CS 341: Algorithms

Charles Shen

Fall 2016, University of Waterloo

Notes written from Jeffrey Shallit's lectures.

Feel free to email feedback to me at ccshen902@gmail.com.

Contents

1 Selection in deterministic linear time

← October 25, 2016

2 Lower bounds

3 Adversary arguments in general

More generally, we can think of a lower bound proof as a game between the algorithm and an "adversary".
The algorithm is "asking questions" (for example, comparing two elements of an array), while the adversary is answering them (providing the results of the comparison).

The adversary should be thought of as a very powerful, clever being that is answering in such a way as to make your algorithm run as slowly as possible. The adversary cannot "read the algorithm's mind", but it can be prepared for anything the algorithm might do.
Finally, the adversary is not allowed to "cheat"; that is, the adversary cannot answer questions inconsistently. The adversary does not not need to have a particular input in mind at all times when it answers the questions, but when the algorithm is completed, there must be at least one input that matches the answers the adversary gave (otherwise it would have cheated).

The algorithm is trying to run as quickly as possible.
The adversary is trying (through its cleverness) to *force* the algorithm to run slowly.
This interaction proves that the lower-bound obtained by this type of argument applies to *any possible* algorithm from the class under consideration.

Theorem 3.0.1. *Every comparison-based algorithm for determining the minimum of a set of n elements must use at least $\frac{n}{2}$ comparisons.*

Proof. Every element must participate in at least one comparison; if not, the not compared element can be chosen (by an adversary) to be the minimum.
Each comparison compares 2 elements.
Hence, at least $\frac{n}{2}$ comparisons must be made. □

Theorem 3.0.2. *Every comparison-based algorithm for determining the minimum of a set of n elements must use at least $n-1$ comparisons.*

Proof. TODO: Look at Lecture 13 for proof □

4 Finding both the max and min of a list of n numbers

It is possible to compute both the maximum and minimum of a list of n numbers, using $\frac{3}{2}n - 2$ comparisons if n is even, and $\frac{3}{2}n - \frac{3}{2}$ comparisons if n is odd.
It can be proven that *no comparison-based method* can correctly determine both the max and min using fewer comparisons in the worst case.

We do this by constructing an adversary argument.

In order for the algorithm to correctly decide that x is the minimum and y is the maximum, it must know that

1. every element other than x has won at least one comparison (i.e. $x > y$ if x wins a comparison with y),

2. every element other than y has lost at least one comparison

Calling a win W and a loss L, the algorithm must assign $n-1$ W's and $n-1$ L's. That is, the algorithm must determine $2n-2$ "units of information" to always give the correct answer.

We now construct an adversary strategy that will force the algorithm to learn its $2n-2$ "units of information" as slowly as possible.
The adversary labels each element of the input as N, W, L, or WL. These labels may change over time.
N signifies that the element has never been compared to any other by the algorithm.
W signifies the element has won at least one comparison.
L signifies the element has lost at least one comparison.
WL signifies the element has won at least one and lost at least one comparison.

← October 27, 2016

So the the adversary uses the following table The adversary also tentatively assigns values to the elements, which may change over time.
However, they can only change in a fashion *consistent* with previous answers. That is, an element labelled L may only *decrease* in value (since it lost all previous comparisons, if it is decreased, it will still lose all of them), and an element labelled W may only increase in value.
An element labelled WL cannot change.

labels when comparing elements (x, y)	the adversary's response	the new label	unit of information given to the alg.
(N, N)	$x > y$	(W, L)	2
(W, N) or (WL, N)	$x > y$	(W, L) or (WL, L)	1
(L, N)	$x < y$	(L, W)	1
(W, W)	$x > y$	(W, WL)	1
(L, L)	$x > y$	(WL, L)	1
(W, L) or (WL, L) or (W, WL)	$x > y$	no change	0
(WL, WL)	consistent with assigned values	no change	0

For example, consider the following sequence of possible questions asked by the algorithm and the adversary's responses:

```
-------------------------------------------------------------
Algorithm  |   Adversary's responses and worksheet
compares   |   x1      x2      x3      x4      x5      x6
-------------------------------------------------------------
 (x1, x2)  |  W-20    L-10       N       N       N       N
 (x4, x5)  |  W-20    L-10       N     W-30    L-15       N
 (x1, x4)  |  W-40    L-10       N    WL-30    L-15       N     (*)
 (x3, x6)  |  W-40    L-10    W-11    WL-30    L-15    L-2
 (x2, x5)  |  W-40   WL-10    W-11    WL-30     L-7    L-2
 (x3, x1)  | WL-40   WL-10    W-50    WL-30     L-7    L-2
 (x2, x4)  | WL-40   WL-10    W-50    WL-30     L-7    L-2     (**)
 (x5, x6)  | WL-40   WL-10    W-50    WL-30    WL-7    L-2
-------------------------------------------------------------
```

At this point the algorithm knows that x_3 is the maximum, since it is the only element that has never lost a comparison, and x_6 is the minimum, since it is the only element that has never won a comparison.
Note that in step (*), the adversary was forced to reassign the value he had previously assigned to x_1, since x_1 had to win the comparison against $x_4 = 30$. This is permitted, since x_1 had won every previous comparison and so increasing its value ensures consistency with previous answers.
Also note that step (**) is superfluous, as the algorithm didn't learn any new information.

Theorem 4.0.1. *Every* comparison-based *method for determining both the maximum and minimum of a set of n numbers must use at least $\frac{3}{2}n - 2$ comparisons (if n even), or $\frac{3}{2}n - \frac{3}{2}$ comparisons (if n odd), in the worst case.*

Proof. Suppose that n is even.
As shown before, the alg. must learn $2n - 2$ units of information.
The most it can learn in one comparison is 2 units; when both elements have never participated before in a comparison, labelled by (N, N). This can happen at most $\frac{n}{2}$ times.
To learn the remaining $n - 2$ units of info., the alg. must ask $n - 2$ questions.
The total number of questions is therefore at least $\frac{n}{2} + n - 2 = \frac{3}{2}n - 2$.
The same kind of argument works for n odd. □