# Combinatorial Optimization: Mutual Relations among Graph Algorithms

EVA MILKOVÁ
Department of Informatics and Quantitative Methods
University of Hradec Králové
Rokitanského 62
CZECH REPUBLIC
eva.milkova@uhk.cz http://lide.uhk.cz/fim/ucitel/milkoev1/

*Abstract:* - The Theory of Graphs is a wonderful, practical discipline. Informatics has played a big part in its development, and these two fields are strongly interconnected. This can, perhaps, mainly be seen in the design of computer algorithms. On the one hand, there are many methods which can be used for solving the same problem, while on the other hand, using effective modifications of one algorithm, we can devise methods of solving various other tasks. To educate students in the area close connected with Graph Theory and Computer Science, called as Combinatorial or Discrete Optimization, it is important to make them familiar with certain algorithms in contexts to be able to get deeper into each problem and entirely understand it. In the paper we present just a few ideas that have proved successful in teaching and learning this quite young part of mathematics.

*Key-Words:* - Graph Algorithms, Minimum Spanning Tree Problem, Breadth-First-Search, Depth-First-Search, Dijkstra's Algorithm, Maze Problem, Eulerian Graph

## 1 Introduction

Mathematics is one of the oldest science however the area known as Combinatorial or Discrete Optimization close connected with Graph Theory and Computer Science is quite young.

When we deal with a particular problem we try to examine it from more than one point of view if possible and discuss various approaches to its solution. On the one hand, there are many methods which can be used for solving the same problem, while on the other hand, using effective modifications of one algorithm, we can devise methods of solving various other tasks.

In this paper ***we discuss mutual relationships between solutions to some known problems***. We devote attention to the well-known Minimum Spanning Tree Problem, including Jarník's (Jarník's-Prim's resp.) approach to it, at first. We meet readers with different descriptions of some discussed solutions as well.

Then we discuss the relationship between Dijkstra's algorithm finding the shortest path and the Jarník's-Prim's solution to the above mentioned Minimum Spanning Tree Problem.

It is followed by illustration of how the most used searching algorithms, Breadth-First-Search and Depth-First-Search, are also connected with the Jarník's algorithm. We describe the properties of the Breadth-First-Search Tree and Depth-First-Search Tree and mention how the properties of the Search Trees influence our approach to the solutions of the other commonly used algorithms.

Finally we deal with the maze problem. We remind three approaches to the solution of this problem, Trémaux, Tarry, and Edmonds-Johnson algorithms, their mutual relationship and relation of Edmonds-Johnson algorithm to the problem of how to find Eulerian trail in an Eulerian graph.

At the end of the paper we mention one multimedia program that serves as a visual representation of basic graph-concepts and graph-algorithms using a colouring process on graphs created within the program and emphasize its advantages for enhancing teaching and learning the discussed topic.

## 2 Minimum Spanning Tree Problem

In the contemporary terminology the Minimum Spanning Tree (MST in short) problem can be formulated as follows (see [1]):
Given a connected undirected graph $G = (V, E)$ with $n$ vertices, $m$ edges and real weights assigned to its edges (i.e. $w$: $E \rightarrow$R). Find among all spanning trees

of $G$ a spanning tree $T = (V, E')$ having minimum value $w(T) = \Sigma(w(e); e \in E')$, a so-called minimum spanning tree.

"The Minimum Spanning Tree problem is generally regarded as a cornerstone of Combinatorial Optimization. Its importance and popularity stem from several reasons. The MST problem may be efficiently solved for large graphs by several algorithms. It has wide application. Methods for its solution have produced important ideas in modern combinatorics and have played central role in the design of graph algorithms."[2]

First formulation of the problem was given in 1926 by the Czech mathematician Otakar Borůvka. (Remark: Otakar Borůvka was introduced to the problem by his friend, Jindřich Saxel, an employee of the West_Moravian Powerplants. It was at that time that electrification of the south and west parts of Moravia was beginning, and Borůvka was asked for help in solving the problem. The challenge was how and through which places to design the connection of several tens of municipalities in the Moravia region so that the solution was as short and consequently as low-cost as possible. Borůvka not only correctly stated this problem but also solved it in the papers [3] and [4]).

## 2.1 Borůvka's algorithm

There are various descriptions of Borůvka's solution in most of the modern textbooks. We introduce *two algorithms* solving the problem as an edge-colouring process (see [1], [2]).(Remark: The survey of the works devoted to the MST problem until 1985 is given in the article [5] and this historical paper is followed up in the article [2] *Otakar Borůvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history*.)

It is necessary to remember that Borůvka's solution of the MST problem presumes distinct edge-weights in the given graph. However, this condition does not restrict the universality of the problem (e.g. we can list all edges and in the case that two edges are equal weights the first on our list we consider as the smaller one.)

Borůvka's algorithm – first description

1. Initially all edges of the graph $G$ are uncoloured and let each vertex be a trivial blue tree.

2. Repeat the following colouring step until there is only one blue tree.

*Colouring step*: For each blue tree $T$, select the minimum-weight uncoloured edge incident to $T$ (i.e. edge having one vertex in $T$ and the other not). Colour all selected edges blue.

3. Blue coloured edges form the unique minimum spanning tree.

(Remark: The distinct edge-weights guarantee that the Borůvka's solution finishes by gaining the unique blue minimum spanning tree of $G$.)

Borůvka's algorithm – second description

1. *Coloring*: For each vertex $v$ of the given graph $G$ we color blue the minimum-weight edge incident to $v$.

2. *Contraction*: We replace each blue tree by a single vertex. In this procedure we eliminate loops (i.e. edges with both ends in the same blue tree) and all the parallel edges (i.e. edges between the same pairs of blue trees) with the exception of the lowest weight edge.

3. We apply the algorithm recursively to find the blue spanning tree $T'$ of contracted graph.

The minimum spanning tree $T$ is formed by the contracted blue edges together with the edges of $T'$.

On figures Fig.1 – Fig.5 second description of Borůvka's algorithm of the MST problem is illustrated there using the following graph $G_1$.

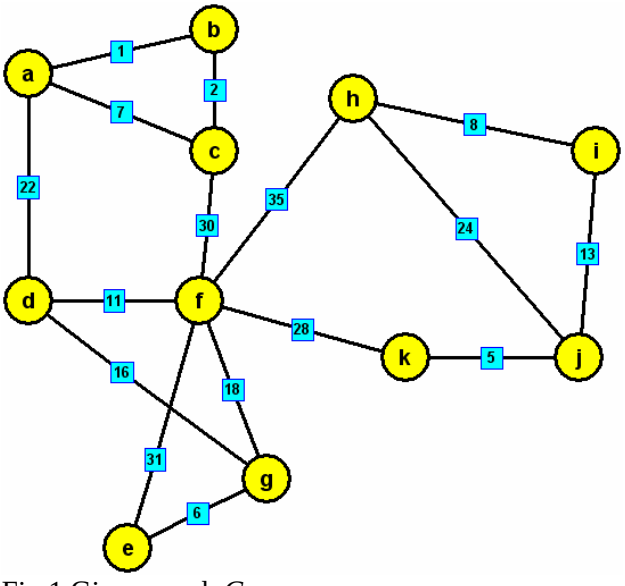


Fig.1 Given graph $G_1$

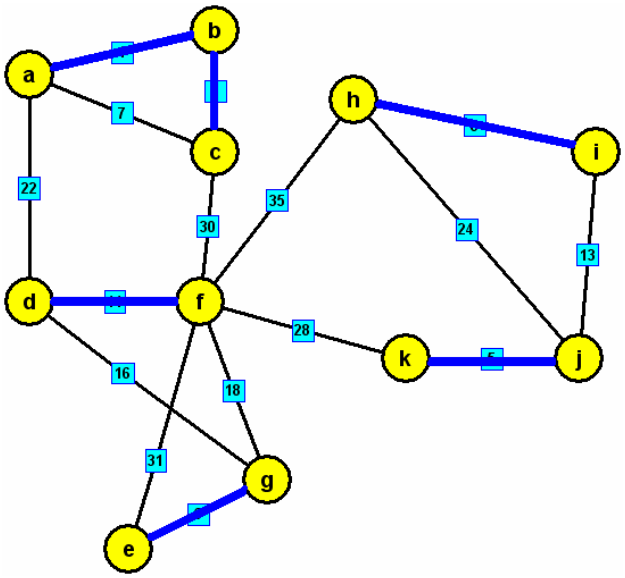


Fig.2 *Coloring*

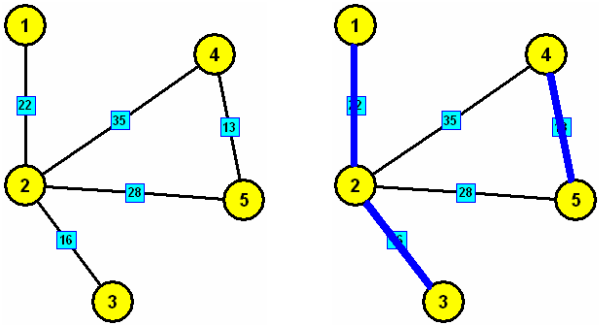


Fig.3 *Contraction* and *Coloring*

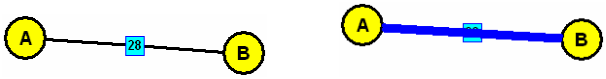


Fig.4 *Contraction* and *Coloring*

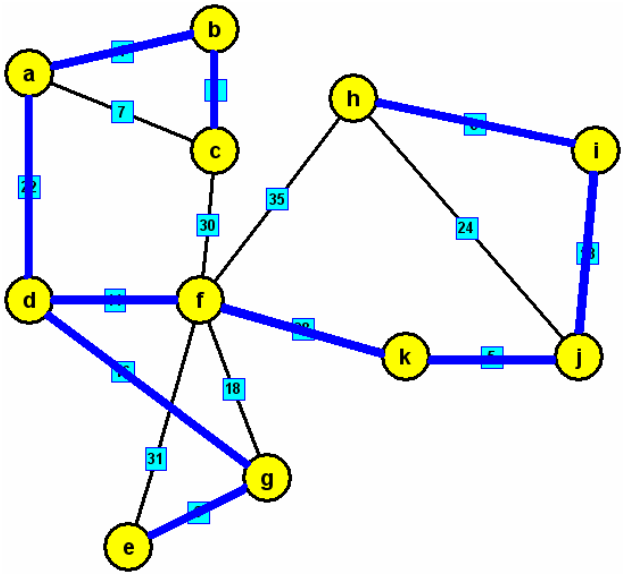


Fig.5 The found unique minimum spanning tree of the given graph $G_1$

## 2.2 Jarník's algorithm

Another Czech mathematician, Vojtěch Jarník, quickly realized the novelty and importance of the problem after reading Boruvka's paper. However the solution seemed very complicated to him. He started to think about another solution and soon afterwards wrote a letter to Otakar Borůvka in which he suggested a much easier method and consequently he published it in the article [6]. In the present terminology we can describe it as follow [1].

Jarník's algorithm

1. Initially all vertices and edges of the graph $G$ are uncoloured. Let us choose any single vertex and suppose it to be a trivial blue tree.
2. At each of $(n - 1)$ steps, colour the minimum-weight uncoloured edge, having one vertex in the blue tree and the other not, blue. (In case, there are more such edges, choose any of them.)
3. The blue coloured edges form a minimum spanning tree.

Fig. 6 illustrates four first steps of Jarník's solution of MST problem on the graph $G_1$ represented by the Fig.1 supposing the vertex *a* be the initial trivial blue tree. (Remark For clearer illustration of the consecutively obtained blue trees we also denote vertices by the color blue.) Obviously, the whole solution is the same as that on the Fig.5.

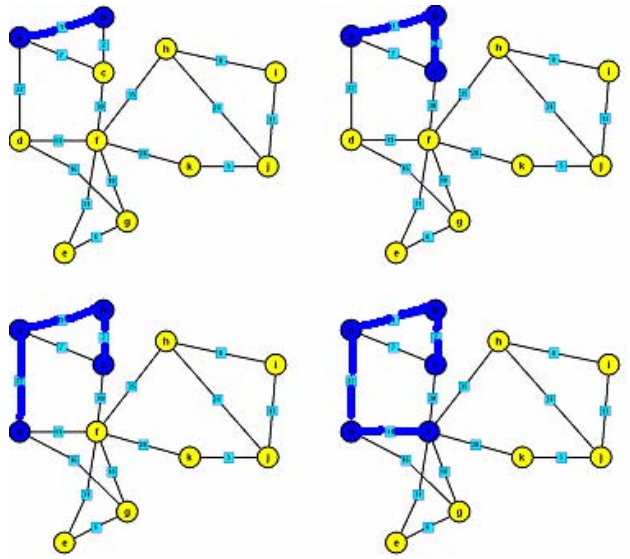

Fig.6 Jarník's solution of the MST problem

Much later and unaware of Jarník's solution, R.C. Prim created the same solution as Jarník, during the time of newly developing field, computer

science. He used a more detailed implementation suitable for computer processing in the paper [7].

To use the following description of his solution let us *consider weights w(e) assigned to edges of a given graph as* ***distances***. (see [1])

Jarník's-Prim's algorithm

1. Let us choose any single vertex $a$ and suppose it to be a blue tree. Put the value $(0, a)$ by the vertex $a$. By each vertex $v$ which doesn't belong to the blue tree, the actual information $(f(v), u)$ is saved, describing the nearest distance $f(v)$ between the vertex $v$ and the blue tree (from the vertex $u$). Thus initially by each vertex $v \neq a$ put the value $(w(\{a, v\}, a)$ if $v$ is a neighbor of the vertex $a$ and the value $(\infty, a)$ if $v$ is not a neighbor of the vertex $a$.

2. At each of $(n - 1)$ steps take the following commands:

- choose a vertex $z$ with the actual information $(f(z), t)$ such that $f(z) = \min\{f(v);\ v$ doesn't belong to the blue tree$\}$,
- colour the corresponding edge $\{z, t\}$ blue,
- by each neighbor $v$ of the vertex $z$, change the value $(f(v), u)$ to the value $(w(\{z, v\}), z)$ in the case that $w(\{z, v\}) < f(v)$.

3. The blue coloured edges form a minimum spanning tree.

Using adjacency matrix of the graph $G_1$ on the Fig.1 let us illustrate Jarník's-Prim's solution of MST problem (see Fig.7).

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| a | | 1 | 7 | 22 | | | | | | | |
| b | 1 | | 2 | | | | | | | | |
| c | 7 | 2 | | | | 30 | | | | | |
| d | 22 | | | | | 11 | 16 | | | | |
| e | | | | | | 31 | 6 | | | | |
| f | | | 30 | 11 | 31 | | 18 | 35 | | | 28 |
| g | | | | 16 | 6 | 18 | | | | | |
| h | | | | | | 35 | | | 8 | 24 | |
| i | | | | | | | | 8 | | 13 | |
| j | | | | | | | | 24 | 13 | | 5 |
| k | | | | | | 28 | | | | 5 | |

Fig.7 Adjacency matrix of the graph $G_1$

a (0,a)
b (**1**,a)
c (7,a) (**2**,b)
d (22,a)(22,a)(**22**,a)
e (∞,a) (∞,a) (∞,a) (∞,a) (31,f)(**6**,g)
f (∞,a) (∞,a) (30,c)(**11**,d)
g (∞,a) (∞,a) (∞,a) (16,d)(**16**,d)
h (∞,a) (∞,a) (∞,a) (∞,a) (35,f) (35,f) (35,f) (35,f) (24,j) (**8**,i)
i (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (**13**,j)
j (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (**5**,k)
k (∞,a) (∞,a) (∞,a) (∞,a) (28,f) (28,f) (**28**,f)

Fig.8 Jarník's-Prim's solution, blue coloured edges are {a,b},{b,c},{a,d},{g,e},{d,f},{d,g},{i,h},{j,i}, {k,j},{f,k}

## 2.3 Kruskal's algorithm

The third classical solution of the MST problem was discovered by J. B. Kruskal [8]. We describe it as an edge-colouring process as well and illustrate it again on the graph $G_1$ represented by the adjacency matrix on Fig.7, this time with help of disjoint sets of vertices (each set will represent one blue tree) using operation *find*$(x)$ and *union*$(M, N)$ (see Fig.9).

Kruskal's algorithm

1. Initially all edges of the graph $G$ are uncoloured. Let us order the edges in nondecreasing order by weight. Let each vertex be a trivial blue tree.

2. At each of $m$ steps decide about colouring exactly one edge if it is coloured by blue colour or not. The edges are examined in a sequence defined by above-mentioned ordering. The chosen edge is coloured blue if and only if it doesn't form a circle with the other blue edges.

3. The blue coloured edges form a minimum spanning tree.

| sorted edges | disjoint sets of vertices {a} {b} {c} {d} {e} {f} {g} {h} {i} {j} {k} | colour blue |
|---|---|---|
| {a,b} | {**a**} {**b**} {c} {d} {e} {f} {g} {h} {i} {j} {k} | YES |
| {b,c} | {a, **b**} {**c**} {d} {e} {f} {g} {h} {i} {j} {k} | YES |
| {k,j} | {a, b, c} {d} {e} {f} {g} {h} {i} {**j**} {**k**} | YES |
| {e,g} | {a, b, c} {d} {e} {f} {g} {h} {i} {j, k} | YES |
| {a,c} | {**a**, b, **c**} {d} {e} {f} {g} {h} {i} {j, k} | NO |
| {h,i} | {a, b, c} {d} {e} {f} {g} {**h**} {**i**} {j, k} | YES |
| {d,f} | {a, b, c} {**d**} {e} {**f**} {g} {h, i} {j, k} | YES |
| {i, j} | {a, b, c} {d, f} {e} {g} {h, **i**} {**j**, k} | YES |
| {d,g} | {a, b, c} {**d**, f} {e} {**g**} {h, i, j, k} | YES |
| {f,g} | {a, b, c} {d, **f**, **g**} {e} {h, i, j, k} | NO |
| {a,d} | {**a**, b, c} {**d**, f, g} {e} {h, i, j, k} | YES |
| {h,j} | {a, b, c, d, f, g} {e} {**h**, i, **j**, k} | NO |
| {f,k} | {a, b, c, d, **f**, g} {e} {h, i, j, **k**} | YES |
| {c,f} | {a, b, **c**, d, **f**, g, h, i, j, k} {e} | NO |
| {e,f} | {a, b, c, d, **f**, g, h, i, j, k} {**e**} | YES |
| {f,h} | {a, b, c, d, **f**, g, **h**, i, j, k, e} | NO |

Fig.9 Kruskal's solution of MST problem on $G_1$

**2.4 Summary**

Comparing the solutions written above we can characterize the basic difference as follows:

In Borůvka's solution, at each step the union of all the blue trees being the nearest to one another is demonstrated.

Jarník's solution at each of ($n$-1) steps spreads the only blue tree that contains the initial vertex by the nearest vertex.

Kruskal's solution connects the two nearest blue trees in one blue tree at each step in which one edge is coloured blue.

Two different descriptions of Borůvka's algorithm allow a better insight to his solution. Both descriptions of Jarník's solution are important for description of other algorithms. Namely, from the first algorithm (see 2.2, Jarník's algorithm) we can proceed to the searching methods (see section 4) and from the second algorithm (see 2.2, Jarník's-Prim's algorithm) we can continue to the algorithm finding the shortest path between two vertices of a connected undirected *non-negative*-weighted graph (see the following section 3).

Students should be aware that, so far, all known methods solving MST problem make use of the various combinations of the following two dual properties of trees ([2], [9]).

*Cut rule*: The optimal solution $T$ to MST problem contains an edge with minimal weight in every cut.

*Circle rule*: The edge of the circle $C$ whose weight is larger than the weights of the remaining edges of $C$ cannot belong to the optimal solution $T$.

Thus Borůvka's and Jarník's (Jarník's-Prim's) solutions are based on the cut rule only. Kruskal's algorithm combines both rules according to the initial order of edges, and thereby points out the blue one.

Kruskal's provided also the following similar elegant solution concerning the circle rule.

Kruskal's dual algorithm

1. Initially all edges of the graph $G$ are uncoloured. Let us order the edges in nonincreasing order by weight. Let each vertex be a trivial blue tree.
2. At each of $m$ steps decide about colouring exactly one edge if it is coloured by red colour or not. The edges are examined in a sequence defined by above-mentioned ordering. The chosen edge is coloured red if and only if it belongs to a circle that does not have a red coloured edge.
3. Uncoloured edges form a minimum spanning tree.

We shall not discuss here the complexity of the mentioned algorithms except to note that:

1. Borůvka's algorithm, thanks its suitability for parallel computation, is the basis of the fastest known algorithms solving the MST problem,
2. "Methods solving the MST problem using mainly the circle rule seem to be less efficient"[9]

# 3 Shortest path - Dijkstra's algorithm

In the second section above we discussed the shortest connection of all $n$ vertices in a weighted connected undirected graph, considering the given weights as distances between two vertices. Could the solution of the MST problem also serve as a solution to the problem of how to find the shortest path from one vertex to another in a connected undirected graph with $n$ vertices and non-negative weights assigned to its edges?

The answer is no. See the shortest path from the vertex $a$ to the vertex $h$ in the graph $G_1$ (see Fig.1) going through vertices $b$, $c$ and $f$. Its length is 68 while the length of the path between the same vertices in the blue minimum spanning tree of the graph $G_1$ (see Fig. 5) is 87!

However, it seems that there must be a relationship between the two problems.

Let us consider Jarník's-Prim's algorithm again (see section 2.2) and imagine the only modification: At each step of the algorithm, by each vertex $v$ which doesn't belong to the blue tree, save the actual information describing *the nearest distance between the vertex v and the* ***initial vertex*** $a$ (instead the nearest distance between the vertex $v$ and the blue tree). In this way we really get the correct solution, namely the solution found in 1950's by E.W.Dijkstra (see [10]) applied on a connected directed *non-negative*-weighted graph.

As a lucid illustration of the relation see the Fig.10 where the Dijkstra's algorithm finding the shortest path from the vertex $a$ to the vertex $h$ in the graph $G_1$ (represented by adjacency matrix on Fig.7) is described. Compare the process with the process on the Fig.8.

a (0,a)
b (**1**,a)
c (7,a) (**3**,b)
d (22,a)(22,a)(**22**,a)
e (∞,a) (∞,a) (∞,a) (∞,a) (64,f)(**44**,g)
f (∞,a) (∞,a) (33,c)(**33**,c)
g (∞,a) (∞,a) (∞,a) (38,d)(**38**,d)
h (∞,a) (∞,a) (∞,a) (∞,a) (68,f) (68,f) (68,f) (68,f) (**68**,f)
i (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (79,j)
j (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (∞,a) (**66**,k)
k (∞,a) (∞,a) (∞,a) (∞,a) (61,f) (61,f) (**61**,f)

Fig.10 Dijkstra's solution (a ,b, c, f, h) to the shortest path between two vertices in the graph $G_1$

# 4 Graph Searching

A consecutive searching of vertices and/or edges occurs either directly or indirectly in almost all graph algorithms (see e.g. recent WSEAS transactions papers [11], [12], [13]).

Breadth-First-Search and Depth-First-Search are two of the most used graph search algorithms. These methods are usually explained on a rooted tree at first and then the used ideas are extended to graphs.

In the following text we present mutual relations between each of these algorithms and Jarník's solution of MST problem to get Breadth-First-Search Tree and Depth-First-Search Tree. Then we describe properties of these trees and mention their use by solving other graph problems.

## 4.1 Graph Searching and MST Problem

Let us imagine a connected undirected graph with all edges having the same weight (e.g. weight $w(e) = 1$ for each edge $e$) and let us trace the Jarník's method for gaining the minimum spanning tree on this graph (see section 2.2). One can see that at each step an arbitrary edge, having one vertex in the blue tree and the other not, is coloured blue. A consecutive adding vertices into the blue tree can be understood as a consecutive search of them. Hence, to get either the Breadth-First Search or Depth-First Search algorithm for ***consecutive search of all vertices*** of the given connected undirected graph $G$, we simply modify Jarník's method in the following way (see [14]).

Breadth-First Search: At each step we choose from the uncoloured edges, having one vertex in the blue tree and the other not, such an edge having the end-vertex being added to the blue tree as the first of all in blue tree vertices belonging to the mentioned uncoloured edges and colour it blue. (Remark: To identify such a end-vertex we store vertices adding into the blue tree in the data structure queue (FIFO)).

Depth-First Search: At each step we choose from the uncoloured edges, having one vertex in the blue tree and the other not, such an edge having the end-vertex being added to the blue tree as the last of all in blue tree vertices belonging to the mentioned uncoloured edges and colour it blue. (Remark: To identify such a end-vertex we store vertices adding into the blue tree in the data structure stack (LIFO)).

The following figures Fig.11 - Fig.15 illustrates four first steps of both search methods on the graph $G_1$ (Fig.1) starting in vertex *a*, and the created blue spanning trees. To see mutual relations compare created blue trees on figures Fig.6, Fig13 and Fig15.

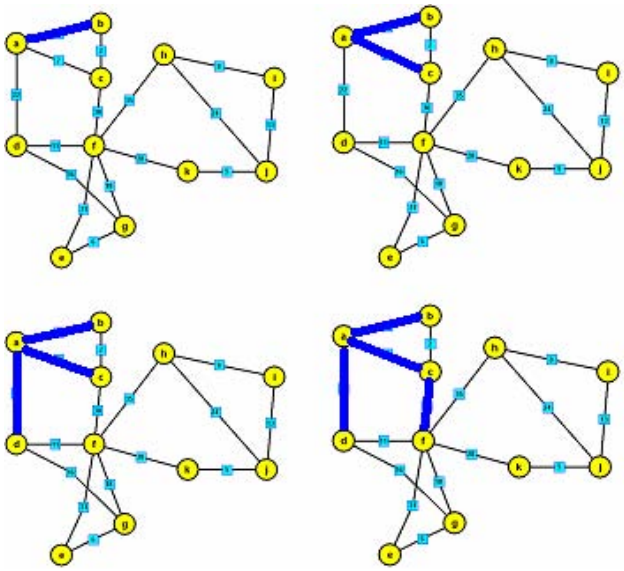

Fig.11 Breadth-First Search on the graph $G_1$ starting with the vertex *a*

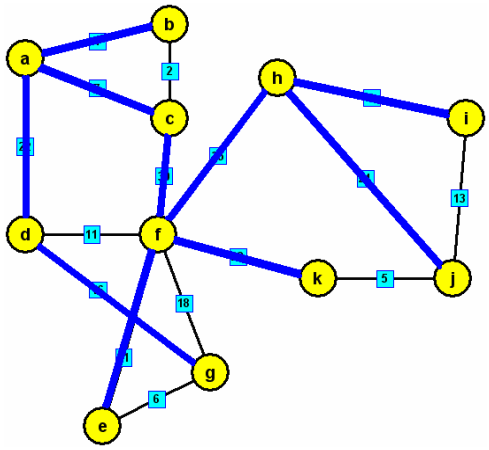


Fig.12 Blue tree obtained as the result of Breadth-First Search on the graph $G_1$ starting with *a*

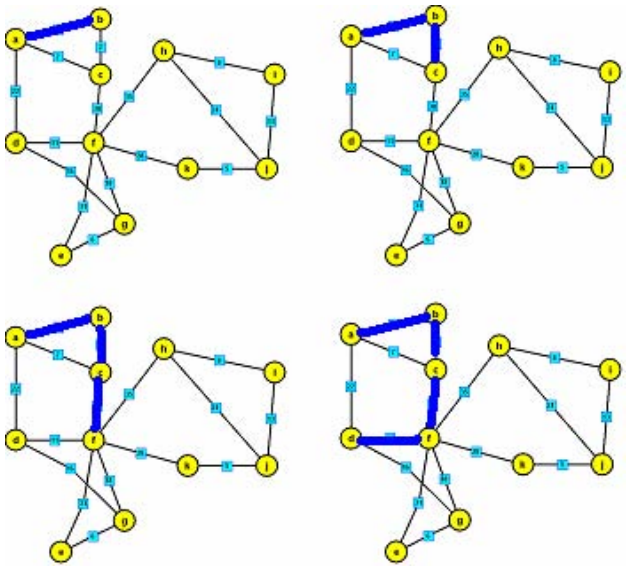

Fig.13 Depth-First Search on the graph $G_1$ starting with the vertex *a*

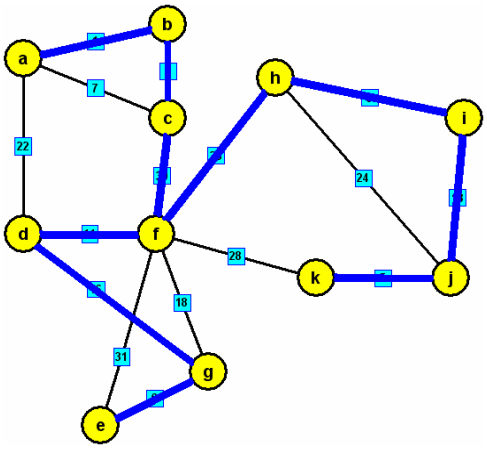


Fig.14 Blue tree obtained as the result of Depth-First Search on the graph G1 starting with *a*

## 4.2 Breadth and Depth-First-Search Trees

Let us describe both above illustrated blue trees as rooted trees with the root *a* (Fig.15-Fig.16).

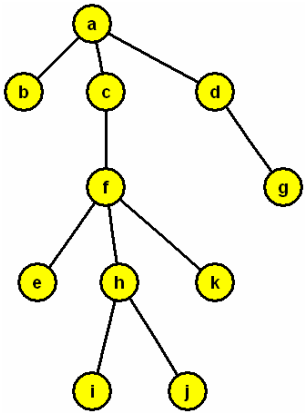


Fig.15 Rooted tree to the blue tree on the Fig.12

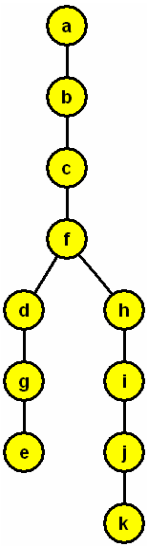


Fig.16 Rooted tree to the blue tree on the Fig.14

Generally, let us denote by $(T_B, v)$ the rooted tree with the root $v$, where $T_B$ is the tree gained by the Breadth-First Search with the initial vertex $v$. This rooted tree $(T_B, v)$ we will call the *Breadth-First Search Tree*. By analogy, let us denote by $(T_D, v)$ the rooted tree with the root $v$, where $T_D$ is the tree gained by the Depth-First Search with the initial vertex $v$. This rooted tree $(T_D, v)$ we will call the *Depth-First Search Tree*.

There are the two following obvious statements.

**Statement 1:**

Given $G$ connected undirected graph. If $(T_B, v)$ is blue Breadth-First Search Tree of $G$, then the end-vertices of each uncoloured edge of $G$ belong either to the same level or to the adjacent levels of $(T_B, v)$.

**Statement 2:**

Given $G$ connected undirected graph. If $(T_D, v)$ is blue Depth-First Search Tree of $G$, then for the end-vertices of each uncoloured edge of $G$ it follows that one is the ancestor of the other in $(T_D, v)$.

From the statement 1 (statement 2 resp.) describing the property of Breadth-First-Search Tree (Depth-First-Search Tree resp.) the other statements follow, as e.g.

**Statement 3:**

Given $G$ connected undirected graph and $(T_B, v)$ its blue Breadth-First Search Tree. There is a circle of odd length in $G$ if and only if there is an uncoloured edge having both end-vertices in the same level of $(T_B, v)$.

**Statement 4:**

Given $G$ connected undirected graph and $(T_D, v)$ its blue Depth-First Search Tree. For vertices of $G$ it follows:

a) $v$ is a cut vertex if and only if $v$ has at least two direct descendants in $(T_D, v)$,

b) $x \neq v$ is a cut vertex if and only if there is direct descendant $y$ of $x$ in $(T_D, v)$, such that neither $y$ nor a descendant of $y$ is connected by uncoloured edge of $G$ with an ancestor of $x$ in $(T_D, v)$.

### 4.3 Summary

Using Breadth-First-Search and Depth-First-Search and the mentioned statements we are able easily formulate other algorithms as, for example, *algorithms determining if the given graph is bipartite or not, or if there is a circle in the given graph containing the given vertex (edge respectively), algorithms determining the girth of the given graph, algorithms finding in the given graph all cut vertices and all 2-connected subgraphs as well*. (Remark: Proofs of all statements together with the detailed descriptions and proofs of all algorithms mentioned above are available in [14].)

However, to get needed information about end-vertices of uncoloured edges of a connected undirected graph it is also necessary to arrange a consecutive search of edges (obviously together with consecutive search of vertices to get searching trees for validity of statements). It is easy to obtain such algorithms using a small modification of the above mentioned search algorithms. Especially, we enhance the algorithms including data structures FIFO and LIFO as follows.

Breadth-First Search of vertices and edges

1. Initially all vertices and edges of the graph $G$, with $n$ vertices and $m$ edges, are uncoloured. Let us choose any single vertex, put it into FIFO, colour it blue and search it.

2. While the FIFO is not empty do the following commands:

- choose the first vertex $x$ in FIFO,
- **if** there is an uncoloured edge $\{x, y\}$ **then**
  **if** the vertex $y$ is uncoloured, then search and colour blue the vertex $y$ and the edge $\{x, y\}$, and put the vertex $y$ into FIFO
  **else** (i.e. if the ucoloured edge $\{x, y\}$ has both vertices already in the blue tree) search and colour the edge $\{x, y\}$ green

  **else** remove the vertex $x$ from FIFO (i.e. remove $x$ in the case that it isn‘t end-vertex of any uncoloured edge).

Depth-First Search of vertices and edges

1. Initially all vertices and edges of the graph $G$, with $n$ vertices and $m$ edges, are uncoloured. Let us choose any single vertex, put it into LIFO, colour it blue and search it.

2. While the LIFO is not empty do the following commands:

- choose the last vertex x in LIFO,
- **if** there is an uncoloured edge {x, y} **then**
  **if** the vertex $y$ is uncoloured, then search and colour blue the vertex $y$ and the edge $\{x, y\}$, and put the vertex $y$ into LIFO
  **else** (i.e. if the ucoloured edge $\{x, y\}$ has both vertices already in the blue tree) search and colour the edge $\{x, y\}$ green

  **else** remove the vertex $x$ from LIFO (i.e. remove $x$ in the case that it isn‘t end-vertex of any uncoloured edge).

When searching *green* edges we can achieve needed information. Let us illustrate it at the following example.

*Example*

Decide if the graph, represented by the following adjacency matrix, is bipartite (i.e. if there does not exist a circle of odd length).

| | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| a | | 1 | 1 | 1 | | | | |
| b | 1 | | 1 | | | | | 1 |
| c | 1 | 1 | | 1 | 1 | | | |
| d | 1 | | 1 | | 1 | 1 | 1 | |
| e | | | 1 | 1 | | | | |
| f | | | | 1 | | | 1 | 1 |
| g | | | | 1 | | 1 | | |
| h | | 1 | | | | 1 | | |

We use Breadth-First-Search starting with arbitrary vertex and when searching *green* edge we determine if its both end-vertices are in the same level of the created Breadth-First-Search tree. If there is no such an edge, the graph is bipartite.

*Solution*

| FIFO | Breadth-First-Search tree | green edge |
|---|---|---|
| **a** | a(0,a) | |
| **a**,b | a(0,a), b(1,a) | |
| **a**,b,c | a(0,a), b(1,a), c(1,a) | |
| **a**,b,c,d | a(0,a), b(1,a), c(1,a), d(1,a) | |
| **b**,c,d | a(0,a), b(**1**,a), c(**1**,a), d(1,a) | {b,c} |

Both end-vertices of the edge {b,c} are in the same level, thus *the given graph is not bipartite*.

All matter explained in this chapter was used on connected graphs, and can, when dealing with particular components of the given graph, be easily extended to graphs that need not be connected.

## 5 Mazes and labyrinths

There is an old question: "How to escape from a maze or labyrinth?" In the graph terminology the maze problem requires a walk, which contains every edge of the graph. Although it is ancient problem the seriously examination of it started not before the 19th century. The efficient methods for solution of this problem were created by Trémaux in 1882 and by Tarry in 1895 (see [15]).

Let us imagine a maze as a graph (each passage is represented by an edge and each junction by a vertex) and remind both methods using graph terminology.

Trémaux's rules

1. Each edge is traversed exactly once in one direction,
2. Do not return along the edge which has led to a vertex for the first time unless you cannot do otherwise,
3. If you come along the edge to the already visited vertex go immediately along the same edge back.

Tarry's rules

1. Each edge is traversed exactly once in one direction,
2. Do not return along the edge which has led to a vertex for the first time unless you cannot do otherwise.

It is obvious that Trémaux algorithm is a special case of Tarry's algorithm. Moreover, *Trémaux algorithm is identical with our Depth-First Search of vertices and edges algorithm* (see the section 4.3) if we consider it also as walk through edges.

Let us complete this part with a note that another special case of Tarry's algorithm was found in 1973 by Edmonds and Johnson.

Edmonds-Johnson's rules

1. Each edge is traversed exactly once in one direction,
2. Do not return along the edge which has led to a vertex for the first time unless you cannot do otherwise,
3. If you have more edges available when leaving a vertex prefer this edge which has not been already visited.

This algorithm also gives a suitable solution to the problem of how to find an Eulerian trail in an Eulerian graph. *When looking for Eulerian trail in the given Eulerian graph it is sufficient to use Edmonds-Johnson algorithm and consider the "back" traverse of edges*. This is the solution.

We shall emphasize that the Edmonds-Johnson's approach to searching edges of a connected undirected graph, as opposed to the Trémaux ones, does not search vertices. This can be clearly seen on the data structure LIFO. Using the Trémaux approach each vertex occurs in LIFO exactly once, however using the Edmonds-Johnson's approach each vertex occurs as many times as it was traversed.

## 6 Program Graphs

The figures used in this paper ware created within the program "Graphs" [16]. Let us very briefly introduce this useful multimedia study material.

The program "Graphs" enables the creation of a new graph, editing it, working on it (moving, colouring vertices, edges, etc.), saving graph in the program, and saving the graph in bmp format.

Regarding the topic discussed in this paper the most important option of the program is the possibility to open more than one window so that two (three) objects can be compared at once (see Fig.17).

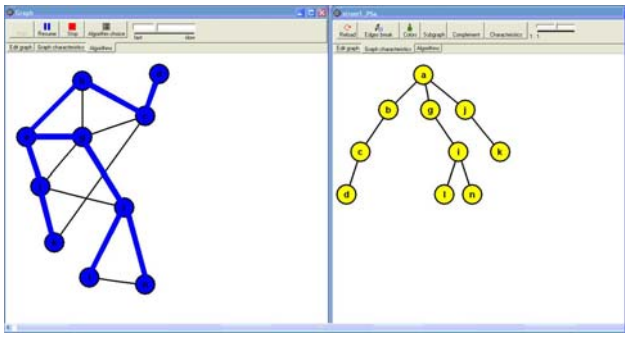

Fig.17 Program "Graphs" – two opened windows; left: Breadth-First-Search of the given graph; right: its Breadth-First-Search tree

Several algorithms in different windows can be started at a time as well. This option is very useful when illustrating the differences between two (three) algorithms applied on the same graph.

## 6 Results and Conclusion

Well-prepared students in the area of Graph Theory and Combinatorial Optimization should be able to describe various practical situations with the aid of graphs, solve the given problem expressed by the graph, and translate the gained solution back into the initial situation.

The method described in the paper of how they have been made familiar with combinatorial algorithms in contexts enables them not only to get deeper insight into the subject matter but also to enhance their logical thinking and their facility to solve everyday life practical situations. Thus, they gain many useful ideas and inspiration for their own solutions to tasks within various research areas.

*References:*


[1] Milková, E., THE MINIMUM SPANNING TREE PROBLEM: Jarník's solution in historical and present context, *Electronic Notes in Discrete Mathematics*, 28 (2007), pp. 309–316.

[2] Nešetřil, J., Milková, E., Nešetřilová, H., Otakar Borůvka on Minimum Spanning Tree Problem, *Discrete mathematics*, 233 (2001), pp. 3-36.

[3] Borůvka, O., O jistém problému minimálním, *Práce Mor. Přírodověd. Spol v Brně*, 3, 1926, pp. 37-58.

[4] Borůvka, O.: Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí. *Elektrotechnický obzor*, 15, 1926, 153-154.

[5] Graham, R. L., Hell, P., On the History of the Minimum Spanning Tree Problem, *Annals of the History of Computing* 7, 1, 1985, pp. 43-57.

[6] Jarník, V., O jistém problému minimálním, *Práce Mor. Přírodověd. Spol.v Brně*, 6, 1930, pp. 57-63.

[7] Prim, R. C., Shortest connection networks and some generalizations, *Bell Syst. Tech. J.*, 36, 1957, pp. 1389-1401.

[8] Kruskal, J. B., On the shortest spanning tree of a graph and the travelling salesman problem, *Proc. Amer. Math. Soc.*, 7, 1956, pp. 48-50.

[9] Tarjan, R. E., Data structures and network algorithms, *Ch. 6, CBMS Regional Conf., SIAM*, Philadelphia, 1983

[10] Dijkstra, E. W., A note on two problems in connection with graphs, *Numer. Math.*,1, 1959, pp. 269-271.

[11] Brendel, R., Krawczyk , H., Application of Social Relation Graphs for Early Detection of Transient Spammers, *WSEAS TRANSACTIONS on INFORMATION SCIENCE & APPLICATIONS*, Issue 3, Volume 5, 2008, pp.267-276.

[12] Nhor Sok Lang, Takao Shimomura, Quan Liang Chen, Kenji Ikeda, Context-Dependent Extensible Syntax-Oriented Verifier with Recursive Verification, *WSEAS TRANSACTIONS on INFORMATION SCIENCE & APPLICATIONS*, Issue 2, Volume 5, 2008, pp.44-53.

[13] Azlinah Mohamed, Marina Yusoff, Itaza Afiani Mohtar, Sofianita Mutalib, Shuzlina Abdul Rahman, Constraint Satisfaction Problem Using Modified Branch and Bound Algorithm, *WSEAS TRANSACTIONS on COMPUTERS*, Issue 1, Volume 7, 2008, pp.44-53.

[14] Milková, E., *Optimalizace, třídění a prohledávání stromů*, diploma thesis, Charles University, Faculty of Mathematics and Physics, 1997.

[15] Biggs, N. L., Lloyd, K. E., Wilson, R. J.,*Graph theory 1736-1936*, Clarendon Press, Oxford, 1976.

[16] Pozdílek M., *Grafové algoritmy: vizualizace*, Hradec Králové, diploma thesis, 2004.