**Network flow for bipartite matching**

Implement the Ford-Fulkerson algorithm using the shortest augmenting paths method for bipartite matching. See, in particular, slides 59 and 70 of the [first Network Flow slides (Links to an external site.)](http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI.pdf), while noting that the problem is specifically for bipartite matching (each edge and reverse edge will have capacity 0 or 1). Output the pairs matched in some maximal matching of the input (there could be more than one, but you just need to output one). The input file should be hardcoded to have the name “program3data.txt”. The first line will hold the number of nodes n, which will be even. The next n/2 lines will store the names of items to be matched in the “left” set. Then, n/2 further lines will store the names of items to be matching in the “right” set. (You don’t actually need to distinguish between them.) Next, the number of edges will be stored and following that will be the edges (one per line) indexing the nodes using 1-based indexing. Here is an example:

10

Anibel

Belinda

Clark

Delphina

Ernie

Dozer

Sassy

Betty

Tiger

Pippy

10

1 6

1 8

2 6

2 7

2 8

2 9

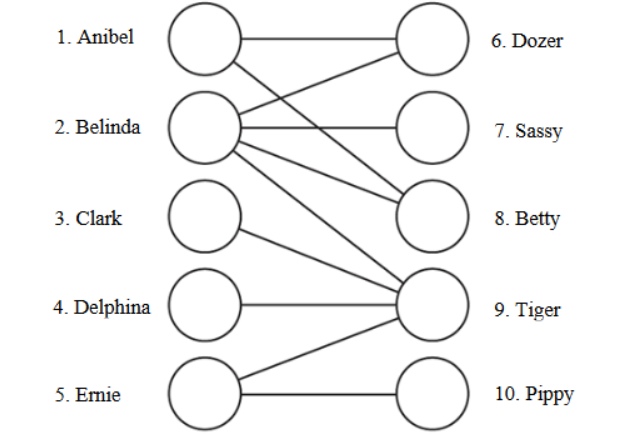
3 9

4 9

5 9

5 10

Here is [program3data.txtPreview the document](https://canvas.uw.edu/courses/1408299/files/67839193/download?wrap=1). This represents the following graph:



Output the matches in the order that the “left” items appear in the file and the total number of matches. One correct output would be (there are others):

Anibel / Betty

Belinda / Dozer

Clark / Tiger

Ernie / Pippy

4 total matches

Here is a summary of the algorithm:

Read data from file into Graph G.

Create residual graph Gf by adding source and sink nodes and edges from source and to sink.

Initialize flow f to zero along each edge.

While not done:

Construct level graph LG from Gf using breadth-first search (delete back and cross edges).

If no path exists from source to sink (i.e., sink not found during BFS), output matching, done.

Initialize location to source node, path to empty.

While not stuck at source:

If location is sink:

Augment flow with path.

Update Gf.

Delete edges in path from LG.

Set location to source.

Clear path.

Else:

If stuck, retreat:

Delete current node and incoming edges from LG.

Delete last edge from path.

Else:

Advance along some edge in LG that leaves current location.

Update current path.

More details:

1. You may use an adjacency matrix or adjacency list representation at your discretion.
2. You may use either Java or C++. You may use Java libraries and C++ STL for data structures. Try not to use anything compiler or version dependent.
3. You may work in pairs, but you are not required to. Pairs must both be present when all work is done.
4. Turn in your code on Canvas. Each pair should turn in only one set of code, but mark both members’ names clearly in the code. The partner who did not turn in code should submit a comment indicating who turned in the code they worked on.
5. Your code should be able to handle at least 100 nodes. I have posted the above data set. I will test with a different data set.
6. Your code should run in O(n5/2) time. Based on our analysis in lecture, this will be the case if your phases run in O(n2) time.
7. See the posted grading rubric for details on how I grade.