Lower bound method: Adversarial Method

Lecturer: Xiaodi Wu Scribe: Xiaodi Wu

• Reading: This lecture note.

1 Motivation

We introduce the decision tree method in the last lecture. Here, we demonstrate first that the decision tree method is not all powerful, which calls for new lower bounding methods. To that end, we introduce another generic lower bounding method, called the adversarial method.

From the last lecture, the decision tree method leads to an $\Omega(\log(n))$ lower bound for binary search, (or any search), even for unsorted sequence. One would expect a much better lower bound, e.g., $\Omega(n)$, for the search problem in an unsorted sequence. However, the decision tree method fails to derive such a lower bound.

2 Adversarial Method

The adversarial method is, basically, to try to come up with a worst case input for the algorithm and find out the complexity of the algorithm on that case. Of course, one can find a worst case input without going through the adversarial method. However, the adversarial method provides a systematical way of searching for a worst case.

Assume again that we focus only on the deterministic algorithm. For any such algorithm $A$, imagine an adversary interacts with $A$ and plays the role as providing input information to $A$. ($A$ does not see the input directly, but only sees the input through the adversary). The adversary on the other side does not determine the input ahead of time, but rather generates the input on the fly as long as the input is consistent and valid (e.g., the information from the adversary about the input should be the same if $A$ makes the same query, and so on).

The purpose of the adversary is to come up with the choice of the input such that $A$ needs to spend as long as possible time in order to answer correctly on all possible inputs. At the end of the day, one can construct a worst case input using the above methodology and analyze the complexity of $A$ on that input, which is a lower bound of the complexity of $A$.

3 Examples

Let us see how one could apply the adversarial method to concrete problems.

3.1 Lower bound for the search problem

Assume there is a sequence $B$ of (not necessarily sorted) $n$ items, and one is asked to search for a key $k$ among the $n$ items, or return no such key. For any algorithm $A$ that accesses at most $n-1$ out of $n$ items in the input, construct the following adversary:

- Let $i_1$ be the first item accessed by $A$. The adversary chooses $B[i_1] = 1$. The algorithm $A$ will (adaptively) choose $i_2$, the next item to access, depending on $i_1$ and $B[i_1]$.
- The adversary sets $B[i_2] = 2$. The algorithm $A$ chooses similarly again $i_3, i_4, \ldots, i_{n-1}$. After each such choice, we set $B[i_j] = j$ (note that $B[i_j]$ will also affect the future choices of $i_j', j' > j$).

1Contents will appear in bonus homework but not in exams.
Assume that there is an algorithm $A$ that accesses at most $n - 1$ items in the input. Through the above procedure, we produce an input (with $n - 1$ out $n$ items determined) that in $A$’s eye, the $n - 1$ items in the input are $1, 2, \cdots, n - 1$. Let the key to search for be $n$.

Because $A$ accesses at most $n - 1$ items, so $A$ needs to make a decision and output the result. Because the key $n$ is not among $1, 2, \cdots, n - 1$, $A$ can only output that the key exists and is stored in the position that hasn’t been accessed (let the position be $i_n$), or output that the key does not exist.

In the former case, we will set $B[i_n] = 1$ (or anything other than $n$). In the latter case, we will set $B[i_n] = n$ (any correct algorithm should output $i_n$). In both cases, $A$’s output is incorrect, which contradicts the correctness of $A$ on all inputs. Thus, we know there is no such an algorithm $A$ that accesses at most $n - 1$ items and any $A$ needs to access $n$ items.

### 3.2 Lower bound for the min (max) problem

Assume there is a sequence $B$ of $n$ items, and one is asked to find the min (max) of $B$. We claim one can derive a lower bound $n$ using the adversarial method. We can even recycle part of the argument in the above case.

- Let $i_1$ be the first item accessed by $A$. The adversary chooses $B[i_1] = 1$. The algorithm $A$ will (adaptively) choose $i_2$, the next item to access, depending on $i_1$ and $B[i_1]$.

- The adversary sets $B[i_2] = 2$. The algorithm $A$ chooses similarly again $i_3, i_4, \cdots, i_{n-1}$. After each such choice, we set $B[i_j] = j$ (note that $B[i_j]$ will also affect the future choices of $i_{j'}, j' > j$).

Assume that there is an algorithm $A$ that accesses at most $n - 1$ items in the input. Through the above procedure, we produce an input (with $n - 1$ out $n$ items determined) that in $A$’s eye, the $n - 1$ items in the input are $1, 2, \cdots, n - 1$.

Because $A$ accesses at most $n - 1$ items, so $A$ needs to make a decision and output the result. $A$ only sees the value $1, 2, \cdots, n - 1$. It makes sense for $A$ to choose the minimum out of these $n - 1$ items (that is 1) or output a random guess (that is out of $[1, n - 1]$). If $A$ outputs 1, we will set $B[i_n] = 0$. If $A$ outputs a random guess, we will set $B[i_n] = 1$. Thus, in both cases, $A$’s output is incorrect, which contradicts the correctness of $A$ on all inputs.