Assalammualaikum wr wb.

Mohammad Hanif Furqan Aufa Putra

5025221161

PERBAIKAN TUGAS 6 PBO C

Student's queue in a canteen

Time limit **1** second

Memory limit **128** MiB

In ADA University students like Programming competitions very much, so each student belongs to one (and only one) team. But the rules of different competitions are different, and not usually one team consists of 33 members like according to ICPC rules. Any team can contain any number of students (but of course more than 00).

Students like to come to their canteen which is a building C and spent their free time with a cup of coffee. Students in ADA are very smart, they do not want to stand in standard queue for delicious coffee. They decided to establish some rules that only they would follow.

If a student enters the queue, he first searches the queue from head to tail to check if some of his teammates (student of the same team) are already in the queue. If yes, he enters the queue right behind them (behind all his teammates). If not, he enters the queue at the tail and becomes the new last element (bad luck). Dequeuing is done like in normal queues: students are processed from head to tail in the order they appear in the queue.

Your task is to write a program that simulates such a queue.

Input data

First line contains number of teams t (1≤t≤1000)t (1≤t≤1000). Each of the next tt lines describes one team. First element in the line is the number n (1≤n≤1000)n (1≤n≤1000) of students in a team. Next nn integers in a line give the ID's (0≤ID≤106)(0≤ID≤106) of students in a team.

Finally, a list of commands follows. There are two different kinds of commands:

- ENQUEUE x — student xx enters into the queue
- DEQUEUE — process the first student and remove him from the queue

Output data

For each DEQUEUE command print the element which is dequeued on a single line.

Examples

Input example #1

```
2
3 1 2 3
```

```
3 4 5 6
ENQUEUE 1
ENQUEUE 4
ENQUEUE 2
ENQUEUE 5
ENQUEUE 6
ENQUEUE 3
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
```

Output example #1

```
1
2
3
4
5
6
```

Input example #2

```
2
3 1 2 3
3 4 5 6
ENQUEUE 1
ENQUEUE 4
ENQUEUE 2
DEQUEUE
DEQUEUE
ENQUEUE 5
ENQUEUE 3
ENQUEUE 6
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
```

Output example #2

```
1
2
4
5
6
3
```

Input example #3

```
3
3 11 12 13
3 24 25 26
3 47 48 49
ENQUEUE 11
ENQUEUE 47
ENQUEUE 48
ENQUEUE 12
ENQUEUE 24
ENQUEUE 49
DEQUEUE
DEQUEUE
DEQUEUE
```

```
ENQUEUE 13
DEQUEUE
DEQUEUE
DEQUEUE
```

Problem ini adalah sebuah simulasi antrian yang melibatkan para mahasiswa dari beberapa tim. Mahasiswa-mahasiswa tersebut bergabung dalam antrian berdasarkan aturan tertentu, dan tujuannya adalah untuk memproses masuk dan keluar mereka dengan benar.

Dibawah ini penjelasan langkah-demi-langkah bagaimana kode pertama bekerja dan bagaimana ia mensimulasikan masalah yang diberikan untuk setiap input:

1. Pertama, kode ini membaca jumlah tim (t).

2. Kode ini menginisialisasi antrian bernama `team` dan sebuah array antrian bernama `each_team`. Antrian `team` digunakan untuk melacak urutan keberadaan tim-tim dalam antrian utama. Array `each_team` dari antrian menyimpan mahasiswa-mahasiswa dari masing-masing tim.

3. Kode ini menggunakan peta yang disebut `on_team` untuk memetakan ID mahasiswa ke nomor tim mereka. Peta ini membantu menentukan tim mana yang dimiliki oleh seorang mahasiswa berdasarkan ID-nya.

4. Kemudian kode memasuki loop while, yang akan terus berjalan hingga mencapai akhir input (EOF).

5. Di dalam loop, kode membaca perintah (ENQUEUE atau DEQUEUE) dan memprosesnya sesuai.

6. Jika perintah adalah DEQUEUE:

- Kode mengambil nomor tim mahasiswa yang berada di depan antrian utama menggunakan `team.front()`.

- Mencetak ID mahasiswa yang berada di depan antrian tim yang sesuai menggunakan `each_team[can_team].front()`.

- Menghapus mahasiswa tersebut dari depan antrian tim (`each_team[can_team].pop()`).

- Jika antrian tim menjadi kosong, maka tim tersebut dihapus dari antrian utama (`team.pop()`).

7. Jika perintah adalah ENQUEUE:

- Kode membaca ID mahasiswa.

- Kode menemukan nomor tim mahasiswa menggunakan `on_team[id]`.

- Jika antrian tim kosong (artinya tidak ada anggota tim dalam antrian utama), maka tim tersebut ditambahkan ke dalam antrian `team`.

- Kemudian kode menambahkan ID mahasiswa ke dalam antrian tim (`each_team[id_team].push(id)`).

Eksekusi langkah demi langkah:

1. Membaca jumlah tim (t). Dalam hal ini, t = 2.

2. Menginisialisasi struktur data dan mengisi peta `on_team` dengan ID-ID mahasiswa dan nomor tim mereka yang sesuai.

3. Memasuki loop while untuk memproses perintah.

- ENQUEUE 1: Mahasiswa 1 dari tim 1 masuk ke dalam antrian.

- ENQUEUE 4: Mahasiswa 4 dari tim 2 masuk ke dalam antrian.

- ENQUEUE 2: Mahasiswa 2 dari tim 1 masuk ke dalam antrian.

- ENQUEUE 5: Mahasiswa 5 dari tim 2 masuk ke dalam antrian.

- ENQUEUE 6: Mahasiswa 6 dari tim 2 masuk ke dalam antrian.

- ENQUEUE 3: Mahasiswa 3 dari tim 1 masuk ke dalam antrian.

4. DEQUEUE: Memproses mahasiswa pertama dalam antrian.

- Mahasiswa tersebut berasal dari tim 1, jadi kita mengeluarkan dan mencetak 1.

5. DEQUEUE:

- Mahasiswa berikutnya dalam antrian juga berasal dari tim 1, jadi kita mengeluarkan dan mencetak 2.

6. DEQUEUE:

- Mahasiswa berikutnya dalam antrian berasal dari tim 2, jadi kita mengeluarkan dan mencetak 3.

7. DEQUEUE:

- Mahasiswa berikutnya dalam antrian berasal dari tim 1, jadi kita mengeluarkan dan mencetak 4.

8. DEQUEUE:

- Mahasiswa berikutnya dalam antrian berasal dari tim 2, jadi kita mengeluarkan dan mencetak 5.

9. DEQUEUE:

- Mahasiswa terakhir dalam antrian berasal dari tim 2, jadi kita mengeluarkan dan mencetak 6.

Sedangkan dalam kode kedua yang menggunakan vector of queue

Langkah-langkah:

1. Membaca jumlah tim (t). Dalam kasus ini, t = 2.

2. Menginisialisasi struktur data dan mengisi peta `on_team` dengan ID-ID mahasiswa dan nomor tim mereka yang sesuai.

3. Menginisialisasi array antrian `each_team` dengan ukuran `t` (jumlah tim). Setiap elemen array ini mewakili antrian anggota tim tertentu.

4. Memasuki loop while untuk memproses perintah.

- ENQUEUE 1: Mahasiswa 1 dari tim 1 masuk ke dalam antrian tim 1.

- ENQUEUE 4: Mahasiswa 4 dari tim 2 masuk ke dalam antrian tim 2.

- ENQUEUE 2: Mahasiswa 2 dari tim 1 masuk ke dalam antrian tim 1.

- DEQUEUE: Memproses mahasiswa pertama dalam antrian.

- Mahasiswa tersebut berasal dari tim 1, jadi kita mengeluarkan dan mencetak 1.

- DEQUEUE:

- Mahasiswa berikutnya dalam antrian juga berasal dari tim 1, jadi kita mengeluarkan dan mencetak 2.

- DEQUEUE:

- Mahasiswa berikutnya dalam antrian berasal dari tim 2, jadi kita mengeluarkan dan mencetak 4.

- ENQUEUE 5: Mahasiswa 5 dari tim 2 masuk ke dalam antrian tim 2.

- ENQUEUE 6: Mahasiswa 6 dari tim 2 masuk ke dalam antrian tim 2.

- ENQUEUE 3: Mahasiswa 3 dari tim 1 masuk ke dalam antrian tim 1.

- DEQUEUE:

  - Mahasiswa berikutnya dalam antrian berasal dari tim 1, jadi kita mengeluarkan dan mencetak 3.

- DEQUEUE:

  - Mahasiswa berikutnya dalam antrian berasal dari tim 2, jadi kita mengeluarkan dan mencetak 5.

- DEQUEUE:

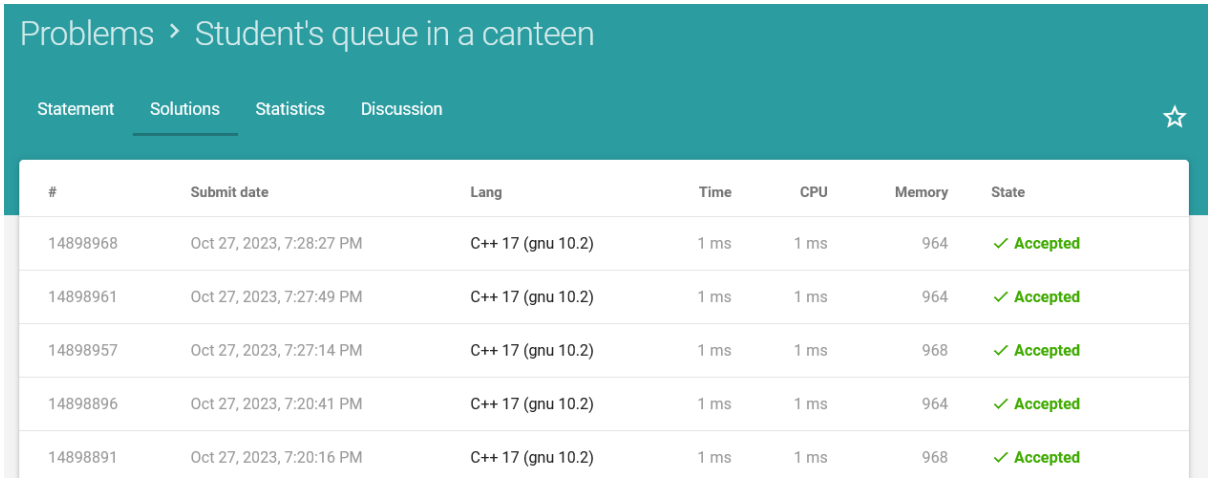
  - Mahasiswa terakhir dalam antrian berasal dari tim 2, jadi kita mengeluarkan dan mencetak 6.

Perbedaan antara kode pertama dan kode kedua terletak pada cara struktur data dikelola. Kode kedua menggunakan vektor antrian `each_team`, yang dibuat sebelumnya dengan ukuran `t`. Dengan menggunakan vektor, kode kedua menghindari penggunaan `map` dan mungkin lebih efisien dalam pengelolaan memori dan akses ke antrian anggota tim.

Namun, perbedaan tersebut tidak selalu membuat kode kedua lebih cepat secara signifikan. Kinerja kode bisa dipengaruhi oleh banyak faktor, termasuk implementasi compiler dan platform tempat kode dijalankan. Lebih baik menguji kinerja sebenarnya di berbagai situasi sebelum menentukan perbedaan performa yang signifikan.

Hal ini dibuktikan dari hasil run untuk tiap kode selama 5x seperti yang terlampir dibawah ini

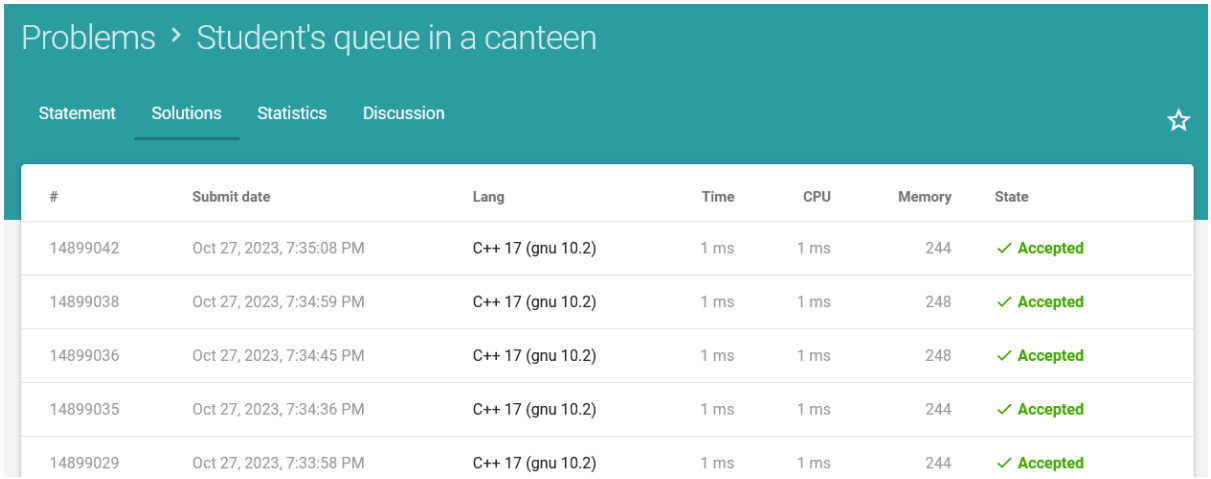
Hasil Verdict AC Kode dari Bapak:



Dengan rata-rata: Waktu = 1ms, CPU = 1ms, Memory = 965.6MiB

Sedangkan saat menggunakan Vector of Queue didapatkan hasil seperti yang terlampir dibawah ini

Hasil Verdict AC Kode Vector of Queue:



Problems › Student's queue in a canteen

Statement | Solutions | Statistics | Discussion

#	Submit date	Lang	Time	CPU	Memory	State
14899042	Oct 27, 2023, 7:35:08 PM	C++ 17 (gnu 10.2)	1 ms	1 ms	244	✓ Accepted
14899038	Oct 27, 2023, 7:34:59 PM	C++ 17 (gnu 10.2)	1 ms	1 ms	248	✓ Accepted
14899036	Oct 27, 2023, 7:34:45 PM	C++ 17 (gnu 10.2)	1 ms	1 ms	248	✓ Accepted
14899035	Oct 27, 2023, 7:34:36 PM	C++ 17 (gnu 10.2)	1 ms	1 ms	244	✓ Accepted
14899029	Oct 27, 2023, 7:33:58 PM	C++ 17 (gnu 10.2)	1 ms	1 ms	244	✓ Accepted

Dengan rata-rata: Waktu = 1ms, CPU = 1ms, Memory = 245.6MiB

Dan dibawah ini merupakan Kode dari Bapak dan kode vector of queue

Kode dari Bapak:

```cpp
#include <map>
#include <cstdio>
#include <queue>
#include <algorithm>
using namespace std;
queue<int> team, each_team[1001];
map<int, int> on_team;

int main()
{
    int t;
    scanf("%d", &t);
    for (int i = 0; i < t; i++)
    {
        int n;
        scanf("%d", &n);
        for (int j = 0; j < n; j++)
        {
            int id;
            scanf("%d", &id);
            on_team[id] = i;
        }
    }
    char ch[20];
    while (scanf("%s", ch) != EOF)
    {
        if (ch[0] == 'D')
        {
            int can_team = team.front();
            printf("%d\n", each_team[can_team].front());
            each_team[can_team].pop();
            if (each_team[can_team].empty())
            {
                team.pop();
            }
        }
        else
        {
            int id;
            scanf("%d", &id);
            int id_team = on_team[id];
            if (each_team[id_team].empty())
            {
                team.push(id_team);
            }
            each_team[id_team].push(id);
        }
    }
    return 0;
}
```

Kode untuk Vector of Queue:

```cpp
#include <map>
#include <cstdio>
#include <queue>
#include <algorithm>
using namespace std;

vector<queue<int>> each_team;
map<int, int> on_team;
queue<int> team;

int main()
{
    int t;
    scanf("%d", &t);

    each_team.resize(t);
    for (int i = 0; i < t; i++)
    {
        int n;
        scanf("%d", &n);
        for (int j = 0; j < n; j++)
        {
            int id;
            scanf("%d", &id);
            on_team[id] = i;
        }
    }
    char ch[20];
    while (scanf("%s", ch) != EOF)
    {
        if (ch[0] == 'D')
        {
            int can_team = team.front();
            printf("%d\n", each_team[can_team].front());
            each_team[can_team].pop();
            if (each_team[can_team].empty())
            {
                team.pop();
            }
        }
        else
        {
            int id;
            scanf("%d", &id);
            int id_team = on_team[id];
            if (each_team[id_team].empty())
            {
                team.push(id_team);
            }
            each_team[id_team].push(id);
        }
    }
    return 0;
}
```

Dengan menjalan program tersebut diperoleh hasil seperti dibawah ini dengan menggunakan test case yang saya coba buat

Success #stdin #stdout 0.01s 5540KB

stdin

```
2
3 7 8 11
3 4 5 6
ENQUEUE 7
ENQUEUE 4
DEQUEUE
ENQUEUE 5
ENQUEUE 8
ENQUEUE 6
DEQUEUE
ENQUEUE 11
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
```

stdout

```
7
4
5
6
8
11
```

Success #stdin #stdout 0.01s 5292KB

stdin

```
2
3 7 8 11
3 4 5 6
ENQUEUE 7
ENQUEUE 4
DEQUEUE
ENQUEUE 5
ENQUEUE 8
ENQUEUE 6
DEQUEUE
ENQUEUE 11
DEQUEUE
DEQUEUE
DEQUEUE
DEQUEUE
```

stdout

```
7
4
5
6
8
11
```

Kode Bapak

Kode Vector of Queue

Mohon maaf karena telat dan baru bisa mengumpulkan sekarang

Terima Kasih telah membaca, Waalaikumsalam wr wb.