Wykład 04 w dniu 22.X.08r

Jeszcze raz **Zadanie 4** z **Wykładu 03**, gdzie obliczane były wartości wielomianu

$$w_n(x) = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x^1 + a_0$$

dla szczególnego przypadku $n = 3$. Rozpatrzmy **Algorytm 1** oraz **Algorytm 2** realizujące nasze zadanie.

Dane: $n, a_0, a_1, \ldots, a_n, x$
Wynik: $w = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x^1 + a_n$
1 $w \leftarrow a_0$;
2 **for** $i \leftarrow 1$ **to** n **do**
3 | $w \leftarrow a_i + x \cdot w$;
4 **end**

Algorytm 1: Schemat *Hornera* obliczania wartości wielomianu.

Źródło w pliku `w4a.c`:

```
1  #include <stdio.h>
2
3  const int N=100;
4
5  int main()
6  {
7     double a[N],x,w;
8     FILE *fIN;
9     int i,n;
10
11 /* wczytanie z pliku w4Dane.dat wspolczynnikow wielomianu */
12    fIN = fopen("w4Dane.dat","r");
13    if( fIN==NULL )
14    {
15       printf("ERROR: nie udalo sie otwarcie pliku !?\n");
16       return -1;
17    }
18    fscanf(fIN,"%i",&n);
19    for( i=0 ; i<=n ; i=i+1)
20    {
21       fscanf(fIN,"%lf",&a[i]);
22    }
23    fclose(fIN);
24 /* Sprawdzenie co zostalo wczytane: */
25    printf("n: %i\n",n);
26    for( i=0 ; i<=n ; i=i+1)
27    {
28       printf("%10.2lf\n",a[i]);
29    }
30 /* wczytanie z klawiatury x */
31    printf("\nx: ");
32    scanf("%lf",&x);
33 /* obliczenie wartosci wielomianu */
34    w = a[0];
35    for( i=1 ; i<=n ; i=i+1)
36    {
37       w = a[i]+x*w;
38    }
39 /* wyswietlenie obliczonej wartosci w */
```

```
40     printf("w: %10.2lf\n",w);
41     return 0;
42 }
```

Dane: $n, a_0, a_1, \ldots, a_n, x$
Wynik: $w = a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x^1 + a_n$

```
1 w ← a_n;
2 for i ← 1 to n do
3 |   p ← 1;
4 |   for j ← 1 to i do
  |   |   //  tu obliczamy x^i
5 |   |   p ← x · p;
6 |   end
7 |   w ← w + a_{n-i} · p;
8 end
```

Algorytm 2: Jak *nie naleĹży* obliczać wartości wielomianu !

Źródło w pliku `w4b.c`:

```
1  #include <stdio.h>
2
3  const int N=100;
4
5  int main()
6  {
7     double a[N],x,w,p;
8     FILE *fIN;
9     int i,j,n;
10
11 /* wczytanie z pliku w4Dane.dat wspolczynnikow wielomianu */
12    fIN = fopen("w4Dane.dat","r");
13    if( fIN==NULL )
14    {
15       printf("ERROR: nie udalo sie otwarcie pliku !?\n");
16       return -1;
17    }
18    fscanf(fIN,"%i",&n);
19    for( i=0 ; i<=n ; i=i+1)
20    {
21       fscanf(fIN,"%lf",&a[i]);
22    }
23    fclose(fIN);
24 /* Sprawdzenie co zostalo wczytane: */
25    printf("n: %i\n",n);
26    for( i=0 ; i<=n ; i=i+1)
27    {
28       printf("%10.2lf\n",a[i]);
29    }
30 /* wczytanie z klawiatury x */
31    printf("\nx: ");
32    scanf("%lf",&x);
33 /* obliczenie wartosci wielomianu */
34    w = a[n];
35    for( i=1 ; i<=n ; i=i+1)
36    {
37       p = 1.0;
```

```
38 /* obliczenie x^i */
39       for( j=1 ; j<=i ; j=j+1)
40       {
41           p = x*p;
42       }
43       w = w+p*a[n-i];
44    }
45 /* wyswietlenie obliczonej wartosci w */
46    printf("w: %10.2lf\n",w);
47    return 0;
48 }
```

Testowanie: Działanie obu programów zostało sprawdzone dla danych zapisanych w pliku tekstowym `w4Dane.dat` (program i plik z danymi mają się znajdować w tym samym katalogu):

```
3
0.5
-3.0
5.5
-3.0
```

Wynik: ten sam dla obu programów

```
n: 3
      0.50
     -3.00
      5.50
     -3.00

x: 1.1
w:        0.09
```

Mamy więc dwa algorytmy **Algorytm 1** oraz **Algorytm 2** realizujące to samo zadanie, który z nich zasługuje na tytuł *„Mister Algorytmów Wykładu 04”*? Policzmy ile trzeba wykonać mnożeń (jest to tzw. **operacja dominująca**) liczb rzeczywistych typy `double`. Obliczamy, że należy odpowiednio wykonać

$$f_1(n) = n \qquad \text{dla } \textbf{Algorytmu 1}$$

$$f_2(n) = n + \tfrac{n(n+1)}{2} \qquad \text{dla } \textbf{Algorytmu 2}$$

mnożeń liczb typu `double` aby otrzymać ten sam wynik! Tak więc z uwagi na mniejszą liczbę operacji mnożenia *„Mister Algorytmów Wykładu 05”* został **Algorytm 1**.

. . . . . .

Złożoność obliczeniowa (computational complexity) – nieustanny konkurs *„Mister Algorytmów”*.

Pojęcie *złożoności obliczeniowej* wprowadził Hartmanis, J. & Stearns, R.E. w artykule [2] z 1965r. Był to początek współczenej teoretycznej informatyki (za Fortnow, L. & Homer, S. [1]).

Na potrzeby wykładu będziemy oceniali złożoność obliczeniową algorytmu według liczby wykonań wskazanej operacji dominującej. W rozważanych algorytmach obliczających wartość wielomianu jest to ilość mnożeń liczb typu `double`.

Definicja 1. (Notacja „Wielkie O”)
Niech $f, g : \mathbb{N} \to \mathbb{N}$. *Mówimy, że f jest co najwyżej rzędu g, gdy*

$$\exists_{c>0}\exists_{n_0\in\mathbb{N}}\forall_{n>n_0} cf(n) \leq g(n).$$

Co zapisujemy:

$$f(n) = O(g(n)).$$

Dla **Algorytm 1** mamy $f_1(n) = O(n)$ a dla **Algorytmu 2** $f_2(n) = O(n^2)$.

. . . . . .

Zadanie 5: Dla $n, m \in \mathbb{N}$ wyznaczyć NWD(n, m).

Rozwiązanie: (Algorytm Euklidesa) Niech $n > m$. Określamy ciągi liczb naturalnych:

$$\begin{aligned} n_0 &= n \\ m_0 &= m \end{aligned}$$

oraz dla $i = 0, 1, 2, \ldots$

$$\begin{aligned} n_i &= m_i\, k_i + r_i \\ n_{i+1} &= m_i \\ m_{i+1} &= r_i \end{aligned}$$

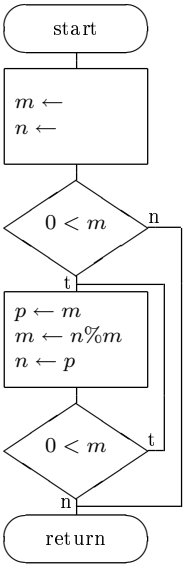
Z własności $r_i > r_{i+1}$ mamy, że istnieje i_o dla którego $r_{i_o+1} = 0$ oraz $r_{i_o} \neq 0$. Dowodzi się, że NWD$(n, m) = r_{i_o}$.

Pseudokod:

Dane: m, n
Wynik: $p \leftarrow$ NWD(m, n)
1 **while** $0 < m$ **do**
2 $p \leftarrow m$;
  // obliczamy resztę z dzielenia n przez m i podstawiamy do m
3 $m \leftarrow n \bmod m$;
4 $n \leftarrow p$;
5 **end**

Algorytm 3: Wyznaczenie NWD(m, n)

Schemat blokowy:



$p \leftarrow$ NWD(m, n)

Źródło w pliku `nwd.c`:

```c
/* NWD(m,n) */
#include <stdio.h>

int main(int argc,char *argv[])
{
    int p,n,m;

    if( argc!=3 )
    {
        puts("ERROR: zla ilosc danych !?");
        for(n=0 ; n<argc ; n=n+1)
        {
            puts(argv[n]);
        }
        return -1;
    }
    sscanf(argv[1],"%i",&m);
    sscanf(argv[2],"%i",&n);
    if( m<1 || n<1)
    {
        puts("ERROR: zle liczby !?");
        puts(argv[1]);
        puts(argv[2]);
        return -2;
    }
    printf("NWD(%i,%i)=",m,n);
    while(0<m)
    {
        p = m;
        m = n%m;
        n = p;
    }
    printf("%i\n",p);
    return 0;
}
```

Sprawdzenie działania:

```
05:28:56 chaos:~/OUT$ gcc -Wall nwd.c -o nwd
05:29:14 chaos:~/OUT$ ./nwd 48 90
NWD(48,90)=6
```

.....

Zadanie 6: (techniczne) Z danego pliku dyskowego wczytać liczby naturalne, po czym

- usunąc liczby parzyste;
- wyznaczyć z pozostałych min, max;
- wyświetlić na ekranie w odwrotnym porządku od wczytanego.

Rozwiązanie nie będzie optymalne.

```c
#include <stdlib.h>
#include <stdio.h>

const int N=1000;

int main(int argc,char *argv[])
{
    int p,q,i,n,L[N],Min,Max;
```

```
9     FILE *fIN;
10
11    if(argc!=2)
12    {
13        puts("ERROR: Zla ilosc argumetow !?");
14        return -1;
15    }
16    fIN = fopen(argv[1],"r");
17    if(fIN==NULL)
18    {
19        printf("ERROR: \"%s\" upss...!?",argv[1]);
20        return -2;
21    }
22 /* wczytujemy dane */
23    fscanf(fIN,"%i",&n);
24    for(i=0 ; i<n ; i=i+1)
25    {
26        fscanf(fIN,"%i",&L[i]);
27    }
28    fclose(fIN);
29 /* wiadomo - sprawdzenie */
30    puts("Wczytano:");
31    for(i=0 ; i<n ; i=i+1)
32    {
33        printf("%6i\n",L[i]);
34    }
35 /* zerujemy liczby parzyste i na wszelki wypadek ujemne */
36    for(i=0 ; i<n ; i=i+1)
37    {
38        if(L[i]%2==0 || L[i]<0)
39        {
40            L[i] = 0;
41        }
42    }
43 /* kompresja tablicy */
44    p = 0;
45    for(i=0 ; i<n ; i=i+1)
46    {
47        if(L[i]==0)
48            continue;
49        if(p<n)
50        {
51            L[p] = L[i];
52        }
53        p = p+1;
54    }
55    if( p==0 )
56    {
57        puts("ERROR: Upss...!?");
58        return -2;
59    }
60    n = p;
61 /* zmieniamy kolejnosc */
62    p = n/2;
63    for(i=0 ; i<p ; i=i+1)
64    {
65        q        = L[i];
66        L[i]     = L[n-i-1];
```

```
67          L[n-i-1] = q;
68      }
69  /* wynik operacji */
70      puts("\nWynik dzialania:");
71      for(i=0 ; i<n ; i=i+1)
72      {
73          printf("%6i\n",L[i]);
74      }
75  /* wyznaczenie min, max */
76      Max = Min = L[0];
77      for(i=1 ; i<n ; i=i+1)
78      {
79          if(L[i]<Min)
80          {
81              Min = L[i];
82          }
83          if(Max<L[i])
84          {
85              Max = L[i];
86          }
87      }
88  /* wiadomo */
89      printf("\nMin: %6i\nMax: %6i\n\n",Min,Max);
90      return 0;
91  }
```

.....

Literatura

[1] Fortnow, L. & Homer, S., *A short history of computational complexity.* In D. van Dalen, J. Dawson, and A. Kanamori, editors, The History of Mathematical Logic. North-Holland, Amsterdam, 2003. (K.T. praca dostępna na stronie autora `http://people.cs.uchicago.edu/~fortnow/papers/`)

[2] Hartmanis, J. & Stearns, R.E. *On the computational complexity of algorithms.* Trans. Amer. Math. Soc. 117, 1965, S. 285-306

Wrocław, dnia 29/10/2008