Banker's Algorithm Complexity Analysis

Shoham Chakraborty

1004351 | CI03

From a quick analysis we can see the following time complexities:

Function Name	Time Complexity
`Banker(…)`	O(1)
`setMaximumDemand(…)`	O(numberOfCustomers * numberOfResources)
`requestResources(…)`	O(numberOfResources)
`releaseResources(…)`	O(numberOfResources)

We can see that the time complexities of functions are smaller compared to the complexity of the `checkSafe()` function, and hence can be ignored.

In `checkSafe()` we have two loops – the outer `while()` loop and the inner loop.

For the inner loop we have three components – `for` loop and the `if` statement, and updating the values of `work`. The resulting complexity from these statements is `O(numberOfCustomers * numberOfResources)`. The analysis in pseudocode is as follows:

```
for I = 1 to N do  // O(numberOfCustomers)
    if ((not FINISH[i]) and
        NEEDi <= WORK) then { // O(numberOfResources)
        WORK = WORK + ALLOCATION_i; // O(numberOfResources)
        FINISH[i] = true;
        NOCHANGE = false;
    }

Complexity
= O(numberOfCustomers * (numberOfResources + numberOfResources))
= O(numberOfCustomers * numberOfResources)
```

This inner loop is wrapped in an outer `REPEAT` loop (implemented as a `while` loop). In the worst possible case, where the last entry satisfies – leading to every process needing to be evaluated. In this case, the loop will take `O(numberOfCustomers)` time.

Hence the resulting time complexity is:
`Complexity = O(numberOfCustomers * numberOfCustomers * numberOfResources)`

Resource Used: https://cis.temple.edu/~ingargio/old/cis307f95/readings/deadlock.html