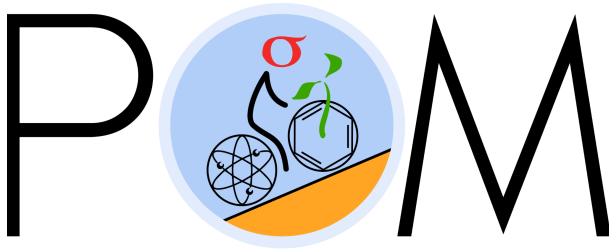
PROBLEM OF THE MONTH



September, 2021

MATHEMATICS

5 points:

A grasshopper starts at the origin and is jumping along a line. For each jump it can choose either direction. First, it jumps 1cm, then 2cm, then 3cm, and so on. Is it possible that after 2021 jumps, the grasshopper ends up exactly at the origin? What is the minimum number of jumps larger than 2021 the grasshopper can take, starting from the origin and jumping in the manner described, to land on the origin again?

Answer: 1) No 2) 2 jumps

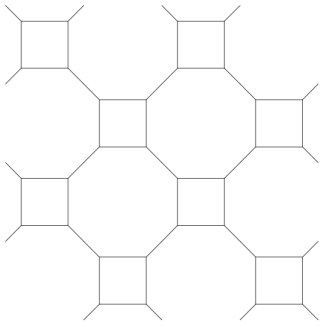
Solution: The order in which the grasshopper takes each jump does not matter (ie, jumping 2021 cm on the first or last jump does not matter). What does matter is the direction, so that we can find the final displacement. So, for example, with 7 jumps, we can group together (1,6), (2,5), (3,4) and 7. We can go right by 1cm and 6cm, then left by 2 and 5 cm, and then right again for 3 and 4, and finally left for 7. The reason this works is because the total amount traveled is 7*8/2 = 28 cm, which is even so we can divide this distance equally between the two sides using the method described.

In the case of 2021, we have that a total of **2021*2022/2 = 2,043,231** cm was traveled. To end up on the origin, we need to travel the same amount to the right and to the left, but 2,043,231 is odd, so this cannot be divided by 2. Thus, the grasshopper cannot land on the origin because it cannot jump an equal amount to the left and right.

After two more jumps though, the grasshopper can reach the origin, because **2,043,231 + 2022 + 2023 = 2,047,276**, which is even so it can be split up on the two sides.

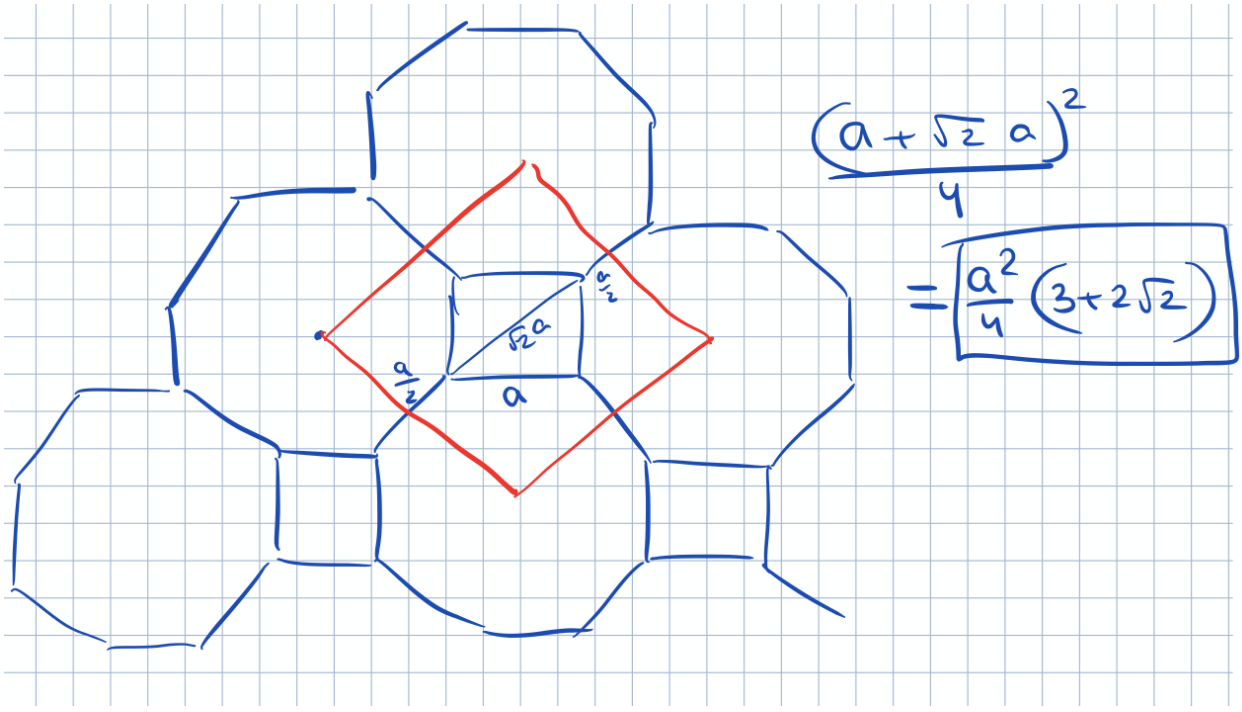
10 points:

Consider the following infinite two-dimensional lattice made out of regular octagons and squares. What is the average area per vertex for this lattice if all edges of the lattice have the same length *a*?



Answer: $a^2 \frac{3+2\sqrt{2}}{4} = a^2\left(\frac{1+\sqrt{2}}{2}\right)^2$

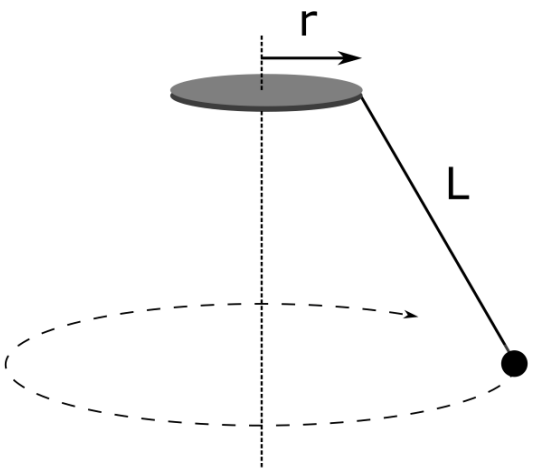
Solution: There are many ways to solve this problem. Here is one:



Here it is clear that the whole lattice can be obtained by translating the square shown in red. This square has an area $(a/2 + \sqrt{2}a + a/2)^2 = a^2(1 + \sqrt{2})^2 = a^2(3 + 2\sqrt{2})$. There are 4 vertices within this cell so the area per vertex is $a^2 \frac{3+2\sqrt{2}}{4}$.

PHYSICS

5 points:

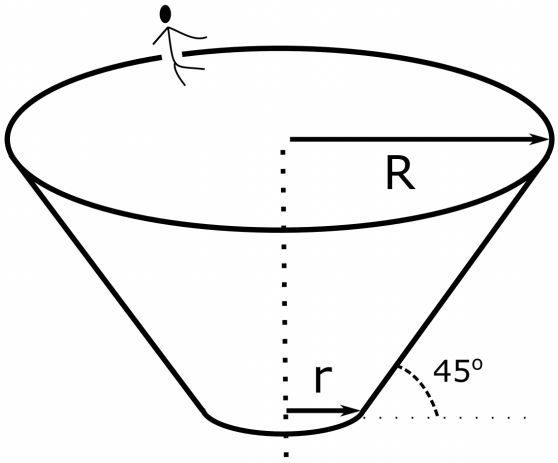


A weight of mass 1kg is hanging from the edge of a disk on a thin thread. The breaking force of the thread is 20 N. The disk can spin on its axis. The disk begins stationary, and then slowly starts to accelerate. To what height with respect to its initial position will the mass rise before the string breaks?

Answer: L/2

Solution: At the breaking point, the force along the string is 20N, and the vertical force is the force of gravity, mg = 10N. Therefore, the string forms an angle of 60 degrees with the vertical axis, which means that the mass will be at height L/2 at breaking point.

10 points:



A water park visitor gets onto a funnel-shaped water slide with a slope of 45 degrees, top opening *R = 5 m*, and bottom opening *r = 1 m*. The rider launches themselves on the water slide horizontally with the speed *v = 5 km/hr*. The slide is wet, so there is no friction. What will be the rider's speed at the exit of the funnel slide?

Answer: 9 m/s

Solution: The funnel is frictionless, and the centripetal force is perpendicular to the direction of motion, so the horizontal velocity will remain the same, but the vertical velocity will increase according to the conversion of energy from potential to kinetic. The horizontal velocity v_h is 5 km/hr = 1.4 m/s. The vertical vertical velocity v_v is obtained from $mgh = mv^2/2$, with h = R-r = 4 m. So, v_v = sqrt(2gh) = 8.9 m/s. The total velocity v = sqrt(v_h^2+ v_v^2) = 9 m/s

CHEMISTRY

5 points:

Emelianiya huxleyi is a tiny unicellular marine algae. This organism is unique because it is arguably the only species whose life cycle directly affects the Earth's climate. These algae produce massive seasonal blooms that affect atmospheric carbon (CO_2) due to two planetary scale processes. The first process is the binding of the atmospheric CO_2 due to photosynthesis. The second process is the conversion of hydrocarbonate ion (HCO_3^-) dissolved in water into insoluble calcium carbonate. The latter is used by algae for building the algal carbonate shell, and it precipitates when the algae die. Formation of calcium carbonate can be illustrated using the following (incomplete) scheme:

$$HCO_3^- + Ca^{2+} \rightarrow CaCO_3$$

Please, complete the equation and tell how this process affects the overall carbon balance in the atmosphere, i.e. how the amount of atmospheric CO_2 changes per one molecule of calcium carbonate formed.

Hint:

What happens to the hydrocarbonate ion (HCO_3^-) when concentration of H^+ (acidity) goes up?

Answer:

The amount of atmospheric CO_2 increases when calcium carbonate is produced by *E. huxleyi*.

Solution:

It is natural to conclude that, since the reaction between dissolved hydrocarbonate ions and calcium ions yields insoluble calcium carbonate, that process leads to a decrease of dissolved carbon in marine water. Keeping in mind that there is an equilibrium between the ocean waters and the atmosphere, it would be logical to conclude that the overall result of the process of calcium carbonate precipitation is the *decrease* of the amount of carbon (in a form of CO_2) in the atmosphere.

However, a conversion of the hydrocarbonate to calcium carbonate requires that the former must dissociate first:

$$HCO_3^- \rightarrow CO_3^{2-} + H^+$$

followed by a reaction between the carbonate and calcium ions:

$$CO_3^{2-} + Ca^{2+} \rightarrow CaCO_3 \downarrow$$

That means formation of one mole of calcium carbonate is accompanied by formation of one mole of H^+, i.e. an acid. What happens to the hydrogen ion (H^+)? It reacts with bases, and the most common base in marine water are hydrocarbonate ions, the same ions that serve as a source for carbonate ions formation (yes, HCO_3^- is a base, for any ion that forms during a dissociation of an acid is a base, i.e. a species capable of binding to H^+). This process occurs according to this equation:

$$HCO_3^- + H^+ \rightarrow H_2CO_3$$

That is the very same process that you can observe when you add vinegar (i.e. acetic acid) to baking soda (sodium hydrocarbonate, a.k.a. sodium bicarbonate). Carbonic acid is unstable, and it quickly decomposes:

$$H_2CO_3 \rightarrow H_2O + CO_2$$

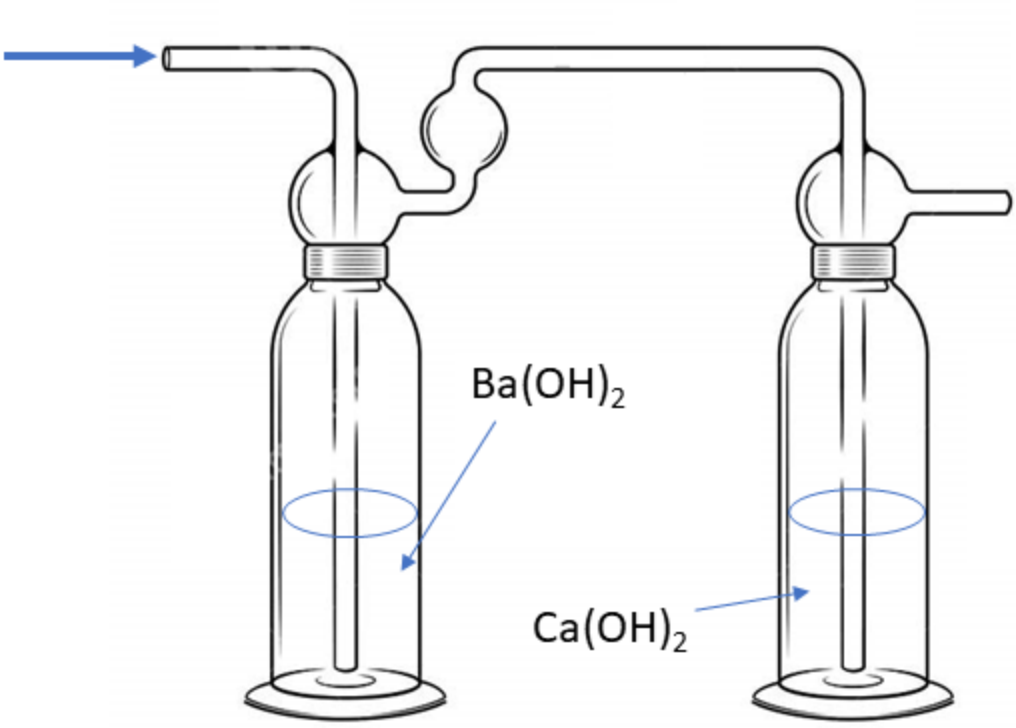
The second product (CO_2) escapes to the atmosphere. Therefore, when *Emelianiya huxleyi* is building their carbonate shells, formation of one molecule of insoluble calcium carbonate is accompanied by formation of one molecule of carbon dioxide, which escapes to the atmosphere. Therefore, instead of decreasing the overall concentration of CO_2 in the Earth's atmosphere, this process leads to its increase.

10 points:

Two Drexel's flasks were connected sequentially (as shown on the figure below), the first one was filled with 40 mL of saturated $Ba(OH)_2$ solution, the second one was filled with 40 mL of saturated $Ca(OH)_2$ solution (the temperature was 20°). Lesha, who was walking by, blew bubbles through them (the direction is shown with a blue arrow), and white precipitation quickly formed in the first flask, whereas the liquid in the second flask remained clear. Zhenya collected the precipitate by filtration, dried it, and its weight appeared to be 197 mg.

Next day, Zhenya prepared fresh solutions, but poured $Ca(OH)_2$ into the first flask, and $Ba(OH)_2$ into the second one. Lesha, who again was walking by, decided to repeat his trick, but precipitation was observed in both flasks. The weight of the two precipitates was 94 and 12 mg, accordingly.

Assuming that the amount of CO_2 exhaled by Lesha was the same both times, how much of CO_2 does he exhale?



Hint:

Since no precipitation occurred in the second bottle on the first day, you can assume that all gas that caused precipitation was fully absorbed by the liquid in the first bottle.

Answer:

The amount of CO_2 exhaled by Lesha is 1 mmol, or 22.4 mL.

Solution:

Reactions in these flasks occur according to equations:

$$Ba(OH)_2 + CO_2 \rightarrow BaCO_3 + H_2O$$

and

$$Ca(OH)_2 + CO_2 \rightarrow CaCO_3 + H_2O$$

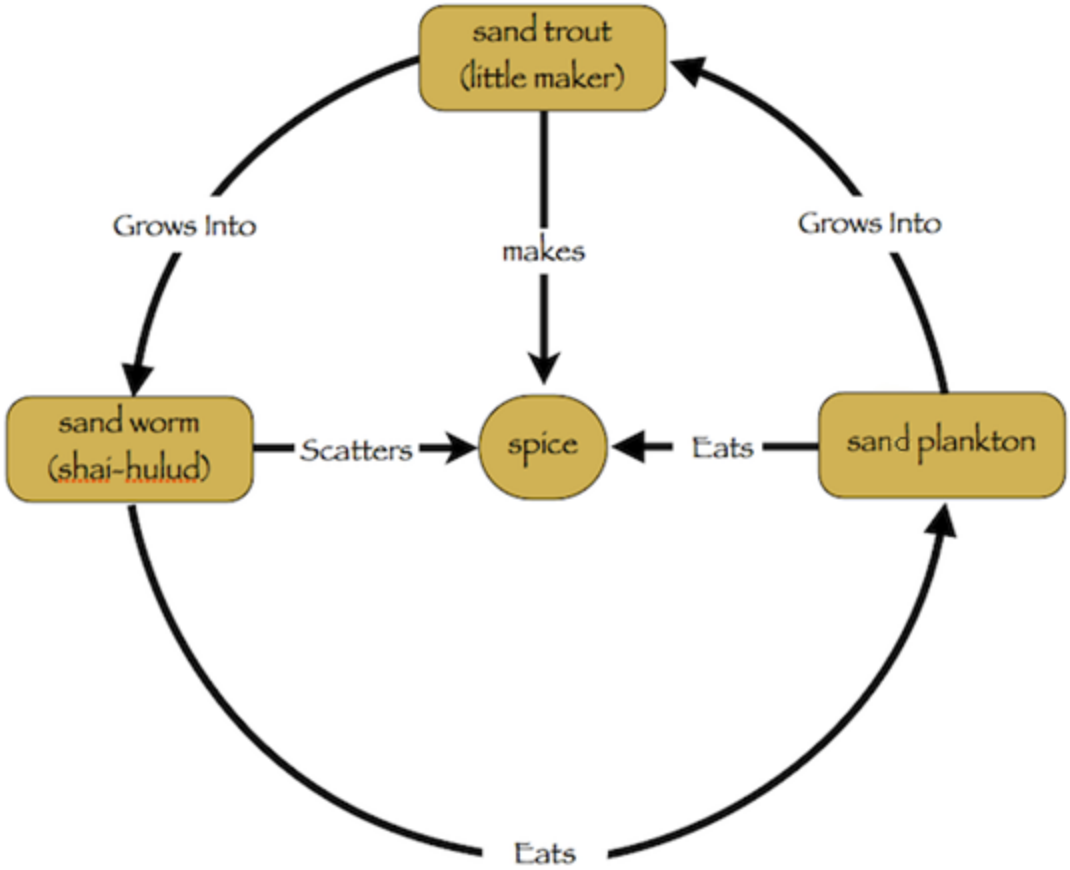
When Lesha blew bubbles for the first time, the amount of barium carbonate (a precipitate in the first bottle) was 197 mg, or exactly 1 mmol (molecular weight of $BaCO_3$ is 197), which means the amount of CO2 exhaled by Lesha was 22.4 L/1000 = 22.4 mL. That is true only if we assume that all CO_2 exhaled by Lesha was absorbed in the first flask. However, it might be possible that

the first flask contained an insufficient amount of $Ba(OH)_2$ to absorb all CO_2 exhaled by Lesha, or that some part of the gas was not absorbed. To check that, let's calculate the amount of $Ba(OH)_2$ in the first flask. Barium and calcium hydroxide solutions are saturated, so their concentrations are 38.9 g/L and 1.73 g/L, respectively. Their molecular weights are 171.34 g/mol and 74.093, so the molarity of these solutions is 227 mM and 24 mM, respectively. 40 mL of each solution contains ~9.1 and ~0.96 mmoles of barium and calcium hydroxide, respectively, which means the first bottle can absorb 1 mmol of CO_2. The lack of precipitation if the second bottle serves as an additional proof for that conclusion. When Zhenya switched the bottles (in the second experiment), some amount of CO_2 could not be absorbed (it contained less than 1 mmol of hydroxide), and the remaining CO_2 was absorbed in the second bottle. If we divide the mass of precipitates in each bottle on their molecular masses, we get 94/100=0.94 and 12/197=0.06 moles, which gives in total 1 mmol, similar to the amount obtained in the first experiment. Therefore, the amount of CO_2 exhaled by Lesha is 1 mmol, or 22.4 mL.

BIOLOGY

5 points:

The planet Arrakis described in the *Dune* novel series is populated by giant sandworms, which play a key role in the planet's ecosystem. The life cycle of Dune's sandworms constitutes a complex ecological feedback system, and different sources provide different versions. One of the summaries is as follows:



During its development, a sandworm passes several phases (Fig.1). Sandworm larvae develop from "sand plankton", which feed upon spice scattered by sandworms. Sandtrout, which are flat and rhombus shaped creatures, find underground water and bind to each other to form "living cisterns" beneath the surface of Arrakis. By doing that, they make the planet dry enough to make the existence of sandworms possible. In those "living cisterns", sandtrout converts water into a mixture known as pre-spice mass and a large amount of CO_2. This causes gigantic explosions, where a huge amount of pre-spice mass is moved closer to the planet surface and the majority of sandtrout are killed. A small amount of survived sandtrout form cysts, which, after

a 6-year hibernation period, yields a pre-sandworm form. The latter may grow into a full-size sandworm, which, like whales, are travelling across the desert sands and devouring sand plankton.

Please answer if the above described ecological system can exist, at least, theoretically. If you think this description is incomplete, explain what is missing from that description. If you think this life cycle is impossible in principle, explain why.

In your explanations, please ignore biochemistry related aspects, focus on ecology.

Hint:

One possible approach would be to look at that from the point of view of conservation laws.

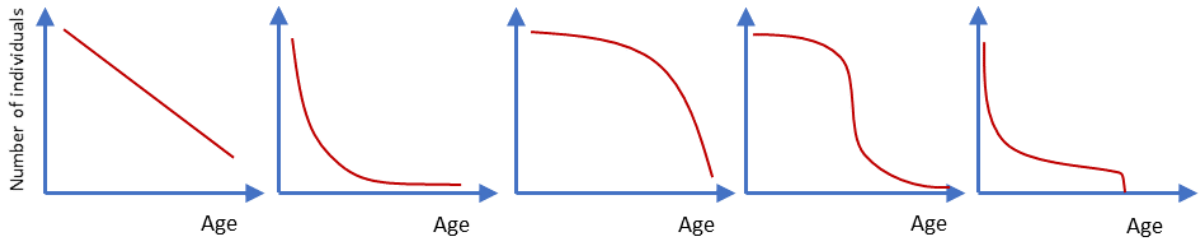
Answer:

Food chains in any stable ecosystem must include three different types of organisms: producers (organisms that produce organic matter from inorganic by using some external energy source), consumers (organisms that consume organic matter produced by producers), and decomposers (organisms that recycle everything that is produced by producers and consumers, including their corpses, into matter, which will be reused by producers in the next cycle). An example is the savannah ecosystem: grass uses minerals from the soil, CO_2 from the air and sunlight to produce organic matter; it is consumed by antelopes (which are also prey for lions), and insects and bacteria in the soil of the savanna decompose animal excrements and their carcasses in order to regenerate inorganic (and simple organic) substances, thereby recreating soil for plants. In parallel, all species emit CO_2, which is the main source of organic matter production by plants. If even one element of this food chain is missing, the ecosystem is unstable and rapidly degrades. It is important that each of the three main players (producer, consumer and decomposer) is of a different species. The reason is obvious: a species based on cannibalism cannot survive. Indeed, if the main food of adult animals is their offspring, the principles of natural selection direct the evolution of this species in opposite directions: if adult animals eat offspring too efficiently, they have an evolutionary advantage over other animals of the same species. However, their effectiveness in hunting juveniles undermines the future of the species as a whole. As a result, the species becomes unstable and will not survive long.

The ecosystem depicted in the above figure implements this erroneous principle: in the Arrakis ecosystem, by and large, there is only one species, and it acts as a producer, consumer and decomposer simultaneously. It is clear that the description of this ecosystem is extremely incomplete, and there must definitely be other species that play an important role in the food web. Hopefully, future scientific expeditions to that planet will fill this gap in our knowledge.

10 points:

A group of researchers monitored the population of several species on a large island X. They obtained the following plots for five different species A, B, C, D, and E (Figs A-E), where population density is plotted vs age. For the convenience of comparison, the data are shown in relative figures.



A B C D E

By looking at the plots, answer the following questions and explain your answers:

1. Which of those species are closer to the top of the food chain pyramid?

2. If few representatives of one of those species will be transferred to an island Y with a similar ecosystem, which of those species is more likely to demonstrate fast and explosive growth?

3. Which of the species A-E is more likely to care for their offspring?

4. Which of those species is more likely to be a mammalian and which is an arthropod?

5. Which of those curves are likely to be a result of incorrect measurements?

Hint:

“K-strategy vs r-strategy”

Answer:

1. Obviously, the typical belief of young populations on **curve B** is to be eaten by someone, so these species are more likely to end up at the bottom of the food chain.

2. Species that are limited by external factors (e.g., predators) are more likely to grow rapidly under new conditions, so the answer is “**B**”.

3. Parental protection of young people is the most important factor in their survival, so the answer is “**C**”.

4. Mammals cannot produce numerous offspring for purely physiological reasons, so the K-strategy is more likely for them. Answers: "**B - arthropods, C - mammals.**"

5. **D**.

Solution:

According to "r / K selection theory" (named after the coefficients in the equation describing population statistics), the two extreme types of strategy or reproduction are: "produce as many offspring as possible and forget about them" and "produce a very limited number of offspring and invest the effort into their survival. " The first strategy leads to curve B (many young animals, only a small part of which survive to adulthood: in this case, the probability of death is inversely proportional to age), the second strategy leads to curve C (most offspring survive, adult animals die for natural reasons, usually in as a result of aging processes: the probability of death is almost constant until a certain threshold age is reached, after which the probability of death increases exponentially). Obviously, there may be intermediate strategies. For example, some reptiles take care of their offspring, so their survival rate is quite high. At the same time, adult animals grow throughout life and do not show signs of aging, so their death is usually the result of an accident (which does not depend or even inversely depends on their age). Curve A describes the population distribution for this type of species. Finally, some species, such as salmon, have a very specific life cycle. They use the "r-strategy" in the first stage of their life, but their puberty does not occur until the end of life. At this time, they spawn and, due to a deep hormonal shift in their body, die almost immediately after that. This situation is described by curve E. We do not know of the species whose age distribution is described by curve D, so the answer to question 5 is “D”. As for the other answers, they can be deduced from the above explanation.

COMPUTER SCIENCE

- Your program should be written in Java or Python-3
- No GUI should be used in your program: eg., easy gui 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 .txt, .dat, .pdf, .doc, .docx, etc. Programs submitted in incorrect format will not receive any points!

According to the famous myth, inhabitants of the Tower of Babel spoke many different languages. Some spoke multiple languages, some spoke just one. For example, Anatu spoke Akkadian and Aramaic, Nergal spoke Aramaic and Moabite, Nuska spoke Moabite and Palmyrene, and Tiamat spoke just Palmyrene. This allowed Anatu, Nergal, Nuska and Tiamat to communicate as there was always a translation path between any of them: indirect communication is possible if there are pairs of people who can use the same language, who can relay information along the path between the individuals. Your task will be to identify the largest groups of people capable of communicating with each other.

Your program will receive an input file named **input.txt,** which contains the data on what languages the inhabitants of the Tower of Babel can speak. Each line of the file contains a name of the inhabitant followed by a list of languages he/she speaks, all in the comma-separated format. For example:

```
Name_1, language_1, language_3
Name_109, language_1, language_31, language_10
Name_2, language_11, language_33, language_10
Name_3, language_11, language_33, language_10
Name_4, language_2, language_3
```

5 points:

Find up to 2 languages that allow the largest group of people to communicate with each other.

Your program should produce an output file named **output.txt** that on the first line lists the 2 languages that allow communication between the largest number of people (in comma-separated format), and on the second line lists the names of all individuals who can communicate with each other using these 2 languages (also in comma-separated format). Both lists should be sorted in ascending order.

In the example above, the combination of language_1 and language_10 lets 4 people to communicate with each other, making the output:

```
language_1, language_10
Name_1, Name_109, Name_2, Name_3
```

Here, Name_2 and Name_3 both can speak to Name_1 through Name_109.

Solution:

Java:

```
/*
Brute force algo:
  for each language
    find people speaking it -> group1
    for each person in the group1
      add their languages into a set
    for each language from this set
      find people speaking it -> group2
      group1 + group2 is our goal; find max of it
*/

import java.io.*;
import java.util.*;
import java.util.stream.Collectors;

public class Babel5 {
  private final Map<String, Set<String>> people = new HashMap<>();
                                    // map of people to languages that they speak
  private final Map<String, Set<String>> languages = new HashMap<>();
                                    // map of languages to people that can speak this language

  void input(String fname) throws Exception {
    try(BufferedReader br = new BufferedReader(new FileReader(fname))) {
      String line;
      while((line=br.readLine()) != null) {
        String[] words = line.split(",");
        String person = words[0].trim();
        people.computeIfAbsent(person, k -> new HashSet<>());
        for(int i=1; i<words.length; i++) {
          String language = words[i].trim();
          people.get(person).add(language);
          languages.computeIfAbsent(language, k -> new HashSet<>());
          languages.get(language).add(person);
        }
      }
    }
  }

  // Note: a group of guys speaking { language1, language2 } are the same as speaking
  // { language2, language1 }  so below can be optimized by excluding the redundant pairs.
  // It's not implemented to keep the code short.
  void findGroup(List<String> maxLanguages, Set<String> maxGuys) {
    for(Map.Entry<String, Set<String>> entry : languages.entrySet()) {
      String language1 = entry.getKey();
      Set<String> guys1 = entry.getValue();
      Set<String> language_set = new HashSet<>();
      for(String guy : guys1)
        language_set.addAll(people.get(guy));
      for(String language2 : language_set) {
        Set<String> group = new HashSet<>(languages.get(language2));
```

```
        group.addAll(guys1);
        //System.out.printf("checking %s and %s\n", language1, language2);
        if(group.size() > maxGuys.size()) {
          maxGuys.clear(); maxGuys.addAll(group);
          maxLanguages.clear(); maxLanguages.add(language1); maxLanguages.add(language2);
          //System.out.printf("new max: %s\n", String.join(",", maxLanguages));
        }
      }
    }
  }

  void output(String fname, List<String> maxLanguages, Set<String> maxGuys) throws IOException
{
    List<String> sortedUniqueLanguages =
       maxLanguages.stream().collect(Collectors.toSet()).stream().collect(Collectors.toList());
    sortedUniqueLanguages.sort(Comparator.naturalOrder());
    List<String> sortedUniqueGuys = maxGuys.stream().collect(Collectors.toList());
    sortedUniqueGuys.sort(Comparator.naturalOrder());
    try(BufferedWriter bw = new BufferedWriter(new FileWriter(fname))) {
      bw.write(String.join(",", sortedUniqueLanguages));
      bw.write("\n");
      bw.write(String.join(",", sortedUniqueGuys));
      bw.write("\n");
    }
  }

  public static void main(String[] args) throws Exception {
    System.out.println("pwd = " + System.getProperty("user.dir"));
    Babel5 babel = new Babel5();
    babel.input("input.txt");
    List<String> maxLanguages = new ArrayList<>();
    Set<String> maxGuys = new HashSet<>();
    babel.findGroup(maxLanguages, maxGuys);
    babel.output("output.txt", maxLanguages, maxGuys);
    System.out.println("end.");
  }
}
```

Python-3:

```
"""
Brute force algo:
  for each language
    find people speaking it -> group1
    for each person in the group1
      add their languages into a set
    for each language from this set
      find people speaking it -> group2
      group1 + group2 is our goal; find max of it
"""

import re

people = {}    # map of people to languages that they speak
languages = {} # map of languages to people that can speak this language

def input(fname):
  with open(fname) as in_file:
```

```
    for line in in_file.readlines():
      words = re.split(",", line.strip())
      person = words[0].strip()
      if person not in people:
        people[person] = set()
      for language in words[1:]:
        language = language.strip()
        people[person].add(language)
        if language not in languages:
          languages[language] = set()
        languages[language].add(person)

# Note: a group of guys speaking { language1, language2 } are the same as speaking
# { language2, language1 } so below can be optimized by excluding the redundant pairs.
# It's not implemented to keep the code short.
def find_group():
  max_languages = []
  max_guys = set()
  for language1, guys1 in languages.items():
    language_set = set()
    for guy in guys1:
      language_set |= people[guy]
    for language2 in language_set:
      guys2 = languages[language2]
      group: set = guys1 | guys2
      # print(f"checking {language1} and {language2}")
      if len(group) > len(max_guys):
        max_guys = group
        max_languages = [language1, language2]
  return max_languages, max_guys

def output(fname, max_languages, max_guys):
  ls = list(set(max_languages)); ls.sort()
  gs = list(max_guys); gs.sort()
  # print(f"result: {ls}, {gs}")
  with open(fname, "w") as out_file:
    out_file.writelines(f"{','.join(ls)}\n")
    out_file.writelines(f"{','.join(gs)}\n")

input("input.txt")
max_languages, max_guys = find_group()
output("output.txt", max_languages, max_guys)
print("end.")
```

10 points:

Find up the largest group of people capable of communicating with each other.

Your program should produce an output file named **output.txt** that one the first line lists the set of languages that enables the communication within the group (in comma-separated format), and on the second line lists the names of all individuals who can communicate with each other (also in comma-separated format). Both lists should be sorted in ascending order.

In the example above, the combination of language_1, language_3 and language_10 lets all people communicate with each other, making the output:

```
language_1, language_10, language_3
Name_1, Name_109, Name_2, Name_3, Name_4
```

Here, Name_2 and Name_3 both can speak to Name_1 through Name_109, and Name_4 is connected to the group via Name_1.

Solution:

Java:

```
/*
If we represent people as nodes, and a pair of persons speaking the same language as an edge connecting these nodes, then we get a graph. A group of folks that can talk to each other (with a help of friends translating) constitutes a connected subgraph. So, our task is to find the largest (by number of nodes) such group.
Our algorithm is the following.
1) Take a random person (a node in our graph).
2) Make a search. This will give us a connected subgraph. We will remember the nodes we visited.
3) For a node from the remaining set repeat 2) until there are no more nodes left.
4) Select the largest subgraph.

Note: we are not asked to provide the minimum list "of languages that enables the communication within the group", therefore we'll list all the common languages within this group. We'll omit unique language for a person but we'll not further prune the graph (until it becomes disconnected).
*/

import java.io.*;
import java.util.*;
import java.util.stream.Collectors;

public class Babel10 {
  private Map<String, Set<String>> people = new HashMap<>();
                                    // map of people to languages that they speak
  private Map<String, Set<String>> languages = new HashMap<>();
                                    // map of languages to people that can speak this language
  // let's use the adjacency list to represent our graph
  private Map<String, Set<String>> adj = new HashMap<>();
                    // key: person, value: list of people with whom he/she can speak the same language
  private Map<String, Integer> name2cluster = new HashMap<>();
               // strictly speaking, it does not have to be a map, we never use its value,
                 // so it could be a set but let's keep it so, just to have the full picture of all
               // the labels
  private Map<Integer, List<String>> cluster2name = new HashMap<>();
  private int clusterLabel = 1;

  void input(String fname) throws Exception {
    try(BufferedReader br = new BufferedReader(new FileReader(fname))) {
      String line;
```

```
      while((line=br.readLine()) != null) {
        String[] words = line.split(",");
        String person = words[0].trim();
        if(!people.containsKey(person))
          people.put(person, new HashSet<>());
        for(int i=1; i<words.length; i++) {
          String language = words[i].trim();
          people.get(person).add(language);
          if(!languages.containsKey(language))
            languages.put(language, new HashSet<>());
          languages.get(language).add(person);
        }
      }
    }
  }

  void buildAdjList() {
    for(String person : people.keySet()) {
      if(!adj.containsKey(person))
        adj.put(person, new HashSet<>());
      for(String language : people.get(person)) {
        for(String guy : languages.get(language)) {
          if(!guy.equals(person))
            adj.get(person).add(guy);
        }
      }
    }
  }

  void cluster() {
    for(String person : adj.keySet()) {
      LinkedList<String> toVisit = new LinkedList<>();
      toVisit.add(person);
      boolean added = false;
      while(!toVisit.isEmpty()) {
        String guy = toVisit.pop();
        if(!name2cluster.containsKey(guy)) {
          name2cluster.put(guy, clusterLabel);
          if(!cluster2name.containsKey(clusterLabel)) {
            LinkedList<String> list = new LinkedList<>();
            list.add(guy);
            cluster2name.put(clusterLabel, list);
          }
          else
            cluster2name.get(clusterLabel).add(guy);
          toVisit.addAll(adj.get(guy));
          added = true;
        }
      }
      if(added)
        clusterLabel++;
    }
  }

  List<String> findMaxCluster() {
              int    keyMax    =    cluster2name.entrySet().stream().max((entry1,    entry2)    ->
entry1.getValue().size() > entry2.getValue().size() ? 1 : -1).get().getKey();
```

```
    return cluster2name.get(keyMax);
  }

  // disregards unique languages, i.e. a language that only 1 person can speak
  // and therefore cannot help in communication for sure
  Set<String> findCommonLanguages(List<String> cluster2nameMax) {
    Map<String, Integer> languagesCount = new HashMap<>();
    for(String person : cluster2nameMax) {
      for(String language : people.get(person)) {
        languagesCount.merge(language, 1, (a, b) -> a + b);
      }
    }
    return languagesCount.entrySet().stream().filter(e -> e.getValue() >
                                                    1).collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue)).keySet();
  }

  void output(String fname, List<String> cluster2nameMax, Set<String> commonLanguages)
              throws IOException {
    try(BufferedWriter bw = new BufferedWriter(new FileWriter(fname))) {
      List<String> commonLanguagesList = new ArrayList<>(commonLanguages);
      commonLanguagesList.sort(Comparator.naturalOrder());
      bw.write(String.join(",", commonLanguagesList));
      bw.write("\n");
      cluster2nameMax.sort(Comparator.naturalOrder());
      bw.write(String.join(",", cluster2nameMax));
      bw.write("\n");
    }
  }

  public static void main(String[] args) throws Exception {
    System.out.println("pwd = " + System.getProperty("user.dir"));
    Babel10 babel = new Babel10();
    babel.input("input.txt");
    babel.buildAdjList();
    babel.cluster();
    List<String> cluster2nameMax = babel.findMaxCluster();
    Set<String> commonLanguages = babel.findCommonLanguages(cluster2nameMax);
    babel.output("output.txt", cluster2nameMax, commonLanguages);
    System.out.println("end.");
  }
}
```

Python-3:

```
"""
If we represent people as nodes, and a pair of persons speaking the same language as an edge
connecting these nodes, then we get a graph. A group of folks that can talk to each other (with
a help of friends translating) constitutes a connected subgraph. So, our task is to find the
largest (by number of nodes) such group.
Our algorithm is the following.
1) Take a random person (a node in our graph).
2) Make a search. This will give us a connected subgraph. We will remember the nodes we
visited.
3) For a node from the remaining set repeat 2) until there are no more nodes left.
4) Select the largest subgraph.
```

```
Note: we are not asked to provide the minimum list "of languages that enables the communication
within the group", therefore we'll list all the common languages within this group. We'll omit
unique language for a person but we'll not further prune the graph (until it becomes
disconnected).
"""

import re

people = {}     # map of people to languages that they speak
languages = {}  # map of languages to people that can speak this language
# let's use the adjacency list to represent our graph
adj = {}          # key: person, value: list of people with whom he/she can speak the same
language
name2cluster = {} # strictly speaking, it does not have to be a dictionary, we never use its
                  # value, so it could be a set, but let's keep it so, just to have the full
                  # picture of all the labels
cluster2name = {}
cluster_label = 1

def input(fname):
  with open(fname) as in_file:
    for line in in_file.readlines():
      words = re.split(",", line.strip())
      person = words[0].strip()
      if person not in people:
        people[person] = set()
      for language in words[1:]:
        language = language.strip()
        people[person].add(language)
        if language not in languages:
          languages[language] = set()
        languages[language].add(person)

def build_adj_list():
  for person in people.keys():
    if person not in adj:
      adj[person] = set()
    for language in people[person]:
      for guy in languages[language]:
        if guy != person:
          adj[person].add(guy)

def cluster():
  global cluster_label
  for person in adj.keys():
    to_visit = [person]
    added = False
    while len(to_visit) > 0:
      guy = to_visit.pop(0)
      if guy not in name2cluster:
        name2cluster[guy] = cluster_label
        if cluster_label not in cluster2name:
          cluster2name[cluster_label] = [guy]
        else:
          cluster2name[cluster_label].append(guy)
        to_visit.extend(adj[guy])
        added = True
```

```
    if added:
      cluster_label += 1

def find_max_cluster():
  len_max = 0
  key_max = None
  for name, cluster in cluster2name.items():
    if len(cluster) > len_max:
      len_max = len(cluster)
      key_max = name
  return cluster2name[key_max]

# disregards unique languages, i.e. a language that only 1 person can speak
# and therefore cannot help in communication for sure
def find_common_languages(cluster2name_max):
  languages_count = {}
  for person in cluster2name_max:
    for language in people[person]:
      if language not in languages_count:
        languages_count[language] = 1
      else:
        languages_count[language] += 1
  common_languages = list(dict(filter(lambda x: x[1] > 1, languages_count.items())).keys())
  return common_languages

def output(fname, cluster2name_max, common_languages):
  with open(fname, "w") as out_file:
    common_languages.sort()
    out_file.writelines(f"{','.join(common_languages)}\n")
    cluster2name_max.sort()
    out_file.writelines(f"{','.join(cluster2name_max)}\n")

input("input.txt")
build_adj_list()
cluster()
cluster2name_max = find_max_cluster()
common_languages = find_common_languages(cluster2name_max)
output("output.txt", cluster2name_max, common_languages)
print("end.")
```