

DSA3 seminar paper

Radoman Radoman 157m/21

University of Novi Sad, Faculty of Sciences

radoman88@gmail.com

1 Introduction

2 BFS and DFS

2.1 Purpose and theoretical background

Both BFS (breadth-first search) and DFS (depth-first search) are tree-search algorithms. They are both used to determine whether a graph is connected.

Theorem

A graph is connected if and only if, you can not split it into two such that there are no edges between them.

An approach to check whether a graph is connected is by checking all the partitions (the problem is that there are 2^{n-1} of them so it is very inefficient). Another way to check whether a graph is connected is by checking all pairs of vertices - if there is a path connecting them (problem here again is that there are n^2 pairs of vertices and for each of them we have to check if there is a path which is also very inefficient on top of number of partitions that we would need to check).

Definition

A graph is connected if and only if it has a spanning tree.

Checking whether a graph is connected in such a way (by determining whether it has a spanning tree) is far and away the most efficient way to check graph connectedness.

Both DFS and BFS work in a similar way (they both produce spanning trees), but with differences that impact the priority of edges selected. Depending on the way we used to choose edges we get different trees.

A **tree-search** algorithm on G :

- > Start with a single root vertex $r \in V(G)$. This is our initial tree (with just one vertex).
- > Repeat (for as long as possible):
 - Do we have a spanning tree?
 - Is the tree edge cut empty?
 - If not, add one edge from the cut to the tree.

If we want to check *connectedness* of G , that is the whole algorithm. As noted above, depending on the way we use to choose edges, different spanning tree will result (if the graph is connected of course).

Define: edge cut, rooted tree (also called r-tree), levels (distance from root to a specific vertex), ancestor, descendant, parent or predecessor and children.

$$v \in V$$

Predecessor function: $p(v)$, for all $\{r\}$

2.2 BFS

Adjacency lists of vertices are considered on a first-come-first serve basis.

Implemented with a *queue*:

Start with just the root in the queue.

Repeatedly pop the head of the queue, and push all its new neighbors to the queue.

For a connected graph, the algorithm will return:

- A spanning tree given by its predecessor function,
- the level function,
- the time function (order in which vertices are added to the tree)

2.2.1 BFS algorithm

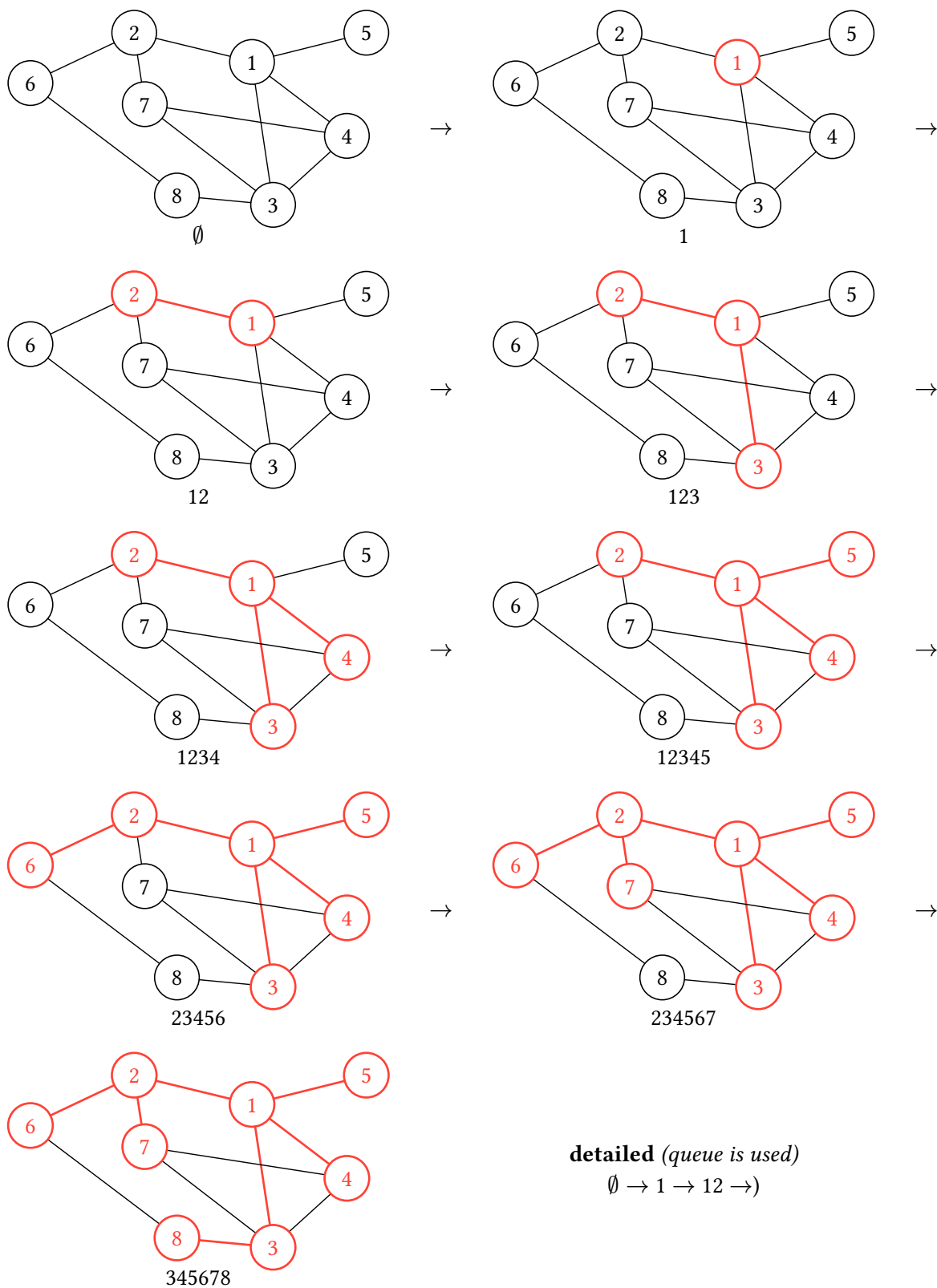
> INPUT: a connected graph G , a vertex $r \in V(G)$

> OUTPUT: an r -tree $T \in G$, its predecessor function p , its level function l , the time function t .

$Q := \emptyset, Q \leftarrow r, l(r) := 0, t(r) := 1, \text{mark } r, i := 1$

while $Q \neq \emptyset$

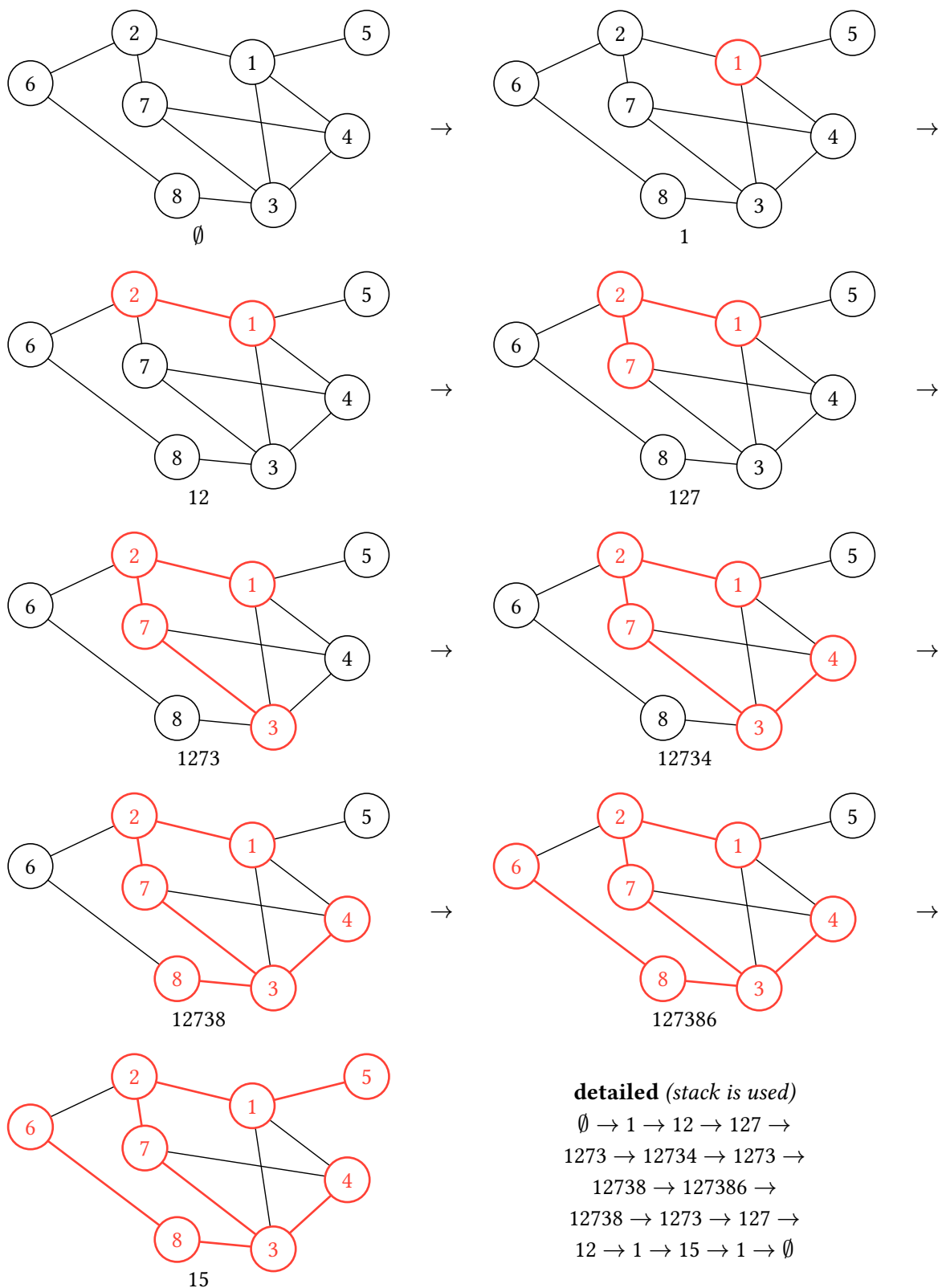
2.2.2 BFS example



2.3 DFS

2.3.1 DFS algorithm

2.3.2 DFS example



detailed (*stack is used*)

$\emptyset \rightarrow 1 \rightarrow 12 \rightarrow 127 \rightarrow$
 $1273 \rightarrow 12734 \rightarrow 1273 \rightarrow$
 $12738 \rightarrow 127386 \rightarrow$
 $12738 \rightarrow 1273 \rightarrow 127 \rightarrow$
 $12 \rightarrow 1 \rightarrow 15 \rightarrow 1 \rightarrow \emptyset$

3 Applications

4 Conculsion