**CS 344: Design and Analysis of Computer Algorithms** **Rutgers: Spring 2022**

# Lecture 3

January 24, 2022

*Instructor: Sepehr Assadi*

## 1 Community Detection Problem

Let us consider the following problem.

**Problem 1** (Community Detection)**.** There are $n$ people in a party for some *odd* integer $n$. We know that (strict) majority of these people belong to a hidden community. Our goal is to find all people that belong to this hidden community. In order to do this, we can ask two people to *greet* each other: if both people belong to the party, they will say the know each other; otherwise, they say that they do not each other.

Design an algorithm for finding all the people in the hidden community using minimal number of greetings[1].

### First Step: Developing Intuition About the Problem

The first step in designing an algorithm for a problem is to develop *intuition* about the problem – this is a necessary step that typically is done *implicitly* and is quite crucial for the success of the algorithm design. At the same time, it is also *never sufficient* for obtaining the final answer; in other words, intuitions are different from final solutions.

There are many ways to develop intuition about a problem. One easy way is to turn the problem into a small "puzzle" on a reasonably small input. For instance, let us see how can we solve this problem when we have $n = 15$ people and 8 of them belong to the hidden community.

- We can ask all pairs of people to greet each other. Then, for any pairs of people that say they know each other, we mark both of them as belonging to the community. This way every person in the hidden community will be marked and no person outside the community can be marked also. Moreover, the number of greetings is the total number of ways of choosing two persons among 15 which is $\binom{15}{2} = \frac{15\cdot 14}{2} = 105$.
- The above solution however seems to be inefficient for different reasons:
  1. If we already have that $A$ knows $B$ and that $B$ knows $C$, there is no reason for us to ask $A$ and $C$ greet also: they surely both belong to the hidden community and know each other.
  2. The above solution will work as long as the size of the hidden community is at least two and not the majority specified in the problem. This means that most likely, we are not using "all our knowledge" about the problem; in other words, one hopes that by using the fact that majority of people belong to the hidden community, we can do much better.
- A simple observation (motivated partly by part (1) above) is that if we can find just a single person in the community, we can find everyone else as well: we just ask this person to greet all of them one by one using 14 greetings.

[1]I.e., our measure of efficiency here is the number of greetings as opposed to the runtime of the algorithm.

- The key observation (motivated by part (2) above) is that if we majority of people belong to the hidden community, finding one among them “should not be hard”: we can pair up people together into groups of size 2 with one extra person; we will then ask the groups to greet each other. If even in one group, the participants know each other we are already done as we found two (and not only one) member of the hidden community. If no group know each other, then we know that each group contains at least one non-member of the community and thus there are 7 (= number of groups) non-members among the people that greeted each other. This necessarily means that the remaining person that was not part of any group should belong to the hidden community and thus we are done again.
- Notice that this new solution now only requires $14 + 7 = 21$ greetings.

Let us now turn this intuition into a proper algorithm for this problem.

## Second Step: Designing an Algorithm

When designing an algorithm for the problem, we can no longer make simplifying assumptions and should work with any valid choice of number of people, i.e., $n$. However—at least in this particular case—our intuition was already pretty close to a complete algorithm and so not much needs to be done. Formally, the algorithm is as follows:

1. **Part one: finding a single member** Pair the participants into groups of size two with one extra person with no group. Ask the pairs in each group to greet each other. If we found a group that know each other, let $P$ be any arbitrary participant of this group. Otherwise let $P$ be the extra person with no group. (In this algorithm, $P$ is supposed to denote a member of the hidden community[2].)
2. **Part two: finding all members** Ask $P$ to greet everyone else. Mark whoever knows $P$ plus $P$ itself as the members of the hidden community.

The above is a complete algorithm for this problem – we clearly specified the steps needed and each step is simple enough for the purpose of this problem. We are *not* done however and the main task is actually ahead of us: we need to *analyze* the algorithm and *rigorously prove* that it indeed solve the problem for us.

## Third Step: Proof of Correctness

Let us now prove that the algorithm correctly identifies the members of the hidden community. Firstly, we claim that at the end of the first part, $P$ indeed belongs to the hidden community. In the case $P$ is a member of a group that knows the other participant this is clear as only members of the hidden community know another person in the party. In the other case, notice that participants of every group that do not know each other necessarily contain a non-member of the community. Since in this case no group know each other, we have that the number of non-members is at least equal to members inside the groups in total. As such, for the majority of people to belong to the hidden community, we should necessarily have that the extra person is a member—thus picking $P$ to be this person ensures that $P$ is indeed a member.

Finally, we can prove the second part as follows. Since $P$ is a member of the hidden community, $P$ will know every other member and no one else. Thus, the set of people that $P$ knows plus $P$ is exactly the set of members of the hidden community. This concludes the proof of correctness of the algorithm.

## Fourth Step: Efficiency Analysis

The final step is then to analyze how efficient is our algorithm. In this course, this almost always corresponds to a *runtime analysis* of the algorithm as we focus on time as the main measure of efficiency—however, in this particular example, the efficiency is actually measured by the number of greetings.

[2]Note that here it is technically not correct to say that “$P$ is a member of the hidden community” at this point as we have not proved anything yet. In other words, the goal of the algorithm is to pick $P$ to be a member, but we still cannot be sure that this is indeed the case.

The number of greetings in the first part of the algorithm is $\lfloor \frac{n}{2} \rfloor = O(n)$ as there are this many groups exactly. In the second part also, the number of greetings is $n - 1 = O(n)$ as the number of people minus $P$ is this many. This means that the algorithm in total requires $O(n)$ greetings[3].

## 2 Mathematical Induction

We now introduce[4] the main tool that is used for proving correctness of algorithm: *mathematical induction.* Let us do a quick reminder of proof by induction in the context of proving mathematical equations. We prove the following two for all integers $n$:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}; \tag{1}$$

$$\sum_{i=1}^{n} i^2 = \frac{n \cdot (n+1) \cdot (2n+1)}{6}. \tag{2}$$

**Proof of Eq** (1)**.**

- *Induction base:* For $n = 1$, $\sum_{i=1}^{1} i = 1$ and $\frac{1 \cdot (1+1)}{2} = 1$ so the hypothesis is true.
- *Induction step:* Suppose the hypothesis is true for all $n \leq k$ and we prove it for $n = k + 1$. We have,

$$\begin{aligned}
\sum_{i=1}^{k+1} i &= \sum_{i=1}^{k} i + (k+1) && \text{(we simply took out the last term in the sum to add it explicitly)} \\
&= \frac{k \cdot (k+1)}{2} + (k+1) && \text{(since the induction hypothesis is true for all } n \leq k\text{)} \\
&= (k+1) \cdot (\frac{k}{2} + 1) && \text{(basic arithmetic: by factoring out } (k+1) \text{ from the two terms above)} \\
&= \frac{(k+1) \cdot (k+2)}{2} && \text{(basic arithmetic: by writing the second term differently)} \\
&= \frac{n \cdot (n+1)}{2}. && \text{(since } n = k+1\text{)}
\end{aligned}$$

So we proved the induction step, which finalizes the proof. In other words, we prove that for *all* $n$:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}.$$

Note that we do *not* need to do anything else – the rest of the proof is taken care of by the "magic" of the induction itself!

**Proof of Eq** (2)**.**

- *Induction base:* For $n = 1$, $\sum_{i=1}^{1} i^2 = 1$ and $\frac{1 \cdot (1+1) \cdot (2 \cdot 1+1)}{6} = 1$ so the hypothesis is true.

[3] If you think of the first inefficient solution we got in the puzzle above as a general algorithm, you see that it requires $\Theta(n^2)$ greetings. So that solution is in fact asymptotically worse than the more efficient algorithm we eventually obtained.

[4] This might not be the correct word as you have surely seen induction before in your courses.

- *Induction step:* Suppose the hypothesis is true for all $n \leq k$ and we prove it for $n = k+1$. We have,

$$\begin{aligned}
\sum_{i=1}^{k+1} i^2 &= \sum_{i=1}^{k} i^2 + (k+1)^2 && \text{(we simply took out the last term in the sum to add it explicitly)} \\
&= \frac{k \cdot (k+1) \cdot (2k+1)}{6} + (k+1)^2 && \text{(since the induction hypothesis is true for all } n \leq k\text{)} \\
&= (k+1) \cdot (\frac{k \cdot (2k+1)}{6} + (k+1)) \\
& && \text{(basic arithmetic: by factoring out } (k+1) \text{ from the two terms above)} \\
&= (k+1) \cdot (\frac{2k^2 + 7k + 6}{6}) && \text{(basic arithmetic: by simplifying the terms)} \\
&= (k+1) \cdot (\frac{(k+2) \cdot (2k+3)}{6}) \\
& \text{(basic arithmetic: } 2k^2 + 7k + 6 = (k+2) \cdot (2k+3)\text{; multiply the right hand side to verify)} \\
&= \frac{n \cdot (n+1) \cdot (2n+1)}{6}. && \text{(since } n = k+1\text{)}
\end{aligned}$$

So we proved the induction step, which finalizes the proof.

## 3 Proof of Correctness via Induction

Proving correctness of algorithms typically involves proving statements about *all* inputs for every possible length. As such, induction is usually the best way for establishing such proofs. We will see numerous examples of this throughout the course and in this lecture already. So for now, let us start with a basic illustration.

Consider the problem of finding the maximum entry of an array $A[1:n]$. We can design the following algorithm for this problem:

1. Iterate over elements of $A$ in order $A[1], A[2], \ldots, A[n]$.
2. Let $max \leftarrow A[1]$ initially and in each iteration $i$, if $A[i] > max$, then set $max \leftarrow A[i]$.
3. At the end, return $max$.

*Proof of Correctness.* We prove by induction that for every $i \in \{1, \ldots, n\}$, the value of $max$ at the end of iteration $i$ is equal to $\max\{A[1], \ldots, A[i]\}$. This implies that for $i = n$ at the end of the algorithm, $max$ is equal to the largest element of $A$ as desired.

**Induction base.** The base case of induction for $i = 1$ is true by definition as $max = A[1]$ and maximum of an array of length one is the only entry itself.

**Induction step.** Suppose the induction hypothesis is true until iteration $i = j$ and we prove it for $i = j+1$. By induction hypothesis, $max = \max\{A[1], \ldots, A[j]\}$ at the beginning of iteration $j+1$. After this iteration, we have $max \leftarrow \max\{max, A[j+1]\} = \max\{A[1], \ldots, A[j+1]\}$. This implies the induction step.

We can conclude the proof of the statement by induction. Now, by setting $i = n$ in the statement above, we obtain that $max$ is equal to the largest entry of $A$, thus validating the correctness of the algorithm.

*Runtime Analysis.* The algorithm iterates over all the $n$ elements of $A$ once, and each iteration takes $O(1)$ time for comparing the numbers and updating the variables, thus the total runtime is $O(n)$.