Tutorial 07 — Analyzing Runtimes for Algorithms

mardi, juin 7

`betterFibLoop()` algorithm

```
int betterFibLoop (int n) {
// Assertion: A nonnegative integer n has been given as input.
    if (n == 0) {
        return 0;
    } else {
        int oldest = 0;
        int middle = 1;
        int i = 1;
        while (i < n) {
            int youngest = oldest + middle;
            oldest = middle;
            middle = youngest;
            i = i + 1;
        }
        return middle;
    }
}
```

1

Prove that `n-i` is a *bound function* for the `while` loop in this algorithm.

Proof

An assertion A is a **loop invariant** for this `while` loop if it satisfies all of the following, whenever the algorithm is executed with the problem's precondition satisfied:

- an execution of the loop test `t` has no side-effects — that is, it does not change the value of any inputs, variables, or global data,
- A is satisfied whenever the loop is reached, during an execution of the algorithm starting with the problem's precondition being satisfied, and
- if A is satisfied at the beginning of any execution of the loop body `S` (when the problem's precondition was satisfied when execution of the algorithm started) then A is satisfied, once again, when this execution of the loop body ends

By examination of the code above, we can see the following:

a) `this` is a doubly linked list, as specified in the precondition, and this global data is not modified in any way when the `while` loop is reached, so this proves the first part of the loop invariant

b) Since the `if` statement on line 2 has to fail in order to reach the `while` loop on line 13, we know that the value of `index` is equal to or in between 0 and `this.length - 1`. This establishes the second part of the loop invariant

c) `i` is initialized to 0 on line 5 of the algorithm above. We also know that `index` is greater than 0 due to the test on line 2 failing. Therefore, the third part of the proposed loop invariant is also true.

d) `currentNode` is initialized to the `head` (the beginning) of the linked list, before the `while` loop is reached. The value of `i` is initialized to 0 before the `while` loop executes. This value is incremented inside the body of the loop, along with `currentNode` being set to the next node in the list. This establishes the last part of the loop invariant which states that `currentNode` is the node at index `i` in the list

Thus, we can conclude that `n-i` is a bound function for this `while` loop.

2

Use this to provide an upper bound for the number of steps executed by this algorithm on a non-negative integer `n`. The "uniform cost criterion" is being used here.

We can trace the execution for this algorithm in order to analyze its runtime, using the uniform cost criterion.

Suppose this algorithm is executed, with the problem's precondition satisfied.

If `n=0` then the total number of steps executed is 2, consisting of lines 3 and 4.

Suppose instead `n >= 1`. Then:

- Line 3 costs 1 step, as the `if` statement is checked and does not pass
- Lines 6, 7 and 8 each cost 1 step each, for a total of 3 steps
- We reach the `while` loop on line 9

    - The **body of the loop** (lines 10-13) gets executed *at most* `n` times, while the **loop test** itself (line 9) gets executed *at most* `n+1` times
        - Each execution of the loop test requires 1 step and each execution of the loop body requires 4 steps. Thus, the total number of steps required *at most* for the `while` loop is:

$$(\texttt{n+1} \cdot 1) + (\texttt{n} \cdot 4) = 5\,\texttt{n} + 1$$

- One last step is executed after the `while` loop, which is the `return` statement on line 15

Therefore, the upper bound for the number of steps executed by this algorithm is given by:

$$(5\,\texttt{n} + 6)$$

`fibPair()` algorithm

```
int[] fibPair (int n) {
// Assertion: A nonnegative integer n has been given as input
    int[] F = new integer[2];
    if (n == 0) {
        F[0] = 0;
        F[1] = 1;
    } else {
        int[] oldF = fibPair(n - 1);
        F[0] = oldF[1];
        F[1] = oldF[0] + oldF[1];
    }
    return F
}
```

3

Write a *recurrence* for the number $T(n)$ of numbered steps executed by this algorithm on a non-negative integer input `n` (once again, the “uniform cost criterion” is being used here to estimate a running time).

We can trace the execution for this algorithm in order to analyze its runtime, using the uniform cost criterion.

Suppose this algorithm is executed, with the problem’s precondition satisfied.

If `n=0` then the total number of steps executed is 5, consisting of lines 3, 4, 5, 6 and 12. Therefore, if `n=0`, then the total cost is 5.

Suppose instead that `n >= 1`. Then:

- Lines 3, 4, 8, 9, 10 and 11 are executed, for a total of 6 steps
  - Line 8 contains a **recursive** call to `fibPair()`, with input `n-1`
  - Thus, we can create a recurrence $T(n)$ to show the number of steps executed by the algorithm:

$$T(n) = \begin{cases} 5 & \text{if } n = 0 \\ T(n-1) + 6 & \text{if } n >= 1 \end{cases}$$

4

Guess a simple *solution* for this recurrence.

We can reach a solution for this recurrence through the process of “unrolling” the recurrence and looking for a repeating pattern at each step.

$$\begin{aligned} T(4) = T(3) + 6 &= 29 \\ T(3) = T(2) + 6 &= 23 \\ T(2) = T(1) + 6 &= 17 \\ T(1) = T(0) + 6 &= 11 \\ T(0) = 5 & \qquad \text{by definition} \end{aligned}$$

From the pattern above, we can see that the closed form of this recurrence would be $T(n) = 6n + 5$.

5

Use your recurrence to prove that your solution is correct.

We can prove that this closed form is correct by using the *standard form of mathematical induction on* n.

BASIS STEP

Using our recurrence when $n = 0$: $T(0) = 5$ by definition

Using our closed form solution when $n = 0$: $6(0) + 5 = 5$

And $5 = 5$, $\therefore$ base case is established.

INDUCTIVE STEP

Let $k \geq 0$. Then it is necessary and sufficient to use the following:

Inductive hypothesis

Assume $T(n)$ to be true for $n = k$. Therefore, *by assumption*, $T(k) = 6k + 5$.

Inductive claim

$T(k+1) = 6(k+1) + 5$

Proof

We will begin with our recurrence as it is established, and try $n = k + 1$.

$T(k+1) = T((k+1) - 1) + 6$

$T(k+1) = T(k) + 6$ We have defined $T(k)$ under our inductive hypothesis:

$T(k+1) = (6k + 5) + 6$

$T(k+1) = (6k + 6) + 5$

$T(k+1) = 6(k+1) + 5$ As required.