Algorithms and Data Structures: Week 3 (SOLUTIONS)

Johnson rev. Bell (Michaelmas 2022)

Question 1

Input: list L, node N, data v
Output: list L with data v inserted before N

```
node M
M.data = v
M.next = N
M.prev = N.prev
N.prev.next = M
N.prev = M
L.size = L.size + 1
```

Of course, the order is important here!

Question 2

The value c that is returned is the number of occurrences of the integer k in the array A.

Question 3

To update we work through the array considering the scores from lowest to highest. If a score is lower than the new one, then it is shifted to the right. This is continued until we find a score higher than the new one which we then write in place of the last score lower than it (which has already been shifted right). There are alternatives. Here is one way of writing the pseudocode (using 1-based array indexing).

Input: high score array H, entry E = [newscore, newname]
Output: updated H

```
if H[10] = NULL or newscore > H[10][1] then
    pos = 10                                  // record position we are looking at
    lower = True                              // is newscore lower than current position?
    while lower = True and pos > 1 do
        if H[pos - 1] = NULL or newscore > H[pos - 1][1] then
            H[pos] = H[pos - 1]                       // shift entry to the right
            pos = pos - 1                             // move focus to the next position
        else
            lower = False                             // higher value has been reached
        end if
    end while
    H[pos] = E
end if
```

Question 4

You need two pointers. To start, point one at the head and the other at the tail. Then move them in step along the list, forwards from the head and backwards from the tail. They will meet at the middle node.

```
Input: list L
Output: the middle node of L
  pointer1 = L.head
  pointer2 = L.tail
  while pointer1 ≠ pointer2 do
    pointer1 = pointer1.next
    pointer2 = pointer2.prev
  end while
  return pointer1
```

Question 5

One possible answer is an array A containing n entries each of which is itself an array containing n entries each of which is either 0 or 1. The value of $A[i][j]$ is 1 if players i and j have met and 0 otherwise (so initially all values are 0). So, for example, if there are 4 players and the only meetings so far were between 2 and 3, and 2 and 4, then we would have

$$A = [[0,0,0,0],[0,0,1,1],[0,1,0,0],[0,1,0,0]]$$

We also have an additional array B containing n entries. The value of $B[i]$ is the total number of meetings that i has made. In our example, $B = [0,2,1,1]$. Player i wins when $B[i] = n - 1$. (Of course, there is lots of redundancy here — $A[i][i]$ will always be zero (you can't meet yourself) and $A[i][j] = A[j][i]$ so you don't need both and we don't really need B at all because the information it contains can be found by looking at A. But this way we can easily keep track of things and avoid the cost of summing the matrix columns on every check for a win state.)

Here's how we update when i and j meet.

```
Input: arrays A and B, players i and j
Output: A and B updated after i and j meet
  if A[i][j] = 0 then
    A[i][j] = 1                          // this is their first meeting
    A[j][i] = 1
    B[i] = B[i] + 1
    B[j] = B[j] + 1
    if B[i] = n - 1  then
      print  i + ' wins'
    end if
    if  B[j] = n - 1 then
      print j + ' wins'
    end if
  end if
```