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 an . . .

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>

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, . . ., 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; j++)
6. //Step over some body...
7. do location<-(location+1) % n; //Next of the last is the first
8. //... and keep on stepping until we see someone who is still alive
9. while (A[location]==0);
10. //Kill him
11. A[location]<-0;
12. //Who stays?
13. for(i<-1; i<=n; i++)
14. if(A[i]<>0)
15. 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 $k$ times; and the do while loop in Lines 7-9 run in $O(n)$ times. Thus, in total, it takes $O(kn^2)$.

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, it makes much more sense to use a *dynamically allocated linked structure* as the data structure behind a solution.

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 \(^1\).

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, each 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 facilitate the aforementioned deletion, we use a *doubly linked list*. To overcome the awkward modular operation as contained in Line 7 of the above algorithm, we 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;
- 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.

3 What should you do?

Come up with a solution 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 \(n = 10, 50,\) and 100.

4 What do you have to send in?

The source code of your program, together with a report on the output. \(^2\)

---

\(^1\) 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.

\(^2\) You might review the sampler to Project 1 as a reference.