Chapter 9 Trees

Discrete Mathematics and Its Applications
Zhejiang University/CS/Course/Xiaogang Jin
E-Mail: xiaogangj@cise.zju.edu.cn

9.1 Introduction to Trees

□ Trees were used as long ago as 1857, when English mathematician Arthur Cayley used them to count the types of chemical compounds.

□ Trees are applied in computer science:

– To construct efficient algorithms for locating items in a list.
– To construct networks with the least expensive set of telephone lines linking distributed computers.
– To construct efficient codes for storing and transmitting data.

□ THE DEFINITION OF TREES

Definition:

A **tree** T is a connected undirected graph with no simple circuit.

A **forest** is a undirected graph with no simple circuit.

Remark:

1) Since a tree cannot have a simple circuit, a tree cannot contain multiple edges or loop.

2) Each connected components of the forest is a tree.

Theorem: An undirected graph is a tree iff there is a unique simple path between any two of its vertices.

Proof: ($\Rightarrow$)

Assume that T is a tree. Then T is a connected graph with no simple circuits.

- T is connected $\Rightarrow$ $\exists$ a simple path between two vertices x and y.
- This path must be unique.

  If there were a second such path, the two paths formed would from a circuit $\Rightarrow$ there is a simple circuit in T.

  $\boxed{\text{contradiction!}}$

Hence, there is unique simple path between any two vertices of a tree.

$(\Leftarrow)$

Assume that there is unique simple path between any two vertices of a graph T.

- T is connected
- T have no simple circuit

Suppose that T had a simple circuit that contained the vertices x and y. Then there would be two simple paths between x and y.

Hence, a graph with a unique simple path between any vertices is a tree.

Definition: A particular vertex of a tree is designated as the **root**. Once we specify the root, we can assign a direction to each edge. We direct each edge away from the root. Thus, a tree together with its root produces a directed graph called a **rooted tree**.

- Parent, Child
- Siblings
- Ancestor, Descendant

□ THE TERMINOLOGY FOR TREES

Definition:

- **Leaf**: Vertices that have no children.
- **Internal vertices**: Vertices that have children.
- Let T be a rooted tree.
  - The **level** $l(v)$: the length of the simple path from root to v.
  - The **height** h of a rooted tree T:

$$h = \max_{v \in V(T)} l(v)$$

- **subtree**: root: a, the subgraph of the tree consisting of a and its descendants and all edges incident to these descendants.

Definition:(Continued)

• A rooted tree is a

– m-**ary tree**: every internal vertex has no more than m children. $m = 2$: **binary tree**.

– **full** m-**ary tree**: every internal vertex has exactly m children.

• **ordered rooted tree**: the children of each internal vertex are ordered.

ordered binary tree: left child, right child, left subtree, right subtree

• A rooted m-ary tree of height h is **balanced** if all leaves are at levels h or $h-1$.

□ PROPERTIES OF TREES

Theorem: A tree with n vertices has $n-1$ edges.

Proof:

– Choose the vertex r as the root of the tree.

– We set up a one-to-one correspondence between the edges and the vertices other than r by associating the terminal vertex of a edge to that edge.

– Since there are $n-1$ vertices other than r, there are $n-1$ edges in the tree.

Theorem: A full m-ary tree with i internal vertices contains $n = mi + 1$ vertices.

Proof:

– Every vertex, except the root, is a child of an internal vertices.

– Since each of the i internal vertices has m children, there are mi vertices in the tree other than the root.

Therefore, the tree contains $n = mi + 1$ vertices.

T be a full m-ary tree. Let i be the number of internal vertices and l be the number of leaves. Once one of n, i and l is known, the other two quantities are determined.

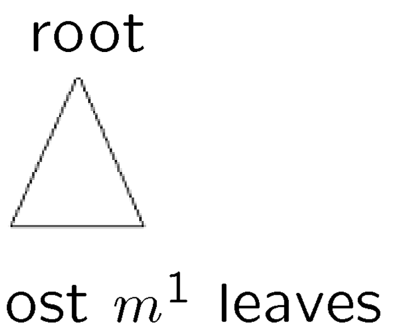
Theorem: A full m-ary tree with
(i) n vertices has $i = (n-1)/m$ internal vertices and $l = [(m-1)n+1]/m$ leaves,
(ii) i internal vertices has $n = mi+1$ vertices and $l = [(m-1)n+1]/m$ leaves,
(iii) l leaves has $n = (ml-1)/(m-1)$ vertices and $i = (l-1)/(m-1)$ internal vertices.

Note: $n = mi + 1$, $n = l + i$.

Theorem: There are at most m^h leaves in m-ary tree of height h.

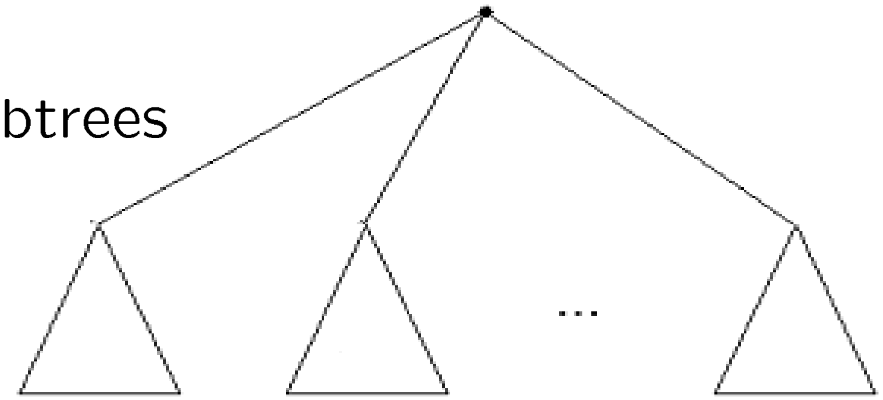
Proof: The proof uses mathematical induction on the height h.

Basis step: $h = 1$



Induction hypothesis:

at most m subtrees



every subtree at most m^{h-1} leaves

Hence, at most $m \cdot m^{h-1} = m^h$ leaves in the rooted tree.

Corollary: If an m-ary tree of height h has l leaves, then $h \geq \lceil \log_m l \rceil$. If the m-ary trees is full and balanced, then $h = \lceil \log_m l \rceil$.

Proof:

- $$l \leq m^h \text{ (by the Theorem)}$$
  $$\Rightarrow \log_m l \leq h$$
  $$\Rightarrow h \geq \lceil \log_m l \rceil \text{ (h is an integer).}$$

- Since the tree is balanced and its height is h, there is at least one leaf at level h. We have

$$m^{h-1} < l \leq m^h \Rightarrow h - 1 < \log_m l \leq h.$$

Hence, $h = \lceil \log_m l \rceil$.

9.2 Applications of Trees

□ INTRODUCTION

Problem:

• How should items in a list be stored so that an item can be easily located?

• What series of decisions should be made to find an object with a certain property in a collection of objects of a certain type?

• How should a set of characters be efficiently coded by bit string?

□ BINARY SEARCH TREES

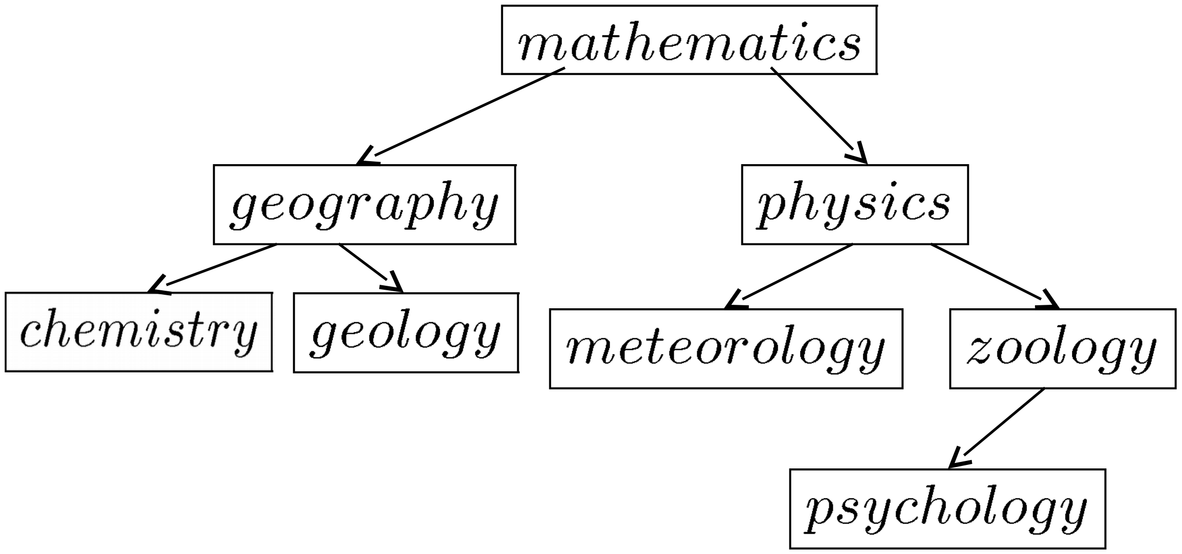
Goal: To implement a searching algorithm that finds a items efficiently when the items are totally ordered.

Definition: A **binary search tree** is a binary tree in which

1) each child of a vertex is designated as a right or left child;
2) no vertex has more than one right or left child;
3) each vertex is labeled with a key, which is one of the items;
4) vertices are assigned keys so that the key of vertex is both larger than the keys of all vertices in its left subtree and smaller than the keys of all vertices in its right subtree.

Example:

Form a binary search tree for the words *mathematics*, *physics*, *geography*, *zoology*, *meteorology*, *geology*, *psychology* and *chemistry*. (using alphabetical order).

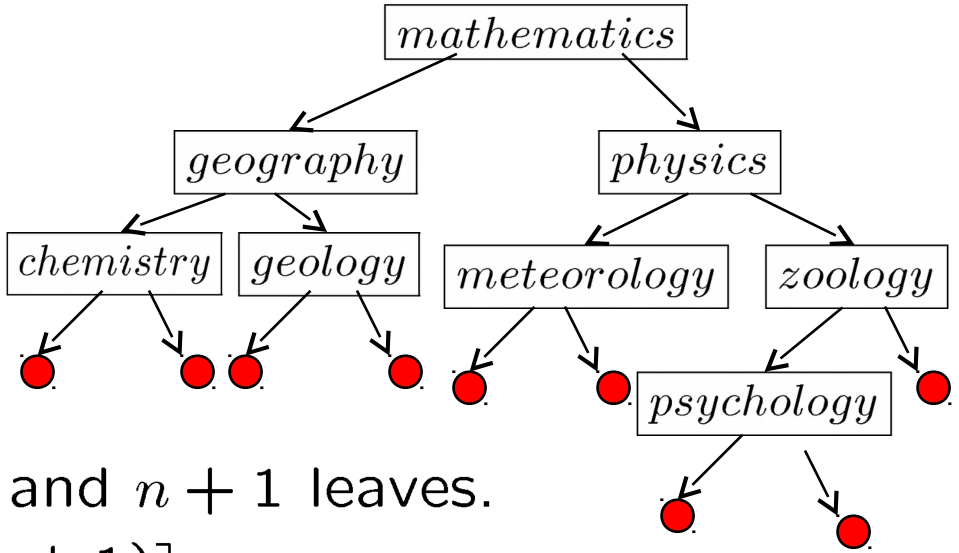


```
ALGORITHM   **Binary Search Tree Algorithm.**
procedure insertion ($T$: binary search tree, $x$: item)
$v :=$ root of $T$
{a vertex not present in $T$ has the value null}
while  $v \neq null$ and $label(v) \neq x$
begin
 if  $x < label(v)$ then
 if  left child of $v \neq null$ then $v :=$ left child of $v$
 else  add new vertex as a left child of $v$ and set $v := null$
 else
 if  right child of $v \neq null$ then $v :=$ right child of $v$
 else  add new vertex as a right child of $v$ and set $v := null$
end
if  root of $T = null$ then  add a vertex $r$ to the tree and label it with $x$
else if  $label(x) \neq x$ then  label new vertex with $x$
{$v$=location of $x$ }
```

Computational Complexity:

- Suppose T be a binary search tree for a list of n items.
- Form a full binary tree U from T by adding unlabeled vertices so that vertex with a key has two children.

1) The most comparisons need to add a new item is the length of the longest path in U from the root to a leaf.
2) The internal vertices of U are the vertices of T.



- U has n internal vertices and $n+1$ leaves.
- The height of $U \geq \lceil \log(n+1) \rceil$.

□ PREFIX CODES

• Consider using bit strings of different lengths to encode letters.

Definition: To ensure that no bit string corresponds to more than one sequence of letters, the bit string for a letter must never occur as the first part of the string for another letter. Codes with this property are called **prefix codes**.

$\{0, 1, 01, 001\}$

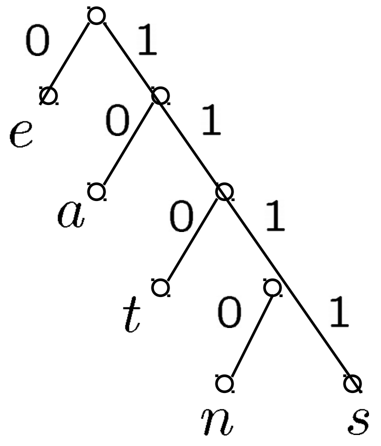
$\{0, 10, 110, 1110, 1111\}$

- A prefix code can be represented using a binary tree:

– Characters are the labels of the leaves

– The edges are labeled so that an edge leading to a left child is assigned a 0 and an edge leading to a right child is assigned a 1.

– The bit string used to encode a character is the sequence of the label of the edges in the unique path from the root to the leaf.



Huffman coding can be used to produce efficient codes based on the frequencies of occurrences of characters.

□ DECISION TREES

Definition: A rooted tree in which each internal vertex corresponds to a decision, with a subtree at these vertices for each possible outcome of the decision, is called a **decision tree**.

9.3 Tree Traversal

- UNIVERSAL ADDRESS SYSTEMS
- TRAVERSAL ALGORITHMS
  - Preorder traversal

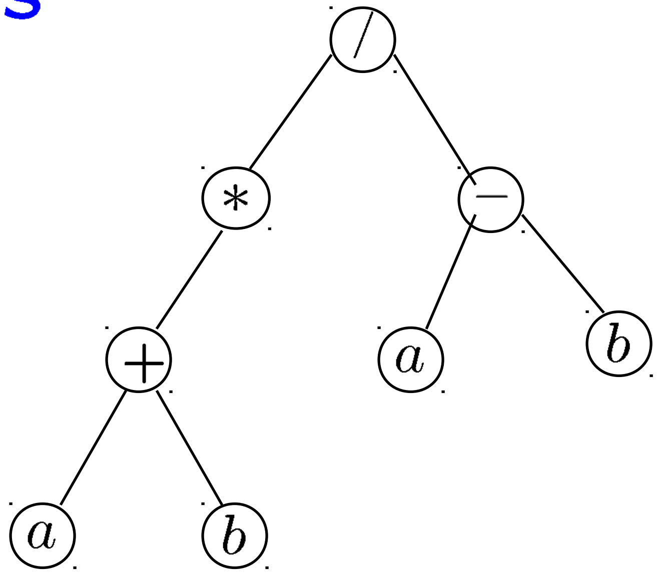
$$/ * + ab - ab$$

  - Inorder traversal

$$(a + b) * a/(a - b)$$

  - Postorder traversal

$$ab + *ab - /$$



Postfix expression

9.4 Spanning Trees

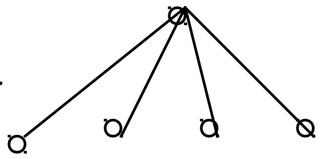
Definition: G be a simple graph, a tree T is a **spanning tree** of G if:

– T is a subgraph of G

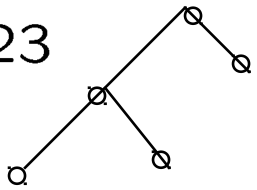
– T contains all the vertices of G

Example: How many non-isomorphic spanning trees of K_5?

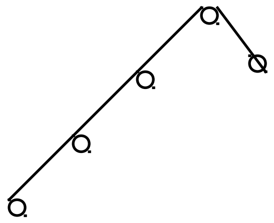
11114



11123



11222



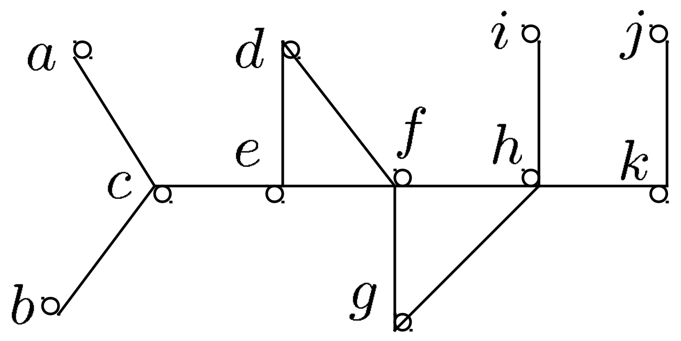
Theorem: A simple graph is a connected if and only if it has a spanning tree.

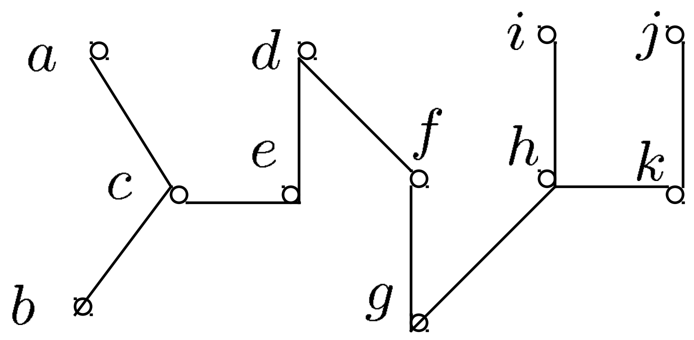
☐ Algorithms for Constructing Spanning Trees

- depth-first search

depth-first search ia also called **backtracking**.

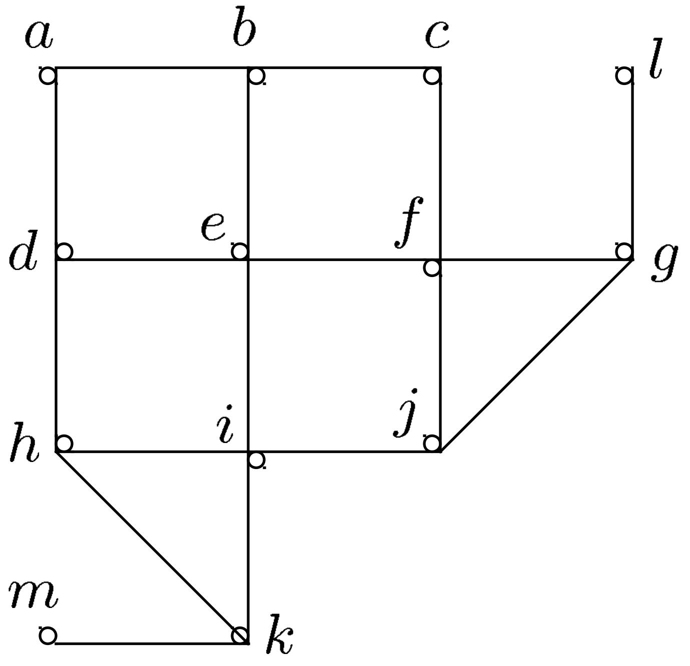
Example: Use a depth-first search to find a spanning tree for the following graph.

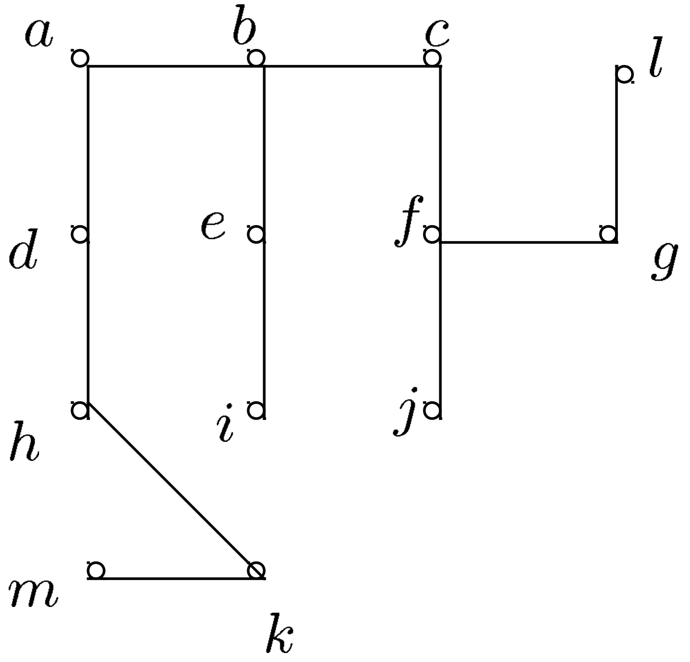




- breadth-first search

Example: Use a breadth-first search to find a spanning tree for the following graph.





9.5 Minimum Spanning Trees

Definition: A **minimum spanning tree** in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges.

□ Algorithms for Minimum Spanning Trees

Two algorithms are presented for constructing minimum spanning trees:

Kruskal's algorithm
Prim's algorithm

- These algorithm are examples of **greedy algorithms**.

– A greedy algorithm is a procedure that makes an optimal choice at each of its steps.

Optimizing at each stage of a algorithm does not guarantee that the optimal overall solution is produce.

– However, the two algorithms are greedy algorithms that do produce optimal solution.

- Kruskal's algorithm:(1956)

– Choose an edges in the graph with minimum weight.

– Successively add edges with minimum weight that do not form a simple circuit with those edges already chosen.

– Stop when $n - 1$ edges have been added.

- Prim's algorithm(1957):

– Begin by choosing any edge with smallest weight, putting in into the spanning tree.

– Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree and not forming a simple circuit with those edges already in the tree.

– Stop when $n-1$ edges have been added.

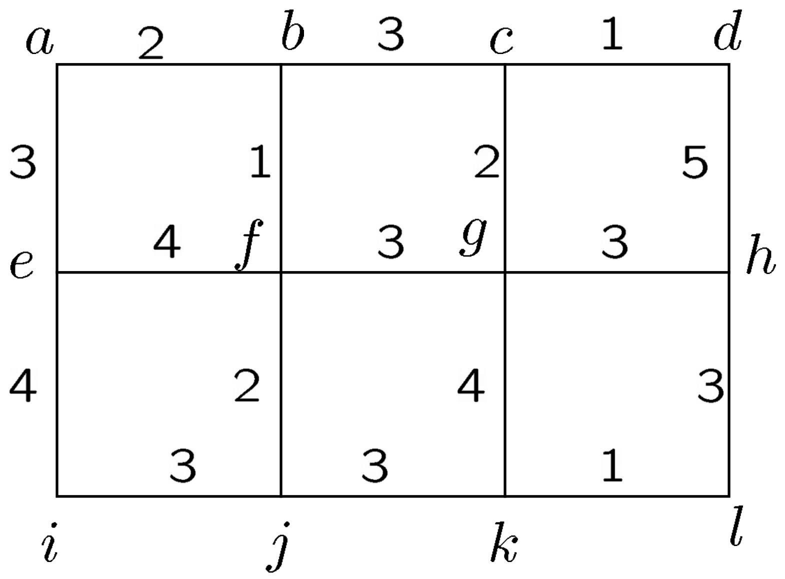
Remark:

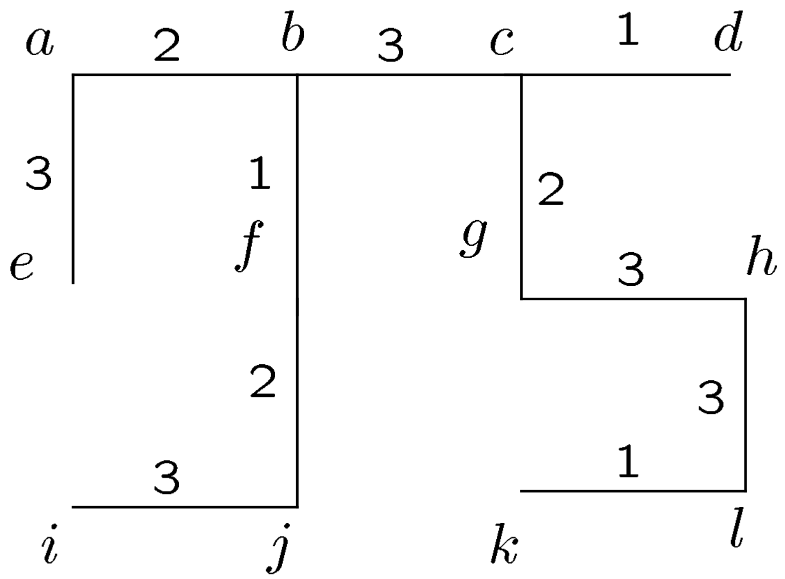
The difference between Prim's and Kruskal's algorithms:

– In Prim's algorithm edges of minimum weight that are incident to a vertex already in the tree, and not forming a circuit, are chosen;

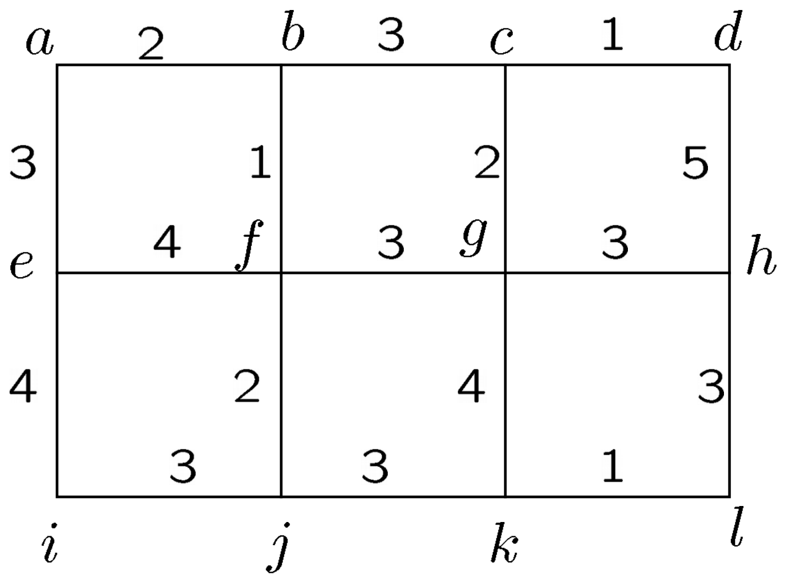
– In Kruskal's algorithm edges of minimum weight that are not necessary incident to a vertex already in the tree, and not forming a circuit, are chosen.

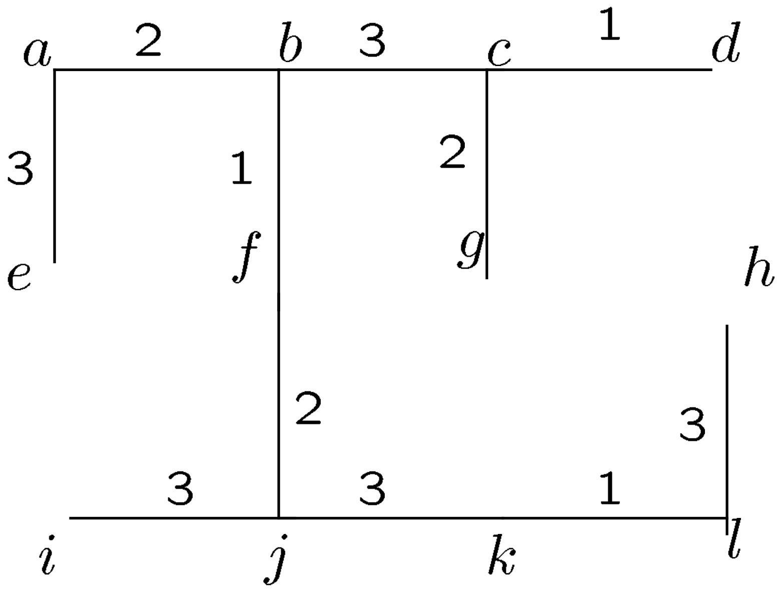
Example: Using Prim's algorithm and Kruskal's algorithm to find a minimum spanning tree in the weighted graph shown below:





Kruskal's algorithm





Prim's algorithm

We will prove that Prim's algorithm produces a minimum spanning tree of a connected weighted graph.

Proof: Let G be a connected weighted graph.

Suppose that the successive edges chosen by Prim's algorithm are $e_1, e_2, \cdots, e_{n-1}$.

Let S be the tree with $e_1, e_2, \cdots, e_{n-1}$ as its edges, and S_k be the tree with $e_1, e_2, \cdots, e_k$.

Let T be the minimum spanning tree of G containing the edges $e_1, e_2, \cdots, e_k$, where k is the maximum integer with the property that a minimum spanning tree exists containing the first k edges chosen by Prim's algorithm.

The theorem follows if we show that $S = T$.

Suppose that $S \neq T$. Consequently, T contains $e_1, e_2, \cdots, e_k$, but not e_{k+1}.

Consider $T \cup \{e_{k+1}\}$.

It must contain a simple circuit.

– The simple circuit must contain e_{k+1}
– There must be a edge in the simple circuit that does not belong to S_{k+1}.

We can find an edge e not in S_{k+1} that has an endpoint that is also an endpoint of one of the edges $e_1, e_2, \cdots, e_k$.

By deleting e from T and adding e_{k+1}, we obtain a tree T' with $n-1$ edges.

T' contains $e_1, \cdots, e_k, e_{k+1}$.

Since e_{k+1} was chosen by Prims algorithm at $k+1$st step, e was also available at that step.

$\boxed{\text{The weight of } e_{k+1} \leq \text{the weight of } e.}$

$\Rightarrow T'$ is also a minimum spanning tree. $\boxed{\textbf{Contradiction!}}$