Chapter 5

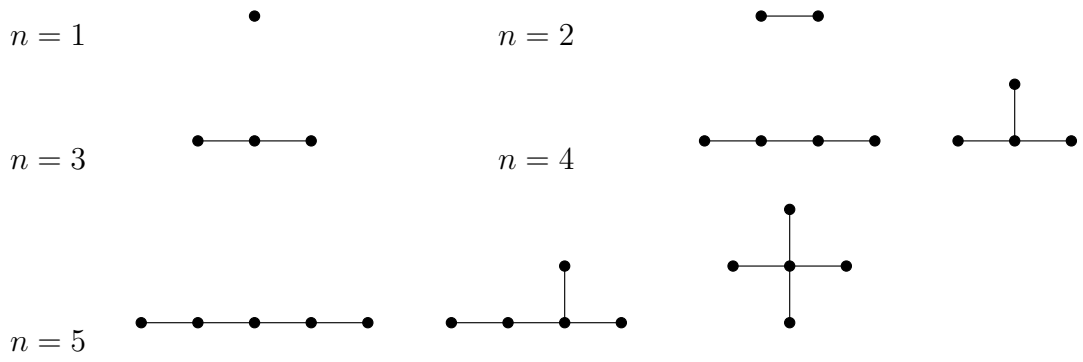
Trees

Trees are an important family of graphs; we have already encountered examples of them. They are of mathematical interest because of their simple structure, and they also play an important role in solving certain types of real-world problems.

5.1 Trees and Forests

Definition 5.1. A simple connected graph with n vertices and $n-1$ edges is called a *tree*. A *forest* is a disconnected graph in which each component is a tree.

Example 5.1. Nonisomorphic trees with n vertices:



n	1	2	3	4	5	6	7	8
number of trees	1	1	1	2	3	6	11	23

Characterisations of a Tree

There are several properties unique to trees, any one of which could be used to define the concept of a tree; what makes the most useful definition will vary according to the appplication. Here we list some useful properties of trees.

Theorem 5.1 (Properties of trees)**.** *Let T be a tree.*

1. *T contains no cycles.*

2. *If e is an edge, then removing e from T ($T - e$) leaves a disconnected graph. (Such an edge is called an* isthmus.*)*

3. *There is exactly one path between any two vertices.*

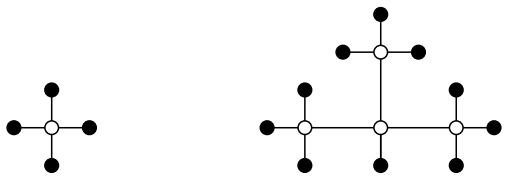
4. *Adding a new edge creates exactly one cycle.*

5. *T contains at least 2 vertices of degree 1 (provided $n \geq 2$). (Such vertices are known as* leaves.*)*

6. *T is bipartite (provided $n \geq 2$).*

Note: Any simple graph with $n - 1$ edges satisfying one of (1)-(4) must be a tree. There are other types of graphs satisfying (5) and (6).

Example 5.2 (organic chemistry)**.** An *alkane* is a molecule consisting of carbon and hydrogen atoms with each hydrogen atom bonded to a single carbon atom, and each carbon atom bonded to four (distinct) carbon or hydrogen atoms, such that the structure contains no cycles. This last requirement implies that the structure of an alkane can be represented by a tree:



∘ carbon, • hydrogen

methane isobutane

If an alkane contains C carbon atoms, what number of hydrogen atoms does it contain?
Solution. Let C represent the number of carbon atoms in the alkane, and H represent the number of hydrogen atoms. Since the corresponding graph is a tree, we know that it has $C + H - 1$ edges. Each carbon atom is represented by a vertex of the graph that has degree 4, and each hydrogen atom by a vertex of degree 1. Therefore, the sum of the degrees of the vertices is equal to $4C + H$, and thus, by the Handshaking Lemma, we have that

$$4C + H = 2(C + H - 1).$$

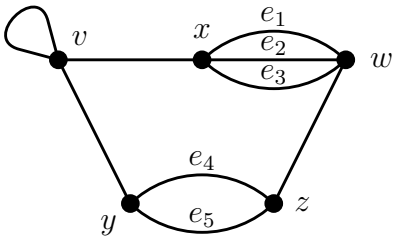
Hence, we conclude that $H = 2C + 2$.

5.2 Spanning Trees

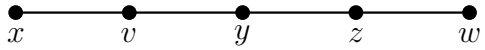
A tree is an example of a graph that is "only just connected," since any graph with the same vertices but fewer edges is disconnected. As a consequence, the solution to optimisation problems relating to connectivity often involve finding trees that are subgraphs of other graphs.

Definition 5.2. A *spanning subtree* of a graph G is a spanning subgraph of G which is also a tree.

Example 5.3.



G



spanning subtree of G

We will now study some applications of this concept.

5.2.1 Minimum Connector

There are many examples of real-world problems relating to connectivity that involve costs associated with providing connectivity.

Example 5.4. BBK oil has 5 sites A, B, C, D, E and wants to set up a system of pipelines so that oil can be pumped from one site to another (possibly via one or more other sites).

The distances (in km) between each of the sites are given in the following table:

	B	C	D	E
A	120	100	280	310
B	–	140	320	220
C	–	–	350	240
D	–	–	–	300

What is the shortest amount of pipeline required?

Example 5.5. A DIY company wants to link up its six superstores S_1, S_2, S_3, S_4, S_5, S_6 via a telecommunications network. The cost, in pounds, of linking two individual stores is given in the following table:

	S_2	S_3	S_4	S_5	S_6
S_1	30	45	X	35	25
S_2	–	30	25	40	30
S_3	–	–	30	35	X
S_4	–	–	–	40	30
S_5	–	–	–	–	40

(X=no connection possible)

What is the minimum cost of setting up the network?

It is often useful to express this type of problem in terms of a *weighted graph*.

Definition 5.3. A *weighted graph* is a graph in which each edge is assigned a numerical value, its *weight*.

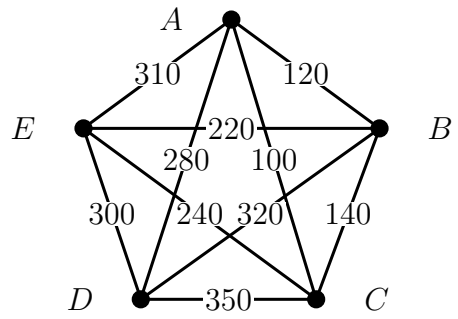
Definition 5.4. If G is a connected weighted graph, then a *minimum spanning tree* or *minimum connector* of G is a spanning tree, T, such that the sum of the weights of the edges of T is as small as possible.

As the name suggests, a minimum connector is the "cheapest" subgraph that connects all the vertices of G.

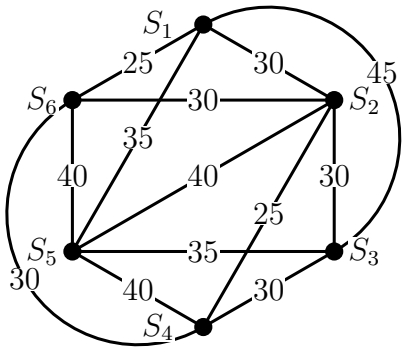
Notes:

1. When looking for a minimum spanning tree of a graph G we can assume that G is simple, since no tree has a loop, and a set of multiple edges $e_1, e_2, \ldots e_s$ can be replaced by a single edge e with weight equal to the minimum of the weights of $e_1, e_2, \ldots e_s$.

2. Examples 5.4 and 5.5 require a minimum spanning tree of the weighted graphs



and



respectively.

Kruskal's and Prim's Algorithms

Given a connected weighted graph G, how can we find a minimum spanning tree for G? One technique that we can use is known as *Kruskal's algorithm*:

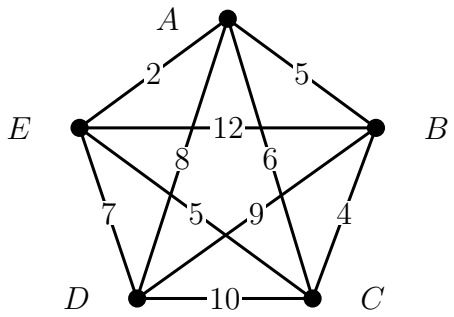
Definition 5.5 (Kruskal's Algorithm)**.** This algorithm finds a minimum spanning tree T for a connected weighted graph G. Start with $T = \emptyset$, then repeatedly add edges (together with the corresponding vertices) to T using the following steps, until it is no longer possible to add any more edges in this manner.

- Let e be an edge of smallest weight in G but not T with the property that $T \cup \{e\}$ does not contain a cycle (if there is more than one such edge of smallest weight it does not matter which one you choose).

- Add the edge e (together with the corresponding vertices) to the graph T.

Kruskal's algorithm always finds a spanning tree of G, and there is no spanning tree of G with a smaller total weight (although G may have more than one possible minimum spanning tree.)

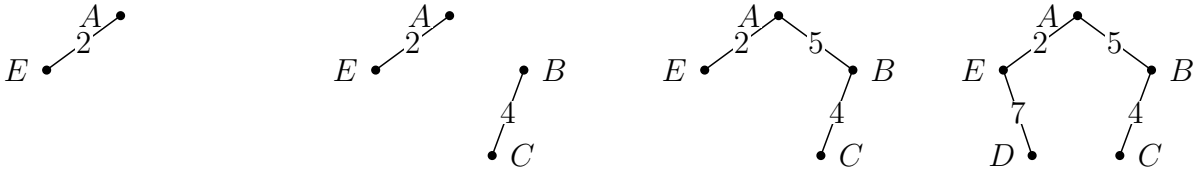
Example 5.6. Use Kruskal's algorithm to find a minimum spanning tree of the following graph:



The edges listed with nondecreasing weight are:

AE	BC	AB	CE	AC	DE	AD	BD	CD	BE
2	4	5	5	6	7	8	9	10	12

We begin by choosing the edge with smallest weight, AE. We then choose the edge with the smallest weight out of the remaining edges, which is BC. For the third edge we choose AB (although CE would also be an acceptable choice here). We do not yet have a spanning tree (vertex D is still isolated) so we must select another edge. Of the remaining edges of G, the edge CE has the least weight, but we cannot choose it, as it would create the cycle $ABCEA$. The edge with the next smallest weight is AC, but we cannot choose it, as it would create the cycle $ABCA$. Thus we choose edge DE, which completes our spanning tree. Adding any further edge at this point would create a cycle.



The total weight of the edges in this tree is $7 + 2 + 5 + 4 = 18$.

Kruskal's algorithm is an example of what is known as a *greedy algorithm:* at each stage we make the best choice currently available (in this case, we choose an edge of smallest possible weight) in the hope that the overall output will be the best possible. For many problems, the use of a greedy algorithm will not lead to the best possible solution, but Kruskal's algorithm is one example of a case in which this approach does work. To prove that Kruskal's algorithm works as claimed, it is necessary to show that it terminates after a finite number of steps, and that its output is indeed a minimal spanning tree of G.

Success of Kruskal's Algorithm. (correctness)
Suppose the graph G has n vertices. If Kruskal's algorithm is applied to G then it must terminate after at most $n-1$ edges have been added, since if T contains $n-1$ edges it is a tree (as it contains no cycles) and hence the addition of any extra edge would create a cycle.

We observe that:

- as T contains no cycles, then after each step of the algorithm it must be a forest;
- after the algorithm terminates, each vertex is part of T, since if a vertex v were not in T then it would be possible to include an edge incident with V in T without creating a cycle;
- after the algorithm terminates, T consists of a single component, for if it had more than one then it would be possible to add an extra edge joining the two componenets without creating a cycle.

These observations imply that after the algorithm terminates, T consists of a spanning tree for G.
(minimality)
We now show that the spanning tree T output by Kruskal's algorithm has a total edge weight that is less than or equal to that of any other spanning tree of G.

Let $S \neq T$ be a spanning tree of G, and let e be the edge of T with smallest weight that is not contained in S. Adding the edge e to S creates a cycle, by Theorem 5.1 part 4. In addition, this cycle contains some edge $e' \neq e$ that is not part of T (since T contains no cycles). Starting with S and replacing the edge e' by the edge e gives a new spanning tree S' that has one more edge in common with T than S does.

Now we want to show that the weight of e' is greater than or equal to that of e. If e' is added to T, it creates a cycle, by Theorem 5.1. As this cycle is not wholly contained in S, it follows that it contains some edge f that is not part of the tree S. At the point in which the edge f was selected by Kruskal's algorithm to be part of T the edge e' would also have been an option (since in the absence of f it does not create a cycle in the remaining vertices of T.) The fact that f was selected rather than e' implies that its weight must be less than or equal to that of e'. However, recall that e was the edge of *smallest* weight that was part of T but not S. This implies that the weight of e is less than or equal to that of f, and hence less than or equal to that of e', as required.

From this we deduce that the total edge weight of S' is less than or equal to that of S. By repeating this process we can transform S into T by a series of steps that each result in a new spanning tree whose total weight is less than or equal to that of the previous one, which indicates that the total weight of T is itself less than or equal to that of S. □

Exercise 5.7. Use Kruskal's algorithm to solve the problems presented in Examples 5.4 and 5.5.

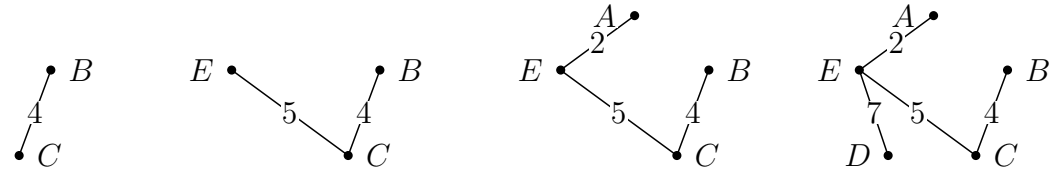
Another way in which we could compute a minimal spanning tree for a connected graph G is through the use of *Prim's algorithm.* This is another example of a greedy algorithm, which works slightly differently to Kruskal's algorithm.

Definition 5.6 (Prim's Algorithm)**.** This algorithm finds a minimum spanning tree T for a connected weighted graph G. Start with $T = \{v\}$ where v is any vertex of G, then repeatedly add edges to T using the following steps, until no more edges can be added in this manner.

- Let e be an edge of smallest weight in G that is incident with precisely one of the vertices of T (if there is more than one such edge of smallest weight it does not matter which one you choose).

- Add the edge e to the graph T.

Example 5.8. Use Prim's algorithm to find a minimum connector of the graph given in Example 5.6.
Solution. Start with the vertex B. The edge of lowest weight incident with B is BC. For the second edge, we choose CE (although AB would also be fine). Considering all edges incident with B, C, or E we see the one of smallest weight is AE. Finally, for the remaining edge we choose ED, since AB or AC would result in the creation of cycles.



(i) (ii) (iii) (iv)

In this case the total edge weight is $4 + 5 + 2 + 7 = 18$. This is a different spanning tree than we found in Example 5.6 using Kruskal's algorithm, yet it has the same (minimum) weight.

In general Prim's algorithm is faster than Kruskal's algorithm, although the speed does depend on the structure of G, the number of edges, and the distribution of the weights. From computer simulations, Prim's algorithm is generally found to be faster when the number of vertices is < 900.

Exercise 5.9. Use Prim's algorithm to solve the problems presented in Examples 5.4 and 5.5.

Exercise 5.10. Villages A, B, C, D and E are joined by roads whose lengths (in miles) are given in the following table:

	B	C	D	E
A	4	8	9	10
B	–	6	X	8
C	–	–	2	4
D	–	–	–	X

X= no road exists

After a large snowstorm, the council wants to grit enough roads to allow residents of any village to drive safely to any other village. What is the smallest number of miles of road that must be gritted?

Exercise 5.11. A gardener has five greenhouses and wishes to build paths so that he can reach them all without having to walk on the lawn. The cost of the gravel required to construct a path is £2 per metre. If the distances in metres between the greenhouses are as given in the following table, what is the minimum amount the gardener will have to spend?

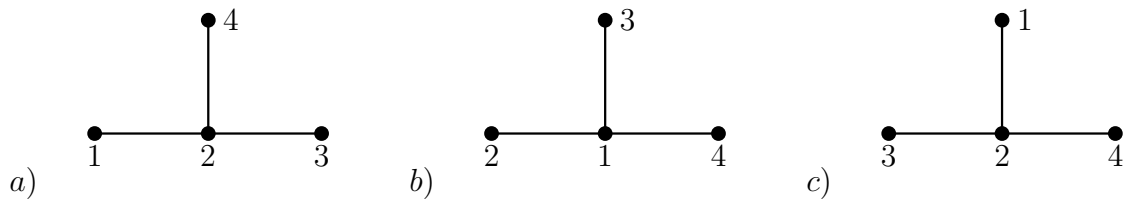
	gh2	gh3	gh4	gh5
gh1	13	14	12	8
gh2	-	3	10	13
gh3	-	-	8	11
gh4	-	-	-	5

5.3 Labelled Trees

Problems involving *graph enumeration* (that is, counting the number of graphs of a particular type with n vertices) have received much attention from mathematicians. For many classes of graphs this is a difficult problem; however, the corresponding problem of counting *labelled graphs* of a particular type with n vertices is frequently much easier to solve. In this section we will study the number of labelled trees on n vertices First, however, we must make precise what we mean by the concept of a labelled tree.

Definition 5.7. A *labelled tree* on n vertices is a tree in which the vertices are labelled by the numbers $1, 2, \ldots, n$. Two labelled trees T and T' are *isomorphic as labelled trees* if there is a graph isomorphism $\varphi : T \to T'$ such that for each vertex v of T, the label on $\varphi(v)$ is the same as that on v.

Example 5.12.



The above graphs are all examples of labelled trees. While the underlying trees are all isomorphic, we note that trees a) and b) are not isomorphic as labelled trees, since in a) the vertex of degree 3 is labelled 2 and in b) it is labelled 1. Similarly, b) and c) are not isomorphic as labelled trees. However, a) and c) are isomorphic as labelled trees, since we can make their labels correspond by simply rotating the leaves about the central vertex.

The number of labelled trees on n vertices is greater than the number of trees, since (as in the above example) there are distinct labelled trees based on the same underlying tree. We will now see a way in which we can determine the total number of labelled trees on n vertices.

Cayley's Theorem for Labelled Trees

The result that details the number of labelled trees on n vertices is known as *Cayley's formula*.

Theorem 5.2 (Cayley's Formula)**.** *The number of labelled trees on n vertices is n^{n-2}.*

Many different proofs of this result are known, which rely on quite varied techniques. Some examples can be found in *Proofs from the Book* by Martin Aigner and Günter M. Ziegler (4^{th} edition, Springer-Verlag, 2009).

Here we will see a proof due to Prüfer, that involves constructing a correspondence between labelled trees and a specific class of sequences.

Definition 5.8. A *Prüfer sequence* is an integer sequence $[a_1, a_2, \ldots, a_{n-2}]$ with $n-2$ elements satisfying $1 \leq a_i \leq n$ for $i = 1, 2, \ldots, n-2$.

Using the results of Chapter 1, we can see straightaway that the number of Prüfer sequences of length $n-2$ is equal to n^{n-2}. To provide a proof of Cayley's formula we will now construct a one-to-one correspondence between the labelled trees on n vertices and the Prüfer sequences of length $n-2$. We begin by showing how we can obtain a Prüfer sequence from a labelled tree in a unique manner.

Obtaining a Prüfer Sequence from a Labelled Tree

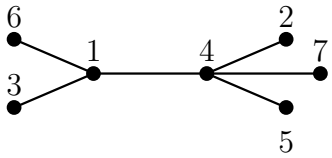
1. Find the leaf with the smallest label. The label of the vertex adjacent to this leaf is the first element of the sequence.

2. Delete the leaf and corresponding edge.

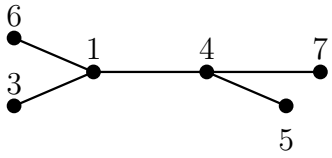
3. Repeat steps 1. and 2. to obtain subsequent elements of the sequence, until only one edge remains.

Since a labelled tree on n vertices has $n-1$ edges, the fact that a new element of the sequence is obtained each time an edge is deleted implies that the resulting sequence has $n-2$ elements. As each of these elements lie between 1 and n it is thus a Prüfer sequence.

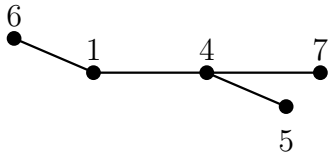
Example 5.13. Construct the Prüfer sequence of the following labelled tree:



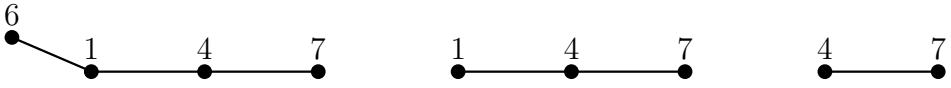
Solution. The leaf with the smallest label is 2, and this vertex is adjacent to vertex 4. Hence the sequence starts [4], and we remove the corresponding edge from the tree.



The leaf with the smallest label is now 3, and this vertex is adjacent to vertex 1. Hence the sequence starts [4, 1], and we remove the corresponding edge from the tree.



We continue in a similar manner:



$[4, 1, 4]$ $[4, 1, 4, 1]$ $[4, 1, 4, 1, 4]$

Hence the Prüfer sequence for this tree is $[4, 1, 4, 1, 4]$.

This correspondence between a labelled tree and a Prüfer sequence is a bijection: given a Prüfer sequence we can work back to construct a unique tree, and that sequence is the sequence we would have obtained had we started with the tree.

Constructing A Labelled Tree from a Prüfer Sequence

We now describe how to start from a Prüfer sequence and work back to obtain the corresponding tree, by adding in the edges in the same order that they would be deleted to construct the sequence.

1. Draw n vertices and assign the labels $1, 2, \ldots, n$ to them.

2. Make a list of the integers $1, 2, \ldots, n$ in order.

3. Find the smallest number i that is in the list but not in the sequence. If j is the first number in the sequence, construct an edge joining vertex i to vertex j.

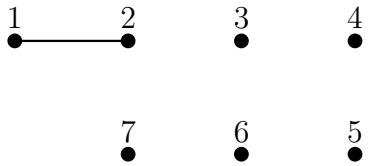
4. Remove i from the list, and remove the first element of the sequence.

5. Repeat steps 3. and 4. until the entire sequence has been deleted, and two numbers k and l remain in the sequence. Construct an edge joining vertices k and l.

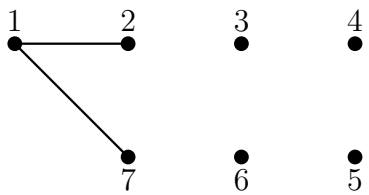
This process precisely reverses the method of obtaining a sequence from a labelled tree: if you start with a labelled tree and derive the corresponding sequence, then applying this process to the sequence will recover the original labelled tree. Thus we have a one-to-one correspondence between labelled trees and Prüfer sequences, and hence the number of labelled trees on n vertices must equal the number of Prüfer sequences of length n, and so Cayley's formula is proved.

Example 5.14. Draw the tree corresponding to the Prüfer sequence $[1, 7, 4, 3, 3]$.

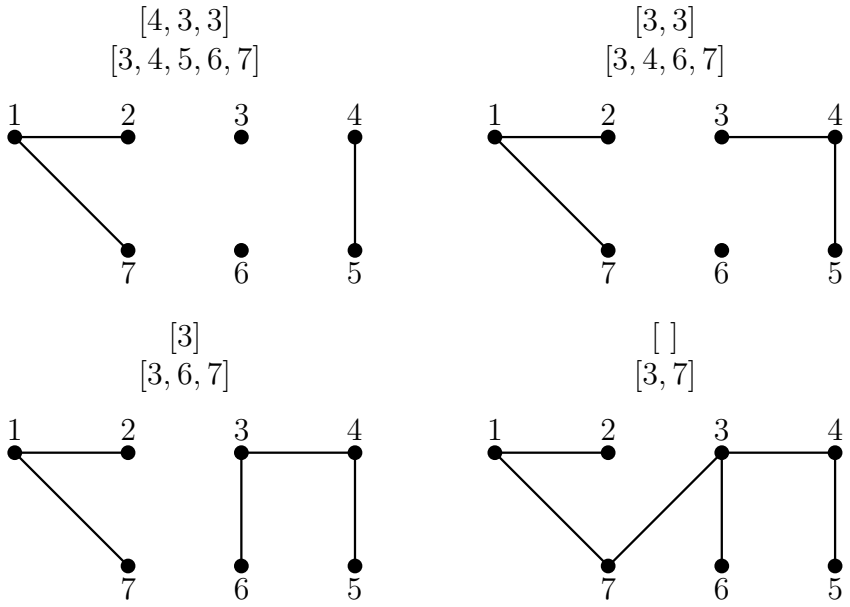
Solution. We start with the sequence $[1, 7, 4, 3, 3]$, and the list $[1, 2, 3, 4, 5, 6, 7]$. The smallest element of the list that does not appear in the sequence is 2, hence we construct the first edge of the tree:



We are left with the sequence $[7, 4, 3, 3]$, and the list $[1, 3, 4, 5, 6, 7]$. Now the smallest table element no longer appearing in the sequence is 1, and so we add the edge $\{1, 7\}$.



We continue in the same manner:



Exercise 5.15. Construct the trees corresponding to the following Prüfer sequences:

1. $[2, 3]$
2. $[5, 5, 5]$
3. $[4, 6, 1, 1]$

Exercise 5.16. Construct the tree corresponding to the Prüfer sequence obtained in Example 5.13, and find the Prüfer sequence of the tree constructed in Example 5.14 to check that the correspondence works as claimed.

Exercise 5.17. Show that you can use a Prüfer sequence to determine the degree of a vertex in a labelled tree without having to reconstruct the tree itself.

Learning Outcomes

After completing this chapter and the related problems you should be able to:

- Recognise and apply the characteristic properties of a tree.
- Model real-world problems using weighted graphs and recognise when such problems can be solved by finding a minimum spanning tree.
- Apply Kruskal's algorithm for determining a minimum spanning tree of a weighted graph.
- Apply Prim's algorithm for determining a minimum spanning tree of a weighted graph.
- Determine when two labelled trees are isomorphic.
- State Cayley's formula for the number of labelled trees.

- Construct the Prüfer sequence of a labelled tree.
- Construct the labelled tree corresponding to a given Prüfer sequence.
- Appreciate the fact that Cayley's formula can be determined by counting Prüfer sequences.