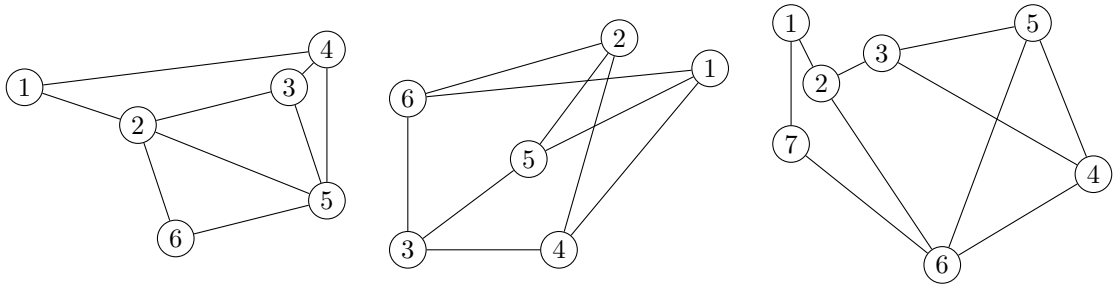
Trees and Tree Encodings

Berkeley Math Tournament

January 22, 2018

Introduction:

Today, we are going to be looking at a special class of graph theory called trees. These structures are an important discipline in mathematics and have wide applications in fields such as computer science and chemistry. Conceptually, a graph H is a collection of a set of vertices (denoted $V(H)$) and a set of edges (denoted $E(H)$) that connect them. We are working with simple graphs so only one edge can connect any two vertices and no vertex can have an edge back to itself. In this power round, we will focus on *labeled graphs* which have distinct integers 1 to n (where n is the number of vertices) associated with each vertex. Examples are shown below:



In the first example above $\{1, 2, 3, 4, 5, 6\}$ is the vertex set $V(H)$, and $\{(1, 2), (2, 3), (2, 6), (3, 5), (1, 4), (4, 5), (5, 6), (2, 5)\}$ is the edge set $E(H)$

Definition. *Two labeled graphs, G and H are **isomorphic** if there exists a one-to-one and onto mapping f between the vertex sets of G and H such that if $(u, v) \in E(G)$ then $(f(u), f(v)) \in E(H)$ and the mapping is **label preserving** (i.e u and $f(u)$ have the same label).*

A simple way to check if two graphs are isomorphic is to see if you can move the vertices of one on top of the other such that the resulting graph preserves the labels and the edge structure. For instance, consider the examples below:

Ex 1.

The two graphs in this example are isomorphic since both graphs contains the edges $(1, 3)$ and $(1, 2)$, and the labels are preserved (i.e 1 maps to 1, 2 maps to 2, and 3 maps to 3).

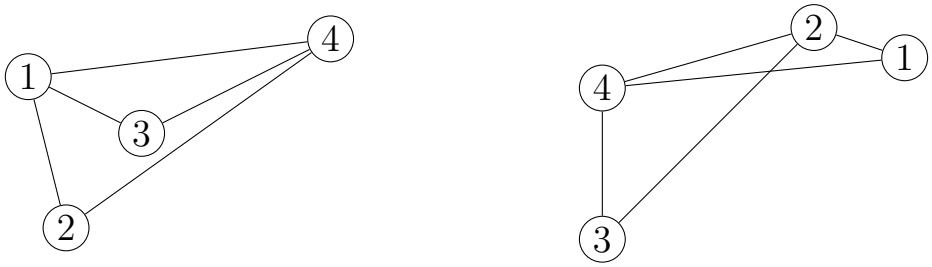
Ex 2.

These graphs however, are not isomorphic. While they are structurally the same (they both consist of a vertex connected to two other vertices), the mapping is not label preserving. In graph one, the vertex in label 1 is connected 3 and 2, whereas in the second graph it is only connected to 3.

Problem 1 [1 pt each]

Determine if the following labeled graphs are isomorphic:

A.



B.

C.

Solution

(***a***.) It is not isomorphic. Looking at the vertex with label 1's neighbors, we see that they are 2, 3, and 4 in the first graph but only 2 and 4 in the second graph.

(***b***.) It is not isomorphic. By counting the edges, we see that the first graph has 9 edges and the second graph has 10 edges, so it cannot be isomorphic.

(***c***.) These two graphs are isomorphic.

Trees and Tree Facts

Before we get to trees, we need to present a couple of definitions.

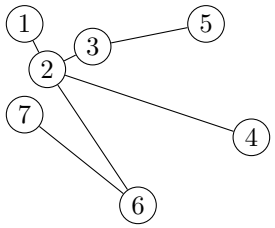
Definition. *Two vertices v_1, v_2 are **neighbors** if there exists an edge connecting them.*

Definition. *A **path** on n vertices is an ordered sequence of **distinct** vertices $v_0 ... v_n$ such that for any $i < n$ there exists an edge between v_i and v_{i+1}. (ie there is an edge between v_1 and v_2, v_2 and v_3...etc.).*

For instance the sequence of vertices corresponding to the labels $\{1, 2, 6, 5, 3\}$ in the first graph of problem $1B$ is a path, but the sequences $\{1, 3, 4, 5\}$ and $\{1, 4, 3, 2, 5, 3\}$ are not.

Problem 2 [2 pts]

Find the longest path in the following graph



Solution

The longest path is $\{7, 6, 2, 3, 5\}$ or $\{5, 3, 2, 6, 7\}$ either order works.

Definition. *A **cycle** on n vertices is a path $v_1 ... v_n$ such that there exists an edge between the starting vertex v_1 and v_n. Cycles must consist of at least 3 vertices.*

An example of a cycle in the first graph of problem $1B$ is the sequence of vertices corresponding to the labels $\{2, 3, 5\}$. However, the sequence $\{1, 2, 3, 5, 2\}$ is not a cycle since the vertex label 2 is repeated.

Definition. *A graph is **connected** if there exists a path between any two vertices in the graph*

Definition. *A **tree** is a graph T on $n \geq 2$ vertices such that for any $u, v \in V(T)$, there exists exactly one path from u to v.*

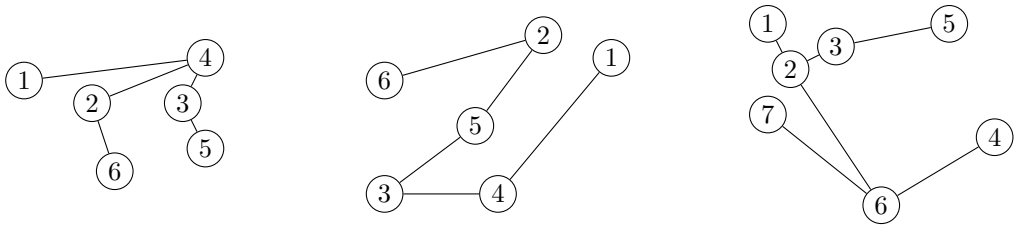
Example. *Examples of trees can be seen below:*

Problem 3 [1 pt each]

Prove that

(***a***.) All trees are connected.

(***b***.) No tree contains a cycle.



Solution

Part a is true simply by the definition of connected. If there is exaclty one path between any two vertices then there must exist a path between any two vertices, so therefore the graph is connected.

For part b, assume that there exists a cycle in the graph. Then there exists a set of vertices $v_1...v_k$ such that there is an edge between v_1 and v_k and $v_1...v_k$ is a path. However, if this is true there exists two paths between v_1 and v_k, the path v_1v_k, since v_1 and v_k share an edge and $v_1...v_k$, a contradiction since there can only be one path between any two vertices.

Definition. *The **degree** of a vertex is the number of distinct neighbors it has. For instance, in the first tree above, the degree of vertex* 4 *is 3 and the degree of vertex* 2 *is 2.*

Definition. *A **leaf** (pl. leaves) of a tree is a vertex of degree 1. For instance, the leaves on the second graph above are vertices* 6 *and* 1*.*

Problem 4 [4 pts]

Prove that every tree contains at least two leaves.

Solution

Consider and the endpoints p and q in the tree's longest path. If the endpoints of the path are not leaves, their degree must be at least 2. If p has a neighbor n in the path, then there exists a cycle from n through p (since they are on the longest path together) and back to n. However trees cannot have cycles, so this is a contradiction. This means that the endpoint's neighbor must not be on the path. However this is then not the longest path in the graph, since the path from q through p to n is a longer path. This is a contradiction, so therefore p must be leaf. The same reasoning can be applied to q, so the graph necessarily has 2 leaves.

Problem 5 [2 pts]

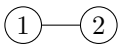
Suppose we have a tree T with $n \geq 3$ vertices. Prove that after removing a leaf and its corresponding edge from T, T is still a tree.

Solution

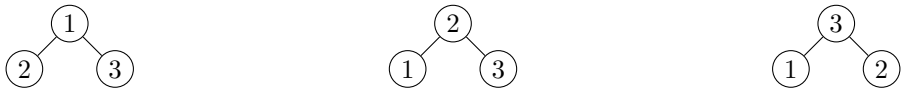
In T there exists exactly one path between any two vertices. We will prove that if we remove a leaf l, this statement is still true. If a given path from v_1 to v_2 contains l, then either $v_1 = l$ or $v_2 = l$, since any vertex that is not an endpoint of a path must connect to at least two distinct vertices (namely its neighbors in the path). Therefore, removing l does not affect whether or not T is a tree since l will not be in its vertex set anymore. If it does not contain l, then it is not affected by its removal.

Counting Labeled Trees

Now that we have the definitions, we want to be able to count all of the labeled trees on n vertices where we treat isomorphic trees the same. The number of labeled trees on two vertices is 1, since it is just the graph below:



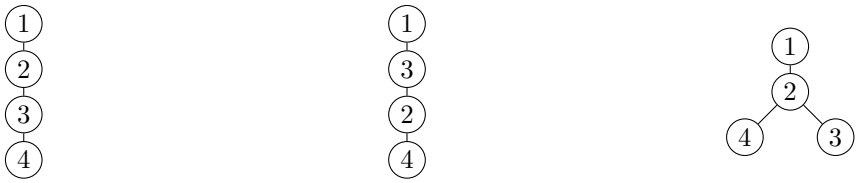
Likewise, the labeled trees up to isomorphism on 3 vertices are listed below:



We note that the label preserving distinction is important when considering labeled trees. These trees are all structurally similar (i.e all have a vertex connected to two others), but because there are different labels in the 'middle' they are counted as different. However, remember that two trees with different structure are also counted as different.

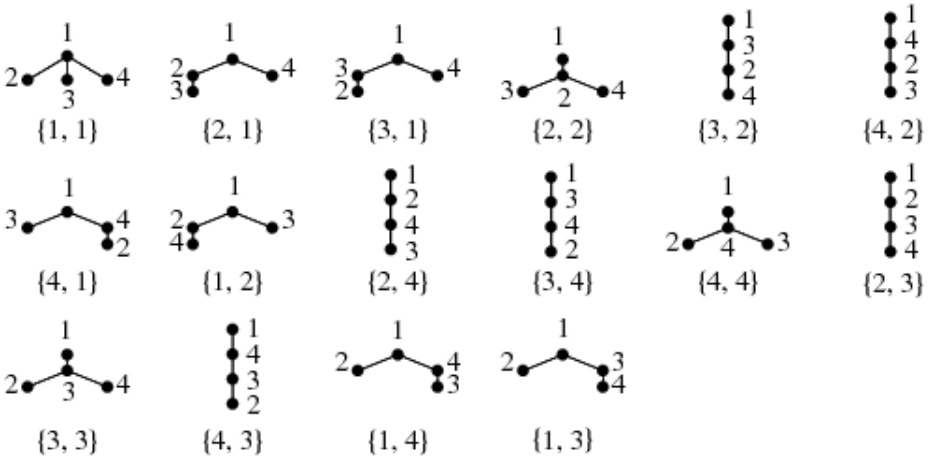
Problem 6 [3 pts]

There are 16 distinct labeled trees on 4 vertices. Draw them out! Three trees have been given to you (you do not have to draw these out).



Solution

All 16 trees are shown below:



Prüfer Codes

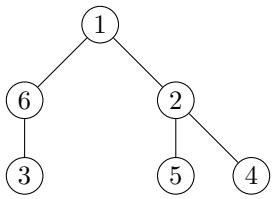
Remember how we proved that every tree must have at least 2 leaves? Well, Prüfer was able to use that to develop a recursive algorithm that equates a labeled tree on n vertices with a sequence of $n-2$ integers.

Algorithm. *Given a labeled tree T.*

1. *Find the value of the greatest labeled leaf.*
2. *Write down the integer label of the leaf's one neighbor.*
3. *Remove the leaf and the edge that goes with it.*
4. *Repeat until you reach a tree with two vertices and a single edge between them.*

The sequence of written labels is the Prüfer code of the tree.

Example. *In the below example, the Prüfer code is* ***{2, 2, 6, 1}***



Problem 7 [4 pts]

Prove that the Prüfer Code algorithm is correct. To do this show that the algorithm never gets stuck when given a labeled tree.

Solution

To prove this, we simply need to show that at each step of the algorithm, it does not get stuck. By Problem 4, we know that we can always find two leaves in any tree. Therefore, we can find the greatest labeled leaf since no two nodes can have the same label. So step 1 is satisfied.

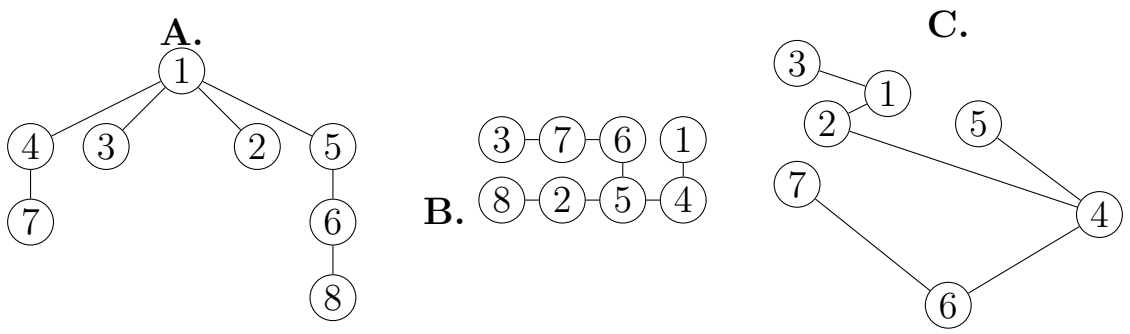
Step 2 is satisfied as long as a leaf has exactly one neighbor, which is true by the definition of leaf.

Step 3 is always satisfied since removing the leaf does not change the fact that T is a labeled tree. Additionally, removing the leaf does not change the fact that T is a tree. This is because every path that goes through the leaf must either start at the leaf or end at the leaf. Otherwise, the node would have degree 2, a contradiction.

Finally we need to prove that the algorithm converges to a tree with two vertices and a single edge between them. Since we know that the algorithm continually produces a tree and continually removes vertices, it is enough to prove that the only tree on two vertices is two vertices with a single edge between them, which is true. Therefore the algorithm never gets stuck and we are done.

Problem 8 [1 pt each]

Compute the Prüfer codes for the following trees:



Solution

(***a***.) {6, 4, 5, 1, 1, 1}

(***b***.) {2, 7, 6, 5, 5, 4 }

(***c***.) {6, 4, 4, 2, 1}

Problem 9 [2 pts]

Prove that once the algorithm terminates, the remaining tree must contain a vertex with the label 1.

Solution

Assume for the sake of contradiction that the remaining tree did not contain the vertex 1. Then, the vertex must have been removed in the algorithmic process. However, this only happens if the vertex is the greatest labeled leaf. Since every tree has at least two leaves, this cannot be the case since 1 is the minimal number on the tree. This is a contradiction.

Problem 10 [3 pts]

Let j_k be the number of times the integer k ($\leq n$) appears in the Prüfer code. Prove that the degree of the node with the k label is $j_k + 1$

Solution

If the vertex corresponding to the k label is a leaf, then since the algorithm only write down the neighbors of leaves, it cannot appear in the sequence. This satisfies the condition above since a leaf has degree 1 and appears 0 times. If k is not a leaf, then it appears in the sequence exactly $d - 1$ number of times where d is the degree of k. This is because exactly $d - 1$ neighbors need to be removed before it becomes a leaf. Therefore, $j_k = d - 1$, exactly what we wanted to prove.

Problem 11 [3 pts]

Find and prove necessary and sufficient conditions on a labeled tree T such that T has a constant Prüfer code (such as 1111 or 222222). In other words, if T satisfies those conditions, it must have a constant Prüfer code and if a T has a constant Prüfer code, it must satisfy those conditions.

Solution

We define a *star* to be a graph with one internal node and $n-1$ leaves. We will prove that G is a star if the internal node has label l, then the G's Prüfer code consists of $n-2$ copies of l. Since the graph is a star, every vertex except the internal vertex is a leaf and neighbors the internal vertex. Therefore, when we apply the algorithm to it, whenever we remove a leaf, we write down the label l. We must do this exactly $n-2$ times before we can have the vertex that corresponds with the label l be a leaf. But, by this time we have already finished the Prüfer Code. So this graph has a constant Prüfer code.

Now, going in the opposite direction, suppose G with n vertices has a constant Prüfer code with integer labels l. Then, by problem 10, the vertex L with label l has degree $n-1$. However if this is true then the graph must be a star since if we were to add an edge between any two vertices v_1 and v_2, we would create a cycle (namely $\{v_1, v_2, L\}$). We cannot add any edges to l since it is already connected to all other vertices.

Problem 12 [2 pts each]

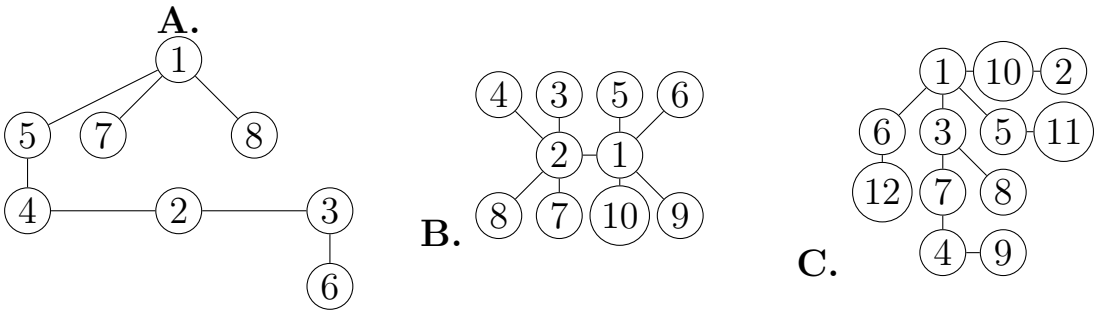
Draw out the trees corresponding to the following Prüfer codes.

(***a***.) $[1, 1, 3, 2, 4, 5]$

(***b***.) $[1, 1, 2, 2, 1, 1, 2, 2]$

(***c***.) $[6, 5, 4, 3, 1, 1, 7, 3, 1, 10]$

Solution



Problem 13 [5 pts / 2 pts]

(***a***.) Find a deterministic algorithm that takes a Prüfer sequence and produces the tree corresponding to the Prüfer code. This will establish a 1-1 correspondence between the set of Prüfer codes and the set of labeled trees. You do not need to prove that your algorithm is correct.

(***b***.) Prove that the number of labeled trees on n vertices is n^{n-2}. This is known as Cayley's theorem.

Solution

(***a.***) We present now the algorithm that reconstructs a tree from the Prüfer code:

Algorithm. *Given a Prüfer code:*

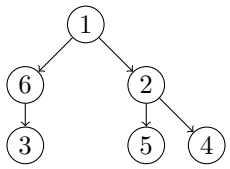
1. *Draw the n nodes of the tree, and label them from 1 to n.*
2. *Make a list of all the integers $(1, 2, \ldots, n)$. This will be called the list.*
3. *If there are two numbers left in the list, connect them with an edge and then stop. Otherwise, continue on to step 4.*
4. *Find the largest number in the list which is not in the sequence. Take the first number in the sequence. Add an edge connecting the nodes whose labels correspond to those numbers.*
5. *Delete the largest number from the list and the first number in the sequence. This gives a smaller list and a shorter sequence. Then return to step 3.*

(***b.***) There is a 1-1 correspondence between sequences of length $n-2$ and the labeled trees. So we only need to count the number of sequences. There are n choices for each of the $n-2$ positions and so there are a total of n^{n-2} labeled trees on n vertices.

Fleiner Codes

We now turn to a more recent paper that provided another proof of Cayley's theorem by constructing another graph encoding. This encoding takes advantage of another fact about trees called edge orientation. Because a tree has no cycles, we can orient edges away from a specific vertex. In our case, we will **always orient edges away from the "1" vertex**. An example is shown below:

Example. *In this example, all of the edges are oriented away from the "1" vertex.*



Because of this orientation, we can now define a function on the edges.

Definition. *Given an edge $e \in E(G)$, we let $c(e)$ be the label of the vertex at the tail of e. For example in the graph above, $c(\{1,2\}) = 1$ since the edge is oriented away from 1 and towards 2. For the remainder of this test, you may assume the algorithm is correct and does not get stuck.*

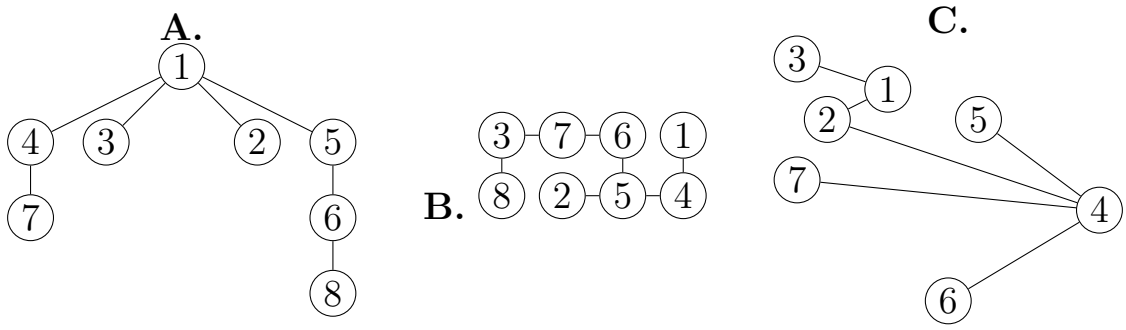
Algorithm. *Given a labeled tree T with n vertices.*

1. *Start at the vertex with label 1.*
2. *Set $F = \{\}$ (make it an empty sequence). This will end up becoming our Fleiner Code*
3. *Repeat the following loop from $i = 2$ up to n.*
   (a) *Traverse the path from label 1 to label i*
   (b) *If you traverse an edge e that you have never traversed before, add $c(e)$ to F*
4. *After running the loop, the sequence F is our* ***Fleiner Code***

Example. *In the above tree example, the Fleiner code is {1, 1, 6, 2, 2}*

Problem 14 [1 pt each]

Find the Fleiner Codes of the following graphs.



Solution

A. $[1, 1, 1, 1, 5, 4, 6]$

B. $[1, 4, 5, 5, 6, 7, 3]$

C. $[1, 1, 2, 4, 4, 4]$

Problem 15 [1 pt]

Prove that the first number of any tree's Fleiner Code must be 1.

Solution

The first number of any tree's Fleiner Code must be 1 because we start at the vertex labeled with 1 and we must traverse to vertex 2, which would take at least 1 other vertex along the path, and since each edge is directed away from 1, the first number of the sequence is 1.

Problem 16 [4 pts]

Find and prove necessary and sufficient conditions on a labeled tree T such that T has a Fleiner Code with all integers distinct.

Solution

We will define a *linear graph* on n vertices as a tree with two leaves v_1 and v_2 such that the path from v_1 to v_2 contains all vertices in the graph. We will prove that a graph G has a distinct Fleiner code if and only if it is a linear graph and the vertex corresponding to label 1 is a leaf.

If G is such a graph, then since every edge is oriented away from 1 the first $n-1$ vertices have exactly one edge coming out of it. Therefore, each label can only be written down once. Since there are $n-1$ edges, $n-1$ distinct vertices must be written down.

Going in the opposite direction, if G is not a linear graph with 1 as a leaf, we will prove that there must be a repeated vertex. If G is a linear graph such that 1 is not a leaf, consider two of its neighbors with labels

v and w. Since both edges from 1 to v must eventually be traversed 1 will be written down twice, so the code will not be distinct. We note that this argument applies for any graph where 1 is not a leaf.

If v with label 1 is a leaf, but G is not linear, then there exists w and l such that l is a leaf and w is not on the path from v to l. Let p be the path from v to l and q be the path from v to w. Finally let d be the vertex where these paths diverge. We claim that the label of d is repeated twice in the Fleiner code. We see that d contains two edges emanating out of it, one towards w and one towards l. Since both of these edges must be traversed, d must be written down twice. Since we have covered all conditions so we are done.

Problem 17 [8 pts]

Prove that the reverse of a labeled tree T's Prüfer code is the last $n-2$ digits of that tree's Fleiner code.

Solution

We shall prove this via induction. In particular, we'll show that the first number in the Prufer code is the last number of the Fleiner code. The first number of the Prufer code is the label of the edge connected to the largest leaf. To do this, we shall prove the following:

Lemma. *Let P_i be the set of all paths from 1 to a leaf (with respect to the oriented pointed graph). Each vertex is contained in at least one P_i*

Proof. Let vertex v be a vertex of greatest distance from 1 that is not contained in such a path. If there v shares an edge with another vertex w and $c(v,w) = v$, then w has greater distance from 1 than v does (since a tree contains no cycles). If $c(v,w) = w$ for each w connected to v, then there is only 1 such w (since otherwise, there would be cycles), and so v is a leaf. But there exists a path from 1 to a leaf, so this is a contradiction. Hence the lemma. □

The lemma proves that every number is contained in a path from 1 to a leaf, and thus, the last number in the Fleiner code is the vertex connected to the largest leaf. Now take away the largest leaf and repeat the process. Because Prufer/Fleiner codes are finite in length, our algorithm terminates. Hence the result.

Problem 18 [6 pts / 2 pts]

(***a.***) Find an prove a formula for the number of labeled trees on n vertices such that the distance between vertices u and v with respective labels 1 and 2 is k. Here distance is defined to be the number of edges in the path from u to v. The formula should be in terms of n and k only.

(***b.***) Calculate the number of labeled trees on 6 vertices such that the distance between vertices u with label 1 and v with label 2 is 3. Once again, distance is defined to be the number of edges in the path from u to v.

Solution

(***a.***) We consider Fleiner Codes. We will start at the beginning and choose numbers that satisfy our condition. We see that as soon as a number is repeated in the code or the number 2 is reached, that the path from 1 to 2 has been traversed. We note that for a distance k between 1 and 2, there must be k numbers in the Fleiner code before either a repetition or the number 2 appears.

The number of choices for the first vertex is 1 (it has to be 1). The number of choices for the second vertex is $n-2$ (it cannot be 1 or 2). The number of choices for the third vertex is $n-3$ (it cannot be 1, 2, or the previous vertex label). We continue this until we reach $n-k$, which will define the k vertices before the repetition. We can write this out in product form as $\prod_{i=2}^{k}(n-i)$ (where the empty product is 1).

Now we need to choose the integer that is repeated. There are $k+1$ possible choices for this vertex (namely each of the k previous labels and 2 itself).

Finally we need to choose the rest of the $n-1-k-1 = n-k-2$ numbers. This can be done in n^{n-k-2} different ways.

Therefore our formula is $\prod_{i=2}^{k}(n-i)\cdot(k+1)\cdot n^{n-k-2}$

(***b.***) We simply plug in our formula for $n=6$ and $k=3$ to obtain $(6-2)(6-3)(3+1)(6^{6-3-2}) = 4\cdot3\cdot4\cdot6 = 288$

Conclusion

Trees are one of the most interesting aspects of mathematics. Yet, the mathematics that was done in this power round is not only an integral part of Graph Theory, but also useful in real-world computer science applications!

Sources

1. https://web.cs.elte.hu/egres/tr/egres-05-16.pdf
2. https://dcg.epfl.ch/files/content/sites/dcg/files/users/andres/GT2015/CayleyPrufer.pdf