CIS 4930 – Problem Solving **Spring 2026**

Scribe 6: Mathematical Induction and Algorithm Design

Instructor: Viet Tung Hoang

Material here is based on the book "Introduction to Algorithms, A creative approach" by Udi Manber.

There is a strong similarity between mathematical induction and recursive algorithms. For example, consider the Inventor's paradox in the divide-and-conquer paradigm; this phrase was actually coined by Polya in the context of teaching mathematical induction. If you struggle with designing recursive algorithms, it may indicate a weak background in mathematical induction. In this note, we will revisit some induction proofs, and then study how to use the mindset of induction for algorithm design.

6.1 Mathematical Induction

6.1.1 Gray Code

THE PROBLEM. Suppose that you have $N = 2^k$ objects; for convenience let's refer to them as $1, \ldots, N$. You want to encode them using k bits per name. The easiest way is to encode object i with the binary representation of the number $i - 1$. However, when we go from i to $i + 1$, we may have to change several bits. For example, consider the case $k = 3$. When you go from 3 to 4, you switch from 011 to 100, meaning 3 bit flips. This kind of big change is undesirable in digital design. Our goal is to find a k-bit encoding s_i for number i so that when we go $s_i \to s_{i+1}$ (including the wrap-around $s_N \to s_1$), there is only a single bit flip. For example, for $k = 2$, we can have $s_1 = 00$, $s_2 = 10$, $s_3 = 11$, and $s_4 = 10$. This kind of encoding is known as Gray code.

STUDENT-TEACHER DIALOG. Below is an imagined dialog between a student and a teacher in solving the problem above.

1. *Teacher*: Can you visualize the problem above, using, say graphs?
2. *Student*: So usually, in graphs, one represents objects by nodes, and their relationship by edges. Here there are N objects, so they should be nodes. It seems to me that for two nodes, we only care if their encoding differ only in a single bit, so we'll connect them if this happens. So basically I'm looking for a cycle $s_1 \to s_2 \to \cdots s_N \to s_1$.
3. *Teacher*: Good, we'll return to this visualization later. We'll try to solve this problem by an induction proof. The base case $k = 1$ is trivial: $s_1 = 0$ and $s_2 = 1$. Let's assume that we know how to develop Gray code for $N = 2^k$ objects. What's your plan to extend it for $2N = 2^{k+1}$ objects?
4. *Student*: Since the number of objects doubles, I think it's natural to divide $2N$ objects to two groups, say $\{1, \ldots, N\}$, and $\{N+1, \ldots, 2N\}$. We then can encode each group separately, using the k-bit Gray code. But I need a $(k + 1)$-bit here, so I'm not sure how to use the induction hypothesis.
5. *Teacher*: If you try to juggle two groups at once, that's not easy. What if you have only one group, say $\{1, \ldots, N\}$ and need to add one extra bit to the encoding $s_1, \ldots, s_N$?
6. *Student*: That's easy. I can simply prepend the bit 0 to the encoding, so s_i becomes $0\|s_i$, where $\|$ denotes string concatenation.

7. *Teacher*: Let's say you can encode objects $1, \ldots, N$ with $s_1, \ldots, s_N$, and encode objects $N+1, \ldots, 2N$ with $s_{N+1}, \ldots s_{2N}$. Now to extend the first encoding, you add 0 to each string $s_1, \ldots, s_N$. Now how you would add a bit to the remaining strings $s_{N+1}, \ldots, s_{2N}$, so that totally you'd have a proper encoding of $2N$ objects? (This doesn't have to be a Gray code.)

8. *Student*: Clearly the first group exhausts all $(k+1)$-bit strings that start with 0. So the second group can only start with 1. So I'll prepend 1 to them, so s_i becomes $1\|s_i$.

9. *Teacher*: Visualize what you have now, and what you want.

10. *Student*: We have Figure 6.1: there are two cycles, but we want $0\|s_N$ and $1\|s_{N+1}$ are connected, and so are $0\|s_1$ and $1\|s_{2N}$.

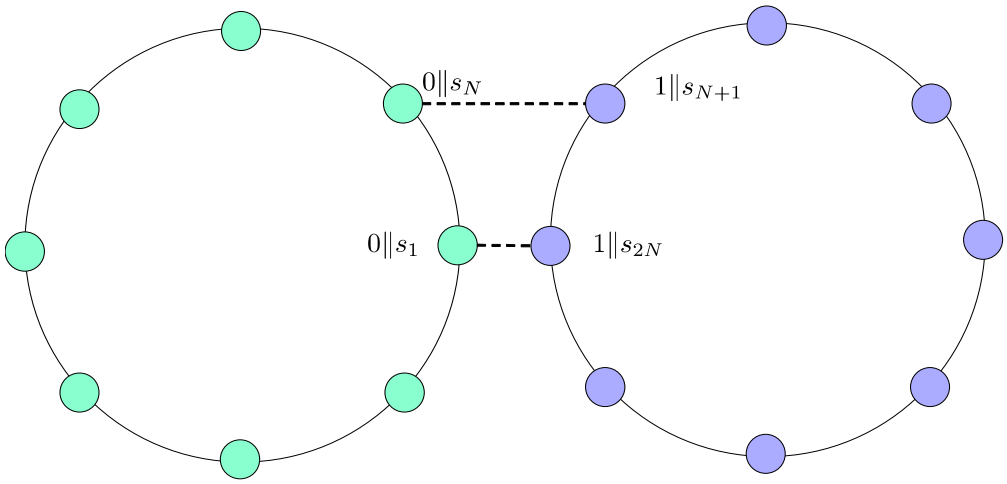

Figure 6.1: Visualization of the inductive construction of Gray code.

11. *Teacher*: Two nodes are connected if they differ only in a bit. If $0\|s_N$ and $1\|s_{N+1}$ are connected, what can you tell about s_N and s_{N+1}?

12. *Student*: They must be the same! Otherwise when we go from $0\|s_N \to 1\|s_{N+1}$, there must be at least two bit flips: one for the first position $0 \to 1$, and at least another from $s_N \to s_{N+1}$. Likewise, we must have $s_1 = s_{2N}$.

13. *Teacher*: So if I give you $s_1, \ldots, s_N$, how would you construct $s_{N+1}, \ldots, s_{2N}$?

14. *Student*: We should have $s_{N+1} = s_N$, $s_{N+2} = s_{N-1}$, and so on, until $s_{2N} = s_1$.

Lesson. What do we see here?

- First, visualization is very powerful. Initially, it's not clear why we need a visualization. After all, it's just a cycle, which is so simple. But eventually, during the inductive construction, things become more complex, and visualization starts to play a role.

- Next, when we go from N to $2N$, it's natural to think of two groups of size N to exploit the induction hypothesis.

- The induction hypothesis gives us an encoding $s_1, \ldots, s_N$ for objects $1, \ldots, N$, but how would we get $s_{N+1}, \ldots, s_{2N}$? You may be tempted to jump to $s_{N+i} = s_i$ for every $i \leq N$, however, it means we have little room left to wiggle. Instead, it's better to keep generic strings $s_{N+1}, \ldots, s_{2N}$, and then later identify the relationship between this group and $s_1, \ldots, s_N$.
- Then, don't attempt to extend two groups simultaneously, because it can be overwhelming. Instead, we break that into several steps: (i) prepend 0 to the first group, (ii) prepend 1 to the second group, and (iii) identify the constraints via the visualization.

6.1.2 Finding Edge-Disjoint Paths in a Graph

THE PROBLEM. Let $G = (V, E)$ be a connected undirected graph. Two paths in G are *edge-disjoint* if they do not contain the same edge. Let O be the set of nodes in V with odd degrees. Note that O has an even number of elements. (Why?) Prove that we can divide the nodes in O into pairs and find edge-disjoint paths connecting nodes in each pair.

STUDENT-TEACHER DIALOG. Below is an imagined dialog between a student and a teacher in solving the problem above.

1. *Teacher*: What's your induction parameter?
2. *Student*: In graphs, we can talk about nodes and edges. So one way is to induct on n, the number of nodes in the graph. Another way is to induct on m, the number of edges. At this point, I don't know which is the correct way.
3. *Teacher*: OK, so let's pick a random way, say the number of nodes. If it turns out to be problematic, we'll go the other way. The base case $n = 1$ is trivial. The graph is just an isolated node, so $O = \emptyset$. So suppose that you know how to handle connected graphs of up to n nodes. Now you have a connected graph G of $n + 1$ nodes. How would you handle it?
4. *Student*: So I probably start with two arbitrary nodes v_1 and v_2 of odd degrees. Because G is connected, I can find a path connecting them. Now, since I need to go to a smaller graph (in the sense of the number of nodes), I think I should remove v_1 and v_2, resulting in a new graph G'.
5. *Teacher*: Look at Figure 6.2. Do you see a problem?
6. *Student*: Yes, originally, $O = \{v_1, v_2, v_3, v_5\}$. After the removal of v_1 and v_2, the nodes v_3 and v_5 have *even* degree, and thus we can't use the induction hypothesis.

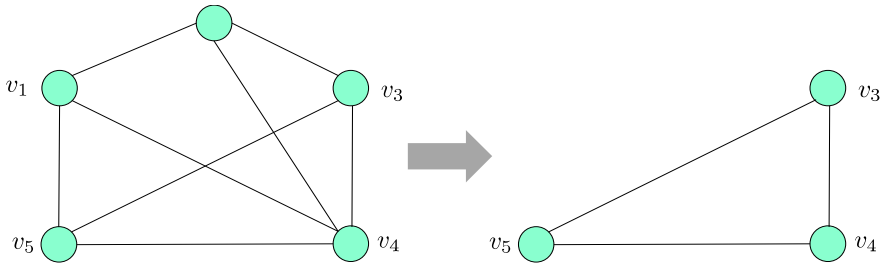

Figure 6.2: Left: the original graph G, and odd-degrees nodes are v_1, v_2, v_3, v_5. Right: the graph G' obtained by removing nodes v_1 and v_2 from G, and there is no odd-degree node.

7. *Teacher*: Right. So let's try inducting on m instead. The base case $m = 1$ is trivial. The graph has two nodes, with an edge connecting them. So suppose that you know how to handle connected graphs of up to m edges. Now you have a connected graph G of $m + 1$ edges. How would you handle it?

8. *Student*: So I probably start with two arbitrary nodes v_1 and v_2 of odd degrees. Because G is connected, I can find a path P connecting them. Now, since I need to go to a smaller graph (fewer edges), I should remove some edges. Which ones should I remove?

9. *Teacher*: To answer that, let's examine our goal. Suppose that $O = \{v_1, v_2, v_3, v_5\}$, as in Figure 6.2. After removing some edges, you again find an arbitrary path P' connecting v_3 and v_5. We need to ensure that regardless of our choice of P', this path should have no common edge with P. What does it tell you?

10. *Student*: It means that I should remove all the edges of P. In the resulting graph, any path will be edge-disjoint with P.

11. *Teacher*: Good. Removing those edges will affect the degrees of the remaining nodes. Does it matter?

12. *Student*: No. For each remaining node, if we remove some of its associated edges, we'll remove exactly two of them, so it doesn't change the oddness of the degree.

13. *Teacher*: Consider the situation in Figure 6.3. Do you see a problem there?

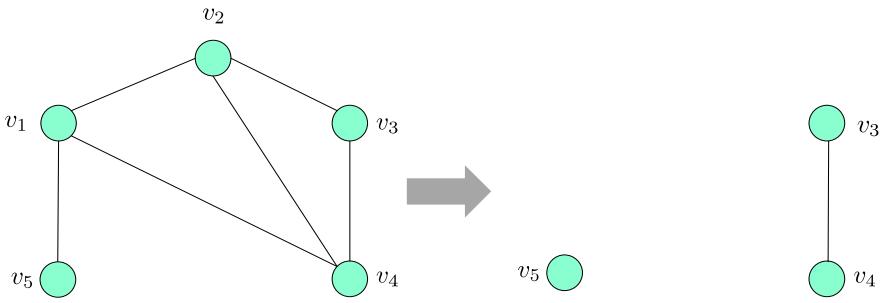

Figure 6.3: Left: the original graph G. Right: the graph G' obtained by removing nodes v_1 and v_2 from G.

14. *Student*: Yes. The new graph is disconnected, but our induction hypothesis only applies to connected ones. So I can no longer use the induction hypothesis.

15. *Teacher*: In this example, $O = \{v_1, v_2, v_3, v_4\}$. In the new graph G', although the graph is disconnected, we still can find a path connecting v_3 and v_4. So this suggests that maybe our approach (inducting on the number of edges) is not wrong, but the induction assumption is too weak. (Recall the Inventor's Paradox.) How would you strengthen it?

16. *Student*: Perhaps we'd prove that for any (possibly disconnected) undirected graph G, we can pair up nodes in O with edge-disjoint paths?

17. *Teacher*: OK, now let's start over. The base case $m = 1$ is still trivial. If we have n nodes, then without loss of generality, v_1 and v_2 are connected, and the rest are just isolated nodes. So $O = \{v_1, v_2\}$ and there's a path connecting them. So suppose that you know how to handle graphs of up to m edges. Now you have a graph G of $m + 1$ edges. How would you handle it?

18. *Student*: So I start with two nodes v_1 and v_2 of odd degrees. However, if I want to connect them, I should pick them in the same connected component of G. (There are always an even number of odd-degree nodes in each connected components.) Then there is a path P connecting them. I then remove the edges in P, resulting in a smaller graph G', and then use the induction hypothesis.

LESSON. What do we see here?

- First, the path to a solution is rarely a linear one, we usually try one route until a dead end, and then retrace our step to find another route. At first we need to find the induction parameter. Sometimes the choice is obvious (as in the Gray code), but here it's not the case. For induction proofs, a choice either works or doesn't, but for algorithm design, a poor choice may still lead to a working algorithm, but with inferior efficiency.
- Next, when we have an obstacle, we need to discern two cases: (i) the obstacle is insurmountable because earlier, we made some poor choices, and (ii) we can still overcome it by, say strengthening the induction hypothesis. In this example, we have both cases.

6.1.3 Exercises

SUM OF SERIES ELEMENTS. Consider the following series: $1, 2, 3, 4, 5, 10, 20, 40, \cdots$ which starts as an arithmetic series, but after the first 5 terms becomes a geometric series. Prove that any positive integer can be written as a sum of distinct numbers from this series.

MAP COLORING. Suppose that we have n lines in the plane, forming regions. Two regions are *neighbors* if they have only an edge in common. Prove that it is possible to color the regions so that neighboring regions have different colors.

6.2 Design of Recursive Algorithms

6.2.1 The Celebrity Problem

THE PROBLEM. Among n people, a *celebrity* is someone who is known by everybody but does not know anybody. We want to identify the celebrity , if one exists, by asking questions only of the form, "Do you know the person over there?" (The assumption is that all the answers are correct.) The goal is to ask as few questions as possible.

STUDENT-TEACHER DIALOG. Below is an imagined dialog between a student and a teacher in solving the problem above.

1. *Teacher*: Can you solve the problem using brute-force? How bad is it?
2. *Student*: Well, I can ask person i if he knows person j, for every $i \neq j$. This requires $n(n-1)$ questions.
3. *Teacher*: OK, let's try to do better using recursion. What about the base case $n = 2$?
4. *Student*: So there are two people. In this case we can't do better than brute-force, which uses 2 questions.
5. *Teacher*: So suppose that we know how to identify the celebrity given n people. How would you deal when there are $n + 1$ people?
6. *Student*: So perhaps I'll first identify the celebrity among the first n people. If there is one celebrity X, I need to to ask two extra questions: (i) Ask person $n + 1$ if he knows X, and (ii) ask X if he knows person $n + 1$.

7. *Teacher*: But what if there's no celebrity among the first n people? You can't conclude that there's no celebrity, because maybe person $n+1$ is the one.

8. *Student*: Well, if that's the case it looks like I have to ask $2n$ questions: if person $n+1$ knows everybody else, and vice versa. So in the worst case, we still have the same cost as the brute-force algorithm.

9. *Teacher*: Let's retrace our steps to see where we made a wrong choice. So initially, you singled out the last person, and apply the recursion on the rest. The choice of the last person seems arbitrary. We need to make a better choice.

10. *Student*: You mean I need to pick a particular person, call him Y, that if we recurse on the group $\{1, \ldots, n+1\} \backslash \{Y\}$, we'll never end up in the worst-case scenario?

11. *Teacher*: Yes. Let's consider the case where there's no celebrity on $\{1, \ldots, n+1\} \backslash \{Y\}$. When can we conclude that there's no celebrity on all n people without any extra cost?

12. *Student*: Well, this can happen only if somehow we already know that Y is a non-celebrity. This means at the beginning we should single out a non-celebrity. But how would we pick one?

13. *Teacher*: Can you design a recursive algorithm on this new problem? Let's start with the base case $n = 2$.

14. *Student*: For the base case $n = 2$, I simply ask if person 1 knows person 2. If yes then person 1 must be a non-celebrity. If no then person 2 must be a non-celebrity. So I only need a single question.

15. *Teacher*: Good. Now, let's say you know know to pick a non-celebrity for $n \geq 2$ people. Now you have $n+1$ people. Proceed.

16. *Student*: So maybe we again use the recursion on the first n people. If it returns a non-celebrity then we are done. If it doesn't return, wait, it must return somebody, because there are always a non-celebrity among $n \geq 2$ people.

17. *Teacher*: This recursive algorithm has running time $T(n+1) = T(n)$ if $n \geq 2$, so basically $T(n) = T(2) = 1$ for every n. If this is constant-time, it means that there's no recursion at all. Can you unfold the recursion to see what's going on?

18. *Student*: If we follow the recursion, it means given $n+1$ people, we need to look at the first n people, and then we look at the first $n-1$ people, and so on, until we have two people. So we basically just need to look at the first two people, and then run the base case.

19. *Teacher*: Back to our original problem. You can find a non-celebrity Y using one question. You then can check if $\{1, \ldots, n+1\} \backslash \{Y\}$ has a celebrity. If yes then we need two extra questions. If no then we are done. How many questions do you need in total?

20. *Student*: So $Q(n+1) = Q(n) + 3$ if $n \geq 2$, and $Q(2) = 2$. This means $Q(n) = 3n - 4$.

Lesson. What do we see here?

- Designing a recursive algorithm is very similar to doing an induction proof. In a proof, you may need a lemma that requires another induction. Likewise, in your design, you need to solve a related problem that requires another recursive algorithm.
- Next, when we apply the recursive algorithm or use the induction hypothesis on a smaller problem, it may involve a choice; if you're not careful, you may implicitly make a choice without even realizing it. Sometimes any choice will work, but sometimes a wrong choice will lead to a dead end in proofs, or to an inferior algorithm.

6.2.2 Maximal Induced Subgraph

THE PROBLEM. We want to invite guests from a list of n people. However, we want to make sure that each guest is a friend of at least k other guests so that he feels comfortable. If there are many choices, we want to invite as many people as possible.

Here's a way to formulate the problem above using graphs. We can represent each person as a node, and two nodes are connected if and only if these two people are friends of each other. Let $G = (V, E)$ be the corresponding (undirected) graph. We want to find a subgraph H of G such that each node in H has degree at least k. If there are many possible choices of H, we want to find one of maximum size.

HOW TO USE INDUCTION MINDSET. Instead of thinking about our algorithm as a sequence of steps that a computer has to take to calculate a result, think of *proving a theorem* that the algorithm exists via induction. The idea is to imitate the steps we take in proving a theorem, in order to gain insight into the problem, and then later use that to design an algorithm.

What would be the induction parameters? Here we have two numbers: n (the number of nodes) and k (the threshold of required degree). We're more familiar with inducting on the graph size, so let's do the induction on n.

Suppose we know how to solve the problem for graphs of size at most n. Now, consider a graph G of size $n + 1$. We have to find a maximum subgraph H of a certain property. If we don't care about the property, what's the maximum choice of H? Of course it's G. So if every node of G has degree at least k, we can simply pick $H = G$.

What if there's some node v such that $\deg(v) < k$? To go to a smaller graph, it's natural to think that we should remove v. After all, this person can't be our guest. This results to a smaller subgraph G' where we can use our induction hypothesis.

How would we turn the proof above into an algorithm? Here's the first attempt. Initially, we need to calculate the degrees of the nodes, which takes $O(m)$ time, where m is the number of edges of G. Then we need to check if there's a node v of degree $\deg(v) < k$. If no then we output G. Otherwise we can remove this node and recurse on the resulting graph.

The algorithm above is not very satisfactory. If G consists of a complete graph of $n/2$ nodes, and $n/2$ isolated nodes, then the running time is $\Theta(n^3)$: we'll have to do $\Theta(n)$ steps to weed out the isolated nodes and end up with the complete graph, and each step uses $\Theta(n^2)$ running time because we have to re-calculate the degrees.

To reduce the running time of calculating the degrees, note that when we remove a node, we only affect the degree of the neighbors. So we can combine the removal of v and the updating the degrees of the neighbors of v, using totally $O(\deg(v))$ time. So the total cost is $O(n + m)$.