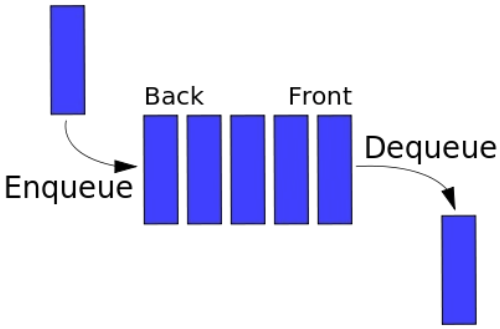
Queue
(Programming Club 6)

Mihai Enache – M.I.Enache@sms.ed.ac.uk, UG4, School of Informatics
5 November 2018

1. Introduction

We continue our series of data structures with the queue. A queue, similar to the previous concepts that you studied (arrays, linked lists and stacks), is a collection of elements. The operations that can be performed on it are removing an element from the front of the queue (dequeue) or adding an element to the back of the queue (enqueue). It really just emulates the behaviour of a queue of people when you go to the cinema, for example. Only the first person in the queue can enter the movie room. If you want to join, you first have to go to the back of the queue and wait for your turn.



(source: Wikipedia)

2. Applications

One of the common applications of a queue is finding a route (usually the shortest) between two cells in a matrix, or more generally, in a graph (we'll discuss graphs in a later session but for now let's stick to matrices, or 2D arrays).

Let's consider the matrix below:

1	0	0	1	1
1	1	0	0	0
0	1	1	1	1
1	1	0	0	1
1	1	1	1	1

Let's assume that the grid represents a building with rooms. Each of the cell is a room. You can move from one cell to an adjacent one (above, right, down or left, but not diagonally). Cells with 0 represent locked rooms through which you can't pass. Cells containing 1 represent unlocked room where you are allowed to pass. Your task is to determine the smallest number of rooms you have to traverse if you start in cell (1, 1) and have to reach cell (5, 5).

The solution to this task makes use of a queue. We keep a queue of all the rooms for which it is possible to reach and we also store the shortest possible distance in a separate matrix. Initially, the only room we can reach is the starting one (cell (1, 1)). While the destination is not found and while

we can still access new rooms, we keep adding to the queue. If the destination is found or no more rooms can be opened, it means that we have to stop our algorithm. The solution is given in pseudo-code below.

```
method findShortestPath(A, n, m)
    A – the grid representing the building
    n – the number of rows in the grid
    m – the number of colums in the grid

    initialize queue Q
    initialize matrix D[n][m] of distances such that D[i][j] is -1
representing that cell (i, j) hasn't been reached yet

    Q.enqueue((1, 1)) // add the starting cell to the queue
    D[1][1] = 0 // mark the number of cells needed to reach it

    while Q is not empty:
        x, y = Q.dequeue() // get the coordinates of the current cell
        for all (x_n, y_n) in neighbors of (x, y):
            if A[x_n][y_n] == 0: // check if the cell is locked
                 skip to next neighbor
            if D[x_n][y_n] == -1 or D[x][y] + 1 < D[x_n][y_n]:
                // if the cell is reached for the first time or if
                // we found a better path than before, we have to add
                // it to the queue as it might be part of an optimal
                // route to the target cell
                Q.enqueue((x_n, y_n))
                D[x_n][y_n] = D[x][y] + 1 // update its current dist

    return D[n][m] // return -1 if the cell can't be reached or
                   // the smallest number of cell through which we
                   // have to pass in order to reach it
```

- extension 1: think about how you can extend the above solution so that you also print the cells you have to traverse, not only the number of cells!
- extension 2: how about the case in which every room has an associated cost of passing through it? Your task is to adapt the above solution so that it minimizes the total cost you accumulate on your path from the start cell to the destination.

3. Resources

1. Queue in:
a) C++: https://en.cppreference.com/w/cpp/container/queue
b) Java: https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html
c) Python: https://docs.python.org/2/library/queue.html

2. Queue problems:
a) https://www.hackerrank.com/interview/interview-preparation-kit/stacks-queues/challenges
b) Implement a class Queue that supports the queue operation. Hint: linked list. How about if you know that the number of elements in the queue will never be above N? Can you think of something that would simplify the implementation that uses linked lists?