6.11 Three Problems Exhibiting Interesting Proof Techniques[3]

In this section, we present three unrelated problems involving sequences and multisets. Each algorithm is an example of a different proof technique. The first algorithm utilizes the principle of strengthening the induction hypothesis. We strengthen the induction hypothesis four times during the development of this solution, leading to an efficient algorithm. The second algorithm is an example of an obvious technique — improving the "theorem" by eliminating all unnecessary assumptions. The example shows that this principle is not always straightforward. This third example shows how to improve an algorithm by choosing the base of the induction wisely.

6.11.1 Longest Increasing Subsequence

Let S be a sequence of distinct integers $x_1, x_2, ..., x_n$. An **increasing subsequence (IS)** of S is a subsequence $x_{i_1}, x_{i_2}, ..., x_{i_k}$, with $i_1 < i_2 < \cdots < i_k$, such that, for all $1 \le j < k$, we have $x_{i_j} < x_{i_{j+1}}$. A **longest increasing subsequence (LIS)** of S is an increasing subsequence of maximum length.

> **The Problem** Find a longest increasing subsequence of a given sequence of distinct integers.

The algorithm we develop in this section is an excellent example of the principle of strengthening of the induction hypothesis. We will strengthen the hypothesis several times, each time as a result of problems encountered in the previous attempt. Consider first the straightforward induction.

> **Induction hypothesis (first attempt):** *Given a sequence of size < m, we know how to find a longest increasing subsequence of it.*

The base case consists of sequences of size 1 for which the problem is trivial. Given a sequence of size m, we find an LIS of its first $m-1$ elements, and consider x_m. If x_m is greater than the last element in the LIS, given by the induction, then x_m can be appended to the LIS, creating a new longer LIS, and we are done. Otherwise, however, it is not clear how to proceed. For example, there may be several different LISs and x_m may extend one of them, but not necessarily the one found by the induction. The next step seems to be a strengthening of the induction hypothesis as follows:

> **Induction hypothesis (second attempt):** *Given a sequence of size < m, we know how to find all the longest increasing subsequences of it.*

[3] This section can be skipped at first reading.

From "Introduction to Algorithms: A Creative Approach" by Udi Manber Addison-Wesley 1989

The base case is still trivial. We use induction in the same way, except that now we can check x_m against *all* of the LISs and find whether a longer IS exists. This attempt solves the previous problem, but it introduces another problem — we now have to find *all* LISs. If x_m cannot extend any LIS, then there may still be an IS of length 1 less than the longest, and x_m can extend it, which will create a new LIS. It seems that we have gotten ourselves into a hole, because we now have to find all ISs of largest and second largest length. But to find all the second largest ISs, we will need to find all the third largest ISs, then all fourth largest, and so on. This is a good example where strengthening the induction hypothesis is overdone.

Let's look back at the stronger induction hypothesis. Do we really need *all* LISs? We need only to know whether x_m can extend one of them. Can we somehow find the "best" one in terms of potential of extension? The answer is positive. The best LIS is the one that ends with the smallest number! If we can extend any LIS, we can surely extend this one. (There may be several different LISs that end with the same number, and they are all equivalent in terms of extension potential. For simplicity, we talk about "the best one" instead of "an arbitrary best one.") Let's try another induction hypothesis, this one a little weaker than the last one:

> **Induction hypothesis (third attempt):** *Given a sequence of size < m, we know how to find a longest increasing subsequence of it, such that no other longest increasing subsequence of it has a smaller last number.*

The base case is still trivial. Given x_m, we can determine whether it can be appended to the LIS found by the induction. Assume that the LIS is of length s. If x_m can be added, then we have a new LIS, which is longer than the previous one; thus, this new LIS is unique, so it is definitely the "best" one, and we are done. Otherwise, we know that no longer increasing subsequence exists. But we are still not done. It may be the case that x_m cannot be added to the best LIS (since it is smaller than the last number in that LIS), but it can be added to an IS of length $s-1$, making the latter an LIS with a smaller last number. To account for this possibility, we need to know the best IS of length $s-1$. But then again, if the induction hypothesis states that we know the best IS of length $s-1$, then x_m may extend an IS of length $s-2$ making it the new best IS of length $s-1$. We will have to be able to determine whether x_m extends such an IS in order to proceed with the induction. So, we will need to know the best IS of length $s-2$, $s-3$, and so on down to the best IS of length 1, which is simply the smallest number in the sequence so far. (Even without using induction, one can see that shorter ISs cannot be discarded arbitrarily — there is always the possibility that one of these ISs is the start of the final LIS.)

Yet again we try to strengthen the induction hypothesis. We denote by BIS(k) the best increasing subsequence of length k — namely, the one that ends with the smallest number (if there is more than one such subsequence we take an arbitrary one). We denote by BIS(k).*last* the last number in the sequence BIS(k).

> **Induction hypothesis (fourth attempt):** *Given a sequence of size < m, we know how to find BIS(k) for all $k < m-1$, if they exist.*

The base case remains trivial. Given x_m, we have to find which of the BISs it can change. x_m extends a certain BIS(k) if and only if the following two conditions occur: (1) $x_m >$ BIS(k).*last*, so x_m can be added to BIS(k), and (2) $x_m <$ BIS($k+1$).*last*, so BIS(k) with x_m at the end is better than BIS($k+1$). We claim that BIS(1).*last* < BIS(2).*last* < $\cdots$ < BIS(s).*last*, where s is the size of the LIS. This claim is true because, if BIS(j).*last* $\leq$ BIS($j-1$).*last* for some j, then the first $j-1$ numbers of BIS(j) would be better than BIS($j-1$). The algorithm proceeds as follows. Given x_m, we look at the values of BIS(i).*last*, for $i=s, s-1, s-2$, and so on, until we find one, say BIS(j).*last*, which is smaller than x_m. If no such j exists, then x_m is the smallest number in the sequence so far, and it becomes BIS(1). If $j=s$, then we extend BIS(s) with x_m, creating a new BIS($s+1$). (The previous BIS(s) remains unchanged.) Otherwise, we have BIS(j).*last* $< x_m <$ BIS($j+1$).*last*. We then replace BIS($j+1$) with BIS(j)x_m.

This is basically the whole algorithm, and it is quite simple once we use the right induction. Notice that the search can be performed by binary search, because we are searching a sorted set. Hence, each x_m adds at most $O(\log m)$ comparisons, and the total running time is $O(n \log n)$. We leave it to the reader to complete the details of this algorithm, which is not a straightforward task.

6.11.2 Finding the Two Largest Elements in a Set

A common technique, which is important in proving almost any theorem, is to search the proof thoroughly for assumptions or steps that are not essential. Removing such assumptions results in a better theorem. Having inessential assumptions is also sometimes an indication that the proof may be wrong. Quoting Polya and Szego [1927]: "One should scrutinize each proof to see if one has in fact made use of all the assumptions; one should try to get the same consequence from fewer assumptions . . . and one should not be satisfied until counterexamples show that one has arrived at the boundaries of the possibilities." The same is true for algorithms. This principle sounds simple, but many times it is not, as seen in the next example.

The Problem Given a set S of n numbers $x_1, x_2, \ldots, x_n$, find the first and second largest of them.

We are looking for an algorithm that minimizes only the number of comparisons of elements from the set. We ignore other operations. Furthermore, for simplicity, we assume that n is a power of 2.

We try the usual divide-and-conquer technique, by dividing the set S of size n into two subsets P and Q of size $n/2$. If we use straightforward induction, we assume that we know the first and second largest elements of P and Q, denote them by p_1, p_2, and q_1, q_2 respectively, and we try to find the first and second largest elements of S. It is easy to see that two more comparisons are necessary and sufficient to find the first and second largest elements of S. One comparison is between the two maximals p_1 and q_1, and the other