Prüfer sequence

From Wikipedia, the free encyclopedia

In combinatorial mathematics, the **Prüfer sequence** (also **Prüfer code** or **Prüfer numbers**) of a labeled tree is a unique sequence associated with the tree. The sequence for a tree on n vertices has length $n - 2$, and can be generated by a simple iterative algorithm. Prüfer sequences were first used by Heinz Prüfer to prove Cayley's formula in 1918.[1]

Contents

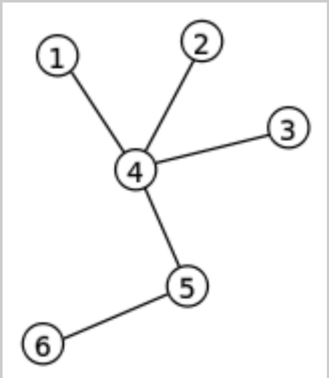
Algorithm to convert a tree into a Prüfer sequence

One can generate a labeled tree's Prüfer sequence by iteratively removing vertices from the tree until only two vertices remain. Specifically, consider a labeled tree T with vertices $\{1, 2, ..., n\}$. At step i, remove the leaf with the smallest label and set the ith element of the Prüfer sequence to be the label of this leaf's neighbour.

The Prüfer sequence of a labeled tree is unique and has length $n - 2$.

Example

Consider the above algorithm run on the tree shown to the right. Initially, vertex 1 is the leaf with the smallest label, so it is removed first and 4 is put in the Prüfer sequence. Vertices 2 and 3 are removed next, so 4 is added twice more. Vertex 4 is now a leaf and has the smallest label, so it is removed and we append 5 to the sequence. We are left with only two vertices, so we stop. The tree's sequence is $\{4,4,4,5\}$.



A labeled tree with Prüfer sequence $\{4,4,4,5\}$.

Algorithm to convert a Prüfer sequence into a tree

Let `{a[1], a[2], ..., a[n]}` be a Prüfer sequence:

The tree will have `n+2` nodes, numbered from `1` to `n+2`. For each node set its degree to the number of times it appears in the sequence plus 1. For instance, in pseudo-code:

```
Convert-Prüfer-to-Tree(a)
n ← length[a]
T ← a graph with n + 2 isolated nodes, numbered 1 to n + 2
degree ← an array of integers
for each node i in T
    do degree[i] ← 1
for each value i in a
    do degree[i] ← degree[i] + 1
```

Next, for each number in the sequence `a[i]`, find the first (lowest-numbered) node, `j`, with degree equal to 1, add the edge `(j, a[i])` to the tree, and decrement the degrees of `j` and `a[i]`. In pseudo-code:

```
for each value i in a
    for each node j in T
        if degree[j] = 1
            then Insert edge[i, j] into T
                 degree[i] ← degree[i] - 1
                 degree[j] ← degree[j] - 1
                 break
```

At the end of this loop two nodes with degree 1 will remain (call them `u`, `v`). Lastly, add the edge `(u,v)` to the tree.[2]

```
u ← v ← 0
for each node i in T
    if degree[i] = 1
        then if u = 0
            then u ← i
            else v ← i
                break
Insert edge[u, v] into T
degree[u] ← degree[u] - 1
degree[v] ← degree[v] - 1
return T
```

Cayley's formula

The Prüfer sequence of a labeled tree on n vertices is a unique sequence of length $n - 2$ on the labels 1 to n — this much is clear. Somewhat less obvious is the fact that for a given sequence S of length n–2 on the labels 1 to n, **there is a *unique* labeled tree whose Prüfer sequence is *S*.**

The immediate consequence is that Prüfer sequences provide a bijection between the set of labeled trees on n vertices and the set of sequences of length n–2 on the labels 1 to n. The latter set has size n^{n-2}, so the existence of this bijection proves Cayley's formula, i.e. that there are n^{n-2} labeled trees on n vertices.

Other applications[3]

- Cayley's formula can be strengthened to prove the following claim:

  The number of spanning trees in a complete graph K_n with a degree d_i specified for each vertex i is equal to the multinomial coefficient
  $$\binom{n-2}{d_1-1,\, d_2-1,\, \ldots,\, d_n-1} = \frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}.$$
  The proof follows by observing that in the Prüfer sequence number i appears exactly $(d_i - 1)$ times.

- Cayley's formula can be generalized: a labeled tree is in fact a spanning tree of the labeled complete graph. By placing restrictions on the enumerated Prüfer sequences, similar methods can give the number of spanning trees of a complete bipartite graph. If G is the complete bipartite graph with vertices 1 to n_1 in one partition and vertices $n_1 + 1$ to n in the other partition, the number of labeled spanning trees of G is $n_1^{n_2-1} n_2^{n_1-1}$, where $n_2 = n - n_1$.
- Generating uniformly distributed random Prüfer sequences and converting them into the corresponding trees is a straightforward method of generating uniformly distributed random labelled trees.

References

1. Prüfer, H. (1918). "Neuer Beweis eines Satzes über Permutationen". *Arch. Math. Phys*. **27**: 742–744.
2. Jens Gottlieb; Bryant A. Julstrom; Günther R. Raidl; Franz Rothlauf. (2001). "Prüfer numbers: A poor representation of spanning trees for evolutionary search" (PDF). *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*: 343–350.
3. Kajimoto, H. (2003). "An Extension of the Prüfer Code and Assembly of Connected Graphs from Their Blocks". *Graphs and Combinatorics*. **19**: 231–239. doi:10.1007/s00373-002-0499-3.

External links

- Prüfer code (http://mathworld.wolfram.com/PrueferCode.html) – from MathWorld

Prüfer Sequence from Labeled Tree

From ProofWiki

Contents

Algorithm

Given a finite labeled tree, it is possible to generate a Prüfer sequence corresponding to that tree.

Let T be a labeled tree of order n, where the labels are assigned the values 1 to n.

> **Step 1**: If there are two (or less) nodes in T, then **stop**. Otherwise, continue on to **step 2**.
>
> **Step 2**: Find all the nodes of T of degree 1. There are bound to be some, from Finite Tree has Leaf Nodes. Choose the one v with the lowest label.
>
> **Step 3**: Look at the node v adjacent to v, and assign the label of v to the first available element of the Prüfer sequence being generated.
>
> **Step 4**: Remove the node v and its incident edge. This leaves a smaller tree T . Go back to **step 1**.

The above constitutes an algorithm, for the following reasons:

Finiteness

For each iteration through the algorithm, **step 4** is executed, which reduces the number of nodes by 1.

Therefore, after $n - 2$ iterations, at **step 1** there will be 2 nodes left, and the algorithm will stop.

Definiteness

> **Step 1**: There are either more than 2 nodes in a tree or there are 2 or less.
>
> **Step 2**: There are bound to be some nodes of degree 1, from Finite Tree has Leaf Nodes. As integers are totally ordered, it is always possible to find the lowest label.
>
> **Step 3**: As the node v is of degree 1, there is a unique node v to which it is adjacent. (Note that this node will not *also* have degree 1, for then vv would be a tree of order 2, and we have established from step 1 that this is not the case.)
>
> **Step 4**: The node and edge to be removed are unique and specified precisely, as this is a tree we are talking about.

Inputs

The input to this algorithm is the tree T.

Outputs

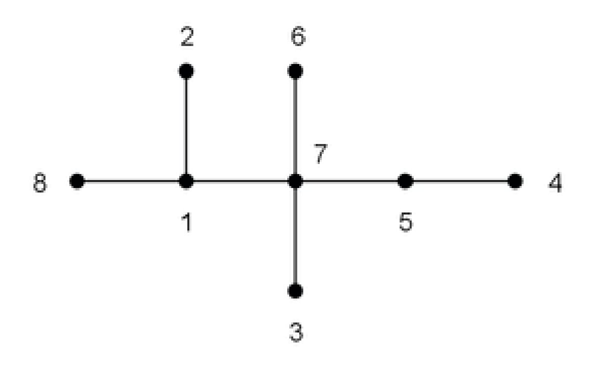
The output to this algorithm is the Prüfer sequence $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n-2})$.

Effective

Each step of the algorithm is basic enough to be done exactly and in a finite length of time.

Example

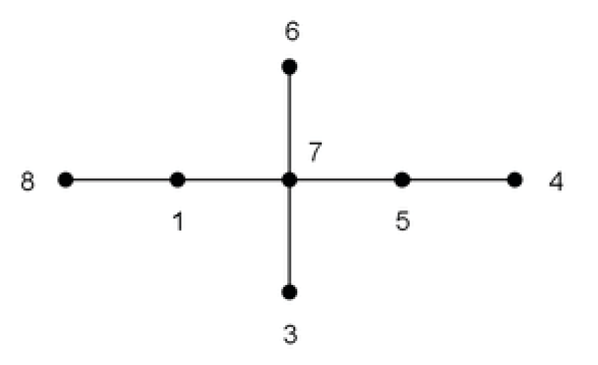
Let T be the following labeled tree:



This tree has 8 nodes, so the corresponding Prüfer sequence will have 6 elements.

Iteration 1

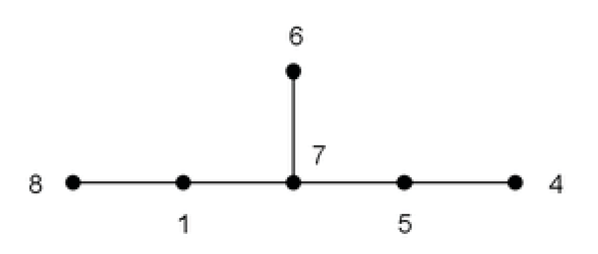
> **Step 1**: There are 8 nodes, so continue to **step 2**.
>
> **Step 2**: The nodes of degree 1 are $8, 2, 6, 4, 3$ Of these, 2 is the lowest.
>
> **Step 3**: 2 is adjacent to 1, so add $\mathbf{1}$ to the Prüfer sequence.
>
> **Step 4**: Removing node 2 leaves the following tree:



At this stage, the Prüfer sequence is $(\mathbf{1})$.

Iteration 2

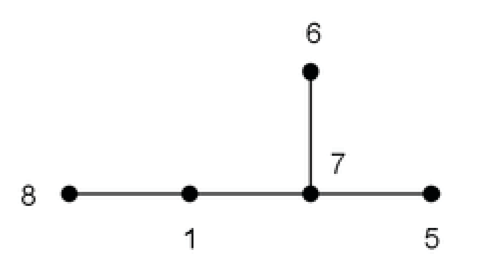
> **Step 1**: There are 7 nodes, so continue to **step 2**.
>
> **Step 2**: The nodes of degree 1 are $8, 6, 4, 3$. Of these, 3 is the lowest.
>
> **Step 3**: 3 is adjacent to 7, so add $\mathbf{7}$ to the Prüfer sequence.
>
> **Step 4**: Removing node 3 leaves the following tree:



At this stage, the Prüfer sequence is $(\mathbf{1}, \mathbf{7})$.

Iteration 3

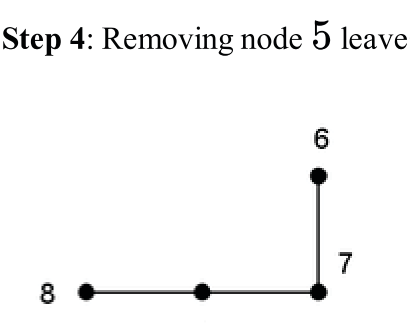
> **Step 1**: There are 6 nodes, so continue to **step 2**.
>
> **Step 2**: The nodes of degree 1 are $8, 6, 4$. Of these, 4 is the lowest.
>
> **Step 3**: 4 is adjacent to 5, so add $\mathbf{5}$ to the Prüfer sequence.
>
> **Step 4**: Removing node 4 leaves the following tree:



At this stage, the Prüfer sequence is $(\mathbf{1}, \mathbf{7}, \mathbf{5})$.

Iteration 4

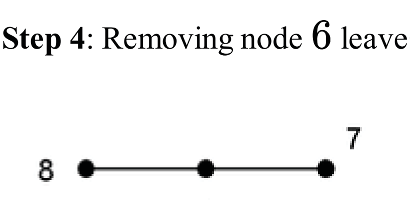
> **Step 1**: There are 5 nodes, so continue to **step 2**.
>
> **Step 2**: The nodes of degree 1 are $8, 6, 5$. Of these, 5 is the lowest.
>
> **Step 3**: 5 is adjacent to 7, so add $\mathbf{7}$ to the Prüfer sequence.
>
> **Step 4**: Removing node 5 leaves the following tree:



At this stage, the Prüfer sequence is $(\mathbf{1}, \mathbf{7}, \mathbf{5}, \mathbf{7})$.

Iteration 5

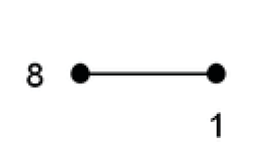
> **Step 1**: There are 4 nodes, so continue to **step 2**.
>
> **Step 2**: The nodes of degree 1 are $8, 6$. Of these, 6 is the lowest.
>
> **Step 3**: 6 is adjacent to 7, so add $\mathbf{7}$ to the Prüfer sequence.
>
> **Step 4**: Removing node 6 leaves the following tree:



At this stage, the Prüfer sequence is $(\mathbf{1}, \mathbf{7}, \mathbf{5}, \mathbf{7}, \mathbf{7})$

Iteration 6

> **Step 1**: There are 3 nodes, so continue to **step 2**.
>
> **Step 2**: The nodes of degree 1 are $8, 7$. Of these, 7 is the lowest.
>
> **Step 3**: 7 is adjacent to 1, so add $\mathbf{1}$ to the Prüfer sequence.
>
> **Step 4**: Removing node 7 leaves the following tree:



At this stage, the Prüfer sequence is $(\mathbf{1}, \mathbf{7}, \mathbf{5}, \mathbf{7}, \mathbf{7}, \mathbf{1})$

Iteration 7

> **Step 1**: There are 2 nodes, so **stop**.

The Prüfer sequence is $(\mathbf{1}, \mathbf{7}, \mathbf{5}, \mathbf{7}, \mathbf{7}, \mathbf{1})$

Labeled Tree from Prüfer Sequence

From ProofWiki

Contents

Algorithm

Given a Prüfer sequence, it is possible to generate a finite labeled tree corresponding to that sequence.

Let $P = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n-2})$ be a Prüfer sequence. This will be called **the sequence**.

It is assumed the sequence is not empty.

Step 1: Draw the n nodes of the tree we are to generate, and label them from 1 to n. This will be called **the tree**.

Step 2: Make a list of all the integers $(1, 2, \ldots, n)$. This will be called **the list**.

Step 3: If there are two numbers left in **the list**, connect them with an edge and then **stop**. Otherwise, continue on to **step 4**.

Step 4: Find the smallest number in **the list** which is not in **the sequence**, and also the first number in **the sequence**. Add an edge to **the tree** connecting the nodes whose labels correspond to those numbers.

Step 5: Delete the first of those numbers from **the list** and the second from **the sequence**. This leaves a smaller **list** and a shorter **sequence**. Then return to **step 3**.

The above constitutes an algorithm, for the following reasons:

Finiteness

For each iteration through the algorithm, **step 5** is executed, which reduces the size of **the list** by 1.

Therefore, after $n - 2$ iterations, at **step 1** there will be 2 numbers left in **the list**, and the algorithm will stop.

Definiteness

Steps 1 and 2: Trivially definite.

Step 3: We are starting with a non-empty Prüfer sequence of length $n - 2$, so **the list** must originally contain at least 3 elements. As the number of elements in **the list** decreases by 1 each iteration (see **step 5**), eventually there is bound to be just two elements in **the list**.

Step 4: As there are more elements in **the list** than there are in the **sequence**, by the Pigeonhole Principle there has to be at least one number in **the list** that is not in **the sequence**.

Step 5: Trivially definite.

Inputs

The input to this algorithm is the Prüfer sequence $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{n-2})$.

Outputs

The output to this algorithm is the tree T.

The fact that T is in fact a tree follows from the fact that:

- T has n nodes and (from the method of construction) $n - 1$ edges;
- Each new edge connects two as yet unconnected parts of T, so every edge is a bridge. Therefore there are no cycles in T, from Condition for Edge to be Bridge.

So T is a tree from Equivalent Definitions for Finite Tree.

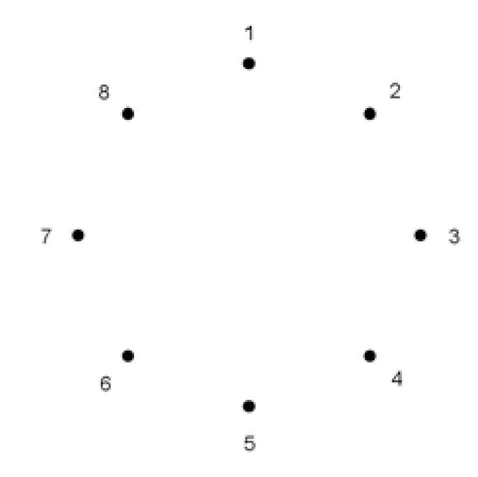
Effective

Each step of the algorithm is basic enough to be done exactly and in a finite length of time.

Example

Let the starting Prüfer sequence be $(\mathbf{1}, \mathbf{7}, \mathbf{5}, \mathbf{7}, \mathbf{7}, \mathbf{1})$

Step 1: **The sequence** is length 6, so the tree will have 8 nodes:

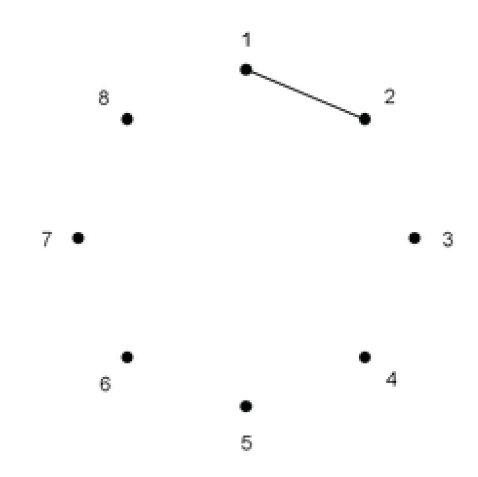


Step 2: We generate **the list**: $(1, 2, 3, 4, 5, 6, 7, 8)$

Iteration 1

Step 3: There are 8 elements in **the list**, so we move on to **step 4**.

Step 4: The smallest number in **the list** which is not in **the sequence** is 2, and the first number in **the sequence** is 1. We join 1 and 2, to obtain this graph:

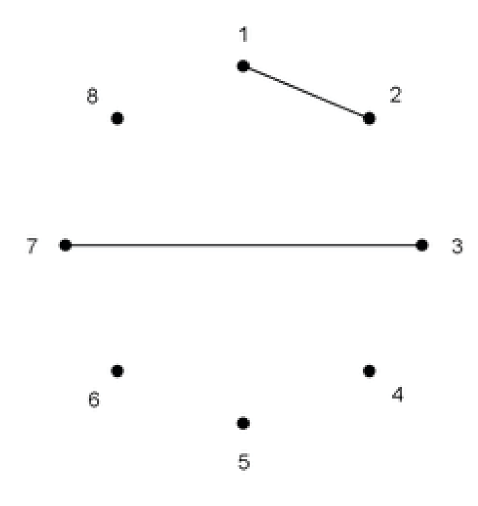


Step 5: We delete 2 from **the list** to obtain $(1, 3, 4, 5, 6, 7, 8)$ and 1 from the start of **the sequence** to obtain $(\mathbf{7}, \mathbf{5}, \mathbf{7}, \mathbf{7}, \mathbf{1})$

Iteration 2

Step 3: There are 7 elements in **the list**, so we move on to **step 4**.

Step 4: The smallest number in **the list** which is not in **the sequence** is 3, and the first number in **the sequence** is 7. We join 3 and 7, to obtain this graph:

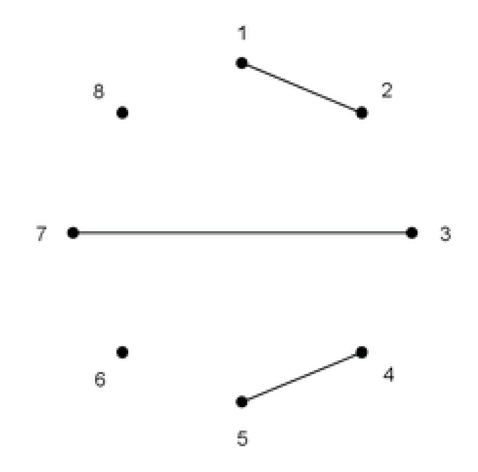


Step 5: We delete 3 from **the list** to obtain $(1, 4, 5, 6, 7, 8)$ and 7 from the start of **the sequence** to obtain $(\mathbf{5}, \mathbf{7}, \mathbf{7}, \mathbf{1})$

Iteration 3

Step 3: There are 6 elements in **the list**, so we move on to **step 4**.

Step 4: The smallest number in **the list** which is not in **the sequence** is 4, and the first number in **the sequence** is 5. We join 4 and 5, to obtain this graph:

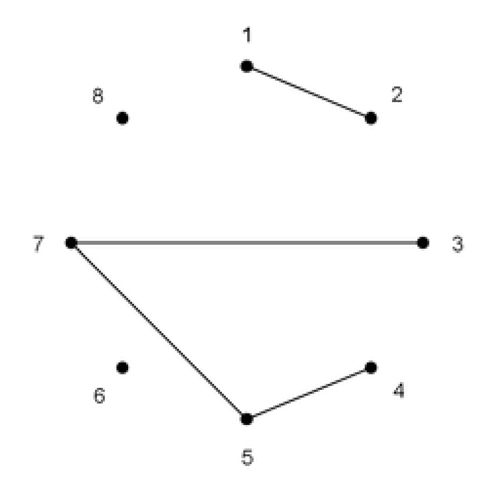


Step 5: We delete 4 from **the list** to obtain $(1, 5, 6, 7, 8)$ and 5 from the start of **the sequence** to obtain $(\mathbf{7}, \mathbf{7}, \mathbf{1})$.

Iteration 4

Step 3: There are 5 elements in **the list**, so we move on to **step 4**.

Step 4: The smallest number in **the list** which is not in **the sequence** is 5, and the first number in **the sequence** is 7. We join 5 and 7, to obtain this graph:

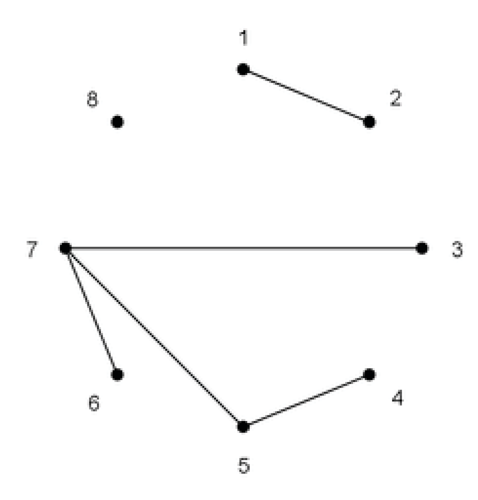


Step 5: We delete 5 from **the list** to obtain $(1, 6, 7, 8)$, and 7 from the start of **the sequence** to obtain $(\mathbf{7}, \mathbf{1})$.

Iteration 5

Step 3: There are 4 elements in **the list**, so we move on to **step 4**.

Step 4: The smallest number in **the list** which is not in **the sequence** is 6, and the first number in **the sequence** is 7. We join 6 and 7, to obtain this graph:

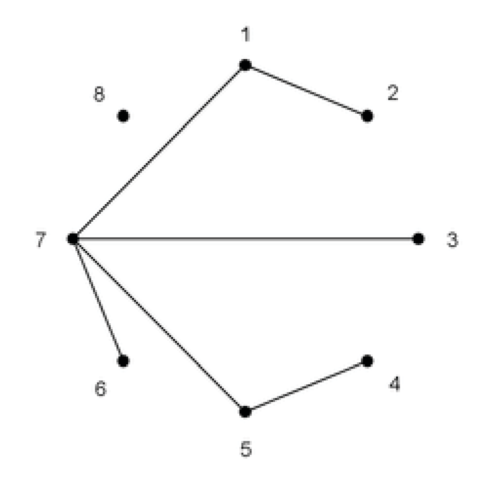


Step 5: We delete 6 from **the list** to obtain $(1, 7, 8)$, and 7 from the start of **the sequence** to obtain $(\mathbf{1})$.

Iteration 6

Step 3: There are 3 elements in **the list**, so we move on to **step 4**.

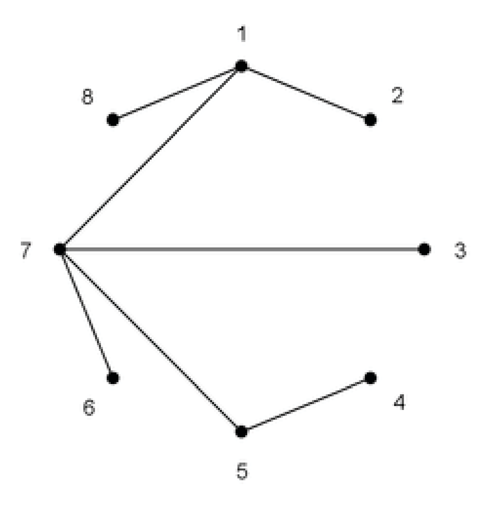
Step 4: The smallest number in **the list** which is not in **the sequence** is 7, and the first number in **the sequence** is 1. We join 7 and 1, to obtain this graph:



Step 5: We delete 7 from **the list** to obtain $(1, 8)$, and 1 from the start of **the sequence**, which is at this point empty.

Iteration 7

Step 3: There are 2 elements in **the list**: $(1, 8)$, so we join them to obtain this graph:



Then we **stop**.

The algorithm has terminated, and the tree is complete.

Rearranging the positions of the nodes, we can draw it like this: