PRINCETON UNIVERSITY

Summary: Algorithms, Part I

Francisco Manuel Garcia Botella
fmgarcia@dtic.ua.es

February 7, 2016

1 CHAPTER 1: UNION-FIND

1.1 BEFORE DEVELOP ALGORITHMS

1. Model the problem. For example: the main elements of the problem that need to be solved.
2. Find an algorithm to solve it.
3. Is it fast enough? Or fits in memory? In many cases, the first algorithm is enough, but others not.
4. If not, figure out why. Check the requirements we need to it. (Fast algorithm, or fit memory)
5. Find a new way to address the problem. We find a new algorithm and check if it works fine.
6. Iterate until satisfied. The algorithm works fine.

We will check if the algorithm works fine. Once we have solved and checked the algorithm, we work in optimize this algorithm, and re-check.

1.2 Dynamic Connectivity

The **Dynamic Connectivity**(**DC**) is a data structure that dynamically maintains information about the connected components of a graph.
The set V of vertices of the graph is fixed, but the set E of edges can change. The three cases, in order of difficulty, are:

- **Incremental connectivity**: Edges are only added to the graph
- **Decremental connectivity**: Edges are only deleted to the graph.
- **Fully connectivity**: Edges can be either added or deleted.

After each addition or deletion of an edge, the DC structure should adapt itself such that it can give quick answers to queries.

1.3 Union-Find

The **Union-Find Algorithm**(**UFA**), is a algorithm which uses a disjoiont-set data structure to solve a problem. For example: We have a set of N elements, we are allowed to merge any two items to consider them equal, i.e., we do that two elements belong to the same group that share the same feature. Too, we are allowed to ask whether two items have the same feature. This algorithm is very slow.

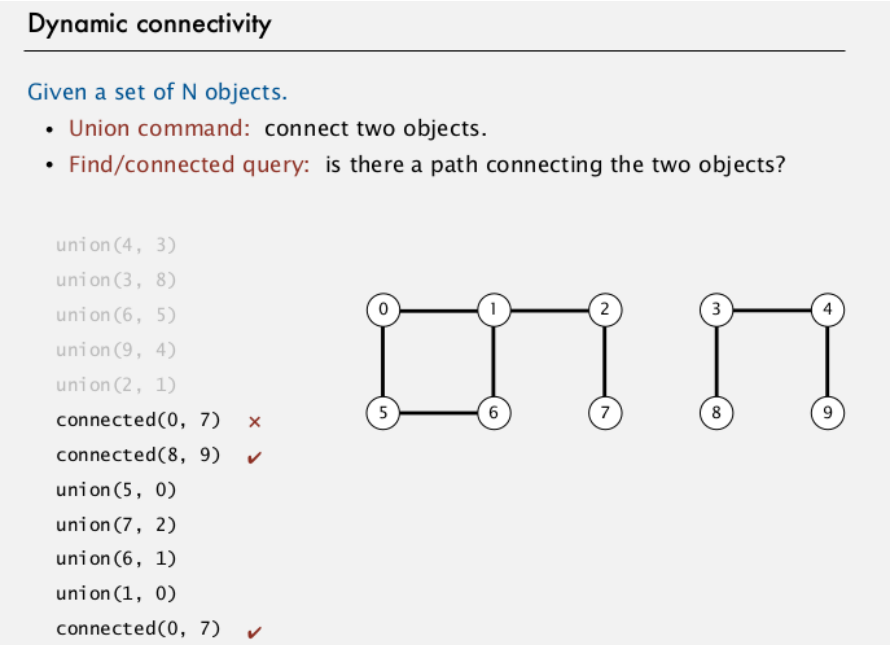

Figure 1.1: Example of Union-Find

To do this, we use the following functions:

	0	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
4-8:	0	1	2	3	8	5	6	7	8	9
9-0:	0	1	2	3	8	5	6	7	8	0
6-1:	0	1	2	3	8	5	1	7	8	0
8-2:	0	1	2	3	2	5	1	7	2	0
9-3:	3	1	2	3	2	5	1	7	2	3
6-3:	3	3	2	3	2	5	3	7	2	3

Table 1.1: Exercise Union-Find

- **union(*A*,*B*)** - Merge A's set with B's set.
- **Find(*A*)** - Finds what set A belong to.
- **Connected(*A*,*B*)** - Check if A edge is connected with B edge.

1.3.1 Exercise Union-Find

Question 1(Seed = 888220): Give the id[] array that results from the following sequence of 6 union operations on a set of 10 items using the quick-find algorithm.

4-8 9-0 6-1 8-2 9-3 6-3

Your answer should be a sequence of 10 integers, separated by whitespace.
Recall: our quick-find convention for the union operation p-q is to change id[p] (and perhaps some other entries) but not id[q].
Response: The answer is: **3 3 2 3 2 5 3 7 2 3**
Here is the id[] array after each union operation:

1.4 Quick Find

1.5 Quick Union

1.5.1 Exercise Quick-Union

Question 2(seed = 653576): Give the id[] array that results from the following sequence of 9 union operations on a set of 10 items using the weighted quick-union algorithm from lecture.

9-4 2-5 5-8 1-0 7-2 5-3 4-0 3-0 4-6

Your answer should be a sequence of 10 integers, separated by whitespace.
Recall: when joining two trees of equal size, our weighted quick union convention is to make the root of the second tree point to the root of the first tree. Also, our weighted quick union algorithm performs union by size (number of nodes) - not union by height - and does not do path compression.
The correct answer is: 1 9 2 2 9 2 2 2 2 2
Here is the id[] array after each union operation:

	0	1	2	3	4	5	6	7	8	9
9-4:	0	1	2	3	9	5	6	7	8	9
2-5:	0	1	2	3	9	2	6	7	8	9
5-8:	0	1	2	3	9	2	6	7	2	9
1-0:	1	1	2	3	9	2	6	7	2	9
7-2:	1	1	2	3	9	2	6	2	2	9
5-3:	1	1	2	2	9	2	6	2	2	9
4-0:	1	9	2	2	9	2	6	2	2	9
3-0:	1	9	2	2	9	2	6	2	2	2
4-6:	1	9	2	2	9	2	2	2	2	2

Table 1.2: Exercise Quick-Union