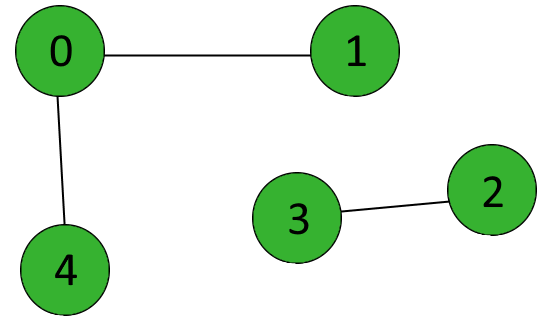
**Flight Connections**

In an effort to connect all airports with minimal costs, an airport authority needs to identify the minimum number of new flight connections so that it is possible to reach all airports from every other airport directly or indirectly.

Airports and flight connections are represented by a boolean matrix. Airports are represented by numbers 0, 1, 2, and so on. So, if airport '0' is connected via a flight with airport '1' then elements [0][1] and [1][0] will have the value true, or false if there is no flight connectivity. All flights are bi-directional.

The function getMinimumConnections accepts a two-dimensional array matrix. The function should return the minimum number of flight connections that need to be added so that all airports are reachable from every other airport, directly or indirectly.

For example, the following code should print 1, as a minimum of 1 additional flight connection will connect all airports (directly or indirectly).



std::vector<std::vector<bool>> matrix {

{false, true, false, false, true},

{true, false, false, false, false},

{false, false, false, true, false},

{false, false, true, false, false},

{true, false, false, false, false}

};

std::cout << getMinimumConnections(matrix) << std::endl; // should print 1

**Starting Coding:**

#include <iostream>

#include <vector>

int getMinimumConnections(const std::vector<std::vector<bool>>& matrix)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

std::vector<std::vector<bool>> matrix {

{false, true, false, false, true},

{true, false, false, false, false},

{false, false, false, true, false},

{false, false, true, false, false},

{true, false, false, false, false}

};

std::cout << getMinimumConnections(matrix) << std::endl; // should print 1

}

#endif

**Test case:**

Example case:

All airports are pre-connected:

Some airports are pre-connected:

Performance test on large number of airports and flight connections:

## [Paper Strip](https://app.testdome.com/questions/84256)

You are given two paper strips. On each strip, all numbers in the range [1, 2, ... N] are written in random order and no number is repeated. Cut the original paper strip into several pieces and rearrange those pieces to form the desired sequence.

Write a function that, efficiently with respect to time used, returns the minimum number of cut pieces needed to perform the described operation.

For example, the following code should display 3 because the pieces used should be (1), (4, 3), and (2):

std::vector<int> original = { 1, 4, 3, 2 };

std::vector<int> desired = { 1, 2, 4, 3 };

std::cout << PaperStrip::minPieces(original, desired) << std::endl;

**Starting Coding:**

#include <string>

#include <iostream>

#include <stdexcept>

#include <vector>

int minPieces(const std::vector<int>& original, const std::vector<int>& desired)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

std::vector<int> original = { 1, 4, 3, 2 };

std::vector<int> desired = { 1, 2, 4, 3 };

std::cout << minPieces(original, desired) << std::endl;

}

#endif

**Test case:**

Example case:

10 elements:

Performance test with a large paper strip:

## [Ship](https://app.testdome.com/questions/35912)

A ship has containerCount containers indexed as 0, 1, ..., containerCount - 1. Each container must be filled in the constructor of the Ship using the fillContainer function, which accepts the container's index and returns the container's cargo.

The code below works, but the server has enough memory only for a ship with few containers. Change the code so that the server has enough memory even for a ship with many containers.

**Starting Coding:**

#include <iostream>

#include <unordered\_map>

#include <functional>

class Ship

{

public:

Ship(int containerCount, std::function<int(int)> fillContainer)

{

for (int i = 0; i < containerCount; i++)

{

this->containers.insert({ i, new int(fillContainer(i)) });

}

}

int peekContainer(int containerIndex)

{

return \*this->containers.at(containerIndex);

}

private:

std::unordered\_map<int, int\*> containers;

};

#ifndef RunTests

int main()

{

Ship ship(10, [](int containerIndex) { return containerIndex; });

for (int i = 0; i < 10; i++) {

std::cout << "Container: " << i << ", cargo: " << ship.peekContainer(i) << "\n";

}

}

#endif

**Test case:**

Ship with a medium number of containers: Memory limit exceeded

Ship with a large number of containers: Memory limit exceeded

## [Date Transform](https://app.testdome.com/questions/80638)

An application requires different date formats to be converted into one common date format.

Implement the function *transformDateFormat*  which accepts a list of dates as strings, and returns a new list of strings representing dates in the format of YYYYMMDD. All incoming dates will be valid dates, but only those in one of the following formats: YYYY/MM/DD, DD/MM/YYYY, and MM-DD-YYYY should be included in the returned list, where YYYY, MM, and DD are numbers representing year, month, and day, respectively.

For example, *transformDateFormat(std::vector {"2010/02/20", "19/12/2016", "11-18-2012", "20130720"}) should return the list {"20100220", "20161219", "20121118"}.*

**Starting Coding:**

#include <stdexcept>

#include <iostream>

#include <vector>

std::vector<std::string> transformDateFormat(const std::vector<std::string>& dates)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

std::vector<std::string> dates = {"2010/02/20", "19/12/2016", "11-18-2012", "20130720"};

std::vector<std::string> reformattedDates = transformDateFormat(dates);

for (auto const& reformattedDate : reformattedDates)

{

std::cout << reformattedDate << std::endl;

}

}

#endif

**Test case:**

Example case: Exception

Lists containing a single format: Exception

Lists containing multiple formats: Exception

Lists containing multiple formats, some formats should be ignored: Exception

## [Platformer](https://app.testdome.com/questions/35833)

A character in a platformer game is standing on a single row of floor tiles numbered 0 to N, at position X.

When the character moves, the tile at the previous position disappears. The character can only move left and right, and always jumps over one tile, and any holes. The character will not move if there are no tiles left to move to (you do not need to implement this in the code).

Implement a class that models this behavior and can report the character's position efficiently with respect to time used.

For example, Platformer(6, 3) creates a row of 6 tiles (numbered 0 to 5) and a character positioned on tile 3 {0 1 2 [3] 4 5}. A call to jumpLeft() moves the character two tiles to the left and the tile at position 3 disappears {0 [1] 2 4 5}. A subsequent call to jumpRight() moves the character two tiles to the right and the tile at position 1 disappears, skipping tiles that have disappeared {0 2 [4] 5}. Calling position() method at this point should return 4.

**Starting Coding:**

#include <stdexcept>

#include <iostream>

class Platformer

{

public:

Platformer(int n, int position)

{

throw std::logic\_error("Waiting to be implemented");

}

void jumpLeft()

{

throw std::logic\_error("Waiting to be implemented");

}

void jumpRight()

{

throw std::logic\_error("Waiting to be implemented");

}

int position()

{

throw std::logic\_error("Waiting to be implemented");

}

};

#ifndef RunTests

int main()

{

Platformer platformer(6, 3);

std::cout << platformer.position() << std::endl; // should print 3

platformer.jumpLeft();

std::cout << platformer.position() << std::endl; // should print 1

platformer.jumpRight();

std::cout << platformer.position() << std::endl; // should print 4

}

#endif

**Test case:**

Example case: Exception

10 tiles: Exception

Performance test with a large number of tiles: Exception

## [Function Cache](https://app.testdome.com/questions/38165)

The caching feature of the FunctionCache class does not work in the implementation provided.

Find and fix the bug.

**Starting Coding:**

#include <iostream>

#include <unordered\_map>

#include <memory>

#include <functional>

class FunctionCache

{

class Parameters

{

public:

Parameters(int first, int second)

{

this->first = first;

this->second = second;

}

int first;

int second;

};

public:

FunctionCache(std::function<int(int, int)> function): function(function) {}

int calculate(int first, int second)

{

auto args = make\_shared<Parameters>(first, second);

auto it = calculations.find(args);

if (it != calculations.end())

return it->second;

int calculation = function(first, second);

calculations[args] = calculation;

return calculation;

}

private:

std::unordered\_map<std::shared\_ptr<Parameters>, int> calculations;

std::function<int(int, int)> function;

};

#ifndef RunTests

int modulo(int a, int b)

{

std::cout << "Function modulo has been called.\n";

return a % b;

}

int main()

{

FunctionCache cache(modulo);

// Function modulo should be called.

std::cout << cache.calculate(5, 2) << std::endl;

// Function modulo should be called.

std::cout << cache.calculate(7, 4) << std::endl;

// Function modulo shouldn't be called because we have already made a call with arguments (5, 2)!

std::cout << cache.calculate(5, 2) << std::endl;

}

#endif

**Test case:**

Example case:

100 calculations:

Performance test with a large number of calculations:

## [Book Sale](https://app.testdome.com/questions/37640)

Write a method that, efficiently with respect to time used, finds the n-th lowest selling book in the list. Each element of the sales array represents a single sale of a book with that book's id. The n-th lowest selling book is the book that has more sales than n-1 books. Assume that book sales counts are unique.

For example, nthLowestSelling({ 5, 4, 3, 2, 1, 5, 4, 3, 2, 5, 4, 3, 5, 4, 5 }, 2) should return 2. In the array, the book with the id 1 was sold once, id 2 twice, id 3 three times, id 4 four times, and id 5 five times, making the book with the id 1 the lowest selling book in the array and id 2 the second lowest selling book.

**Starting Coding:**

#include <vector>

#include <stdexcept>

#include <iostream>

int nthLowestSelling(const std::vector<int>& sales, int n)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

std::vector<int> input;

input.push\_back(5);

input.push\_back(4);

input.push\_back(3);

input.push\_back(2);

input.push\_back(1);

input.push\_back(5);

input.push\_back(4);

input.push\_back(3);

input.push\_back(2);

input.push\_back(5);

input.push\_back(4);

input.push\_back(3);

input.push\_back(5);

input.push\_back(4);

input.push\_back(5);

int x = nthLowestSelling(input, 2);

std::cout << x;

}

#endif

**Test case:**

Example case:

Additional list elements:

Performance test with a large number of elements:

## [Stack to Vector](https://app.testdome.com/questions/55005)

Consider the following function:

Select the computational complexity of the *stackToVector* function if the *reverseItems* argument is false:

O(1)

O(n)

O(n\*log(n))

O(n^2)

Select the computational complexity of the *stackToVector* function if the *reverseItems* argument is true:

O(1)

O(n)

O(n\*log(n))

O(n^2)

Graphical user interface, text, application, email

Description automatically generated

## [Window Manager](https://app.testdome.com/questions/35966)

The window manager handles windows in a graphical user interface. Windows are displayed on top of each other, the top one being completely visible. When new window is opened, it is placed on top of all the other windows.

Implement the class WindowManager, that contains the following methods that should be efficient with respect to time used:

* Open - Opens a window with the given name and puts it on top of all other windows.
* Close - Closes a window with the given name. Every open window will have a different name.
* GetTopWindow - Returns the name of the top window.

For example, if Calculator, Browser and Player are opened (in that order) and after that Browser is closed, Player is the top window. See the image for details.

A picture containing graphical user interface

Description automatically generated

**Starting Coding:**

#include <iostream>

#include <string>

class WindowManager

{

public:

void open(const std::string windowName)

{

}

void close(const std::string windowName)

{

}

std::string getTopWindow()

{

return NULL;

}

};

#ifndef RunTests

int main()

{

WindowManager wm;

wm.open("Calculator");

wm.open("Browser");

wm.open("Player");

wm.close("Browser");

std::cout << wm.getTopWindow();

}

#endif

**Test case:**

Example case: Exception

Small number of windows: Exception

Performance test on a large number of windows: Exception

## [Archive](https://app.testdome.com/questions/35481)

A customer has complained that their archive application still occupies a big chunk of memory although they have removed all the records from it.

Fix the problem.

**Starting Coding:**

#include <iostream>

#include <unordered\_map>

#include <string>

class Archive

{

class Record

{

public:

Record(std::string title, std::string content)

{

this->title = title;

this->content = content;

}

std::string title;

std::string content;

};

public:

void add(int id, std::string title, std::string content)

{

records[id] = new Record(title, content);

}

void remove(int id, std::string& title, std::string& content)

{

std::unordered\_map<int, Record\*>::iterator it = records.find(id);

Record\* doc = it->second;

title = doc->title;

content = doc->content;

records.erase(it);

}

void clear()

{

records.clear();

}

private:

std::unordered\_map<int, Record\*> records;

};

#ifndef RunTests

int main()

{

Archive archive;

archive.add(123456, "Hamlet", "Hamlet, Prince of Denmark.");

archive.add(123457, "Othello", "Othello, the Moore of Venice.");

std::string title, content;

archive.remove(123456, title, content);

std::cout << title << '\n';

std::cout << content;

archive.clear();

}

#endif

**Test case:**

Add and remove: Memory limit exceeded

Add and clear: Memory limit exceeded

Add and destroy: Memory limit exceeded

## [Unique Product](https://app.testdome.com/questions/59537)

Write a method that, **efficiently** with respect to time used, finds the first product in a vector that occurs only once in that vector. If there are no unique products in the vector, an empty string should be returned.

For example, for vector products containing { "Apple", "Computer", "Apple", "Bag" }, firstUniqueProduct(products) should return "Computer".

#include <stdexcept>

#include <iostream>

#include <string>

#include <vector>

std::string firstUniqueProduct(const std::vector<std::string>& products)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

std::vector<std::string> products = { "Apple", "Computer", "Apple", "Bag" };

std::cout << firstUniqueProduct(products);

}

#endif

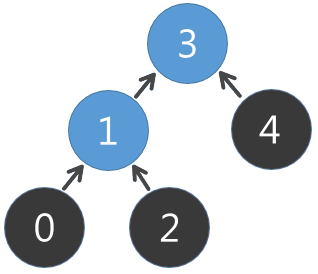
**Internal Nodes**

A tree is an abstract data structure consisting of nodes. Each node has only one parent node and zero or more child nodes. Each tree has one special node, called a root, which has no parent node. A node is called an internal node if it has one or more children.

A tree can be represented by an array P where P[i] is the parent of node i. For the root node r, P[r] equals -1.

Write a function that, **efficiently** with respect to time used, counts the number of internal nodes in a given tree.

For example, the tree represented by the array {1, 3, 1, -1, 3} has 5 nodes, 0 to 4, of which 2 nodes are internal nodes (only nodes 1 and 3 have children).



#include <vector>

#include <stdexcept>

#include <iostream>

int countInternalNodes (const std::vector<int>& tree)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

std::vector<int> tree;

tree.push\_back(1);

tree.push\_back(3);

tree.push\_back(1);

tree.push\_back(-1);

tree.push\_back(3);

std::cout << countInternalNodes(tree);

}

#endif

## Fire Dragon

Scientists have discovered a species of fire-breathing dragons. DNA analysis of the dragon reveals that it is a reptile evolved from a common ancestor of crocodile, hundreds of millions of years ago. Even though they're related, the different reptile species cannot cross-breed.

Researchers would like to develop a lifecycle model of this rare species, in order to better study them. Complete the implementation below so that:

* The *FireDragon* species implements the *Reptile* class.
* When a *ReptileEgg* hatches, a new reptile will be created of the same species that laid the egg.
* An *std::logic\_error* is thrown if a *ReptileEgg* tries to hatch more than once.

#include <stdexcept>

#include <functional>

class ReptileEgg;

class Reptile

{

public:

virtual ~Reptile() {};

virtual ReptileEgg\* lay() = 0;

};

class ReptileEgg

{

public:

ReptileEgg(std::function<Reptile\* ()> createReptile)

{

throw std::logic\_error("Waiting to be implemented");

}

Reptile\* hatch()

{

throw std::logic\_error("Waiting to be implemented");

}

};

class FireDragon : public Reptile

{

public:

FireDragon()

{

}

ReptileEgg\* lay()

{

throw std::logic\_error("Waiting to be implemented");

}

};

#ifndef RunTests

int main()

{

Reptile\* fireDragon = new FireDragon();

ReptileEgg\* egg = fireDragon->lay();

Reptile\* childDragon = egg->hatch();

}

#endif

## Battery

A *Battery*'s charge is represented as a float and is empty (0.0) by default.

Implement the *charge* function so that the battery's charge is increased by the supplied value.

For example, if the *Battery*'s current charge is 0.0, then *charge(battery, 1.0f)* should increase the battery's charge field by *1.0f*.

#include <algorithm>

#include <iostream>

struct Battery

{

float charge = 0.0f;

};

void charge(Battery\* battery, float charge)

{

throw std::logic\_error("Waiting to be implemented");

}

#ifndef RunTests

int main()

{

Battery battery;

charge(&battery, 1.0f);

std::cout << battery.charge << std::endl;

}

#endif