One goal of this assignment is to teach you how to write code that can elegantly process a very large number of inputs. As the inputs get very large and numerous, it is not practical to type them in each time you run the program, or to write a program that can only compute the answer for one graph.

I purposely gave you a lot of inputs to make you think about this. Real research projects can have billions of inputs!

Also: I often run one program and use its output as input to another program. It's harder to parse the file if you include text as well as integers.

Example:

Run a program to generate some class of graphs: planar graphs (plantri), arbitrary small graphs (geng), fullerenes (fullgen), trees…

Take each graph and test it: possibly to compute some parameter to see if a conjecture is true or not.

Command file for running on small fullerenes (run_com):

```
time  a.out  1 < c020 > o020
time  a.out  1 < c024 > o024
time  a.out  1 < c026 > o026
time  a.out  1 < c028 > o028
time  a.out  1 < c030 > o030
time  a.out  1 < c032 > o032
time  a.out  1 < c034 > o034
time  a.out  1 < c036 > o036
time  a.out  1 < c038 > o038
time  a.out  1 < c040 > o040
time  a.out  1 < c042 > o042
time  a.out  1 < c044 > o044
time  a.out  1 < c046 > o046
time  a.out  1 < c048 > o048
time  a.out  1 < c050 > o050
time  a.out  1 < c052 > o052
time  a.out  1 < c054 > o054
time  a.out  1 < c056 > o056
time  a.out  1 < c058 > o058
time  a.out  1 < c060 > o060
```

To run this:
source run_com

Program computed a minimum dominating set of each graph in the input file.

Timing data for all small fullerenes:

n	#	time
20	1	0
24	1	0
26	1	0.004
28	2	0
30	3	0.004
32	6	0.02
34	6	0.016
36	15	0.076
38	17	0.092
40	40	0.672

n	lb	#	time
42	11	45	0.504
44	11	89	2.6
46	12	116	2.728
48	12	299	13.66
50	13	271	13.592
52	13	437	58.023
54	14	580	58.44
56	14	924	295.042
58	15	1205	248.143
60	15	1812	1109.341

= 4.9 minutes

= 18.5 minutes

For 40: 0.672u 0.000s 0:00.67 100.0% 0+0k 0+24io 0pf+0w

n	LB	#LB	#LB+1	#LB+2	Adj. list
40	10	1	21	18	0.156
42	11	1	44	0	0.104
44	11	0	55	34	0.532
46	12	6	110	0	0.544
48	12	1	109	89	2.62
50	13	6	265	0	2.54
52	13	0	270	167	10.58
54	14	19	561	0	10.38
56	14	1	470	453	51.46
58	15	23	1182	0	42.11
60	15	0	1014	798	183.7

The input files I gave you have rotation systems that may not represent planar embeddings (e.g. iprism).

I added a link to a file iplanar_prism.txt that should contain rotation systems for planar embeddings.

Some new rotation systems added to bottom of assignment description:

Here are some input files of graphs with maximum degree 4 you could use for testing:

a1_input.zip

You can download this file by right clicking on it and using the "Save as" option and then to unzip it use:

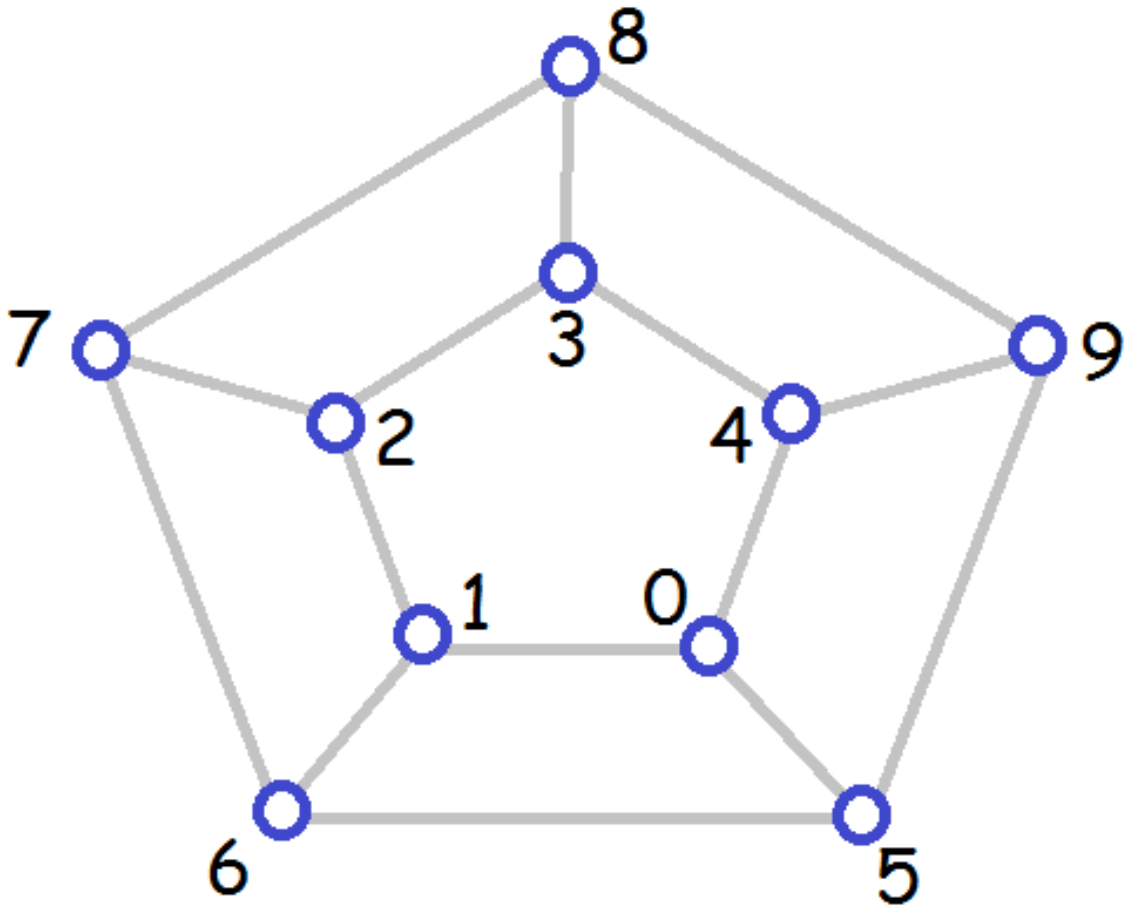
unzip a1_input

Inside the directory a1_input, you will find a file called README.txt that explains the contents of the input files.

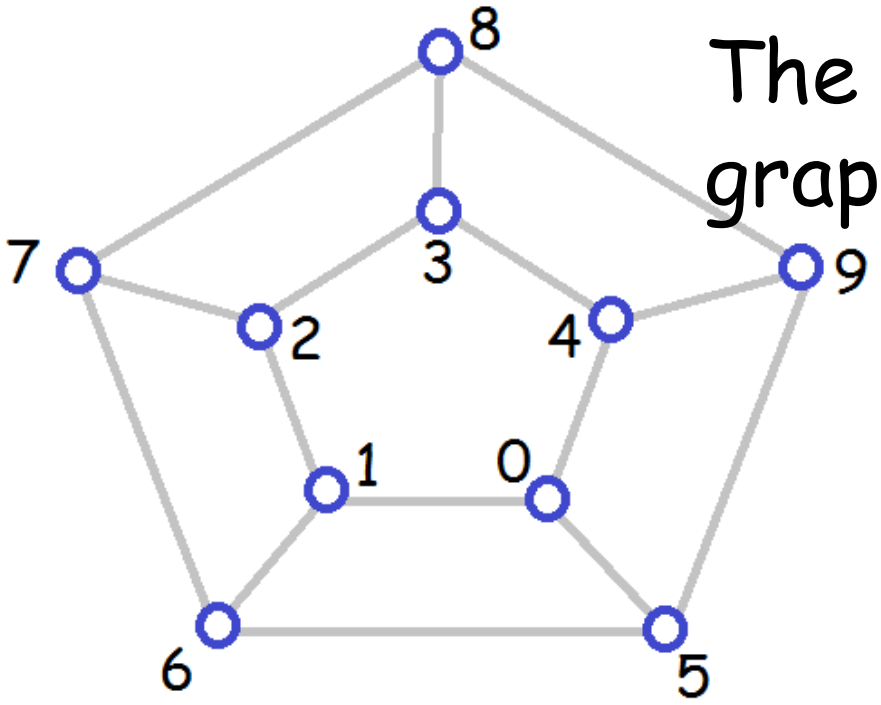
The graphs in these files do not necessarily have rotation systems that represent planar embeddings. The embeddings in iprism are NOT planar. If you would like planar embeddings of the prisms for testing, use this input file (right click on the name and use "Save as" to save on your computer):
iplanar_prism.txt

NEW

If you have a file like this
(the first graph in iplanar_prism.txt):

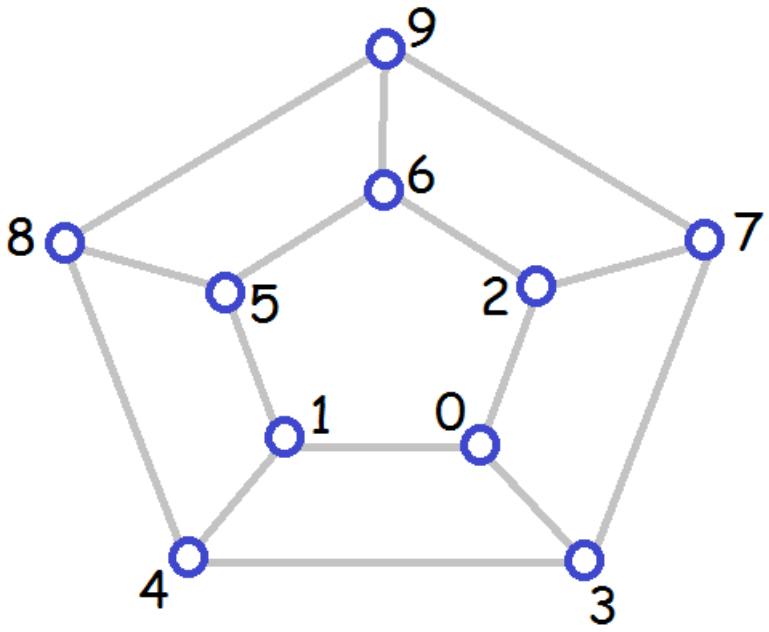



```
10
3 1 4 5
3 2 0 6
3 3 1 7
3 4 2 8
3 0 3 9
3 0 9 6
3 1 5 7
3 2 6 8
3 3 7 9
3 4 8 5
```

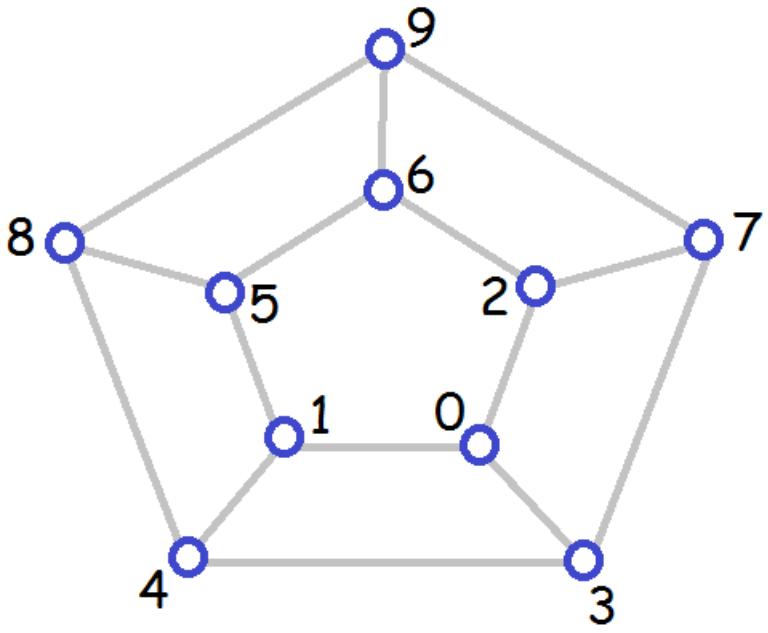


The input graph.

The renumbered rotation system:



The renumbered rotation system:



The output:

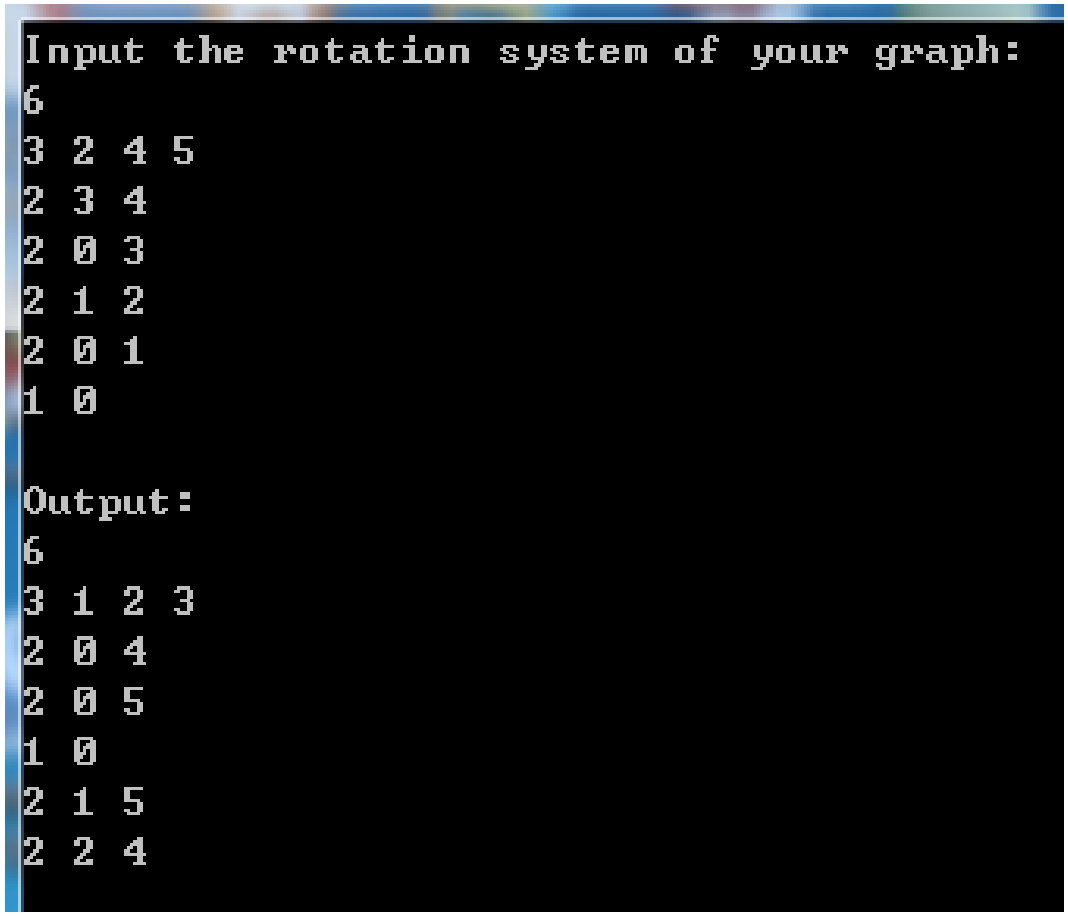
10
3 1 2 3
3 0 4 5
3 0 6 7
3 0 7 4
3 1 3 8
3 1 8 6
3 2 5 9
3 2 9 3
3 4 9 5
3 6 8 7

Input file

```
10
3 1 4 5
3 2 0 6
3 3 1 7
3 4 2 8
3 0 3 9
3 0 9 6
3 1 5 7
3 2 6 8
3 3 7 9
3 4 8 5
12
3 1 5 6
3 2 0 7
3 3 1 8
3 4 2 9
3 5 3 10
3 0 4 11
3 0 11 7
3 1 6 8
3 2 7 9
3 3 8 10
3 4 9 11
3 5 10 6
```

Output file

```
10
3 1 2 3
3 0 4 5
3 0 6 7
3 0 7 4
3 1 3 8
3 1 8 6
3 2 5 9
3 2 9 3
3 4 9 5
3 6 8 7
12
3 1 2 3
3 0 4 5
3 0 6 7
3 0 7 4
3 1 3 8
3 1 8 9
3 2 9 10
3 2 10 3
3 4 11 5
3 5 11 6
3 6 11 7
3 8 10 9
```



If you add text like this to prompt for the graph and indicate the output, it will mess up your output file (I won't be able to use it as an input file).

Also, if I have 100000 graphs in the file you won't want to see these text pieces printed 100000 times as the program runs.

You might want more readable debugging output:

Graph 1:

Input graph:

0(3): 1 4 5
1(3): 2 0 6
2(3): 3 1 7
3(3): 4 2 8
4(3): 0 3 9
5(3): 0 9 6
6(3): 1 5 7
7(3): 2 6 8
8(3): 3 7 9
9(3): 4 8 5

After clockwise BFS:

0(3): 1 2 3
1(3): 0 4 5
2(3): 0 6 7
3(3): 0 7 4
4(3): 1 3 8
5(3): 1 8 6
6(3): 2 5 9
7(3): 2 9 3
8(3): 4 9 5
9(3): 6 8 7

Graph 2:

Input graph:

…