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

1 What did we do?

In Project 1, we solved this problem, with the help of an array \( A[n] \), which initially contains the \( n \) indices, 1, 2, \ldots, \( n \), by using essentially the following algorithm to solve \( J(n, k) \):

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
13. //Who stays?
14. for(i<-1; i<=n; i++)
15. if(A[i]<>0)
16. print A[i];
There are three layers of loops in this implementation. 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 \( O(kn^2) \). \(^1\)

Space wise, more and more units in \( A[n] \) get “killed”, but they are still there, which seems to be a waste.

2 What could we do?

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, 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.\(^2\)

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.

To facilitate the aforementioned deletion, 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 declaration in page 20.). Whenever we “kill” a person, we simply delete her node. To overcome the awkward modular operation as contained in Line 7 of the above algorithm, we also add this circular feature. A doubly linked list, but not a singly linked list, would lead to a \( \Theta(1) \) deletion, reflecting the classic trade off between space and time.

This solution is better since

- It uses exactly the amount of space as needed at the moment;
- it will get rid of the do while loop as contained in the above algorithm, thus it takes \( \Theta(kn) \) to complete; and
- it also gets rid of the for loop in Lines 13 through 15, since the index of the last remaining node is the answer we are looking for.

\(^1\)Detailed analysis shows that Line 7 is done \((n - 1)(k - 1)\) times, Line 9 is done at most \((k - 1)(n - 1) + \frac{(n-3)(n-2)}{2}\) times, and Line 10 is done at most \(\frac{(n-2)(n-1)}{2}\) times since the number of “holes” increases from 0 to \( n - 1 \), and the algorithm might have to step over all these holes. Since \( k = O(n) \), this algorithm takes \( \Theta(n^2) \) to run.

\(^2\)Also check out the operations and their Java implementation in Pages 20 through 28 in the lecture notes of Basic ones of the Data structures Part.
3  What should you 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 $k = 3, 4,$ and $12$, calculate $J(n, k)$ for $n = 10, 50, \text{and } 100$.

4  What do you have to send in?

The source code of your program, together with a report on the output.

5  A general grading guideline

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

$\leq 4$: Implement $J(n,k)$ based on the circular doubly linked list structure, as declared in Page 20 of the lecture notes of the Basic ones of the Data structures Part.

$\leq 5$: A lab report is sent in, containing correct output for the assignments as made in Section 3.

---

3You might review the sampler to Project 1 as a reference.