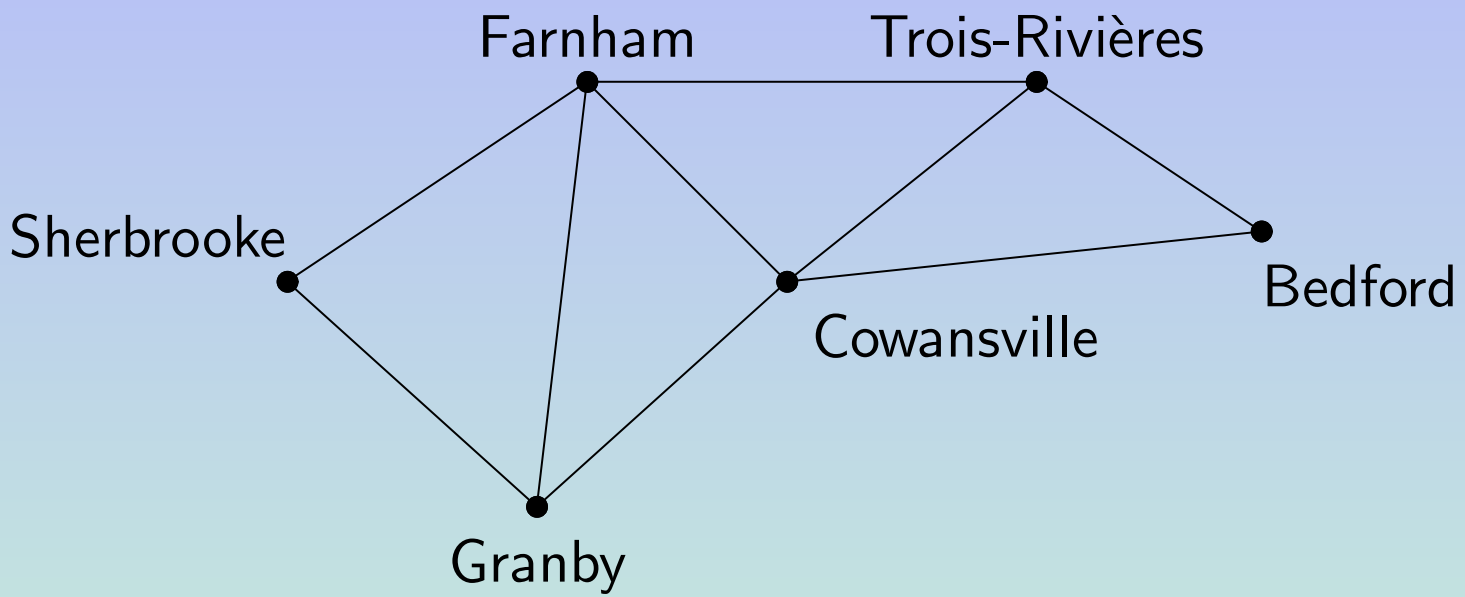
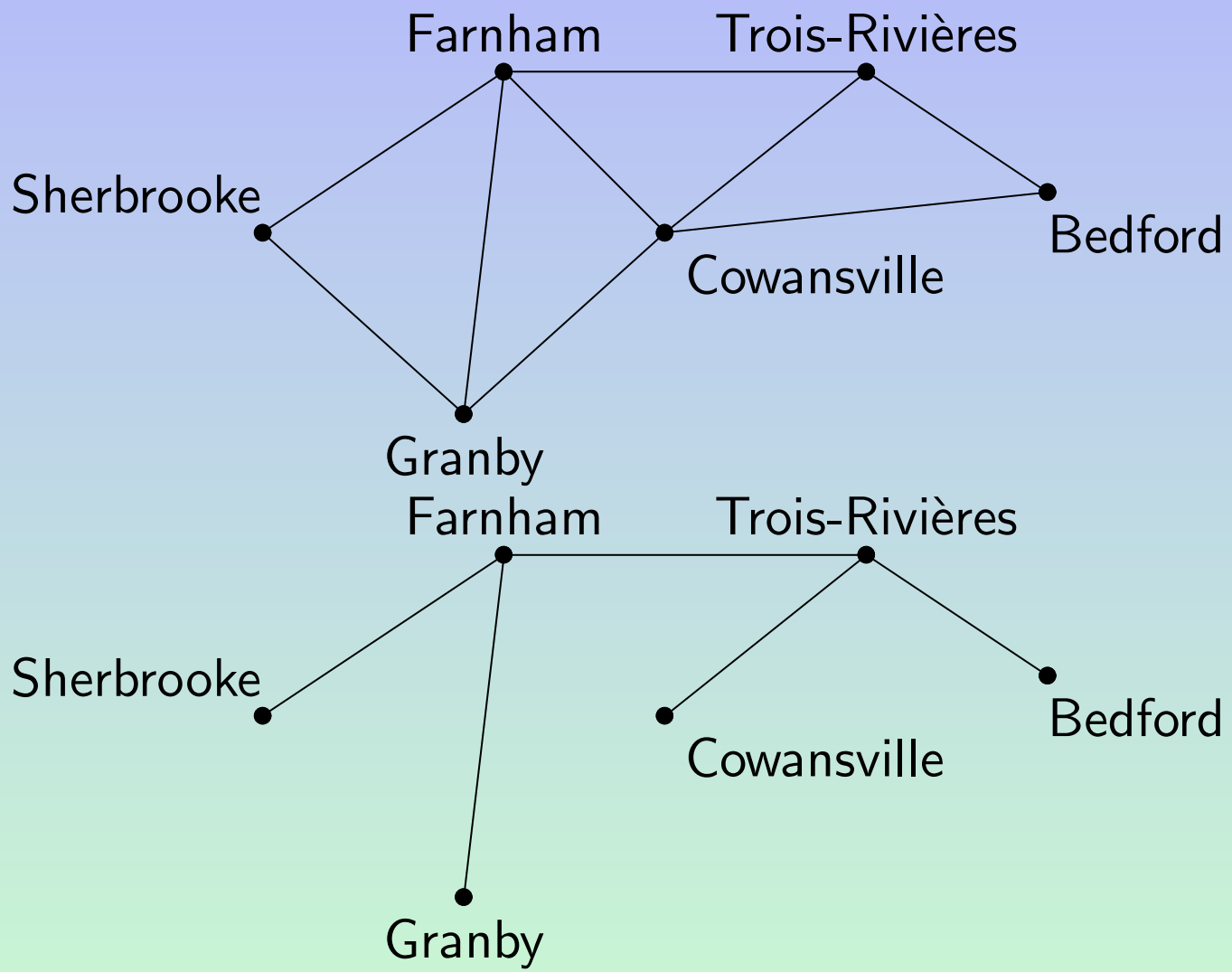
Spanning Trees
Discrete Mathematics

Example: Road System



In winter, the highway department wants to plow the fewest roads so that there will always be cleared roads connecting any two towns.

Spanning Tree



Spanning Tree

Definition

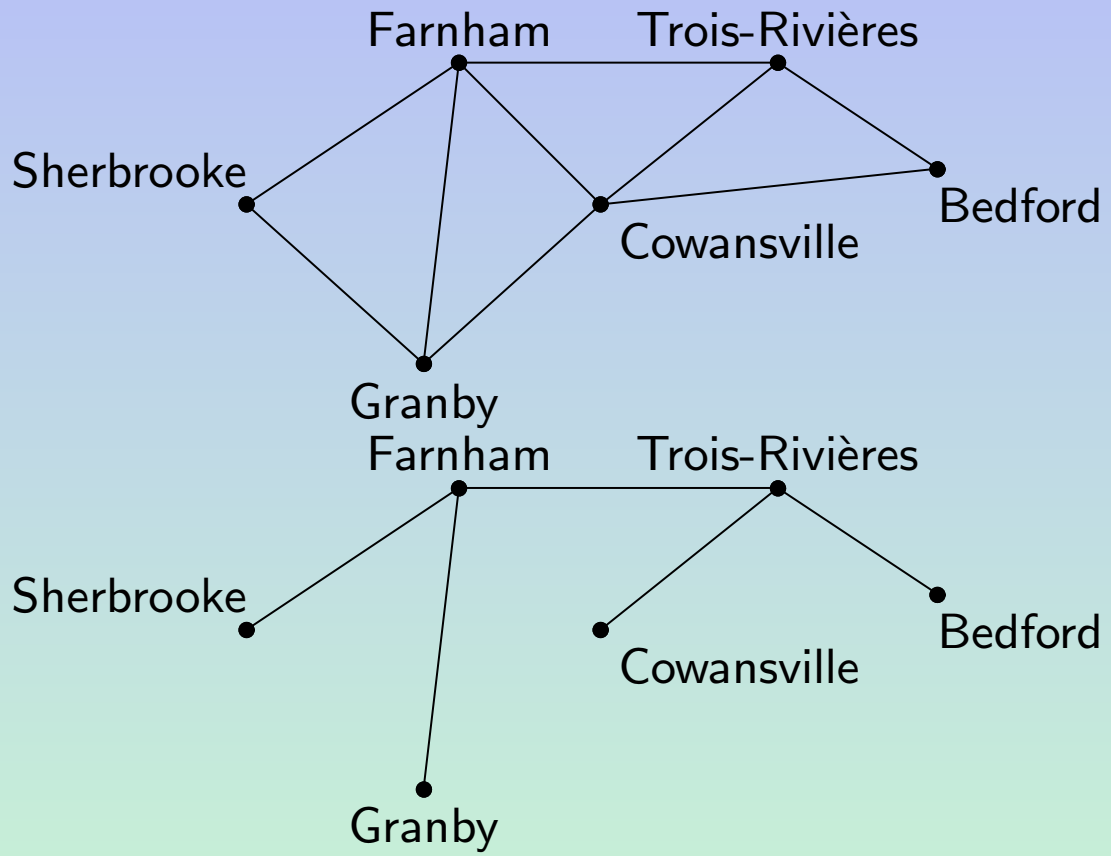
Let G be a simple graph. A **spanning tree** of G is a subgraph of G that is a tree containing every vertex of G.

Theorem

A simple graph is connected if and only if it has a spanning tree.

How to Build a Spanning Tree of a Graph

Remove edges to cut simple circuits until convergence.



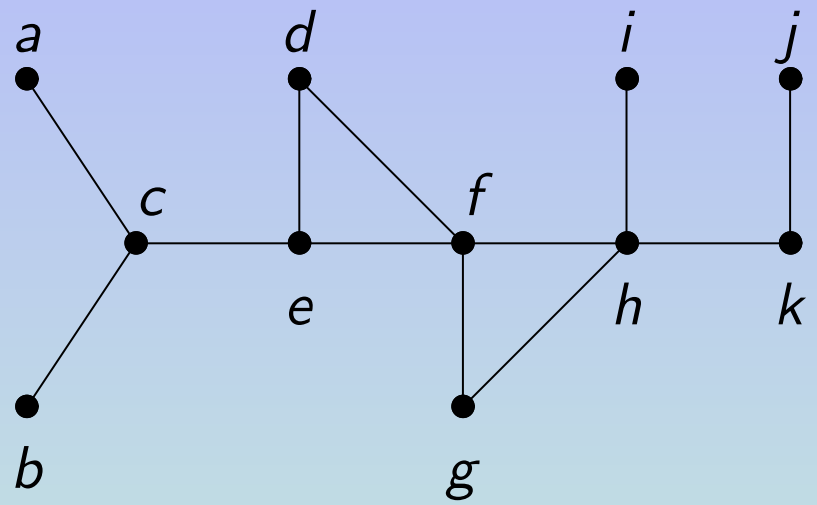
This algorithm is inefficient because it requires that simple circuits be identified.

Depth-First Search

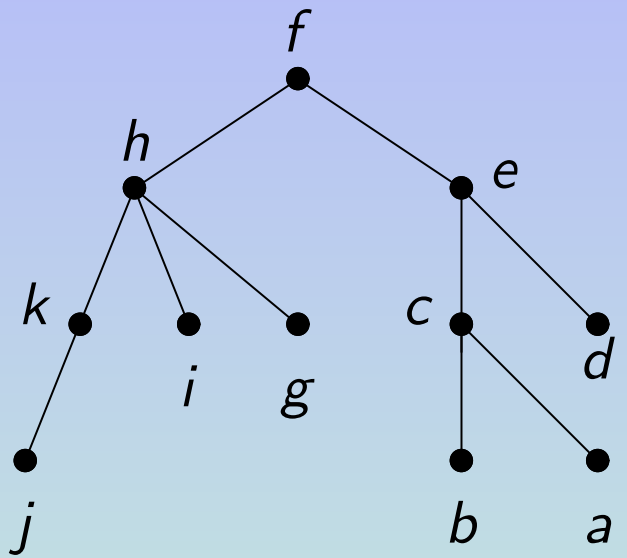
We can build a spanning tree for a connected simple graph using **depth-first search**. We will form a rooted tree, and the spanning tree will be will be the underlying undirected graph of this rooted tree.

The depth-first search starting at a given vertex calls the depth-first search of the neighbour vertices. If a vertex u has many neighbour vertices v_1, v_2, v_3, ..., the depth-first search at u calls the depth-first search at v_1, which calls the depth-first search of all neighbour of v_1, and so on recursively, and this before the depth-first search of v_2, v_3, ... The depth-first search is also called **backtracking**, because the algorithm returns to vertices previously visited to add paths.

Example of Depth-First Search



Grapg G



Spanning tree of G

We arbitrarily start with vertex f. A spanning tree is built by successively adding edges incident with vertices not already in the tree, as long as this is possible, then backtracking this path to add edges and vertices not already added to the tree.

Algorithm of Depth-First Search

```
procedure visit (v: a vertex of G)
for each vertex w adjacent to v
begin
    if w is not yet in T then
    begin
        add the vertex w and the edge {v, w} to T
        visit(w)
    end
end
```

Note: T is the spanning tree in construction.

Continued on next page...

Algorithm of Depth-First Search (cont.)

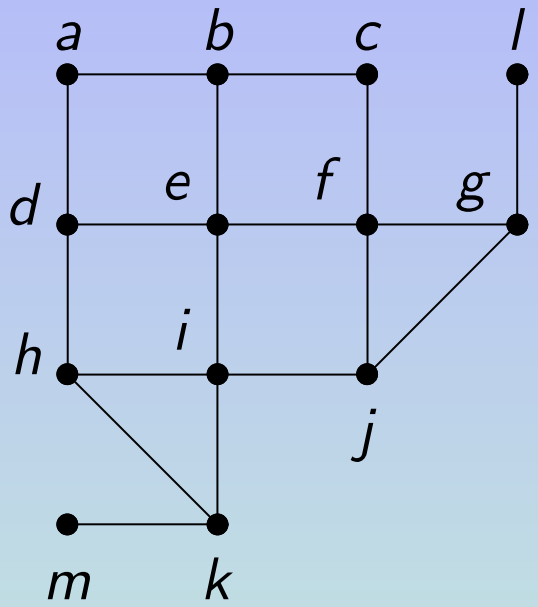
The *depth-first search* algorithm consists to do the recursive call to the algorithm *visit* with initial argument an arbitrarily vertex v_i chosen to be the root.

procedure *depth-first search*
(G: simple connected graph G with vertices $v_1, ..., v_n$)
$T :=$ an initial tree with v_i as root and no more vertices
visit(v_i)
$\{T$ is a spanning tree of $G\}$

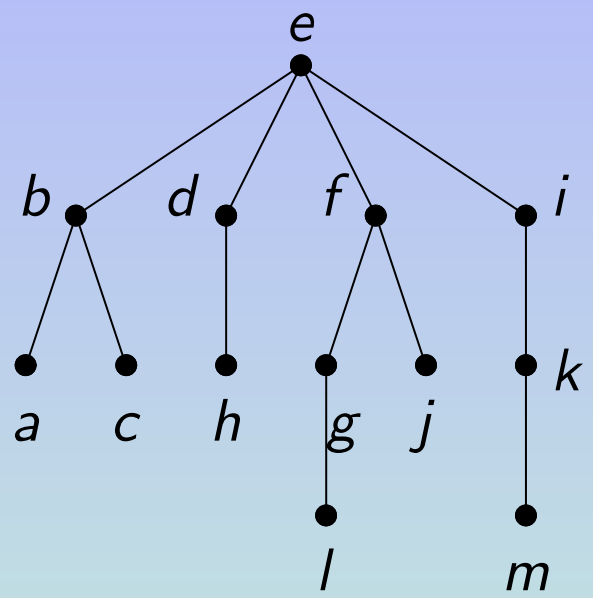
Breadth-First Search

First, a vertex of the graph G is arbitrarily chosen as the root of the spanning tree. Then all the neighbours of the root are added to the spanning tree, then all the neighbours of the neighbours of the root are added to the spanning tree, and so on until there is are more vertices of the graph G to add to the spanning tree.

Example of Breadth-First Search



Simple graph G



Spanning tree

The vertex e is arbitrarily chosen as root. The breadth-first search visits neighbour vertices b, d, f and i of e, then neighbours vertices a, c, h, g, j and k of the neighbours b, d, f and i, and finally the neighbours l and m of the neighbours of the neighbours.

Breadth-First Search Iterative Algorithm

procedure *Breadth-First Search*
(G: simple graph with vertices $v_1, ..., v_n$)
T := a tree initialized with v_i as root
old list := list initialized with the root v_i
new list := list initialized to the empty list

Continued on next page...

Breadth-First Search Iterative Algorithm (cont.)

while *old list* is not empty
begin
    remove the first vertex v from *old list*
    for each neighbour w of v
    begin
        if w is not in T **then**
        begin
            add w and the edge $\{v, w\}$ in T
            add w in *new list*
        end
    end
    old list := *new list*
    new list := $\emptyset$
end
$\{T$ is a spanning tree of $G\}$

Backtracking Applications

There are problems that can be solved only by performing an exhaustive search of all possible solutions.

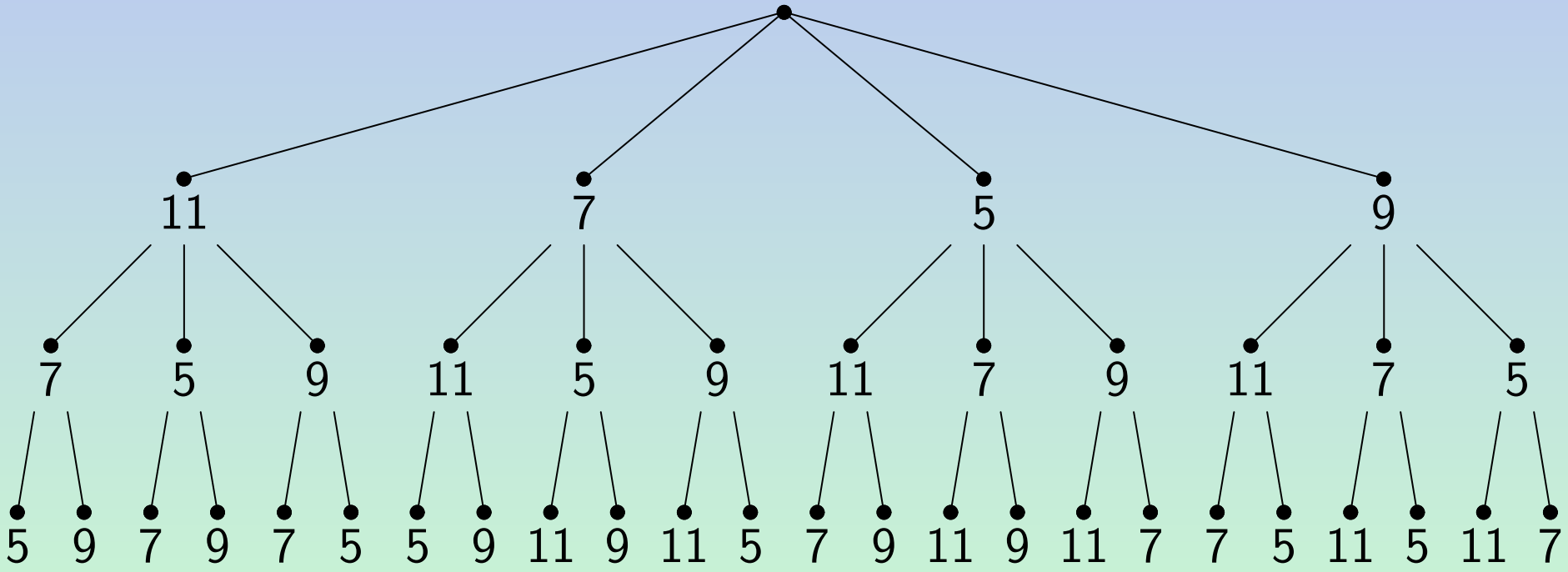
One way to to search systematically for a solution is to use a decision tree.

Each internal vertex represent a decision and each leaf a possible solution.

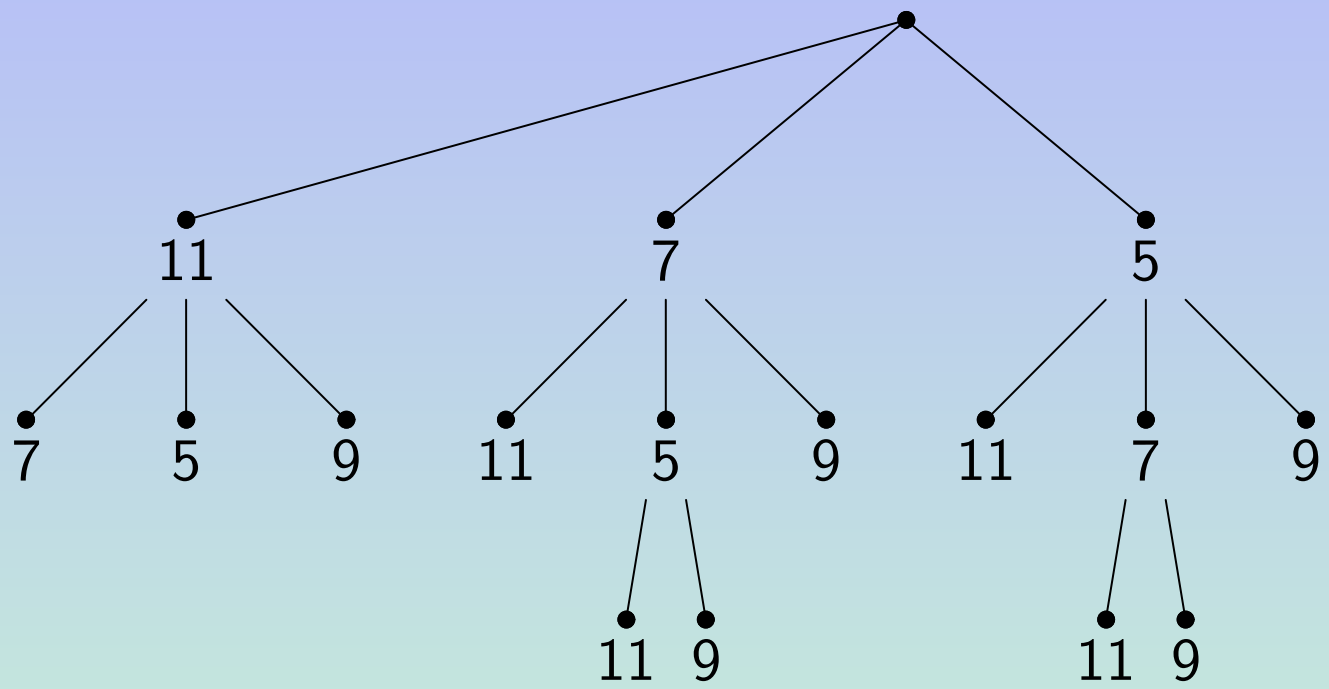
A problem may have many solutions. We perform a depth-first search of the decisions tree and stop when a solution is found. If no solution is found, as we looked at all possibilities, then we can assert than there is no solution.

Example of Backtracking

Problem: Find a subset of the set $\{11, 7, 5, 9\}$ such that the sum of the elements of the subset is 14. Here is the exhaustive list, as a decision tree, of all possible subsets (and solutions).



Example of Backtracking



This subtree, of the decision tree with all possible solutions, represent the depth-first search done until the first solution $\{5, 9\}$ were found.