# Sprawozdanie z zadania 5:
# "Protokół Diffiego-Hellmana"

27 listopada 2024

**Autorzy:**

- Maria Małasiewicz, (metoda brutalna, funkcje: potega, euklid
- Maciej Pestka, (metoda Shanksa, porównanie dwóch metod)
- Zuzanna Strauss (funkcje: generator, generatory)

## 1 Wymagania podstawowe

1. znalezienie liczby $a$ dowolna metoda i opisanie podjętych kroków w sprawozdaniu.

## 2 Wymagania dodatkowe

2. użycie dwóch metod,
3. porównanie metod, ich szybkości oraz dokładności,
4. napisanie i przetestowanie działania funkcji potęga(a, b, p) wyliczającą $ab\ (mod\ p)$ szybszą metodą wskazaną na laboratorium,
5. napisanie i przetestowanie działania funkcji generator(g, p) sprawdzająca, czy $g$ jest generatorem w pierścieniu $\mathbb{Z}_p^*$
6. napisanie i przetestowanie działania funkcji generatory(g, p) wypisująca generatory $g$ w pierścieniu $\mathbb{Z}_p^*$ (zapis do pliku),
7. napisanie i przetestowanie działania funkcji euklid(a, p) wyliczający $a^{-1}$ w $\mathbb{Z}_p^*$

## 3 Lista spełnionych wymagań:

1, 2, 3, 4, 5, 6, 7

## 4 Opis realizacji zadania:

**1. Znalezienie liczby $a$ dowolna metoda i opisanie podjętych kroków w sprawozdaniu:**

W celu znalezienia liczby $a$ napisano metodę `metodaBrutalna(g,p,h)`, która iteruje przez kolejne liczby od 0 do $p-1$ ($a \in [0, p)$) i sprawdza czy dana liczba spełnia równanie $g^a \ (mod\ p) = h$. Jeśli równanie jest spełnione, funkcja zwraca znalezioną liczbę i kończy działanie.

**2. użycie dwóch metod:**

Jako drugą metodę wybrano algorytm Shanksa. Napisano metodę `metodaShanksa(g,p,h)`, która na początku liczy pierwiastek z $p$, następnie tworzy listę $1, g, g^2, \ldots, q^{[\sqrt{p}]-1}$. Następnym krokiem metoda liczy $hg^{-1[\sqrt{p}]}$ dla kolejnych $1 = 1, 2, \ldots$ i sprawdza czy jest na liście, którą wygenerował szybciej, do pomocy obliczenie liczby przedziwniej użyto metody `euklid`. Gdy wyliczona liczba jest na liście to wtedy program zna i oraz j. Potem program wylicza $a$ liczy z $a = i[\sqrt{p} + j$. Na samym końcu zwraca liczbę $a$ oraz czas wykonana wszystkich obliczeń i działań. Do pomocy wykorzystana metody użyto pętli for i while.

**3. porównanie metod, ich szybkości oraz dokładności:**

Podczas dopierania różnych parametrów p i h metoda brutalna wykonywała się wolniej niż algorytm Shanksa. Obie metody dawały poprawnie wyniki $a$ i takie same, ze każdym razem. Dla $h = 527$ użyto kilku parametrów $p$, to były wartości $p = \{1019, 1117, 1303, 1607, 1933, 1049, 527\}$ Dla $h = 520$, użyto te same liczby $p$ co dla $h = 527$.

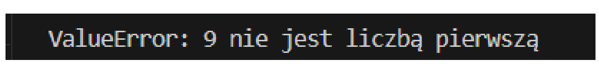
**4. Napisanie i przetestowanie działania funkcji potęga(a, b, p) wyliczającą $a^b \ (mod\ p)$ szybszą metodą wskazaną na laboratorium:**

Została stworzona funkcja `potega(a,b,p)`, która oblicza $a^b \ (mod\ p)$ przy użyciu pętli. Przed rozpoczęciem pętli tworzona jest zmienna `wynik` przyjmująca na starcie wartość $a$. Następnie w pętli `for`, $b-1$ razy, obliczana jest nowa wartość wyniku, wykonując równanie $wynik \cdot a \ (mod\ p)$. Funkcja kończy działanie zwracając końcową wartość wyniku.

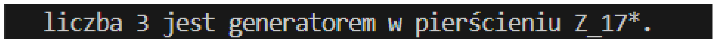
Do testowania funkcji `potega(a,b,p)` powstała funkcja `potega_test()`, która sprawdza dla czterech liczb pierwszych: $491, 4523, 30011, 144773$ o wspólnym generatorze 13 poprawność wyników, jak i czas ich obliczania, który następnie zostaje przedstawiony na wykresie. Potęgi są wyliczane dla kolejnych $b \in [1, p)$, co krok $10^i$, gdzie $i$ to indeks liczby pierwszej w liści (np. $i = 0$ dla 491).

**5. napisanie i przetestowanie działania funkcji generator(g,p) sprawdzająca, czy g jest generatorem w pierścieniu $\mathbb{Z}_p^*$,**

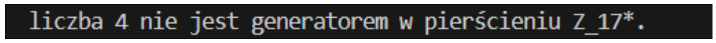
Na samym początku sprawdzane jest, czy liczba $p$ jest liczbą pierwszą poprzez podzielenie $p$ przez każdą liczbę całkowitą do p//2 i sprawdzenie, czy reszta z dzielenia daje 0. Jeżeli tak, wznoszony jest `ValueError` z komunikatem, że $p$ nie jest liczbą pierwszą.




Generowana jest lista `l`, która zawiera wszystkie liczby całkowite od 1 do $p-1$, czyli elementy pierścienia $\mathbb{Z}_p^*$. Tworzona jest również pusta lista `gl`. W pętli w zakresie do p-1 liczona jest wartość $g^k \ mod\ p$ i dodawana jest do listy `gl`. Po wykonaniu całej pętli, lista `gl` jest sortowana i porównywana z listą `l`. Jeżeli obie listy się pokrywają, zwracana jest wartość `True` i wyświetlany jest odpowiedni komunikat. Jeżeli listy różnią się, zwracana jest wartość `False` i odpowiedni komunikat.



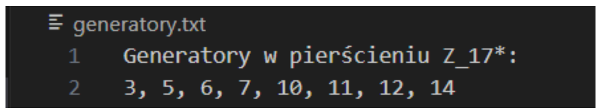

Rysunek 1: Komunikat, gdy liczba jest generatorem




Rysunek 2: Komunikat, gdy liczba nie jest generatorem

**6. napisanie i przetestowanie działania funkcji generatory(p) wypisująca generatory g w pierścieniu $\mathbb{Z}_p^*$ (zapis do pliku)**

Funkcja głównie opiera się na poprzedniej funkcji `generator(g,p)`. Na początku, tworzona jest pusta lista gen. Następnie, w pętli w zakresie do $p - 1$ sprawdzane jest, czy `generator(i,p)` zwraca wartość `True`. Jeżeli tak, to i dodawane jest do listy `gen`. Po wykonaniu całej pętli otwierany jest plik o nazwie 'generatory.txt', pierwsza linijka tego pliku zawsze zawiera opis, w drugiej linijce przepisywane są generatory z listy `gen`.




**7. napisanie i przetestowanie działania funkcji euklid(g, p) wyliczający $a^{-1}$ w $\mathbb{Z}_p^*$:**

Do wykonania algorytmu Euklidesa utworzona została funkcja euklid(g, p). Funkcja tworzy tablicę dwuwymiarową na wzór tabelki pokazanej na wykładzie, w której zapisuje wartości kolejnych iteracji. Zaczynamy od tablicy, która w pierwszym wierszu ma kolejno wartości $p, 1, 0$, a w drugim - $g, 0, 1, -[\frac{p}{g}]$. Następne wiersza są wyliczane następująco:

Niech $w1$ oznacza drugi od końca wiersz o wyrazach $w1_1, w1_2, w1_3, w1_4$, a $w2$ - wiersz przedostatni.

- W pierwszej kolumnie znajduje się reszta z dzielenia $w1_1$ i $w2_1$,
- w drugiej kolumnie -wynik równania: $w2_2 \cdot w2_4 + w1_2$,
- w kolumnie trzeciej -wynik równania: $w2_3 \cdot w2_4 + w1_3$
- w czwartej - cześć całkowita z dzielenia $w2_1$ przez wartość z pierwszej kolumny przemnożona przez $-1$ .

Nim jednak zostanie wyliczona czwarta kolumna, sprawdzane jest czy wartość w pierwszej wynosi 1. Jeśli tak to zwracany jest element trzeciej kolumny oraz funkcja kończy działanie. W przeciwnym wypadku wyliczany jest nowy wiersz.

## 5 Wnioski:

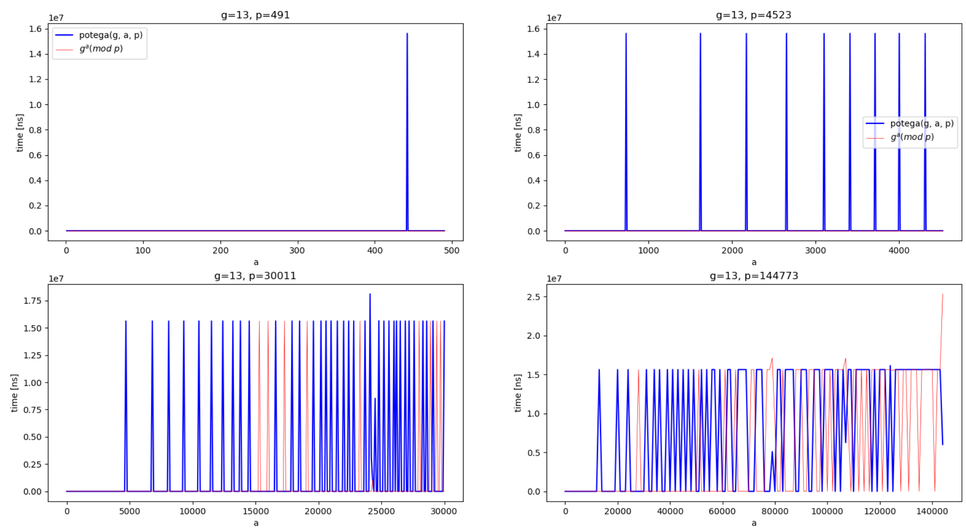
Podczas testowania funkcji `potega`, czas wykonania funkcji nie wychodził na krótszy niż standardowe obliczania, co można zobaczyć na rysunku 3. Czas także różnił się przy różnych uruchomieniach, zatem najprawdopodobniej wpływ na czas wykonania mają działające w tle procesy.

## 6 Prawa autorskie do kodu:

Wszystkie funkcje zostały napisane przez autorów na podstawie materiału z zajęć.

## 7 Źródła

Slajdy z laboratorium.




Rysunek 3: Wykresy z funkcji `potega_test()`