forthright48

Learning Never Ends

Monday, January 29, 2018

Prufer Code: Linear Representation of a Labeled Tree

I guess this is going to be my first post (apart from the contest analysis') which is not about Number Theory! It's not about graph either, even though the title has "Tree" in it. This post is actually about Combinatorics.

Prufer code, in my opinion, is one of the most underrated algorithms I have learned. It has a wide range of applications, yet, very few people seem to know about it. It's not even a hard algorithm! It's very easy and you should understand it intuitively.

In this post, we will be discussing what is Prufer Code and its conversion to and from a tree. On next post, we will look into some of its application.

Prufer Code: What is it?

Every labeled tree of N nodes can be uniquely represented as an array of $N-2$ integers and vice versa

The linear array representation of a labeled tree is called Prufer Code. In case you are wondering what's a labeled tree, it's just a tree with distinguishable node, i.e, each node is marked such that they can be distinguished from one another.

Okay, before we move on to how to covert a labeled tree into Prufer Code and vice versa, let me tell you how I came to know about Prufer Code. If you are a problem setter, the story will be helpful to you.

How I came to know about Prufer Code

Follow by Email

Email address... Submit

Labels

Analysis Arithmetic Function Backtrack Big Int Binary Bitwise Combinatorics Complexity Contest CPPS D&C Divisors Factorial Factorization GCD Graph Language LCM Logarithm Math Modular Arithmetic Number Theory Optimization Primality Test Prime Proof Repeated Squaring Sequence Sieve SPOJ Theorem Tree UVa

Blog Archive

▼ 2018 (2)
- ► Feb (1)
- ▼ Jan (1)
  - Prufer Code: Linear Representation of a Labeled Tr...

► 2017 (2)
► 2015 (35)

So a few years ago, I used to create challenges for HackerRank as a "Challenge Creator". So one day, I created a challenge involving tree. Since the problem had a tree in it, I had to create some random trees for the data set. I used the following algorithm to create the random trees:

```
1. Start with node 1. Initially, there is only node 1 by itself. Totally independent.
2. Now, for each node x from 2 to N:
  a. Node 1 to x-1 has already been processed. They are now connected as a tree.
  b. Choose a node randomly from node 1 to x-1 and connect node x with it.
3. Your tree is ready.
```

I thought this was a pretty reasonable process for getting myself a tree. So I submitted the challenge for testing. Back then, Wanbo used to test all my challenges. He no longer works at HR now. Anyways, as a tester, it was Wanbo's duty to write bad solutions to test my challenge. He looked into the dataset and realized that my data set wasn't truly random. He quickly figured out a pattern and wrote a heuristical solution that got AC. So he sent my challenge back to me and asked to improve the dataset.

I started to analyze my tree generator again. Why is it not random? What's wrong with it? After some thinking, I realized that in the pseudocode above, node number 1 has a higher chance of getting a child. As a result, trees created by that generator had higher degrees for nodes with a lower index. This can't be called a random tree generator.

Once I realized my mistake, I notified Wanbo about it. He then suggested me to google Prufer Code.

And that's how I came to know about Prufer code. After learning Prufer Code, creating random trees became trivial. All I had to do is create a random array of length $N-2$ and then convert it to a labeled tree. It was truly random without any pattern.

Creating random trees is just one of the application of Prufer Code. There is more.

Let us see how to find the Prufer Code of a given tree.

Convert a Labeled Tree into Prufer Code

The process of converting a labeled tree into its Prufer Code is very simple. The pseudocode is given below:

```
1. Find the smallest leaf node of the given tree. Let it be x. Let the neighbor of node
x be y.
2. Write down the value of y on a piece of paper.
3. Remove node x from the tree.
4. Repeat step 1 to 3 until there are only 2 nodes remaining.
5. Once you are done, there will be a sequence of numbers on the piece of paper. That's
```


Follow me on Twitter

Tweets by @forthright48

MohammadSamiul Islam @forthright48

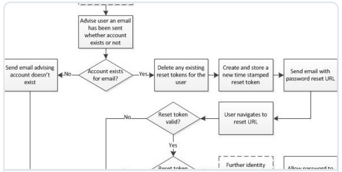
I don't know why I have been using default bash shell all these years :| ohmyz.sh Glad I found @ohmyzsh

oh my zsh
Oh-My-Zsh is a d…
ohmyz.sh

Jul 19, 2018

MohammadSamiul Islam @forthright48

Resetting password is such a headache! Maybe I should dump basic auth and just move to third party auth service.troyhunt.com/everything-you…



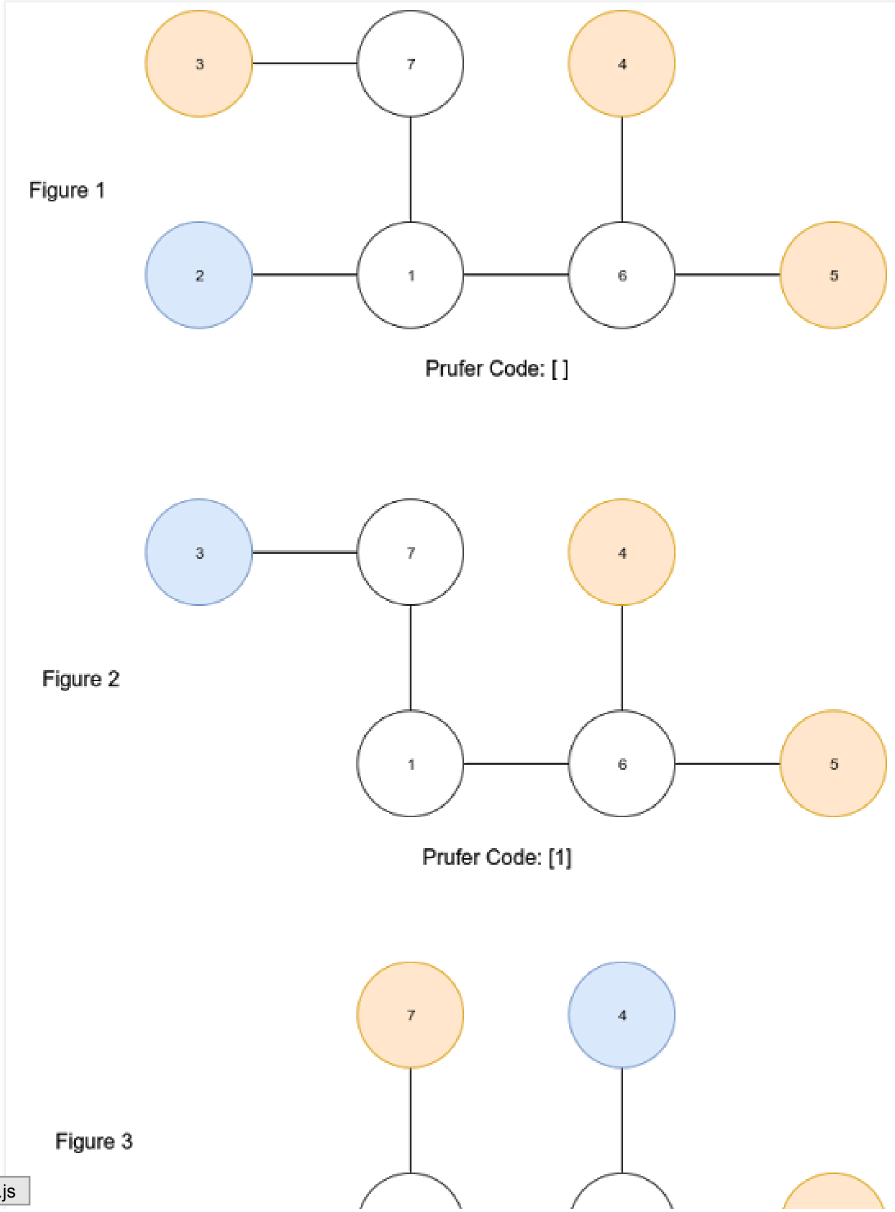
Embed View on Twitter

Tweets by @forthright48

Followers

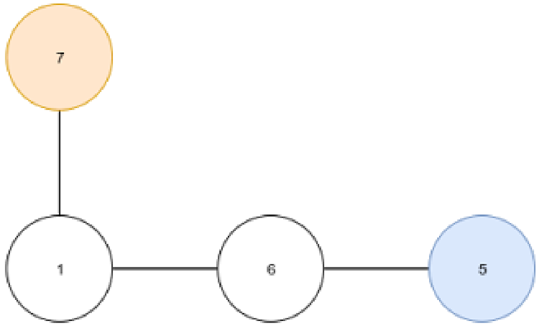
In order to help you understand, I even drew a picture!

Here, the white nodes are non-leaf nodes, the green nodes are leaf nodes and the blue nodes are the smallest leaf node for that tree.



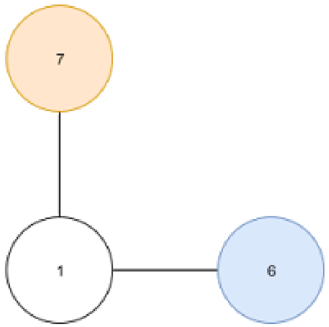
Prufer Code: [1, 7]

Figure 4



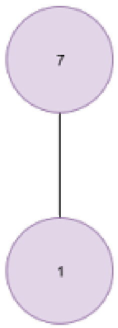
Prufer Code: [1, 7, 6]

Figure 5



Prufer Code: [1, 7, 6, 6]

Figure 6



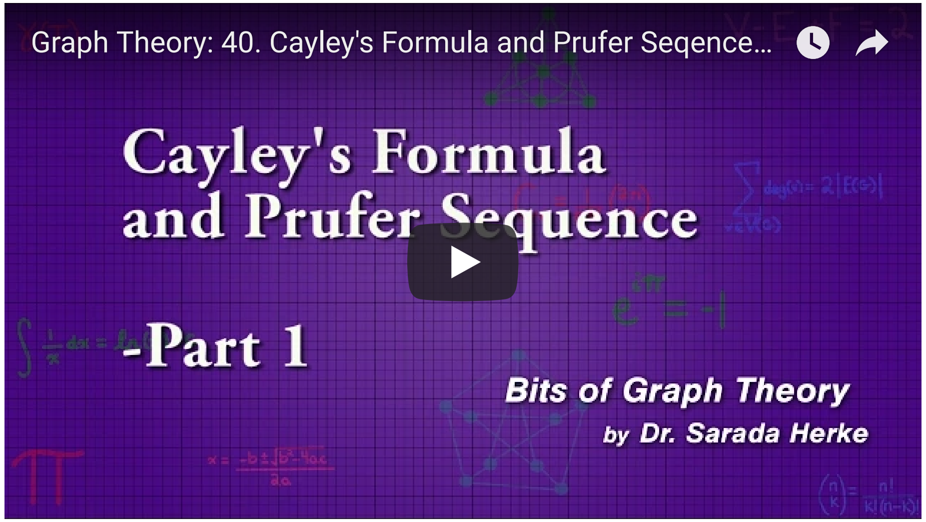
Prufer Code: [1, 7, 6, 6, 1]

So for the tree given on picture, the Prufer Code is $1, 7, 6, 6, 1$. It's simple right?

Before we move on, we will also note down few properties of Prufer Code:

1. If a node has degree d, then that node will appear in prufer code exactly $d - 1$ times.
2. Leaves never appear in Prufer Code.

If you are still having difficulty in understanding, have a look at this video.



Code for Converting Tree into Prufer Code

We are just going to convert the idea we discussed above into code. Nothing fancy.

```
/*
 Tree to Prufer Code
 Complexity: O(VlogV)
*/

vector<int> treeToPrufercode (int nodes, vector<pair<int,int>> &edges) {
    unordered_set<int> neighbors[nodes+1]; // For each node, who is it's neighbor?

```

```cpp
    for( int i = 0; i < edges.size(); i++ ) {
        pair<int,int> edge = edges[i];
        int u = edges[i].first; int v = edges[i].second;
        neighbors[u].insert(v);
        neighbors[v].insert(u);
    }

    priority_queue<int> leaves;
    for ( int i = 0; i <= nodes; i++ ) {
        if (neighbors[i].size() == 1 ) {
            leaves.push(-i); // Negating since we need min heap
        }
    }
    vector<int> pruferCode;
    int need = nodes - 2;
    while(need--) {
        int leaf = -leaves.top(); leaves.pop();
        int neighborOfLeaf = *(neighbors[leaf].begin());
        pruferCode.push_back(neighborOfLeaf);
        // Remove the leaf
        neighbors[neighborOfLeaf].erase(leaf);
        // The neighbor can become a new leaf
        if(neighbors[neighborOfLeaf].size() == 1) {
            leaves.push(-neighborOfLeaf);
        }
    }
    return pruferCode;
}
```


Complexity: $O(VlogV)$

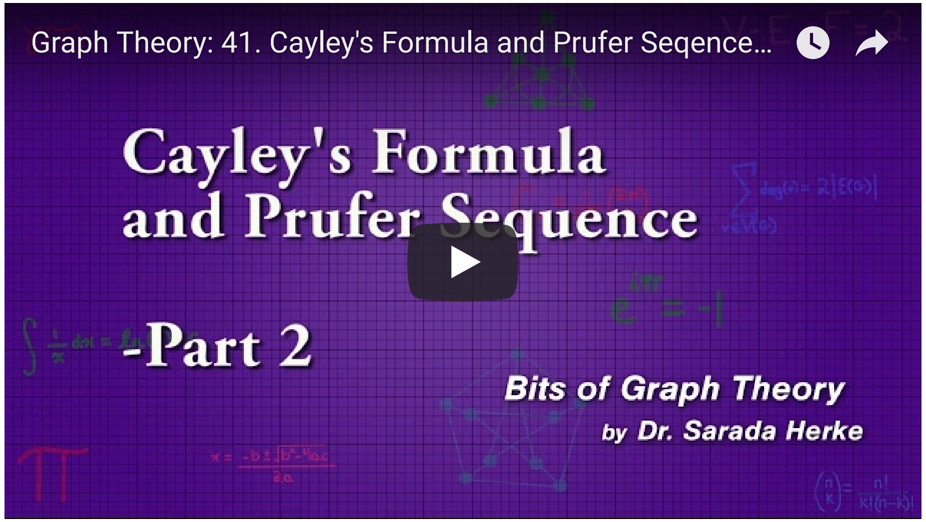
I have tested the above code with some hand generated test cases. I would have felt much more confident if I could have gotten AC in some problem from an online judge. But unfortunately, I never found any such problem. The other way around is more common.

Convert a Prufer Code into Labeled Tree

So given a Prufer Code, how do we convert it into a tree? The algorithm is just reverse of what we have done before.

```
1. Find the node numbers who are missing in the Prufer Code. They are the leaves of the
tree since leaves never make it to Prufer Code. Let L be the set of leaf nodes.
2. Select the smallest leaf from L and connect it to the first element of Prufer Code.
3. Remove the first element from Prufer Code. Let the removed element be X.
4. Check if X is still present in the Prufer code or not. If it has disappeared, then X
has become a leaf itself. Add it in the set L.
5. Repeat step 2 to 4 until the full sequence disappears.
6. Once the process ends, you will have a connected labeled tree.
```

I am not going to draw a picture again. Too much effort. Just watch the following video if you are still confused.



Code for Converting Prufer Code to Tree

We will convert the process we discussed above into code.

```
/*
  Prufer Code to Tree
  Complexity: O(VlogV)
 */

vector<pair<int,int>> pruferCodeToTree(vector<int> &pruferCode) {
    // Stores number count of nodes in the prufer code
```

```
    unordered_map<int,int> nodeCount;

    // Set of integers absent in prufer code. They are the leaves
    set<int> leaves;

    int len = pruferCode.size();
    int node = len + 2;

    // Count frequency of nodes
    for ( int i = 0; i < len; i++ ) {
        int t = pruferCode[i];
        nodeCount[t]++;
    }

    // Find the absent nodes
    for ( int i = 1; i <= node; i++ ) {
        if ( nodeCount.find ( i ) == nodeCount.end() ) leaves.insert ( i );
    }

    vector<pair<int,int>> edges;
    /*Connect Edges*/
    for ( int i = 0; i < len; i++ ){
        int a = prufer[i]; // First node

        //Find the smallest number which is not present in prufer code now
        int b = *leaves.begin(); // the leaf

        edges.push_back({a,b}); // Edge of the tree

        leaves.erase ( b ); // Remove from absent list
        nodeCount[a]--; // Remove from prufer code
        if ( nodeCount[a] == 0 ) leaves.insert ( a ); // If a becomes absent
    }

    // The final edge
    edges.push_back({*leaves.begin(), *leaves.rbegin()});
    return edges;
```

Loading [MathJax]/extensions/MathMenu.js

```
45   }
```

prufer_code_to_tree.cpp hosted with ❤ by GitHub view raw

Complexity: $O(VlogV)$

Coverting Prufer Code to Tree is much more common than the other way around. There is even a problem on Timus on this conversion: Timus 1069. Prufer Code. I believe you now have enough knowledge to solve Timus 1069.

Conclusion

We now know how to convert a labeled tree into Prufer Code and vice versa. Not only we know the process, we even know how to code them.

On next post (Application of Prufer Code: Random Tree Generation, Cayley's Formula and Building Tree from Degree Count), we will look into some of its common applications and solve some interesting problems related to Prufer Code. Make sure you understand the process clearly before proceeding.

If you enjoyed reading the post, then don't forget to share it with your friends.

Resources

1. Wiki - Prüfer sequence
2. Youtube - Graph Theory: 40. Cayley's Formula and Prufer Seqences part 1/2
3. Youtube - Graph Theory: 41. Cayley's Formula and Prufer Seqences part 2/2

Related Problems

1. Timus 1069 - Prufer Code

Next Posts

1. Application of Prufer Code: Random Tree Generation, Cayley's Formula and Building Tree from Degree Count

Posted by Mohammad Samiul Islam

 aph, Sequence, Tree

No comments:

Post a Comment

Leave comments for Queries, Bugs and Hugs.

Enter your comment...

Comment as: Google Accoun

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Popular Posts

SPOJ LCMSUM - LCM Sum

Problem Problem Link - SPOJ LCMSUM Given n, calculate the sum $LCM(1,n) + LCM(2,n) + \ldots + LCM(n,n)$, where $LCM(i,n)$ denotes the ...

Euclidean Algorithm - Greatest Common Divisor

Problem Given two number A and B, find the greatest number that divides both A and B. What we are trying to find here is the Greatest Comm...

Extended Euclidean Algorithm

Extended Euclidean Algorithm is an extension of Euclidean Algorithm which finds two things for integer a and b: It finds the value of...

Chinese Remainder Theorem Part 1 - Coprime Moduli

Second part of the series can be found on: Chinese Remainder Theorem Part 2 - Non Coprime Moduli Wow. It has been two years since I pub...

Prufer Code: Linear Representation of a Labeled Tree

I guess this is going to be my first post (apart from the contest analysis') which is not about Number Theory! It's not about graph ...

Loading [MathJax]/extensions/MathMenu.js ratosthenes

Problem Given two integers A and B, find number of primes inside the range of A and B inclusive. Here, $1 \leq A \leq B \leq 10^{...

Sieve of Eratosthenes - Generating Primes
Problem Given an integer N, generate all primes less than or equal to N. Sieve of Eratosthenes - Explanation Sieve of Eratosthenes ...

Number of Digits of Factorial
Problem Given an integer N, find number of digits in $N!$. For example, for $N = 3$, number of digits in $N! = 3! = 3\times 2\times 1...

Euler Totient or Phi Function
I have been meaning to write a post on Euler Phi for a while now, but I have been struggling with its proof. I heard it required Chinese Rem...

Chinese Remainder Theorem Part 2 - Non Coprime Moduli
As promised on the last post, today we are going to discuss the "Strong Form" of Chinese Remainder Theorem, i.e, what do we do whe...