Proceedings of the International Conference IMCS-60, MSU
October 10–13, 2024, Chisinau, Republic of Moldova

Some Applications of the Depth-First Search Algorithm for Undirected Graphs

Constantin Ciubotaru

Abstract

Applications of the deep first search algorithm (DFS) have been developed that allow at a single traversal of the graph: a) to check the connectivity/biconnectivity of the graph, b) to build the spanning tree, c) to highlight the cut vertices, d) to calculate the biconnected components, e) to build the fundamental set of cycles.

The proposed algorithms were programmed and tested.

Keywords: undirected graph, biconnected graph, spanning tree, DFS, cut vertices, fundamental cycles.

1 Introduction

DFS is an algorithm for depth traversal of a tree or graph [1, 2]. Initially the depth traversal method and spanning trees were proposed by the French mathematician Charles Pierre Trémaux (19th century) to solve the labyrinth problem.

The algorithm is also requested to solve many problems based on the processing of tree structures. For example, the Travelling Salesman problem, the realization of the Ford-Fulkerson algorithm and the backtracking algorithms, the solution of some artificial intelligence problems, web-crawling, etc. Lately, the algorithm has been successfully used to check the planarity of graphs and graph drawing [3, 4, 5, 6].

In the following, we present a modified version of the DFS algorithm taking into account the ideas presented in [4], which allows us in a single traversal to test connectivity and biconnectivity of a graph, highlight cut vertices and biconnected components, and build the fundamental set of cycles.

©2024 by Constantin Ciubotaru

2 Preliminary notions

The undirected graph will be denoted by $G = (V, E)$, and the adjacency list of any vertex v – by $adj(v)$. A sequence of vertices with the property that any two consecutive vertices are adjacent is called *path*: $(v_1\, v_2\, v_3 \ldots v_n)$ or $((v_1\, v_2)\, (v_2\, v_3)\, \ldots\, (v_{n-1}\, v_n))$, where: $(v_i, v_{i+1}) \in E$ for $1 \leq i < n$. The empty path is denoted by *nil* or (). A path is called *elementary* (*simple*) if all participating vertices (edges) are distinct.

A path in which the first vertex coincides with the last is called *cycle*. The cycle is *elementary* if it consists only of distinct vertices, excluding the first and the last. We will also use the traditional definitions for *connected* and *biconnected* graph, *bicomponent*.

The tree $T = (V, E_t)$ is called a *spanning tree* for the graph $G = (V, E)$ if it contains exactly all the vertices of G and E_t is obtained from E by omitting some edges, $E_t \subseteq E$.

The omitted edges form the set of *back edges*. Adding one back edge to the spanning tree will generate a unique cycle. This cycle is called $fundamental\ cycle$. The set of all fundamental cycles represents the $fundamental\ set\ of\ cycles$.

3 The modified DFS algorithm

The variable *stack* is used to collect all bicomponents. The final set of bicomponents is denoted by *bicomponents*. For collecting the lists of cut vertex and back edges the variables *cutvertex* and *backedges* are used. The set *notconnected* at the end of algorithm will contain the list of all unreachable vertices from the root (if the graph is not connected or $\varnothing$ otherwise). Three additional lists of size $|V|$ are introduced: *tree*, *ord*, and *up*.

When traversing the graph, any vertex v receives the number which corresponds to its visit order, using counter *visit* and list *ord*. The root will be visited first, assigning its visit the order equal to 0. For a connected graph, any vertex can be considered the root, thus obtaining multiple spanning trees for the same graph. It is clear that any ancestor

u of vertex v $(u\,v)$ or $(u\,v_1\,v_2\ldots v_n\,v)$ has a visiting number less than v, $ord(u)< order(v)$. For any vertex v, we denote by $tree(v)=u$ the unique direct ancestor of the vertex v in the tree. Thus, the final value of the vector *tree* will correspond to the spanning tree. Incidence lists *sptree* could be also obtained, if necessary.

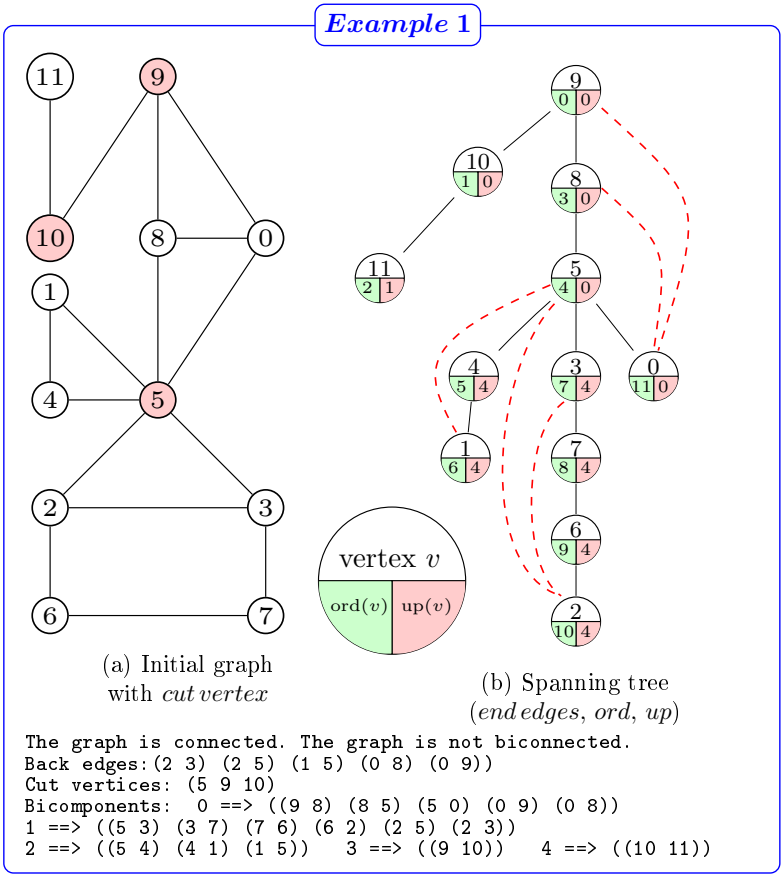


Figure 1. Initial graph with cut vertex and spanning tree

An important structure presents the vector *up*. For each vertex v, $up(v)$ will receive the *ord* value of the closest ancestor to the root that can be accessed from v using at most one back edge. First, all initial

values were assigned. At this stage, for any vertex v, there will be set $ord(v) = -1$, $up(v) = -1$, $tree(v) = -1$, $sptree(v) = nil$. $ord(v) = -1$ means that vertex v has not yet been visited. Exception for *root*: $order(root) = 0$, $up(root) = 0$, $tree(root) = root$, conventional. For the sets *cutvertex*, *notconnected*, and *bicomponents*, the initial value is set to $\varnothing$, and for backededges, it is set to E. If i is direct descendant of k and $up(i) \geq ord(k)$, then k will be cut vertex (Figure 1). An important property of the *root* should be mentioned. The *root* of a *sptree* is a cut vertex if and only if $sptree(root) > 1$.

At the next stage, the sets *ord*, *up*, *cutvertex*, and *tree* are built. Also, *stack* is managed. At the time of the appearance of a new bicomponent in the top of the *stack*, the bicomponent is extracted and placed in the *bicomponents*. Finally, the connectivity/biconnectivity of the graph was checked, the set *backedges* and the lists *sptree* were built. Also, the check is performed if the *root* is a cut vertex.

The results of performing the algorithm are presented in Figure 1 (Example 1): (a) – initial graph and cut vertices, (b) – spanning tree, back edges, *ord* and *up* values, bicomponents.

The formal proof of the correctness of the algorithm can be found in [4].

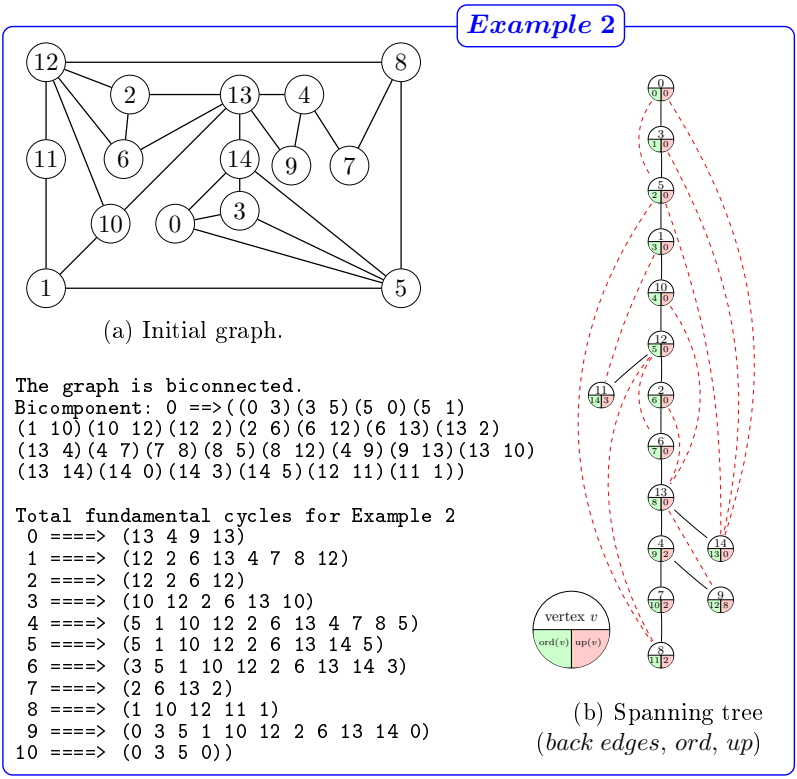
4 Building the fundamental set of cycles

To build the fundamental set of cycles, the idea of the Paton algorithm [7] was used taking into account that the spanning tree and the set of back edges are already constructed. One end edge added to the spanning tree generates a unique fundamental cycle. So, for any end edge, there exists one fundamental cycle, so that a one-to-one correspondence between fundamental cycles and back edges is obtained.

The variable *cycles* is used to store all the fundamental cycles built. For each vertex u of the spanning tree, a list of paths $l = (uv_1 \dots v_i v)$, $i \geq 1$ (checkup list) with generation prospects of cycles is built. This set can be built not for all vertices u, for example, if $sptree(u)$ or/and $sptree(v)$ are empty. In Figure 2 (Example 2), the vertices 4,6,7,8,9,11,14 satisfy this property. The following situations are possible:

(1) If $sptree(v) \neq \varnothing$, then for any $z \in sptree(v)$, a new path is

generated l=($u\ v_1 \ldots v_i\ v\ z$), candidate for generating new cycles. (2) If $u \in bef(v)$ ($bef(v)$ denotes the set of back edges for vertex v), then a new cycle is obtained



(a) Initial graph.

(b) Spanning tree (*back edges*, *ord*, *up*)

```
The graph is biconnected.
Bicomponent: 0 ==>((0 3)(3 5)(5 0)(5 1)
(1 10)(10 12)(12 2)(2 6)(6 12)(6 13)(13 2)
(13 4)(4 7)(7 8)(8 5)(8 12)(4 9)(9 13)(13 10)
(13 14)(14 0)(14 3)(14 5)(12 11)(11 1))

Total fundamental cycles for Example 2
 0 ====> (13 4 9 13)
 1 ====> (12 2 6 13 4 7 8 12)
 2 ====> (12 2 6 12)
 3 ====> (10 12 2 6 13 10)
 4 ====> (5 1 10 12 2 6 13 4 7 8 5)
 5 ====> (5 1 10 12 2 6 13 14 5)
 6 ====> (3 5 1 10 12 2 6 13 14 3)
 7 ====> (2 6 13 2)
 8 ====> (1 10 12 11 1)
 9 ====> (0 3 5 1 10 12 2 6 13 14 0)
10 ====> (0 3 5 0))
```

Figure 2. Fundamental cycles

($u\ v_1\ \ldots v_i\ v\ u$), which is included in the set *cycles*. In this case, the back edge ($u\ v$) is excluded from *beckedges*, and the path l is excluded from the checkup list.

(3) If $sptree(v) = bef(v) = \varnothing$, then l is removed from the checkup list and another candidate path will be examined. The process will continue as long as there are new chains in checkup list.

5 Conclusion

The advantage of the algorithm lies in the possibility of verification the connectivity/biconnectivity of undirected graphs, construction of a set of characteristics specific to undirected graphs performing a single traversal of the graph. The functional implementation of the algorithm can be convenient when developing other applications. Most of the functions having an intrinsic structure can be used separately. In the future, it is expected to use this algorithm to develop new methods for checking planarity and graphs drawing.

Acknowledgments. The project SIBIA 011301 has supported a part of the research for this paper.

References

[1] S. Even, G. Even. *Graph Algorithms*, Second Edition. Cambridge University Press, 2012, ISBN 978-0-521-73653-4, pp. I–XII, 1–189.

[2] A. Gibbons. *Algorithmic Graph Theory*, Cambridge University Press, Cambridge, 1985, 259 p.

[3] H. de Fraysseix, P. Ossona de Mendez, P.Rosenstiehl. *Trémaux trees and planarity*. International Journal of Foundations of Computer Science, vol. 17, no. 5, pp. 1017–1029, 2006.

[4] John Hopcroft, Robert Tarjan. *Efficient planarity testing*. Journal of the ACM, vol. 21, no. 4, pp. 549–568, October 1974.

[5] Takao Nishizeki, Md. Saidur Rahman. *Planar Graph Drawing*. Lecture Notes Series on Computing 12, World Scientific, 2004, ISBN 981-256-033-5, pp. 1–312.

[6] Roberto Tamassia (Ed.) *Handbook of Graph Drawing and Visualization*. Published by Chapman & Hall, 2016, pp. 1–862.

[7] K. Paton. *An algorithm for finding a fundamental set of cycles of a graph*. Commun. ACM, vol. 12, no. 9 (Sept. 1969), pp. 514–518.

Constantin Ciubotaru

Moldova State University, Vladimir Andrunachievici Institute of Mathematics and Computer Science
E–mail: constantin.ciubotaru@math.usm.md, chebotar@gmail.com
ORCID: https://orcid.org/0009-0005-8896-0966