LABORATORIUM NR 1

# EKSPERYMENTALNA ANALIZA ZŁOŻONOŚCI CZASOWEJ

## ZADANIE AiSD.1.1

Poniżej podane są przykładowe pseudokody trzech procedur oraz implementacja pierwszej każdej z nich, wraz z przykładowym sposobem pomiaru czasu działania (patrz także plik `AiSD_01_1.c`). Przeanalizuj pseudokody tych procedur i „pobieżnie" oszacuj, jaka jest złożoność czasowa każdej z tych procedur. Następnie w oparciu o przykładowy pomiar czasu działania przetestuj doświadczalnie zaproponowane oszacowania.

- Procedura1(n:integer)
```
begin
x:=0.0;
for i:=n downto 1 do
    if nieparzyste(i) then begin
        for j:=1 to i do ;
        for k:=i+1 to n do x:=x+1;
    end
return x;
end
```

- Procedura2(n:integer) [Pytanie: Co oblicza Procedura2?]
```
begin
utwórz tablicę A[1...n] z losowymi liczbami całkowitymi z przedziału [-10,10];
x:=0.0;
for d:=1 to n do
    for g:=d to n do begin
        suma:=0.0;
        for i:=d to g do
            suma:=suma+A[i];
        x:=max(x,suma);
    end
return x;
end
```

- Procedura3(n:integer)
```
begin
for i:=1 to sqrt(n) do begin
    j:=1;
    while j<sqrt(n) do j:=j+j;
    end
end
```

**Przykładowy pomiar czasu (w pliku AiSD_01_1.c).** /© P. Pączkowski/

- Wywołujemy badaną procedurę dla różnych wartości $n$, np. $n = 5, 10, 15, \ldots$, mierząc w programie rzeczywisty czas $T(n)$ działania odpowiedniej procedury /zmienna `Tn`/.
- Następnie, jeśli $F(n)$ jest oszacowaniem na czas procedury /zmienna `Fn`/, np. $F(n) = 5 \cdot n$, wyliczamy ilorazy $F(n)/T(n)$.
- Jeżeli czas rzeczywisty zgadza się z teoretycznym oszacowaniem, wówczas otrzymane ilorazy powinny wyjść mniej więcej stałe.

```
// kompilować z opcjami -lrt -lm, tj. np. gcc AiSD_01_1.c -lrt -lm
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#define MAX 600001
#define MLD 1000000000.0
//////////////////////////////////////////
//   PROCEDURA 1                        //
//////////////////////////////////////////
double procedura1(long int n){
 float x=0;
 long int i,j,k;

 for(i=n-1;i>=1;i--){
  if((i % 2) == 1){
   for(j=1;j<i+1;j++) ;
   for(k=i+1;k<n+1;k++) x=x+1;
  }
 }
 return x;
}
//////////////////////////////////////////
//   PROCEDURA 2                        //
//////////////////////////////////////////
int procedura2(long int n){
int *A;
long int d,g,i,x,suma;

 A=(int *) malloc(n*sizeof(int));
 for(i=0;i<n;i++)
   A[i]=(rand()% 21)-10;

 x=0;
 for(d=1;d<n+1;d++)
 for(g=d;g<n+1;g++){
   suma=0;
   for(i=d;i<g+1;i++)
    suma=suma+A[i];
   if (suma>x) x=suma;
 }

free(A);
return x;
}
//////////////////////////////////////////
//   PROCEDURA 3                        //
//////////////////////////////////////////
void procedura3(long int n){
 long int i,j,k;

 for(i=1;i<(int)sqrt(n)+1;i++){
  j=1;
  while (j<sqrt(n)) j+=j;
 }
}
```

```
//////////////////////////////////////////
//   POMIAR CZASU DLA WIELU PRÓB          //
//////////////////////////////////////////
int main(){
  struct timespec tp0, tp1;
  double Tn,Fn,x;
  long int n; // liczba testów

for(n=100;n<30000;n=n+100){

clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&tp0);

// przykładowe obliczenia
   x=procedura1(n);
// x=procedura2(n);
// procedura3(n);

clock_gettime(CLOCK_PROCESS_CPUTIME_ID,&tp1);

// zgadywana funkcja czasu
       Fn=5*n;
//     Fn=20000*n;
//     Fn=n*n*n;
//     Fn=n*log(n);
//     Fn=n*n*sqrt(n);
//     Fn=n*n;
//     Fn=n*n/1000;

  Tn=(tp1.tv_sec+tp1.tv_nsec/MLD)-(tp0.tv_sec+tp0.tv_nsec/MLD);
  printf("n: %5ld \tczas: %3.10lf \twspolczynnik: %3.5lf\n",n,Tn, Fn/Tn);
}
return 1;
}
```

**ZADANIE AiSD.1.2** (5+1* pkt.)
W pliku `AiSD_01_2.c` znajdują się (zaimplementowane) funkcje, które dla ustalonej zerojedynkowej macierzy kwadratowej $M_{n\times n}$ wyznaczają jej podmacierz o największej liczbie jedynek (interpretując to graficznie – zacieniowany prostokąt o największym polu). Przeanalizuj kody tych funkcji (jaka jest idea tych algorytmów? (1+1* pkt.)) oraz oszacuj również ich złożoność czasową (1 pkt). Następnie w oparciu o pomiar czasu działania z zadania AiSD.1.1 przetestuj doświadczalnie zaproponowane oszacowania (1.5 pkt). Ponadto przetestuj (1.5 pkt), czy i jak zmieniają się te oszacowania dla każdej z funkcji, gdy macierz wejściowa składa się z (a) samych zer, (b) samych jedynek.

- Dla każdego z algorytmów należy podać (na kartce i skan/zdjęcie wysłać na adres e-mail `zylinski@inf.ug.edu.pl`): zaproponowane oszacowanie z analizy kodu, odgadnięte oszacowanie z eksperymentu, odgadnięte oszacowanie z eksperymentu, gdy macierz wypełniona jest samymi 0-ami, odgadnięte oszacowanie z eksperymentu, gdy macierz wypełniona jest samymi 1-mi, oraz ideę algorytmu.

- Aby oszacować złożoność, należy przyjrzeć się implementacji. Przykładowo, jeśli mielibyśmy cztery zagnieżdżone pętle do cyklicznego przeszukiwania macierzy, to można oszacować złożoność np. na „około $n \cdot n \cdot n \cdot n = n^4$, czyli rzędu $\Theta(n^4)$" (o ile jest to prawda). Nie ma potrzeby podawania pseudo-kodu.

- Użycie funkcji do pomiaru czasu (pod odpowiedni system) potrzebne jest do weryfikacji w/w oszacowania, tzn. wyznaczenia, ile ta procedura zajmuje czasu „zegarowego" (przez daną maszynę); tu można w pełni oprzeć się na implementacji tego pomiaru z zadania AiSD.1.1.

- Jeśli chodzi o ideę algorytmu, to powinna ona być spojrzeniem „z góry". Znów dla przykładu, jeśli mielibyśmy „cztery zagnieżdżone pętle", to ważne jest, za co te pętle tak naprawdę odpowiadają. Chodzi o wyjaśnienie, że np. „te cztery zagnieżdżone pętle" mogą odpowiadać za „generowanie (współrzędnych) punktów w przestrzeni trójwymiarowej (trzy pierwsze pętle), a następnie sprawdzenie własności $X$ dla każdego z wygenerowanych punktów (czwarta pętla)". Więc nie chodzi o przełożenie języka implementacji na język polski, a o wyjaśnienie, jaka (w sensie ogólnym) jest teoretyczna idea rozwiązania. Analiza kodu i zrozumienie kryjącej się za nim idei algorytmu jest zazwyczaj trudnym zadaniem, niemniej warto spróbować (i przekonać się o tym).