LABORATORY ASSIGNMENT 2
Problem 1

Draghici Bogdan
The Faculty of ACE, Craiova
First Year, Calculatoare romana
CR 1.2.A

May 19, 2019

Contents

1 Problem statement **2**

2 Pseudo-code description of the algorithms used **2**
2.1 Creating the circular list 2
2.2 The main algorithm 3

3 Algorithms' memory and computational complexity **3**
3.1 Best case scenario 4
3.2 Worst case scenario 4

4 Experimental data **4**
4.1 Data generation function 4
4.2 Data interpretation and graphics 5

5 Results and Conclusion **6**
5.1 Summary of achievements 6
5.2 Brief description of the most challenging achievement 6
5.3 Future directions for extending the lab homework 6

1 Problem statement

N children play the well-known ala-bala-portocala game. They are placed on a circle and start counting from 1 to k. The k-th child is removed from the game. The process continues with the rest of the children, each time counting from 1 up to k, until all the children are removed from the game.
Implement an algorithm for printing out the order in which the children are removed from the game using a circular list.

Input: Read the number N of chidren and the number k, then the n values, one for each child from the game.

Output: Write in order the children's values that are removed.

2 Pseudo-code description of the algorithms used

2.1 Creating the circular list

In order to create the circular list, I used a structure for doubly-linked lists and a repetitive structure with n steps in which to read the values of elements and add them to the circular list using the function:
Function dl-push-element-end(head, tail, new value)

1. info of new element = new value
2. last element = previous of tail
3. next of last element = new element
4. next of new element = tail
5. previous of new element = last element
6. previous of tail = new element

On line 1, the info of new element takes the value read in the repetitive structure, then find the *last element* of the list and make if point to the *new element* added (lines 2 and 3). After this, the *new element* will point to the previous *last element* of the list and the tail of the list (lines 4 and 5). And on the last line the tail will point to the new element.

2.2 The main algorithm

The main algorithm used to solve the problem is contained in the next function:
Function ala-bala-portocala(head,tail,k)

```
1.  element = next of head
2.  while info of head != 0, do
3.      for i = 1, k - 1, execute
4.          element = next of element
5.      popped element = element
6.      write info of element
7.      next of previous element = next of element
8.      previous of next element = previous of element
9.      element = next of element
10.     remove popped element
11.     info of head = info of head - 1
```

On line 1 of the function, the variable *element* takes the position of the first element from the circular list. The repetitive structure from line 2 will repeat till the circular list is empty, in *info of head* being stored the number of elements from the list. Next we count k children using another repetitive structure, knowing that we already counted the first one. After this, the child that must be removed is the current position of the element, and we memorize it in the variable popped element. We write the value of the child that must be eliminated (line 6), we put the previous child next to the next child (lines 7 and 8) and replacing the current one with the next child(line 9). Then we eliminate from the *popped element*from the cycle and decrement the number of the remaining children(lines 10 and 11).

3 Algorithms' memory and computational complexity

In order to remove a child, the algorithm makes k steps: *k-1* in the repetitive structure from line 3, and another one line 9. For eliminating all the n children, it has to make n*k steps. So the algorithm has n*k computational complexity.
The algorithm is using n + 4 memory locations of the size of a doubly-linked structure and 2 memory locations for int variables. In the end, the algorithm is using n memory locations.

3.1 Best case scenario

In the best case scenario, the number k equal with 1. In this instance,the function does the following:

1. 1 elemental time for the first attribution of *element*
2. 2 for each comparison of *info of head* with 0
3. 2*n, n for each attribution of iterator and verification of the condition
4. 4*n for the rest of attributions
5. n for writing the n children
6. n for freeing the n spaces of memory alocated for the children
7. 2*n for the decrementations of *head*

The function ala-bala-portocala is making 1 + 2 + 2*n + 4*n + n + n + 2*n = 10*n + 3 elemental times.

3.2 Worst case scenario

There does not exist an exact value the worst case scenario, because in the algorithm are no if structures and the value of k can be bigger than the number n. An estimative value of the worst case scenario is:
1+1+n*((2*(k-1)+1+4+1+1+2))=2+n*(2*k+7) elemental times.

4 Experimental data

4.1 Data generation function

For the generation of the input data sets I used the rand() function from the library math.h, and also an function implemented by myself, that generates a random number from a given range.
Function random(min, max)

1. counter = max - min, number = 0
2. **while** counter != 0 **do**
3. number = number * RAND_MAX + rand()
4. counter = counter RAND_MAX

5. return number % (max - min + 1) + min

RAND_MAX is a constant which represents the biggest value that can be generated using the function rand(), and it's value is 32767.
On line 2 I initialized counter with the length of the range and the variable in which the number will be created with 0. On line 3 is a repetitive structure that will be executed for log(RAND_MAX) counter times, in order to obtain a number in the range. On line 4, the number is multiplied by RAND_MAX and also added a new random value generated by the rand() function. On line 5 the counter is divided by the maximum value that can be taken by a number generated in rand().
So, when the number is multiplied by RAND_MAX, the counter is divided by RAND_MAX, in order to get to the given range as quickly as possible. After the repetitive structure ends, the number created might exceed the range length, so we must do the rest of the division with the range length+1, then add the first value of the range in order to obtain a number between min and max.

4.2 Data interpretation and graphics

Because the algorithm has O(n*k) complexity, I used for the abscise axis the produce between n and k.

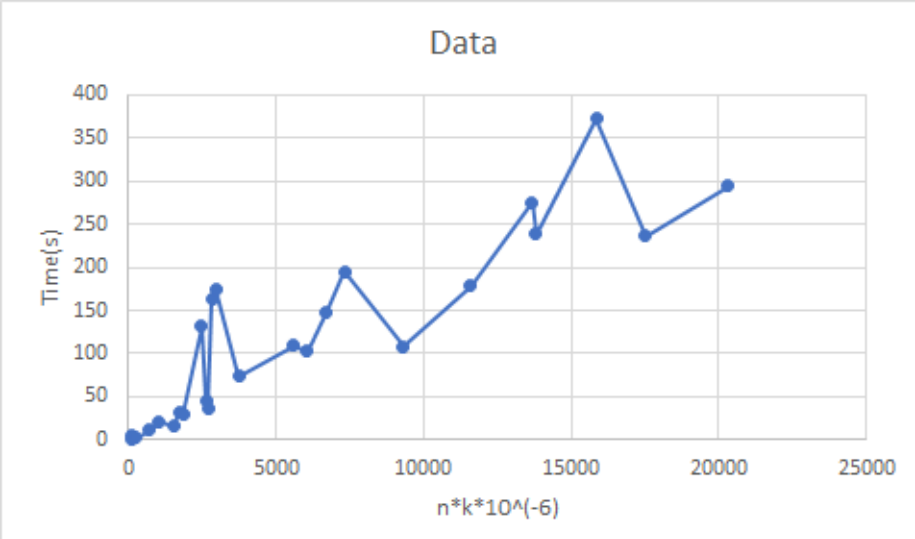

Figure 1: Data

As it can be seen, the time is increasing way faster than the data input.

5 Results and Conclusion

5.1 Summary of achievements

One of my achievements was implementing a way to find the k-th element in a circular list and continuing from that point counting another k elements, without creating a function that does so. I also found out that any operation with lists can be implemented directly in the algorithm, without the use of any function, but at the cost of a larger and harder to understand code.

5.2 Brief description of the most challenging achievement

The most challenging achievement was to find a way to remove an element from the list without the need of another function, implementing it in a simple way directly inside the problem's algorithm.

5.3 Future directions for extending the lab homework

I want to find and optimal way, with no need for too many cases and variables, in which to implement an algorithm that is solving the problem using simple linked lists.