Introduction:

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

I interpreted the problem as follows:

$F(n)$ 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 $Blue\ Arcs$ the $Blue\ Graph$, and each $Blue\ Graph$ has the following characteristics:

1. There is exactly one node with exactly one outgoing $Blue\ Arc$ and exactly one node with exactly one incoming $Blue\ Arc$.
2. There are exactly n additional nodes, each of which has:
    a. Exactly one incoming $Blue\ Arc$ and
    b. Exactly one incoming arc $Blue\ Arc$
        i. 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 $Blue\ Graph$ 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 $n + 1$ $Blue\ Arc$s.

Each $Blue\ Graph$ has exactly $n + 2$ nodes, and $n + 1$ $Blue\ Arc$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 n 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 $n\ /\ 2$ $Red\ Edge$s.
2. Each $Red\ Edge$ is incident to two nodes in the $Blue\ Graph$.
3. No $Red\ Edge$ is incident to the start or end of the path.
4. The $Blue\ Arc$s together with the $Red\ Edge$s result in a connected graph.

A $Red\ Completion\ of\ a\ Blue\ Graph$ is the addition of red edges to the $Blue\ Graph$, that results in a connected graph. Our goal is to find the total number of $Red\ Completions$ for the different $Blue\ Graph$s.

The $Blue\ Graphs$, and the $Blue\ Vector$s $[m_0, m_2, \ldots, m_{n-1}]$.

In each $Blue\ Graph$, we let m_0 be the number of $Blue\ Arc$s in the path. We also let m_2 be the number of 2-cycles, m_3 be the number of 3-cycles, … and m_{n-1} be the number of $(n-1)$-cycles. Because there are exactly $n + 1$ arcs, and at least two of them are in the path, there are no cycles of length n or higher.

From this, we see that the number of $Blue\ Arc$s in the path plus the number of $Blue\ Arc$s in the cycles must equal $n + 1$:

$$m_0 + \sum_{k=2}^{n-1} m_k \times k = n + 1.$$

Note that the set of numbers $[m_0\ m_2, \ldots, m_{n-1}]$, determines a $Blue\ Graph$; we call this vector of numbers a $Blue\ Vector$. The $Red\ Completions$ of one $Blue\ Vector$ are all different graphs from the $Red\ Completions$ of a different $Blue\ Vector$.

Hence, we list the $Blue\ Vectors$ and for each one, compute the number of $Red\ Completions$. We add these all up to get $F(n)$.

Our notation for $Blue\ Vector$s is $[m_0\ m_2, \ldots, m_{n-1}]$, but we suppress $m_k, m_{k+1}, \ldots, m_{n-1}$ if they are all zero.

Example 1: $n = 4$

This is the example given in the website. There are three $Blue\ Vectors$:

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 $Red\ Edges$ to match the nodes p_1, p_2, p_3, and p_4 of the path; there are no cycles so the graph is automatically connected. p_1 can be matched to any of three choices, and the matching can be completed only one way for each of these. Hence, the $Blue\ Vector$ [5] has exactly three $Red\ Completions$.

For the $Blue\ Vector$ [3 1], we must match the nodes p_1 and p_2 of the path to the 2-cycle's nodes; if we were to match p_1 to p_2 we could not use the $Red\ Edges$ to connect the path to the 2-cycle. Hence, there is only one $Red\ Completion$ for [3 1].

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

Our summary includes the number of $Red\ Completions$ for each $Blue\ Vector$, and the running total. Hence our summary for $n = 4$ is:

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

Example 2 $n = 6$

For $n = 6$, there are seven $Blue\ Arcs$, 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 $Red\ Completions$. No matter which $Blue\ Vector$ we are considering, we label the nodes of the path that need to be matched to other nodes, $p_1, p_2, \ldots$.

The $Blue\ Vector$ [7] has only a path of seven $Blue\ Arc$s and the vertices of the path that must be matched are $p_1, \ldots, p_6$. For any of the five ways of matching p_1 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 $5 \times 3 = 15\ Red\ Completions$ for the $Blue\ Vector$ [7].

The $Blue\ Vector$ [3 0,0,1]'s path has p_1 and p_2 and these must be matched with two of the nodes of the directed 4-cycle. If p_1 were matched to p_2, it would be impossible to connect the $Blue\ Cycle$ to the path. Hence, p_1 must be matched to a node in the $Blue\ Cycle$ and it is irrelevant which one is chosen. Then there are three ways of matching p_2 to the other nodes in the $Blue\ Cycle$ and after each is done, the remaining two nodes of the $Blue\ Cycle$ must be matched to each other. Hence, there are only $1 \times 3 \times 1 = 3\ Red\ Completion$s for [3 0,0,1].

Example 3 $n = 8$

For $n = 8$, there are nine $Blue\ Arc$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 $Blue\ Arc$s and hence there are two nodes p_1 and p_2 of the path that must be matched. In addition, there are three $Blue\ Cycle$s, each with two arcs. To connect the path to the cycles, p_1 cannot be matched to p_2 and so it must be matched to one of the cycles' nodes. Let C_1 be the cycle that p_1 is matched to. We cannot match p_2 to the other node of C_1 or else the path and C_1 cannot be connected to the remaining two cycles. Hence p_2 must be matched to a node in a different cycle, which we will call C_2, and we will let C_3 be the remaining cycle. We cannot match the unmatched nodes of C_1 and C_2 or else we cannot connect C_3. Hence, the remaining nodes of C_1 and C_2 must be matched to the two nodes of C_3 and the number of $Red\ Completions$ is only one!

My Algorithm (Part 1):

Given n, I partition the $n+1$ arcs into a path and cycles. For each way of doing this, we compute the number of $Red\ Completion$s. In this section, we discuss the algorithm for generating the different partitions. The results of this algorithm for $n = 4, 6, and\ 8$ appear at the beginning of the lines in the Examples above.

We initialize the algorithm with $[n+1]$. Given a $Blue\ Vector$, our algorithm returns the next $Blue\ Vector$. It does this by shuffling the nodes in the cycles unless it cannot, at which point, it decreases the number of $Blue\ Arc$s in the path and starts over.

Within a given path length m_0, we have $n+1-m_0$ $Blue\ Arc$s to allocate to the cycles. If the number of $Blue\ Arc$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 $Blue\ Arc$ that we cannot put anywhere.

We let $nAvailable$ be the number of $Blue\ Arc$s from the 2-cycles. If $nAvailable = 3$ or $nAvailable \geq 5$, we increase the number of 3-cycles by one. We subtract three from $nAvailable$ and use these to create 2-cycles and up to one 3-cycle.

Otherwise, we add to $nAvailable$ the number of $Blue\ Arc$s from the 3-cycles and check to see if we can increase the number of 4-cycles by 1 by using the $nAvailable\ Blue\ Arc$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 $nAvailable$ is big enough to create a cycle that is "one bigger" than the last contribution to $nAvailable$. 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 $[n+1]$, we must decrease it by two. Otherwise, we decrease it by one.
2. For a given path length p, let q be the number of $Blue\ Arc$s in the cycles: $q = n+1-p$. We have:
    a. We will always start with $[p\ q\ /\ 2]$ or $[p\ (q-3)\ /\ 2,1]$ depending on whether q is even or odd.
    b. We will always end with $[p\ 0,0,\ldots,q]$

My Algorithm (Part 2):

For each $Blue\ Vector$, we compute the number of $Red\ Completion$s. We use a recursive algorithm which is basically:

1. Choose some vertex v_0 that is incident to no $Red\ Edge$.
2. For each "legal" vertex v_k to connect with v_0 with a $Red\ Edge$, connect it and count the number of $Red\ Completion$s. Add these up.

We must choose a vertex v_0 in such a way that guarantees the graph will stay connected. Then we must define the v_ks. We also do not wish to use every "legal" vertex to connect v_0 to.

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

We choose for v_0, the lowest numbered vertex in an $Opened\ Component$ that is not $Completed$. We keep track of the number of $Opened\ Vertice$s and call this $nInOpened$. These are the vertices that are in some $Opened\ Component$ but are not incident to a $Red\ Edge$.

We state without proof that if we connect v_0 to some other $Opened\ Vertex$, and get the number of $Red\ Completion$s, that will be the same as the number of $Red\ Completion$s were we to connect v_0 to any other $Opened\ Vertex$. Hence, we connect v_0 to an arbitrary $Opened\ Vertex$, get the number of $Red\ Completion$s, and multiply that by the number of $Opened\ Vertice$s that we can connect v_0 to.

We still must consider the $Red\ Completion$s that arise from connecting v_0 to some $Component$ that is not $Opened$. 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 $Red\ Completion$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 v_0 to a vertex v_k in an $Opened\ Component$ if $nOpened = 2$, unless this is the last $Red\ Edge$ to create.

Our Problem:

This algorithm works for $n = 24$, but certainly not $n = 50$, much less $n = 50000$. In fact, we cannot even generate the $Blue\ Vector$s in a reasonable time. Perhaps, we can cut down on the $Blue\ Vector$s that we need to generate and compute the number of $Red\ Completion$s.