# Queues in C

Ying Pei Lin

Fall 2024

## Implementing a queue by linked list

To implement a queue in C with linked list, we have to track the first cell and the next cell of each cell.

To initialize a queue, we allocate memory for the queue and set the first cell to `NULL`.

```
queue *create_queue() {
  queue *q = (queue*)malloc(sizeof(queue));
  q->first = NULL;
  return q;
}
```

To check if the queue is empty, we check if the first cell is `NULL`.

```
int empty(queue *q) {
  return q->first == NULL;
}
```

To add an element to the queue, we allocate memory for the new node, set the value, and find the last element in the queue. If the last element is not `NULL`, which means the queue is not empty, we set the next of the last element to the new node. Otherwise, the new node is the first element in the queue.

```
void enque(queue* q, int v) {
  // Init the new element
  node *nd = (node*)malloc(sizeof(node));
  nd->value = v;
  nd->next = NULL;

  // Find the last element in the queue
  node *prv = NULL;
  node *nxt = q->first;
```

```
  while (nxt != NULL) {
    prv = nxt;
    nxt = nxt->next;
  }

  // Check if the new one is the first element in the queue
  if (prv != NULL) {
    prv->next = nd;
  } else {
    q->first = nd;
  }
}
```

To remove an element from the queue, we free the first element and set the next element as the new first element. Note that if the queue is empty, the first element is `NULL` and the function will return 0, and therefore the queue can only store integers other than 0.

```
int dequeue(queue *q) {
  int res = 0;
  if (q->first != NULL) {
    node* temp = q->first;
    res = temp->value;
    q->first = q->first->next;
    free(temp);
  }
  return res;
}
```

## Improving the queue

In the previous implementation, we have to iterate through the queue to find the last element to add the new element next to it. This operation is $O(n)$, which is not efficient. By keeping track of the last element as we track the first element, we can add the new element in $O(1)$ time.

The optimized implementation is as follows:

```
void enque(queue* q, int v) {
  // Init the new element
  node *nd = (node*)malloc(sizeof(node));
  nd->value = v;
  nd->next = NULL;

  // Check if the new one is the first element in the queue
```

```
  if (q->last != NULL) {
    q->last->next = nd;
  } else {
    q->first = nd;
  }
  q->last = nd; // Update the last element
}
```

To compare the performance of the two methods, I tested the time to add and remove elements from the queue with sizes ranging from 10 to 100,000 elements.

First, we compare the time to add elements to the queue, as shown in Figure 1, we can see that the optimized implementation is much faster than the previous one. The time to add elements with the optimized implementation is almost constant, while the time to add elements with the previous implementation increases linearly with the number of elements.

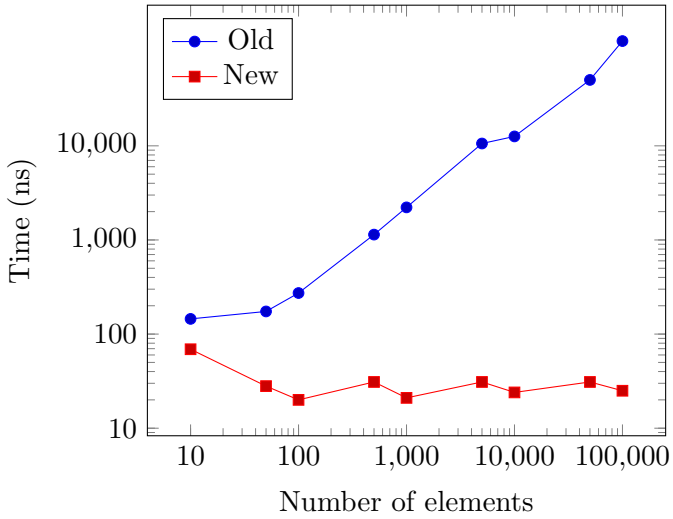


Figure 1: Time to add elements from the queue

Next, we compare the time to remove elements from the queue, as shown in Figure 2, both implementations have similar performance. The time to remove elements with both implementations is constant, which is expected since the time to remove elements from the queue should be $O(1)$.

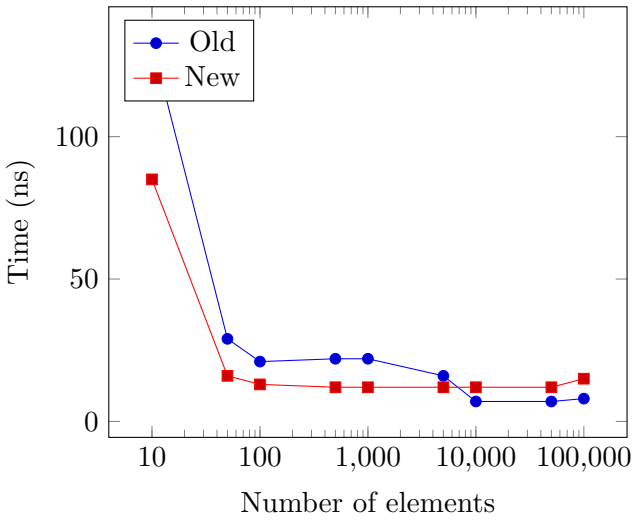


Figure 2: Time to remove elements from the queue