BLG336E - Analysis of Algorithms II
2017-2018 Spring, Project 3 Report
Submission Deadline: 08.05.2018, 23:59

Kadir Emre Oto (150140032)

1) Present your implementation **in detail**. Be as clear as possible, do not let any subtle points make you lose points. It will be great to hear **what** algorithms you used, **why** you used, what **subproblems** emerged and you solved, or were there any chances of **reduction**?

In this assignment, we are expected to split a graph into exactly two components which number of nodes are at least 2 by removal nodes and the purpose is to minimize the removal nodes.

At first sight, we can implement the **brute-force** algorithm that tries all removal node combinations and check the graph is divided into two components. But the complexity of brute-force algorithm is $O(n \cdot 2^n)$. To reduce the complexity, binary search or bisection methods may applied, but the algorithm would be still too slow.

On the other hand, we can adapt the problem to min-cut (max-flow) problem. Min-cut problem tries to find at least how many edges should be discarded to cut the connection from predefined source and sink, and there are various efficient algorithms to solve this problem such as Ford-Fulkerson ($O(E \cdot max(f))$) and Edmonds–Karp ($O(VE^2)$).The complexities look much more efficient than brute-force. But the algorithms run directed graphs and designed for removal edges. So we should change the algorithm a little, we can use the transformation in Figure 1. In this way, we have a weighted & directed graph and all nodes can be used just one time because the weight of the edge between a node and its copy node is 1 and can be used one time.

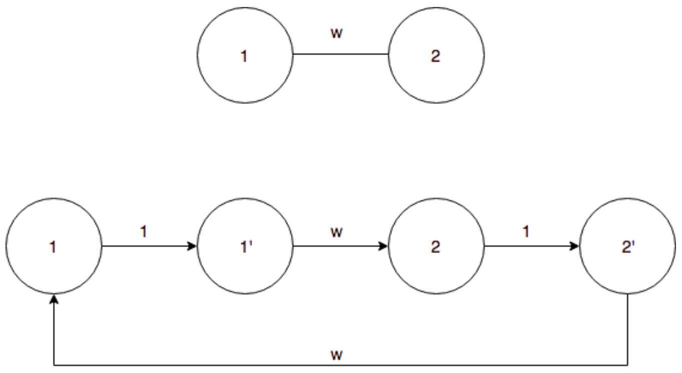

Figure 1. Transformation of the graph

After that observation, we can just use a flow algorithm over the transformed graph; however, we have another problem here, because we do not have a source and sink yet. First we should determine a source and sink, but we do not know which nodes are better. Hence we can try all possible node pairs as source and sink, but it adds $O(N^2)$ complexity to overall complexity. We do not want this additional complexity, so to improve the solution, I used a better approach to choose source and pair: for all nodes as source, we choose the node which is farthest node to source as sink. Thereby, additional $O(n)$ complexity will be added, and the overall complexity will be $O(E \cdot max(F) \cdot N)$. Because all edges in the graph has unit weight and there is no duplicate edge in the graph, the complexity is reduced to $O(NE^2)$.

2) Run Ford-Fulkerson on the given graph G, write out its residual graph R especially with back edges. Do you have any comment about the existence of back edges in R, **what** do they represent and **why** are they used?

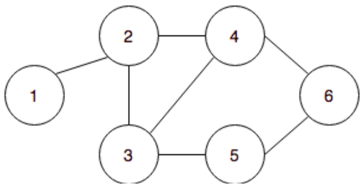

Figure 2. Given Graph

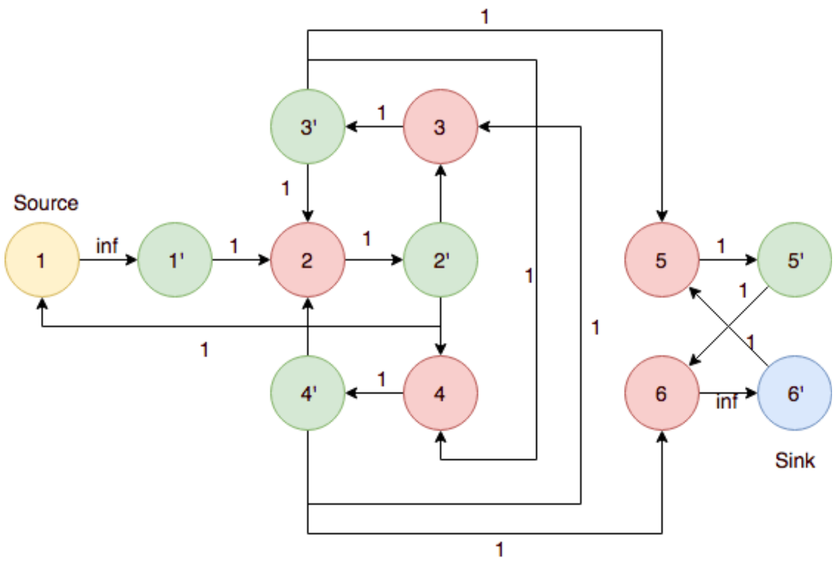

Figure 3. Transformed Version of Given Graph

The algorithm first finds this path: (1 -> 1’ -> 2 -> 2’ -> 4 -> 4’ -> 6 -> 6’) and the maximum flow of this path is 1. And algorithm cannot find another path, and returns 1. Blue paths are residual paths in figure 4.

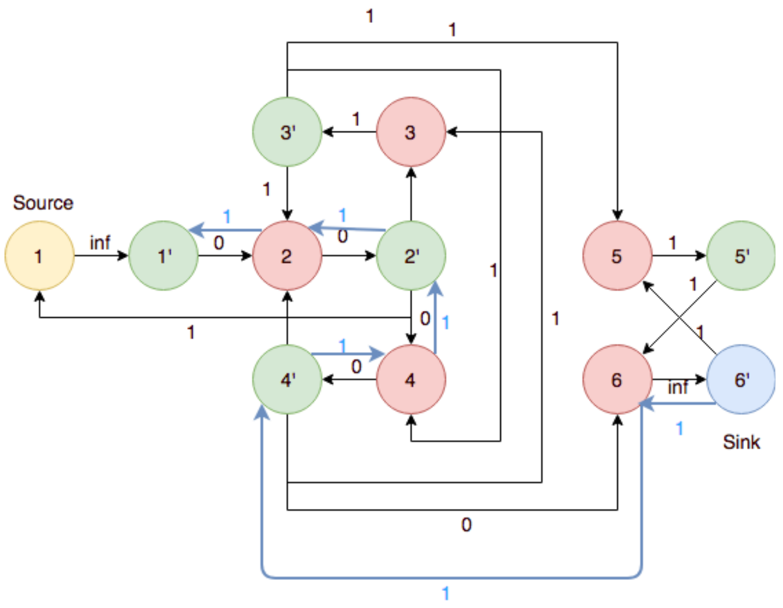

Figure 4. The second and final step of the residual graph