PROBLEM OF THE MONTH

POM

2021 - 2022

November, 2021

MATHEMATICS

10 points:

Find the number of positive integers n larger than 2021 that divide $1^n + 2^n + \ldots + (n-1)^n$. Show/prove why.

Answer:

Infinitely many

Solution:

In fact, all odd numbers larger than 2021 satisfy this condition. Let n be an odd integer larger than 2021. We note that the sum has $n - 1$ summands, which make $(n-1)/2$ pairs of terms of the following form: $k^n + (n-k)^n$. Since $(-k)^n = -k^n$ is true, we can see that n divides $k^n + (n-k)^n$. Then we obtain that each pair of terms is divisible by n, hence the entire sum is also divisible by n.

PHYSICS

The Lorentz force describes how electric and magnetic fields act on charged particles: the Lorentz force comprises two components, F_E and F_B. The force from the electric field with magnitude E on a particle with charge q is:

$$F_E = qE,$$

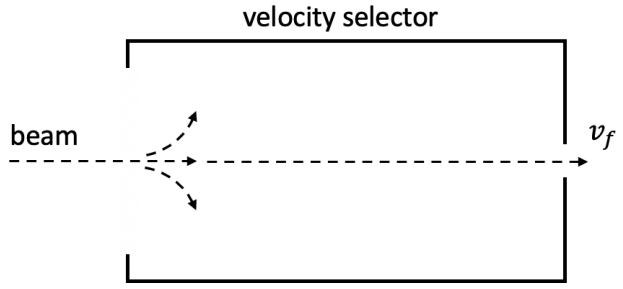
acting parallel to the direction of E. The force on the particle moving with velocity v perpendicular to a magnetic field with magnitude B is:

$$F_B = qvB,$$

acting perpendicular to both v and B (see right-hand rule).

5 points:

Consider a particle accelerator which generates a beam of atoms of different species and a range of velocities. Assume all atoms are singly-ionized, i.e. have lost one electron. A *velocity selector* apparatus is shown in the diagram below. The device consists of a tube with a slit at the end, and uniform magnetic and electric fields are created in the tube. For an appropriately chosen field combination, only one velocity of particles (for any mass) will go through the slit. What must the magnetic and electric fields be (their magnitude and direction) to obtain a specific speed, v_f?

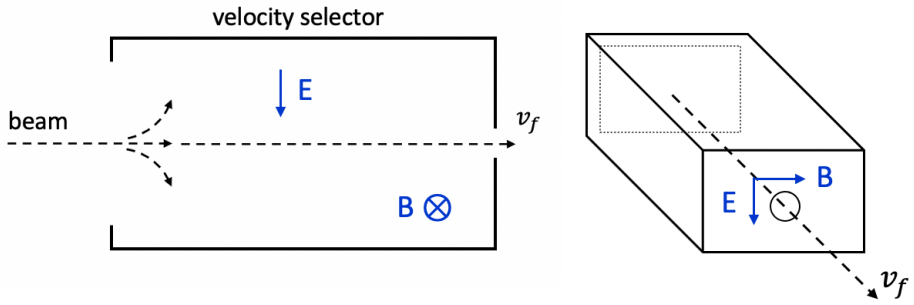


Answer:

For example: orient the electric field with magnitude E down and the magnetic field with magnitude B into the page, with respect to the original diagram.

Solution:

A straight-forward solution is to orient the electric field with magnitude E down and the magnetic field with magnitude B into the page. In diagrams, scientists often use the symbol ⊗ to indicate that something points into the page and the symbol ⊙ for pointing out of the page. The same setup is depicted below in 3D for clarity.

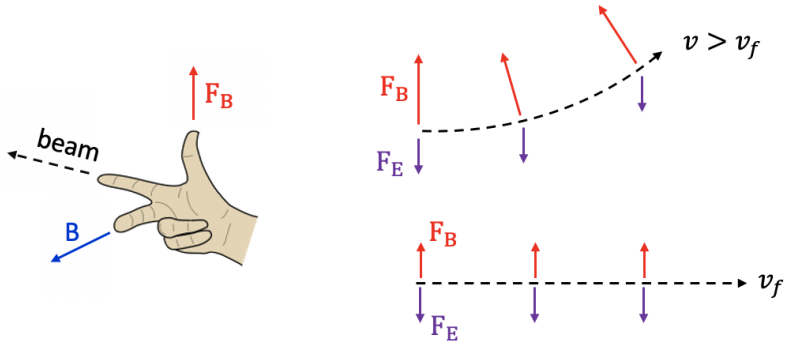


As the atoms have all lost one electron, their charge q is positive. In this configuration, the electric field produces a force pointing down. From the right-hand rule, the magnetic field will produce a force pointing up when the atoms first enter the velocity selector. If the two forces are equal on the atom, it will not be deflected and continue to travel in a straight line:

$$qE = qvB$$

Then to select the speed v_f to pass unperturbed through the slit, the ratio of the magnitude of the electric field over the magnitude of the magnetic field must be equal to that speed:

$$E/B = v_f$$



10 points:

Once you have filtered the beam so that all particles are moving with the same speed, how can you use a combination of magnetic and electric fields to sort particles based on their mass? Specifically, your apparatus should translate mass m into a position coordinate x (one can then scan an electric charge meter across a range of positions x and from the electric current infer the mass composition of the beam of particles).

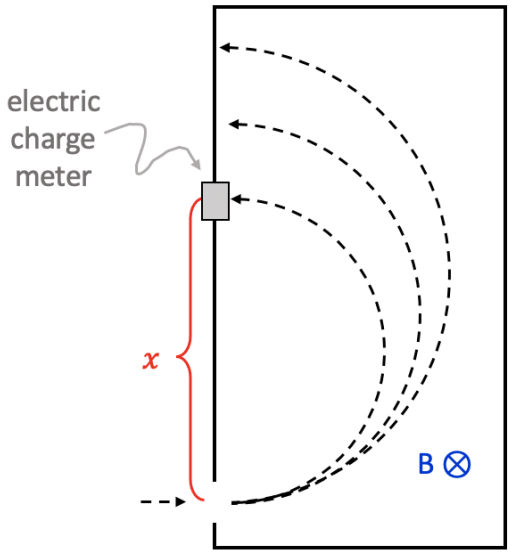
Suppose now that the beam of particles contains a small fraction of doubly-ionized atoms in addition to the majority singly-ionized atoms. If you are trying to detect singly-ionized Carbon-14 atoms, which doubly-ionized atom would give a false-positive on your detector?

Answer:

Orient the magnetic field into the page. The false-positive doubly-ionized atom will have twice the atomic mass of the singly-ionized Carbon-14 atoms: 28 atomic mass units. Silicon-28, for example.

Solution:

To sort the particles by mass, we don't need an electric field: we can get away with using only the magnetic field whose force on the particle depends directly on the velocity. We can orient the magnetic field into the page. Notice from the right-hand rule that the magnetic force will always point perpendicular to both the motion of the particle and the magnetic field. When the velocity and a constant acting force are pendicular to one another, the particle does not gain or lose energy. This would mean that the speed remains constant but the direction will change. This type of force is called a *centripetal force*, explored in the September Physics Problem of the Month. Hence we can solve for the radius *R* of the centripetal force that the particle experiences due to the magnetic field, where *m* is the mass of the particle:



$$qvB = m v^2/R$$

If we define the position *x* of our electric charge meter as *x* = *2R* (the diameter of the trajectory), then we can directly relate *x* to the mass *m*:

$$x = 2 m v/ q B$$

Notice that if we hadn't first selected particles of a specific speed then the position would depend on both the mass and the speed and it would not be a very effective mass sorter.

If our initial beam also contains some doubly-ionized atoms, then sometimes we will attribute the wrong mass to the particle. These atoms will have twice the charge $(2\ q)$ of the singly-ionized atoms . What will be the relative mass and charge of the doubly-ionized atom that goes to the same position as the singly-ionized atom? Setting the positions of the two types of atoms equal:

$$2\,m_1\,v/\,q\,B\,=\,2\,m_2\,v/\,(2\,q)\,B$$
$$2\,m_1\,=m_2$$

So the false-positive doubly-ionized atom will have twice the atomic mass of the singly-ionized Carbon-14 atoms: 28 atomic mass units. Silicon-28, for example.

CHEMISTRY

5 points:

Diamond and graphite are two allotropic forms of carbon, which can be converted into each other under some specific conditions. Would it be possible to develop a hypothetical enzyme or chemical catalyst for conversion of graphite to diamond at room temperature and normal pressure? Explain your answer.

Answer:

Enzymes are catalysts, which means that they just accelerate transformations, but do not change a position of equilibrium. At room temperature and low pressure, diamond is a metastable state of carbon, which means its energy is higher than the energy of graphite. Diamonds exist in nature just because their transformation to graphite is very slow (longer than hundreds million of years). That means the enzyme that accelerate that reaction would lead to an opposite transformation: conversion of diamonds to graphite.

10 points:

An article in SciNews (http://www.sci-news.com/othersciences/chemistry/liquid-gallium-carbon-dioxide-conversion-10164.html) describes the results of recent studies published in a reputable journal “Advanced materials” (Tang et al. Liquid-Metal-Enabled Mechanical-Energy-Induced CO_2 Conversion. Advanced materials 2021, 2105798). According to those studies, a catalytic transformation of CO_2 back to gaseous oxygen and carbon is possible, and the whole process requires as little as 230 kWh of energy for conversion of one ton of CO_2. Assuming that the heat of combustion of carbon is 32.8 MJ/kg, and the efficiency of fossil fuel power plants is ~ 40%, estimate if that newly discovered process can be implemented for generation of electricity with a zero carbon footprint. Verify your conclusion in terms of general conservation laws.

Answer:

That article is a result of an experimental error (unfortunately, that sometimes happens in science).

Indeed, the equation of the reaction that converts converts CO_2 back into carbon and O_2 is as follows:

$$CO_2 \rightarrow C + O_2 \qquad (1)$$

This reaction is a reverse reaction of the combustion of carbon:

$$C + O_2 \rightarrow CO_2 \qquad (2)$$

Clearly, if the amount of energy consumed during the reaction 1 is smaller than the amount of energy produced in the reaction 2, that would be a violation of the energy conservation law, so the process described in the article cannot help us to minimize our carbon footprint.

BIOLOGY

5 points:

The genomic DNA fragment shown below is known to encode a C-terminal fragment of some protein sequence (In other words, peptide synthesis ends somewhere in that fragment). The starting nucleotide is not known. Read the amino acid sequence and present it in the form of three and single letter codes. The sequence shown below is a sense strand, a.k.a. a coding strand.

- …..tttccatcggtatggcttctgccatgccctaattcttgatattcttggggtgg …. -

Answer:

We don't know the start position, which means we can split the sequence on codons in three different ways, e.g.

… ttt cct tcg …, or
… ttc cat cgg …, or
… tcc atc ggt.

And, accordingly, the peptide sequence will be Phe Pro Ser …, Phe His Trp …, or Ser Ile Gly, accordingly. That means, to read the actual peptide sequence, we need to know a correct "reading frame" (the example provided above shows three different reading frames, but only one is correct, because it corresponds to a correct start position, which we don't know).

Fortunately, we can resolve this problem, because we have a clue: the peptide sequence ends somewhere in this DNA segment, which means it contains one (out of three) stop codons. These codons (TAG, TAA, or TGA) do not encode any amino acid, and when a ribosome arrives to them, it stops and dissociates from RNA.

If we look at the sequence, there are two stop codons:

- …..tttccatcggtatggcttctgccatgccc**taa**ttct**tga**tattcttggggtgg ….

Clearly, peptide synthesis stops at the first one, so the reading frame can be easily restored:

- …..tt tcc atc ggt atg gct tct gcc atg ccc taa ttcttgatat ….
- Ser Ile Gly Met Ala Ser Ala Met Pro (stop)

(Actually, we realized that the sequence had a typo: the codon "TCT" was supposed to be "TGT", and in that case, the sequence in a single letter mode would read "S I G M A C A M P"). Due to a type, it is "SIGMASAMP", which demonstrates a danger of point mutations, which can have much more negative consequences when they occur in a human body :-)

10 points:

Below, the segments of aligned genomes of several closely related species are shown. These segments encode for the same protein, which is pretty conserved among those species. It is very sensitive to mutations, so replacement of even one amino acid usually leads to a significant loss of the protein's functionality, which has a detrimental effect on viability.

Read the sequence encoded by that DNA segment. Similar to the previous problem, it is a coding strand, and a reading frame is unknown.

```
ttgagccaatcggggagaatgagaccatctgctcctt
agagcctatcggggagaatgagacgatctgctccaa
aagaacccattggtgaaaacgaaacaatttgttctt
ggacccaataggtgaaaacgagacgatttgttcgcc
gagccaatcggagagaatgagactatctgctcgct
ggagccaatcggcgagaacgagactatatgctcgcc
cgagccaataggtgagaacgagacgatttgttcgg
gggacccaataggtgaaaacgagacgatttgttcga
gaacctattggcgaaaacgaaactatttgttcttaa
```

Answer:

In a single letter amino acid code, the sequence is:

```
E P I G E N E T I C S
```

Solution:

First, we should correctly align the sequences, because they contain mutations. One important thing that we know about these sequences is that all of them encode the same important protein, which means the amino acid sequence is most likely the same. In other words, it is composed of synonymous codons. An example of synonymous codons is glycine codons (GGT, GGC, GGA, GGG), where the first two letters are the same, but the last letter may be different. That means the sequences are organized in this way:

```
NN?NN?NN?NN?
```

Where “N” is some specific nucleotide, which is conserved among all sequences, whereas “?” may vary.

The situation may be more complicated, because some totally different codons may be synonymous too, for example, TCC and AGT encode for serine. That is less common, but we should keep it in mind.

To align the sequences, let’s try to find pairs of nucleotides that are conserved in most sequences, for example “gg” in the middle. We get this:

```
  ttgagccaatcggggagaatgagaccatctgctcctt
   agagcctatcggggagaatgagacgatctgctccaa
  aagaacccattggtgaaaacgaaacaatttgttctt
   ggacccaataggtgaaaacgagacgatttgttcgcc
    gagccaatcggagagaatgagactatctgctcgct
   ggagccaatcggcgagaacgagactatatgctcgcc
   cgagccaataggtgagaacgagacgatttgttcgg
  gggacccaataggtgaaaacgagacgatttgttcga
    Gaacctattggcgaaaacgaaactatttgttcttaa
```

Most likely, these pairs set a reading frame. That is easy to check, because if we check every third position in these sequences, we see that these positions are frequently mutated, whereas other positions are not.

```
 A gag cct atc ggg gag aat gag acg atc tgc tcc aa
aa gaa ccc att ggt gaa aac gaa aca att tgt tct t
 g gac cca ata ggt gaa aac gag acg att tgt tcg cc
   gag cca atc gga gag aat gag act atc tgc tcg ct
 g gag cca atc ggc gag aac gag act ata tgc tcg cc
 c gag cca ata ggt gag aac gag acg att tgt tcg g
gg gac cca ata ggt gaa aac gag acg att tgt tcg a
   gaa cct att ggc gaa aac gaa act att tgt tct taa
```

The rest is simple: using the codon table, we can easily decipher this sequence:

Glu Pro Ile Gly Glu Asn Glu Thr Ile Cys Ser

Or, if we record it using a single letter code, it will look like:

E P I G E N E T I C S

LINGUISTICS

5 points:

The following sentences have been translated from English to a certain constructed language from the early 20th century. The translations are given without punctuation or capitalization and in a random order:

was the scientist in the house?
i am not in the house.
is the baby with you?
the scientist will not have been with the baby.
i will have been with the baby and the scientist.
i will be in the town with the baby.

ab nela in at dubal
ab eta ir at ragab ud at rilica
ab ema in at ducaf ir at ragab
at rilica neta ir at ragab
wela ragab ir ac
weka at rilica in at dubal

Match the sentences to their English translations, then translate the following sentences:

will the scientist have been with you?
i was in a town with the scientist.
you are not in the town.

Answer:
was the scientist in the house?
weka at rilica in at dubal
i am not in the house.
ab nela in at dubal
is the baby with you?
wela ragab ir ac
the scientist will not have been with the baby.
at rilica neta ir at ragab
i will have been with the baby and the scientist.
ab eta ir at ragab ud at rilica
i will be in the town with the baby.
ab ema in at ducaf ir at ragab

will the scientist have been with you?
weta at rilica ir ac?
i was in a town with the scientist.
ab eka in (a) ducaf ir at rilica (with or without the ‘a’)
you are not in the town
ac nela in at ducaf.

Solution:

This language is Ro, a constructed language developed in the early 20th century. Sentences can be matched using logical reasoning - for example, dubal appears twice, as does the word house.

Sentence structure is subject-verb-object (except questions, which are verb-subject-object), so mostly the same as english.

Verbs have a tense, and can have a w- prefix to indicate a question or a n- prefix to indicate a negation. Verbs do not have a number. The subject of the verb is placed directly next to the verb, either as a pronoun or noun.

Vocabulary:
ab = i
ac = you
at = the
ducaf = town
rilica = scientist
ragab = baby
dubal = house
in = in
eba = to be

Tenses:
eka = was
ema = will be
eta = will have been
ela = am/is

my mistake: no translation was provided in the puzzle of ‘a’, the indefinite article. a fun fact if you’re curious, Ro uses ‘ap’ for the English ‘a’

10 points:

This language was built around the concept that words that are closely related in meaning should also be closely related in spelling. Given some sentences in this language and their English translation, fill in the blanks in the following column of words and their translations.

Sing a song of six-pence, a pocket-ful of rye.
Four and twenty black-birds baked in a pie.
When the pie was opened the birds began to sing.
Wasn’t that a dainty dish to set before the king?
The king was in his counting house, counting out his money.
The queen was in the parlour, eating bread and honey.
The maid was in the garden, hanging out the clothes.
When down came a blackbird and pecked off her nose!

Miqadec ap misoda iv zal-gerdac, ap pegap-gisob iv luraf.
Zaf ud zacax bodom-mula pokicef in ap polar.
Avit at polar jeboded at mulaz jekibef migadeb.
Weka ni am ap lagom dogab dabiseb ik at rajad?
At rajad eka in ashe zojer-dubal, zojer ashe gerac.
At rajoda eka in at dugaf, ligiber polab ud pojaq.
At ramibaf eka in at dalida, paniker at dibacz.
Avit oj kepelef ap bodom-mula ud ligibef of ashe meraq!

Fill in the vocabulary table below (each ? represents one letter, each _ represents a full word)
song - _
house - _
bird - _
kingdom - ???oda
prince - ??kmad
baked - _
to bake - _
number - ??
four - _
five - ???

Answer:
song - *misoda*
house - *dubal*
bird - *mula*
kingdom - *raj*oda
prince - *ra*kmad
baked - *pokief*
to bake - *pokieb*
number - *za*
four - *zaf*
five - *zag*

Solution:

song - *misoda* - from line 1

house - *dubal* - from line 5 (and the 5pt problem)

bird - *mula* - from line 2

**there is a typo in line 2, it's singular where it should be plural. mula is the singular

kingdom - *raj*oda ** - the root is taken from king and queen (rajad & rajoda)

**in the poem queen should be rajaf. rajoda = kingdom is correct.

prince - *ra*kmad - see above

baked - *pokief* - from line 2

to bake - *pokieb* - infinitive form modeled in “to set” from line 4, where the -f is removed and replaced with -b

number - *za* - root of four, from line 2

four - *zaf* - from line 2

five - *zag* - uses the number root za, with an ending -g. Remember four is zaf, and six is zal (line 1) and the language is supposed to proceed logically. But why from -f to -l? That’s -f, -g, -h, -i, -j, -k, -l ? Following this sequential logic, the numbers (not seen in the poem) would go za, zab, zac, zad, zaf, zag (number, 1,2,3,4,5). -h, -i, -j, and -k are unused as they would repeat sounds used in the previous five numbers, making six *zal*.

COMPUTER SCIENCE

- Your program should be written in Java or Python-3
- No GUI should be used in your program: eg., easygui in Python
- All the input and output should be via files named as specified in the problem statement
- Java programs should be submitted in a file with extension .java; Python-3 programs should be submitted in a file with extension .py
- **No .pdf, .doc, .docx, etc! Programs submitted in incorrect format will not receive any points!**

Six Degrees of Separation

The famous adage states that all people on Earth are at most 6 social connections away from each other, which means that a chain of "friend of a friend" statements can be made to connect any two people in a maximum of six steps.

This month's problems will have a common input format. Each line in the input file **input.txt** will contain a representation of “A is a friend of B,” with A and B separated by space. For example:

```
Alice Bob
Alice Ben
Alice Carry
Bob Ben
Ben George
George Alice
Alice Laura
Carry Laura
```

Note that this set of connections can be depicted as a non-directed graph, where people are nodes (called *vertices*), and their friendships are links between the corresponding nodes (called *edges*).

5 points:

Given an input as described above, your program should find the person with the most friends of her friends. The name of the person should be written to the **output.txt** file.

Solution:

Python-3:

```
import numpy as np
import os

def emptyGraph():
    return {}

def addUndirEdge(G,u,v):
    if u not in G:
        G[u] = {}
    if v not in G:
        G[v] = {}
    G[u][v] = 1
    G[v][u] = 1
    return
```

```
def readGraph(input_file):
    '''This procedure takes the name of a text file describing a directed or
    undirected graph and returns a data structure in memory.
    The file is structured as follows: each line lists an edge as a pair u,v,
for an unweighted
    graph. The data structure it returns is a dictionary-of-dictionaries
adjacency structure with
    strings as keys.
    '''
    with open(input_file, 'r') as f:
        raw = [s.split(' ') for s in f.read().splitlines()]
    G = emptyGraph()
    for edge in raw:
        addUndirEdge(G, edge[0], edge[1])
    return G

def writeGraph(G, output_file):
    # G is a dictionary of dictionaries
    with open(output_file, 'w') as f:
        for u in G.keys():
            for v in G[u]:
                f.write("{} {}\n".format(u,v))
    return

def FoFcounts(G):
    fof_dicts = {}
    fof_counts = {}
    for u in G:
        fof_dicts[u] = {}
        for v in G[u]:
            for w in G[v]:
                fof_dicts[u][w] = 1
        fof_counts[u] = len(fof_dicts[u]) - 1
    return fof_counts

def argmax(counts_dict):
    maxsofar = -np.inf
    maxindex = None
    unique = True
    for u in counts_dict:
        if counts_dict[u] > maxsofar:
            maxsofar = counts_dict[u]
            maxindex = u
            unique = True
        elif counts_dict[u] == maxsofar:
            unique = False
    return maxindex, maxsofar, unique

def main():
    G = readGraph("input.txt")
    nodename, fofcount, unique = argmax(FoFcounts(G))
    with open("output.txt", 'w') as f:
        f.write(nodename)
```

```
        f.write("\n")

if __name__ == "__main__":
    main()
```

10 points:

Given an input as described above, your program should find the pair of people who have the largest **number of paths with 6 edges** connecting them. Your algorithm should be able to handle graphs with thousands of nodes and edges in a reasonable amount of time (say at most a minute). The names in the pair should be written to the **output.txt** file in sorted order in a space-separated format.

Note: for simplicity, it is allowed for a path to pass through the same node more than once.

Solution:

Python-3:

```
import numpy as np
import os

def emptyGraph():
    return {}

def addUndirEdge(G,u,v):
    if u not in G:
        G[u] = {}
    if v not in G:
        G[v] = {}
    G[u][v] = 1
    G[v][u] = 1
    return

def readGraph(input_file):
    '''This procedure takes the name of a text file describing a directed or
    undirected graph and returns a data structure in memory.
    The file is structured as follows: each line lists an edge as a pair u,v,
for an unweighted
    graph. The data structure it returns is a dictionary-of-dictionaries
adjacency structure with
    strings as keys.
    '''
    with open(input_file, 'r') as f:
        raw = [s.split(' ') for s in f.read().splitlines()]
    G = emptyGraph()
    for edge in raw:
        addUndirEdge(G, edge[0], edge[1])
    return G

def writeGraph(G, output_file):
    # G is a dictionary of dictionaries
    with open(output_file, 'w') as f:
        for u in G.keys():
            for v in G[u]:
```

```
                f.write("{} {}\n".format(u,v))
    return

def countpathsSource(G, u, k=6):
    # Assume G is a (directed) graph,
    # u and v are names of nodes in G
    # k is an integer at least 1
    n = len(G)
    counts = {}
    for x in G:
        counts[x] = 0
    counts[u] = 1
    for i in range(0,k):
        # counts[x] is the number of paths with i edges from u to x
        # newcounts[x] will store number of paths of length i+1 from u to x
        newcounts = {}
        for x in G:
            newcounts[x] = 0
        for x in G:
            for y in G[x]:
                newcounts[y] += counts[x]
        counts = newcounts
        # Now counts[x] stores # of paths of length i+1 from u to x
    return counts

def maxpaths(G):
    maxsofar = -np.inf
    maxpair = []
    otherpair = []
    is_unique = True
    for u in G:
        counts = countpathsSource(G,u)
        for v in counts:
            if counts[v] > maxsofar and u != v:
                maxsofar = counts[v]
                maxpair = [u,v]
                is_unique = True
                otherpair = []
            elif counts[v] == maxsofar and maxpair != [v,u]:
                is_unique = False
                otherpair = [u,v]
    maxpair.sort()
    otherpair.sort()
    return maxpair, maxsofar, is_unique, otherpair

def main():
    G = readGraph("input.txt")
    maxpair, maxsofar, is_unique, otherpair = maxpaths(G)
    with open("output.txt", 'w') as f:
        f.write("{} {}\n".format(maxpair[0], maxpair[1]))

if __name__ == "__main__":
    main()
```