Graph Algorithms in mutual Contexts

EVA MILKOVÁ, ANTONÍN SLABÝ
Department of Informatics and Quantitative Methods
University of Hradec Králové
Rokitanského 62
CZECH REPUBLIC

Abstract: Mathematics belongs to the oldest science however the area known as Combinatorial or Discrete Optimization close connected with Graph Theory and Computer Science is quite yang. To educate students in this area it is important to meet them familiar with combinatorial algorithms in contexts to be able to get deeper into each problem and entirely understand it. This paper illustrates several algorithms solving the known problems on graphs and emphasizes different approaches to the solution of the same problem on the one hand and mutual relationships among methods solving various problems on the other hand.

Key-Words: Graph Theory, Minimum Spanning Tree Problem, Breadth-First-Search, Depth-First-Search, Dijkstra’s Algorithm, Maze Problem, Eulerian Graph

1 Introduction

When we deal with a particular problem within the subject Theory of Graph 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 all 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.

This paper discusses 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 relationship between Dijkstra’s algorithm finding the shortest path and the Jarník’s-Prim’s algorithm solving 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.

2 Minimum Spanning Tree Problem

In the contemporary terminology the Minimum Spanning Tree (MST in short) problem can be formulated as follows [1]:

Given a connected undirected graph $G = (V, E)$ with $n$ vertices, $m$ edges and real weights assigned to its edges (i.e. $w: E \rightarrow \mathbb{R}$). Find among all spanning trees of $G$ a spanning tree $T = (V, E')$ having minimum value $w(T) = \sum w(e)$, 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.

2.1 Borůvka’s algorithm

First formulation of the problem was given in 1926 by the Czech mathematician Otakar Borůvka [2]. Let us introduce his solution written in the present terminology. There are various descriptions of Borůvka’s solution in most of the modern textbooks. We illustrate two algorithms solving the problem as an edge-colouring process (see [1], [4]).(Remark: The survey of the works devoted to the MST problem until 1985 is given in the article [3] and this historical paper is followed up in the article [4] Otakar Borůvka on minimum spanning tree problem: Translation of both the 1926 papers, comments, history.)
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 $T'$.

Let us illustrate the MST problem using Borůvka’s algorithm – second description on the following graph.

2.2 Jarník’s algorithm
Another Czech mathematician, Vojtěch Jarník, quickly realized the novelty and importance of the problem after reading the 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 [5]. In the present terminology we can describe it as follow [1].

Jarník’s algorithm
1. Initially all 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.

Much later, in the time of newly developing field, computer science, R.C. Prim, who, just as the others, wasn’t aware of Jarník’s solution, created the same solution as Jarník’s solution but he included a more
Let each vertex be a trivial blue tree.

Let us order the edges in nondecreasing order by weight.

Let each vertex be a trivial blue tree.

Let us order the edges in nonincreasing order by weight.

spanning tree.

describe them as follow [1].

2.3 Kruskal’s algorithm

To the classical solutions of the MST problem belong also two solutions given by J. B. Kruskal [7]. We describe them as follow [1].

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.

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.

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.

Kruskal’s dual solution consequently rejects edges belonging to a circle having no red coloured edge.

Two different descriptions of Borůvka’s algorithm enable to get better insight to his solution. Both descriptions of Jarník’s solution are useful for description of another algorithms. Namely, from the first algorithm (see 2.2, Jarník’s algorithm) we can continue to the searching methods (see the 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).

3 Shortest path - Dijkstra’s algorithm

In the second section above we discussed 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 serve also as a solution of the problem 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. As an example confirming this answer the graph on the Fig.6 can serve. The blue minimum spanning tree of this graph is formed by edges $\{c, d\}, \{d, g\}$ and $\{g, e\}$, thus the length of the path from $c$ to $e$ in the blue minimum spanning tree is 13, however the length of the shortest path between these two vertices is 10, direct through the edge $\{c, e\}$.

Fig.6 The given graph
However, it seems that there must be a relationship between the two problems.

Thus 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 fifties of last century by E.W.Dijkstra (see [8]).

4 Graph Searching
To the most used graph searching algorithms belong two well-known algorithms Breadth-First-Search and Depth-First-Search. Their description is usually based on using of date structures, either queue (FIFO – for Breadth-First-Search) or stack (LIFO – for Depth-First-Search).

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 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 of vertices into the blue tree can be understood as a consecutive searching of them. Hence, to get either the Breadth-First Search or Depth-First Search algorithm for consecutive searching of all vertices of the given connected undirected graph G, we simply modify Jarník’s method in the following way (see [9]).

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 (see Fig.7 and Fig.8).

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 (see Fig.9 and Fig.10).

Both searching algorithms, described in this way, finish their running with finding a blue spanning tree of the given graph G.

Example
Let us have the following graph G.

![Graph G](image)

Using Breadth-First Search on the given graph G, starting with the vertex c, the vertices will be searched in the order c, a, d, f, e, b, g, and the following blue tree will be created.

![Blue tree of Breadth-First Search](image)

However, using Depth-First Search on the given graph G, starting with the vertex c, the vertices will be searched in the order c, a, d, b, e, g, f, and the following blue tree will be created.

![Blue tree of Depth-First Search](image)

4.2 Breadth and Depth-First-Search Trees
Let us write both above illustrated blue trees on Fig.8 and Fig.9 as rooted trees with the root c (Fig.10 and Fig.11).
Fig. 11 Rooted tree to the blue tree on the Fig. 9

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.

**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 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 [9].)

To get needed information about end-vertices of uncoloured edges it is necessary to arrange also consecutive search of edges (of course together with consecutive search of vertices to get searching trees for validity of statements). It is easy to obtain such algorithms using small modification of above mentioned searching algorithms. Especially, we enhance the algorithms as follow.

**Breadth-First Search of vertices and edges**

1. Initially all 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 edge \(\{x, y\}\) has one vertex in the blue tree and the other not, 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 uncoloured 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 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 edge \(\{x, y\}\) has one vertex in the blue tree and the other not, 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 uncoloured 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).

By searching green edges we can achieve needed information.

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 [10]).

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.
3. If you have more edges available when leaving a vertex prefer this edge which has not been already visited.

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. And this algorithm gives also suitable solution of the problem how to find Eulerian trail in an Eulerian graph.

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.

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.

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 way, described in the paper, of how they have been meeting familiar with combinatorial algorithms in contexts enables them not only to get deeper insight into the explained 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 inspirations for their own solutions of tasks within various research areas.

References: