## Zajęcia 19

**Temat:** Struktury danych 2

**Czas trwania:** 2x45 min

**Cel zajęć:**

projektuje i programuje proste problemy z różnych dziedzin, stosuje przy tym: instrukcje wejścia/wyjścia, wyrażenia arytmetyczne i logiczne, instrukcje warunkowe, instrukcje iteracyjne, tablice, rekurencję, pisze własne funkcje rekurencyjne, zbiory i operacje na zbiorach, struktury danych, biblioteka STL, testuje poprawność programów dla różnych danych, posługuje się zintegrowanym środowiskiem programistycznym przy pisaniu, uruchamianiu i testowaniu programów;

**Efekty:**

- umie uruchomić potrzebne oprogramowanie,
- umie napisać program z wykorzystaniem funkcji rekurencyjnej, równaniami rekurencyjnymi,
- zna struktury danych: stos, kolejkę, vector, pair, kolejka priorytetowa

**Formy i metody pracy:** praca samodzielna, pogadanka, dyskusja, omówienie

| Zadania do wykonania na zajęciach | Treści programowe |
|---|---|
| 1. Kamienie | M.3, P.2.16, P.2.18, A.4.1, A.4.2 |
| 2. Loginy | M.3, P.2.16, P.2.18, P.2.20, A.4.2, A.4.5 |
| 3. Izba przyjęć | M.3, P.2.16, P.2.18, P.2.20, A.4.5, A.4.9 |

**Materiały do zajęć**

https://en.cppreference.com/w/

**Zadania do wykonania w domu:**

**Kodowanie permutacji (II OI):**

https://szkopul.edu.pl/problemset/problem/ioeAqb7fRkdNzdUVS0cvMSKm/site/

**Samochodziki (XII OI):**

https://szkopul.edu.pl/problemset/problem/DNXEM9WX4TRCI0fFbWedf1qq/site/

# ZADANIA I ROZWIĄZANIA:

## Zadanie 1. Kamienie

Limit pamięci: 64MB

Janek jest kolekcjonerem kamieni. Zbiera je i kataloguje, nadając każdemu unikatowy numer. Dba o to, by każdy kamień był inny i nie pozwala, aby w kolekcji znalazły się dwa takie same. Kiedy chodzi plażą, gdzie może znaleźć nowe okazy do swoich zbiorów, podnosi każdy napotkany kamień, ocenia go, nadaje mu numer katalogowy i – jeśli jeszcze takiego w kolekcji nie posiada – zabiera go. Trudno jest mu jednak szybko ocenić, czy kolejny znaleziony kamień jest potrzebny. Pomóż mu!

### Wejście

W pierwszym wierszu standardowego wejścia zapisana jest jedna liczba całkowita $n$ ($2 \le n \le 1\,000\,000$) oznaczająca liczbę kamieni. Kolejne $n$ wierszy zawiera po jednej liczbie całkowitej $k_i$ ($0 \le k_i \le 10^9$) oznaczającej numer katalogowy $i$-tego kamienia.

### Wyjście

W $n$ wierszach standardowego wyjścia Twój program powinien zapisać po jednej literze ‘T’ lub ‘N’, oznaczającą przydatność kamienia w kolekcji Janka.

### Przykład

| Wejście | Wyjście |
|---|---|
| 5<br>1<br>2<br>4<br>2<br>5 | T<br>T<br>T<br>N<br>T |

### Rozwiązanie

Bardzo proste zadanie, którym możemy użyć struktury `set` z biblioteki STL.

```
wczytaj n
dla i=1,2,…,n
      wczytaj x
      jeżeli x nie należy do zbioru
            wypisz ‘T’
            dodaj x do zbioru
      przeciwnie
            wypisz ‘N’
```

Jeżeli będzie to wskazane, można wytłumaczyć na zajęciach ideę drzew BST (zasada działania, bez implementacji) oraz drzew samorównoważących się w odniesieniu do kontenerów set i map z biblioteki STL wraz z informacją o złożoności obliczeniowej poszczególnych operacji (wstawienia, usunięcia, szukania, przejścia do poprzedniego/następnego).

## Zadanie 2. Loginy

Dostępna pamięć: 128MB

Pan Integer pracuje nad otwarciem nowego portalu internetowego. Jednym z elementów systemu jest poczta elektroniczna. Pan Integer rozdzielił już pracę pomiędzy zespoły. Twoim zadaniem będzie opracowanie modułu rejestracji loginów.

Moduł rejestracji otrzymuje proponowany login. Jeśli taka nazwa jeszcze nie wystąpiła w bazie, moduł wysyła informację *OK* do użytkownika i zapisuje go w bazie. Jeśli podana nazwa już występuje w bazie, wówczas tworzony jest nowy login, system wysyła go do użytkownika i zapisuje w bazie. Nowy login tworzony jest przez dodanie kolejnego jak najmniejszego numeru do proponowanej przez użytkownika nazwy.

### Wejście

W pierwszej linii wejścia znajduje się jedna liczba całkowita $n$ ($1 \ n \ 5 \cdot 10^5$), oznaczająca liczbę rozpatrywanych loginów.

W kolejnych $n$ liniach znajdują się rozpatrywane loginy. Każdy login składa się wyłącznie z małych liter alfabetu łacińskiego i jest nie dłuższy niż 30 znaków.

### Wyjście

Na wyjściu dla każdego loginu wypisywana jest odpowiedź wysyłana dla użytkownika: *OK* – w przypadku udanej rejestracji, lub nowy login dla już istniejącej nazwy.

### Przykład

| Wejście | Wyjście |
|---|---|
| 6<br>mirek<br>katarzyna<br>kirek<br>mirek<br>nowy<br>mirek | OK<br>OK<br>mirek1<br>mirek2<br>OK<br>mirek3 |

### Ocenianie

| Podzadanie | Ograniczenia | Punkty |
|---|---|---|
| 1 | $n \leq 10^3$ | 30 |
| 2 | brak dodatkowych założeń | 70 |

### Rozwiązanie

Bardzo proste zadanie, którym możemy użyć struktury `map` z biblioteki STL.

Zauważmy, że kluczem w naszej mapie będzie wczytany login, zaś wartością liczba dotychczas użytych takich loginów. Dla ułatwienia kod programu z komentarzami:

```
string s;
//lista loginów
map<string,int>a;
//liczba zapytań
```

```
cin>>n;
//dla każdego zapytania
for (int i=0; i<n; i++){
  //wczytaj oczekiwany login
  cin >> s;
  //jeżeli podany login istnieje, wartość elementu
  //o podanym kluczu będzie większa od 0
  if ( a[s]>0 )
  //wówczas wypisujemy login wraz z liczbą dotychczasowych wystąpień
    cout << s << a[s] << "\n";
  //jeżeli login jeszcze nie wystąpił
  else
    //wypisz login bez zmian
    //cout << "OK" << "\n";
  //zwiększ liczność aktualnego loginu o 1
  a[s]++;
}
```

## Zadanie 3. Izba przyjęć

Dostępna pamięć: 256MB

Na izbie przyjęć chorzy pacjenci oczekują na przyjęcie z przejęciem patrząc na zegarek. Ale to nie czas decyduje o kolejności. Ważniejszy jest stan chorego. Dlatego często na SORach oznacza się pacjentów opaskami w kolorach czerwonym, żółtym i zielonym. Kolor czerwony oznacza, że pacjent musi być przyjęty w pierwszej kolejności - w trybie natychmiastowym. Taki kolor przydzielany jest osobom w bardzo złym stanie, często przywiezionym przez karetki. Kolor żółty jest przydzielany pacjentom, którzy wymagają pilnej interwencji lekarza. Kolorem zielonym są oznaczani pacjenci z najniższym stopniem zagrożenia życia, przyjmowani w trzeciej kolejności.

Pan Integer opracował dokładniejszy sposób określania stanu pacjentów. Wprowadził $k$ kolorów opasek. Jeśli mamy pacjentów z opaskami w kolorach $i$ oraz $j$ i wiemy, że $i<j$, to pacjent z opaską w kolorze $i$ zostanie przyjęty jako pierwszy. Dla pacjentów z tym samym kolorem opaski liczy się czas przyjęcia.

Pan Integer wymyślił system, który ma przyśpieszyć pracę pobliskiej izby przyjęć i obserwuje teraz, jak działa on w rzeczywistości. Co sekundę przychodzi nowy pacjent lub jeden z oczekujących pacjentów jest przyjmowany przez dyżurującego lekarza. Pan Integer potrafi natychmiast sklasyfikować przybyłych (określić kolor ich opasek). Nie nadąża jednak ze wskazywaniem pacjentów, którzy mają być w danym momencie obsłużeni. Czy potrafisz napisać program, który pomoże mu w pracy?

### Wejście

W pierwszej linii wejścia znajduje się jedna parzysta liczba całkowita $n$ ($1 \le n \le 2 \cdot 10^6$). W kolejnych $n$ liniach znajduje się po jednej liczbie całkowitej $ki$ oznaczającej kolor opaski pacjenta, który przyszedł w $i$-tej sekundzie ($1 \le k_i \le 109$) lub informacja o przyjęciu w tym momencie kolejnej osoby przez lekarza ($k_i = -1$).

Możesz założyć, że zawsze jest ktoś, kto oczekuje pomocy i że wszyscy pacjenci zostaną kiedy obsłużeni. Możesz również założyć, że testach ocenianych na 30 punktów zachodzi warunek $n \le 10^3$.

## Wyjście

Na wyjściu powinno znaleźć się *n* liczb całkowitych oznaczających kolejność przyjęcia pacjentów przez lekarza. Pacjentów oznacz czasem przyjścia na izbę przyjęć.

## Przykład

| Wejście | Wyjście |
|---|---|
| 10<br>3<br>5<br>2<br>-1<br>4<br>-1<br>1<br>-1<br>-1<br>-1 | 3<br>1<br>7<br>5<br>2 |

## Rozwiązanie

W tym zadaniu najwygodniejsze będzie użycie kolejki priorytetowej (struktura `priority_queue` z biblioteki STL). Nasza kolejka uwzględnia przede wszystkim stan pacjenta (kolor), a dopiero później jego czas przybycia. Zadeklarujmy strukturę, która będzie przechowywać te dwie informacje. Zauważmy, że kolejka priorytetowa przechowuje na początku element najważniejszy. Uwzględnimy ten fakt przeciążając operator mniejszości (wymaganie kolejki priorytetowej). Teraz nie zostało nam nic innego niż zasymulować pracę izby przyjęć.

Kod programu wraz z komentarzami:

```
struct pacjent {int kolor, czas;};
//pacjent pomocniczy
pacjent p;
priority_queue <pacjent> q;
//przeciążamy operator mniejszości, liczy się
//w pierwszej kolejności kolor, mniej ważny jest ten o większym
numerze
bool operator <(const pacjent &a, const pacjent &b){
  return (a.kolor>b.kolor) || (a.kolor==b.kolor && a.czas>b.czas);
}
```

Główna część programu:

```
//liczba zdarzeń na izbie przyjęć
cin>>n;
for (int i=1; i<=n; i++){
  //kolor pacjenta z sekundy i
  cin >> p.kolor;
  //jeżeli kolor jest równy -1, to informacja o przyjęciu
  //najbardziej potrzebującego pacjenta
  if (p.kolor==-1) {
    //zdejmujemy z kolejki pierwszego oczekującego
```

```
    p = q.top(); q.pop();
    //wypisz jego czas przyjęcia na izbę
    cout << p.czas << "\n";
  }
  //nowy pacjent
  else {
    //zapamiętaj jego czas przyjęcia
    p.czas=i;
    //wstaw go do kolejki oczekujących
    q.push(p);
  }
}
```