Relabel-to-front algorithm

Marek Višňovský xvisno00@stud.fit.vutbr.cz
Jan Wozniak xwozni00@stud.fit.vutbr.cz

1 Introduction

This projects purpose was to create simple application for education and visualisation of Relabel-to-front algorithm. It is implemented purely in Java using Netbeans IDE.

2 Theoretical Analysis

The Relabel-to-front is quite efficient algorithm that computes a maximal flow in oriented graph[1]. It is a special variant of push-relabel, where the algorithm maintains list of overflowing nodes. The list is scanned as the algorithm proceeds and the excess flow in each node is discharged [2].

The main function of the algorithm is *relabel-to-front* which takes three arguments, graph G, source node s and sink node t as could be seen in algorithm 1. The operation *discharge(u)* in algorithm 2 is essential core of the algorithm, takes a node u and modifies heights and flow in the graph by chosing suitable push and relabel operations to ensure entire excess flow is taken care of. If during discharge a node is relabeled, it is moved to the start of the list. The algorithm then treats the next node in the list and continues the procedure until every overflowing node has been discharged. Function *push(u, v)* pushes through edge from node u to node v the most excess flow possible. Function *relabel(u)* changes node's u height accordingly.

During initialization height of each node is set to 0 except source, which is set to n. Flow on all edges is set to 0. Then we push the most possible flow on all edges from source and then continue according to the rest of algorithm 1.

Algorithm 1 Relabel-to-front(G, s, t)

```
initialize preflow
initialize node list L containing V − {s, t}
while u ≠ null do
    u ← L.get_next
    old_height ← u.height
    discharge(u)
    if old_height ≤ u.height then
        L.set_first(u)
    end if
end while
```

Algorithm 2 Discharge(u)

```
while u.excess > 0 do
    v ← u.next_neighbour
    if v ≠ null then
        relabel(u)
    else
        if (u, v).residual > 0 && u.height > v.height then
            push(u, v)
        end if
    end if
end while
```

Computational Complexity

Every discharge operation without relabel advances u, the current node within list L. Then if we have n discharge operations without relabel, we have the current node $u = null$ and the algorithm terminates. Therefore, the number of calls to discharge is at most $n(\#relabels + 1) = O(n^3)$.

A relabel operation at a node is constant time – increasing label and setting current-neighbour. In total we have $O(n^2)$. The cost of push operation is at max $O(n^3)$.

Concluding from the informations above the complexity of relabel-to-front algorithm is $O(n^3)$ where n is number of nodes [3].

3 Implementation

The program runs in two separate threads, one is GUI and second is relabel to front algorithm. Entire project is separated into two packages, `GUI` and `Graph`. Graph package contains following classes

- *Graph* – basic representation of simple oriented graph
- *Edge* – representation of edges
- *Node* – representation of nodes
- *GraphListener* – interface for communication between UI
- *RelabelToFront* – class extending graph of relabel to front algorithm implementation

The `RelabelToFront` stores as protected arrays for flow, capacity, height, excess and the list of nodes to discharge. The redundant structure is created for easier reinitialization of algorithm, if the user changes the nodes in graph and restarts algorithm over again.

4 User Interface

Our main goal was to create simple and intuitive, yet powerful user interface. We tried to achieve this by minimization of the number of control elements down to 2 panels, one for graph building purposes and another one for algorithm control.

Creating a simple graph

Left panel of application's main window serves for graph creation. Users are able to add or remove nodes from pop-up menu, that shows up after right click of the mouse on the desired location. Nodes are represented as black circles with its id displayed inside. Right next to them, there's an indicator of current height of the node, which plays role in algorithm itself.

Right clicking on the node invokes the same pop-up menu, but with more options enabled, such as deletion of selected node, starting edge from that node, selection of this node as source or sink of the graph.

Edges are displayed as lines, with an arrow in the middle pointing in the direction of that edge, as relabel-to-front algorithm works with oriented graphs. Next to this arrow, there are two numbers separated by slash, indicating current flow and capacity of the edge. Edges are created from pop-up menu mentioned above, after right clicking on desired start node and then selection of the destination node (this is done again by clicking with any mouse button on desired node). Loops are not allowed.

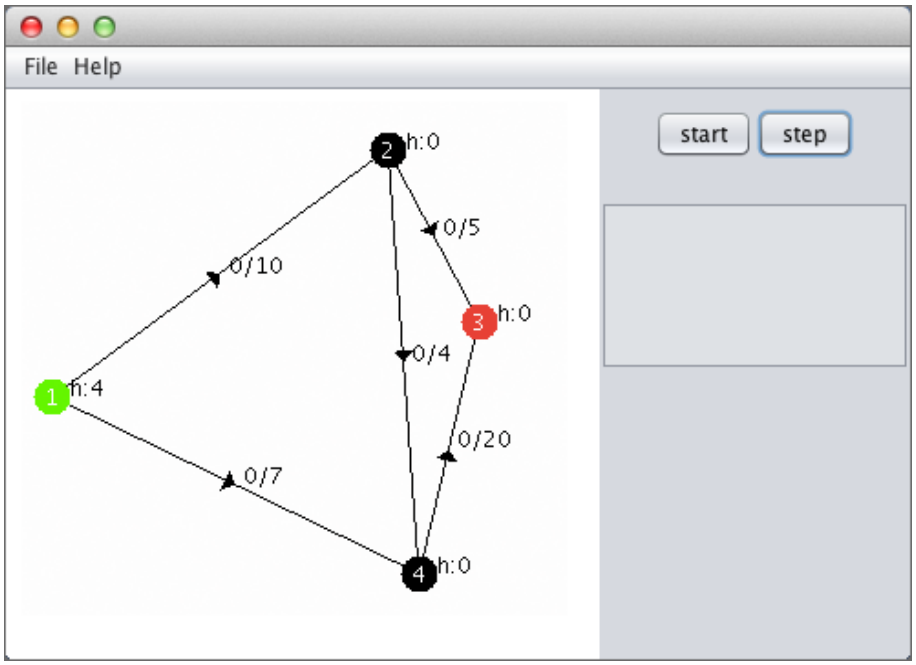

Figure 1: Simple graph created in our application. Node 1 is selected as source, node 4 as sink.

Saving and loading graphs

Application allows to store and load graphs in GraphML format. This is a widely-used xml format for graph representation. More info about this format can be found here: http://graphml.graphdrawing.org/. The application can load GraphML files created by an application itself, other files must contain definition of additional attributes such as x_position, y_position and height for nodes, and capacity for edges. For more details, look at the saved file itself, the structure is, thanks to xml format, easily readable.

Running an algorithm

To run relabel-to-front algorithm, one must select source and sink nodes first. Source node will be colored green and sink node red. After that, users can run algorithm simply by clicking on the start button in the panel on the right. This will initialize all variables necessary for the algorithm. Then, algorithm continues by simple steps after clicking on the step button. Changes to the graph are highlighted:

- node, which turns yellow, had its height changed in the last step (height number turns to red)
- edge, which flow has changed, is highlighted by yellow border, and its flow and capacity marking is written in red

There are steps of the algorithm, when relabel-to-front works with virtual edges, which are not visible to the user. Information about these changes (and also about all visible changes) can be found in text area below the control buttons. Final flow is displayed here too.

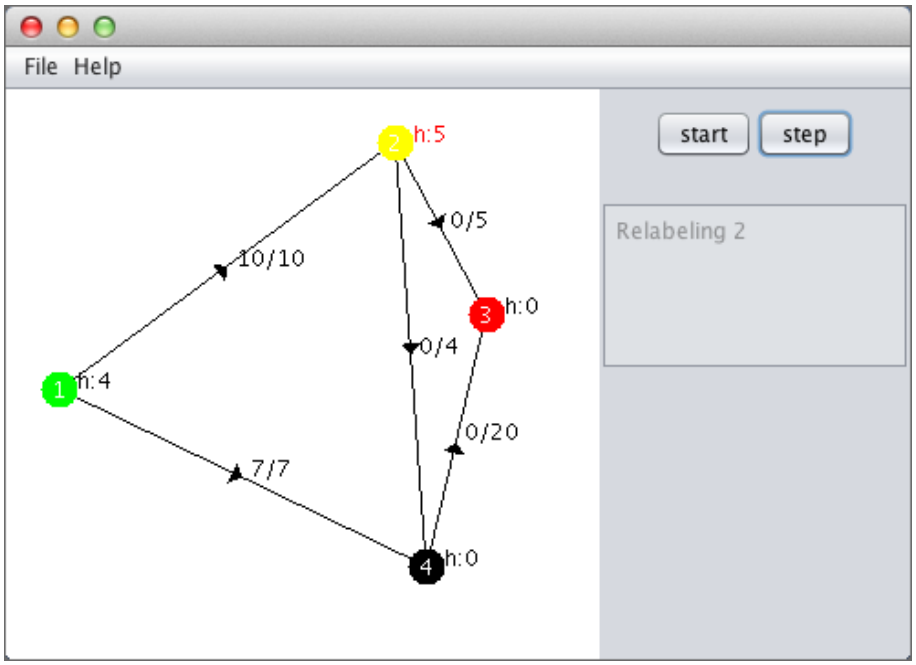

Figure 2: Running an algorithm. Node 2 is being relabeled, so it is painted in yellow.

5 Conclusion

The application visualizes relabel to front algorithm without any optimalization in comprehendable way for a user with basic knowledge of graphs and flow in graphs. It allows user to create or import graph and then execute step by step the algorithm which produces helpful notes for the user.

Literature

[1] *Wikipedia – the free encyclopedia.* [cit. 2012-12-11]
<http://en.wikipedia.org/wiki/PushE2%80%93relabel_maximum_flow_algorithm>

[2] A. Bhusnurmath. *Applying Convex Optimization Techniques to Energy Minimization Problems in Computer Vision.* [cit. 2012-12-11]
<http://www.cis.upenn.edu/~cjtaylor/PUBLICATIONS/pdfs/BhusnurmathPHD08.pdf>

[3] E. Mayr, H. Räcke. *Relabel to front – Lecture.* [cit. 2012-12-10]
<http://wwwmayr.informatik.tu-muenchen.de/lehre/2011WS/ea/split/sub-Relabel-to-front-single.pdf>