Chapter 3: Trees and Arborescences

3.1. Definition of a tree, Co-tree

- A tree is an undirected graph that is connected and has no cycles.

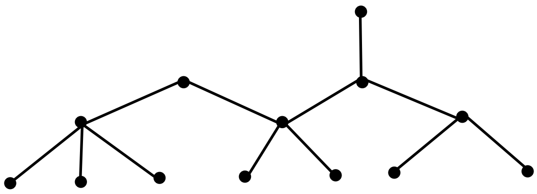
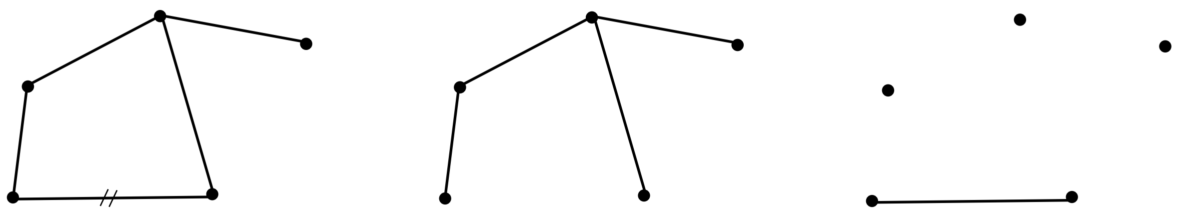


Fig. 3.1: Example of a tree

- If G is an undirected, connected graph, the tree T of the graph G is a partial graph, connected, and without cycles.
- The co-tree T' associated with T is the complementary partial graph of T with respect to G.

Example 3.1: Building a tree, co-tree from a graph

Let the following graph G be:



(G) **The tree T associated with the graph G** **The co-tree T'associate**

Fig. 3.2: Associated Graph, Tree and Co-Tree

3.2. Definition of a forest:

- Is an undirected graph, without cycles (connectivity is not necessary)

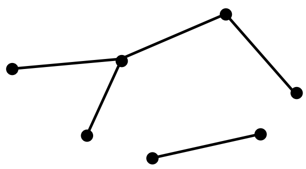
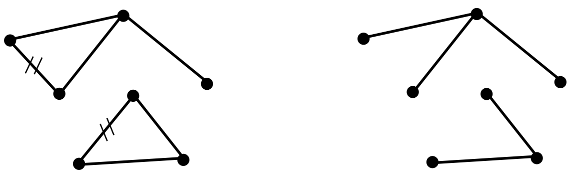


Fig.3.3: Definition of a forest

- A forest of a graph G is a partial, unconnected graph of G, without cycles.
- A forest is not a tree (consists of several trees).



A graph G A forest associated with the graph G

Fig.3.4: Difference between a tree and a forest

Remarks

In a tree, Co-tree, forest we distinguish two types of vertices:

- Vertices with multiple incident edges (d(x) >1) (incoming or outgoing edges)
- Vertices having only one incident edge are called pendant vertices (vertices of degree 1).

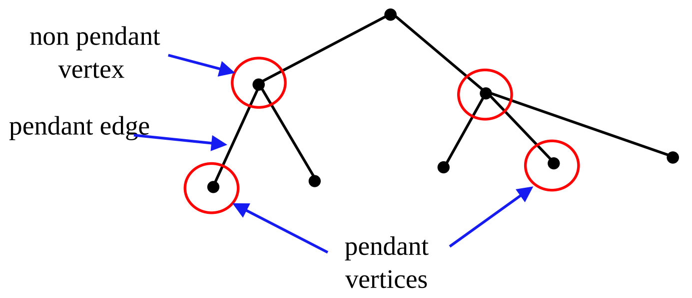


Fig. 3.5: Pendant vertices and non pendant vertices

3.3. Some properties of a tree

- A tree of order≥2 admits at least two pendant vertices.
- Every connected graph G admits a tree as a partial graph (eliminating cycles)
- A tree of order n is of size $(n-1)$

- If G is a connected undirected graph of order n and size m, then it admits a tree T of order *n* and size (*n-1*) and a co-tree of order *n* and size *m-(n-1)*
- Deleting an edge from a tree disconnects the tree (gives two connected components)
- Any edge of a tree is an isthmus (connects 2 connected components), it is an edge which is not contained in any cycle.
- Any pair of vertices *x* and *y* in tree T is connected by a unique chain.
- Adding an arc/edge to the tree creates a single cycle.

3.4. Definition of an arborescence

Let G be a directed graph, we call an arborescence of a directed tree such that there exists a particular vertex *r*, such that there exists a path from *r* to any other vertex of G, *r* is called the root of the arborescence.

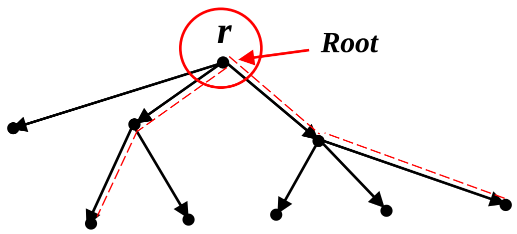


Fig.3.6: Defining an arborescence

Observation: An arborescence is a directed tree with a root.

3.5. Properties of an arborescence

- An arborescence of order≥2 admits at least one pendant vertex.

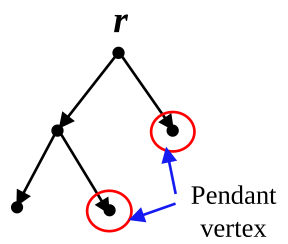


Fig. 3.7: Definition of pendant vertices

- Deleting a pendant vertex from an arborescence of order≥2 gives an arborescence.

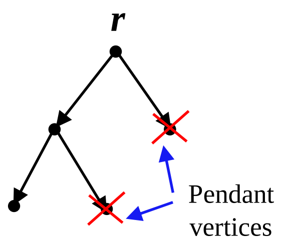


Fig. 3.8: Result of Deleting a pendant vertex

- In an arborescence with a root r, there exists a unique path from r to any vertex x≠r. (if ∃2 paths⇒ ∃cycle⇒contradiction with the definition (is not an arborescence))

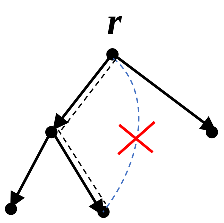


Fig. 3.9: Path uniqueness in an arborescence

- In an arborescence A with a root r, $d^-(r)=0$ and $d^-(x)=1$ for any vertex x≠r

- If A is an arborescence, the set of descendants of x $\Gamma^{+}(x)$ (the successors of x) generates an arborescence A′ of a root x for any vertex x.

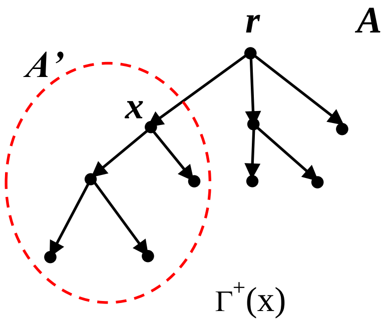


Fig. 3.10: Successors of a vertex in an arborescence

- Deleting an arc from an arborescence A gives two (02) disjoint sub-arborescences A1, A2.

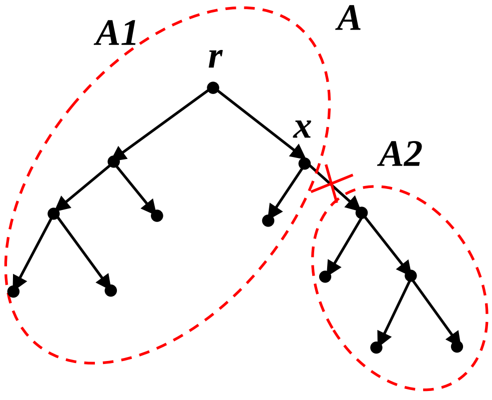


Fig. 3.11: Deleting an arc from an arborescence

- If G is a directed graph with a root *r*, then there exists in G an arborescence A with root *r*, which is a partial graph of G.

3.6. Representation of an arborescence

An arborescence can be represented by tables (vectors and matrices) seen in chapters II. However, it admits a more efficient representation using linked lists (dynamic representation) as follows:

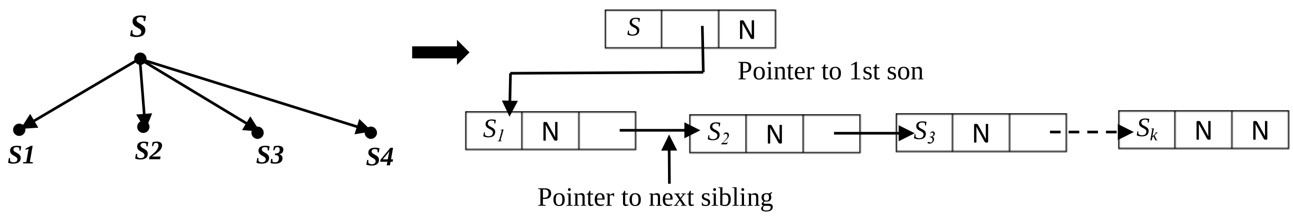


Fig. 3.12: Dynamic representation of an arborescence

Example 3.2: Dynamic representation of an arborescence

Consider the following arborescence:

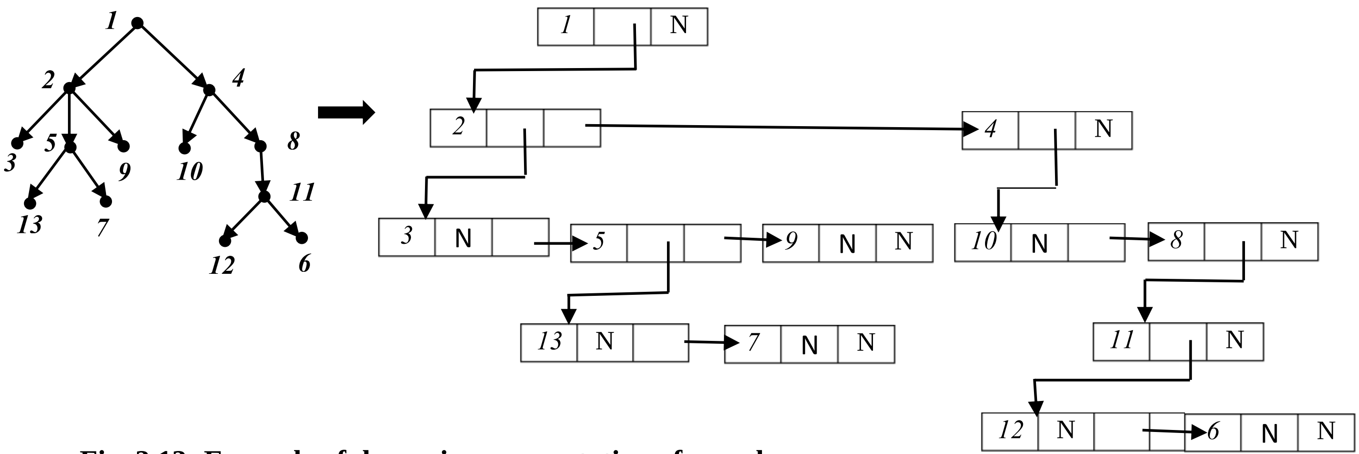


Fig. 3.13: Example of dynamic representation of an arborescence

3.7. Browsing an arborescence:

Several browsing/navigating modes are possible:

1. Width path: 1, 2, 4, 3, 5, 9, 10, 8, 13, 7, 11, 12, 6
2. In-depth path: 1, 2, 3, 5, 13, 7, 9, 4, 10, 8, 11, 12, 6

Remark

Three types of navigation are also used: preorder, inorder, post-order (prefixed, infixed, postfixed)

3.8. Cycles and associated vectors:

Let G(X, U) be a directed graph of order |X| = *n* and size |U| = *m*. We can match any cycle *c* with a vector $v_c = (v_1, v_2, \ldots, v_i, .., v_m)$, with:

$$v_i = \begin{cases} +1 \ \ if\ the\ arc\ v_i \in C\ and\ it\ is\ in\ the\ direction\ of\ the\ path\ of\ c \\ -1 \ \ if\ the\ arc\ v_i \in C\ and\ it\ is\ in\ the\ inverse\ direction\ of\ the\ path\ of\ c \\ 0 \ \ if\ the\ arc\ v_i is\ not\ in\ C \end{cases}$$

Remark : We are talking here about a cycle instead of a circuit, because the component vectors have different directions.

Example 3.3: Finding vectors associated with cycles

Let the graph G be defined as follows:

X = {a, b, c, d, e, f, g, h}

Let the cycle c_1 = (1, 3, 5, 9, 8, 2)

The associated vector is as follows:

$v_{c1} = (+1, -1, +1, 0, -1, 0, 0, -1, +1, 0)$

Let the cycle c_2 = (1, 3, 10, 8, 2)

The associated vector is as follows:

$v_{c2} = (+1, -1, +1, 0, 0, 0, 0, -1, 0, +1)$

Let the cycle c_3 = (5, 9, 10)

The associated vector is as follows:

$v_{c3} = (0, 0, 0, 0, -1, 0, 0, 0, +1, -1)$

Let the cycle c_4 = (4, 10, 8)

The associated vector is as follows:

$v_{c4} = (0, 0, 0, -1, 0, 0, 0, -1, 0, +1)$

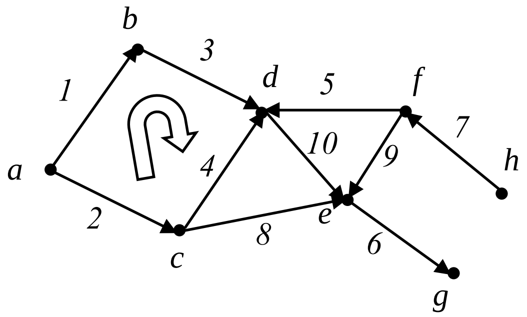


Fig. 3.14: Cycles and associated vectors

3.9. Independent cycles:

The cycles c_1, c_2, …, c_k are said to be independent if the corresponding vectors v_{c1}, v_{c2}, …, v_{ck} are linearly independent.
i.e., there exists a relation of the form:
$\alpha_1 v_{c1} + \alpha_2 v_{c2} + \ldots\ldots + \alpha_k v_{ck} \neq \vec{0}$ with : $\alpha_1, \alpha_2, \ldots, \alpha_k$ are real numbers not all equal to zero, otherwise (i.e., $\alpha_i = 0 \quad \forall i = 0, 1, \ldots$) they are said to be dependent.

For example, the cycles c_1, c_2, c_3 are independent because we have:
$-v_{c1} - v_{c2} + v_{c3} \neq \vec{0} \;\; (\alpha_1 = -1, \alpha_2 = -1, \alpha_3 = +1)$

Verification :
(-1)*(+1, -1, +1, 0, -1, 0, 0, -1, +1, 0) + ***(-1)*** (+1, -1, +1, 0, 0, 0, 0, -1, 0, +1) + ***(+1)*** (0, 0, 0, 0, -1, 0, 0, 0, +1, -1) = ***(-2, +2, -2, 0, 0, 0, 0, 0, +2, -2)*** $\neq \vec{0}$

Theorem:
Let $c_1, c_2, c_3, \ldots, c_k$ cycles, if each cycle contains an arc that the others do not contain, then the cycles c_1, c_2, c_3, …, c_k form a set of independent cycles.

3.10. Definition of a cycle base:

A cycle base is a minimal set of independent cycles *ci* corresponding to vectors v_{ci} (linearly independent), such that any vector of the graph G can be expressed in function of this base (in function of vectors of this base).

Example 3.4: Defining a cycle base in a graph

$c_1 = (1, 6, 2) \Rightarrow v_{c1} = (+1, -1, 0, 0, 0, +1)$
$c_2 = (1, 6, 3) \Rightarrow v_{c2} = (+1, 0, +1, 0, 0, +1)$
$c_3 = (2, 3) \Rightarrow v_{c3} = (0, -1, +1, 0, 0, 0)$
$c_4 = (1, 4, 5, 2) \Rightarrow v_{c4} = (+1, -1, 0, +1, -1, 0)$
$c_5 = (6, 5, 4) \Rightarrow v_{c5} = (0, 0, 0, +1, -1, +1)$
$c_6 = (1, 4, 5, 3) \Rightarrow v_{c6} = (+1, 0, +1, +1, -1, 0)$

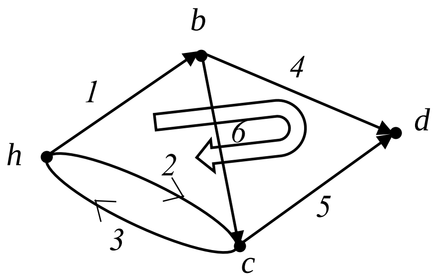


Fig. 3.15: Definition of a cycle base

We note that c_1, c_3, c_5 are independent, because each cycle contains a vector that the others do not contain c_1(**1**), c_3(**3**), c_5(**5/4**) ===> form a base of cycles BC.

Theorem: Let G(X, U) be a graph of order *n* and size *m*, consisting of *p* distinct connected components.
The dimension of the cycle base of this graph is given by the relation:

$$\boldsymbol{V(G) = m - n + p}$$

V(G) is called the cyclomatic number of G.

3.11. Definition of a cocycle base

G(X, U) a graph, $U = \{u_1, u_2, \ldots, u_m\}$, $|X| = n$, $|U| = m$
Let *A* be a subset of vertices of X, $A \subset X$
A cocycle θ is the set of arcs connecting A and (X – A), i.e.,

$$\theta = w(A) = w^+(A) \cup w^-(A)$$

We associate with the cocycle θ the vector $v_\theta = (\theta_1, \theta_2, \ldots, \theta_m)$ defined as follows:

$$\theta_i = \begin{cases} +1 \ \ if \ \theta_i \in \ \mathrm{w}^+(\mathrm{A}) \\ -1 \ \ if \ \theta_i \in \ \mathrm{w}^-(\mathrm{A}) \\ 0 \ \ Otherwise \end{cases}$$

Example 3.5: Defining a cocycle base in a graph

Let the following graph G be:

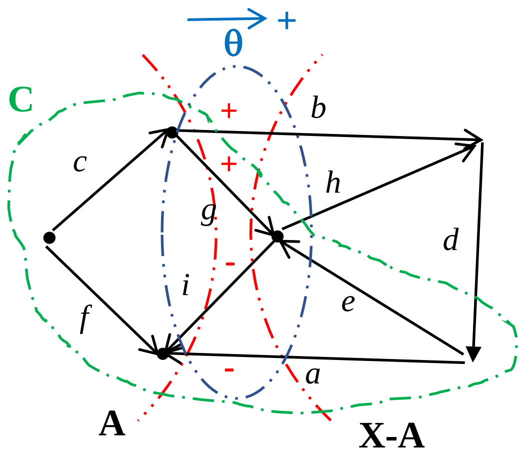


Fig. 3.16: Definition of a cocycle base

Let the cycle c = {*a*, *e*, *g*, *c*, *f*} and the cocycle θ = {*b*, *g*, *i*, *a*}
Vectors associated with the cycle c and the cocycle θ will be as follows:

Table. 3.1: Vectors associated with cycles and cocycles

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	
Vc	+1	0	+1	0	-1	-1	+1	0	0	
$V\theta$	-1	+1	0	0	0	0	+1	0	-1	
$Vc \ . \ V_\theta$	-1	0	0	0	0	0	+1	0	0	**0**

$$===>\sum \vec{V_c} \, . \, \overrightarrow{\mathrm{V}_\theta} = 0$$

OBS: the scalar product of a cycle and a cocycle is zero

The set of linearly independent cocycles forms a cocycle base for the graph G.
The dimension of this base $\lambda(G)$ is given by the relation:

$$\boldsymbol{\lambda(G) = n - 1}$$

λ(G) is also called the cocyclomatic number of G.

3.12. Algorithm for searching a cycle base of a connected graph

Let G(X, U) be a connected graph

1. Find a maximal tree T in the graph G (which contains all vertices)
2. Adding an arc $\vec{u}$ from the co-tree T' to T creates a unique cycle c_u oriented in the direction of $\vec{u}$
3. Keep all the unique obtained cycles ==> form the sought base (are linearly independent)

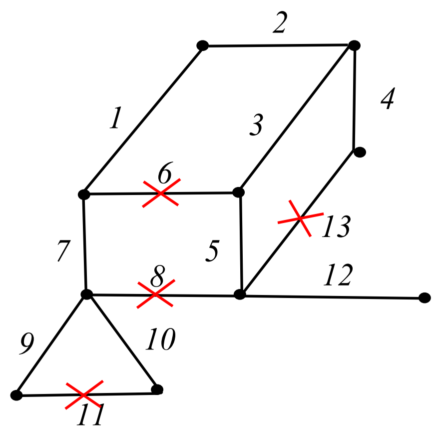
Example 3.6: Looking for a cycle base in a graph

We consider the following graph:

$n = 10$; $m = 13$

We have: n = 10 vertices in the tree ==> the size of the resulting tree T is:

$m' = n - 1$ = 10 – 1 = 9 edges



∃ only one connected component ($p = 1$)

The cyclomatic number (the dimension of the cycle base) is $V(G) = m - n + p = 13 - 10 + 1 = 4$

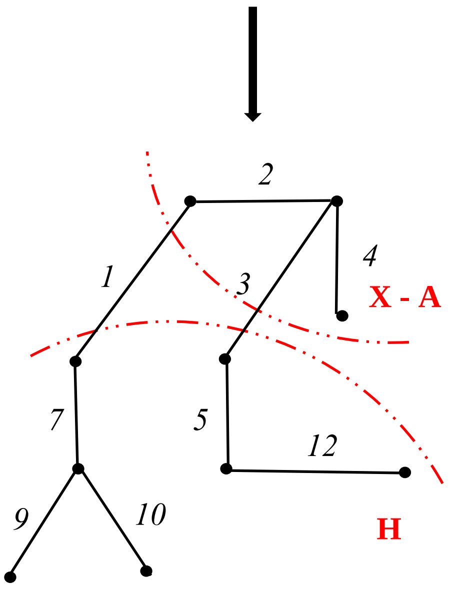
The cycles that form the cycle base are:

$C_6 = (1, 2, 3, \mathbf{6})$

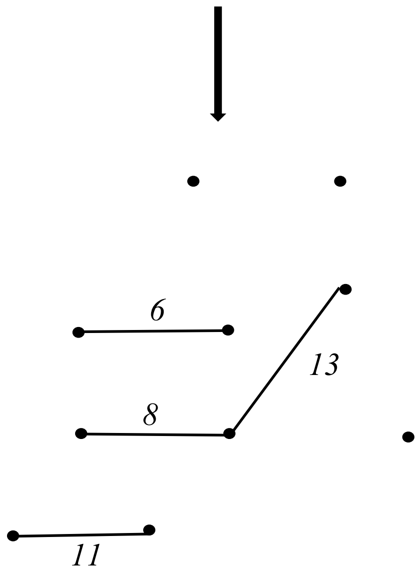
$C_8 = (1, 2, 3, 5, \mathbf{8}, 7)$

$C_{11} = (9, 10, \mathbf{11})$

$C_{13} = (3, 4, \mathbf{13}, 5)$



The obtained Maximal Tree



The obtained co-tree

Fig. 3.17: Cycle base search process

OBS: If it is necessary, we give an arbitrary orientation to the different edges, and to the cycles the same direction of the added edge (which closes the cycle).

3.13. Searching for a cocycle base

1. Divide X into A and $\boldsymbol{(X - A)}$
2. The V arcs including $I(V) \in A$ and $T(V) \in (X - A)$ form a cocycle θ_V
3. The set of obtained unique cocycles (each cocycle contains an arc that the others do not contain) form the sought base (are linearly independent)

3.14. Algorithm for searching a maximal tree

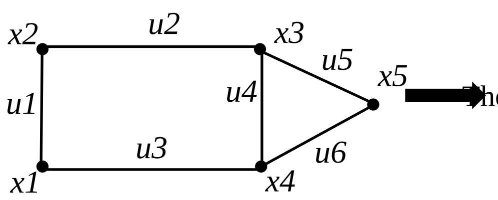
G(X, U) is a connected graph of order n and size m.

A maximal tree of G is a partial graph T of G, connected, without cycle of order n and size $(n - 1)$.

The algorithm for constructing the tree T of the graph G consists of taking the $(n - 1)$ edges which do not close cycles (We keep all the vertices and delete edges)

Example 3.7: Finding a maximal tree

Let G(X, U) with: $|X| = n = 5;\ |U| = m = 6$



The algorithm

Edges	$u1$	$u2$	$u3$	$u4$	$u5$	$u6$
Decision	OK	OK	OK	No	OK	No

The Maximal Tree

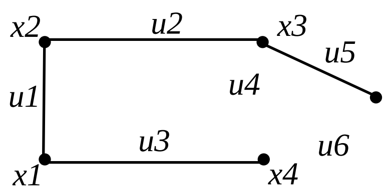


Fig. 3.18: Approach to find a maximal tree in a graph

3.15. Searching for a Maximal tree with Minimum weight (Algorithm of Kruskal 1)

Let G(X, U, L) be a valued, undirected, connected graph with all edge lengths different (if $u \neq v \implies L(u) \neq L(v)$)

(i) The graph is represented by the list of arcs sorted according to their increasing weights.

(ii) Let's start with an empty graph, and we successively take the first *(n – 1)* edges which do not close cycles with those already taken.

(iii) The *(n – 1)* retained edges form a maximal tree with minimum weight.

OBS: the maximal tree with minimum weight consists of *(n – 1)* edges)

Example 3.8: Finding a maximal tree of minimum weight by applying KRUSKAL 1

Example: Let G = (X, U, L)

With: X = {1, 2, 3, 4, 5, 6, 7, 8, 9}

U = {a, b, c, d, e, f, g, h, i, j, k, l, p, q}

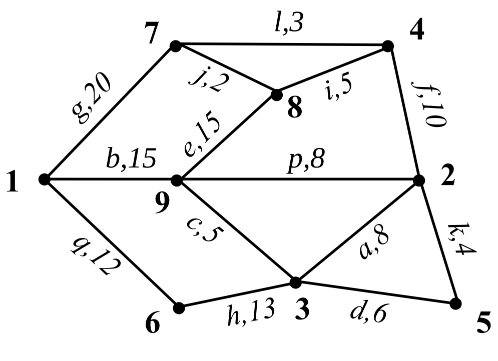


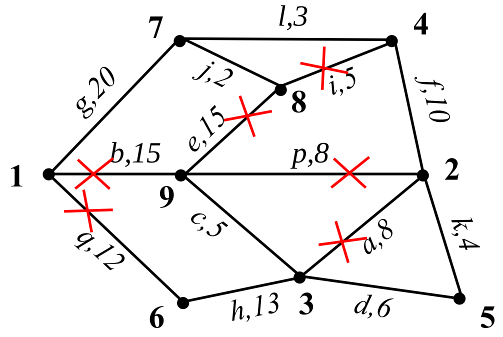
Fig. 3.19: Search for a maximal tree of minimum weight in a valued graph

Application of KRUSKAL 1 algorithm:

Unsorted edge list

Table. 3.2: Sorted list of edges representing a graph

Edges	*a*	b	c	d	e	f	*g*	*h*	*i*	*I*	*k*	*L*	*p*	*q*
Ext.Init	3	1	3	3	8	4	1	6	4	7	2	7	9	1
Ext.Ter	2	9	9	5	9	2	7	3	8	8	5	4	2	6
Weight	8	15	5	6	15	10	20	13	5	2	4	3	8	12



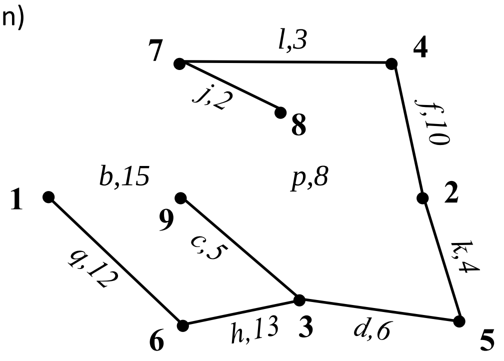
After sorting we will have:

List of edges sorted by their weight:

Table. 3.3: Weight-sorted edge list representing a graph

Edges	*I*	*L*	*k*	c	*i*	d	*a*	*p*	f	*q*	*h*	b	e	*g*
Ext.Init	7	7	2	3	4	3	3	9	4	1	6	1	8	1
Ext.Ter	8	4	5	9	8	5	2	2	2	6	3	9	9	7
Weight	2	3	4	5	5	6	8	8	10	12	13	15	15	20
Decision	ok	ok	ok	ok	X	ok	X	X	ok	ok	ok	X	X	X

We stop here, because we have the size of the maximum tree is n -1 = 9 – 1 = 8 edges (already taken)



Maximum tree of minimum weight
Nb.edges = 9 – 1 =8; TotalWeight = 55

Fig. 3.20: Search process of maximum Tree of minimum weight in a valued graph

3.16. Kruskal Algorithm 2

(i) Represent the graph by the list of (arcs/edges) given as input

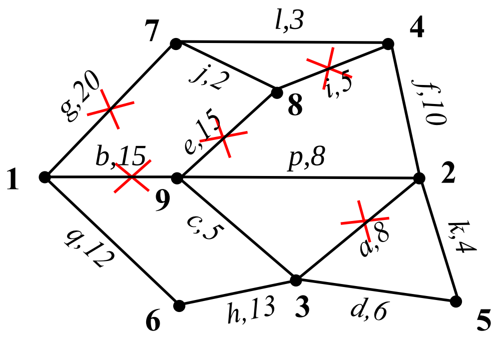
(ii) Let's start with an empty graph, we take the arcs successively and as soon as the arc currently being processed forms a cycle with those already taken, we remove the arc of the maximum weight from the cycle

(iii) The arcs retained are those of a tree of minimum weight

Application to the previous example:

Table. 3.4: Unsorted list of edges representing a graph

Edges	*a*	b	c	d	e	f	*g*	*h*	*i*	*I*	*k*	*L*	*p*	*q*
Ext.Init	3	1	3	3	8	4	1	6	4	7	2	7	9	1
Ext.Ter	2	9	9	5	9	2	7	3	8	8	5	4	2	6
Weight	8	15	5	6	15	10	20	13	5	2	4	3	8	12
Decision	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	ok	X	ok
Final Dec	X	X	ok	ok	X	ok	X	ok	X	ok	ok	ok	X	ok



Maximum tree of minimum weight

Fig. 3.21: Search for a maximal tree of minimum weight by applying KRUSKAL 2

Remark :

By applying the same algorithms we can construct the maximal tree of maximum weight.