Discrete mathematics

Petr Kovář & Tereza Kovářová
`petr.kovar@vsb.cz`

VŠB – Technical University of Ostrava

Winter Term 2022/2023
DiM 470-2301/02, 470-2301/04, 470-2301/06

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

The translation was co-financed by the European Union and the Ministry of Education, Youth and Sports from the Operational Programme Research, Development and Education, project "Technology for the Future 2.0", reg. no. CZ.02.2.69/0.0/0.0/18_058/0010212.

About this file

This file is meant to be a guideline for the lecturer. Many important pieces of information are not in this file, they are to be delivered in the lecture: said, shown or drawn on board. The file is made available with the hope students will easier catch up with lectures they missed.

For study the following resources are better suitable:

- Meyer: Lecture notes and readings for an http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2005/readings/" (weeks 1-5, 8-10, 12-13), MIT, 2005.
- Diestel: Graph theory http://diestel-graph-theory.com/ (chapters 1-6), Springer, 2010.

See also http://homel.vsb.cz/~kov16/predmety_dm.php

Lecture overview

Chapter Trees

- motivation
- basic tree properties
- rooted trees
- isomorphism of trees
- spanning trees of graphs

Chapter Trees

Motivation

Among the most common structures in both nature and mathematics are trees (objects with “tree” structure).
There exist a vast amount of objects, that can be described by a “tree”.

- genealogy trees
- evolutionary tree
- electrical circuits
- hierarchical structure (chief and subordinated)
- branching in a search

Common property: no “cycle” in the structure.

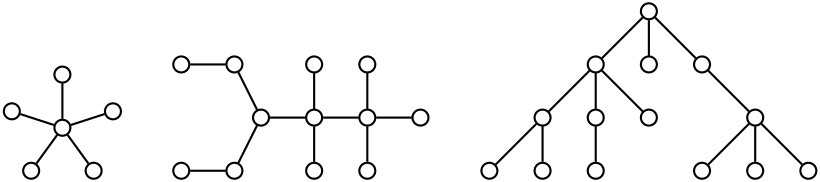
Elementary properties of trees

We say, that a graph is acyclic, if it does not contain a cycle. thus if no its subgraph is isomorphic to a cycle.

Definition

A simple connected graph that acyclis is a tree.

A forest is a graph, whose components are trees.



Note to terminology

- Forest is a (finite simple) acyclic graph.
- Tree is a connected forest.

Seems awkward. . .

Vertices of degree 1 are called leaves.
All other vertices are non-leaf vertices.

Lemma

A tree with more than one vertex contains at least one leaf.

Proof Connected graph with more than one vertex cannot have a vertex of degree 0. Let us take any tree T and some vertex v. Now we construct a longest possible trail S in T starting at v. S starts with any edge from v (such an edge exists, why?). In every consequent vertex u of the trail S

- either u is of degree at least 2 and we can extend S by another edge (notice: if some vertex would repeat in trail S, then S would contain a cycle, that would be a subgraph of tree T, which contradicts the definition of a tree),
- or u is the last vertex of the trail (u is of degree 1),

Since T is finite, we surely find such vertex of degree 1 in any tree T. □

Note

It's easy to prove, that every nontrivial tree T contains at least two leaves.

Questions

- How is called a tree with, precisely two vertices of degree 2 and no vertex with larger degree?
- Does there exist a tree with a vertex of degree k and less than k vertices of degree 1?
- Can you prove the previous assertion?
- How many edges have to be removed from K_n to obtain a tree?

Theorem

A tree on n vertices has precisely $n - 1$ edges.

Proof

We proceed by induction on n.

Basis step: A (trivial) tree with one vertex has $n - 1 = 0$ edges.

Inductive step: Let T be any non-trivial tree on $n > 1$ vertices. By induction hypothesis every tree with less than n vertices has one edge less than vertices.

By the previous lemma T has a vertex of degree 1. By $T' = T - v$ we denote the graph, which arises from T by removing vertex v ("shaving").

- After removing a leaf the graph remains connected (no path between two vertices different from v does not pass through a vertex of degree 1), T' is connected.
- Removing a vertex/an edge no cycle arises, T' is also acyclic.

By induction hypothesis T' has one edge less then vertices, thus T' has $(n - 1) - 1$ edges. Hence the original tree T has one edge more, i.e. $(n - 1) - 1 + 1 = n - 1$ edges.

The claim follows by induction. $\square$

Example

In the database there are 12 objects and 34 relations between the objects. We want the structure of objects draw as a graph in which objects correspond to vertices and relations to edges.
a) Will the resulting graph be a tree?
b) Will the resulting graph always be connected?

a) The resulting graph cannot be a tree, it has to contain cycles. A tree on 12 vertices has precisely 11 edges (relations).
Even if there were 11 relations, we cannot guarantee the resulting graph to be a tree, why?

b) The resulting graph can, but does not have to be connected.
Connectivity depends on the stored structure.
E.g. it could be a graph with one component close to K_9 and three isolated vertices (K_9 has $\binom{9}{2} = 36$ edges, we can remove any 2 edges).

If the graph has 12 vertices and more than 55 edges, the resulting graph has to be connected. K_{12} has 66 edges and is edge 11-connected. After removing any less than $66 - 55 = 11$ edges the graph remains connected.

on proving theorems of the form $A \Rightarrow B$

Suppose A is the premise and B is the conclusion of the theorem.

Direct proof consists of a sequence of valid implications.

$$A = A_0 \Rightarrow A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n = B$$

Indirect proof is a direct proof of the theorem $\neg B \Rightarrow \neg A$, which has the same truth value table as $A \Rightarrow B$.

$$\neg B = A_0 \Rightarrow A_1 \Rightarrow A_2 \Rightarrow \cdots \Rightarrow A_n = \neg A$$

In a proof by contradiction we assume that both the premise A and the negation of the conclusion $\neg B$ are true. By a sequence of valid implications we obtain a contradiction. By a contradiction we mean that both V and its negation $\neg V$ are true simultaneously, which is not possible.

$$A \wedge \neg B \Rightarrow \cdots \Rightarrow V \wedge \neg V$$

Assuming $\neg B$ leads to a contradiction, thus B holds.

Theorem

In a tree there exists exactly one path between every pair of vertices.

Proof By contradiction.
We assume the premise (T is a tree) and the negation of conclusion (there exists a pair of vertices T connected by none or at least two different paths).

Since T is connected (by definition of a tree), there is a path between any pair of vertices u, v. Now if u, v are connected by two different paths, their union is a walk in T and after "deleting" all repeated vertices in this walk we obtain a cycle in T, which contradicts the premise that there is no cycle in T.

The negation of the conclusion leads to a contradiction, thus thus there is precisely one path between any u and v. $\square$

Note

Paths from u to v and the "reversed" path from v to u we consider a single path between u, v.

We know already, that a tree on n vertices contains $n-1$ edges. Adding one new edge

- does not violate connectivity,
- violates the state of being acyclic.

A tree with one additional edge contains a cycle, we show that there is just one such cycle.

Corollary

By adding one (new) edge to a tree (on at least three vertices) we obtain a graph with a single cycle.

Proof Suppose there is no edge uv in a tree T.
By adding edge uv precisely one cycle arises by joining uv and a unique path between u and$v$ in T (unique by previous theorem). $\square$

Note

By adding at least two edges to a tree, the number of emerging cycles depends on where we add the edges.

Given a connected graph G and determining k-connectivity we asked how many edges at least have to be removed from G to obtain a disconnected graph. Now we ask

- how many edges at most can be removed from G to obtain a connected graph or, conversely
- how many edges at least have to remain in G for G to remain connected.

Trees are graphs that are both connected and no edge can be removed without loosing connectivity.

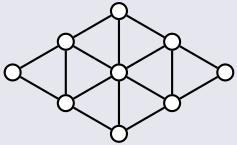
Theorem

A tree is the *minimum connected* graph (on a given set of vertices).

Proof A tree is connected by definition. If a graph contains a cycle, it remains connected also after removing any edge of this cycle. Thus the minimum connected graph is a tree.
Conversely, if after removing edge uv form a tree T the resulting graph remains connected, then between u, v in T would exist two paths: u, v-path in $T \setminus uv$ and edge uv. This contradicts previous theorem. Thus, tree is the minimum connected graph on a given set of vertices. $\square$

Example

At most how many edges can be removed from the graph G so that the graph remains connected?



Graph G.

By the previous theorem the resulting graph has to be a tree.
The graph G has 9 vertices and 16 edges, thus by the theorem on the number of edges in a tree at most 8 edges can be removed.

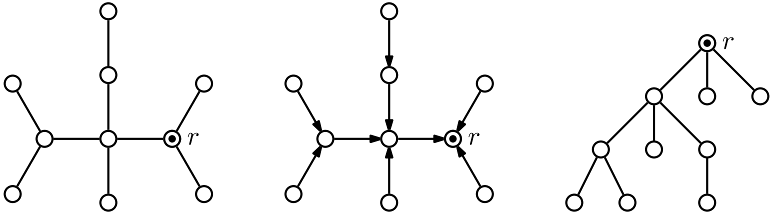
Moreover, one can remove such 8 edges that both removed edges and the remaining edges for a connected factor. Can you find such 8 edges?

Rooted trees

Is certain instances of "trees" it is convenient to select a vertex, called root, (as the "start" of data). Rooted trees have their origins also in family trees, (Tiggers "family tree") which implied terminology.

Definition

A rooted tree is a tree T along with one significant root vertex $r \in V(T)$, denoted by (T, r), we say tree T with root r.

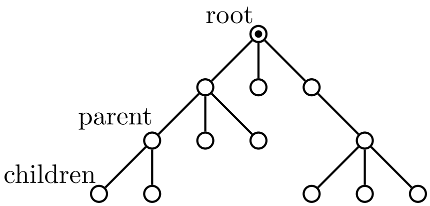


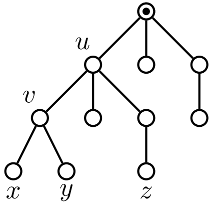
There is a difference between a „tree" and a „rooted tree", which has some extra information.

Root will be drawn on top.

Definition

Take a rooted tree (T, r) and a pair of adjacent vertices u, v, such that u is the neighbor of v on the path to r. Then u is called the parent of v and v is called the child of u.





Sometimes we will use other terms as „grandfather“, „sibling“, . . .

Notice: by choosing a different root the parent-child relationship can swap.

Definition

Vertices without children in non-trivial graphs are called bottom vertices.

Notice the bottom vertices are leaves, but not all leaves are necessarily bottom vertices.

Definition

Center of a tree T is the vertex or edge in T determined by the following algorithm:

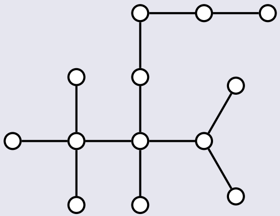
1. If a tree T has a unique vertex v, then v is the center of T.
   If a tree T has two vertices, its center is the edge joining the two vertices.
2. Otherwise we create a (smaller) tree $T' \subset T$ by deleting all leaves of T (shaving). It is obvious, that T' is not empty; proceed by step 1.

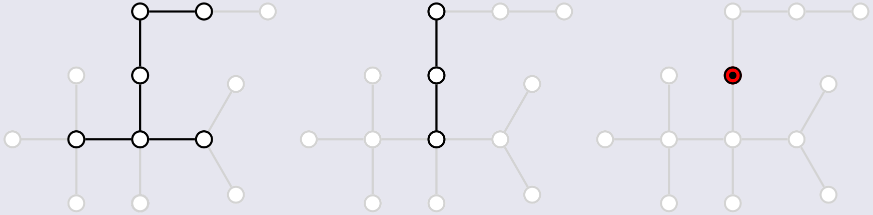
The center of T' obtained by recursion is also the center of T.

The process of removing leaves is called shaving.

Example

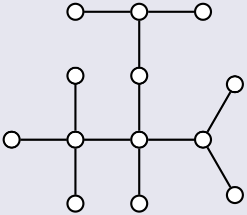
Find the center of tree T_1.

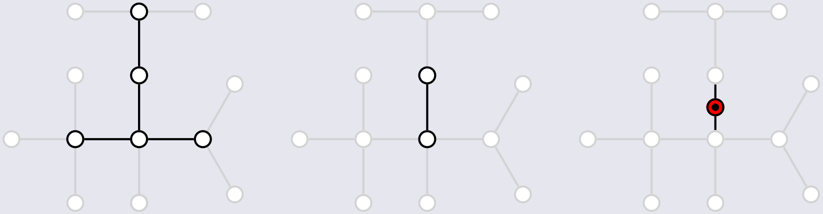




Example

Find the center of tree T_2.





Notice: we added a new vertex (and two edges instead of one edge).

The root and the center

The root can be any vertex in a given tree; the root does not have to be the center.

If the root of a tree has to be determined *uniquely*, then *center* is the best candidate (it is determined uniquely).

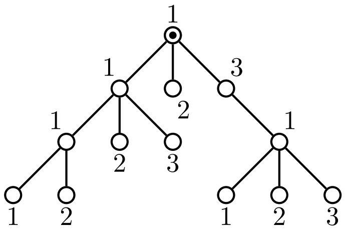
If the center is an edge add a new vertex onto the central edge so that it "splits" the edge into two.

Ordered rooted trees

Another information assigned to rooted trees is the ordering of children of every vertex (ordering the ancestors in one generation by their birth date).

Definition

The rooted tree (T, r) is ordered, if for every vertex is the order of its children determined uniquely ("from left to right").



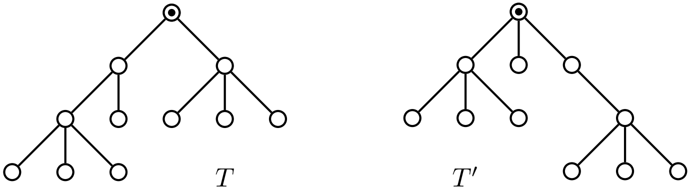
Formally: ordered rooted tree is (T, r, f), where T is a tree and r its root. Function $f : V(T) \to \mathbb{N}$ assigns each vertex it order among siblings $1, 2, \ldots, k$.

Isomorphism of trees

The notion of isomorphism of trees is a special case of isomorphism of graphs. Two trees are isomorphic, if they are isomorphic as graphs.
Recall that no fast algorithm for deciding whether two general graphs are isomorphic is known. Trees are such a special class of graphs that, for trees such algorithm does exist!
Before we give the algorithm, we have to introduce several terms.

Definition

Two rooted trees (T, r) and (T', r') are isomorphic if there exists an isomorphism of T and T', that takes root r onto root r'.

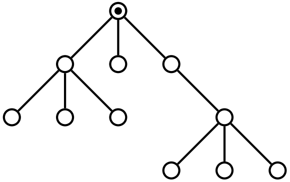


Two isomorphic trees that are not isomorphic as rooted trees.

Notice: the root is different in T and T'.

Definition

Two ordered rooted trees are isomorphic, if there exists an isomorphism of rooted trees, such that it *preserves the order of children* of every vertex.



Two isomorphic rooted trees that are not isomorphic as ordered rooted trees.

Notice: the order of children of the root differs.

Definition

Subtree of a vertex u of a given rooted tree (T, r) is each component of the graph $T - u$, which contains some child x of u.

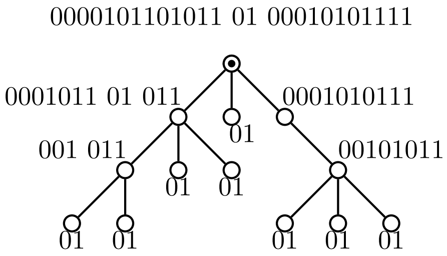
Each subtree of vertex u is again a (rooted) tree.

Encoding ordered rooted trees

To every ordered rooted tree we can easily assign a string of 0 and 1 which uniquely determine the tree.

Definition

Code of an ordered rooted tree is constructed recursively by joining codes of all subtrees of the root, ordered in a particular (uniquely chosen) ordering and enclosed in a pair of 0 and 1 (see figure).



Coding a rooted tree.

Note

Instead of “0” and “1” one can use, e.g. “(” and “)” or „A“, „B“.

Ordered rooted trees given by their code

We described how to obtain a code for a given rooted tree tree.
Now we show the reverse: how to draw a rooted tree given by its code.

Lemma

Take the code of an ordered rooted tree. The corresponding tree can be drawn by the following algorithm:

- when reading “0” at the beginning put the pen on the paper, draw the root vertex,
- when reading another “0” draw an edge to a child vertex of the current vertex,
- when reading “1” return to the parent of the current vertex or lift the pen if the current vertex is the root.

Notice: not every sequence of 0 and 1 is a code of some tree (see discussion).

Minimum code

We can consider tree codes as strings and we can order these strings uniquely, e.g. lexicographically.

Suppose the symbol 0 precedes symbol 1 in the dictionary.
E.g. the string 000111 precedes codes 001011, 0011, and 01.

One has to distinguish *code of an ordered rooted tree* and *minimum code of a rooted tree*:

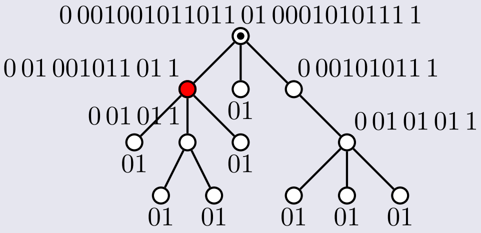
- drawing a tree given by the code of an ordered rooted tree (T, r), we obtain (T, r) again,
- drawing a tree given by the minimum code, the order of children can differ from the order in (T, r).

We say the rooted tree (T, r) „was reordered".

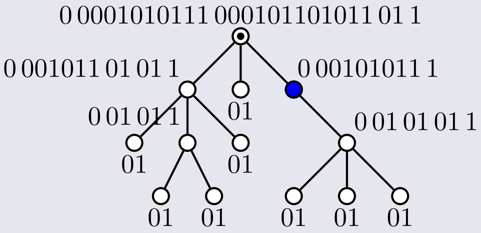
Note

The upper bound on time complexity for finding a minimum code of one tree is $O(n^3)$.

Example



The code of an ordered rooted tree.



The minimum code of an ordered rooted tree.

When determining isomorphism of two arbitrary trees we

- find the center of each tree,
- the center of each tree we choose as the root,
- we find the minimum codes (we order the codes of the children lexicographically by their codes)
- we use the following Lemma which guarantees the uniqueness of each code.

Lemma

Two ordered rooted trees are isomorphic if and only if their codes, obtained as described above, are the same strings.

The process results in the algorithm described below.

Algorithm Determining isomorphism of trees

Algorithm determines if two trees T and U are isomorphic ($T \overset{?}{\simeq} U$)

```
// Let T, U be two trees with the same number of vertices.
Input < trees T and U;
for (X=T,U) {
   // find the centers of U, T
   x = center(X);
   if (x is one vertex)
      r = x;
   else
     add new vertex r, replace edge x=uv by edges ru, rv;
   k[X] = minimum_code(X,r);
}
if ((|V(T)|==|V(U)|) && (k[T]==k[U] as strings))
   print("Trees T and U are isomorphic.");
else
   print("Trees T and U are not isomorphic.");
exit;
```

Algorithm ...continued (finding the minimum code)

Function `minimum_code(X,r)` finds for tree X with root r (lexicographic) minimum code.

```
// the input is a rooted tree (or a subtree)
input < rooted tree (X,r);

function  minimum_code(tree X, vertex r) {
   if (X has one vertex)
      return "01";
   Y[1...d] = {connected components X-r, subtrees without r};
   s[1...d] = {roots of subtrees Y[] in corresponding order};
               // roots are the children of r
   for (i=1,...,d)
       k[i] = minimum_code(Y[i],s[i]);
   sort lexicographic so that k[1] <= k[2] <= ... <= k[d];
   return "0"+k[1]+...+k[d]+"1";
}
```

Functions recursively constructs the minimum code of tree X.

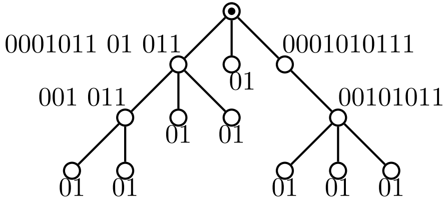
Note

Notice that in the Algorithm we check whether both have the same number of vertices.
For example paths P_{2n} and P_{2n+1} are not isomorphic, but since the center of path P_{2n} is the „middle“ edge, by finding the center we add a new vertex to this edge and obtain a second path P_{2n+1}. Without checking the number of vertices the algorithm could give a wrong answer.

Questions

- Is the following code a minimum code? Why?
- How would a minimum code look like?

0000101101011 01 00010101111



Spanning tree

Recall the following terms:

- subgraph, factor
- connected and disconnected graph
- labeled graph, graph labeling

Definition

A spanning tree of a connected graph G is such a factor of G, which is a tree. (Factor of G is a subgraph, which contains all vertices of G.)
Weight of a spanning tree in a labeled graph G is the sum of labels of all edges of the spanning tree.

We say "edge label" and "spanning tree weight".

The importance of "spanning trees" lies in the minimality with respect to number of edges, while connectivity is preserved.
(we have a Theorem about the minimum connected subgraph)

The label of every edge can differ, we obtain:

Minimum spanning tree problem (MST)

Given a connected labeled graph G with non-negative edge labels w. The task is to find such a spanning tree T of G, which has among all spanning trees the minimum weight. Formally

$$MST = \min_{\text{span.tree}\, T \subseteq G} \left(\sum_{e \in E(T)} w(e) \right).$$

Questions

- How many edges has a spanning tree of a connected graph with n vertices?
- Is it possible to find a minimum spanning tree in a graph with negative labels?
- Is every connected factor with minimum edge labels a spanning tree?

We present several algorithms for finding a minimum spanning tree in a non-negatively labeled graph.

Algorithm Greedy minimum spanning tree algorithm

We have a labeled graphs G with non-negative labels w of edges. By m we denote the number of edges of G.

- We order the edges of G in an non-decreasing order according their labels:

$$w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m).$$

- We start with an empty set of edges $T = \emptyset$ for the spanning tree.
- For $i = 1, 2, \ldots, m$ we take the edge e_i and if by adding it to the set T no cycle (induced by $T \cup \{e_i\}$) originates, we include e_i into T. Otherwise we "discard" e_i.
- At the end T contains all edges of a minimum spanning tree of graph G with labels w.

We show that the algorithm works correctly.

Theorem

The greedy algorithm finds a minimum spanning tree of a connected graph.

Proof By contradiction. Let T be the set of edges obtained by the Algorithm. Suppose that $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. Let T_0 by the set of edges of such a minimum spanning tree (multiple spanning trees can have the same weight), which matches T in the most first edges. If $T_0 = T$, algorithm works correctly.

Suppose now that $T_0 \neq T$ and we obtain a contradiction. Thus we show that $T_0 \neq T$ cannot occur.

By $j > 0$ denote such an index, that the sets T_0 and T match in the first $j-1$ edges $e_1, \ldots, e_{j-1}$, but they do not match in edge e_j. Thus $e_j \in T$, but $e_j \notin T_0$. (According the algorithm $e_j \notin T$ and $e_j \in T_0$ cannot happen.) graph $T_0 \cup \{e_j\}$ contains the graph with edges precisely one cycle C. Cycle C cannot be a subgraph of the spanning tree T, thus there is an edge e_k in C, such that $e_k \notin T$ and $k > j$. Since $w(e_k) \geq w(e_j)$, the spanning tree with edges $T' = \left(T_0 \setminus \{e_k\}\right) \cup \{e_j\}$ (swapping edges e_k and e_j) does not have higher weight than T_0, but it matches T in more first edges! This is a contradiction with the choice of T_0. $\square$

This greedy algorithm was introduced first by Kruskal in 1956. But it is known that Kruskal continued the work of a Czech mathematician Otakar Borůvka.
Already in 1926 solved Borůvka the question of building an optimal electrical network in southern Moravia and described a very similar algorithm in great detail using matrices.

Algorithm Borůvka's minimum spanning tree algorithm

Suppose G is a positively weighted graph with edges labeled by pairwise different labels.
At the beginning we order the edges according their increasing labels $w(e_1) < w(e_2) < \ldots < w(e_m)$.
The spanning tree we construct by adding edge e_i (for $i = 1, 2, \ldots, n$), if no cycle originates by adding e_i.

In response to Borůvka's work Vojtěch Jarník designed in 1929 a similar algorithm.

The Jarník's algorithm is known as Prim's algorithm (1957).

Algorithm Jarník's minimum spanning tree algorithm

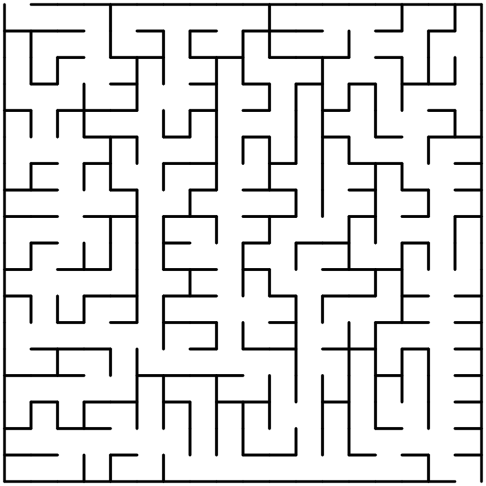
We do not order the edges. We construct the minimum spanning tree starting from any vertex. In every step we choose the edge with the smallest label with one end-vertex among the vertices in the already constructed subgraph and the other end-vertex among remaining vertices.

Notice:

- in Jarník's algorithm we need not to sort the edges,
- we do not need to check whether adding an edge produces a cycle, we save time.

Examples of the Greedy algorithm and Jarník's algorithm are given at `http://homel.vsb.cz/~kov16/predmety_dm.php`

MSP algorithms can be used for constructing labyrinths.



Labyrinth constructed using Jarník's algorithm.

For details see textbook „Úvod do teorie grafů“ (in Czech) or on-line.

Next lecture

Chapter Graph colorings and graph drawing

- motivation
- graph coloring
- drawing graphs in a plane
- recognizing planar graphs
- map coloring and planar graphs coloring