Queues

Ulises Méndez Martínez

Algorist Weekly Talks

ulisesmdzmtz@gmail.com

May 13, 2016

Overview

FIFO or LILO?

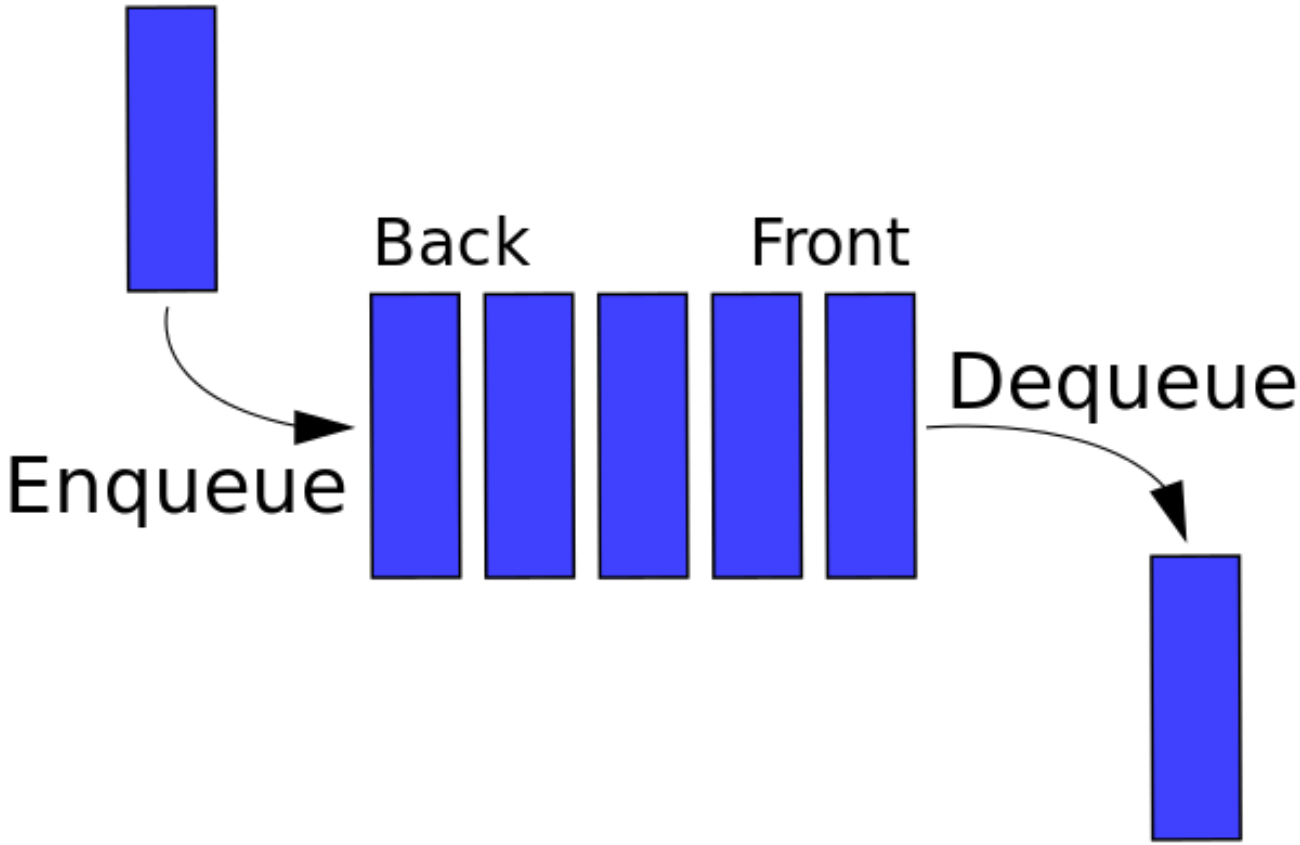

Figure : Representation of a FIFO (first in, first out) queue

FIFO

In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once a new element is added, all elements that were added before have to be removed before the new element can be removed.

C++ STL Queue

std::queue

template <class T, class Container = deque<T> > class queue;

FIFO queue

queues are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

- empty
- size
- front
- back
- push_back
- pop_front

The standard container classes deque and list fulfill these requirements. By default, if no container class is specified for a particular queue class instantiation, the standard container deque is used.

Throwing cards away I (Easy)

Description

Given is an ordered deck of n cards numbered 1 to n with card 1 at the top and card n at the bottom. The following operation is performed as long as there are at least two cards in the deck:
Throw away the top card and move the card that is now on the top of the deck to the bottom of the deck.
Your task is to find the sequence of discarded cards and the last, remaining card.

Simulation: $n = 6$

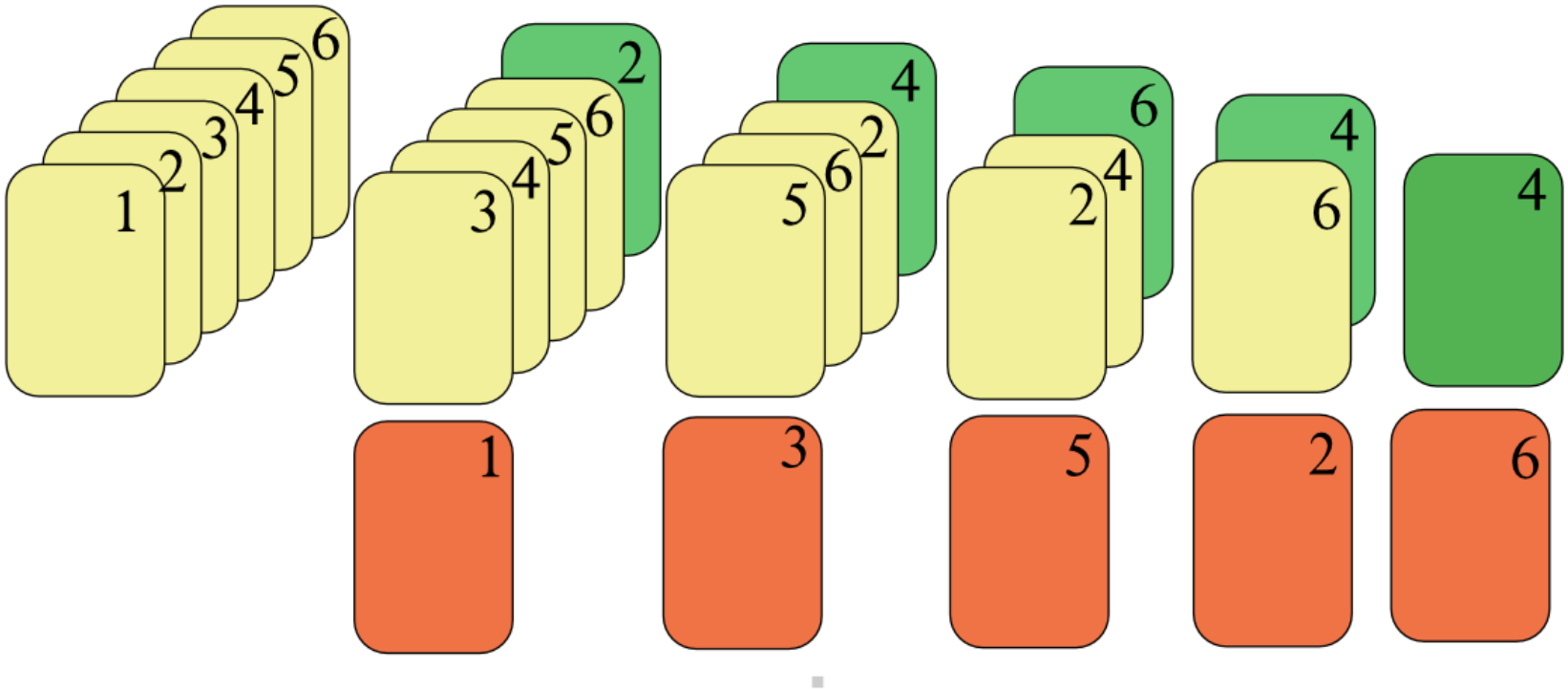

Figure : Discarded cards: 1, 3, 5, 2, 6

Figure : Remaining card: 4

Solution using queue: C++

Example (C++ Implementation)

```
while(scanf("%d",&n), n) {
    queue <int> Q;
    for(card=1,cnt=0; card<=n; card++)
        Q.push(card);
    printf("Discarded␣cards:");
    while(Q.size() > 1) {
        card = Q.front(); Q.pop();
        printf("%s%d",(cnt++?",␣":"␣"),card);
        card = Q.front(); Q.pop();
        Q.push(card);
    }
    printf("\nRemaining␣card:␣%d\n",Q.front());
}
```

Priority Queue

Wikipedia

- In computer science, a priority queue is an abstract data type which is like a regular queue or stack data structure, but where additionally each element has a "priority" associated with it. In a priority queue, an element with high priority is served before an element with low priority.
- While priority queues are often implemented with heaps, they are conceptually distinct from heaps. A priority queue is an abstract concept like "a list" or "a map";

C++ STL Priority Queue

std::priority_queue

template <class T, class Container = vector<T>,
class Compare = less<typename Container::value_type>> class priority_queue;

Priority queue

Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains.

The container shall be accessible through random access iterators and support the following operations:

- empty
- front
- pop_front
- size
- push_back

The standard container classes vector and deque fulfill these requirements. By default, if no container class is specified for a particular priority_queue class instantiation, the standard container vector is used.

Jesse and Cookies (Medium)

Description

Jesse loves cookies. He wants the sweetness of all his cookies to be greater than value K. To do this, Jesse repeatedly mixes two cookies with the least sweetness. He creates a special combined cookie with:

$$\text{sweetness} = (1x\text{Least sweet cookie} + 2x\ \text{2nd least sweet cookie}).$$

He repeats this procedure until all the cookies in his collection have a sweetness $\geq K$.

You are given Jesse's cookies. Print the number of operations required to give the cookies a sweetness $\geq$ K. Print -1 if this isn't possible.

Simulation: $N = 6, K = 7$

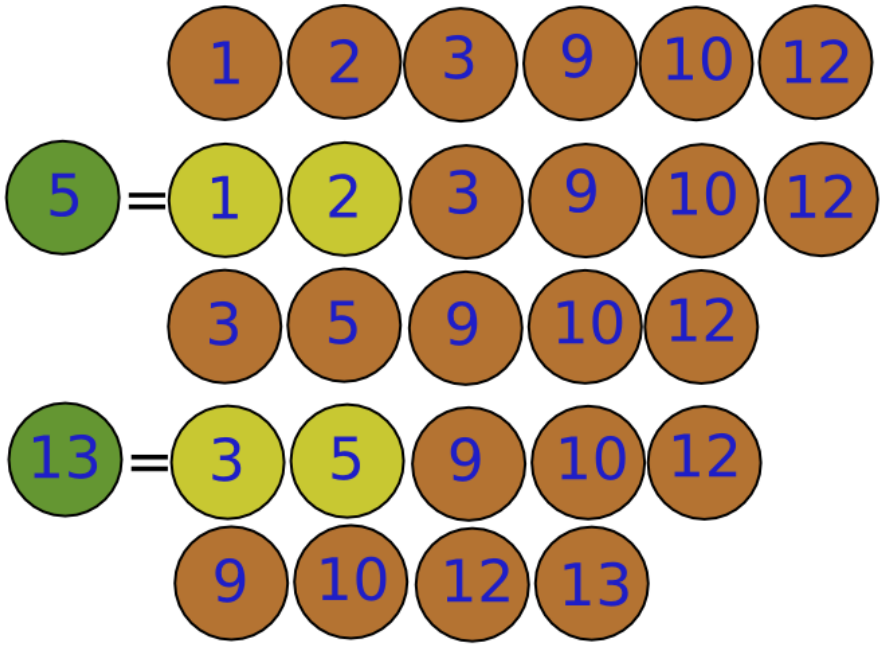

Figure : Output: 2

Solution using priority queue: C++

Example (C++ Implementation)

```
priority_queue<i64, vector<i64>, greater<i64> > PQ;
scanf("%d %lld",&n,&k);
for(int i=0; i<n; i++) {
    scanf("%lld",&cookie);
    PQ.push(cookie);
}
while(PQ.size() > 1) {
    if(PQ.top() >= k) break;
    first = PQ.top(); PQ.pop();
    second = PQ.top(); PQ.pop();
    cookie = first + second * 2LL;
    PQ.push(cookie); cnt++;
}
printf("%d\n",(PQ.top()<k)?-1:cnt);
```