Computer Science 331

Getting from Pseudocode to a Bound on Running Time

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #6

Outline

Objective and Strategy

Objective: use code (or pseudocode) to estimate the *worst-case running time* of a program (or algorithm).

Useful Values:

- Worst-case running time (exact)
- Upper and lower bounds on worst-case running time

Strategy: consider subprograms . . .

- beginning with individual statements . . .
- then considering progressively larger subprograms . . .
- until the whole program has been considered.

Case: Program is a Single Statement

Example: `x := 1`

Amount to charge:

- 1 unit

Use this charge for:

- a single arithmetic or Boolean operation
- a comparison
- an assignment of a value to a variable

Another Example

Example: `x := y := 1`

Amount to charge:

- 2 units (one per assignment)

Comments:

- be careful with compound statements
- one line does not always equal one unit!

Case: Program is a Sequence of Subprograms

Structure to Consider: $S_1; S_2$

Worst-Case Running Time: If

- worst-case running time of S_1 is T_1, and
- worst-case running time of S_2 is T_2,

then

- worst-case running time of entire program is *at most:* $T_1 + T_2$

Explanation:

- S_1 and S_2 are executed sequentially, so runtime is the sum of each of their runtimes
- T_1 and T_2 are upper bounds on the runtimes of S_1 and S_2, so $T_1 + T_2$ is an upper bound on the total runtime

Case: Program is a Conditional Statement

Structure to Consider:

if P **then**
  S_1
else
  S_2
end if

Worst-Case Running Time: if

- worst-case running time of S_1 is T_1, and
- worst-case running time of S_2 is T_2,

then

- worst-case running time of program is: $\max(T_1, T_2) +$ cost of evaluating P

Case: Program is a Loop

Structure to Consider:

while P **do**
  S
end while

We need to know:

- the worst-case cost to evaluate P
- the worst-case cost to execute S
- the maximum number of executions of the loop body

Problem:

- it is not even clear that this will halt!

First Objective: Counting Executions of the Loop Body

Recall that a *Loop Variant* is an integer-valued function f of variables such that

- the value of f decreases by at least 1 each time loop body is executed;
- the test P is **false** if the value of f is ≤ 0

The *existence* of a loop variant implies that the loop terminates if each evaluation of P and each execution of the loop body terminates.

Useful fact:

Executions $\leq f$ evaluated at the initial values of its variables

Next Objective: Bounding Total Running Time

Suppose:

- Loop body is executed at most k times
- Worst-case cost for each evaluation of the loop test P is $\leq T_1$
- Worst-case cost for each execution of the loop body S is $\leq T_2$

Then:

- *Total* cost for *all* executions of test P is at most: $(k+1)T_1$
- *Total* cost for *all* executions of loop body is at most: kT_2
- Therefore, the *total* cost to execute the loop is at most: $(k+1)T_1 + kT_2$

If cost of jth iteration of S is $T_2(j)$: $\quad (k+1)T_1 + \sum_{j=0}^{k} T_2(j)$

Example

Suppose A is an integer array with length n, *key* is an integer, and the following code is executed.

```
i := 0
while ((i < n) and (A[i] <> key)) do
  i := i + 1
end while
```

Loop Variant for this program's loop: $f(n,i) = n - i$

- i increases after each iteration, so $f(n,i)$ decreases
- $f(n,i) \leq 0$ if $i \geq n$ and the loop terminates if $i \geq n$

What about 2nd condition in test? ignore (doesn't affect worst case)

Example, Continued

Maximum number of executions of the loop body:

- $f(n,0) = n - 0 = n$

Worst-case cost to evaluate test:

- 3 units (two comparisons, one Boolean operation)

Worst-case cost for an execution of the loop body:

- 2 units (one addition, one assignment)

Upper bound on worst-case cost to execute the loop:

- $3(n+1) + 2n = 5n + 3$

Case: Program is a Nested Loop

Structure to Consider:

while P_1 **do**
  while P_2 **do**
    S
  end while
end while

Method:

- compute worst-case cost of inner loop as above
- compute cost of outer loop using computed inner loop cost as the worst-case cost of the outer loop's body

An example will be covered in next week's labs.

Case: Program Calls Itself a Constant Number of Times

Example: Fibonacci Number Program

public int **fib**(int n)
if $n == 0$ **then**
  return 0
else if $n == 1$ **then**
  return 1
else
  return $\text{fib}(n-1) + \text{fib}(n-2)$
end if

Objective: Writing an Expression for the Running Time

Let $T(n)$ be the number of steps used on input n. Then

$$T(n) \leq \begin{cases} 2 & \text{if } n = 0, \\ 3 & \text{if } n = 1, \\ 6 + T(n-1) + T(n-2) & \text{if } n \geq 2. \end{cases}$$

This is an example of a *recurrence relation*:

- $T(n)$ expressed using the same function T evaluated at **smaller** inputs
- Explicit (non-recursive) values of T given for small inputs n (base cases)

$T(2) \leq 6 + T(1) + T(0) = 11$, $T(3) \leq 6 + T(2) + T(1) = 20$, etc...

Analysis of Recursive Programs

Students who have already completed MATH 271 should be able to solve these problems *now*. Students in MATH 271 should be able to do so after studying *mathematical induction*.

Exercises:

1. Use the above information to prove that
$$T(n) \leq 6 \times 2^n - 4$$
for every integer $n \geq 0$.
2. Use the above information to prove that
$$T(n) \leq 6 \times \text{fib}(n+1) - 4$$
for every integer $n \geq 0$.

Wait a Minute . . .

Question:
How do you establish *correctness* of a simple recursive program?

Exercise (For Between Now and Next Class):
Try to think of a “proof rule” that could be used to establish partial correctness of a function like the one given above.

Additional References

Proving That a Given Bound on Worst-Case Running Time is Correct:

- See the discussion of *mathematical induction* in your MATH 271 textbook!

Finding a Bound on Worst-Case Running Time:

- Cormen, Leiserson, Rivest and Stein
  Introduction to Algorithms, Second Edition
  - Appendix A (Summations): For analysis of *loops*
  - Chapter 4 (Recurrences): For analysis of *recursive procedures*