Lekcja 11. (p)

Temat: **Największy i najmniejszy, czyli jak znaleźć NWD i NWW**.

Cele lekcji:

Poznanie algorytmów Euklidesa i różnicy pomiędzy optymalną a nieoptymalną wersją. Ułożenie programów obliczających NWD i NWW .

Uczeń:

– wie, że istnieją algorytmy rozwiązujące podstawowe i bardziej skomplikowane problemy np. matematyczne, opracowane przez uczonych na przestrzeni dziejów cywilizacji
– zna działanie algorytmu Euklidesa w obu postaciach – optymalnej i nieoptymalnej oraz wie, jakie są różnice pomiędzy nimi
– umie zilustrować działanie algorytmu Euklidesa przykładami i porównać obie metody
– umie zapisać algorytm Euklidesa za pomocą schematu blokowego i dyskutować jego działanie
– wykorzystanie NWD do obliczenia NWW
– umie ułożyć programy wg obu algorytmów
– uzasadnia sposób przedstawiania wyniku działania programu
– dyskutuje strukturę ułożonego programu i decyzje użycia danych bibliotek i rozkazów

Przebieg lekcji:

1. **Stworzenie i analiza algorytmu wyszukującego NWD i NWW.**
2. **Działanie algorytmu Euklidesa.**

Niezoptymalizowany algorytm Euklidesa

Sposób rozwiązania jest następujący.

Wybieramy większą z dwóch liczb i zamieniamy ją na różnicę większej i mniejszej. Czynność tą powtarzamy do momentu uzyskania dwóch takich samych wartości.

Prześledźmy przykład dla liczb 12 i 18. Wiadomo, że NWD(12,18)=6.

12	18
12	18 - 12 = 6
12 - 6 = 6	6

Dla liczb 28 i 24 NWD(28,24)=4:

28	24
28 - 24 = 4	24
4	24 - 4 = 20
4	20 - 4 = 16
4	16 - 4 = 12
4	12 - 4 = 8
4	8 - 4 = 4

Warto zauważyć, że ten algorytm jest bardzo niewydajny. Gdy dobierzemy odpowiednio liczby, ilość operacji znacznie się zwiększy. Przeanalizujmy takie liczby jak 1 i 10000:

$$\begin{array}{c|l} 1 & 10000 \\ 1 & 10000 - 1 = 9999 \\ 1 & 9999 - 1 = 9998 \\ \vdots & \vdots \\ 1 & 4 - 1 = 3 \\ 1 & 3 - 1 = 2 \\ 1 & 2 - 1 = 1 \end{array}$$

W tym przypadku najmniejszy wspólny dzielnik jest równy jeden. Żeby to stwierdzić, należy wykonać 9999

kroków pętli. Dla większych liczb algorytm może nie sprostać zadaniu.

Rozwiązanie iteracyjne:

```
#include<iostream>
#include<cstdlib>
using namespace std;

int NWD(int a, int b)
{
    while(a!=b)
       if(a>b)
           a-=b; //lub a = a - b;
       else
           b-=a; //lub b = b-a
    return a; // lub b - obie zmienne przechowują wynik NWD(a,b)
}

int main()
{
    int a, b;

    cout<<"Podaj dwie liczby: ";
    cin>>a>>b;

    cout<<"NWD("<<a<<","<<b<<") = "<<NWD(a,b)<<endl;

    system("pause");
    return 0;
}
```

Rozwiązanie rekurencyjnie:

```
#include<iostream>
#include<cstdlib>
using namespace std;

int NWD(int a, int b)
{
   if(a!=b)
     return NWD(a>b?a-b:a,b>a?b-a:b);
   return a;
}

int main()
{
    int a, b;
```

```
    cout<<"Podaj dwie liczby: ";
    cin>>a>>b;

    cout<<"NWD("<<a<<","<<b<<") = "<<NWD(a,b)<<endl;

    system("pause");
    return 0;
}
```

Zakładamy, że liczby $a>0,b>0$

.

Uwaga! **NWD(a,0)=a**, gdzie a>0, **NWD(0, b)=b**, gdzie b>0. Ten przypadek nie jest rozpatrzony w algorytmie.

Zoptymalizowany algorytm Euklidesa

W przypadku optymalnego rozwiązania NWD postępujemy następująco:

załóżmy, że wyznaczamy NWD dwóch liczb naturalnych a

i b

. W każdym przejściu pętli wykonujemy dwie operacje

$$a=b$$

$$b=a \bmod b$$

Czynności te powtarzamy do momentu, gdy zmienna b

osiągnie wartość zero. Zmienna a

będzie przechowywać wtedy największy wspólny dzielnik liczb podanych na wejściu. Przeanalizujmy przykłady:

Policzmy $NWD(12,18)=6$

:

$$\begin{array}{r|l} 12 & 18 \\ 18 & 12 \bmod 18 = 12 \\ 12 & 18 \bmod 12 = 6 \\ 6 & 12 \bmod 6 = 0 \end{array}$$

Dla liczb 28 i 24:

$$\begin{array}{r|l} 28 & 24 \\ 24 & 28 \bmod 24 = 4 \\ 4 & 24 \bmod 4 = 0 \end{array}$$

Natomiast dla liczb 10000 i 1:

$$\begin{array}{r|l} 10000 & 1 \\ 1 & 10000 \bmod 1 = 0 \end{array}$$

Rozwiązanie iteracyjne w C++:

```
#include<iostream>
#include<cstdlib>
using namespace std;

int NWD(int a, int b)
{
    int pom;

        while(b!=0)
    {
                pom = b;
                b = a%b;
                a = pom;
        }

    return a;
}

int main()
{
    int a, b;

    cout<<"Podaj dwie liczby: ";
    cin>>a>>b;

    cout<<"NWD("<<a<<","<<b<<") = "<<NWD(a,b)<<endl;

    system("pause");
    return 0;
}
```

Rozwiązanie rekurencyjne:

```
#include<iostream>
#include<cstdlib>
using namespace std;

int NWD(int a, int b)
{
    if(b!=0)
        return NWD(b,a%b);

    return a;
}

int main()
{
    int a, b;

    cout<<"Podaj dwie liczby: ";
```

```
    cin>>a>>b;

    cout<<"NWD("<<a<<","<<b<<") = "<<NWD(a,b)<<endl;

    system("pause");
    return 0;
}
```

Warto zauważyć, że algorytm poprawnie wyznacza NWD

w przypadku, gdy jedna z liczb jest równa zero.

Pamiętaj, że NWD dwóch zer nie istnieje!!!

Ciekawe rozwiązanie

```
int euklides(int a, int b)
{
        while(b)
                swap(a %= b, b);
        return a;
}
```

3. **Utworzenie programu w C++ realizujące w/w zadanie:**
   a. **deklaracja zmiennej (typy zmiennych);**
   b. **pętla for i jej indeksowanie**
   c. **operatory warunkowe: if, else, if else, n%i**
   d. **testowanie program**

Powtórz zakres ćwiczenia z lekcji, przetestuj program.

Sprawdzenie NWD i NWW dla liczb – program ułamki.

```
#include<iostream>
#include<cstdlib>
using namespace std;
int A,a1,a2,B,b1,b2;
int c1,c2,d1,d2,m,mianownik;
int X,x1,x2,W,w1,w2;
int a,b;
int NWD(int a, int b)
{
  while(a!=b)
    if(a>b)
      a-=b; //lub a = a - b;
    else
      b-=a; //lub b = b-a
  return a; // lub b - obie zmienne przechowują wynik NWD(a,b)
}

int NWW(int a, int b)
{
```

```
        int pom;
pom=(a*b)/NWD(a,b);
   return pom;
}

int main()
{
    cout <<"podaj część całkowitą liczby 1 (A)"; cin >> A;
    cout <<"podaj licznik liczby 1 (a1)"; cin >> a1;
    cout <<"podaj mianownik liczby 1 (b1)"; cin>> a2;

    cout <<"podaj część całkowitą liczby 2 (B)"; cin >> B;
    cout <<"podaj licznik liczby 2 (b1)"; cin >> b1;
    cout <<"podaj mianownik liczby 2 (b2)"; cin >> b2;
   cout <<endl<<"Wprowadziłeś ułamki"<<endl<<endl;

   cout <<"   "<<a1<<"          "<<b1<<endl;
   cout <<A<<" ------"<<" + "<<B<<" -------"<<endl;
   cout <<"   "<<a2<<"          "<<b2<<endl;

   cout <<"postac ogolna zapisu ułamków"<< endl;
   c1=A*a2+a1;
  c2=a2;
  d1=B*b2+b1;
  d2=b2;
   cout <<c1<<"        "<<d1<<endl;
   cout <<"------"<<" +  -------"<<endl;
   cout <<c2<<"        "<<d2<<endl;
m=NWD(c2,d2);
  cout << "wspólny dzielnik wynosi = "<<m<< endl;
mianownik=NWW(c2,d2);
  cout << "wspólny mianownik = "<<mianownik<< endl;
cout << "postac ułamka po wyznaczeniu wspólnegp mianownika"<<endl;
x1= mianownik/c2*c1+mianownik/d2*d1;
x2=mianownik;
cout <<x1<<endl;
cout <<"------"<<endl;
cout <<x2<<endl;
if (x1>x2)
{ W=x1/x2; w1=x1-W*x2; w2=x2;
cout <<"   "<<w1<<endl;
cout <<W<<" ------"<<endl;
cout <<"   "<<w2<<endl;
m=NWD(w1,w2);
cout <<"wspólny dzielnik dla licznika i mianownika = "<< m<<endl;

cout << "ułamek doprowadzony do najprostrzej postaci"<<endl;
w1=w1/m ;w2=w2/m;
cout <<"   "<<w1<<endl;
cout <<W<<" ------"<<endl;
cout <<"   "<<w2<<endl;
}
return 0;
}
```