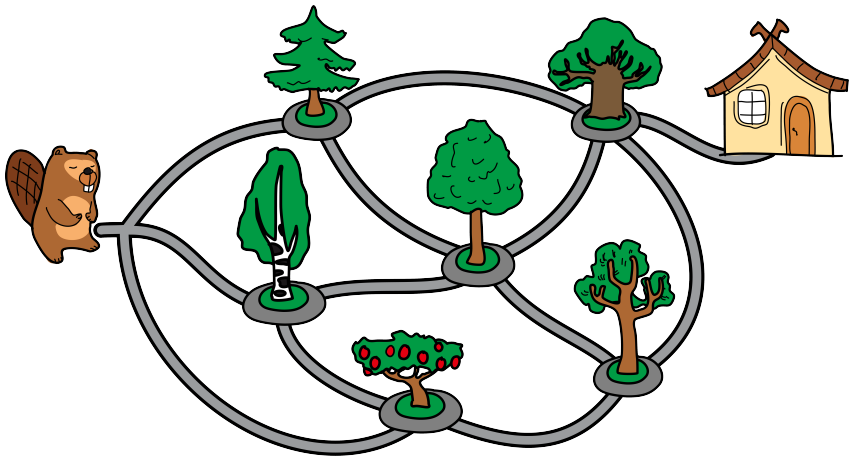
Trail Home

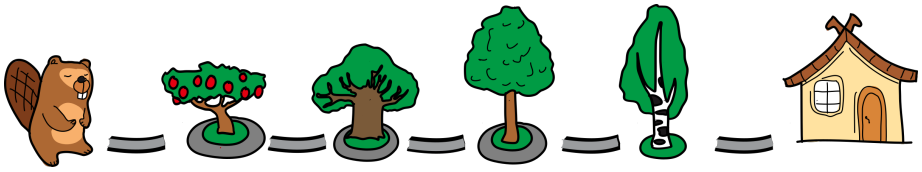
Story

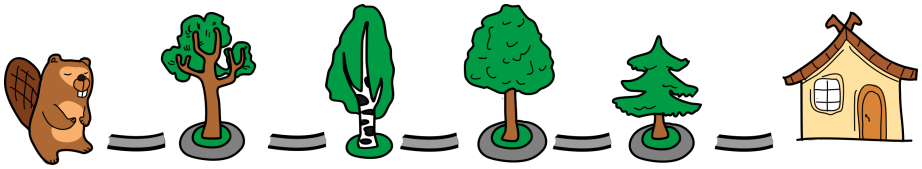
In between a beaver and her home are some trails, where each intersection is marked with a different tree. The beaver walks home using these trails passing exactly four intersections on the way.

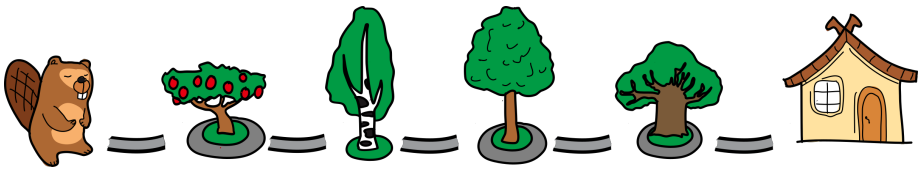


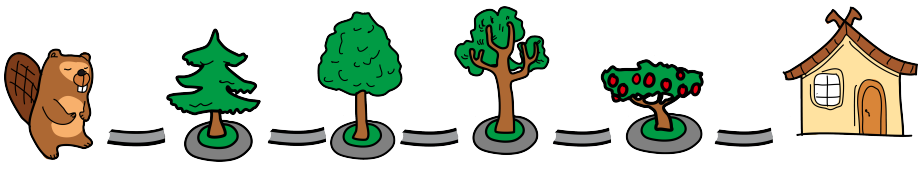
Question

In which order could the beaver have passed the intersections?

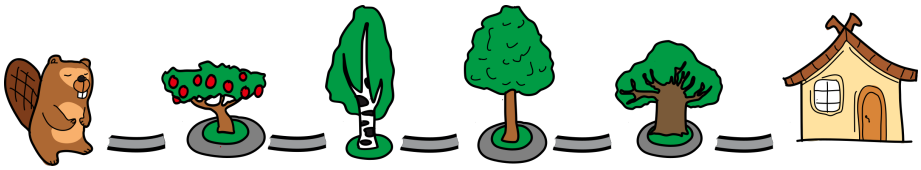
(A) 

(B) 

(C) 

(D) 

Answer

(C) 

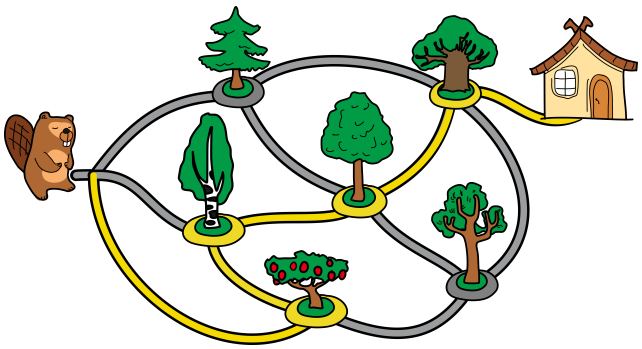
Explanation of Answer

There are several routes that the beaver could have taken home. Of the given options, the only order that matches a route home is Option C.

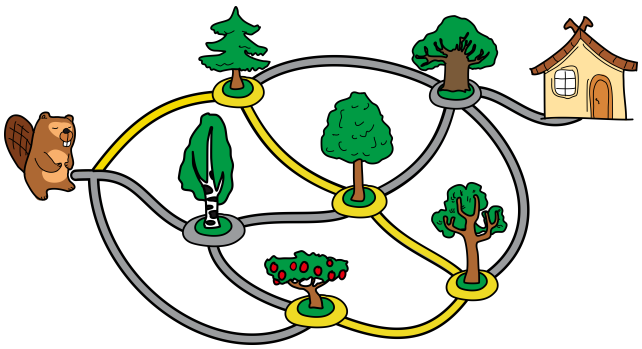
In Option A, it is not possible to reach the intersection marked with  directly after the intersection marked with  .

In Option B, it is not possible for the beaver to reach the intersection marked with  directly from where she starts.

In Option C, the beaver can pass the intersections in the given order and reach home using the route shown.



In Option D, the beaver cannot reach home by passing the intersections in the given order, as shown.



Connections to Computer Science

This problem focusses on keeping track of the *state* that we are currently in, as well as what state we could possibly be in next. Specifically, each tree or house is a possible *state*, and the trails between these states give possibe *next states*.

When a computer program, or *algorithm* is created, keeping track of the state of all the variables is an important process to determine what the program is doing at various points in time. As well, keeping track of the state helps determine if the program is correct. In addition to keeping track of state in computer programs, or *software*, the *central processing unit* (CPU), which is an example of *hardware*, also keeps track of the current state of the machine when programs are running.

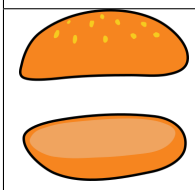
Country of Original Author

Lithuania

Hamburger Recipe

Story

A hamburger is made using the following six ingredients.

Bun	Meat	Sauce	Lettuce	Onions	Cheese
					

The hamburger is made according to the following three rules.

1. The sauce should be directly on top of the meat.
2. The meat and cheese should be somewhere below the lettuce and onions.
3. The onions should not be in contact with the bun.

Question

Which of the following could be the hamburger?

(A) 

(B) 

(C) 

(D) 

Answer

(D) 

Explanation of Answer

We can check each hamburger to see if it follows all the rules.

The hamburger in Option A follows rules 1 and 2. However, the onions are touching the top bun, so it doesn't follow rule 3.

The hamburger in Option B follows rules 1 and 3. However, the cheese is above the lettuce and onions, so it doesn't follow rule 2.

The hamburger in Option C follows rules 2 and 3. However, the cheese is between the meat and the sauce, so it doesn't follow rule 1.

The hamburger in Option D follows all the rules, so this is the correct answer.

Connections to Computer Science

In computer science, determining whether a solution obeys all the given rules is called *constraint checking*. One time this occurs is when information is entered into a *database*. For example, if a database keeps track of employees and their salaries, two constraints may be that each employee has a unique identifier, such as a social insurance number, and each salary needs to be a positive number. When any new employee is added to the database, that new information would need to be checked in terms of those two constraints.

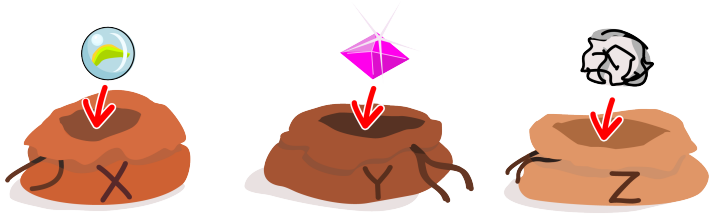
Country of Original Author

South Korea

Lila's Guessing Game

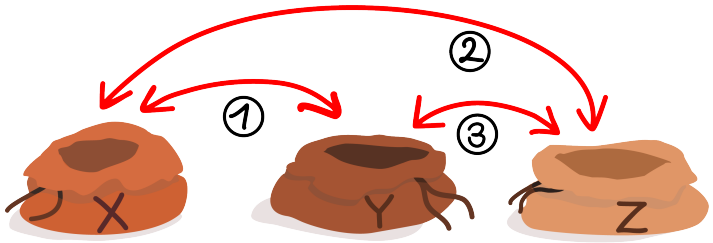
Story

Lila and her friends play a guessing game. To start, Lila puts a marble in bag X, a gem in bag Y, and a crumpled piece of paper in bag Z.



Then, while her friends' eyes are closed, she mixes up the contents of the bags.

1. First, she switches the items in bags X and Y.
2. Then, she switches the items in bags X and Z.
3. Finally, she switches the items in bags Y and Z.



Question

Where are Lila's items now?

(A) The marble is in bag X, the paper is in bag Y, and the gem is in bag Z.

(B) The paper is in bag X, the gem is in bag Y, and the marble is in bag Z.

(C) The gem is in bag X, the paper is in bag Y, and the marble is in bag Z.

(D) The paper is in bag X, the marble is in bag Y, and the gem is in bag Z.

Answer

(B) The paper is in bag X, the gem is in bag Y, and the marble is in bag Z.

Explanation of Answer

Lila switches the items three times. After the first switch the bags look like this:

After the second switch the bags look like this:

After the third and final switch the bags look like this:

Therefore, the paper is in bag X, the gem is in bag Y, and the marble is in bag Z.

Connections to Computer Science

A *permutation* is an arrangement of objects in a particular order. Arranging the objects in a different order creates a different permutation. So the same group of objects can have many permutations. At the start of this task, Lila's items are in the permutation: marble-gem-paper. At the end of the task, the same items are in a different permutation: paper-gem-marble.

An interesting exercise is to determine how many different permutations there are for three objects.

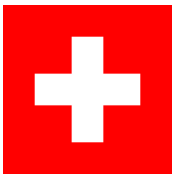
One of the most fundamental concepts in computer science is *sorting*, where items are placed into either ascending or descending order. For instance, when you open a folder, the files are often listed in sorted order by name or by date. Permutations are related to sorting. A sorted list is simply one of many possible permutations of that list.

Many different *sorting algorithms* or sorting techniques have been developed. All sorting algorithms begin with the same permutation (the unsorted list) and they all end with the same permutation (the sorted list). The difference is what happens during the sorting process. The list will pass through many other permutations, but exactly which permutations those are depends on which sorting algorithm has been used.

As well, many sorting algorithms use the idea of *swapping two elements* to move from one permutation to a permutation which is "closer" to being in sorted order.

Country of Original Author

Switzerland



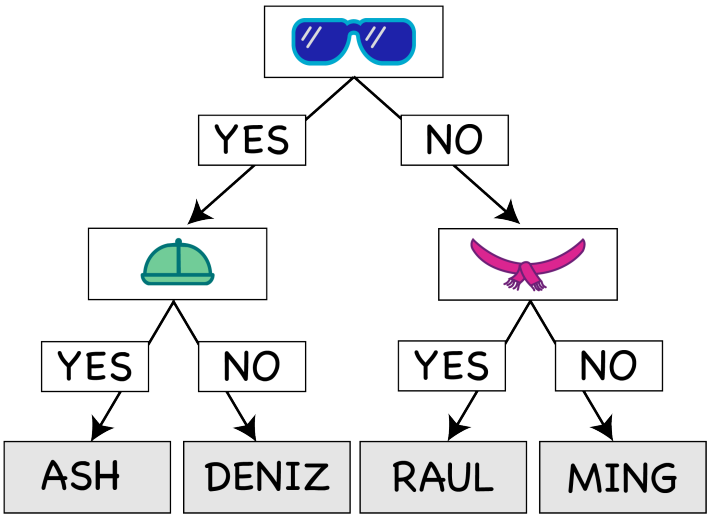
Part B

Remembering Faces

Story

Talia is very forgetful, so she has created a system to help her remember the names of her four group members.

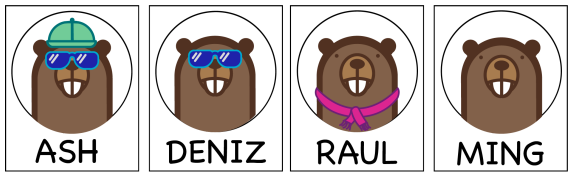
If a group member is wearing sunglasses, Talia checks to see if they are wearing a hat. If they are wearing a hat, then it is Ash, otherwise it is Deniz. If the group member is not wearing sunglasses, Talia checks to see if they are wearing a scarf. If they are wearing a scarf, then it is Raul, otherwise it is Ming.



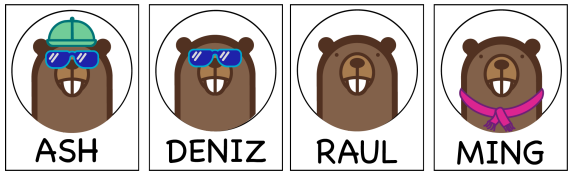
Question

Which of the following correctly matches names with faces?

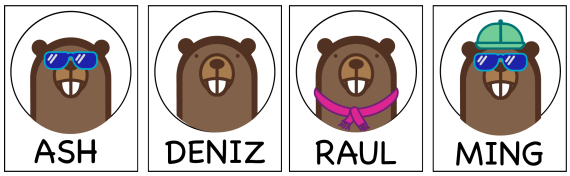
(A)



(B)

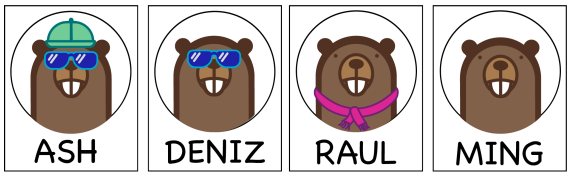


(C)



(D)

Answer

(A) 

Explanation of Answer

Talia's system tells us that

- Ash is wearing sunglasses and a hat,
- Deniz is wearing sunglasses and is not wearing a hat,
- Raul is not wearing sunglasses and is wearing a scarf, and
- Ming is not wearing sunglasses and is not wearing a scarf.

These four things are true for Option A and not the other three options.

Connections to Computer Science

The picture used this problem is called a *decision tree*. A decision tree is composed of *nodes*. The *root node* is where we start the decision process. In this task, this root node is the sunglasses. Some nodes have a set of *binary* decision branches: these are the results of questions being answered either "yes" or "no". The other nodes are *leaf* nodes at the bottom of the tree and they represent the final *outcome* based on the decision path.

Decision trees are used in computer science in *artificial intelligence*, specifically in *machine learning*. For example, when a program is being designed to distinguish between various images, such as "dog", "fish", or "traffic light", various features such as "rectangular shape", "two eyes", "fins" are used as decisions. Each feature has a "yes" or "no" outcome, and with repeated *training*, the program will develop enough distinguishing features to correctly *classify* an image.

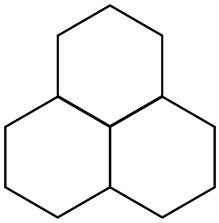
Country of Original Author

Australia

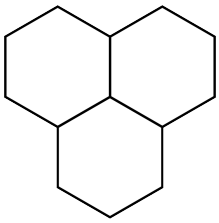
Colourful Tower

Story

Luis has hexagon pieces in three different colours. Whenever Luis arranges three pieces in a way that resembles an upright triangle, the three pieces must either be *all the same colour*, or *all different colours*. These rules do not apply to other three-piece arrangements. In particular:

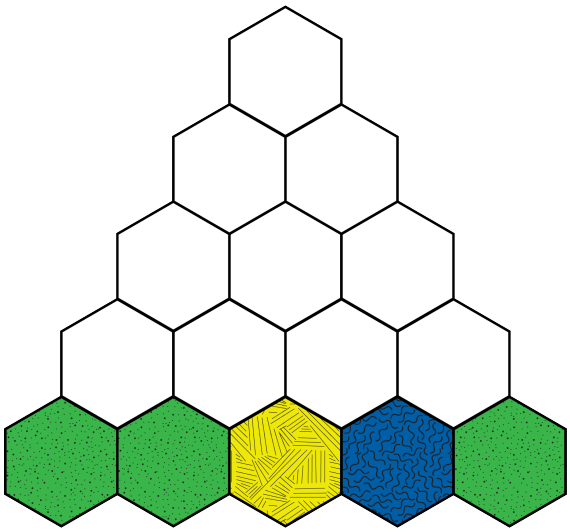


All colours the same
or all colours different



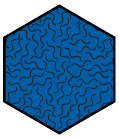
No colour rules

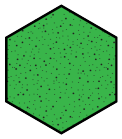
Luis arranges his hexagon pieces in a way that resembles a tower as shown:



Question

Which hexagon piece must be at the very top?

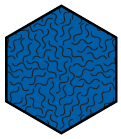
(A) 

(B) 

(C) 

(D) There is more than one possibility

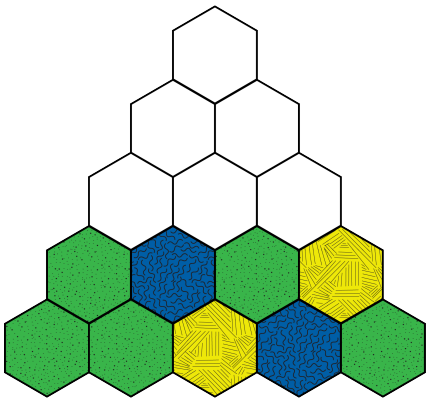
Answer

(A) 

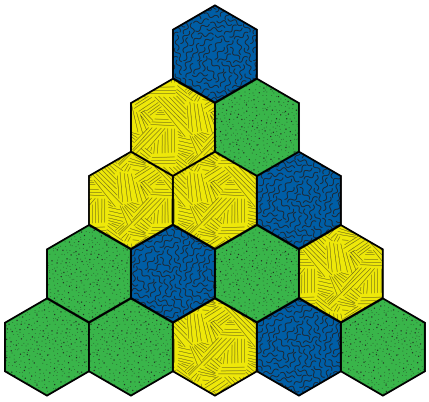
Explanation of Answer

Once we know the colours of two hexagon pieces beside each other, we can determine the colour of the piece directly above them. If two pieces side-by-side are the same colour, then the piece above them will be that colour also. However, if two pieces side-by-side are different colours, then the piece above them will be neither colour.

Using this idea, the next row up from the bottom will be:



We can continue building the tower from the bottom up until it is complete:



The hexagon piece at the very top is blue.

Connections to Computer Science

This task relies on two fundamental concepts in *algorithms*: the concept of *conditional decisions* and *repetition*.

The conditional decision in this task can be thought of as the following:

```
IF colour of cell below left is the same as colour of cell below right
THEN
    colour of this cell is the same as cell below left
OTHERWISE
    colour of this cell is different than both cell below left and cell below right
```

Conditional statements are often represented in *programming languages* using *if-then-else* statements, which allow certain steps to happen only if a given decision/question has a *true* value.

To solve the entire task, the above steps need to be *repeated*, from the bottom levels up to the very top level. In computer programs, repetition is often represented as a *loop*.

Country of Original Author

Vietnam

Apples, Bananas, Broccoli, and Carrots

Story

Some fruit (apples and bananas), and some vegetables (broccoli and carrots) are placed on four plates:

Then the following actions are performed, in the order given:

1. One banana is added to each plate.
2. Each plate with less than four items in total, is removed.
3. All the fruit is removed from each plate.
4. Each plate with at least one carrot and no other fruit or vegetables, is removed.

Question

How many plates remain after all the actions are performed?

(A) 0

(B) 1

(C) 2

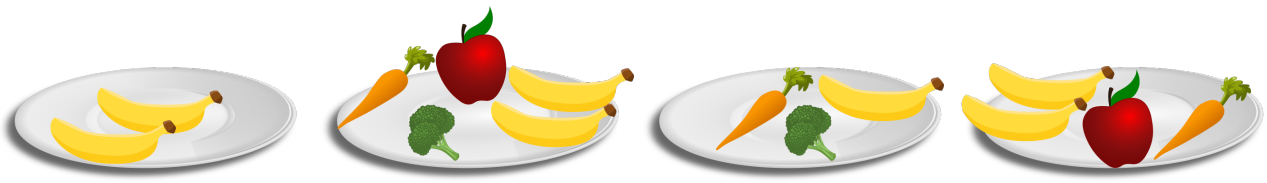
(D) 3

Answer

(B) 1

Explanation of Answer

After the first action, the plates will look like this:



After the second action, two plates will remain:

After the third action, the plates will look like this:

After the fourth action, one plate will remain:

Another way to see this is to notice that the only plates remaining after all the actions are performed will be those that start with some broccoli and at least three items in total. (Can you see why?) Exactly one plate fits these criteria.

Connections to Computer Science

To answer this task, we can carefully follow the steps of an *algorithm*. Each step consists of an action where the input and output of the action is a list of plates.

There are two main *algorithmic patterns* used in this task that transform the list of plates: the *mapping* pattern and the *filtering* pattern.

The *mapping* pattern can be thought of as doing the same operation on every item in the list. Specifically, for this task, adding a banana to each plate or removing all the fruit from each plate are examples of mapping.

The *filtering* pattern can be thought of as keeping (or removing) a subset of items from the list that satisfy certain conditions. For this task, plates with less than four items are removed at step 2 of the algorithm, and plates with at least one carrot and no other fruit or vegetables are removed at step 4.

When computer scientists implement algorithms using a programming language, they often make use of common patterns such as mapping and filtering. For example, Python is a popular modern language with features built into the core language to support the filtering and mapping idioms. As well, the key programming model used to process *big data* (involving terabytes of data) is called *MapReduce*, which combines together mapping and filtering.

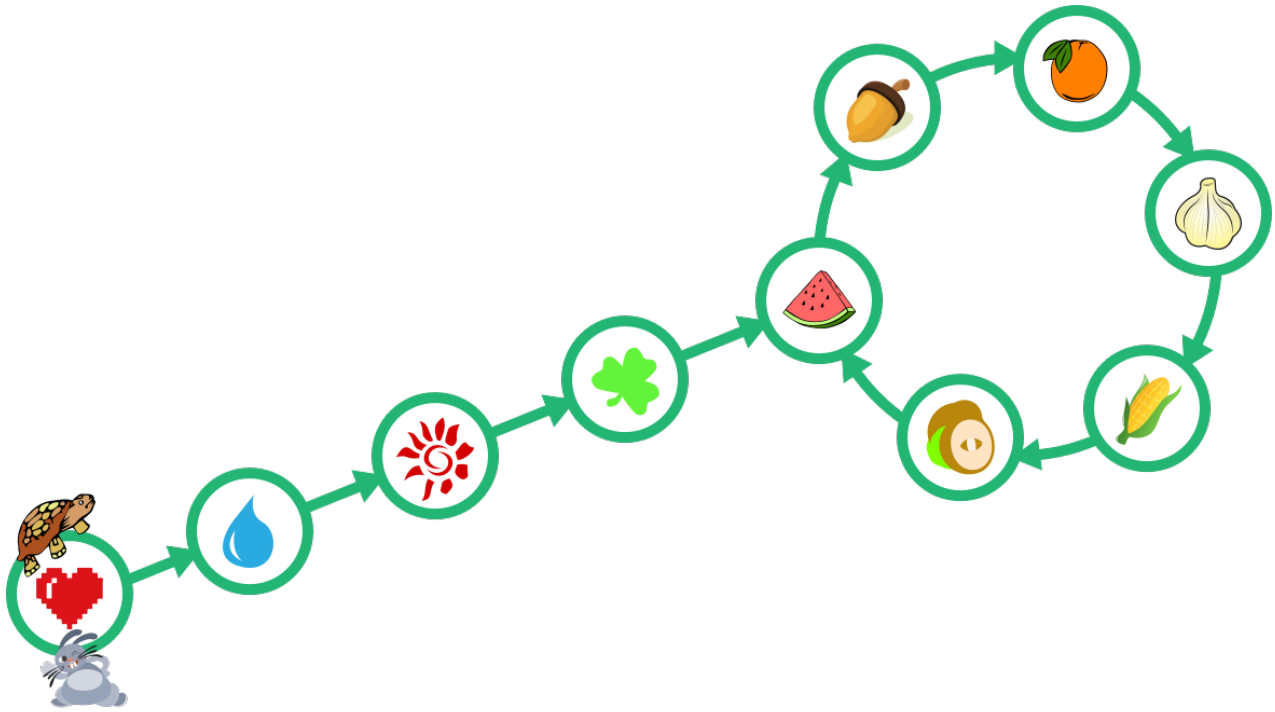
Country of Original Author

Canada

Tortoise and Hare

Story

A tortoise and a hare follow the arrows in the diagram shown.



They both start at the same time at the circle labelled with a heart. The tortoise moves from one circle to the next in two minutes. The hare moves from one circle to the next in one minute.

Question

Where do the tortoise and hare meet for the first time after they begin moving?

(A) 

(B) 

(C) 

(D) 

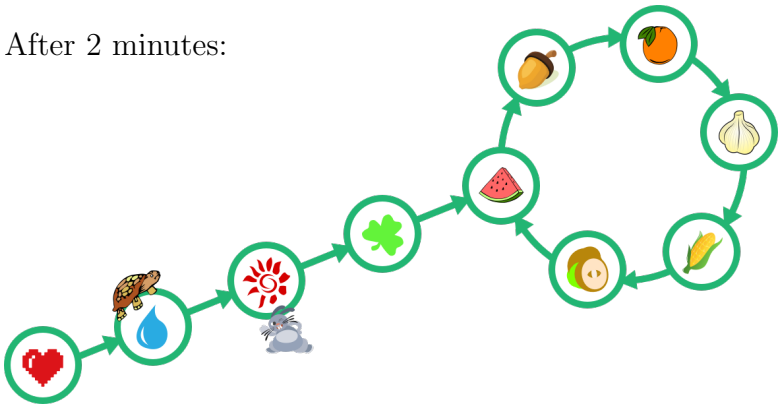
Answer

(C) 

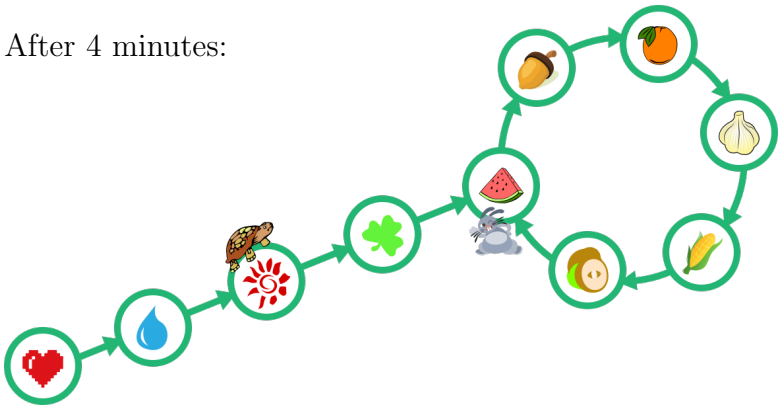
Explanation of Answer

Because of the speed of the tortoise, the tortoise and hare will only both be at a circle after an even number of minutes. The figures below show the locations of the tortoise and the hare after every 2 minutes.

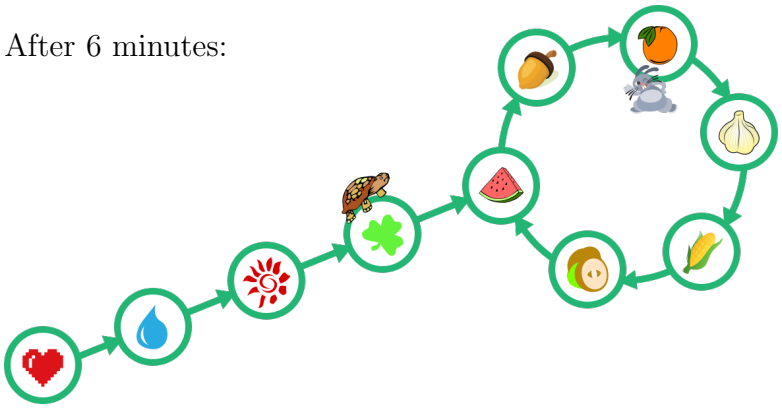
- After 2 minutes:



- After 4 minutes:

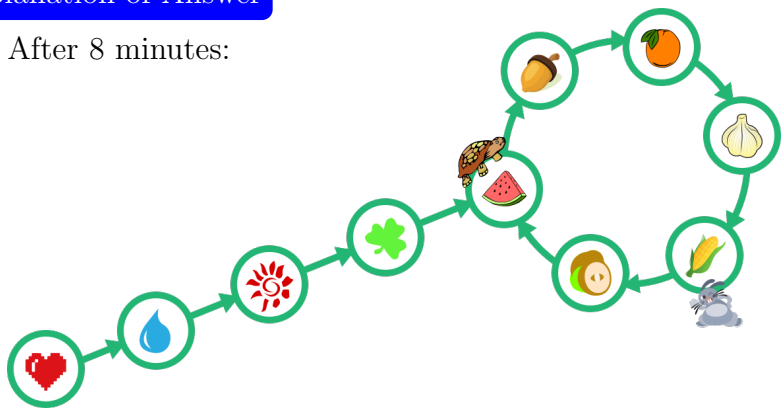


- After 6 minutes:

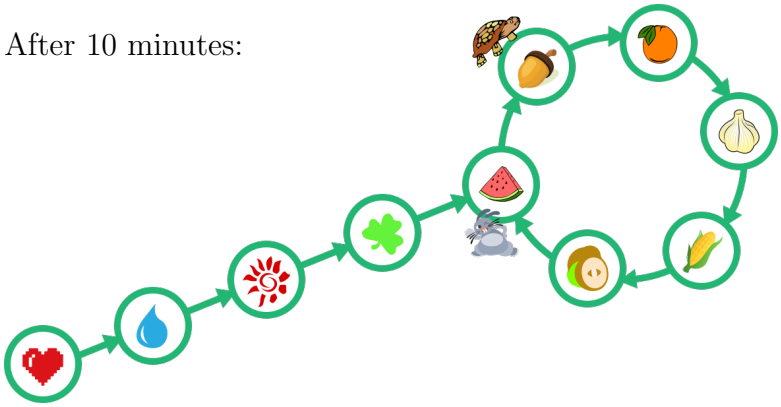


Explanation of Answer

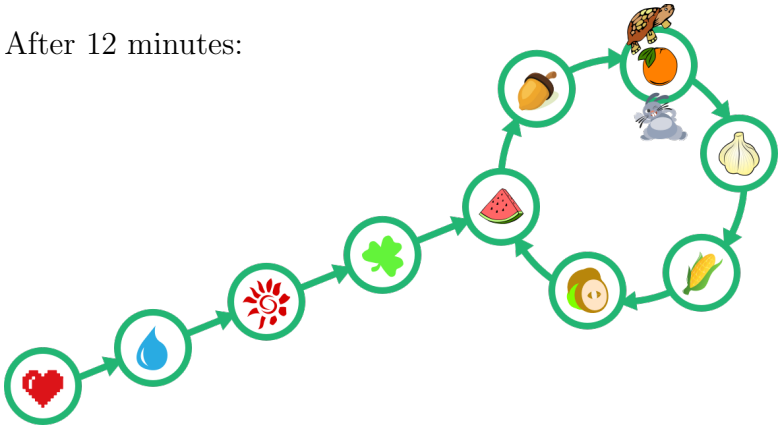
- After 8 minutes:



- After 10 minutes:



- After 12 minutes:



We see that the tortoise and hare meet for the first time at the orange (Option C).

Connections to Computer Science

The task is about traversing a specific data structure, called a *directed graph*, which in this case contains a *cycle*.

A *linked list* is a data structure used to store items that have to be connected in sequence. These items are also called *elements* or *nodes*. Examples of using linked lists include the steps in a recipe, the landmarks on the way from one location to another, or even the operations in an algorithm. To maintain their connection, each item knows the "address" of the next item. More formally, each node maintains a *pointer* to the next node. Usually, a linked list is *linear* — if we begin at the first node and follow the pointers, we will reach the end without visiting any node twice.

If it so happens that we visit a node twice, then we are caught in a *cycle*. How do computers detect the existence of cycles? One ingenious approach, attributed to the American computer scientist Robert W. Floyd, is the *tortoise-and-hare algorithm* (in reference to the Aesop fable). As demonstrated in this task, it features two "pointers" that move through the linked list at different speeds: one moves twice as fast as the other. If they meet, then we can conclude that there is a cycle. Otherwise, the linked list is linear.

Cycle detection is an important task in computer science. For instance, it can be used to check if our code is repeating a sequence of tasks endlessly (*infinite loop*), which prevents our program from stopping. A more advanced application is related to the analysis of the quality of *random number generators*, especially those that are utilized in the encryption or protection of sensitive data. They usually have pre-cycle that does not repeat and then they cycle. The part of the path presented as a straight line corresponds to the pre-cycle, the round part corresponds to the cycle. Having longer cycle lengths is a key characteristic that makes secure algorithms stronger and more difficult to crack.

Country of Original Author

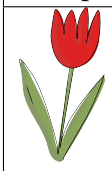
Philippines

Part C

Spring Blossom

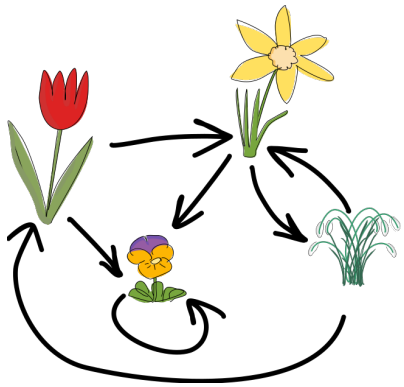
Story

Janine is planting a row of seven flowers in her flowerbed. She has the following types of flowers.

Tulip	Daffodil	Pansy	Snowdrop
			

She plants her flowers in her flowerbed according to the following plan.

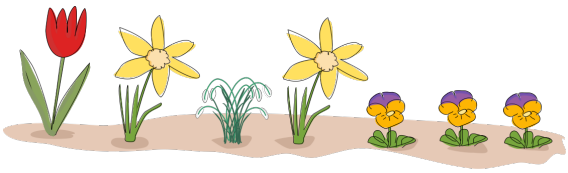
1. The flowers must be planted in a row from left to right.
2. Any flower can be planted in the leftmost spot.
3. Two flowers can be planted next to each other only if the diagram shows an arrow from the flower being planted first to the flower being planted next.

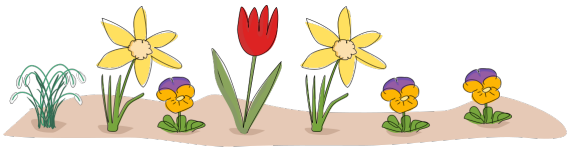


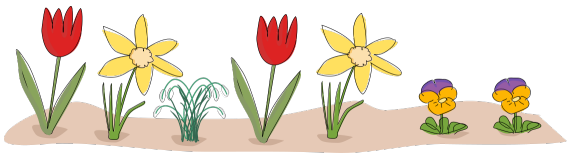
For example, Janine can plant a tulip and then a daffodil to its right because there is an arrow from the tulip to the daffodil. However, she cannot plant a daffodil and then a tulip to its right because there is no arrow from the daffodil to the tulip.

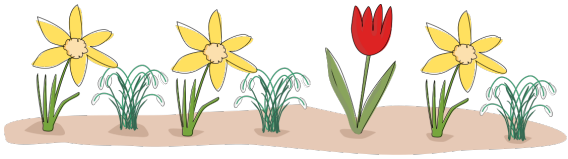
Question

Which flowerbed **could not** possibly be Janine's?

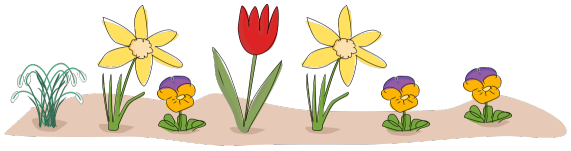
(A) 

(B) 

(C) 

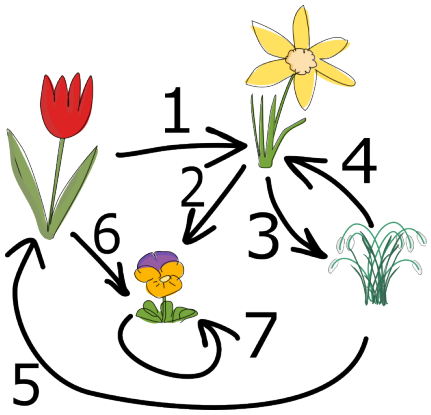
(D) 

Answer

(B) 

Explanation of Answer

Suppose we number the arrows in Janine's diagram as follows:



The flowerbed in Option A can be planted by starting with the tulip and then following arrows 1, 3, 4, 2, 7, and 7.

The flowerbed in Option C can be planted by starting with the tulip and then following arrows 1, 3, 5, 1, 2, and 7.

The flowerbed in Option D can be planted by starting with the daffodil and then following arrows 3, 4, 3, 5, 1, and 3.

For the flowerbed in Option B, you can start with the snowdrop and then follow arrows 4 and 2. However, there is no arrow leading from the pansy to the tulip and so this entire flowerbed cannot be planted.

In fact, once Janine plants a pansy, she has to fill all the remaining spots with pansies because there is no arrow leading away from the pansy to any other flower.

Connections to Computer Science

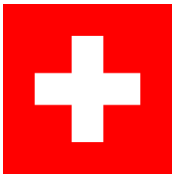
This task is based on the concept of a *finite state machine*. Each flower is a *state* and there are *transitions* from one flower to another flower indicated by the arrows in the diagram. A transition from flower A to flower B indicates that flower A can have flower B planted to its right.

In finite state machines, there is usually a *start state* and a set of *final states*, but for this problem, we can start in any state (i.e., plant any flower first) and we do not need to stop at any particular flower.

One use of finite state machines in computer science is in *text pattern matching*. For example, a finite state machine can be used to check the validity of the format of an e-mail address: specifically, we can describe a finite state machine that checks if the input is composed of letters and numbers, followed by an @ character, followed by a valid domain, such as `somewhere.com`.

Country of Original Author

Switzerland



Hide and Seek

Story

Four of Gosia's friends are hiding in a park. No two friends are hiding in the same spot. Gosia knows the following information about who is hiding where:

- Beka or Nissa is hiding behind the trees.
- Rona or Pasha is hiding behind the fountain.
- Beka or Nissa is hiding behind the bench.
- Rona or Beka is hiding behind the lamppost.

Question

What is Rona hiding behind?

(A) The trees

(B) The fountain

(C) The bench

(D) The lamppost

Answer

(D) The lamppost

Explanation of Answer

Pasha must be behind the fountain since we are told that someone else is hiding in the other three spots. Therefore, Rona is not behind the fountain. The only other option for Rona is the lamppost.

Connections to Computer Science

This task involves *logical reasoning*. Specifically, reasoning about the four places using logical expressions involving *AND*, *OR*, or *NOT*. That is, each piece of information involves using an OR expression to state that one of those friends is hiding behind each particular object. The entire list of four pieces of information can be thought to be connected together with an AND: one friend is hiding behind the trees AND one friend is hiding behind the fountain AND so on. We are also given that two friends are NOT hiding behind the same object.

Logic plays a key role in computer science in a huge variety of areas: databases, programming languages, artificial intelligence, hardware and software design and verification, etc. Logic is one of the fundamental concepts that underlies *computational thinking*: being able to take information, model it logically, and make *logical conclusions* from that model.

Country of Original Author

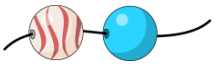
Uruguay

Beach Necklaces

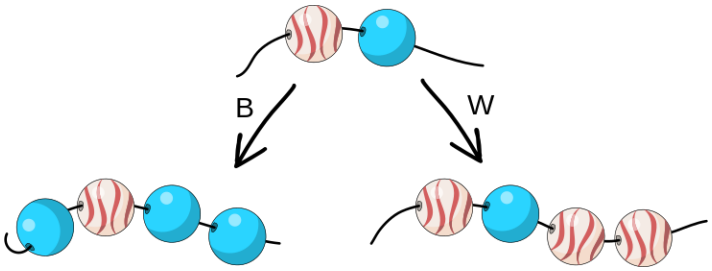
Story

Bashir makes necklaces using wavy beads and blue beads. He always makes them as follows.

1. Place one wavy bead and one blue bead on a string with the wavy bead to the left of the blue bead.



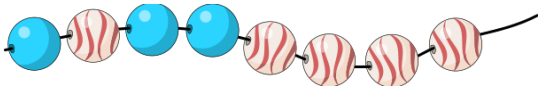
2. Do one of the following two actions.
   - *Action B*: Add a blue bead to both ends of the string.
   - *Action W*: Add two wavy beads to the rightmost end of the string.

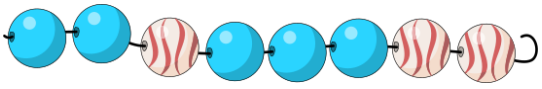


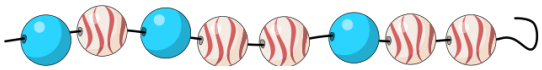
3. Repeat step 2 until the necklace is complete.

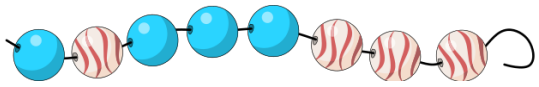
Question

Which necklace below **cannot** be made by Bashir?

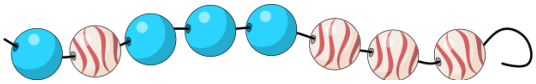
(A) 

(B) 

(C) 

(D) 

Answer

(D) 

Explanation of Answer

Each necklace begins with one wavy bead and one blue bead to its right.

The necklace in Option A can be made by taking *Action B*, then *Action W* and then *Action W* again.

The necklace in Option B can be made by taking *Action B*, then *Action B* again and then *Action W*.

The necklace in Option C can be made by taking *Action W*, then *Action B* and then *Action W*.

Bashir cannot have made the necklace in Option D. One way to see this is to notice that there is only one place on that necklace with a wavy bead to the left of a blue bead. If Bashir had made the necklace in Option D, this must have been where he started. These two beads are surrounded by blue beads which means the only possible first action taken by Bashir is *Action B*. At that point, *Action B* would produce another blue bead on the left side of the necklace and *Action W* would produce a wavy bead where there is a blue bead. Either way, Bashir cannot make the necklace in Option D.

Another clever way to rule out Option D is to notice that Bashir always starts with one blue bead and when he adds more blue beads, he always does so two at a time. This means that there must be an odd number of blue beads. For the same reason, there must always been an odd number of wavy beads. In Option D, there are four blue beads and four wavy beads. Since four is an even number, this is impossible.

Connections to Computer Science

In this task, beads could only be added or removed to the ends of the necklace string. It was not possible to add or remove a bead to the middle directly.

The necklace is similar to a *data structure* used in computer science called a *double-ended queue* or *deque*.

One common use of a deque is to store web browser's history: there are "back" and "forward" buttons in most web browsers that behave in a similar way to the ends of the necklace. Other applications of deques include scheduling print jobs and verifying the validity of math expressions such as $((1 + 2) * (3 + 4))$. Checking for matching parentheses can be done in a way that is quite similar to how the validity of the necklaces was verified.

Country of Original Author

Slovakia

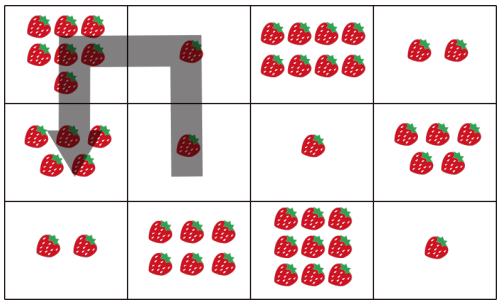
Strawberry Patch

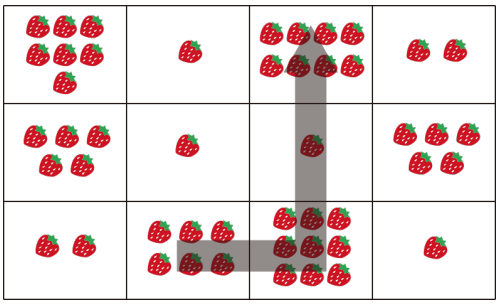
Story

Every day, a beaver goes to a strawberry patch for dessert. It starts eating strawberries from one of the twelve fields in the patch. Then it moves either north (↑), south (↓), east (→), or west (←) to a neighbouring field exactly three times.

This behaviour ensures that the beaver eats strawberries from exactly four fields and leaves the rest of the strawberries for others to enjoy.

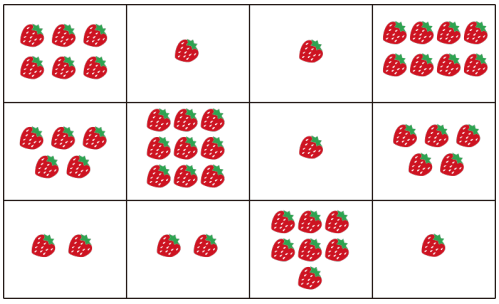
For example, in the strawberry patch shown, a beaver could follow the path shown on the left and eat $1 + 1 + 7 + 5 = 14$ strawberries or follow the path shown on the right and eat $6 + 9 + 1 + 8 = 24$ strawberries.





Question

What is the maximum number of strawberries the beaver could eat from the following patch?



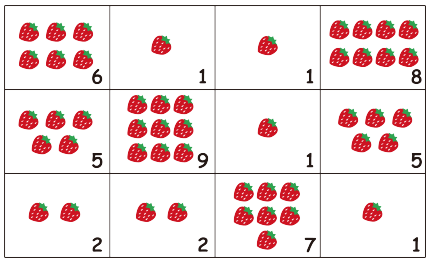
(A) 21 (B) 22 (C) 23 (D) 24

Answer

(C) 23

Explanation of Answer

Here is the strawberry patch with numbers showing the amount of strawberries in each field.



Let's call the six fields with 1 or 2 strawberries *bad* and the other fields which all have at least 5 strawberries *good*.

Because of how the beaver moves, it is impossible for it to visit four good fields.

There are not too many ways in which the beaver can visit three good fields. The total strawberries that the beaver will eat in each of these possibilities is shown below.

$$\begin{aligned} 6+5+9+1 &= 21 \\ 6+5+9+2 &= 22 \\ 5+9+2+7 &= 23 \\ 5+9+1+7 &= 22 \\ 5+9+1+5 &= 20 \\ 9+1+5+8 &= 23 \\ 7+1+5+8 &= 21 \end{aligned}$$

For each of these possibilities, the beaver can eat the number of strawberries in more than one order, but the order does not affect the total number of strawberries it eats.

If the beaver visits only two good fields, then it can eat at most $9+8=17$ strawberries from these good fields, and at most $2+2=4$ strawberries from two bad fields. Therefore, the beaver can eat at most $17+4=21$ strawberries if it visits two good fields. Note that there isn't actually a way to visit the fields with 9, 8, 2 and 2 strawberries but what matters is that the beaver certainly cannot eat more than 21 strawberries by visiting only two good fields. It is clear that for this particular strawberry patch, it also cannot do any better by visiting only one good field or no good fields.

This argument shows that the beaver can eat at most 23 strawberries. One way that it can actually do this is by starting at the field with 9 strawberries and then moving east, east and north. This path corresponds to the beaver eating $9+1+5+8=23$ strawberries which is one of the possibilities listed above.

Connections to Computer Science

In computer science, many problems aim to find the *maximum* or *minimum value*: doing the most work, spending the least amount of money, or, as in this task, eating the most strawberries. In these problems, there are many different ways to do the work, to spend money, or to eat strawberries. Within these ways, the challenge is to find those with maximum or minimum values; they are called the *optimal solutions*.

There are many methods to find the optimal solution. One method is to list all the possible solutions, calculate their values, and choose the one(s) with the best value. This method is called *brute force search* or *exhaustive search*. However, this method may take tremendous computing time if there are many options.

Therefore, it is helpful to analyze the problem and find constraints which can reduce the number of options and find the optimal solution faster. For many problems, computer scientists know methods to find optimal solutions efficiently. In this task, dividing the fields into "good" and "bad" fields helps find an optimal solution.

However, for other problems, such as the *travelling salesperson problem*, there is no known method to find an optimal solution efficiently.

Country of Original Author

Taiwan