Task: SKO
Matchings in a Tree

UFAM Workshop, contest #2. Source file `sko.*` Available memory: 128 MB.

Byteman loves matchings, especially in graphs. A matching is a subset of edges in an undirected graph, such that every vertex has at most one incident edge that belongs to the matching. Byteman knows that finding a maximum-cardinality matching (i.e. containing the biggest possible number of edges) is a polynomial-time problem, although for non-bipartite graphs algorithms are quite complicated. Recently he has learned that counting matchings is a hard problem and we do not know a polynomial-time algorithm for it.

Everything gets simpler, however, in case of graphs that are trees. For them there is a linear-time algorithm for finding a maximum-cardinality matching and finding the number of maximum-cardinality matchings. Byteman decided to give such a problem in his programming contest. However, he got stuck preparing the test cases for the problem, and he needs your help.

For a given number m write a program that generates a small tree that has exactly m different maximum-cardinality matchings. Two matchings are different if there is an edge that is contained in exactly one of these matchings.

Input

In the only line of the input there is one integer m ($1 \leq m \leq 1\,000\,000$) specifying the number of maximum-cardinality matchings that tree must have.

Output

In the first line of the output you must print one positive integer n, specifying the number of vertices in the generated tree. The tree cannot have more than 1000 vertices (i.e. $1 \leq n \leq 1000$). The vertices are numbered from 1 to n. In the next $n-1$ lines you must print the descriptions of edges of the tree; the i-th of these lines must contain two positive integers, separated with a single space, specifying the numbers of vertices connected with the i-th edge.

If there is more than one correct solution, you can print any of them. It is possible to generate at least one tree satisfying the requirements, for all values of m.

Example

For the input data:

```
3
```

the correct result is:

```
5
1 2
2 3
3 4
4 5
```

Explanation of the example: A path of four edges has three matchings of cardinality 2.