Wrap around

in Java

Algorithms and data structures ID1021

Johan Montelius

Spring 2026

Introduction

The easiest way to implement a queue is to use a linked list and add and remove elements from it's two ends. This might however be expensive depending on the price for memory allocation etc. An alternative approach is to use an array that can use the memory more efficiently; this approach is as you will see more complicated.

Using an array

In the same way as you implemented a stack both by using a linked list and using an array you can implement a queue using an array. The idea is simple but the implementation will probably take you some time since there are many corner cases. Before you start coding you need to draw pictures of the different situations and only then do the coding. You could of course cheat and google for "queue array implementation" to look at any of the six million examples but if you do it from scratch yourself, you will learn so much more so give it a try.

the basic idea

The basic idea is of course to have the item in the queue represented in an array. Let's say that the array has a length of n and that the first element of the queue is at position 0 and the last element at position $k-1$ (k is the first free slot). Adding a new item to the queue can then be implemented simply by writing the new item at position k and increment k by one.

What should we do when we remove an item from the array? One solution would be to move all items from 1 to $k-1$ one step closer to the beginning. This would work but it is of course a very costly operation. A

better approach is to keep track of the first element in the queue using an index in the array. Let's say that the first element is at position i, removing and element is then simply done by returning the element at position i and increment i by one. So far so good, this is the general plan but this is where it starts to be complicated.

What should you do when k is equal to $n-1$? You can still add a new item at position $n-1$ but what should k be? If we increment k to n then we will add the next item at position n which of course is outside the array - we need to be careful now. We could allocate a new larger array and copy the items over to the new array, this would work and you would have a solution that looks similar to your stack implementation.

If you implement this and ship it as your contribution in a larger systems that in average does a thousand add and remove operations a second, how large would your array be after an hour? How large would it be after a year?

wrap around

There might be a better idea and that is to at least first make use of the available space in the array before extending it. The first item is at position i so the slots between 0 and $i-1$ are free. Could we set k to be 0 and carry on as before?

This is what we will do but now we of course need to be careful when we deallocate an item. We can deallocate an item when i is $n-1$ but then i should, as k, also be updated to 0. If you think about it both i and k are incremented by one modulo n.

All fine and dandy but we need to be careful - if i is or becomes equal to k :-0 If i is equal to k the queue is empty and we can not remove an item (this is the same as before). If k becomes equal to i after an enqueue operation, it means that the whole array is filled with valid items ... but it looks empty since i is equal to k. What do we do?

The solution is to allocate a new larger array (in the same way as you did with the stack implementation). You then copy all items from position i to $k-1$ (modulo n) to the new array starting at index 0. If everything works OK, you can discard the original array and replace it with the copy (remember to also update the index that points to the first item and the first free position).

In order to make this work you of course need to keep track of the size of the array, n, so this will be one additional property of the queue.

Note - when you remove an item from the queue you should also set the position to `null`. We do not want to have references to items that are no longer needed. If wee keep a reference this item will be copied in each invocation of the garbage collector.

In you report describe how you implemented the dynamic queue (not simply listing the code). Your report should be a like a tutorial for someone

who has not heard about this wrap-around solution.