User Manual for union-find

Allen Bates, *Phillip Conrad*, *Janice Neighbor*, *William Warren*

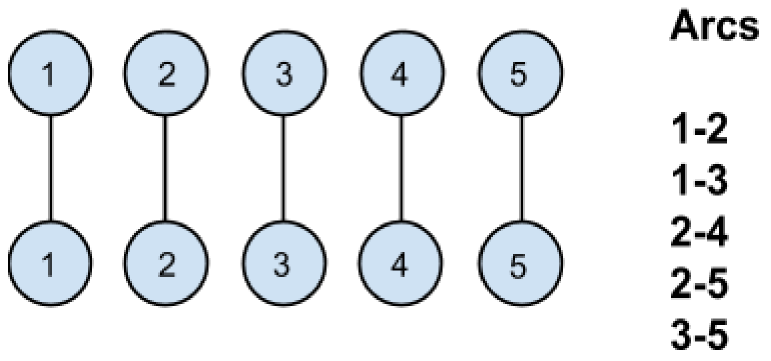
Introduction

Graph problems related to cycles are common in Computer Science. Cycle detection is used in everything from routing circuits in chip designs to finding loops in network topologies and programs. This program was designed to use the Union-Find algorithm to identify the independent cycles in an undirected graph.

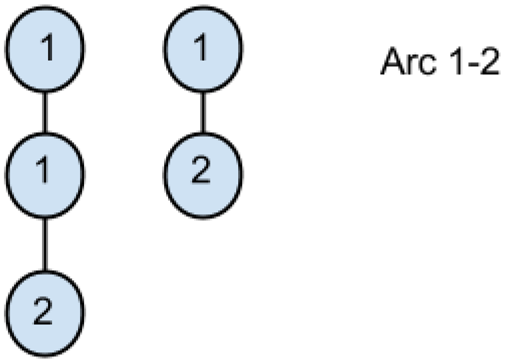
Algorithm

The Union-Find algorithm works in two stages: forest creation, then the arc-adding process.

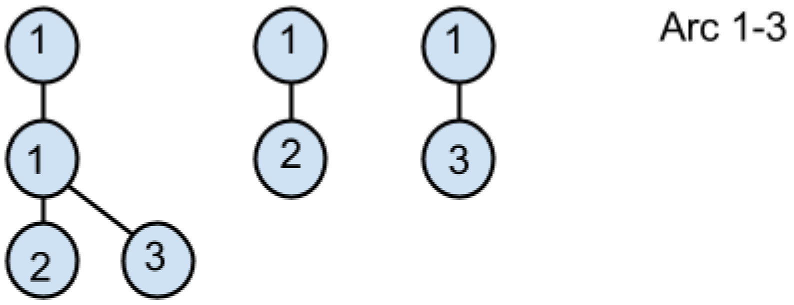
We begin by iterating over the input and create a new node instance (linked with itself as parent) for each unique node id encountered. This is our initial forest.



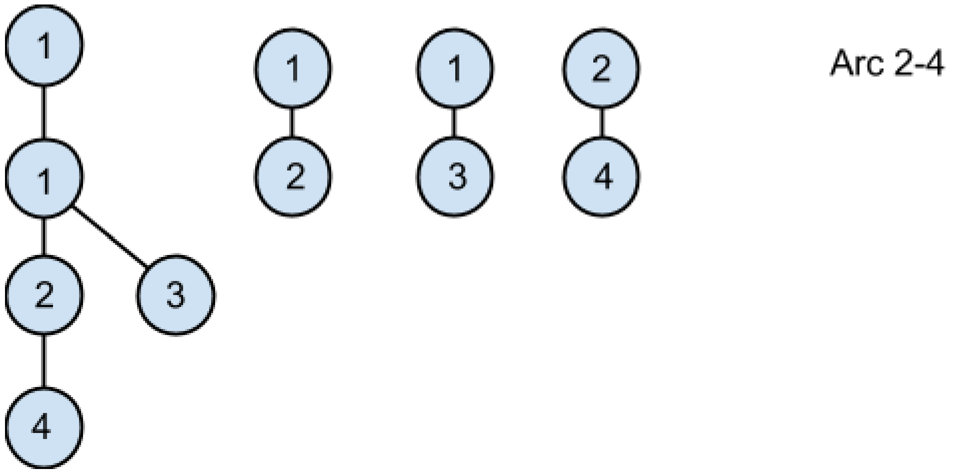
We now begin the process of building trees by “unioning” arcs and their subnodes together into progressively bigger trees.



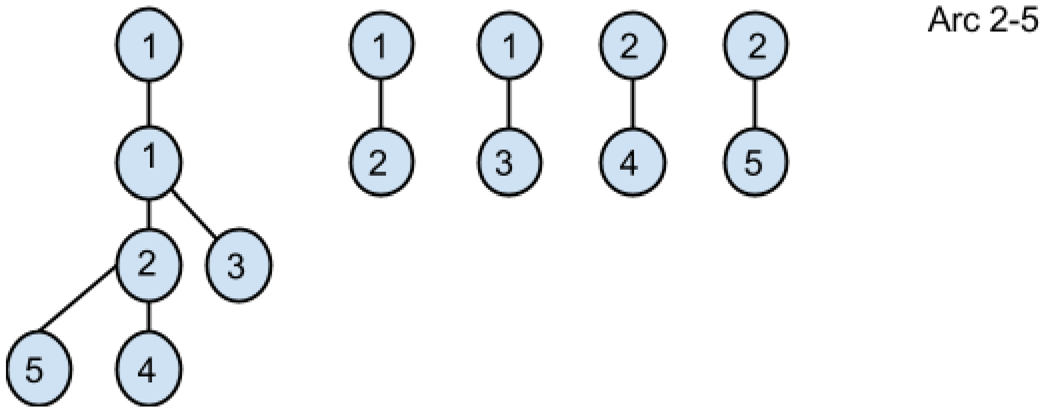
This process can be best described as “re-parenting” of nodes, and for each arc (A ← B), we reparent node B so that its parent becomes node A.



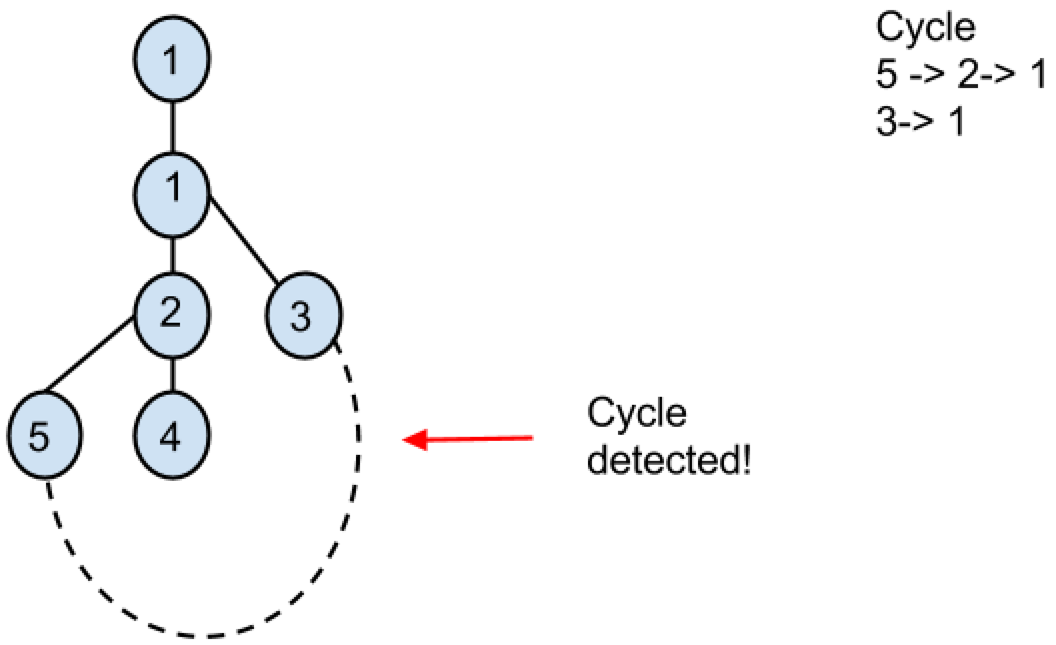
In order to know if it’s safe to reparent a node, we trace down both to the root of both the trees for node A and node B. If the roots differ, then it is safe to reparent B to A.



However, if the roots are the same, then reparenting B to A would create a cycle, and the program outputs the offending arcs involved in the cycle. Note that it does NOT add the offending arc! To do so would be self-defeating.



This process of unifying trees by reparenting nodes continues until all of the arcs in the input data have been processed, and all cycles detected have been output.



Software Implementation

Node Class

The *Node* class allows us to creates the nodes of the graph. Each has two integer members, one for its own id, and the other for its parent.

Arc Class

The *Arc* class is a simple carrier for our input data. Its **toString()** method is used extensively in the *UnionFind* class for dumping cycles to output.

UnionFind Class

The *UnionFind* class is the class which does most of the work. Its member functions correspond roughly to the stages of the Union-Find algorithm.

The first function called in practice is the **addArc()** function, which is used to build the vector of input arcs, and the **isArcUnique()** helper function prevents duplicate arcs from appearing in that vector.

Once the arcs have been created, the **buildForest()** function is called to construct the initial forest of nodes. It iterates through all of the arcs, and adds each node id to a set of node ids. After running through the data, the function then constructs a map of nodes and their corresponding integer ids (specifically member: **map<int, Node> nodes**).

At this point, the programmer is expected to call the **unionFind()** function to actually run the algorithm over the input data.

Internally the **unionFind()** function calls the descriptively named **addArcToForest()** function once for each Arc in the input data set.

The **addArcToForest()** function internally calls the **find()** function twice, once for each node id in the Arc instance provided.

The **find()** function recursively finds and returns the root of each tree, and if the two roots are the same (e.g. have the same id), then adding the Arc would cause a cycle. The function then outputs the arcs involved in the cycle.

If the roots differ, however, it is safe to add the arc, and the appropriate reparenting operation is run.

Once the **unionFind()** function is done adding all of the arcs to the tree, the programmer can then print out the cycles by calling the **dumpTrees()** function, which simply returns a string of all the cycles detected up to that point.

Operators Class

The operators class was created to save us some typing when outputting nodes. For example:

cout << node;

instead of

cout << node.toString();

Main Program

The capstone of our efforts is the Main class. When run, it reads in the arc data from the file specified by the user.

The first line of said input file tells the user how many arcs it contains. The program then uses that number as a parameter of a loop which reads in each pair of values (a, b), and calls **addArc()** once for each pair.

The main program then calls the **buildForest()** function to create the initial forest of nodes. It then calls the **unionFind()** function to add all of the input arcs to the forest and detect any cycles present. Lastly, the program calls **dumpTrees()** to output the cycles it found.

Interpreting the output

After the program is run the output a cycle will show the arcs down each node’s path to its root, then the offending arc. An example is shown below:

Input Data:

```
0 0
1 0
2 0
1 2
```

Output:

```
A: (  1 ->   0)(  0 ->   0)
B: (  2 ->   0)(  0 ->   0)
Bad Arc: (  1 ->   2)
```

This can be read as:

> “Path A starts at node 1, and has an arc to node 0. Once at node 0, we can see that we're at the root of the path, since the node is linked to itself. For path B the same procedure is followed, arriving at node 0 as the root. Since both root nodes are the same, we have a cycle, caused by the offending arc (1 → 2).”

Citations

- "Cycle Detection." *Wikipedia*. Wikimedia Foundation, 21 Apr. 2014. Web. 22 Apr. 2014. <http://en.wikipedia.org/wiki/Cycle_detection>.

- "Union-Find Algorithm **|** Set 1 (Detect Cycle in a an Undirected Graph)." *GeeksforGeeks*. Ed. Aashish Barnwal. Geeks for Geeks, n.d. Web. 22 Apr. 2014. <http://www.geeksforgeeks.org/union-find/>.

- Buell, Duncan. "Union Find Algorithm." CSCE240. University of South Carolina, Columbia. 15 Apr. 20014. Lecture.