Tree Representation Using Arrays

Iwa Ovyawan Herlistiono[1]

[1]Informatics Department, Engineering Faculty Universitas Widyatama, Bandung
ovyawan.herlistiono@widyatama.ac.id
Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 20 April 2021

Abstract:Tree as a non-linear data structure is very useful in the field of informatics. This paper discusses the technique of storing tree data into arrays. The trees discussed are the Binary Tree and General Tree types.

Keywords: Data Structure, Tree, Binary Tree, Prufer Number, Sequential encode, array.

1. Introduction

A tree is a set of one or more vertices, where one vertex is defined as the root and the rest can be divided into a number of non-empty sets, and each of these sets is a sub-tree of the root.

The formal definition of a tree is:
Given a Graph G = {V, E}.
with
V is the set of Vertices, and
E is the set of edges.
Thus, a Tree T is a subset of Graph G, which is connected, undirected, and does not contain cycles.

2. Theory

In the discussion about trees, there are a number of terms that must be known and will be discussed briefly.

Root vertex: Root R is a vertex that is at the very top in a tree. If R contains NULL, then the tree is empty.

Sub-trees: If the root vertex R does not equal NULL, then the T1, T2, and T3 trees are called sub-trees of root R.

Leaf vertex: A vertex A is called a leaf vertex if and only if it has no more vertices below it (terminal vertex).

Path: (also called walk) is a sequence of edge tracing from vertex A to vertex n.

Ancestor vertex: A vertex A is called an Ancestor if there is a vertex predecessor in the path from root to vertex A. Root R has no ancestor.

Descendant vertex: Conversely, a vertex A is called a descendant vertex, if there is a successor of vertices from vertex A in the path to the leaf. Leaf vertex has no descendant.

Level: Every vertex in Tree T, is always assigned a level such that root R is at Level 0. The successor of root is at Level 1. Every vertex is always at a higher level than its ancestor. All child vertices are at the +1 parent level.

Degree: The degree (degrees) of a vertex is equal to the number of children the vertex has. Thus, the degree of the leaf vertex is zero.

In-degree: The in-degree of a vertex V is the number of edges leading to vertex V. Whereas the Out-degree of a vertex V is the number of edges leaving vertex V. [1]

3. Scope of Problem

Given that there is such a broad discussion about trees, it is necessary to limit the scope of binary tree and general tree storage techniques with vertices labeled using arrays only.

4. Discussion

Trees are widely used in the field of Informatics, for example, storing hierarchical data, Huffman coding for data compression, analyzing chess games, building an artificial intelligence system and many more.

The following will discuss the tree data structure storage technique.

4.1. **Binary Tree and Sequential encode**.

Binary tree, is a subset of trees, with each node having only one in-degree and (maximum) two out-degrees. Figure 1, shows a complete binary tree.

Algorithm 1, is used to store data from the tree in Figure 1 into an array.

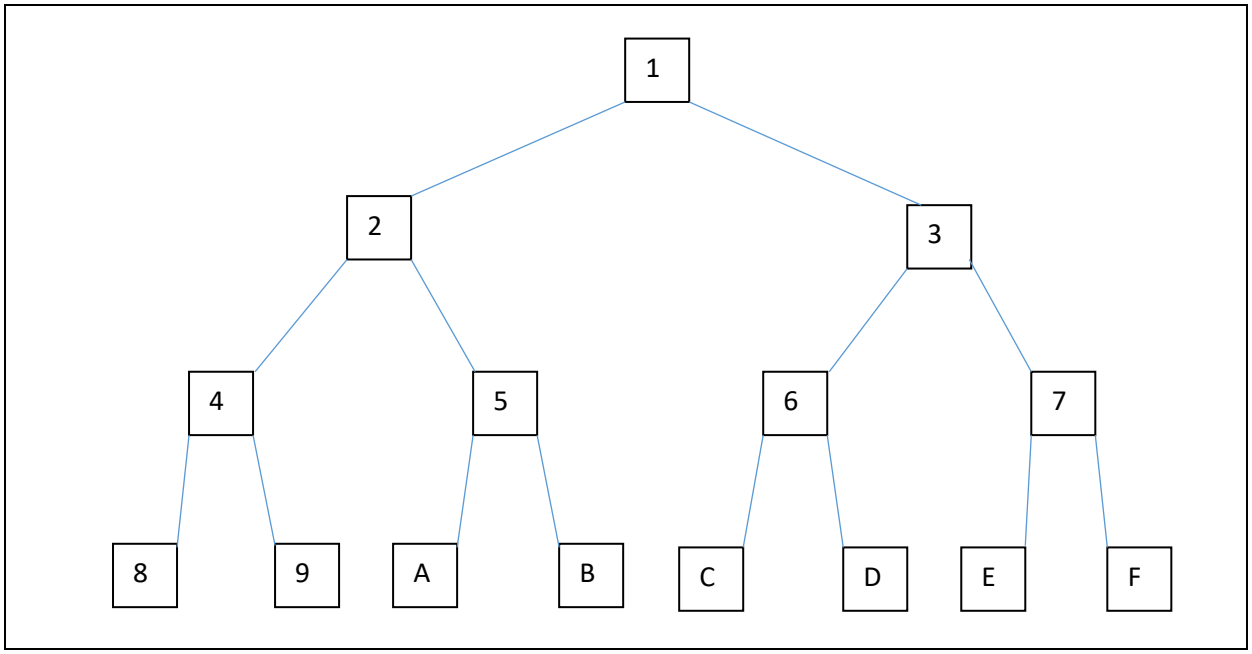

Fig 1. Complete Binary Tree.

```
encode1()
j = 0
untuk setiap level dalam tree:
  untuk setiap verteks di dalam level:
    tree[j] <- verteks
    inc(j)
end.
```

Algorithm 1. Encode Tree.

In the algorithm above, the Tree is visited level by level. At each level, the labels of the vertices are recorded in a one-dimensional array [0 .. n] sequentially. The results of recording the array against the tree in Figure 1, can be seen in Figure 2. As shown, for a tree with n vertices, n storage space is required as well. Recording is done efficiently and there is no waste of storage space. The tree hierarchy hasn't changed either. For example, the child of vertex i, is assigned to locations 2i and 2i + 1. Complexity The encode1 algorithm runs at O (n ^ 2).

1	2	3	4	5	6	7	8	*9*	A	B	C	D	E	F

Fig 2. encoded Binary Tree.

An encoding process will be useless if there is no way to recover the stored data (decoding process). The tree decode process is carried out using the information in the previous paragraph and returns the tree to its original state. Decode algorithm can be seen in Algorithm 2. [2] The decode_tree algorithm returns the tree to its initial state before it was encoded, and the tree hierarchy remains unchanged. The complexity of the decode_tree algorithm runs at O (n), which means that the algorithm is very good. Nothing is better than O (n).

```
decode1()
untuk i dari 1 sampai n:
  draw_edge(i, 2i)
  draw_edge(i, 2i + 1)
end.
```

Algorithm 2. Decoded Tree.

However, these encode and decode techniques are only good for a complete binary tree. If used on an uncomplete binary tree, as shown in Figure 3., this technique will cause problems. The problem is a waste of storage space. The tree in Figure 3., will be stored in the array shown in Figure 4.

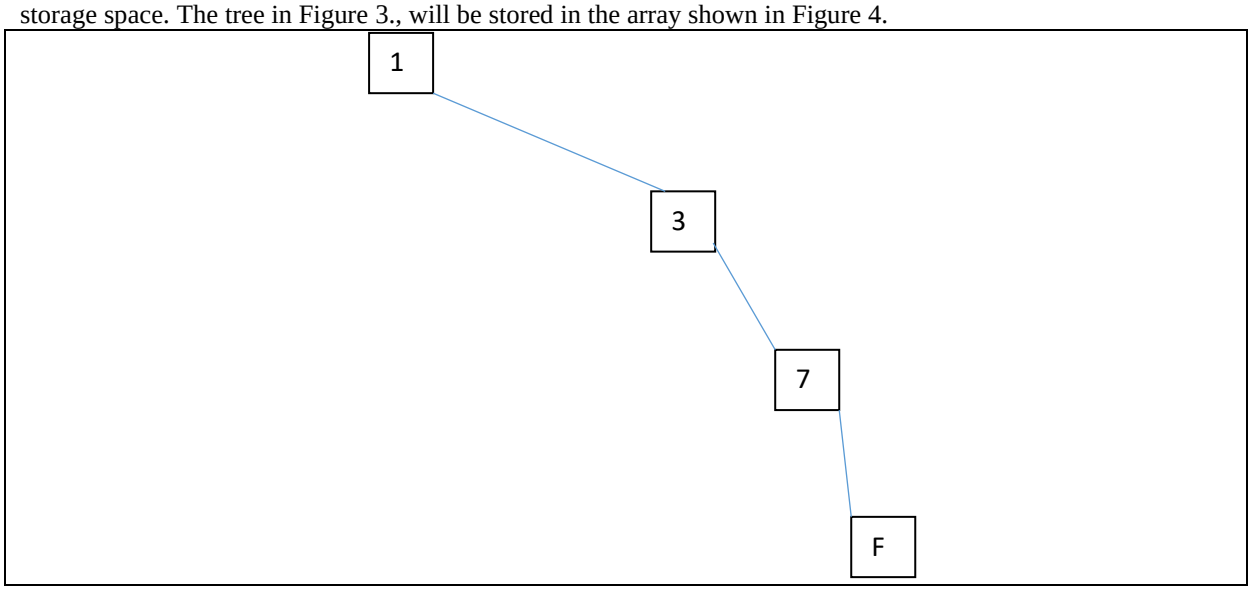

Fig 3. Uncompleted Binary Tree

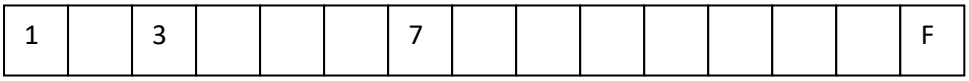

Fig 4. Array for Uncomplete Binary Tree in fig 3.

Seeing this problem, another more reliable algorithm is needed. The algorithm must be effective in decoding and must be efficient in terms of using storage space. In addition, it is too naive to expect that there will only be two choices for solving a problem. [2]

4.2. **Prüfer Number**.

Prüfer Number, is an encode and decode tree technique (cq General Tree) based on one of the Cayley theorems, which states that: "There are exactly n ^ (n-2) different trees, for a Graph G with n vertices.". [3] The theorem was proved by Prüfer in 1918 and derived by creating an algorithm to encode and decode a tree [4, 5].

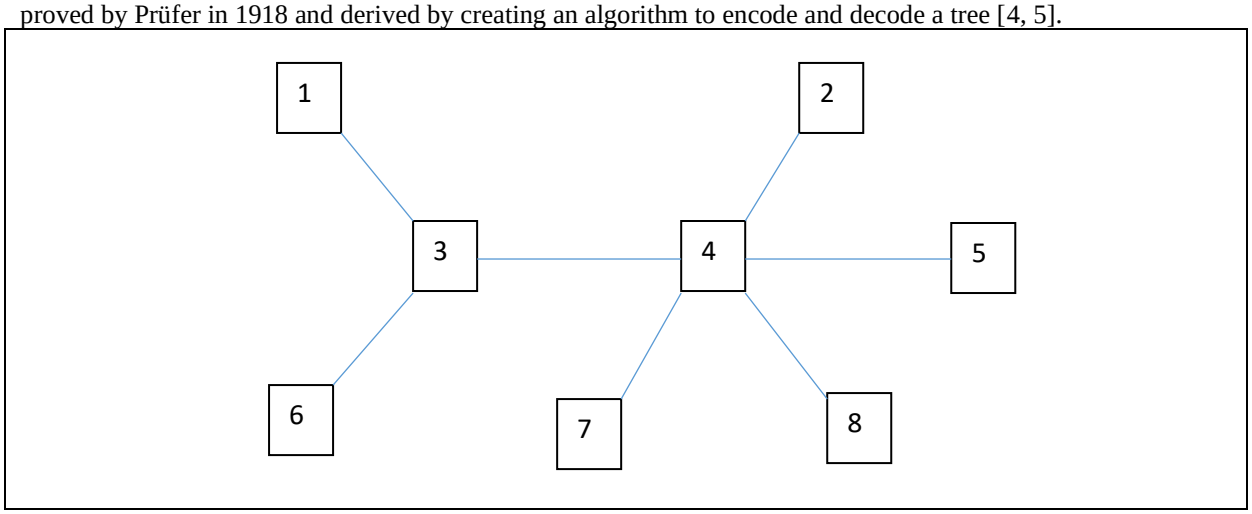

Fig 5. General Tree

The Prufer Number encoding algorithm is very simple and easy to follow. Consider the tree in Figure 5. The whole process is formulated into Algorithm 3.

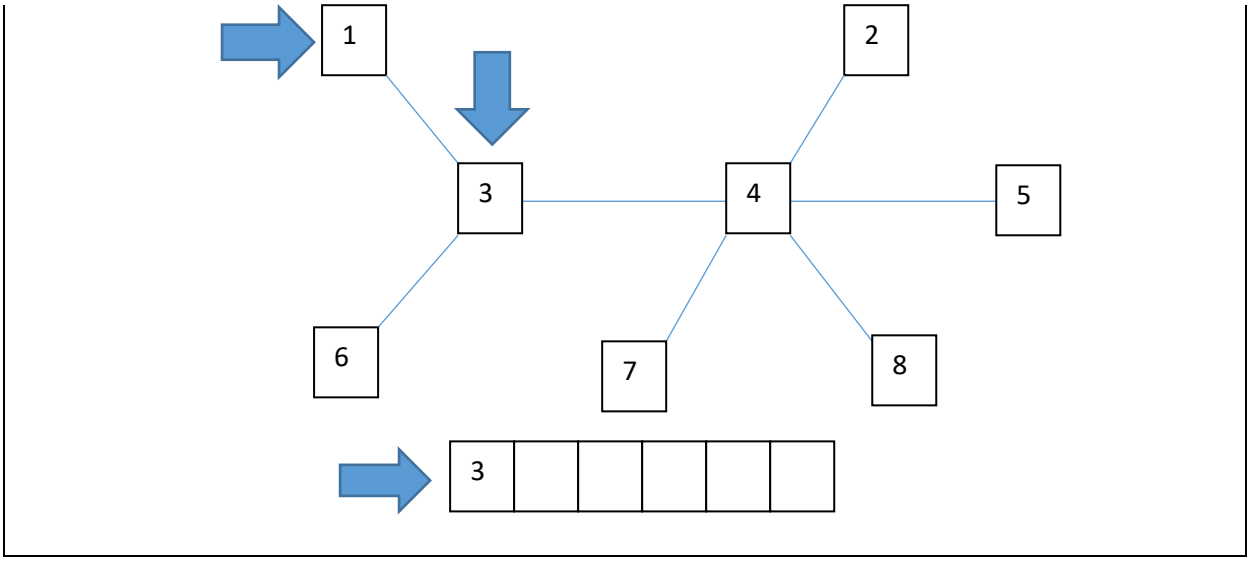

Fig 6. Prufer Number Encode.

The encoding process for storing a tree in the Prufer Number starts by finding the smallest leaf in the tree, which is vertex 1. Then find the parent, vertex 3. Then save the parent label as the first Prufer Number. Next, the edges between the smallest leaf and its parents are removed. Thus the leaf vertex is no longer part of the tree. The tree after the first step, changes to Figure 7.

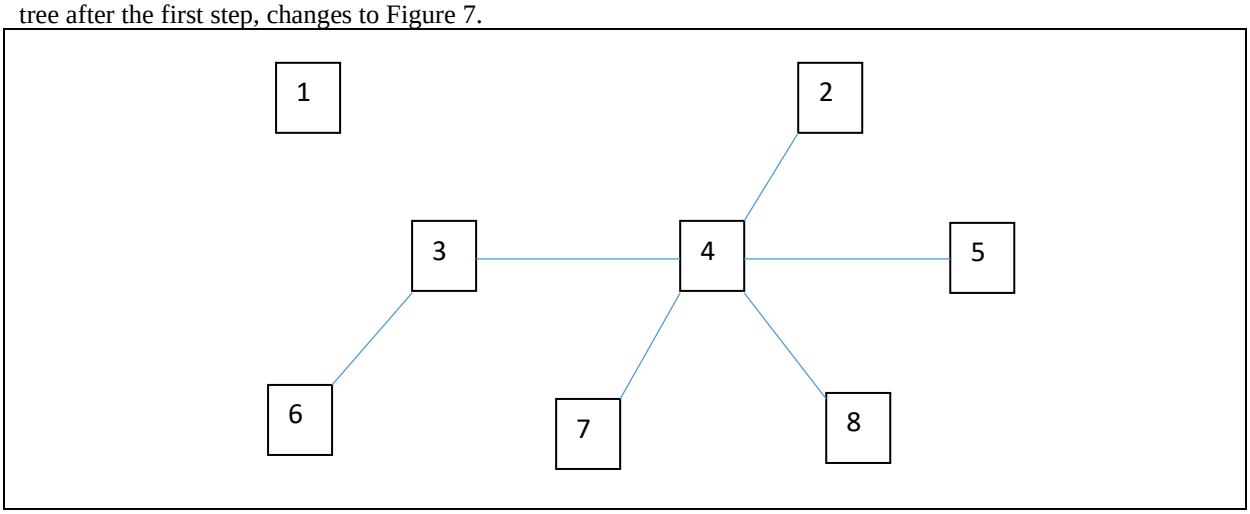

Fig 7. Tree after 1st iteration

Proses diulang, sekarang leaf terkecil adalah 2, dan parent-nya adalah 4. Tambahkan 4 pada Prufer Number, lalu hapus edge antara 2 dan 4. Vertex 2, seperti juga vertex 1, sekarang bukan lagi menjadi bagian dari tree.

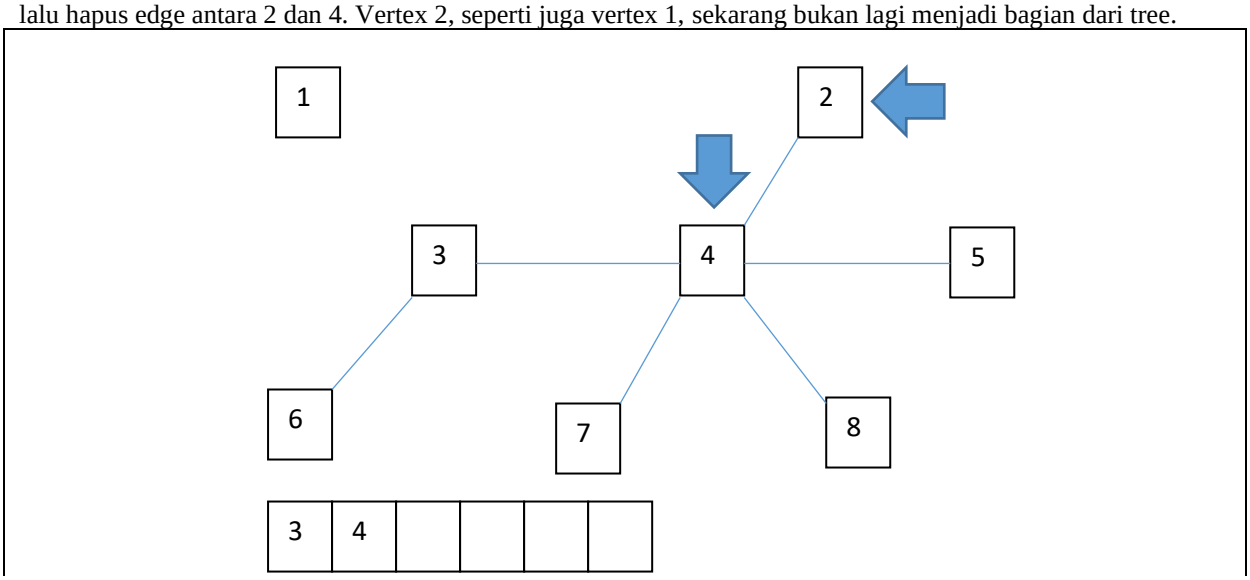

Fig 8. 2nd Iteration

The process is repeated, until there are two vertices and one edge left, Figure 9. As you can see, algorithm 3 runs on O (n), and takes n - 2 storage space, where n is the number of vertices in the tree.

```
encode2()
sampai tersisa 2 verteks:
  h <- leaf terkecil
  prufer[i] <- parent(h)
  hapus_edge(h, parent(h))
end.
```

Algorithm 3. Encode Prufer Number.

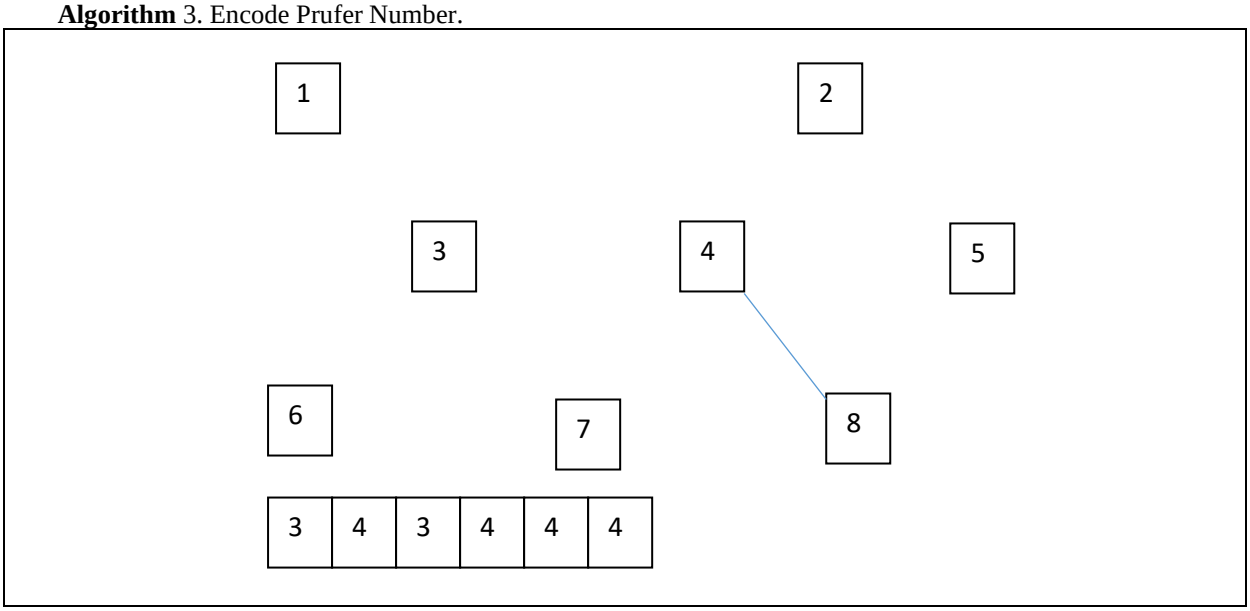

Fig 9. Final State Tree and Prufer Number.

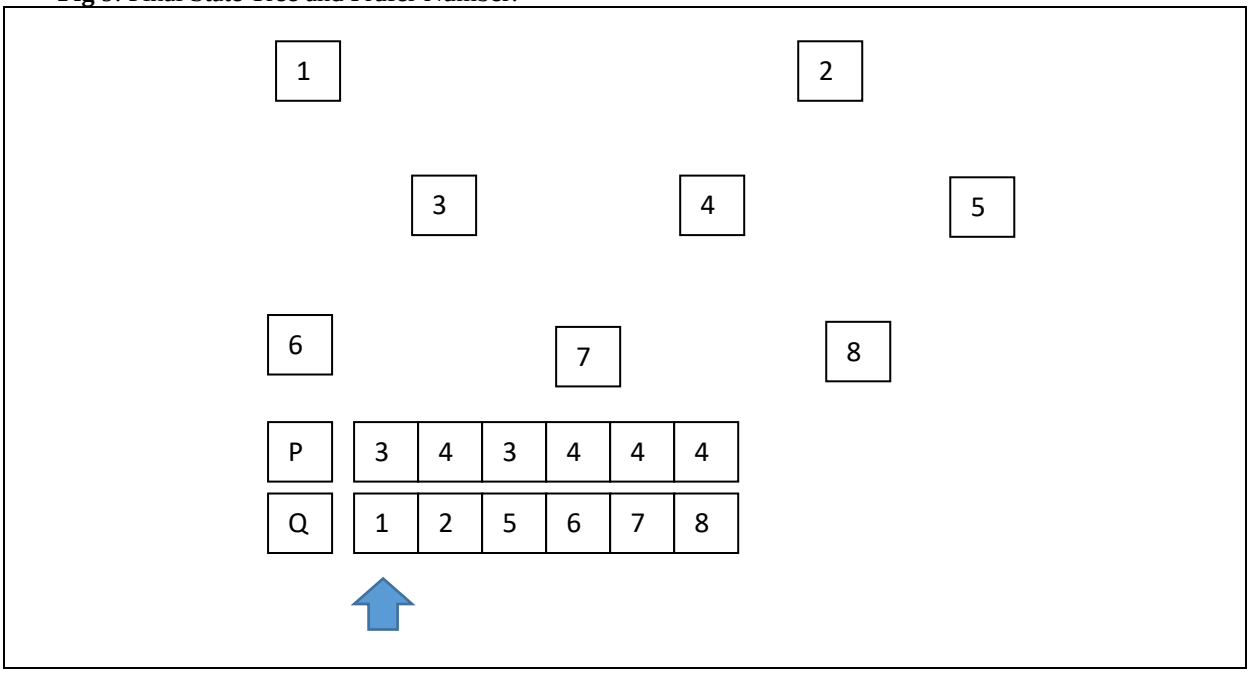

Fig 10. Encode Prufer Number.

The next step is to change the Prufer Number back to a tree. Take a look at Figure 10. The decoding process requires the original Prufer Number, P [i], and another list, Q [i], which contains all the vertices not in the original Prufer Number, ordered from smallest to largest.

Take the first Prufer Number and Q. Connect the two vertices. If P [1] is no longer present in the remainder of P, transfer P [i] into Q. Delete Q [1] and P [1]. The process is repeated until there are two vertices in Q. Connect the two, and the process stops. The whole process is formulated into Algorithm 4.

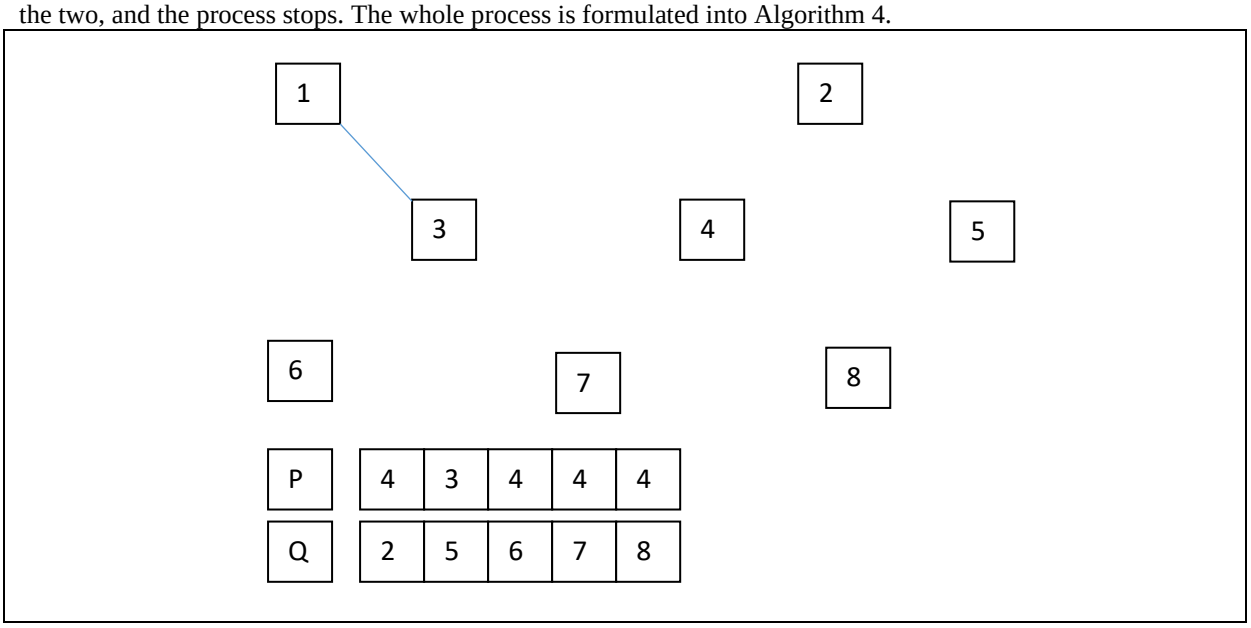

FIg 11. 1st iteration result

```
decode2()
p <- original prufer
q <- semua verteks yang tidak ada dalam p,
     diurut ascending.

untuk semua verteks dalam p:
   p1 <- p[1]
   q1 <- q[1] // q dengan nilai terkecil
   draw_edge(p1, q1)
   hapus(p[1])
   hapus(q[1])
   jika p1 tidak ada lagi dalam sisa p:
      1. tambahkan p1 pada q.
      2. urutkan kembali q.
akan tersisa tepat 2 buah verteks dalam q.
draw_edge(q[1], q[2]
end.
```

Algorithm 4. Decode Prufer Number.

The Decode algorithm runs on O (n) and requires 2n of storage space. This is due to the creation of an additional list (list Q) to accommodate all the vertices that were not in the original Prufer Number.

5. Conclusion

From the two types of tree data storage techniques that have been discussed, it can be concluded that the two techniques both have O (n) complexity in decoding. The Prufer Number technique at the time of encoding has the advantage in terms of using storage space over the sequential technique which has problems if the Binary Tree is incomplete. Conversely, when decoding, the Prufer Number Technique requires additional space, which is not needed in sequential techniques.

Reference

1. Thareja, Reema, "Data Structure Using C", 2nd Edition, Oxford University Press, 2014.
2. P., Insap, Santosa, "Struktur Data Menggunakan Turbo Pascal 6.0", Penerbit Andi Offset, Yogyakarta, 1992.
3. Jabarullah, N. H., Surendar, A., Arun, M., Siddiqi, A. F., & Krasnopevtseva, T. O. (2020). Microstructural Characterization and Unified Reliability Assessment of Aged Solder Joints in a PV Module. IEEE Transactions on Components, Packaging and Manufacturing Technology, 10(6), 1028-1034.
4. Caminiti, S., Finocchi, I., Petreschi, R., "On coding labeled trees". Theoretical Computer Science. 382(2): 97–108. doi:10.1016/j.tcs.2007.03.009, https://reader.elsevier.com/reader/sd/pii/S030439750700171 5?token=107889735D0DEAF9F16FB4EF415F9E18C4AC8FAA8582273252E145277FB79BA0A57C41AC8D91DE16B69CEFA012AF8163, tanggal akses: 01, Januari, 2021
5. Maake, B. M., & Tranos, Z. U. V. A. (2019). A SERENDIPITOUS RESEARCH PAPER RECOMMENDER SYSTEM. International Journal of Business and Management Studies, 11(1), 39-53.