CMSC 351 - Introduction to Algorithms
Spring 2012
Lecture 4

Instructor: MohammadTaghi Hajiaghayi
Scribe: Rajesh Chitnis

1 Design of Algorithms using Induction

So far we have seen the use of induction in the proof of theorems and showing correctness of algorithms. In a sense, the induction idea gives us **recursive** algorithms.

1.1 Finding one-to-one mappings

Let f be a function that maps a finite set $A = \{1, 2, \ldots, n\}$ to itself. Assume f is represented by an array $f[12\ldots n]$ such that $f[i]$ holds the value of $f(i)$ which is an integer between 1 and n. We call f a **one-to-one** function if for every element j, there is at most one element i that is mapped to j. Consider an example of a function in Figure 1.

The Problem: Given a finite set A and a mapping $f : A \to A$ find a subset $S \subseteq A$ with maximum number of elements such that

1. The function f maps every element of S to some other element within S itself.
2. No two elements of S are mapped to the same element, i.e., f is one-to-one when restricted to S.

Algorithm: If f is originally one-to-one then taking $S = A$ we are done. If on the other hand $f(i) = f(j)$ for some $i \neq j$ then S cannot contain both i and j. We can try all subsets $S \subseteq A$, but the running time is $2^n n$ which is exponential. We want a more efficient algorithm. For example, in Figure 1 we have $f(2) = 1 = f(3)$. So S cannot contain both 2 and 3. The choice between 2 and 3 is important: if we eliminate 3, then 1 is eliminated and then 2 is eliminated. However if we instead eliminate 2 only then we obtain $\{1, 3\}$ as a good set.

So reducing the problem to a smaller one is important and non-trivial. We can use induction hypothesis: We know how to solve the problem for sets with

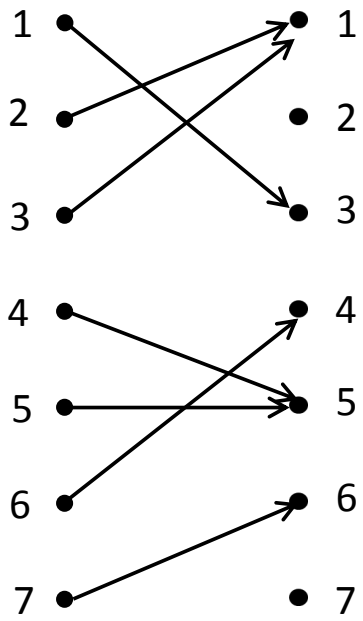

Figure 1:

$n-1$ elements. The base case is trivial: if there is only one element in the set then it must be mapped to itself. But the induction hypothesis is non-trivial when we have a choice (e.g. 2 and 3 in Figure 1). The key idea is that the element i that has no element mapped to it cannot belong to S and thus we must remove it. This is because $i \in S$ and $|S| = k$ implies that the k elements of S must be mapped to at most $k-1$ elements of S and hence f cannot be one-to-one when restricted to S. So we remove $\{i\}$ from A and iterate. Note that the reduced problem is exactly the same (except the size) as the original problem. We have to be careful though: the only condition we had previously was that f maps A to itself. This condition is still maintained for the set $A \setminus \{i\}$. By this algorithm we need to consider only n elements instead of 2^n choices for S. Thus this algorithm is much more efficient.

See Figure 5.3 in the book [1] for the algorithm described above.

2 The Celebrity Problem

The Celebrity problem is a nice example in algorithm design whose solution does not require scanning all the data or even a significant part of it.

Definition 1 *A **celebrity** among n persons is someone who is known by everyone but does not know anyone.*

The problem is to identify the celebrity, if one exists, by asking questions only of the form "Excuse me, do you know the person over there?". We assume all answers are correct and even the celebrity answers. The goal is to minimize the number of questions. Since there are $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs we need at most $n(n-1)$ questions.

The Problem Given a $n \times n$ matrix A whose (i, j) entry is 1 if person i knows person j and 0 otherwise, determine whether exists an i such $A(j, i) = 1$ for all $j \neq i$ and $A(i, k) = 0$ for all $k \neq i$.

Again the idea is to use induction: the base case with two people is easy. Say we know the solution for $n-1$ people: can we then use it to solve the problem for n people. Note that since there is at most one celebrity we have the following three cases:

1. The celebrity is among the first $n-1$ people.
2. The celebrity is person n.
3. There is no celebrity.

Case 1 is easy: just check if person n knows the celebrity and the celebrity does not know person n. The other two cases are harder. To determine if person n is the celebrity we need to ask $2(n-1)$ questions and then we can have the $n(n-1)$ questions that we tried to avoid. The trick is to solve the problem **backward**. It may be hard to identify a celebrity, but it is easier to identify someone as a non-celebrity. If we eliminate someone, then the size of the problem is reduced from n to $n-1$. The algorithm is as follows:

1. We ask Alice whether she knows Bob.
2. If Alice says NO, then Bob is not a celebrity and we can eliminate him.
3. If Alice says YES, then Alice is not a celebrity and we can eliminate her.

Note that in either case we eliminate one person. We can find, by induction, the celebrity among the remaining $n-1$ people. If there is no celebrity, then the algorithm terminates; otherwise we check that all the eliminated guys know the candidate celebrity and that the candidate celebrity does not know any of the eliminated guys. We have at most $3(n-1)$ questions: $n-1$ questions in the first phase to eliminate $n-1$ people, and then at most $2(n-1)$ questions to verify that the candidate is indeed a celebrity. This is much better than the $n(n-1)$ questions needed in the naive worst case scenario.

Note that here our reduced instance of $n-1$ people is **smart** and not arbitrary. See the code of the algorithm in the book [1].

3 Evaluating Polynomials

The Problem: Given a sequence of real numbers $a_n, a_{n-1}, \ldots, a_1, a_0$ and a real number x compute the value of the polynomial $p_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$.

Induction can lead to a very good solution. The first natural attempt is to reduce the problem by removing a_n and thus we are left with evaluating the polynomial $p_{n-1}(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$. Induction hypothesis is "We know how to evaluate the polynomial $p_{n-1}(x)$ given $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ and x". Now $p_n(x) = p_{n-1}(x) + a_n x^n$, i.e. we need to compute x^n, multiply it by a_n and add the result to $p_{n-1}(x)$. Here we need $\frac{n(n+1)}{2}$ multiplications and n additions. Note here that the power of x is computed from scratch.

Let us use a stronger induction hypothesis: "We know how to compute $p_{n-1}(x)$ and x^{n-1} given $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ and x" Then we need one multiplication for x^n, one for $a_n x^n$ and one addition at the end for a total of $2n$ multiplications and n additions. What if we remove a_0 instead of a_n? Define $p'_{n-1}(x) = a_n x^{n-1} + a_{n-1}x^{n-2} + \ldots + a_1$. The induction hypothesis (reversed order) since we removed a_0 instead of a_n is "We know how to evaluate $p'_{n-1}(x)$ given the coefficients $a_n, a_{n-1}, \ldots, a_2, a_1$". Then $p_n(x) = xp'_{n-1}(x) + a_0$ which needs one multiplication and one addition. This gives a total of n additions and n multiplications. This is called as **Horner's** rule for evaluating polynomials.

Algorithm 1 HORNER'S RULE

Input: A real number x and coefficients $a_0, a_1, \ldots, a_{n-1}, a_n$.
Output: A real number $p = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1}x^{n-1} + a_n x^n$.

```
1: p = a_n
2: for i = 1 to n do
3:     p = xp + a_{n-i}
4: end for
```

4 Finding the maximum consecutive subsequence

The Problem: Given a sequence $x_1, x_2, \ldots, x_n$ of real numbers (not necessarily positive), find a subsequence $x_i, x_{i+1}, \ldots, x_j$ (of consecutive elements) such that the sum of the numbers in it is maximum over all subsequences of consecutive elements. If all numbers are negative, then the maximum consecutive subsequence is empty with sum 0.

Example: Consider the sequence $2, -3, 1.5, -1, 3, -2, -3, 3$. The maximum consecutive subsequence is $1.5, -1, 3$ with sum 3.5.

The straightforward induction hypothesis is "We know how to find the maximum subsequence in a sequence of size $< n$". For $n = 1$ the max subsequence consists of a single number if it is non-negative or is empty otherwise. Consider $S = \{x_1, x_2, \ldots, x_n\}$ of size $n > 1$. We know the max subsequence in

$S' = \{x_1, x_2, \ldots, x_{n-1}\}$. If max subsequence is empty in S', then every number in S' is negative and we only need to consider x_n. So suppose the max subsequence in S' is $S'_M = \{x_i, x_{i+1}, \ldots, x_j\}$ for some $1 \leq i \leq j \leq n-1$. If $j = n-1$ (namely the max subsequence is a **suffix**), we only need to check x_n for an extension. However if $j < n-1$, then there are two possibilities: either S'_M remains maximum or there is another subsequence, which is not maximum in S', but is maximum in S when x_n is added to it.

The idea is to strengthen the induction as knowing just the max subsequence is not enough. So we have a stronger induction hypothesis "We know how to find, in any sequence of size $< n$, a maximum subsequence overall, and the max subsequence that is a suffix". Now that we know both, it is much easier. We add x_n to the max suffix. If this sum is more than the global max subsequence, then we update it (as well as the new suffix). Otherwise we retain the previous max subsequence. Note that we are not done yet. We also need to find the new max suffix: just blindly adding x_n is not good. If this sum is negative, we take the empty set as max suffix. So for some constant c we can find the max consecutive subsequence with cn operations instead of the naive way to try all i, j which will take $\binom{n}{2}$ operations.

See the code of the algorithm in the book [1].

5 Strengthening the Induction Hypothesis

Instead of $P(< n) \Rightarrow P(n)$, we can add another assumption Q which makes the proof easier [P and Q]$(< n) \Rightarrow$ P(n). However this is the common mistake. We should prove [P and Q]$(< n) \Rightarrow$ [P and Q](n). It is crucial to follow the induction hypothesis precisely.

See common error in Section 5.11 in the book [1].

References

[1] Udi Manber, *Introduction to Algorithms - A Creative Approach*