Project 5: Josephur’s Problem (II)

A story goes like this: After a fierce battle, some soldiers were caught by the enemy. The enemy decided to kill all of them except one, who would be sent back as a messenger.

The killing was about to begin: Start with the first person among those $n$ soldiers, who were numbered from 1 to $n$, sitting around a circle, every other person will be hanged except the last one, who would be the survivor. It was said that a smart soldier thought of a way to avoid being killed. The question is how did he do it?

The more general problem is to determine the survivor number $J(n, k)$, i.e., where to sit so you will always be the survivor, if you are one of $n$ people sitting around a circle, and every $k$th person will be killed, starting from the first one? Below are a few values for $J(n, 2)$.

<table>
<thead>
<tr>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J(n, 2)$</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

We dug it out at the very beginning of this course as Project 1.

1 What did we do?

When you solved this problem in Project 1, using an array $A[n]$, which initially contains the $n$ indices, 1, 2, ..., $n$, the gist of your solution to get $J(n, k)$ is essentially the following:

1. Location<-0;
2. //Kill n-1 of them
3. for(i<-1; i<n; i++)
4.    //Step over k of them
5.     for(j<-1; j<=k-1; j++)
6.        //Step over k-1 bodies. Note that the next of the last is the first
7.          location<-(location+1) % n;
8.        //Walking over all the dead ones
9.          while(A[location]==0)
10.            location<-(location+1) % n;
11.        //We have stepped over k-1 bodies, now kill the next by putting into
12.          //the cell with, e.g., 0
14.        //Who stays?
15.    for(i<-1; i<=n; i++)
16.        if(A[i]<>0)
17.          print A[i];
There are three layers of loops in this algorithm: The one in Line 3 runs $n - 1$ times, the one in Line 5 runs $k - 1$ times; and the while loop in Lines 9-10 run in $O(n)$ times. Thus, in total, it takes $O(kn^2)$.

Space wise, when more and more units in $A[n]$ get "killed", they are still there, which seems to be a waste.

If you used `ArrayList`, it appears to take just $O(n^2)$, but since it is just a sugar coated `array`, it is still $O(n^3)$.

2 How could we do better?

Just use what we need, and really remove what we no longer need. This calls for a dynamically allocated space, e.g., a “circular doubly linked list”, as shown in Figure 1.

![Circular Doubly Linked List](image)

Figure 1: An example of a circular doubly linked list

Since the members of such a group change over time, instead of using a statically allocated data structure such as an array, which stays there throughout the life time of the program, it makes much more sense to use a dynamically allocated linked structure as the data structure behind a solution for this Josephus’s problem, which comes and goes depending on the actual need for the moment.

In particular, in this project, you are expected to find a better solution, based on the circular doubly linked list as discussed in §10.2 of the textbook.

We use such a linked list to keep whatever we need at the moment: at the very beginning, the list is empty. We gradually insert one index at a time, until it contains $n$ nodes, when all the $n$ soldiers are added. This dynamic data structure would allow us to use the exact amount of space as we need at the moment.

Furthermore, we use a doubly linked list where each node contains an index and two links: one pointing to the previous one, and the other pointing to the next node (Check out the

---

1 Detailed analysis shows that Line 7 is done $(n - 1)(k - 1)$ times, Line 9 is done $(k - 1)(n - 1)$ times, and Line 10 is done at most $n - 1$ times since the number of “holes” increases from 0 to $n - 1$, and the algorithm steps over all these holes, when it runs for the final time. Since $k = O(n)$, this algorithm takes $\Theta(n^3)$ to run.

2 Also check out the operations and their Java implementation in Pages 20 through 29 in the lecture notes of Basic ones of the Data structures Part.
declaration in Section 3.1). Whenever we “kill” a person, we simply delete the associated node. Such a structure would lead to a $\Theta(1)$ deletion, reflecting the classic trade off between space and time.

Finally, to naturally materialize the idea of a circle, where “the next one of the last one is the very first one” (Cf. Have a look at Figure 1), we also have to add this circular feature, which overcomes the awkward modular operation as contained in Line 7 of the above algorithm.

This solution is better since

- It uses exactly the amount of space as needed at the moment, leading to a more efficient solution in terms of space;
- it will get rid of the do while loop as contained in the above algorithm, thus it takes $\Theta(kn)$ to complete; thus more efficient time wise; and
- it also gets rid of the for loop in Lines 15 through 17, since the index of the last remaining node is the answer we are looking for.

3 What should you do?

3.1 The circular doubly linked list structure

Below shows a doubly linked list as given on Pages 20 through 29 of the Data Structure lecture notes, and most of the operations are implemented there, as well.

```java
public class DoubleLinkedList{
    private DoubleNode head;

    public DoubleLinkedList(){
        head=null;
    }

    public DoubleNode search(int key);
    public void addFirst(int key);
    public void addLast(int key);
    public int delete(int key);
    private DoubleNode delete(DoubleNode node);
}
```

Assignment: Turn the above into the required circular doubly linked list, as shown in Figure 1.
3.2 What to do?

Come up with a solution for the $J(n,k)$ function based on this circular doubly linked list structure as described in the last section that will

1. determine where you should sit, if you are among 40 people for whom all but the last one will be hanged, and every other person will be hanged immediately. You also have to figure out the order in which the 39 poor guys will be picked up;

2. determine $J(n,2)$ for $n = 2$ up to $n = 100$; and

3. for $k = 3, 4, \text{ and } 12$, calculate $J(n,k)$ for $n = 10, 50, \text{ and } 100$.

3.3 How to do it?

You might go through the following steps to have a firm idea as how to proceed:

1. Read through the discussion about a doubly linked list on Pages 20 through 29 of the lecture notes on Data structure, and revise the doubly linked list structure to a circular doubly linked list, where the first node “points” to the last one, and the last one “points” back to the first one.

2. Revise the above methods as shown in Section 3.1 so that they would fit into a circular doubly linked list structure. Then come up with a method $J(n, k)$, as defined earlier.

3. If you are not sure whether your $J(n, k)$ method works, hand trace the $J(n, k)$ operation with the structure as shown in Figure 1, where $n = 3, k = 2$, to see how to delete two people, until there is one left, which should be 3, according to the sampler data as given earlier.

   Remember: If you are not sure if an algorithm works, your computer won’t, either. 😞

4. Come up with a driver, and verify the results you will have got for this implementation agree with those as shown in the sampler for Project 1.

4 What do you have to send in?

The source code of your program, together with a detailed report on your implementation of the circular doubly linked list structure, i.e., how do you revise the structure and the operations of the above doubly linked list structure so that it becomes a circular one; and the solutions to the assignment as given in Section 3.2.

$^3$As mentioned above, these results must agree with those as shown in the sampler to Project 1.
5 A general grading guideline

$\geq 1$: A syntactically correct and executable program is submitted by the deadline

$\leq 3$: A correct implementation of the circular doubly linked list structure, and its operations

$\leq 4$: Implement $J(n, k)$ based on the circular doubly linked list structure.

$\leq 5$: A lab report, including a discussion of your implementation of circular doubly linked list structure, as discussed in Section 3.1 and the correct output for the assignments as made in Section 3.2.