Opis zadania

Zbadać, czy jakaś wartość w tablicy jest dominująca: to znaczy czy więcej niż połowa wyrazów tablicy jest jej równa. Nie zakładamy niczego o zbiorze, z którego pochodzą elementy tablicy – nie można więc wykonywać na nich żadnych operacji arytmetycznych czy ich porównywać relacją porządku. Możemy jedynie stwierdzać czy są one sobie równe, co jest wykonywane w czasie stałym – O(1). Należy też zbadać złożoność opracowanego przez siebie algorytmu lub algorytmów. Metody testowania i oceny złożoności powinny być precyzyjnie opisane.

Sposób wykonania programu

By wykonać jak najlepiej zadanie wykonałem kilka funkcji określających, czy w tablicy o długości N znajduje się jakaś wartość dominująca.

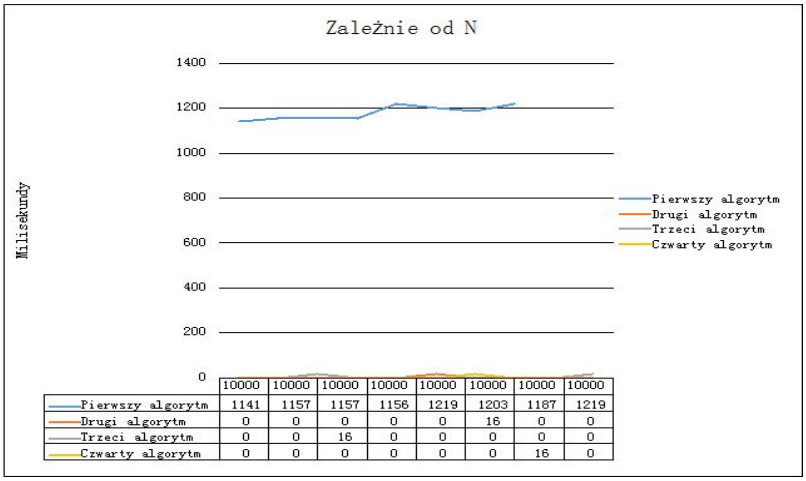
Wartości jakie mogły przyjąć elementy tablicy, były liczbami całkowitymi dodatnimi, pseudolosowymi z zakresu Z tj. [0,z). W związku z tym, przy próbach z Z innym od 2, element dominujący najczęściej nie występował.

Pierwsza funkcja(algorytm) wykorzystuje najbardziej „toporny” algorytm.

```
for(int i = 0; i < n; i++){
    for(int j = 0; j < n; j++){
        if(t[i] == t[j]){
            licznik++;
        }
    }
    if( (licznik*2) > n){
        wartosc = t[i];
        ile = licznik;
        //cout << "Najwiecej war
    }
    licznik = 0;
}
```

Pętla zewnętrzna „przegląda” kolejno wszystkie elementy tablicy wejściowej, a pętla wewnętrzna przegląda ile elementów jest równych t[i]. Algorytm ten jak możemy wywnioskować działa tylko przy niedużych wielkościach wejściowej tablicy.

Na poniższym wykresie widzimy jak przy tablicy o 10000 elementów, czas wykonania programu oscyluje w okolicach 1150-1200 milisekund. Dlatego pominąłem ten algorytm przy dalszych działaniach.



	10000	10000	10000	10000	10000	10000	10000	10000
Pierwszy algorytm	1141	1157	1157	1156	1219	1203	1187	1219
Drugi algorytm	0	0	0	0	0	16	0	0
Trzeci algorytm	0	0	16	0	0	0	0	0
Czwarty algorytm	0	0	0	0	0	0	16	0

Druga funkcja(algorytm) wykorzystuje pomocniczą tablice możliwych wartości tablicy wejściowej.

Tablica wejściowa „tablica” jest przeszukiwana tylko raz dla każdej możliwej jej wartości, przy okazji zliczając ilość wystąpień danej wartości.

```
for(int i=0; i<z; i++){
    for(int k=0; k<n; k++){
        if(i == tablica[k]){
            tablica2[i] += 1;
            //cout << "id " << ta
        }
    }
}
```

Na koniec sprawdza, czy któryś element tablicy „tablica2” (pomocniczej) wystąpił wystarczająco dużo razy, by być elementem dominującym.

Trzecia funkcja(algorytm) działa na tej samej zasadzie co druga, jednakże nie wykorzystuje tablicy pomocniczej do zliczania wystąpień, a jedynie zmienną. W tym przypadku wiemy jedynie ile razy wystąpił najczęściej powtarzający się element tablicy i czy jest on elementem dominującym.

```
for(int i = 0; i < z; i++){
    for(int j = 0; j < n; j++){
        if(i == t[j]){
            licznik++;
        }
    }
    if( (licznik*2) > n){
        wartosc = t[i];
        ile = licznik;
        //cout << "Najwiecej wart
    }
    licznik = 0;
}
```

Czwarta funkcja(algorytm) „dzieli” tablice wejściową na mniejsze o długości:

Początkowo - <0, z)

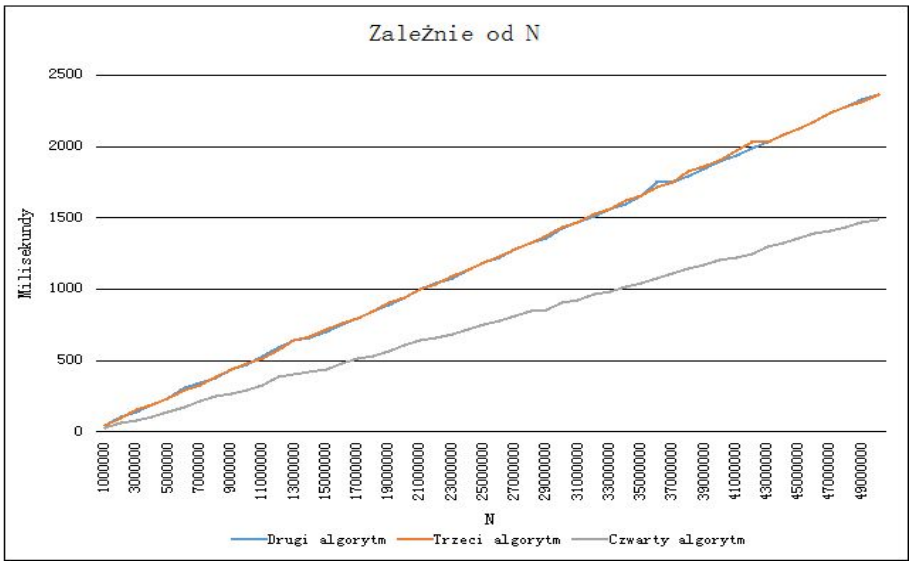
Następnie - <k, k+l) , gdzie l jest zmienna iteracyjna w zakresie <0,z) i k jest powiększane w każdym obiegu pętli o 3.

```
while(k < n){
    //cout << "Sprawdzam: ";
    for(int l = 0; l < z ; l++){
            if( (k+l) < n ){  // Za
           // cout <<"[" <<k+l<<"]"
            t2[t[k+l]] += 1;
            }
    }
   // cout << endl;
    k += z;
}
```

Algorytm wykorzystuje również tablice pomocniczą t2 do zliczania wystąpień danych elementów tablicy wejściowej.

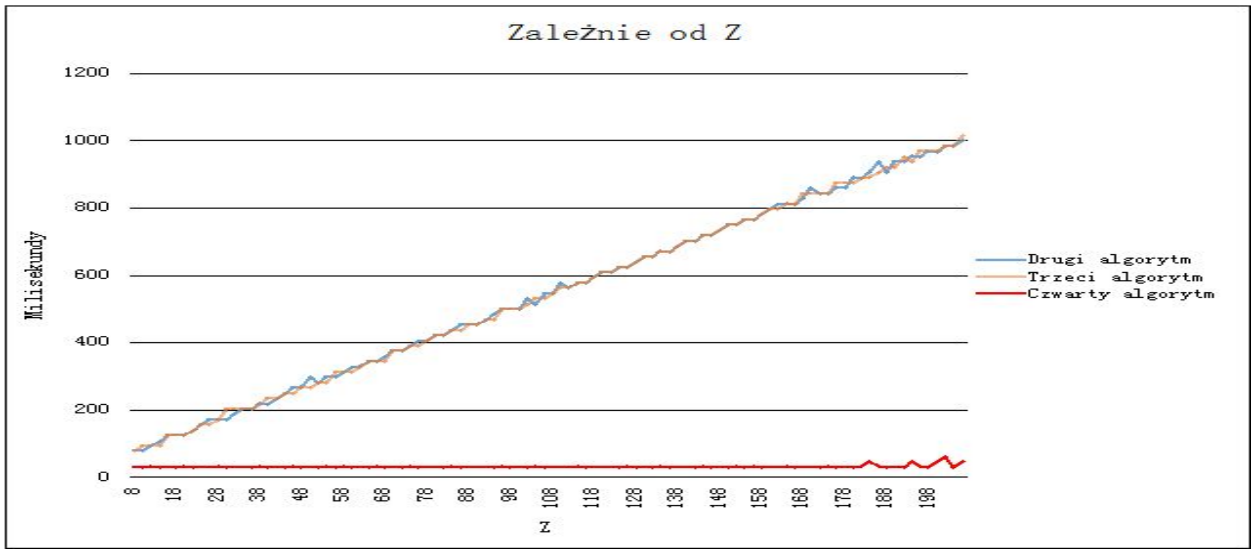
Wykresy z wynikami badań

Badamy czas wykonania algorytmów zależnie od wielkości tablicy wejściowej (tj. od N). Skok N: 1000000.



Czas wykonania drugiego i trzeciego algorytmu, jak widzimy jest prawie identyczny.

Badamy czas wykonania algorytmów zależnie od wielkości zakresu losowanych liczb pseudolosowych, czyli od ilości możliwych wartości elementów tablicy wejściowej (tj. od Z). Skok Z: 2.



Wnioski

Jak widzimy ostatni algorytm okazuje się znacznie szybszy w drugim przypadku badań, ponieważ nie jest zależny od zakresu Z. Na jego korzyść działa fakt, że „dzieli” tablicę na mniejsze.

Wymyślone i badane przeze mnie algorytmy są złożoności obliczeniowej liniowej O(n), co uważam za dobry efekt.

Michał Błaszczyk

Nr albumu 345096