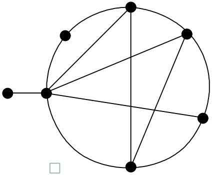
1 What is a Graph

Graphs present a key concept of discrete mathematics. Informally, a graph consists of

- *vertices*, sometimes called nodes (“dots”),
- and *edges* (“arcs”) between pairs of vertices.



□

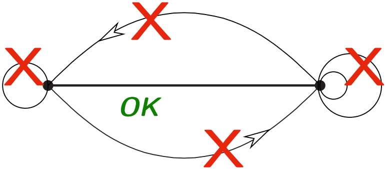
Importantly, graphs are easy to draw and understand from a picture, and easy to process by computer programs, too. □

Brief outline of this lecture

- What is a graph – basic terminology,
  examples and trivial classes of graphs, vertex degrees.
- Subgraphs and isomorphism, recognizing (non-)isomorphic graphs.
- Trees and forests – graphs without cycles.
- Directed graphs and multigraphs.

1.0 Defining a GRAPH – a foreword

- First to say, we are going to define a *simple undirected graph*...
  - NO multiple edges
  - NO arrows
  - NO loops



- Not to forget, our graphs are finite.

simple undirected graph = jednoduchý neorientovaný graf
bez násobných hran, bez šipek, bez smyček

1.1 Defining a GRAPH

Definition 1.1. ***A graph*** (formally, a *simple undirected* graph) is a pair $G = (V, E)$, where V is the *vertex set* and E is the *edge set* – a subset of pairs of vertices.

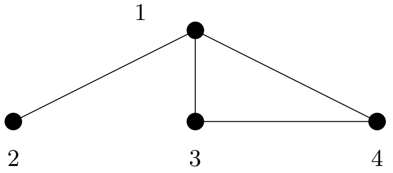
Unless stated otherwise, our graphs have finite and nonempty vertex sets. □

Notation: An edge between vertices u and v is denoted by $\{u, v\}$, or shortly uv.
The vertices joined by an edge are *adjacent* or *neighbours*.
The vertex set of a (known) graph G is referred to as $V(G)$, the edge set as $E(G)$. □

Fact: A graph is, in algebra terms, a symmetric irreflexive binary relation. □

Remark: How can we describe a graph?
Either by listing the vertices and edges, or with a nice picture. . .

$$V = \{1, 2, 3, 4\}, \quad E = \Big\{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{3, 4\}\Big\}$$



□

Which one do you like more?

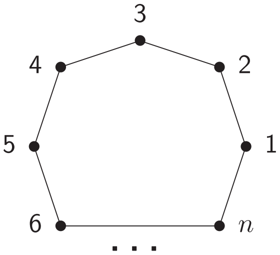
simple undirected graph = jednoduchý neorientovaný graf
vertex (vertices) = vrchol (y), edge = hrana, adjacent = sousední

Basic graph classes

One often refers to some basic special graphs by descriptive names such as the following. □

The cycle of length n has $n \geq 3$ vertices joined in a cyclic fashion with n edges:

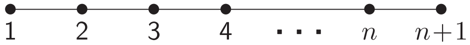
$\boldsymbol{C_n}$



□

The path of length n has $n+1$ vertices joined consecutively with n edges:

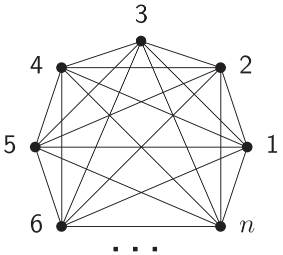
$\boldsymbol{P_n}$



cycle of length = kružnice délky, path = cesta

The complete graph *(clique)* on $n \geq 1$ vertices has n vertices, all pairs forming edges (i.e. $\binom{n}{2}$ edges altogether):

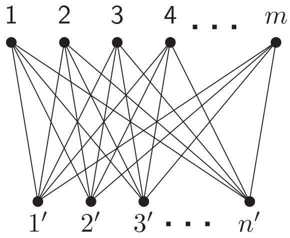
K_n



□

The complete bipartite graph on $m \geq 1$ plus $n \geq 1$ vertices has $m + n$ vertices in two parts, and the edges join all the $m \cdot n$ pairs coming from different parts:

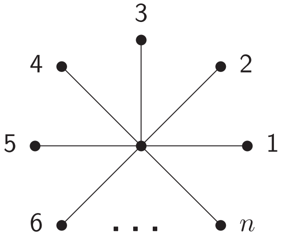
$K_{m,n}$



complete graph = úplný graf (klika), bipartite = bipartitní

The star with $n \geq 1$ rays is just a special name for the compl. bipart. graph $K_{1,n}$:

$\boldsymbol{S_n}$

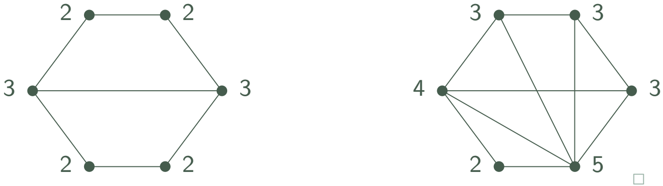


star with rays = hvězda s paprsky / rameny

Vertex degrees in a graph

Definition 1.2. ***The degree*** of a vertex v in a graph G, denoted by $d_G(v)$, equals the number of edges of G incident with v. An edge e is *incident* with a vertex v if v is one of the ends of e. □

In this example, the degrees are written at the vertices.



□

Definition: A graph is *d-regular* if all its vertices have the same degree d.

Notation: The *maximum* degree in a gr. G is denoted by $\Delta(G)$ and *minimum* by $\delta(G)$. □

Theorem 1.3. *The sum of the degrees of all vertices of a graph is always even, equal to twice the number of edges.* □

Proof. When summing the degrees, every edge has two ends and hence it is counted twice. □

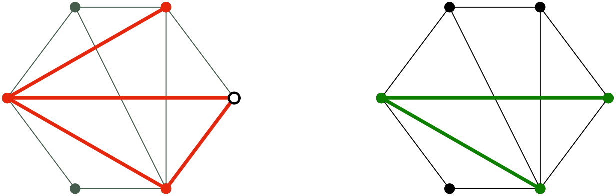
degree = stupeň vrcholu, incident ∼ připojený, regular = pravidelný, even = sudý

1.2 Subgraphs and Isomorphism

Definition: A *subgraph* of a graph G is any graph H on a subset of vertices $V(H) \subseteq V(G)$, such that the edges of H form a subset of the edges of G and have both ends in $V(H)$.

We write $H \subseteq G$ as for the set inclusion. □

Why, on the left hand side, the red subsets do not form a subgraph?



□

On the right hand side, we have got a well-formed subgraph in green colour. □

Definition: An *induced subgraph* is a subgraph $H \subseteq G$ such that all edges of G between pairs of vertices from $V(H)$ are included in H.

A *spanning subgraph* is a subgraph $H \subseteq G$ such that $V(H) = V(G)$.

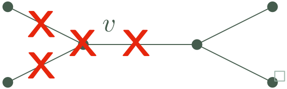
subgraph = podgraf, induced subgraph = indukovaný podgraf, spanning = pokrývající

Elementary graph operations

- *Deleting an edge or a vertex* from a graph G:

  $G \setminus e :=$ the spanning subgraph of G such that $E(G \setminus e) = E(G) \setminus \{e\}$,

  $G \setminus v :=$ the induced subgraph of G such that $V(G \setminus v) = V(G) \setminus \{v\}$.



- One can analogously delete a set of vert./edges from G: $G \setminus X$, $G \setminus F$. □

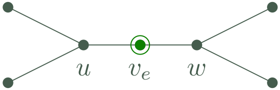
- *Adding a vertex or an edge* to a graph G, for $v \notin V(G)$, $e = uw \notin E(G)$:

  $G + v :=$ the graph on $V(G+v) = V(G) \cup \{v\}$ and $E(G+v) = E(G)$,

  $G + e :=$ the graph on $V(G+e) = V(G)$ and $E(G+e) = E(G) \cup \{e\}$, assuming $u, w \in V(G)$. □

- *Subdividing an edge* $e = uw \in E(G)$ in a graph G:

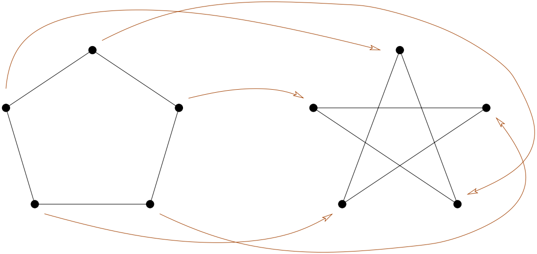
  – the result is the graph $(G \setminus e) + v_e + \{uv_e, v_ew\}$.



subdividing = podrozdělení

Isomorphism of graphs

Definition 1.4. ***An isomorphism*** between graphs G and H is a bijective mapping (*one to one*) $f : V(G) \to V(H)$ such that, for every pair of vertices $u, v \in V(G)$, it holds that $\{u, v\}$ is an edge of G if and only if $\{f(u), f(v)\}$ is an edge of H. □



An isomorphism mapping between two pictures of the cycle of length 5. □

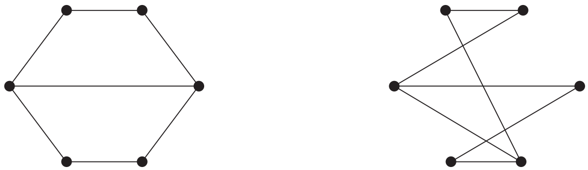
Graphs G and H are *isomorphic* if there exists an isomorphism between them. We write $G \simeq H$. □

Fact: Isomorphic graphs have the same numbers of vertices and of edges.

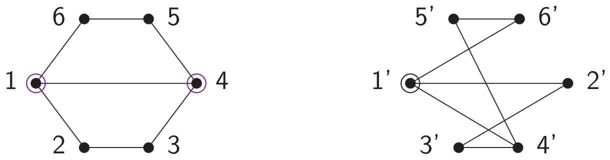
Fact: If a bijection f is an isomorphism, then f must map the vertices of same degrees, i.e. $d_G(v) = d_H(f(v))$. The converse is not sufficient, though!

isomorphism = isomorfismus / izomorfismus, bijective = vzájemně jednoznačné

Example 1.5. *Are the following two graphs isomorphic?*



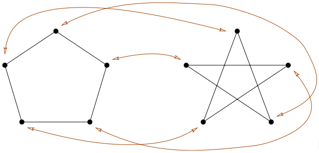
We shall first compare the numbers of vertices and of edges. (Agree.)□ Then we compare their degree sequences, i.e., the multisets of vertex degree values. (Again, they agree $2,2,2,2,3,3$.)□ This means we have found no easy distinction, and the graphs might (or may not!) be isomorphic. We have to start looking for all possible bijections between them. □



In this particular case, it helps to notice that the two degree-3 vertices on the left are symmetric, and so we may choose any one of them to map it to the leftmost vertex of the second graph. Numbering the vertices by $1,2,3,4,5,6$, we can construct the rest of the mapping (already have $1 \to 1'$) as in the picture. □

Graphs as isomorphism classes

Usually, we do not treat graphs as particular pictures or presentations, but more generally. . .



□

Proposition 1.6. *The relation "to be isomorphic $\simeq$" is an equivalence relation on the class of all graphs.*

Proof. We easily show that $\simeq$ is reflexive, symmetric, and transitive. □ □

The more suitable view of "a graph"

- An important corollary of Proposition 1.6 is that the class of all graphs is partitioned by $\simeq$ into the *isomorphism classes*. □
- Hence when we speak about "a graph", we (usually) actually mean its whole isomorphism class. A particular presentation of the graph then does not matter.

isomorphism class = třída isomorfismu

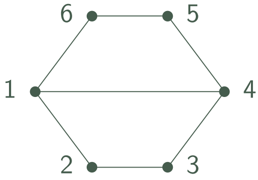
Specialized subgraph terms

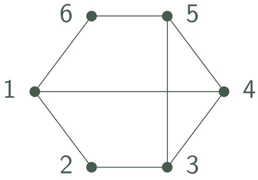
Notation: Consider an arbitrary graph G.

- A subgraph $H \subseteq G$ isomorphic to a cycle is called a *cycle in G*.
- Specially, a *triangle* in G is a subgraph isomorphic to the cycle of length 3. □
- A subgraph $H \subseteq G$ isomorphic to a path is called a *path in G*.
- A cycle / path of length k is also shortly called a *k-cycle / k-path*. □
- A subgraph $H \subseteq G$ isomorphic to a complete graph is called a *clique in G*. □
- An induced subgraph $H \subseteq G$ isomorphic to a cycle is called an *induced cycle in G*. Analogously for an induced path...

Does it make sense to speak separately about an induced clique? And induced star? □

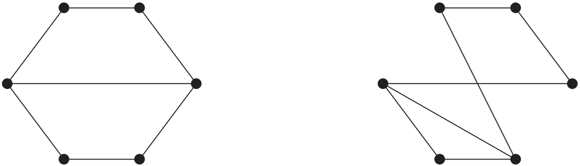
What subgraphs and induced subgraphs can you find in the following graphs?





triangle = trojúhelník , induced cycle / path = indukovaná kružnice /cesta

Example 1.7. *Are the following two graphs isomorphic?*



□

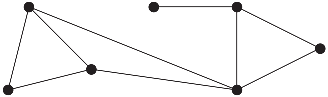
We start analogously to previous Example 1.5. Both these graphs have the same numbers of vertices and edges, and the same degree sequence $2, 2, 2, 2, 3, 3$. However, if one tries to find an isomorphism, even exhaustively, (s)he fails. What is going on? □

Which property of the graphs prevents us from finding an isomorphism? Since there are (especially for larger graphs) too many potential bijective mappings to try them exhaustively in practice, we shall look for some other, *ad-hoc*, approaches. . .
This time, the first graph has no triangle and the second one has one! Hence they cannot be isomorhpic. □ □

Fact: No universal and efficient way of deciding an isomorphism between two graphs is currently known (the problem is not known in P). On the other hand, however, the problem is neither NP-hard to our knowledge, and so a general polynomial algorithm might emerge in the future. . .

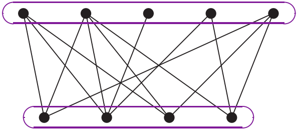
Some special kinds of graphs

Connected graphs are those graphs G such that, for any two vertices $u, v \in V(G)$, there is a path in G between the ends u and v.



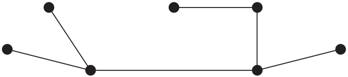
– more details in Lecture 2. . . □

Bipartite graphs are subgraphs of some complete bipartite graph $K_{m,n}$.



– more details in Lecture 6. . . □

Trees are connected graphs with no cycles as subgraphs → see next.

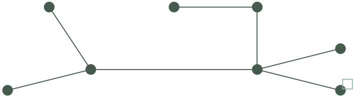


connected = souvislý (ne "spojitý"!), bipartite = bipartitní, tree = strom (graf bez kružnic)

1.3 Trees; Basic properties

Definition 1.8. ***A tree*** is a connected graph T without cycles (acyclic).

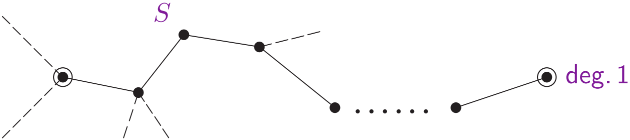
A graph which is composed of disjoint trees is called a *forest*.



Notice that a tree must be a simple graph since a loop is a cycle of length one and parallel edges form a cycle of length two. Cf. Section 1.4. . . □

Lemma 1.9. *A tree on more than one vertex contains a vertex of degree* 1 *(a "leaf").* □

Proof: We consider a longest possible path S in our tree T. Since there are no cycles in T, the endvertices of the path S must have no other edges incident with them (or S can be prolonged). Hence the degree of these ends is one.



□

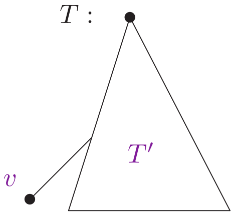
tree = strom, forest = les, acyclic = acyklický, bez kružnic, leaf = list

Number of edges of a tree

Theorem 1.10. *An n-vertex tree has exactly $n-1$ edges for $n \geq 1$.* □

Proof: By induction on $n \geq 1 \ldots$

- A one-vertex tree has $n-1=0$ edges; true. □



- Let a tree T have $n>1$ vertices.

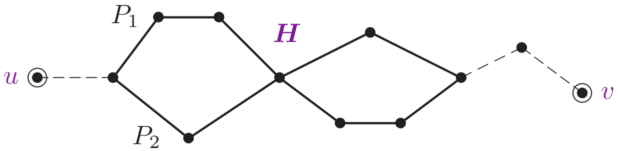
  By Lemma 1.9, T has a vertex v of degree 1. Denote by $T' = T \setminus v$ the subgraph obtained from T by deleting v. Then T' is also connected and acyclic, and hence T' is a tree on $n-1$ vertices. By the induction assumption, T' has $n-1-1$ edges, and so T has $n-1-1+1=n-1$ edges.

□

Unique paths in trees

Theorem 1.11. *Any two vertices of a tree are connected via exactly one path.* □

Proof: There exists at least one path by the definition; cf. connectivity.



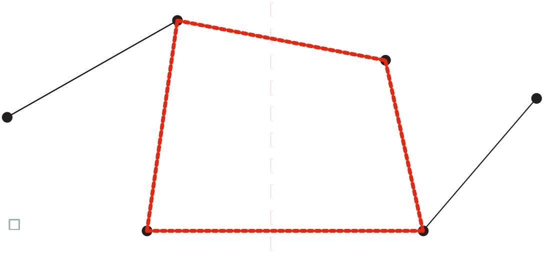
If there were two distinct paths P_1, P_2 between u, v, then their symmetric difference (w.r.t. edge sets) — a subgraph $H = P_1 \Delta P_2$ of nonempty edge set — has all degrees even and not all zero. On the other hand, H as a subgraph of a tree is acyclic, and hence its components are trees (not all isolated vertices), and by Lemma 1.9 there has to be a vertex of degree 1 in H, a contradiction. □ □

Corollary 1.12. *If a single new edge is added to a tree, this creates exactly one cycle.*

Proof: If u, v are not neighbours in a tree T, then the new edge uv forms one cycle with the unique $u\!-\!v$ path from Theorem 1.11. □

Trees as minimal connected graphs

- If T is a tree, then removal of any edge from T disconnects it. (Hence, nothing "smaller than T" can be connected.) □
- Conversely, if a connected graph had a cycle, then it would not be minimal among its connected subgraphs. (We could remove one edge from the cycle.) □

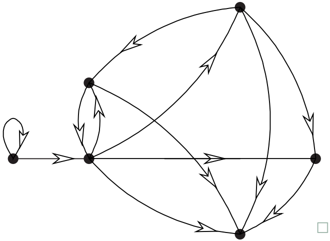


□

1.4 Directed graphs and Multigraphs

On some occasions, we need to express a "direction" of a graph edge. □

We then speak about *directed graphs* in which oriented edges, called also *arcs*, are ordered pairs of vertices (and they are drawn as arrows).



Definition 1.13. ***Directed graph*** *(digraph)* is a pair $D = (V, A)$ where $A \subseteq V \times V$.

The notions of subgraphs and of isomorphism naturally extend to digraphs.

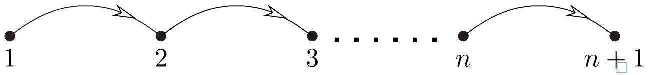
Fact: Digraphs correspond to binary relations which need not be symmetric. □

Notation: An edge / arc $e = (u, v)$ in a digraph D has its *tail* in u and the *head* in v, or e joins "from u to v". The opposite arc (v, u) is distinct from (u, v) !

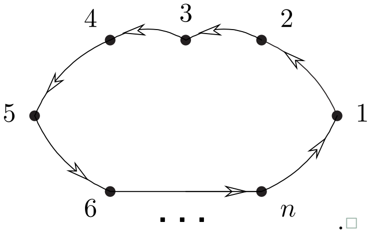
digraph = orientovaný graf, oriented edge = orientovaná hrana nebo šipka

Directed counterparts of basic classes

For example, a *directed path* of length $n \geq 0$ looks as follows



and a *directed cycle* of length $n \geq 1$ is



Definition: The number of arcs from u in a digraph D is called the *outdegree* $d^+_D(u)$, and the number of arcs towards u is the *indegree* $d^-_D(u)$.

directed path / cycle = orientovaná cesta / kružnice (nebo cyklus)
indegree = vstupní stupeň, outdegree = výstupní stupeň

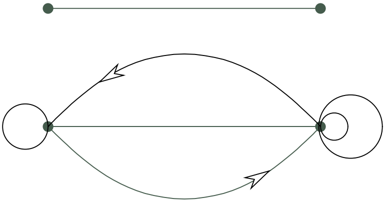
Generalizing graphs even more. . .

- On some other occasions one may want to speak about structures in which more than one edge exist between one pair of vertices, and the edges might have mixed types (undirected or directed, loops). □
- This leads to so called *incidence model* of a (multi)graph in which edges are elements on their own, along with the vertices; as opposed to our default *adjacency model* where only vertices are considered as the entities. □

Definition: A *mixed multigraph* is a triple $M = (V, F, \varepsilon)$ where $V \cap F = \emptyset$ and $\varepsilon : F \to \binom{V}{2} \cup V \cup (V \times V)$ is an *incidence mapping* of the (multi)edges.

In the definition,

- $\binom{V}{2}$ represents the unoriented edges,
- V the unoriented loops, and
- $V \times V$ the oriented edges and loops.



multigraph = multigraf, multiedge = násobná hrana (mezi stejnou dvojicí vrcholů)

1.5 Appendix: The degree sequence of a graph

Definition: The *degree sequence* (called also the score) of a graph G is the collection of degrees of the vertices of G, written in a sequence of natural numbers which is (usually) sorted as nondecreasing or nonincreasing.

In abstract graphs, their vertices usually have no names, and so we have to sort their degree sequence somehow. The particular custom is not important. □

Just to quickly ask, why the sequence $1, 2, 3, 4, 5, 6$ cannot be a degree sequence of a graph?□ (Is the sum even? No. . .)

And what about the sequence $1, 2, 3, 4, 5, 6, 7$? □

Theorem 1.14. *Let $d_1 \le d_2 \le \cdots \le d_n$ be a sequence of natural numbers. There exists a simple graph on n vertices having a degree sequence*

$$d_1,\ d_2, \ldots,\ d_n$$

if, and only if, there exists a simple graph on $n-1$ vertices having the degree sequence

$$d_1,\ d_2, \ldots,\ d_{n-d_n-1},\ \ d_{n-d_n} - 1, \ldots,\ d_{n-2} - 1,\ d_{n-1} - 1\,.$$

degree sequence = posloupnost stupňů neboli skóre grafu

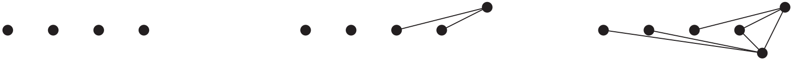
Example 1.15. *Is there a simple graph with degree sequence*

$(1,1,1,2,3,4)$?

Using Theorem 1.14 we modify the sequence to $(1,0,0,1,2)$, □
then sort again as $(0,0,1,1,2)$,
and continue analogically with the next step, arriving at the seq. $(0,0,0,0)$. □

A graph with the last degree sequence clearly exists, and so by the equivalence claim in Theorem 1.14, our graph exists as well. □

How can we construct such a graph? See. . .



□

$(1,1,1,1,2,3,4,6,7)$? □

The first step analogically translates to $(1,0,0,0,1,2,3,5)$,
sorted as $(0,0,0,1,1,2,3,5)$. □
One more step and we get $(0,0,-1,0,0,1,2)$. What does it mean? □

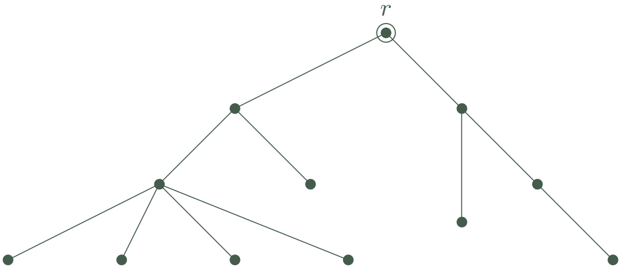
Since the degrees of a graph cannot be negative, a graph with the last sequence does not exist, and neither does the original one!

□

1.6 Appendix: Rooted trees and Tree isomorphism

The introduction of *rooted trees* is motivated primarily by practical applications in which a "tree data structure" has to be grasped at some vertex (the root), and also by historical use of family-trees. Actually, the traditional family-trees motivated much terminology of graph trees. □

Definition 1.16. ***A rooted tree*** is a tree T together with a "highlighted" *root* $r \in V(T)$, i.e. shortly a pair (T, r).

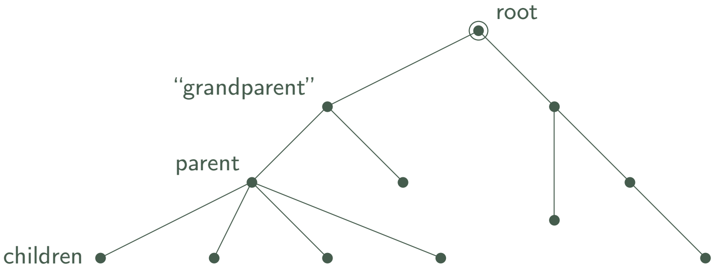


Interestingly, trees in computer science grow top-to-bottom. . .

rooted tree = kořenový strom

Definition: Consider a rooted tree (T, r) and its vertex v. Let u denote the neighbour of v on the unique path towards the root r. Then u is the *parent* of v, and v is a *child* (descendant) of u. □

The root has no "ancestors".



□

Definition: A vertex of degree 1 in a tree is called a *leaf*.

Strictly saying, a root of degree 1 is also a leaf, but we shall use this term carefully for roots to avoid confusion.

Notice that a leaf (which is not a root) has no descendants.

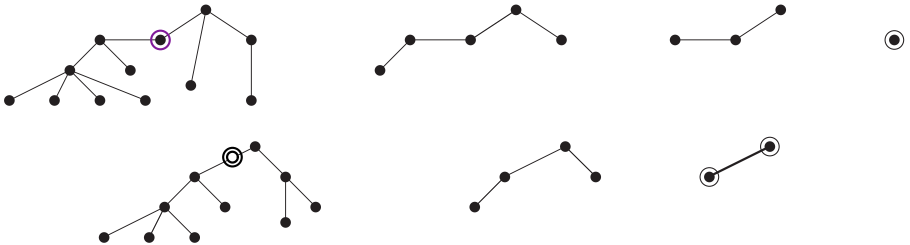
parent = rodič, child = potomek, leaf = list

Center of a tree

Definition: The *center* of a (nonempty) tree T is the vertex or the edge found by the following recursive procedure:

- If T is a single vertex or a single edge, then this (T) is the center. □
- Otherwise, we create a smaller tree $T' \subset T$ by removing all leaves of T at the same time. This T' is nonempty, and we determine the center of T' recursively. The center of T is identical to this center of T'. □

Example 1.17. *The definition of a tree center is illustrated by the two examples:*



□

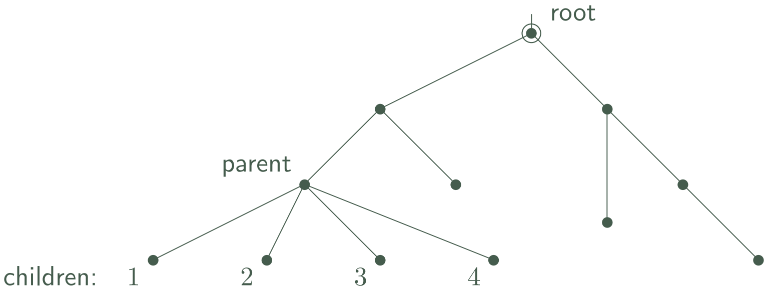
□

center = centrum stromu

Ordered rooted trees

Definition: A rooted tree (T, r) is *ordered* if the orderings of the children of each its vertex are prescribed.
A rooted ordered tree is also called a *planted tree*. □

A rooted ordered tree can be visualized as a tree drawn in the plane without "crossings" and with a marked root. The cyclic ordering of the child edges (against the parent edge / the root mark) then gives the tree ordering of the children.

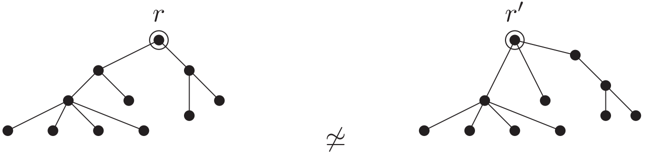


ordered rooted tree = uspořádaný kořenový strom

Tree isomorphism

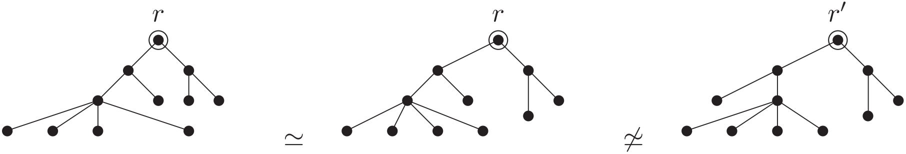
The isomorphism problem of trees, as a special instance, has already been defined and studied on general graphs. Tree isomorphism problem, however, has one additional interesting aspect — that there is an efficient and straightforward solution to it. □

Definition: Two rooted trees (T, r) and (T', r') are *isomorphic* if there is an isomorphism between T and T' such that r is mapped onto r'.



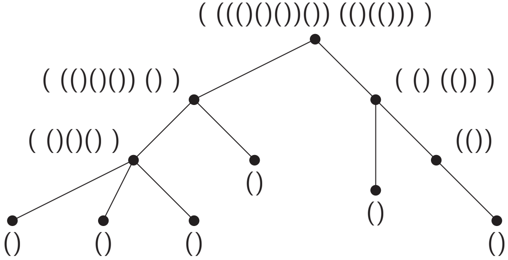
□

Definition: Two rooted ordered trees are isomorphic if there exists a rooted isomorphism between them which preserves the orderings of children at each vertex.



Coding of ordered rooted (planted) trees

Definition:



The *code* of a rooted ordered tree is determined recursively from the codes of the subtrees of its root, concatenated in a prescribed order and enclosed in a pair of parentheses. □

Remark: Instead of ‘(’ and ‘)’, one may use other pair of symbols such as ‘0’ and ‘1’.

Lemma 1.18. *Two rooted ordered trees are isomorphic if and only if their codes are identical as strings.*

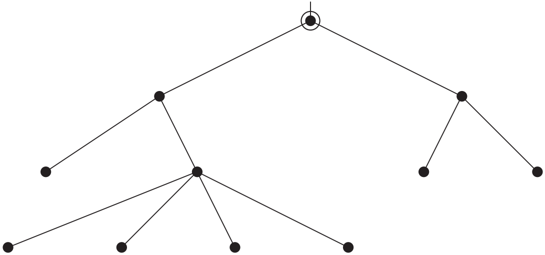
Proof: Easily by induction on the depth... □

Example 1.19. *Draw a planted tree with the following code*

$$(\, (()(()()()())) \, (()()) \,) \, .$$ □

If s is the code of a rooted ordered tree T, then—as easily follows from the definition—T can be drawn using the following procedure:

- Reading the first char '(', we put the pen on the paper (marking the root). □
- Reading every next '(', we draw the edge to the next child of the current vertex. □
- Reading each ')', we return the pen to the parent. In the root, we lift the pen and are finished. □



□

Isomorphism testing for trees

The core idea behind the tree isomorphism testing algorithm is to compare their codes which are rooted in the centers and ordered lexicographically among the children.

Algorithm 1.20. ***Testing isomorphism of two trees.***

```
input < trees $T$ and $U$;
if ($|V(T)| \neq |V(U)|$) return 'Not isomorphic.'; □
(T,r) ← rooted_at_center(T);    (U,s) ← rooted_at_center(U);
k ← minimal_code(T,r);          m ← minimal_code(U,s);
if (k = m as strings) return 'Isomorphic.';
else return 'Not isomorphic.'; □

Function minimal_code(tree X, root r) {
    if ($|V(X)| = 1$) return "()";
    d ← number of components of X\r, i.e. subtrees of r;
    foreach (i ← 1,...,d) {
        Y[i] ← $i$-th connected component of X\r;
        s[i] ← $i$-th neighbour of r, i.e. the root of the subtree Y[i];
        k[i] ← minimal_code(Y[i],s[i]);
    }
    sort as k[1] ≤ k[2] ≤ ... ≤ k[d] lexicographically;
    return "("+k[1]+...+k[d]+")" as string concatenation;
}
```

The proof

A mathematical proof of Algorithm 1.20 is given as follows.

Theorem 1.21. *Let T, U be two trees on the same number of vertices, and let (T', r) and (U', s) be their rooted trees obtained in the first step of Algorithm 1.20 (r, s are the centers of T, U). Then:*

a) T and U are isomorphic if, and only if, (T', r) is isomorphic to (U', s).

b) (T', r) is isomorphic to (U', s) if, and only if,

$$\mathtt{minimal\_code}(\mathtt{T}', \mathtt{r}) = \mathtt{minimal\_code}(\mathtt{U}', \mathtt{s}).$$ □

Proof (sketch): Claim (a) immediately follows from the unique choice of a center. □

Claim (b) is proved by induction on the depth of our rooted trees (T', r) and (U', s). If their depths differ, then they are clearly not isomorphic. On the other hand, two rooted trees of depth 0 are always isomorphic with the code '()'. So let (T', r) and (U', s) be both of depth $\ell > 0$. If their codes in the algorithm are identical, then (T', r) and (U', s) are clearly isomorphic. □

Conversely, if (T', r) and (U', s) are isomorphic, then there exists a "matching" bijection between the subtrees of their roots, and hence the codes of matching subtrees are identical by the inductive assumption. Since the codes of the subtrees are sorted in the same way, we get $\mathtt{minimal\_code}(\mathtt{T}', \mathtt{r}) = \mathtt{minimal\_code}(\mathtt{U}', \mathtt{s})$. □