,s) i Co(i,s) są spełnione}

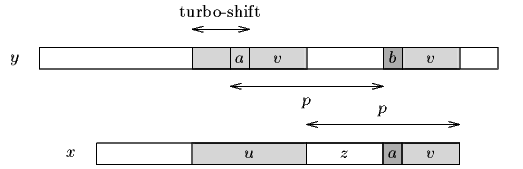
Wartości pomocnicze dla bad-character shift również przechowywane są w specjalnej tablicy (bmBS), której długość jest równa ilości różnych znaków x. Tablica jest zdefiniowana następująo

:

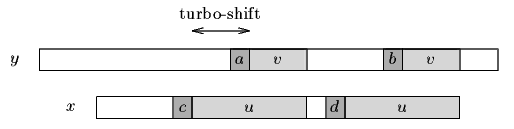
bmBc[c] = min{i : 1  <= i <m-1 and x[m-1-i]=c} jeśli c występuje w x.

Dla pozostałych znaków wartość tablicy wynosi m.

Definiuje ona indeks najwcześniejszego wystąpienia każdej z liter alfabetu we wzorcu licząc od prawej strony. Algorytm Turbo w stosunku do zwykłego algorytmu Boyera – Moore’a jest rozbudowany o jeszcze jedno przesunięcie, zwane przesunięciem turbo, które może nastąpić jedynie po przesunięciu good-suffix shift. Przesunięcie turbo może wystąpić gdy podczas próby dopasowania sufiks wzorca, który pasuje do tekstu, jest krótszy od tego zapamiętanego w poprzedniej próbie. W tym przypadku niech u będzie zapamiętanym czynnikiem a v sufiksem dopasowanym w aktualnej próbie, takim, że uzv jest sufiksem x. Niech a i b będą znakami, które powodują niedopasowanie w aktualnej próbie, odpowiednio we wzorcu i tekście. Wtedy av jest sufiksem x a także u ponieważ |v| < |u| (a także dlatego, że poprzednie przesunięcie było przesunięciem good-suffix),. Znaki a i b występują w tekście w odległości p i sufiks x o długości |uzv| posiada fragment o długości p =|zv| ponieważ u jest fragmentem granicznym w |uzv| i dlatego nie może nachodzić obu wystąpień dwóch różnych znaków a i b w odległości p w tekście. Najmniejsze możliwie przesunięcie ma długość |u| - |v| i nazywamy je przesunięciem turbo.



W przypadku, gdy |v| < |u| jeśli długość bad-character shift jest większe niż długość good-suffix shift oraz od długości turbo przesunięcia, wtedy długość aktualnego przesunięcia musi być niemniejsza od |u|+1. Jest tak, gdyż dwa znaki c i d są różne, ponieważ przyjęliśmy, że poprzednie przesunięcie było przesunięciem typu good-suffix.



Wtedy przesunięcie turbo większe od przesunięcia turbo, ale mniejsze od |u|+1 wyrównałoby c i d do tego samego znaku w v.

Algorytm przedstawia się następująco:

1. Oblicz tablice Bad-character Shift oraz good-suffix Shift, na podstawie wzorów i zapisz w tablicach bmGs oraz BBC
2. Oznacz j jako aktualną pozycję przeszukiwania tekstu i przyjmij wartość 0
3. Oznacz u jak długość ostatniego bloku i przyjmij wartość 0
4. Oznacz shift jako długość ostatniego przesunięcia i przyjmij wartość m
5. Dopóki j <= n – m wykonuj
   1. Oznacz jak i aktualną pozycję przeszukiwania wzorca i przyjmij wartość skrajnie prawą (m-1)
   2. Dopóki aktualnie dopasowywane znaki x i y się zgadzają wykonuj
      1. Zmniejsz i od 1
      2. Jeśli u != 0 oraz i == m – 1 – shift, to i = i – u
   3. Jeśli i < 0, oznacza to, że znaleziono wystąpienie x na pozycji j, należy zakończyć algorytm
   4. Oznacz v jako długość aktualnie przeszukanego bloku i przyjmij jego długość jako m – 1 –i
   5. Oznacz jako wartość przesunięcia turbo u – v i umieść w zmiennej turboShift
   6. Oznacz jako wartość przesunięcia Bad – charakter wartość bmBc[y[i + j]] – v
   7. Przyjmij jako wartość shift największą spośród wartości turboShift, Bad-character shift oraz bmGs[i] (good-suffix shift)
   8. Jeśli największą z wartości jest wartość good-suffix, to uaktualnij długość u jako większą spośród wartości m – shift oraz v (pierwszy wariant turboShift)
   9. W przeciwnym przypadku przyjmij u = 0 i jeśli turboShoft < bcShift to shift jest wartością większą spośród u+1 oraz shift (drugi wariant turboShift)
   10. Zwiększ j o wartość shift

Powyższy algorytm jest jednym z najszybszych algorytmów do przeszukiwania tekstów. Faza wstępnego przetwarzania jest rzędu 0(m+σ), faza przeszukiwania jest rzędu 0(n), a także Ω(n/m) i dokonywanych jest co najwyżej 2n porównań.

### Algorytm wyszukiwania grupy wzorców

#### Algorytm bezpośredniego wypisywania

Algorytm ten opiera się na obserwacji polegającej na tym, że na początku przeszukiwań niemal na każdej pozycji w przeszukiwanej liczbie PI otrzymamy pewne dopasowanie do liczby z przedziału, o ile ten przedział jest odpowiednio duży. Niech przedział z którego szukamy liczb będzie [a;b]. Niech ilość cyfr w liczbie a wynosi k, a w liczbie b – l. Idea algorytmu opiera się na tym, że z pozycji j w liczbie PI pobieramy podciąg o długości l i sprawdzamy czy taka liczba występuje w przedziale i nie została jeszcze odnaleziona. Jeśli tak, to dołączamy ją do wyników, w przeciwnym przypadku odrzucamy skrajnie prawą cyfrę i sprawdzamy czy liczba na pozycjach od j do l-2 występuje w grupie nie znalezionych cyfr o długości l-2. Kontynuujemy ten proces, aż do liczb o długości cyfr wynoszącej 1, następnie przesuwamy okno wyszukiwania o jedną pozycję w prawo.

Przy implementacji tego algorytmu będą niezbędnie rozpatrzenie dwóch kwestii pobocznych:

1. Czy przeszukiwanie liczb o długości l jest jeszcze opłacalne?

Niech p oznacza satysfakcjonujące prawdopodobieństwo dołączenia losowej liczby do wyników wyszukiwania dla tej grupy cyfr. Wtedy wyszukiwanie nie będzie opłacalne, gdy liczba liczb do wyszukania spadnie poniżej wartości p \* (liczba wszystkich liczb o długości l)

1. Koszt orzekania, czy dana liczba została znaleziona powinien być minimalny.

Do tego celu może być wykorzystana długa tablica bitów – ustawienie n bitu oznacza, że n-ta liczba została odnaleziona. Można również zastosować podejście, w którym nie będzie sprawdzane, czy dana liczba została znaleziona i każdy wyszukany wzorzec będzie dołączany do wyników.

Ostatecznie algorytm można przedstawić następująco:

1. Określ k,l jako odpowiednio ilość cyfr w najmniejszej i największej liczbie z przedziału [a;b]
2. Dla każdej grupy liczb o długościach od k do l ustal satysfakcjonujące prawdopodobieństwa przeszukiwania p
3. Przyjmij startową pozycję przeszukiwania w tekście jako j = 0
4. Dopóki przeszukiwanie jest opłacalne dla jakiejkolwiek grup cyfr wykonuj
   1. Pobierz wartość najdłuższej grupy liczb, dla której przeszukiwanie jest opłacalne i oraz j < n – maxl zapisz w zmiennej maxl.
   2. Wykonuj następujące instrukcje dopóki maxl > 0
      1. Sprawdź czy liczba y[j+maxl-1] znajduje się w zbiorze liczb o długości maxl do wyszukania. Jeśli tak, to usuń ją ze zbioru i dołącz do zbioru wyników
      2. Ustaw jako maxl największą liczbę cyfr grupy liczb dla której wyszukiwanie jest opłacalne mniejszą od aktualnej maxl

#### Algorytm Aho – Corasicka

Algorytm Aho – Corasicka jest szybkim algorytmem wyszukiwania pozycji w tekście kilku wzorców naraz. Opiera się na specjalnej strukturze danych będącej połączeniem drzewa Trie i automatu skończonego. Drzewo Trie dla zbioru wzorców *P* charakteryzuje się następującymi własnościami:

* Każda krawędź drzewa jest etykietowana symbolem alfabetu
* Każde dwie krawędzie wychodzące z wierzchołka mają różne etykiety
* Etykieta wierzchołka v jest konkatenacją symboli występujących na etykietach krawędzi ścieżki prowadzącej od korzenia do wierzchołka v i jest określana jako L(v)
* Dla każdego wzorca z P istnieje wierzchołek v, taki że L(v)=P
* Etykieta L(v) każdego z wierzchołków jest równa pewnemu wzorcowi z P

Algorytm konstrukcji drzewa dla zbioru wzorców przedstawia się następująco:

1. Do struktury danych dołącz korzeń, dodawaj kolejno wzorce , według schematu:
   1. Zaczynając od korzenia, podążaj ścieżką wierzchołków wyznaczoną przez kolejne symbole wzorca :
      1. Jeśli ścieżka się skończy, nim skończą się symbole wzorca , dodawaj kolejne krawędzie i wierzchołki zgodnie z kolejnymi pozostałymi symbolami wzorca
      2. Oznacz ostatni wierzchołek na ścieżce symbolem i

Powyższy algorytm za złożoność O(

Tak stworzone drzewo zostaje zamienione w automat poprzez wprowadzenie indeksowania wierzchołków oznaczające stan automatu (korzeń ma indeks 0, pozostałe dowolnie) oraz następujących funkcji:

* Goto: g(q,a) zwraca stan, jeśli będąc w stanie q zostanie natrafiony symbol a (dotyczy bezpośredniego chodzenia po drzewie)
* Failure: f(q) dla q != 0 zwraca stan, do którego nastąpi przejście w przypadku niedopasowania

Powyższe funkcje składają się na funkcję przejścia automatu. Oprócz nich wykorzystywana jest także funkcja out(q), która zwraca indeksy wzorców rozpoznanych przy wejściu w stan q.



Rysunek Przykładowe struktura danych dla słów { he, she, his, hers }

Funkcja Goto zdefiniowana jest bezpośrednio przez krawędzie drzewa. Algorytm określania wartości failure i out funkcji (oparty na przeszukiwaniu wszerz):

1. out(v) = P dla wierzchołka v o etykiecie P
2. Goto(0,a)=0 dla każdego a należącego do alfabetu, ale nie będącego etykietą krawędzi wychodzącej z korzenia
3. Niech Q oznacza pustą kolejkę
4. Dla każdej litery a z alfabetu
   1. Jeśli q(0,a) = q != 0 wtedy
      1. f(q)=0
      2. Dodaj q do kolejki Q
5. Dopóki kolejka Q jest niepusta
   1. Zdejmij r z kolejki
   2. Dla każdego a z alfabetu
      1. Jeśli q(r,a) = u (niepusta)
         1. Dodaj u do kolejki Q
         2. v = f(r)
         3. Dopóki q(v,a) puste i f(v) niepuste wykonuj v= f(v) (szukanie właściwego sufiksu)
         4. f(u) = q(v,a)
         5. out(u) = out(u) U out(f(u)) (wykonywane, ponieważ wzorce rozpoznawane przy f(u) (jeśli takie istnieją) i tylko takie, są właściwymi sufiksami L(u) i powinny być rozpoznane w stanie u

Posiadając tak skonstruowaną strukturę danych łatwo zaimplementować poszukiwanie – zwiększając aktualny indeks tekstu przesuwamy się po drzewie zgodnie z funkcją Goto – jeśli nie jest ona zdefiniowana dla aktualnego znaku tekstu, należy użyć funkcji failure. Gdy napotkany zostanie wierzchołek, dla którego out jest niepuste, oznacza to, ze wzorce określone przez funkcję out zostały znalezione na pozycji którą łatwo obliczyć korzystając z aktualnego indeksu tekstu i długości odnalezionego wzorca.

Złożoność konstrukcji automatu jest rzędu 0(n), gdzie n to sumaryczna długość wszystkich wzorców. Algorytm wyszukiwania jest rzędu 0(n+m+z), gdzie m to długość tekstu, natomiast z to ilość wystąpień wszystkich wzorców w tekście.

#### Podsumowanie algorytmów tworzących funkcję

Tworzenie funkcji dla przedziału [a;b] przedstawia się następująco:

1. Jeśli b – a + 1 < K, wykonaj szukanie algorytmem Turbo Boyera-Moore’a
2. Jeśli K <= b – a + 1 < M, wykonaj szukanie algorytmem Aho – Corasicka. Jeśli liczba wartości funkcji do znalezienia spadnie do K, idź do 1
3. Jeśli M <= b – a +1, wykonaj szukanie algorytmem bezpośredniego wypisywania. Jeśli liczba wartości funkcji do znalezienia spadnie do M, lub wyszukiwanie nie będzie opłacalne tym algorytmem (patrz opis), idź do 2.

Wartości liczb K i M będą dobrane doświadczalnie, tak aby algorytm działał maksymalnie szybko.

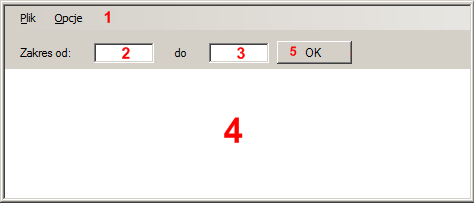
# Interfejs użytkownika

Dla przejrzystości i wygody użytkownika zdecydowaliśmy się podzielić projekt na trzy niezależne od siebie programy (jednak o w miarę możliwości zbliżonym w wyglądzie i głównych cechach interfejsie:

## Interfejs użytkownika części programu generującego liczbę

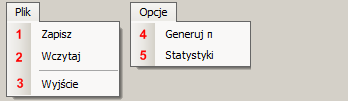
Interfejs użytkownika części programu odpowiedzialnego za generowanie liczby przedstawia się następująco:

* + - 1. *Okno główne*



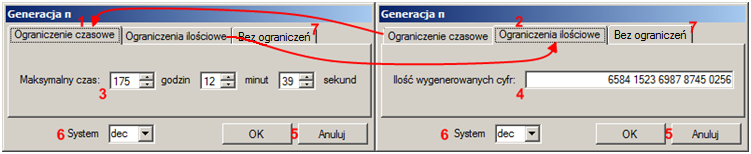
Rysunek (4.1.1.1)

* + - * 1. Menu główne (patrz 4.1.1.2)
        2. Zakres wyświetlanej wygenerowanej liczby w polu 4.1.1.1.4 - wartość *od*
        3. Zakres wyświetlanej wygenerowanej liczby w polu 4.1.1.1.4 - wartość *do*
        4. Pole do wyświetlanie wygenerowanej liczby po odpowiednim formatowaniu (patrz 3.1.1.5)
        5. Przycisk OK. do potwierdzania wprowadzonego zakresu
      1. *Menu główne*



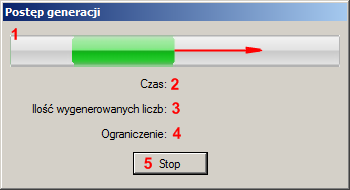
Rysunek (4.1.1.2)

* + - * 1. Zapis uprzednio wygenerowanej liczby . W przypadku, gdy w programie generacja nie została jeszcze przeprowadzona, lub liczba nie została wczytana, opcja ta jest wyszarzona
        2. Wczytanie wygenerowane już liczby . Powoduje otwarcie okna dialogowego do otwarcia pliku.
        3. Wyjście z programu
        4. Powoduje otwarcie okna opcji generacji liczby (patrz 4.1.1.3)
        5. Powoduje wyświetlenie statystyk związanych z generowanie liczby (np. średni czas, największa ilość wygenerowanych cyfr w rozwinięciu, średni czas wygenerowania N cyfr z rozwinięcia)
      1. *Okno opcji generacji liczby π*



Rysunek (4.1.1.3)

* + - * 1. Przełączenie okna na wybór ograniczenia czasowego
        2. Przełączenie okna na wybór ograniczenia ilościowego, co do liczby wygenerowanych cyfr w liczbie
        3. Szczegóły ograniczenia czasowego – możliwość ustawienia godzin, minut i sekund
        4. Szczegóły ograniczenia ilościowego – możliwość ustawienia ilości cyfr do wygenerowania w rozwinięciu
        5. Przyciski OK. i Anuluj służące odpowiednio do rozpoczęcia obliczeń (patrz 4.1.1.4) jak i anulowania dokonanych ustawień.
        6. System, w jakim nastąpi generowanie liczby – odpowiednio dziesiętny i heksadecymalny. Liczba po wygenerowaniu zostanie wyświetlona (i ew. zapisana) w wybranym systemie.
        7. Przełączenie okna na opcję generacji π bez ograniczeń
      1. *Okno generowania liczby – pojawia się jako okno modalne w trakcie generowanie liczy*



Rysunek (4.1.1.4)

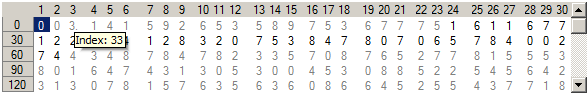
* + - * 1. Pasek postępu generowania – przesuwający się cyklicznie prostokąt pokazujący tym samym, że obliczenia są w toku
        2. Dotychczasowy czas trwania obliczeń
        3. Ilość dotychczas wygenerowanych cyfr w liczbie
        4. Ograniczenie na generowaną liczbę, w zależności od wybranej opcji czasowe lub ilościowe
      1. *Fragment okna głównego odpowiedzialny za wyświetlanie wygenerowanej liczby .*

Poniższy rysunek przedstawia fragment 4.1.1.1.4 okna głównego odpowiedzialny za wyświetlanie wygenerowanej liczby . Dla ustalenia uwagi liczba przedstawiona została w zapisie dziesiętnym, zmiana ustawień opcji generacji spowoduje podobne przedstawienie liczby w zapisie heksadecymalnym.

Strzałki z boku służą do przewijania wyświetlanej liczby. W górnym wierszu prezentowane są indeksy x cyfr z rozwinięcia, zaś w kolumnie z lewej strony indeksy y. Liczba wierszy zależna jest od długości okna, liczba kolumn z liczbami jest stała i wynosi trzydzieści, cyfry w wierszu są grupowane, co trzy (co sześć cyfr dla przejrzystości jest większa przerwa) – patrz rysunek 4.1.1.5.

Dłuższe przetrzymanie kursora myszki nad komórką z cyfrą powoduje pojawianie się obok tooltipa z dokładnym indeksem konkretnej cyfry (wyliczanego jako *indeks* *x + indeks y*).

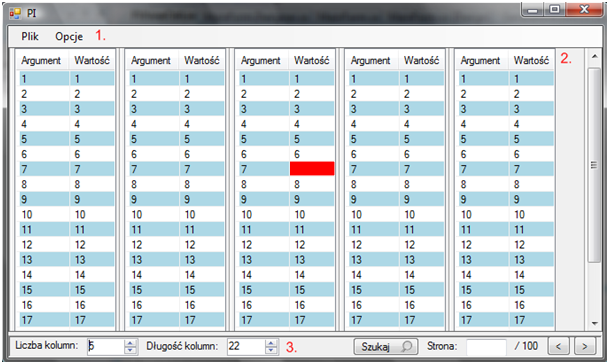
Dodatkowo do przeszukiwania rozwinięcia liczby można użyć pól do wprowadzania zakresu (patrz 4.1.1.1.4). Wybranie takiego sposobu przeglądania spowoduje pokazanie wyświetlanych cyfr z danego zakresu tak, aby pierwsza z nich mieściła się w prezentowanej przestrzeni (w miarę możliwości ostatnia również) – pozostałe widoczne cyfry zostaną wyszarzone. Na poniższym przykładzie pokazywane są cyfry z zakresu od 24 do 62.



Rysunek (4.1.1.5)

## Interfejs użytkownika części programu generującego funkcję

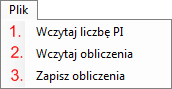
* + 1. **Okno główne aplikacji**



Rysunek (4.2.1)

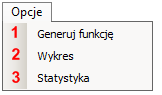
Główne okno aplikacji podzielone jest na trzy części – menu górne (1), widok wyników (2) oraz panel nawigacyjny (3). Widok wyników zbudowany jest z kolumn. Każda kolumna składa się z tabeli reprezentującej wyniki – w postaci argumentu funkcji oraz wartości funkcji. Wartości nieodnalezione funkcji poza pustym polem wartość wyróżnione są czerwonym kolorem komórki.

* + 1. **Menu górne**

****

Rysunek (4.2.2)

Menu plik umożliwiać określenie źródła liczby (1), zapisać obliczenia, (gdy już zostały wygenerowane), oraz wczytać obliczenia (2).



Rysunek (4.2.2)

Menu opcje umożliwia rozpoczęcie generacji funkcji (1), obejrzenie statystyk (3) oraz narysowanie wykresu funkcji na podstawie wygenerowanej funkcji (2) – patrz 4.2.7. W przypadku, jeśli nie została wygenerowana jeszcze żadna funkcja i tym samym niemożliwe jest wykreślenie wykresu opcja (2) jest wyszarzona.

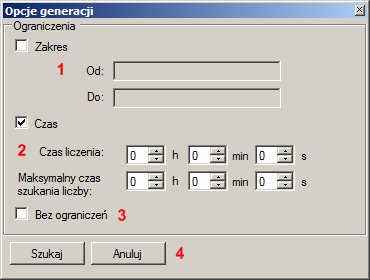
* + 1. **Panel nawigacyjny**

navigationpanel.png

Rysunek (4.2.3)

Panel nawigacyjny umożliwia nawigację po wynikach funkcji – przy pomocy kontrolek można określać liczbę wyświetlanych kolumn oraz ich długość. Przeglądanie podzielone jest na strony, między którymi można się przemieszczać przy użyciu strzałek nawigacyjnych. Przycisk szukaj otwiera okno szukania argumentów.

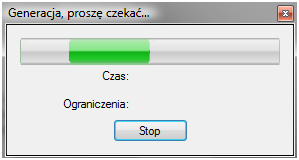
* + 1. **Opcje generacji**



Rysunek (4.2.4)

Opcje generacji służą określeniu ograniczeń dla algorytmu wyszukującego funkcję. Można określić, z jakiego zakresu liczby są poszukiwane(1), całkowity czas poszukiwań oraz maksymalny czas poszukiwania dla pojedynczej liczby (2), możliwe jest również generowanie funkcji bez żadnych nałożonych ograniczeń(3). Wciśnięcie przycisku Szukaj rozpoczyna poszukiwanie funkcji. Jeśli już jakieś źródło funkcji jest określone, to następuję kontynuacja poszukiwań w zależności od źródła. Wciśnięcie przycisku Anuluj zamyka okno.

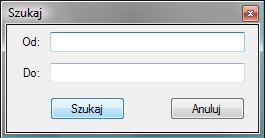
* + 1. **Okno generacji**



Rysunek (4.2.5)

Okno generacji zawiera pasek postępu, licznik czasu obliczeń oraz dane ograniczeń. Wciśnięcie przycisku Stop wstrzymuje obliczenia.

* + 1. **Panel szukania**

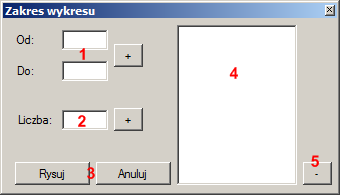
****

Rysunek (4.2.6)

Panel szukania umożliwia znalezienie zakresu liczby (argumenty) liczb w wynikach obliczeń.

Na podstawie przetrzymywanej w pamięci reprezentacji liczby , która została wygenerowana lub odczytana z wcześniej przygotowanego pliku (o tym później) oraz liczb podanych przez użytkownika w polach [3.2.4.1], przy użyciu algorytmu szybkiego odszukiwania wzorca w tekście zostaje wyszukane pierwsze wystąpienie każdego wzorca z przedziału w rozwinięciu (dziesiętnym lub szesnastkowym w zależności od ustawień).

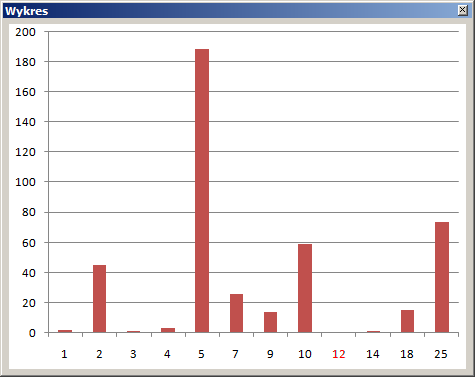
* + 1. **Wykresu funkcji**

****

Rysunek (4.2.7.1)

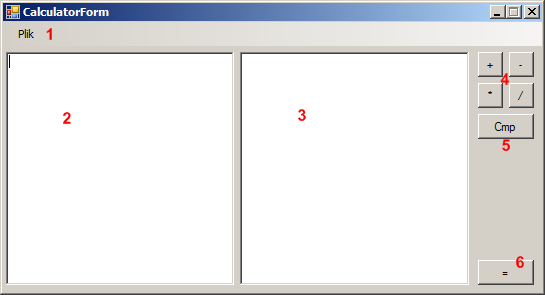
Po wybraniu opcji rysowania wykresu wyliczonej funkcji użytkownik ma możliwość wyboru zakresu w oknie modalnym, z jakiego będzie rysowana dana funkcja. Przycisk (1) służy do dodania zakresu do wykreślania na wykresie, przycisk (2) do dodania pojedynczego argumentu. Ustalone zakresy argumentów / pojedyncze argumenty wyświetlane są na liście (4). Usunięcie pozycji z tej listy możliwe jest przy pomocy przycisku (5). Przyciski z grupy (6) służą do zatwierdzenia dokonanego wyboru i pokazania użytkownikowi wykresu w oknie (4.2.7.2) lub anulowania.

Użytkownikowi prezentowany jest wykres funkcji w postaci „słupkowej” z odpowiednio przeskalowaną osią wartości, tak, aby wszystkie „słupki” zmieściły się na wykresie. W przypadku argumentów funkcji, dla których nie znaleziono odpowiadających im wartości na wykresie zostanie to zaznaczone czerwonym kolorem (patrz 4.2.7.2) w podpisie argumentu. Poniżej prezentowany jest przykładowy wykres:



Rysunek (4.2.7.2)

## Interfejs użytkownika kalkulatora dużych liczb

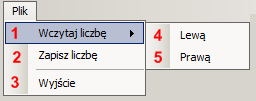


Rysunek (4.3.1)

Kalkulator dużych liczb umożliwia wykonanie podstawowych operacji arytmetycznych na liczbach. Wczytanie liczb następuje poprzez menu (1).

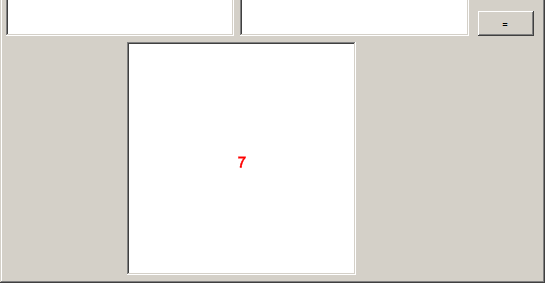
Liczby w polu tekstowym (2, 3 i 7) wyświetlanie są w podobny sposób jak w przypadku programu generującego liczbą π, tj. liczba cyfr w wierszu jest stała i wynosi trzydzieści, cyfry w wierszu są grupowane, co trzy (co sześć cyfr dla przejrzystości jest większa przerwa) – patrz rysunek 4.1.1.5. Każde pole tekstowe posiada paski przewijania umożliwiające przewijanie, gdy wczytana liczba nie mieści się w polu. Przewijanie w obu polach następuje symultanicznie.

Wybór operacji następuje poprzez wybranie odpowiedniego działania arytmetycznego (4) lub operacji porównania (5) – powoduje to wciśnięcie na stałe wybranego przycisku. Aby wykonać wybraną operacje należy następnie nacisnąć przycisk (6). Wynik operacji arytmetycznych jest widoczny w pojawiającej się dodatkowej części okna (7) – patrz 4.3.3, natomiast wyniki operacji porównania widoczne są w polach tekstowych poprzez wyróżnienie na czerwono różnic we wczytanych liczbach. W oknie programu (poniżej pól tekstowych) są wtedy dostępne dodatkowe przyciski do przejścia do następnego „różnicy” oraz powrotu do poprzedniej.



Rysunek (4.3.2)

Opcje menu użytkownika umożliwiają wczytanie liczby (1) do lewego (4) i prawego (5) pola tekstowego. Możliwy jest również zapis liczby będącej wynikiem operacji arytmetycznej (2) tylko, jeśli taka liczba jest dostępna (tzn. została wykonana już jakaś operacja arytmetyczna), w przeciwnym wypadku pole to jest wyszarzane. Opcja (3) powoduje wyjście z programu.



Rysunek (4.3.3)

1. dane za http://gmplib.org/ [↑](#footnote-ref-2)
2. dane za oficjalną dokumentacją biblioteki http://gmplib.org/manual/ [↑](#footnote-ref-3)
3. patrz 2 [↑](#footnote-ref-4)
4. patrz 2 [↑](#footnote-ref-5)