Zajęcia A-5: “Algorytm Euklidesa”

Cel zajęć i efekty uczenia

Główne cele zajęć / materiał do opanowania:

- Opanowanie i samodzielna implementacja algorytmu Euklidesa

Dodatkowe cele:

- Utrwalenie zasady działania pętli typu *while*
- Intuicja złożoności logarytmicznej jako efektywnej, a liniowej jako nieefektywnej, jeśli wejściem algorytmu są duże liczby
- Umiejętność krytycznego spojrzenia na algorytm, szukanie przypadków trudnych dla algorytmu

Zadanie do rozwiązania na sprawdzarce

Algorytm Euklidesa
Dane są liczby a, b w zakresie typu int, obliczyć i wypisać NWD(a,b).

Plan zajęć

Szacunkowy czas trwania: 2 godziny lekcyjne.

1. Pojęcie wspólnych dzielników dwóch liczb, pojęcie największego wspólnego dzielnika
    - *Uczniowie powinni mieć świadomość, że wspólnych dzielników może być wiele, a największy dzieli wszystkie pozostałe.*
2. [opcjonalnie] Algorytm z lekcji matematyki (rozłóż na czynniki pierwsze i wybierz wspólne) - czemu go nie stosujemy?
    - *Na tym etapie nie jest jeszcze jasne, jak rozkładać na czynniki pierwsze - ale widać, że nie ma oczywistej metody. Warto wspomnieć, że ostatecznie rozkład na czynniki jest trudnym problemem, znacznie trudniejszym niż NWD, i na nim opiera się algorytmy kryptograficzne.*
3. Własność odejmowania: NWD(a,b) = NWD(b,a-b)
    - *Przejście (a,b) -> (b,a-b) jest zachowuje wszystkie wspólne dzielniki, a więc i największy. Można dla ilustracji rozpisać przykład na małych liczbach.*
4. Algorytm iteracyjny z pętlą *while*.
    - *Uwaga! Wciąż jest mowa o algorytmie z odejmowaniem.*
5. Przetestowanie algorytmu na kilku przykładach
6. Nieefektywność dla dużych liczb (kontrprzykłady)
    - *Tutaj warto zagrać z uczniami w grę: teraz to oni “są sprawdzarką” i muszą dobrać taki test, żeby rozważany program go nie przeszedł.*

    - *Nie operujemy (na razie) na funkcjach, tylko pytamy o konkretne liczby.*
    - *Jeśli znajdą, na przykład (1000000000,1), możemy pytać dalej: jak poradzą sobie z programem, który na wstępie ma rozpatrzone skrajne przypadki b=1 lub b|a.*
    - *Teraz zauważamy, że algorytm może wykonać liczbę operacji bliską a (lub a/2), czyli bardzo dużą. Żeby działał efektywnie, trzeba go przyspieszyć.*
7. Przyspieszenie algorytmu: operacja dzielenia z resztą
    - *Zauważamy, że podzielenie z resztą a%b to "skrót" dla odejmowań, które w algorytmie i tak by nastąpiły - jeśli działała poprzednia wersja, to i ta zadziała.*
8. [opcjonalnie] Szukanie pesymistycznych przypadków (liczby Fibonacciego)
9. [opcjonalnie] Argument na dobrą złożoność algorytmu
    - *W każdym wypadku jedna z liczb zmniejsza się co najmniej dwukrotnie - a zatem nie może być więcej kroków niż log a + log b.*