Breadth-First-Search Tree – Levels and Subtrees

Eva Milková

***Abstract* —**The Breadth-First-Search algorithm belongs to the most used searching algorithms. The Breadth-First-Search tree has its special properties that can be convenient to be used in various other algorithms. Deep analysis of the topic and discussion on mutual relationships between solutions to problems allows the teacher to enhance student's logical thinking and support their understanding to more complex algorithms. Moreover, contrary to the intuition that a problem is solvable by a polynomial algorithm, theoretical background of the appropriate topic can show that the problem is much more complex and a fast algorithm is unlikely to exist. The paper presents a comprehensive view on problems that are based on Breadth-First-Search tree property.

Keywords BFS tree, BFS tree property, circles with the given property.

Introduction

It is a fact generally agreed on by mathematics educators and researchers in the field of mathematics education that problem solving is a cornerstone of mathematics taught. Problems and issues should be presented in a more challenging way than just as tasks in which an individual or a group of students are expected to demonstrate to their teacher how they can apply the mastered algorithms. The presentation and processing of problems should develop students' intellectual activity and encourage their creativity in the solving process (cf. [1, 2]).

The Graph Theory is a discipline, which can put forward some very entertaining problems, and informatics has played a big part in its development. These two fields are strongly interconnected, which can mainly be seen in the design of computer algorithms. 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. To get a deeper insight into a problem it is important to examine it from more than one point of view and discuss various approaches to its solution.

In the paper, whose aim is to point out the importance of a proper subject matter analysis being seen in mutual relations, we devote attention to the Breadth-First-Search tree and its property. There exist various descriptions of the Breadth-First-Search (BFS in short) algorithm (see e.g. [3 - 9]).

The paper briefly presents the BFS algorithm formulated according to [10] as an edge colouring process as well as BFS tree property. Then the necessary and sufficient conditions for several statements connected with the BFS tree property are discussed and on the basis of them, various simple algorithms using vertex coding are summarized.

Breadth-First-Search

The Breadth-First-Search (BFS shortly) algorithm together with the Depth-First-Search algorithm belongs to the most used searching algorithms.

Let us formulate the BFS algorithm searching both vertices and edges of a given undirected graph in the form of an edge colouring process. Let us deal with a connected graph (in the case of a disconnected graph, the algorithm is applied on its components).

BFS algorithm of vertices and edges

```
1. Initially all vertices and edges of the given connected
   undirected graph $G$, with $n$ vertices and $m$ edges, are
   uncoloured. Let us choose any single vertex, insert it
   into queue $Q$, colour it blue and search it.
2. while $Q$ is not empty do the following commands:
   o choose the first vertex $x$ in $Q$
   o if there is an uncoloured edge $\{x, y\}$ then
        if the vertex $y$ is uncoloured then
          search and colour blue both the vertex $y$ and
          the edge $\{x, y\}$, and insert the vertex $y$ into $Q$
        else
          search and colour the edge $\{x, y\}$ red
     else
       delete the vertex $x$ from $Q$
```

A. Relation to the Minimum Spanning Tree Problem

A lucid interpretation of the BFS algorithm and BFS tree (see thereinafter) can proceed from the Jarník's method solving the minimum spanning tree problem, see [11]:

Minimum Spanning Tree Problem. Given a connected undirected graph $G = (V, E)$ with n vertices, m edges and real weights assigned to its edges (i.e. $w: E \to \mathrm{R}$). 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.

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.

Using a connected undirected graph with all edges having the same weight, e.g. weight $w(e) = 1$ for each edge e, and tracing the Jarník's algorithm solving the minimum spanning tree on this graph, one can see that at each step an arbitrary edge, having one vertex in the blue tree and the other not, is coloured blue. Consecutive adding vertices into the blue tree can be understood as a consecutive search of them. Hence, to get either the Breadth-First Search for consecutive search of all vertices of the given connected undirected graph G, we simply modify Jarník's algorithm in the following way.

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.

B. Breadth-First-Search Tree

In connection with the above mentioned interpretation it is evident that applying the BFS algorithm to an undirected connected graph the blue coloured edges form a spanning tree T. If we represent the gained spanning tree T by a rooted tree with the root in initial vertex v (i.e. vertex in which the BFS algorithm starts) we get so called BFS tree (T, v) with the root v.

Definition. Let G be a connected undirected graph, let v be a vertex of G, and let T be its spanning tree gained by the BFS algorithm starting in vertex v. Let us call an appropriate rooted tree (T, v) as **BFS tree (T, v) with the root v**, let us call the edges of G that do not appear in (T, v) as **non-tree edges**, and let us call the components of the forest $F = (T, v) - v$ as **(T, v)-subtrees**.

Let us denote the level of (T, v) where the vertex y lies by $h(y)$ supposing $h(v) = 0$.

Remark: Obviously, the concept (T, v)-subtrees denotes all subtrees (T', r) with the root r, where r is a direct descendant of the initial vertex v in BFS tree (T, v).

C. Breadth-First-Search Tree Property

Theorem. The end-vertices of each non-tree edge of G belong either to the same level or to the adjacent levels of (T, v).

Proof. Let $\{x, y\}$ be a non-tree edge of G. We may assume that the vertex y was put into Q later than the vertex x (for otherwise we would go analogically). Obviously, $h(z) \geq h(x)$ for each vertex z put into Q later than x and $h(z) \leq h(x) + 1$ for each vertex z put into Q before x. Hence, with regard to the assumption and because the vertices x and y are adjacent in G, $h(x) \leq h(y) \leq h(x) + 1$.

Corollary. The length of the shortest path from vertex v to a vertex y in G equals $h(y)$.

D. Necessary and Sufficient Conditions

In the practice there are various problems looking for and dealing with a circle with the given property. Problems connected with Hamiltonian graphs are a case of such problems. Nevertheless, the existence of various suitable circles with the given property can be efficiently solved, using the observation of the BFS tree.

Theoretical background belongs to the most difficult parts in the engineering education. The following proofs can serve also as a lucid explanation of necessary and sufficient conditions.

Let G be a connected undirected graph, let v be a vertex of G, and let T be its spanning tree gained by the BFS starting in vertex v.

Observing both the levels and the subtrees of the given BFS tree (T, v) *both necessary and sufficient conditions* for the existence of a circle with the given property obviously follow from the theorem and corollary (see statements 1, 2 and 3).

Firstly, let us observe end-vertices of non-tree edges regarding to the levels of a given BFS tree (T, v) similarly as in the corollary concerning the length of the shortest path P_{vy}.

Statement 1. There is a circle of odd length in G if and only if there is a non-tree edge having both end-vertices in the same level of (T, v).

Proof. Let $\{x, y\}$ be a non-tree edge of G, $h(x) = h(y)$. Let z be the first ancestor of both vertex x and y in (T, v). Obviously, $P_{zx} \cap P_{zy} = \emptyset$, length P_{zx} = length P_{zy}. Hence, $P_{zx} \cup P_{zy} \cup \{x, y\}$ creates a circle of odd length.

On the other hand, if there is no non-tree edge having both end-vertices in the same level of (T, v) then G is bipartite graph and thus there is no circle of odd length in G.

Secondly, let us observe end-vertices of non-tree edges regarding to (T, v)-subtrees.

Statement 2. There is a circle containing a vertex v in G if and only if there is a non-tree edge having its end-vertices in different (T, v)-subtrees.

Proof. Let $\{x, y\}$ be a non-tree edge of G, whereas x and y belongs to different (T, v)-subtrees. Let us suppose $x \in (T', a)$ and $y \in (T', b)$. Obviously, $P_{ax} \cap P_{by} = \emptyset$. Hence, $P_{ax} \cup P_{by} \cup \{a, v\} \cup \{v, b\} \cup \{x, y\}$ creates in G a circle containing vertex v.

On the other hand, if there is no non-tree edge having its end-vertices in different (T, v)-subtrees, then there is either no non-tree edge in (T, v) and thus there is no circle in G at all, or each non-tree edge has its end-vertices in the same (T, v)-subtree and thus each circle belongs to a subtree (T', r), where r is a direct descendant of the initial vertex v. Hence, in G there is no circle containing a vertex v.

Statement 3. There is a circle containing an edge $\{v, w\}$ in G if and only if there is a non-tree edge having one end-vertex in the (T, v)-subtree with the root w and the other end-vertex in another (T, v)-subtree.

Proof. Let $\{x, y\}$ be a non-tree edge of G, x and y belong to different (T, v)-subtrees, whereas one of them belongs to (T', w). A circle containing an edge $\{v, w\}$ in G can be determined accordingly as above.

On the other hand, if there is no non-tree edge having one end-vertex in the (T, v)-subtree with the root w and the other end-vertex in another (T, v)-subtree, then there are three possible cases obviously leading to nonexistence of a circle containing an edge $\{v, w\}$ in G:

- there is no non-tree edge in (T, v)
- each non-tree edge has its end-vertices in the same (T, v)-subtree
- there is no non-tree edge in (T, v) having one end-vertex in the subtree (T', w)

BFS Modifications

Let us suppose an undirected connected graph G with n vertices and m edges. The above given corollary and three statements directly lead to a formulation of the following six simple BFS algorithm modifications, having the time complexity equal to $O(n + m)$.

A. BFS Tree Levels

Running the BFS algorithm starting at vertex v, and saving the level $h(y)$ at each step by each vertex $y \neq v$, we achieve

1. algorithm determining the length of the shortest path from the vertex v to a vertex y in G

and examining if $h(x) = h(z)$ when searching a non-tree edge $e = \{x, z\}$, we easily achieve

2. algorithm determining if there is an odd circle in G

B. BFS Tree (T, v)-subtrees

Running the BFS algorithm starting at vertex v, and at each step saving by each vertex $y \neq v$ the information about (T, v)-subtree in which the vertex y lies, and examining relevance of x and z to (T, v)-subtrees when searching a non-tree edge $e = \{x, z\}$ we achieve

1. algorithm determining if there is a circle containing the given vertex v
2. algorithm determining if there is a circle containing the given edge $\{v, w\}$

Note that there is another simple algorithm determining if there is a circle containing the given edge $\{v, w\}$ in G (otherwise determining if the edge is/is not a cut edge) based on the BFS or Depth-First-Search algorithm applied to $G - \{v, w\}$ starting either at the vertex v or w.

C. BFS Tree levels and (T, v)-subtrees

Running the BFS algorithm starting at vertex v, and saving at each step saving by each vertex $y \neq v$, both the level $h(y)$ and the information about (T, v)-subtree in which the vertex y lies, and examining relevance of x and z to (T, v)-subtrees when searching a non-tree edge $e = \{x, z\}$ we achieve

1. algorithm determining the length of the shortest circle containing the given vertex v if there is any
2. algorithm determining the length of the shortest circle containing the given edge $\{x, z\}$ if there is any

Note that as soon as we find the first non-tree edge $\{x, z\}$ having its end vertices in different (T, v)-subtrees, we examine $h(x)$, $h(z)$. If $h(x) = h(z)$, we finish the algorithm, however, if $h(x) \neq h(z)$ (assuming $h(x) < h(z)$, we have to continue searching vertices belonging to the level $h(x)$ to recognize if there is an edge having both end-vertices all at once in different (T, v)-subtrees and in the same levels. Thus, if there is a circle containing a given vertex v, the length of the shortest circle containing a given vertex v is either $2h(x) + 1$ or $2h(x) + 2$.

Example

Given the graph represented by the following adjacency matrix.

	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*
a			1	1					1
b				1	1	1			
c	1						1		1
d	1	1							
e		1					1		
f		1						1	
g			1		1			1	
h						1	1		
i	1		1						

Let us determine one of the shortest circles containing the vertex g if there is any.

We use the following vertex coding - starting BFS algorithm with the vertex g, at each step we save for each vertex $y \neq g$ the information describing the *ancestor* of y, the *number* of subtree in which the vertex v lies and the *level* $h(v)$.

queue Q	BFS tree	non-tree edges
$\boldsymbol{g}$	g	
$\boldsymbol{g},c$	g, $c(g,1,1)$	
$\boldsymbol{g},c,e$	g, $c(g,1,1)$,$e(g,2,1)$	
$\boldsymbol{g},c,e,h$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$	
$\boldsymbol{c},e,h,a$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$, $a(c,1,2)$	
$\boldsymbol{c},e,h,a,i$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$, $a(c,1,2)$,$i(c,1,2)$	
$\boldsymbol{e},h,a,i,b$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$, $a(c,1,2)$,$i(c,1,2)$,$b(e,2,2)$	
$\boldsymbol{h},a,i,b,f$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$, $a(c,1,2)$,$i(c,1,2)$,$b(e,2,2)$,$f(h,3,2)$	
$\boldsymbol{a},i,b,f,d$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$, $a(c,1,2)$, $i(c,1,2)$,$b(e,2,2)$,$f(h,3,2)$,$d(a,1,3)$	
$\boldsymbol{a},i,b,f,d$	g, $c(g,1,1)$,$e(g,2,1)$,$h(g,3,1)$, $a(c,\mathbf{1},2)$, $i(c,\mathbf{1},2)$,$b(e,2,2)$,$f(h,3,2)$,$d(a,1,3)$	$\{a,i\}$

i,*b*,*f*,*d*	*g*, *c*(*g*,1,1),*e*(*g*,2,1),*h*(*g*,3,1), *a*(*c*,1,2), *i*(*c*,1,2),*b*(*e*,2,2),*f*(*h*,3,2),*d*(*a*,1,3)	
b,*f*,*d*	*g*, *c*(*g*,1,1),*e*(*g*,2,1),*h*(*g*,3,1), *a*(*c*,1,2), *i*(*c*,1,2),*b*(*e*,**2**,**2**),*f*(*h*,3,2),*d*(*a*,**1**,**3**)	{*a*,*i*}, {*b*,*d*}
b,*f*,*d*	*g*, *c*(*g*,1,1),*e*(*g*,2,1),*h*(*g*,3,1), *a*(*c*,1,2), *i*(*c*,1,2),*b*(*e*,**2**,***2***),*f*(*h*,**3**,***2***),*d*(*a*,1,4)	{*a*,*i*},{*b*,*d*} {***b***,***f***}

The non-tree edge $\{b, f\}$ determines *the circle (g,e,b,f,h,g) of the length 5, which is the shortest circle in the given graph containing the vertex g*.

D. Remarks

Applying the BFS modification solving the shortest circles containing the given vertex on each vertex in the given graph, algorithm of time complexity $O(n^3)$ determining the girth of the given graph (i.e. the length of the shortest circle), is formulated.

Saving at each step by each vertex $y \neq v$ also the information describing the ancestor of y we can easily determine the shortest path from the vertex v to a vertex y according to the BFS tree with the root v, as well as a circle with the given property (see the example above).

Using the BFS algorithm we are also able to formulate an algorithm searching all the shortest paths between two given vertices. Its description based on an *x-y* Shortest Path Tree construction is given in [10], where an *x-y* Shortest Path Tree is defined as follows:

Let G be a connected undirected graph, let x and y be vertices of G. An *x-y Shortest Path Tree* is a rooted tree $T_{x,y}$ with the root $y[1]$ and with leaves $x[1], ..., x[k]$, where k is the number of the shortest paths from x to y existing in G, and where each path in $T_{x,y}$ leading from $x[i]$, $i = 1, ..., k$, to $y[1]$ represents exactly one of the shortest paths from x to y in G.

The algorithm can be intuitively understood with the help of and according to the following illustration.

Illustration. Let us find all the shortest paths between c and f in the graph on Fig. 1. Let us suppose that for each vertex v the number $h(v)$ has been already determined (see the number next each vertex on Fig. 1).

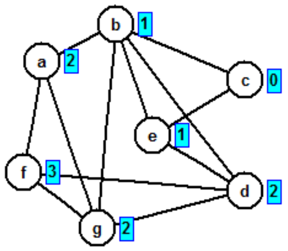

Fig. 1

Queue Q: $f[1](3)$, $a[1](2)$, $d[1](2)$, $g[1](2)$, $b[1](1)$, $b[2](1)$, $e[1](1)$, $b[3](1)$, $c[1](0)$, $c[2](0)$, $c[3](0)$, $c[4](0)$

$T_{x,y}$: $f[1](f[1])$, $a[1](f[1])$, $d[1](f[1])$, $g[1](f[1])$, $b[1](a[1])$, $b[2](d[1])$, $e[1](d[1])$, $b[3](g[1])$, $c[1](b[1])$, $c[2](b[2])$, $c[3](e[1])$, $c[4](b[3])$

Solution: (c, b, g, f); (c, e, d, f); (c, b, d, f); (c, b, a, f)

The complexity of an *x-y* Shortest Path Tree construction depends on the number of the shortest paths existing in the given graph. Their number can grow exponentially with the growth of vertices, see e.g. the following Fig. 2.

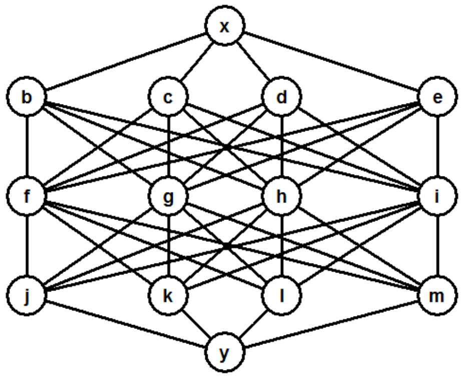

Fig. 2

ODD CIRCLE CONTAINING A GIVEN VERTEX

One could consider that an algorithm looking for an odd circle containing a given vertex (edge resp.) can be easily formulated, based on the statement 1 and the statement 2 (the statement 3 resp.).

However, let us observe Fig. 3 and Fig. 4, where two graphs are represented as BFS trees (thick lines = blue edges) completed by non-tree edges (thin lines). For the existence of an odd circle containing a given vertex in G, using these figures and the above given statements 1 and 2, we can formulate an obvious sufficient condition that is however not necessary (see statement 4 and Fig. 3), and a necessary condition that is however not sufficient, see statement 5 and Fig. 4.

Statement 4. If there is a non-tree edge having its end-vertices all at once in the same level of (T, v) and in different (T, v)-subtrees, then there is an odd circle containing the given vertex v.

Proof. Let $\{x, y\}$ be a non-tree edge of G, $h(x) = h(y)$, x and y y belongs to different (T, v)-subtrees. Let us suppose $x \in (T', a)$ and $y \in (T', b)$. Obviously, the length of the circle $P_{ax} \cup P_{by} \cup \{a, v\} \cup \{v, b\} \cup \{x, y\}$ containing vertex v in G is odd.

Statement 5. If there is an odd circle containing the given vertex v, then there is a non-tree edge having its end-vertices in the same level of (T, v) and there is a non-tree edge having its end-vertices in different (T, v)-subtrees.

Proof. follows directly from statement 1 and 2.

On the other hand, Fig. 3 is a lucid example that the statement 4 is not a necessary condition and Fig. 4 confirms that the statement 5 is not a sufficient condition for the existence of an odd circle containing a given vertex in *G*.

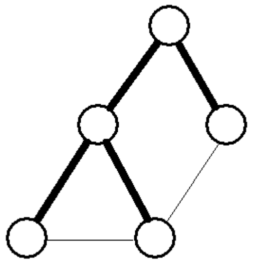
Fig. 3

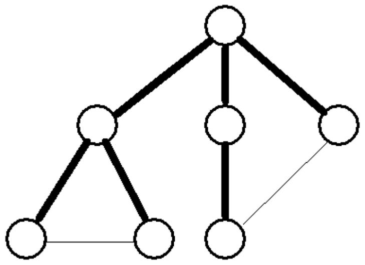
Fig. 4

To determine if there is an odd circle containing a given vertex (edge resp.) can be in general, similarly as to determine if a graph is Hamiltonian, difficult.

SHORTEST CIRCLE CONTAINING TWO GIVEN VERTICES

Let us think about an algorithm looking for a circle containing two given vertices *x* and *y* in the given graph *G*. In connection with the previous chapters let us consider such a shortest circle.

An intuitive idea aims to apply the following commands: find the shortest path P_{xy} (thin lines); remove its edges from *G*; in the gained graph find another path P_{xy} (thick lines) if there is any. However, the following figures (Fig. 5 – Fig. 8) show that this idea fails. In the graphs on Fig. 5 and Fig. 7 a circle of the length 8 exists. However, the above given commands find a circle of the length 9 in the first example (Fig. 6) and even no case of such a circle in the second example (Fig. 8).

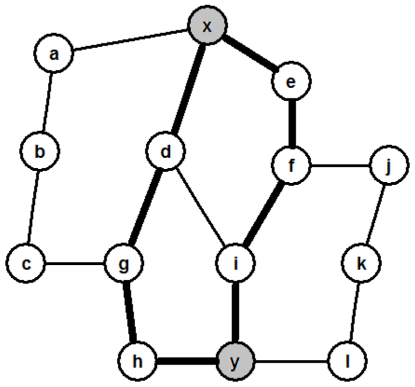

Fig. 4

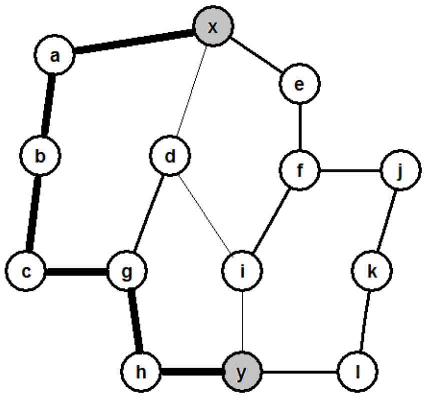

Fig. 5

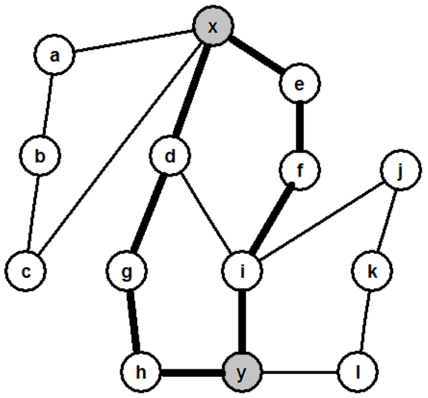

Fig. 6

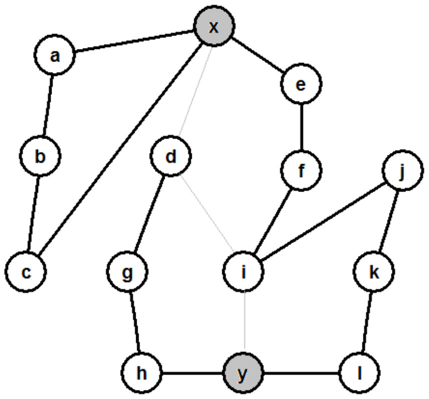

Fig. 7

To formulate an algorithm determining the shortest circle containing two given vertices can be quite an uneasy task, see http://cs.stackexchange.com/questions/13194/

PUZZLE

Let us finish the paper with an example whose solution uses the above described subject matter.

Example

In the graph *G* in Fig. 9 find *an odd circle containing the vertex B.*

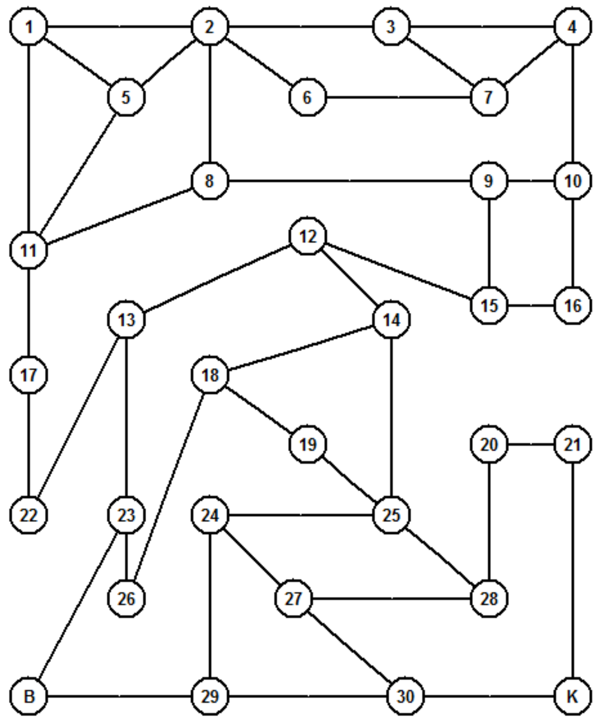

Fig. 9 A given connected undirected graph *G*

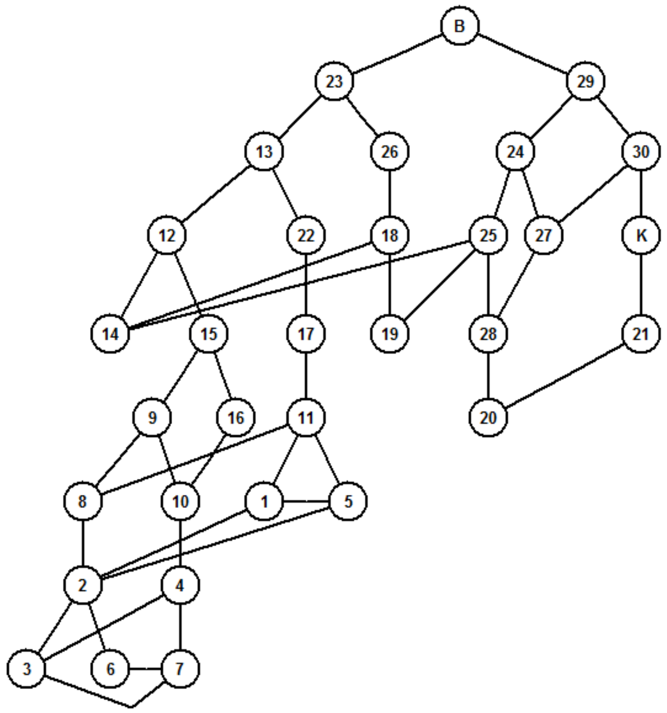

Fig. 10 BFS tree (*T*, *B*) completed by non- tree edges

Solution

Although there is no *non-tree edge having its end-vertices* ***both*** *in the same level of (T, v) and in different (T, v)-subtrees* in the appropriate BFS tree (*T*, *B*) (see Fig. 9), we can see that there is a *non-tree edge having its end-vertices both in the same level of (T, v)*, e.g. {1, 5}, as well as a *non-tree edge having its end-vertices in different (T, v)-subtrees*, e.g. {19, 25}. Due to this fact, in spite of a demanded circle is not visible at once, we go on and try to find it.

One solution is given in Fig. 11.

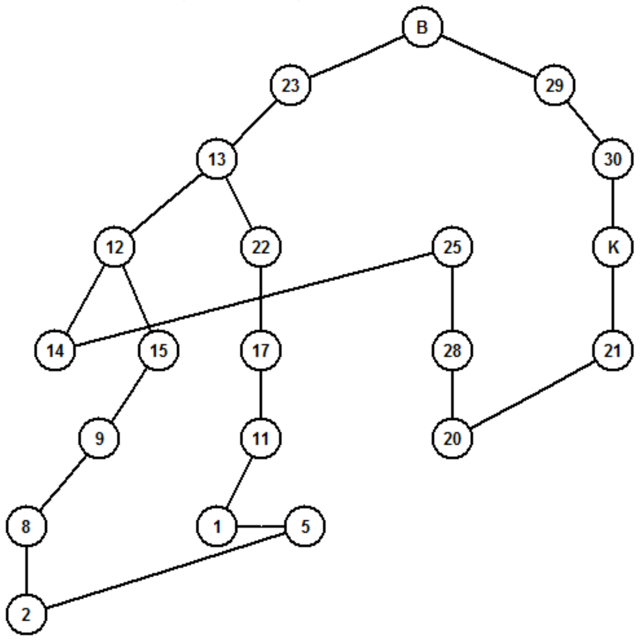

Fig. 11 Solution of the task

Conclusion

In the paper we explored both the levels and subtrees of the BFS Tree appropriate to the BFS algorithm. We formulated polynomial time algorithms directly following from the BFS Tree property. We also mentioned the exponentially growing problem of finding all the shortest paths existing in the given graph.

On two examples, despite their intuitive simplicity, we have shown that solving them can be quite uneasy.

Students usually have problems with theoretical parts of mathematics; they avoid studying proofs of theorems and are willing only to remember them without proper understanding their meanings, even the meanings of theorem formulations themselves. Therefore it is important to practise students' comprehension and skills on simple statements proofs, and to emphasize the meaning of both necessary and sufficient conditions, necessary conditions that are not sufficient, sufficient conditions that are not necessary. It is important to devote enough time to a proper analysis of a discussed subject matter.

An intended topic for future research is to investigate problems relating to the BFS Tree property without having both sufficient and necessary conditions supporting their solutions.

Acknowledgment

This research has been supported by Specific research project of the University of Hradec Kralove, Faculty of Science No. 2113.

REFERENCES

[1] J, Novotná, P. Eisenmann, J. Přibyl, J. Ondrušová, and J. Břehovský, “Problem solving in school mathematics based on heuristic strategies,” Journal on Efficiency and Responsibility in Education and Science, vol. 7, no. 1, 2014, pp.1-6.
[2] P. Zeitz, “The Art and Craft of Problem Solving,“ New York: John Wiley & Sons, Inc., 2007.
[3] H. Cormen, CH. E. Leiserson, R. L. Rivest, C. Stein, “Introduction to Algorithms,“ The MIT Press, 2009.
[4] Demel, J.: Grafy a jejich aplikace (Graphs and their Applications). Academia (2002).
[5] Kučera, “Combinatorial Algorithms (Kombinatorické algoritmy),“ SNTL, Praha, 1989.
[6] W. Lipski, “Combinatorics for Programmers (Комбинаторика для програмистов),“ Мир, Москва 1988.
[7] F. S. Roberts, B. Tesman, “Applied Combinatorics,“ 2nd Edition, Pearson Prentice Hall, Upper Saddle River, NJ, 2004.
[8] S. Skiena, “The Algorithm Design Manual,“ New York: Springer-Verlag, 1998.
[9] Tarjan, R. E.: Data structures and network algorithms. SIAM, Philadelphia (1983).
[10] E. Milková, “BFS Tree and x-y Shortest Paths Tree,” in Proc. of International Conference on Applied Computer Science (ACS), WSEAS Press, Malta, September 15-17, 2010, pp. 391–395.
[11] E. Milková, “Combinatorial Optimization: Mutual Relations among Graph Algorithms,“ WSEAS TRANSACTIONS on MATHEMATICS, Issue 5, Volume 7, May 2008, pp.869-879.