

ZADANIA Z JĘZYKA C DLA GRUP 2., 7. I 9.

Zestaw V - grudzień 2019

15. **Obliczanie liczby e metodą Monte Carlo.** Podobnie jak w Zadaniu 10., w którym obliczaliśmy liczbę π generując długi ciąg par liczb pseudolosowych (x, y) i korzystając z faktu, że — dla liczb o rozkładzie równomiernym ($0 \leq x < 1$, $0 \leq y < 1$) — prawdopodobieństwo trafienia w koło o promieniu 1 wynosi $\pi/4$, możliwe jest skonstruowanie innego *procesu pseudolosowego*, w którym prawdopodobieństwo sukcesu powiązane będzie z podstawą logarytmów naturalnych, tzw. liczbą Eulera (lub Nepera), $e \approx 2.718281828459$.

Okazuje się, że dla losowej permutacji zbioru n elementów (dla ustalenia uwagi: np. *liczb naturalnych* $0, 1, 2, \dots, n-1$) prawdopodobieństwo, że **żaden element nie pozostanie na swoim miejscu** wynosi

$$P_0(n) = 1 - 1 + \dots + \frac{(-1)^n}{n!},$$

co jest numerycznie bliskie $1/e \approx 0.36788$ dla dużych n . (Przykładowo, dla $n = 100$ zgadzać się będzie pierwsze 159 cyfr po przecinku.) Dowód powyższego wzoru można znaleźć np. w podręczniku W. Lipski, *Kombinatoryka dla programistów*, rozdz. 1.13, lub Graham, Knuth, Patashnik, *Matematyka konkretna*, rozdz. 5.3.

Tym razem, **zadanie polega na** napisaniu programu, który generuje ciąg przypadkowych permutacji liczb $0, 1, 2, \dots, n-1$, dla każdej permutacji określa liczbę punktów stałych k , oraz wylicza, ile razy w ciągu N permutacji mamy $k = 0$. Ułamek $N/N_{k=0}$ stanowi oszacowanie liczby e .

Technicznie, generowanie można zacząć od permutacji identycznościowej, którą w pamięci komputera reprezentować będzie tablica *zadeklarowana* następująco:

```
#define n 100
int perm[n];
```

a następnie zainicjowana wartościami:

```
for (j=0; j<n; j++)
    perm[j]=j;
```

Dla takiej permutacji liczba punktów stałych: **k==n**. Można powiedzieć, że nasze działania będą teraz odwrotne do wykonywanych w algorytmie sortowania bąbelkowego (zob. *wyklad04*), chcemy bowiem w naszej początkowo uporządkowanej tablicy wprowadzić *maksymalny bałagan*. Taki cel można osiągnąć, wykonując dużą liczbę następujących kroków:

- a) Losujemy całkowite j z przedziału $0, 1, \dots, n-2$.

- b) Zamieniamy elementy `perm[j]` oraz `perm[j+1]`.
 c) Określamy nową liczbę punktów stałych wg przepisu:

$$\begin{aligned} k += & (\text{perm}[j]==j) + (\text{perm}[j+1]==j+1) \\ & - (\text{perm}[j]==j+1) - (\text{perm}[j+1]==j); \end{aligned}$$

Kroki (a)–(b) powtarzamy np. $N = 10^6$ razy zliczając, ile razy wystąpi $k==0$.

Jak widać, przestawiamy zawsze najbliższych sąsiadów, a zatem określenie nowego k wymaga obliczania wyrażeń odwołujących się wyłącznie do dwóch zamienionych elementów; dzięki temu algorytm jest szybki. Pewna wada przedstwowego podejścia wiąże się z faktem, że kolejne permutacje są do siebie bardzo podobne, potrzeba wielu powtórzeń kroków (a)–(b) aby charaterystyki takie jak k można było uznać za statystycznie niezależne. W szczególności, pewna liczba permutacji wygenerowanych jako pierwsze będzie *obciążona* elementami permutacji identycznościowej. Zasadniczo, te początkowe permutacje należy pominąć w obliczeniach, aby oszacowanie liczby e było *nieobciążone* wyborem permutacji początkowej.

W naszym przypadku, możemy przyjąć, że po pierwszym wystąpieniu $k==0$ wszelkie ślady po permutacji identycznościowej uległy zatarciu. Jako *lepsze oszacowanie liczby e* przyjmujemy zatem ułamek $\tilde{N}/\tilde{N}_{k=0}$, w którym licznik i mianownik oznaczają, odpowiednio: liczbę wszystkich permutacji, oraz l. permutacji bez punktów stałych, liczone *po* pierwszym wystąpieniu $k==0$.

Proszę sporządzić wykresy pokazujące, jak zmienia się $N/N_{k=0}$ oraz $\tilde{N}/\tilde{N}_{k=0}$ w zależności całkowitej liczby iteracji N .

Aby oszacować niepewność statystyczną naszego przybliżenie liczby e , całą symulację (dla wybranego N) można powtórzyć 20 – 30 razy (startując z różnymi podstawami generatora liczb pseudolosowych), a następnie dla serii wyników $\tilde{N}/\tilde{N}_{k=0}$ obliczyć średnią i odchylenie standardowe.