# Introduction:

The official statement of the problem is given here: <https://projecteuler.net/problem=781>

I interpreted the problem as follows:

is the number of graphs with certain properties, and we will discuss what these properties are.

Each graph that qualifies has some blue (directed) arcs and red (undirected) edges. We call the set of the , and each has the following characteristics:

1. There is exactly one node with exactly one outgoing and exactly one node with exactly one incoming .
2. There are exactly additional nodes, each of which has:
   1. Exactly one incoming and
   2. Exactly one incoming arc
      1. For each of these nodes, the two arcs are different.
3. There are no other nodes or arcs.

I interpreted this as saying that each consists of:

1. Exactly one directed path with at least two arcs
2. Zero or more directed cycles, each of which has at least two arcs.
3. There are s.

Each has exactly nodes, and s. For the most part, we will ignore the two nodes that are incident to only one arc and so we are only interested in nodes. These are the internal nodes in the path and the nodes in the cycles.

I interpreted the specification of the red edges to be:

1. There are s.
2. Each is incident to two nodes in the .
3. No is incident to the start or end of the path.
4. The s together with the s result in a connected graph.

A is the addition of red edges to the , that results in a connected graph. Our goal is to find the total number of for the different s.

# The , and the s , , …, .

In each , we let be the number of s in the path. We also let be the number of 2-cycles, be the number of 3-cycles, … and be the number of -cycles. Because there are exactly arcs, and at least two of them are in the path, there are no cycles of length or higher.

From this, we see that the number of s in the path plus the number of s in the cycles must equal :

Note that the set of numbers , …, , determines a ; we call this vector of numbers a . The of one are all different graphs from the of a different .

Hence, we list the and for each one, compute the number of . We add these all up to get .

Our notation for s is , but we suppress , , …, ­ if they are all zero.

# Example 1:

This is the example given in the website. There are three s:

1. [5] (a path of length five and no cycles).
2. [3 1] (a path of length three and one 2-cycle).
3. [2 0,1] (a path of length two, no 2-cycles, and one 3-cycle).

For [5], we must use s to match the nodes , , , and of the path; there are no cycles so the graph is automatically connected. can be matched to any of three choices, and the matching can be completed only one way for each of these. Hence, the [5] has exactly three .

For the [3 1], we must match the nodes and of the path to the 2-cycle’s nodes; if we were to match to we could not use the s to connect the path to the 2-cycle. Hence, there is only one for [3 1].

Similar logic shows that there is only one for [2 0,1].

Our summary includes the number of for each , and the running total. Hence our summary for is:

1. [5], ThisCount[3], RunningTotal[3]
2. [3 1], ThisCount[1], RunningTotal[4]
3. [2 0,1], ThisCount[1], RunningTotal[5]

# Example 2

For , there are seven s, and our summary is:

1. [7], ThisCount[15], RunningTotal[15]
2. [5 1], ThisCount[6], RunningTotal[21]
3. [4 0,1], ThisCount[5], RunningTotal[26]
4. [3 2], ThisCount[1], RunningTotal[27]
5. [3 0,0,1], ThisCount[3], RunningTotal[30]
6. [2 1,1], ThisCount[2], RunningTotal[32] (Corrected from previous versions)
7. [2 0,0,0,1], ThisCount[3], RunningTotal[35]

We will go through some of the logic for computing the number of s. No matter which we are considering, we label the nodes of the path that need to be matched to other nodes, , , … .

The [7] has only a path of seven s and the vertices of the path that must be matched are , …, . For any of the five ways of matching to another node, there are three ways of matching the smallest unmatched node to some other unmatched node. The remaining two unmatched nodes must be matched to each other and so there are for the [7].

The [3 0,0,1]’s path has and and these must be matched with two of the nodes of the directed 4-cycle. If were matched to , it would be impossible to connect the to the path. Hence, must be matched to a node in the and it is irrelevant which one is chosen. Then there are three ways of matching to the other nodes in the and after each is done, the remaining two nodes of the must be matched to each other. Hence, there are only s for [3 0,0,1].

# Example 3

For , there are nine s, and the summary is:

1. [9], ThisCount[105], RunningTotal[105]
2. [7 1], ThisCount[45], RunningTotal[150]
3. [6 0,1], ThisCount[35], RunningTotal[185]
4. [5 2], ThisCount[9], RunningTotal[194]
5. [5 0,0,1], ThisCount[24], RunningTotal[218]
6. [4 1,1], ThisCount[15], RunningTotal[233] (Corrected from previous versions)
7. [4 0,0,0,1], ThisCount[21], RunningTotal[254]
8. [3 3], ThisCount[1], RunningTotal[255]
9. [3 0,2], ThisCount[5], RunningTotal[260]
10. [3 1,0,1], ThisCount[9], RunningTotal[269] (Corrected from previous versions)
11. [3 0,0,0,0,1], ThisCount[15], RunningTotal[284]
12. [2 2,1], ThisCount[3], RunningTotal[287] (Corrected from previous versions)
13. [2 0,1,1], ThisCount[8], RunningTotal[295] (Corrected from previous versions)
14. [2 1,0,0,1], ThisCount[9], RunningTotal[304] (Corrected from previous versions)
15. [2 0,0,0,0,0,1], ThisCount[15], RunningTotal[319]

We will discuss [3 3] in detail. The path has three s and hence there are two nodes and of the path that must be matched. In addition, there are three s, each with two arcs. To connect the path to the cycles, cannot be matched to and so it must be matched to one of the cycles’ nodes. Let be the cycle that is matched to. We cannot match to the other node of or else the path and cannot be connected to the remaining two cycles. Hence must be matched to a node in a different cycle, which we will call , and we will let be the remaining cycle. We cannot match the unmatched nodes of and or else we cannot connect . Hence, the remaining nodes of and must be matched to the two nodes of and the number of is only one!

# My Algorithm (Part 1):

Given , I partition the arcs into a path and cycles. For each way of doing this, we compute the number of s. In this section, we discuss the algorithm for generating the different partitions. The results of this algorithm for appear at the beginning of the lines in the Examples above.

We initialize the algorithm with . Given a , our algorithm returns the next . It does this by shuffling the nodes in the cycles unless it cannot, at which point, it decreases the number of s in the path and starts over.

Within a given path length , we have s to allocate to the cycles. If the number of s in the 2-cycles is sufficient to create a 3-cycle, we will increase the number of 3-cycles by 1. Notice that two 2-cycles is *not* sufficient for creating a 3-cycle, for that would leave a that we cannot put anywhere.

We let be the number of s from the 2-cycles. If or , we increase the number of 3-cycles by one. We subtract three from and use these to create 2-cycles and up to one 3-cycle.

Otherwise, we add to the number of s from the 3-cycles and check to see if we can increase the number of 4-cycles by 1 by using the s. If we can, we will and once again, we will create 2-cycles and possibly one 3-cycle.

We continue this way, stopping when is big enough to create a cycle that is “one bigger” than the last contribution to . If we never get to that point, then we decrease the number in the path and start with 2-cycles and possibly one 3-cycle.

A few observations are in order:

1. If we are decreasing the path length from , we must decrease it by two. Otherwise, we decrease it by one.
2. For a given path length , let be the number of s in the cycles: . We have:
   1. We will always start with or depending on whether is even or odd.
   2. We will always end with

# My Algorithm (Part 2):

For each , we compute the number of s. We use a recursive algorithm which is basically:

1. Choose some vertex that is incident to no .
2. For each “legal” vertex to connect with with a , connect it and count the number of s. Add these up.

We must choose a vertex in such a way that guarantees the graph will stay connected. Then we must define the s. We also do not wish to use every “legal” vertex to connect to.

To do these two tasks, we define an . The s are the path and the cycles. A is if it is either the path or there is a from an to . It is if every vertex in has a incident to it.

We choose for , the lowest numbered vertex in an that is not . We keep track of the number of s and call this . These are the vertices that are in some but are not incident to a .

We state without proof that if we connect to some other , and get the number of s, that will be the same as the number of s were we to connect to any other . Hence, we connect to an arbitrary , get the number of s, and multiply that by the number of s that we can connect to.

We still must consider the s that arise from connecting to some that is not . Here, we connect to only one vertex of one 2-cycle, one vertex of one 3-cycle, etc.. For each of these, we compute the number of s and there is no multiplier.

Care must be taken to ensure that the graph will be connected. We do this by refusing to connect to a vertex in an if , unless this is the last to create.

# Our Problem:

This algorithm works for , but certainly not , much less . In fact, we cannot even generate the s in a reasonable time. Perhaps, we can cut down on the s that we need to generate and compute the number of s.