**Содержание**

[Введение 3](#_Toc165718395)

[Список задач 4](#_Toc165718396)

[Описание решенных задач 6](#_Toc165718397)

[Список консультаций 37](#_Toc165718398)

[Заключение 38](#_Toc165718399)

[Список литературы 40](#_Toc165718400)

[Приложения 41](#_Toc165718401)

Введение

Учебная практика проходила в ФГБОУ ВО «Вятский государственный университет», на кафедре прикладной математики и информатики факультета компьютерных и физико-математических наук с 12.02.2024 г. по 05.05.2024 г., количество недель: 12. Установочная конференция состоялась 12.02.2024 г.

Целью учебной практики является закрепление и углубление теоретической подготовки, приобретение практических навыков и компетенций в сфере профессиональной деятельности, в основном путём самостоятельного решения предусмотренных программой задач.

Форма проведения учебной практики: компьютерная (практикум по решению задач на ЭВМ).

В ходе прохождения практики необходимо реализовать компьютерные программы, каждую на двух или трех языках программирования – С++, C# и Python. Каждая задача сначала должна быть принята системой (http://acm.timus.ru) на обоих языках, а потом нужно защитить предложенный алгоритм решения задачи и его программную реализацию в часы консультаций.

Для получения доступа к архиву задач была пройдена регистрация на сайте http://acm.timus.ru.

Для реализации задач использовалась среда программирования Microsoft Visual Studio.

# Список задач

Индивидуальный список задач из архива задач с сайта http://acm.timus.ru, полученный на установочной конференции:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | ID | Название | Тема | Сложность |
|  | 1002 | Телефонные номера | Графы | 223 |
|  | 1037 | Управление памятью | Структуры данных | 1168 |
|  | 1072 | Маршрутизация | Графы | 493 |
|  | 1107 | Складская задача | Хитрая задача | 419 |
|  | 1142 | Отношения | Комбинаторика | 152 |
|  | 1177 | Сопоставление с шаблоном | Строковые алгоритмы | 1240 |
|  | 1212 | Морской бой | Геометрия | 486 |
|  | 1247 | Проверка последовательности | Задача для начинающих | 238 |
|  | 1282 | Дерево игры | Графы | 336 |
|  | 1317 | Град | Геометрия | 1078 |
|  | 1352 | Простые числа Мерсенна | Теория чисел | 130 |
|  | 1387 | Папа у Васи | Графы | 1392 |
|  | 1422 | Светлячки | Геометрия | 868 |
|  | 1457 | Теплотрасса | Задача для начинающих | 67 |
|  | 1492 | Папа у Васи 2 | Геометрия | 1990 |
|  | 1527 | Плохие дороги | Графы | 1159 |
|  | 1562 | Ананас | Геометрия | 410 |
|  | 1597 | Чернобыльский орел на крышке. Версия 2 | Поиск числа в последовательности | 2028 |
|  | 1632 | Лазеры | Геометрия | 4365 |
|  | 1667 | Квадратная страна 3 | Перебор с возвратом | 1980 |
|  | 1702 | Перекрёсток | Динамическое программирование | 2221 |
|  | 1737 | Мнемоника и палиндромы 3 | Палиндромы | 246 |
|  | 1772 | Лыжни для роботов | Структуры данных | 720 |
|  | 1807 | Патроны для Максима | Теория чисел | 1426 |
|  | 1842 | Локальные корни | Строковые алгоритмы | 6635 |
|  | 1877 | Велосипедные коды | Задача для начинающих | 19 |
|  | 1912 | Руины титанов: проходя сквозь стены | Графы | 1213 |
|  | 1947 | Меандр | Структуры данных | 1646 |
|  | 1982 | План электрификации | Графы | 131 |
|  | 2017 | Меньшее из зол | Графы | 1292 |
|  | 2052 | Физкультура | Жадный алгоритм | 1474 |
|  | 2087 | Паровозики | Графы | 4119 |
|  | 2122 | Хэмминг | Динамическое программирование | 4834 |
|  | 2157 | Прыжок с парашютом | Динамическое программирование | 1175 |

# Описание решенных задач

В ходе прохождения практики мною были решены следующие задачи: 1002, 1072, 1107, 1142, 1212, 1247, 1282, 1317, 1352, 1422, 1457, 1562, 1737, 1877, 1982 всего 15 задач. Каждая задача была реализована на языках программирования С++, C# и Python. На всех трех языках реализованы задачи: 1352, 1457, 1877 (3 задачи), на языках С++ и C# реализованы задачи: 1072, 1107, 1212, 1247, 1282, 1317, 1422, 1562, 1737, 1982 (10 задач), на языках С++ и Python реализованы задачи: 1002, 1142 (2 задачи).

Детально будет описана наиболее интересная с точки зрения решения и тестирования решения задача 1982.

1. ***1002. Телефонные номера***

Ограничение времени: 2.0 секунды  
Ограничение памяти: 64 МБ

В современном мире вы встречаетесь с огромным количеством телефонных номеров, которые со временем становятся всё длиннее и длиннее. И вам приходится запоминать эти номера. Одним из простых способов запоминания является сопоставление букв каждой цифре, как показано на следующем рисунке:

|  |
| --- |
| 1 ij 2 abc 3 def  4 gh 5 kl 6 mn  7 prs 8 tuv 9 wxy  0 oqz |

Таким образом, каждому слову или группе слов может быть сопоставлен уникальный номер, так что можно запоминать слова вместо телефонных номеров. Очевидно, есть особый шарм в том, чтобы найти простую взаимосвязь между словом, используемым для запоминания телефонного номера, и владельцем этого номера. Так, телефонный номер 941837296 вашего друга, играющего в шахматы, может быть прочитан как WHITEPAWN (белая пешка), а номер 2855304 Вашего любимого учителя может быть прочитан как BULLDOG (бульдог).

Напишите программу, находящую самую короткую последовательность слов (имеющую наименьшее количество слов), которая соответствует заданному номеру телефона и заданному списку слов. Соответствие описано на рисунке выше.

### Исходные данные

Ввод состоит из набора тестов. Первая строка каждого теста содержит номер телефона, к которому нужно подобрать мнемонику. Номер состоит не более чем из 100 цифр. Вторая строка содержит общее количество слов в словаре (максимум 50 000). Каждая из оставшихся строк содержит одно слово, состоящее не более чем из 50 строчных латинских букв. Общий размер ввода не превосходит 300 килобайт. Последняя строка ввода содержит число −1.

### Результат

Каждая строка вывода должна содержать кратчайшую последовательность слов, найденную вашей программой. Слова должны быть разделены одиночными пробелами. Если для входных данных нет решения, соответствующая строка вывода должна содержать текст “No solution.”. Если существует несколько решений, имеющих одинаковое количество слов, можете выбрать любое из них.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 7325189087  5  it  your  reality  real  our  4294967296  5  it  your  reality  real  our  -1 | reality our  No solution. |

**Метки:**нет

Начало формы

Конец формы

Сложность: 223

***Описание решения:*** Для решения задачи необходимо, чтобы программа выводила, кратчайшую последовательность слов, которая соответствует заданному номеру телефона и заданному списку слов. Если нет решения, необходимо вывести "No solution.".

Данное решение основывается на поиске совпадений между номером телефона и словами из списка. Для этого используются структуры данных, такие как map для хранения соответствия между буквами цифрами на кнопках телефона, векторы для хранения слов из списка и параметров алгоритма, матрица для хранения соответствия слов и позиций в номере телефона, а также очередь и стек для работы с BFS алгоритмом.

Основной идеей решения является использование алгоритма BFS (поиска в ширину) для поиска кратчайшего пути соответствия слов из списка номеру телефона. Алгоритм вычисляет расстояние и предыдущие узлы для каждой вершины графа, что позволяет найти кратчайший путь от начальной вершины (нулевой позиции в номере) до конечной (последней позиции). Результаты выводятся на экран в виде соответствующих слов из списка. В случае отсутствия решения выводится сообщение "No solution.". Суть решения задачи заключается в поиске слов из словаря, которые могут быть образованы из номера телефона. Для каждого слова из словаря проверяется, можно ли его сформировать из номера телефона.

Листинги программы приведены в Приложении 1.

1. ***1072. Маршрутизация***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

Имеется сеть из нескольких компьютеров, с настроенной по правилам TCP/IP маршрутизацией. Это означает, что:

1. Каждый компьютер имеет 1 или более сетевых интерфейсов;
2. Каждый интерфейс идентифицируется IP-адресом и маской подсети — это два 4-х байтных числа, разделённые точками через каждый байт, причём в двоичном представлении маска подсети выглядит следующим образом: сначала идёт *k* единиц, потом *m* нулей, *k* + *m* = 8\*4 = 32. (Например, 212.220.35.77 — IP-адрес, 255.255.255.128 — маска).
3. 2 компьютера относятся к одной подсети, если (IP1 AND NetMask1) = (IP2 AND NetMask2), где IPi и NetMaski — IP-адрес и маска i-го компьютера, AND — побитовое умножение.
4. Пакет между компьютерами, находящими в одной подсети передаётся непосредственно.
5. Пакет между компьютерами, находящимися в 2-х разных подсетях, проходит через компьютеры, имеющие интерфейсы, подключенные к нескольким подсетям, причём при переходе из подсети в подсеть компьютер, на котором осуществляется этот переход, должен иметь интерфейсы, относящиеся к обеим подсетям.

Задача состоит в том, чтобы найти кратчайший путь пакета между двумя указанными компьютерами.

### Исходные данные

В первой строке стоит число N — количество компьютеров в сети, далее идёт N секций, описывающих интерфейсы каждого компьютера. В первой строке секции стоит число K — количество интерфейсов этого компьютера, затем следуют K строк с описанием интерфейсов в виде пар IP-адресов и масок. В последней строке стоят номера двух компьютеров, между которыми надо найти путь.

Известно, что 2 ≤ N ≤ 90 и K ≤ 5.

### Результат

Если путь существует, выведите «Yes» и в следующей строке через пробел номера компьютеров, через которые проходит путь. Если такого пути не существует, выведите «No».

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 6  2  10.0.0.1 255.0.0.0  192.168.0.1 255.255.255.0  1  10.0.0.2 255.0.0.0  3  192.168.0.2 255.255.255.0  212.220.31.1 255.255.255.0  212.220.35.1 255.255.255.0  1  212.220.31.2 255.255.255.0  2  212.220.35.2 255.255.255.0  195.38.54.65 255.255.255.224  1  195.38.54.94 255.255.255.224  1 6 | Yes  1 3 5 6 |

**Метки:**графы

Начало формы

Конец формы

Сложность: 493

***Описание решения:*** Для решения задачи необходимо найти кратчайший путь пакета между двумя указанными компьютерами в сети, описанной с помощью IP-адресов, масок подсети и интерфейсов компьютеров. Необходимо вывести "Yes" и номера компьютеров, через которые проходит путь, если путь существует. Если пути не существует, вывести "No".

Данный код решает задачу по поиску кратчайшего пути в графе, представляющем сети, имеющие общие узлы. Используются следующие структуры данных. Двумерный вектор строк "inter". В нем каждый вектор содержит строки, представляющие IP-адреса сетей. Двумерный вектор целых чисел "A", представляющий граф сетей, где сети, имеющие общие узлы, соединены ребром. Вектор целых чисел "D", представляющий расстояние от начальной сети до остальных сетей. Вектор целых чисел "P" содержит предыдущую сеть в кратчайшем пути. Очередь "q" используется для реализации алгоритма BFS. Стек "path" нужен для восстановления пути.

Общая идея решения заключается в том, что сначала мы считываем количество сетей и их IP-адреса, далее мы строим граф, где каждая сеть представлена в виде вершины, и соединяем сети, имеющие общие узлы ребром. Затем выполняем обход в ширину от начальной сети до конечной, сохраняя информацию о предыдущей сети в пути. По завершении обхода восстанавливаем путь от конечной сети до начальной и выводим его, если он существует.

Принципиальных отличий кода на C# от кода на C++ нет, но есть разница в вводе данных. В C# считываются строки данных, с помощью метода Split() разбиваются на элементы (строковый массив), они затем переводятся в тип int. В C++ мы считываем по отдельности данные и сразу же ее сохраняем их в переменные.

Листинги программы приведены в Приложении 2.

1. ***1107. Складская задача***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

На складе есть N различных видов товаров. Виды товаров занумерованы числами 1…N. Работники склада сделали K разных наборов из этих товаров. Будем говорить, что два набора “похожи”, если один из них получается удалением одного товара из другого набора или заменой одного товара на другой.

Например, набор “1 2 3 4” похож на наборы “3 2 1”, “1 2 5 3 4”, “1 2 3 4 2” и “1 5 4 3”, но не похож на наборы “1 2”, “1 1 2 2 3 4” и “4 5 3 6”.

Этот склад обслуживает M магазинов (0 < N < M < 101), посылая им наборы товаров. Любые два набора, посылаемые в один магазин, не должны быть похожи. Можно не посылать ни одного набора в какие-то из магазинов.

Напишите программу, которая определит, как распределить все K наборов по этим M магазинам.

### Исходные данные

Первая строка содержит числа N, K и M. Затем следуют K строк, описывающих каждый набор товаров, K ≤ 50 000. Каждая из этих строк начинается количеством товаров в этом наборе, затем записаны номера товаров. Количество товаров в каждом наборе больше 0 и меньше 101. Все числа в этих строках разделяются ровно одним пробелом.

### Результат

Первая строка вывода должна содержать слово YES, если решение существует, или NO в противном случае. Если ответ YES, запишите номера магазинов, в которые нужно послать наборы. Во второй строке запишите номер магазина, в который нужно послать первый набор, в третьей — куда послать второй набор — и так далее. Если существует более одного решения, можете вывести любое из них.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 8 20 12  5 1 3 5 6 4  5 1 3 5 6 3  4 5 6 3 3  4 5 6 3 4  4 4 6 5 8  4 7 7 7 7  3 7 7 7  2 2 2  3 2 2 7  3 1 2 3  3 1 2 4  10 1 2 3 4 5 6 7 8 7 6  10 8 7 6 5 4 3 2 1 2 1  20 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 3 5 7  5 4 6 4 6 4  5 6 4 6 4 6  6 6 6 6 6 6 6  3 6 6 6  1 1  1 2 | YES  2  1  9  1  6  2  4  5  3  7  8  5  4  8  7  9  1  1  2  3 |

**Метки:**хитрая задача

Сложность: 419

***Описание решения:*** Есть N видов товаров, сделанных из K наборов товаров, которые обслуживают M магазинов. Наборы не должны быть похожи в одном магазине. Для решения задачи необходимо определить, возможно ли распределить наборы по магазинам без нарушения условия схожести и вывести номера магазинов, в которые будут отправлены наборы.

Для решения этой задачи мы просто используем формулу , где x – искомый номер магазина. Определим остаток от деления вычисленного числа s на заданное число m и прибавим его к единице. Основная идея решения: считываем данные, используем формулу для решения задачи, выводим результат.

Разница реализаций между языками заключается в том, что в языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), затем они переводятся в тип int. А в C++ мы считываем данных сразу в переменные, не используя массив.

Листинги программы приведены в Приложении 3.

1. ***1142. Отношения***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

### Вступление

Рассмотрим множество сравнимых объектов. Между двумя объектами a и b существует одно из трёх отношений порядка:

*a* = *b*  
*a* < *b*  
*b* < *a*

Так как отношение '=' симметрично, оно не повторяется дважды.

Для 3 сравнимых объектов (a, b, c) существует 13 различных отношений порядка:

*a = b = c       a = b < c       c < a = b       a < b = c  
b = c < a       a = c < b       b < a = c       a < b < c  
a < c < b       b < a < c       b < c < a       c < a < b  
c < b < a*

### Задача

Дано число N, требуется найти количество различных отношений порядка между N объектами.

### Исходные данные

Состоит из множества целых чисел N в пределах от 2 до 10, каждое число расположено в отдельной строке. Ввод завершается числом −1.

### Результат

Для каждого N выведите количество различных отношений порядка.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 2  3  -1 | 3  13 |

**Метки:**нет

Сложность: 152

***Описание решения:*** Дано число N требуется найти количество различных отношений порядка между N объектами. В результате нужно вывести количество различных отношений порядка для каждого N.

Для решения задачи находим общее число отношений порядка для N объектов. Используемые структуры данных – двумерный массив "A" размером MAXN на MAXN для хранения результатов посчитанных отношений порядка для каждой совокупности переменных, и одномерный массив "ans" для хранения суммарных значений отношений для каждой совокупности переменных.

Основная идея решения заключается в использовании формулы для нахождения всех отношений порядка с помощью вложенных циклов. Сначала находим все отношения порядка для одной переменной, затем для двух, и так далее до N переменных. Для каждой совокупности переменных проходим по всем возможным различным отношениям порядка и суммируем их количество. После заполнения массива A и вычисления общего числа отношений для каждой совокупности переменных, программа принимает ввод от пользователя (количество переменных N) и выводит соответствующее количество отношений порядка. Процесс повторяется до ввода значения -1, после которого программа завершает свою работу.

Листинги программы приведены в Приложении 4.

1. ***1212. Морской бой***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

Однажды в школе на уроке информатики Петя и Вася решили сыграть в морской бой. Заканчивая расставлять свои корабли на листочке, Петя задумался: ему вдруг стало интересно, сколькими различными способами он сможет поставить свой последний K-палубный кораблик? Он попробовал быстро посчитать количество способов, но скоро сбился со счета. Тогда Петя посмотрел по сторонам и, неожиданно для себя, увидел стоящие повсюду компьютеры (в этом нет ничего удивительного, ведь, как мы помним, дело было на уроке информатики; но до этого момента Петя был так увлечен подготовкой к игре, что компьютеров не замечал). Подумав немного, он решил написать программу, которая может решить его задачу. Но у него ничего не получилось, поскольку он был двоечником — он отнюдь не первый раз в своей жизни играл в морской бой на уроке! Помогите Пете в его нелегком деле.

### Исходные данные

В первой строке содержатся три числа, разделённые пробелами: размер поля по вертикали N (1 ≤ N ≤ 30000), размер поля по горизонтали M (1 ≤ M ≤ 30000) и количество уже поставленных кораблей L (0 ≤ L ≤ 30). Далее следуют L строк с описанием расставленных кораблей. Описание каждого корабля состоит из записанных через пробел трёх чисел и буквы. Числа — это координаты левого верхнего угла корабля (левая верхняя клетка игрового поля имеет координаты (1, 1)) и количество палуб корабля. Буква же задает ориентацию корабля («V» — если корабль стоит вертикально и «H» — если горизонтально). В последней строке содержится число K — количество палуб корабля, который нужно поставить Пете.

Для тех, кто никогда не играл в морской бой, поясним, что i-палубный корабль — это прямоугольник размером i × 1 клеточку. Корабли могут иметь от одной до четырёх палуб. По правилам игры корабли не могут касаться друг друга рёбрами или вершинами.



### Результат

Выведите количество различных способов, которыми Петя может разместить на поле K-палубный корабль.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 4 4 2  1 2 2 V  3 1 2 H  2 | 4 |

**Метки:**нет

Сложность: 486

***Описание решения:*** Для того, чтобы решить задачу нужно определить количество различных способов, которыми можно поставить последний К-палубный кораблик на игровом поле, при этом нужно учитывать уже расставленные корабли.

Для решения используются структуры данных: "Ship" для хранения информации о каждом корабле (координаты, длина, тип), вектор "p" для хранения позиций кораблей по строкам и столбцам, а также функции для сортировки, удаления дубликатов и подсчета возможных расстановок.

Основная идея решения заключается в переборе всех строк и столбцов игрового поля для определения позиций кораблей, которые могут размещаться в этих строках и столбцах, и подсчете количества возможных расстановок с учетом заданного расстояния между кораблями. После считывания входных данных (размеров поля, количества кораблей, их параметров и расстояния), программа вызывает функцию перебора "solve()", которая возвращает общее количество возможных расстановок кораблей на заданном поле. В конце выполнения программа выводит полученное количество возможных расстановок.

При решении задачи на языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int. При реализации в C++ считываемые значения сразу записываются в переменные. Используется ввод cin. Также в коде C++ мы отключаем синхронизацию потоков ввода-вывода для оптимизации и ускорения выполнения программы "std::cin.tie(nullptr)->sync\_with\_stdio(false);".

Листинги программы приведены в Приложении 5.

1. ***1247. Проверка последовательности***

Ограничение времени: 0.5 секунды  
Ограничение памяти: 64 МБ

Дана последовательность целых чисел A1, A2, …, AS и целое положительное число N. Известно, что все элементы последовательности {Ai} удовлетворяют ограничению 0 ≤ Ai ≤ 100. Кроме того, известно, что сумма всех элементов последовательности равна S + N. Напишите программу, которая для заданной последовательности {Ai} и числа N дает ответ на вопрос: верно ли, что при любых 1 ≤ i ≤ j ≤ S выполняется неравенство

*Ai* + *Ai*+1 + … + *Aj* ≤ (*j* – *i* + 1) + *N* ?

### Исходные данные

В первой строке через пробел даны числа S и N, каждое из которых положительно и не превосходит 30000. Далее следует S строк, в которых, по одному числу в строке, записаны элементы последовательности {Ai}.

### Результат

Выведите слово «YES», если указанное выше неравенство выполняется при всех значениях параметров i и j, и слово «NO» в противном случае.

### Примеры

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 4 3  2  3  0  2 | YES |
| 4 5  1  0  5  3 | NO |

**Метки:**нет

Сложность: 238

***Описание решения:*** Программа должна проверить выполнение неравенства для всех возможных i и j и вывести "YES", если оно верно для всех значений i и j, иначе вывести "NO".

В данном решении используется вектор для хранения последовательности.

Основная идея заключается в том, что мы проверяем корректность последовательности, суммируя элементы по порядку в обе стороны и сравнивая сумму с текущим индексом. Если сумма нарушает условие (меньше текущего индекса), то выводим "NO", иначе выводим "YES". Мы проходим по элементам последовательности дважды – сначала по порядку, затем в обратном порядке, чтобы учитывать суммы как слева направо, так и справа налево. Если нарушение условия происходит хотя бы для одного элемента, мы сразу выводим "NO" и завершаем программу, иначе выводим "YES". Это решение работает за линейное время, которое зависит от количества элементов в последовательности s.

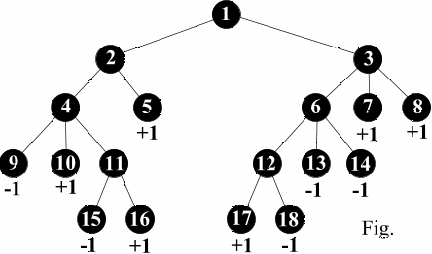
Особенности работы с вводом информации: в С++ мы записываем значения в переменных, а в C# записываем значения в массив и обращаемся к этим значениям через индекс.

Листинги программы приведены в Приложении 6.

1. ***1282. Дерево игры***

Ограничение времени: 1,0 секунды  
Ограничение памяти: 64 МБ

Игра для двух игроков определяется ее деревом. Участники делают ходы по очереди. Первый участник начинает игру. Игра заканчивается либо ничьей, либо победой одного из игроков. Листовые узлы дерева этой игры могут иметь значения, равные одному из трех чисел: «+1» – победа первого участника, «–1» – победа второго участника, «0» – ничья.



Вам предстоит выяснить, кто победит, если оба соперника будут следовать правильной стратегии.

### Исходные данные

Узлы дерева нумеруются последовательными целыми числами. Корень дерева всегда имеет номер 1.

В первой строке записано целое число N (1 < N ≤ 1000) — количество узлов в дереве игры. Следующие N-1 строк описывают узлы – по одной строке на каждый узел (за исключением первого узла). Вторая строка будет содержать описание второго узла дерева, третья строка – описание третьего узла и так далее.

Если узел является листом дерева, то первым символом строки является «L», после него идет пробел, затем идет номер предка этого узла, еще один пробел и результат игры (+1: победа первого игрока, –1: победа второго, 0: ничья).

Если узел внутренний, то строка содержит первый символ «N», пробел и номер предка этого узла.

### Результат

содержит «–1», если выигрывает второй участник, «+1», если побеждает первый, и «0», если результат — ничья.

### Примеры

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 7  № 1  № 1  Л 2 -1  Л 2 +1  Л 3 +1  Л 3 +1 | +1 |
| 7  № 1  № 1  Л 2 -1  Л 2 +1  Л 3 +1  Л 3 0 | 0 |
| 18  № 1  № 1  № 2  Л 2 +1  № 3  Л 3 +1  Л 3 +1  Л 4 -1  Л 4 +1  № 4  № 6  Л 6 -1  Л 6 -1  Л 11 -1  Л 11 +1  Л 12 +1  Л 12 -1 | +1 |

**Метки:**игра

Сложность: 336

***Описание решения:*** Необходимо написать программу, которая определяет результат игры для двух игроков, следующих правильной стратегии, с учетом дерева игры и результатов в листовых узлах. Она должна вернуть "-1" в случае победы второго участника, "+1" в случае победы первого участника, или "0" в случае ничьей.

Используемые структуры данных: структура "Node" для представления узла дерева, вектор "g<Node>" для хранения узлов графа и подузлов.

Данное решение представляет собой рекурсивный обход дерева, который представлен вектором структуры "Node". Структура "Node" содержит значение текущего узла и список подузлов. Функция "solve" рекурсивно обходит дерево, начиная с узла 0, учитывая параметр m. Для каждого узла проверяется наличие подузлов, и если они есть, рекурсивно вызывается функция для подузлов. Значение "value" узлов пересчитывается с учетом параметра m. В функции "main" считывается количество узлов графа, заполняется структура графа g, и выполняется функция "solve" с начальными параметрами. Результат выводится на экран в зависимости от значения в массиве s. Основная идея решения: рекурсивный обход дерева с вычислением оптимального значения для каждого узла с учетом параметра m.

При решении задачи на языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int. В C++ вводимые данные сразу записываются в переменные, без использования массивов. В C++ строчка "constexpr const" в строке "constexpr const char\* s[3] = { "-1", "0", "+1" };" используется для оптимизации программы. А также отключаем синхронизацию ввода-вывода для ускорения программы.

Листинги программы приведены в Приложении 7.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. ***1317. Град***   Ограничение времени: 1.0 секунды Ограничение памяти: 64 МБ  Для защиты от крупных градин автостоянки сделана специальная установка. Крупные градины засекаются радаром, после чего мощный лазер просто испаряет их. Автостоянка представляет собой выпуклый многоугольник и ограничена высоким забором. Лазер находится строго внутри многоугольника на уровне земли. Высота забора h, а мощности лазера хватает на испарение градин на дальности не более d. Градины падают отвесно вниз. Дальности стрельбы достаточно для того, чтобы сбивать падающие прямо на стоянку градины до того, как они опустятся ниже уровня забора, более того, установка в порядке благотворительности сбивает даже те градины, которые не попадают на территорию стоянки. Требуется определить, сколько градин будет сбито. Исходные данные В первой строке записано целое число n (3 ≤ n ≤ 10) — количество вершин многоугольника, и действительное число h (1.00 ≤ h ≤ 100.00) — высота забора. Следующие n строк содержат пары вещественных чисел — координаты вершин многоугольника в порядке обхода. На следующей строке находятся d (h < d < 1000.00) и координаты лазера. Следующая строка содержит k (1 ≤ k ≤ 100) — количество градин. Следующие k строк содержат пары вещественных чисел — координаты точек падения градин. Все координаты по модулю не превышают 1000. Результат Требуется вывести одно число — количество сбитых в воздухе градин. Пример  |  |  | | --- | --- | | **исходные данные** | **результат** | | 4 10.00  1.00 0.00  0.00 1.00  -1.00 0.00  0.00 -1.00  50.00 0.00 0.00  5  0.00 0.00  1.00 1.00  2.00 2.00  3.00 3.00  4.00 4.00 | 3 |   **Метки:**геометрия  Сложность: 1078 |

***Описание решения:*** Для решения данной задачи необходимо определить, сколько градин падающих на автостоянку сбито в воздухе лазером, установленным внутри многоугольника с высоким забором. Вводятся количество вершин многоугольника, высота забора, координаты вершин, дальность стрельбы лазера, координаты лазера, количество градин, их координаты падения. В результате следует вывести количество сбитых в воздухе градин.

Структуры данных, используемые в программе:

1. Структура "point" для хранения координат точек и выполнения операций с векторами
2. Массивы "v[]" и "f[]" для хранения координат градин и вершин многоугольника забора.
3. Массив "ans[]" для отметки доступных градин.

Данная программа решает задачу определения количества градин, которые находятся в области действия лазерного луча и подвергаются уничтожению. Для этого используется метод нахождения пересечения двух отрезков по формуле Крамера. Основная идея решения: Считывание данных о количестве градин, высоте забора, координатах забора и лазера, а также координатах градин. Для каждого сегмента забора и каждой градины определяется пересечение с помощью метода Крамера. Проверяется условие, попадает ли градина в область действия лазера и уничтожается ли она. Если градина уничтожена, то соответствующему элементу массива "ans[]" устанавливается значение "true". Используя функцию "accumulate()", подсчитывается количество уничтоженных градин и далее выводится результат.

При решении задачи на языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int. При реализации в C++ данные сразу после ввода записываются в переменные без преобразований и использования массива.

Листинги программы приведены в Приложении 8.

1. ***1352. Простые числа Мерсенна***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

**Определение.** Если число 2N−1 простое, то оно называется *простым числом Мерсенна*.

Например, 22−1 — первое простое число Мерсенна, 23−1 — второе, 211213−1 — 23-e, 2216091−1 — 31-e.

Искать эти числа без компьютера довольно тяжело. Так, Эйлер в 1772 году нашёл 8-е число, 231−1, но после этого в течение 100 лет не было найдено ни одного! Лишь в 1876 Люк показал, что 2127−1 — простое число. Но он нашел не 9-е, а сразу 12-е, так как 261−1, 289−1, 2107−1 — тоже простые, но были найдены позднее. Новый прорыв случился лишь в 1950-х, когда с помощью вычислительной техники были найдены простые числа Мерсенна с показателями 521, 607, 1279, 2203, 2281. Все последующие числа Мерсенна были найдены с помощью ЭВМ. И для этого не нужно было быть великим математиком, в 1978 и 1979 годах студенты Нолл и Никел нашли 25-е и 26-е числа (21701 и 23209) на мэйнфрейме своего университета, чем и прославились на всю Америку. Но и у современных суперкомпьютеров есть пределы возможностей. Сегодня простые числа Мерсенна ищут десятки тысяч людей по всему миру, объединённые одним метапроектом GIMPS (Great Internet Mersenne Prime Search, www.mersenne.org). На счету GIMPS уже 8 самых больших простых чисел Мерсенна. Их показатели — 1398269, 2976221, 3021377, 6972593, 13466917, 20996011, 24036583, 25964951. 26972593−1 является 38-м простым числом Мерсенна, а про последние 4 ещё нельзя сказать, какие они по счету, т.к. ещё не все меньшие числа проверены. Эти же 4 числа являются самыми большими известными простыми числами.

Последнее число — 225964951−1 — было найдено 18 февраля 2005 года, оно состоит из 7816230 десятичных цифр, тот же, кто найдёт простое число более чем из 10 миллионов цифр, получит приз в $100000. Эти деньги можете выиграть и Вы, если присоединитесь к проекту.

Сейчас от Вас, конечно, не требуется найти за 5 часов 43-е простое число Мерсенна, ведь жюри просто не сможет проверить Ваш ответ. В этой задаче N не превосходит 38. Итак, Вам даётся N, от Вас требуется найти N-е по порядку простое число Мерсенна.

*(Информация приведена по состоянию на март 2005)*

### Исходные данные

В первой строке дано число T — количество тестов. В каждой из последующих T строк дано число N.

### Результат

Для каждого числа N выведите показатель N-го по порядку простого числа Мерсенна.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 13  18  32  24  21  19  34  27  33  20  30  28  29  22 | 3217  756839  19937  9689  4253  1257787  44497  859433  4423  132049  86243  110503  9941 |

**Метки:**теория чисел

Сложность: 130

***Описание решения:*** В задаче требуется найте N-е по порядку простое число Мерсенна, где N не превышает 38. Для каждого числа N из T тестов нужно вывести показатель N-го по порядку простого числа Мерсенна.

Структуры данных, используемые в этом решении: вектор для хранения простых чисел Мерсенна – "data" Вектор для хранения индексов – "result". В этом массиве будут храниться индексы, которые нужны для вывода найденных чисел Мерсенна.

В данном решении используется вектор для хранения известных простых чисел Мерсенна и массив для хранения значений индексов. Основная идея заключается в том, чтобы для каждого введенного индекса находить соответствующее простое число Мерсенна в векторе "data" и выводить его на экран. После ввода количества тестов n, программа последовательно считывает индексы из ввода и находит соответствующее число Мерсенна из вектора "data". Записывает его в массив "result". Затем программа выводит на экран найденные числа Мерсенна для каждого введенного индекса. Это решение эффективно работает за счет заранее заданных простых чисел Мерсенна и использования вектора для их хранения.

Листинги программы приведены в Приложении 9.

1. ***1422. Светлячки***

Ограничение времени: 3.0 секунды  
Ограничение памяти: 64 МБ

### Вступление

Дикий холод пробирает до костей. Синее свечение выжигает глаза. Водоворот силы затягивает всё глубже в сумрак. Туда, откуда не возвращаются… Но это был ещё не конец. Всё решают сила и воля. Последним усилием он поднял тающую тень и вывалился из сумрака в реальный мир — прямо на холодный пол.

Светлый маг Антон Деревенский лежал посреди комнаты рядом с полуразвалившимся диваном и уныло смотрел в потрескавшийся потолок. После вчерашнего голова болела невыносимо. И так хотелось сделать что-нибудь по-настоящему хорошее, доброе. Что-нибудь светлое.

По комнате летали мухи. Много мух. Создания тьмы. Их нужно склонить к свету. Лёгким движением руки г-н Деревенский заморозил всех мух, в результате чего они остались неподвижно висеть в воздухе. Увы, вчерашнее застолье потребовало огромных энергетических затрат, поэтому, решив поджарить всех мух и тем самым превратить их в светлячков, Антон с удивлением обнаружил, что оставшейся у него силы хватит на создание лишь одного файербола, да и то крошечного.

### Задача

Попив водички, г-н Деревенский насчитал N мух, зависших в пространстве в ожидании своей участи. Антон не поленился и весь день пробегал по комнате с линейкой, в результате чего для каждой мухи были определены её декартовы координаты Xi, Yi и Zi. Светлый маг решил обратить к свету максимально возможное количество мух при помощи единственного файербола, который может быть создан в любой точке комнаты и запущен по прямой в любом направлении. Попадая в муху, файербол превращает её в светлячка и продолжает движение в том же направлении. Ввиду нехватки у Антона силы, размерами самого файербола следует пренебречь и считать его материальной точкой.

### Исходные данные

Первая строка содержит целое число N (1 ≤ N ≤ 2000). Каждая из следующих N строк содержит целые числа Xi, Yi и Zi (−107 ≤ Xi, Yi, Zi ≤ 107) для соответствующей мухи.

### Результат

Вывести максимальное количество мух, которых можно превратить в светлячков с помощью файербола.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 6  -1 1 -1  0 1 2  0 0 -1  -2 -1 2  2 -1 -2  -1 0 2 | 3 |

**Метки:** геометрия

Сложность: 868

***Описание решения:*** В условии задачи требуется определить максимальное количество мух, которые может превратить в светлячков маг. Он может осуществить это превращение с помощью единственного "файербола". Координаты каждой мухи заданы в трехмерном пространстве т.е. они находятся в неподвижном, замороженном состоянии в комнате. Файербол может быть создан в любой точке комнаты и запущен в любом направлении по прямой.

Данное решение использует структуру данных "unordered\_map" для хранения векторов и их количества. Ключом в "unordered\_map" является вектор типа vector3D, а значением – количество точек. Хэш-функция в "unordered\_map" используется для определения индекса (хэша) каждого элемента. Благодаря использованию хэш-функции скорость выполнения составляет О(1). Массив "Vector3D[] A" – массив структур "Vector3D", которые представляет собой трехмерные векторы. Структура "Vector3D" – пользовательская структура для представления трехмерного вектора. Она содержит три целочисленных поля x, y, z для хранения координат вектора. Также в структуре есть перегруженные операторы для вычитания (-) и деления (/) векторов, а также операторы сравнения (==, !=). Переменная r – хранит максимальное количество точек в пространстве, обновляется в процессе анализа векторов. Переменная n – используется для ввода количества объектов в пространстве.

Основная идея решение задачи заключается в том, что для каждого вектора из массива A вычисляется разность с остальными векторами, затем находится НОД этой разности и обновляется количество точек с одинаковыми единичными векторами. Максимальное количество таких точек является искомым значением "r". Функции gcd (НОД) используются для нахождения наибольшего общего делителя, что позволяет вычислить векторы с единичными координатами. Данное решение эффективно работает с трехмерными векторами, используя хэширование и операторы для работы с векторами.

При решении задачи на языке C# мы используем структуру данных Dictionary в строчке "Dictionary<Vector3D, int> m = new Dictionary<Vector3D, int>();", а в C++ unordered\_map в строчке "unordered\_map<vector3D, int, vector3D::Hash> m;". Суть их работы одинакова, но в C++ мы для структуры используем дополнительно хэш-функцию. На языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int. В C++ данные считываются и сразу записываются в массивы без использования массива. Кроме того, для оптимизации программы в C++ отключаем синхронизацию потоков ввода-вывода.

Листинги программы приведены в Приложении 10.

1. ***1457. Теплотрасса***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

### Вступление

Я люблю свой город. Меня вдохновляют эти горделиво стремящиеся ввысь хрущёвки и перекопанные ещё в прошлом веке улицы. Мне так близки эти толпы вечно обиженных представителей рабочего класса, прячущихся от армии студентов, охотящихся за стеклотарой пенсионеров, бесконечно милых бомжей и не менее милых молодых людей в чёрных кожаных куртках и кепках-восьмиклинках.

А ещё в городе живёт дедушка. Точнее, жил до тех пор, пока его дом не снесли и не построили на его месте казино. Ничего удивительного, ведь казино городу намного нужнее, чем какой-то там дедушка. Против принципов рыночной экономики не попрёшь.

Поэтому дедушке пришлось переселиться в теплотрассу, проходящую прямо под городом. При всех своих недостатках, проживание в теплотрассе подразумевает бесплатное водоснабжение, отопление и вообще полное отсутствие квартплаты. Словом, дедушку ждёт достойная старость. Спасибо за это родному государству.

Но как бы ни была прекрасна жизнь в теплотрассе, временами дедушке всё же необходимо подниматься на поверхность и посещать один из нескольких важных пунктов. Иногда ему нужно убедиться в отсутствии бесплатных лекарств в поликлинике, запастись продовольствием на рыночной свалке, получить пенсию на почте или отдать эту пенсию внуку - как раз хватит на мороженое!

### Задача

Теплотрасса была построена ещё при Сталине, и поэтому представляет собой прямой тоннель без ответвлений. Любая её точка характеризуется своим магистральным смещением. Магистральное смещение начала теплотрассы, расположенного под зданием городской администрации, равно нулю. Расстояние между любыми двумя точками теплотрассы вычисляется как модуль разности их магистральных смещений.

Так уж получилось, что теплотрасса проходит подо всеми N пунктами, посещаемыми дедушкой. Для каждого канализационного люка, ведущего из теплотрассы прямо в один этих пунктов, известно его магистральное смещение P[i]. Дедушка может вылезать из теплотрассы только через эти люки. Если он попытается вылезти через какой-нибудь другой люк, то бдительные милиционеры сразу же задержат его как опасного бомжа.

Дедушка уже старенький, и усилия, затрачиваемые им на преодоление некоторого расстояния, **пропорциональны квадрату этого расстояния**. Поэтому дедушка хотел бы жить в той точке теплотрассы, для которой среднее арифметическое усилий на достижение каждого из пунктов минимально.

### Исходные данные

Первая строка содержит целое число N (1 ≤ N ≤ 1000). Вторая строка содержит N целых чисел P[i] (0 ≤ P[i] ≤ 106).

### Результат

Вывести магистральное смещение искомой точки не менее чем с шестью знаками после десятичной точки. Если задача имеет несколько решений, то вывести любое из них.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 3  7 4 5 | 5.333333 |

**Метки:**нет

Сложность: 67

***Описание решения:*** Для решения задачи необходимо определить магистральное смещение искомой точки теплотрассы, для которой среднее арифметическое усилий на достижение каждого из "N" пунктов минимально. Суть задачи – найти среднее арифметическое чисел.

В программе используется переменные "p", "N", "sum". "p" – для записи чисел, "sum"- суммирование этих чисел и в дальнейшем вычисления среднего арифметического, а "N" – для ввода количества чисел.

Основная идея программы заключается в следующем. Сначала считываются количество чисел, затем в цикле считываются сами числа, потом они суммируются и далее находится среднее арифметическое. Результат выводится с точностью до 6 знаков после запятой с помощью функции printf. Программа позволяет пользователю ввести любое количество чисел и находит их среднее арифметическое.

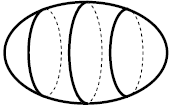
При решении задачи на языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int. При реализации на Python используется сокращенная запись для заполнения массива числами "p = [int(x) for x in sys.stdin.readline().split()]". В C++ просто записываются числа в переменную, без использования массива.

Листинги программы приведены в Приложении 11.

1. ***1562. Ананас***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

Одну из башен комплекса Стражи Урала, как известно, строит бригада Равшана и Джамшуда. На День Строителя Небоскрёбов (да-да, есть и такой) насяльника подарил им ананас! И не простой, а генетически модифицированный! Решили строители поделить ананас как можно более по-братски. Думали-думали и... придумали! Положили ананас на бок, взяли измерительную рулетку и пилу-болгарку и распилили его на n равношироких кусков-кружков. А Равшана (то есть вас), как самого умного, заставили делить эти куски между всей бригадой. Но вы (то есть Равшан) оказались не лыком шиты и сказали, что поделите, как только выясните, сколько весит каждый кусок. Вот только незадача — весов строители в хозяйстве своём не держат. Зато известно, что генетически модифицированный ананас — это вытянутый эллипсоид вращения, шириной a и высотой b (b > a) и что 1 см3 генетически модифицированного ананаса весит 1 грамм. Этих данных, как понял Равшан, хватит ему, чтобы выяснить, сколько весит каждый кусок.



### Исходные данные

В единственной строке записаны 3 числа: a и b — ширина и высота ананаса в сантиметрах, измеренные штангенциркулем (1 ≤ a < b ≤ 100); а также n — количество кусков (1 ≤ n ≤ 100).

### Результат

Выведите вес каждого куска в граммах с точностью до микрограмма в порядке их отрезания. Каждое число выводите в отдельной строке.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 15 20.5 8 | 103.773800  273.585474  386.793256  443.397147  443.397147  386.793256  273.585474  103.773800 |

**Метки:**геометрия

Сложность: 410

***Описание решения:*** Для решения задачи необходимо определить и вывести массу кусочков ананаса, представленного в виде вытянутого эллипсоида вращения. Кусочки представляют равно-широкие кружки. Каждое число нужно выводить в отдельной строке с точностью до микрограмма.

Используются три переменные для хранения вводимых данных.

Исходные данные содержат ширину, высоту и количество кусочков ананаса. В данной программе решается задача о разрезании ананаса на n кусков одинакового веса. Сначала считываются данные – значения ширины и высоты ананаса, а также количество кусочков. Затем вычисляется шаг разрезания и начальное значение x. Последовательно вычисляется размер каждого куска ананаса с помощью функции "pieceSize", которая использует формулу для вычисления размера части ананаса –

.

Результат выводится с точностью до микрограмма.

При решении задачи на языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int и double. При реализации на C++ считываемые данные сразу записываются в переменные.

Листинги программы приведены в Приложении 12.

1. ***1737. Мнемоника и палиндромы 3***

Ограничение времени: 1.0 секунды  
Ограничение памяти: 64 МБ

Как вы помните, когда Васечкин готовил задачу для последнего студенческого соревнования, он долго пытался придумать для неё необычное и сложное название. Но название, придуманное им, оказалось настолько сложным, что никто из участников того соревнования даже не стал читать текст задачи Васечкина.

После этого председатель программного комитета заявил, что отказывается участвовать в подготовке соревнований, пока в программном комитете работают такие неадекватные люди, как Васечкин. Так Васечкин стал новым председателем программного комитета и начал готовить следующее соревнование по программированию.

Васечкин решил, что на этот раз названия всех задач будут состоять только из букв a, b и c и иметь длину n. Кроме того, все названия должны быть чрезвычайно сложными, а именно, никакая подстрока никакого названия, состоящая из двух или более символов, не должна являться палиндромом. Помогите Васечкину и найдите все чрезвычайно сложные названия для задач предстоящего соревнования.

### Исходные данные

В единственной строке записано целое число n (1 ≤ n ≤ 20000).

### Результат

Выведите все различные чрезвычайно сложные названия длины n, состоящие только из букв a, b и c. Названия следует выводить в алфавитном порядке, по одному в строке. Если суммарная длина названий превосходит 100000 букв, выведите единственную строку «TOO LONG».

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 2 | ab  ac  ba  bc  ca  cb |

**Метки:**палиндромы

Сложность: 246

***Описание решения:*** Для решения задачи требуется написать программу, которая будет выводить все различные "чрезвычайно сложные названия", согласно условию, для задач длины n. Названия должны состоять только из букв a, b и c. Если общая длина всех названий превышает 100000 букв, программа должна вывести "TOO LONG".

В программе используются следующие структуры данных. Массив "abc[]" для хранение символов "a", "b" и "c". Очередь "result" с типом string используется в данном решении для хранения строк, которое формируются на основе определенных правил в цикле. Очередь обеспечивает правильную последовательность, в которой формируются новые строки.

Данное решение представляет собой программу, которая генерирует все возможные строки из символов "a", "b", "c" длины не более чем "n". Для этого используется очередь "result" типа "queue<string>", в которой хранятся все текущие строки. Начинается все с того, что в очереди уже есть строки "a", "b", "c". Затем итеративно создаются все возможные строки длины "i", добавляя к каждой строке символ "a", "b" или "c". Проверяется, что последний и предпоследний символы строки не равны новому добавляемому символу "c", чтобы не создавать строки типа "cc". Если все условия выполняются, то мы добавляем новую строку в очередь. Если размер строки умноженный на (i+1) превышает 100000, программа завершает выполнение и выводит сообщение "TOO LONG". В конце программы мы выводим все строки из очереди "result", удаляя их из очереди по одной, до тех пор, пока очередь не станет пустой.

Логика решения задачи на C++ и C# одинакова. Используются одни и те же методы, но с разными названиям в этих языках.

Листинги программы приведены в Приложении 13.

1. ***1877. Велосипедные коды***

Ограничение времени: 0.5 секунды  
Ограничение памяти: 64 МБ

У Дена есть два четырёхзначных кодовых замка для велосипеда. Каждый вечер он ставит велосипед на сигнализацию и пристёгивает к специальной стойке одним из замков. Ден никогда не использует один и тот же замок два вечера подряд. В некоторую ночь злоумышленник попытался с помощью кода 0000 открыть висящий на велосипеде замок. Сработала сигнализация, и вор поспешил скрыться. На следующую ночь он решил попробовать код 0001, затем 0002 и так далее в порядке возрастания номера.

Известно, что Ден не меняет кодов и в ночь, когда вор пришёл впервые, велосипед был пристёгнут первым замком.

### Исходные данные

В первой строке записан код, установленный на первом замке, во второй строке — код, установленный на втором замке. Оба кода — строки длины 4, состоящие из цифр от 0 до 9.

### Результат

Выведите «yes», если злоумышленник рано или поздно взломает замок, и «no» в противном случае.

### Примеры

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 0001  0000 | no |
| 0002  0001 | yes |

**Метки:**задача для начинающих

Сложность: 19

***Описание решения:*** Имеются два четырёхзначных кодовых замка для велосипеда, которые меняются на велосипеде каждый вечер т.е. сначала первый замок, потом второй и далее снова первый. Злоумышленник пытается взломать замок, перебирая по одному коду каждый вечер. Начинает он перебор с 0000 до 9999 с шагом 1. Указаны коды замков. Требуется выяснить удастся ли злоумышленнику взломать хоть один замок.

В решении используются две целочисленные переменные.

Код проверяет два введенных числа на четность и нечетность. Основная идея состоит в том, что если первое число четное или второе число нечетное, то выводится "yes", иначе выводится "no".

Листинги программы приведены в Приложении 14.

1. ***1982. План электрификации***

Ограничение времени: 0.5 секунды  
Ограничение памяти: 64 МБ

В некоторой стране n городов. Правительство решило электрифицировать все эти города. Для начала в k различных городах были построены электростанции. Другие города должны быть связаны с электростанциями линиями электропередач. Между любой парой городов i и j можно построить линию электропередач стоимостью cij рублей. После гражданской войны страна находится в глубоком кризисе, поэтому правительство решило построить всего лишь несколько линий электропередач. Конечно, после постройки линий должен существовать путь по ним от любого города до некоторого города с электростанцией. Найдите минимальную возможную стоимость постройки всех необходимых для этого линий электропередач.

### Исходные данные

В первой строке записаны целые числа n и k (1 ≤ k ≤ n ≤ 100). Во второй строке записаны k различных целых чисел — номера городов с электростанциями. В следующих n строках записана таблица {cij} размера n × n, состоящая из целых чисел (0 ≤ cij ≤ 105). Гарантируется, что cij = cji, cij > 0 для i ≠ j, cii = 0.

### Результат

Выведите минимальную стоимость электрификации всех городов.

### Пример

|  |  |
| --- | --- |
| **исходные данные** | **результат** |
| 4 2  1 4  0 2 4 3  2 0 5 2  4 5 0 1  3 2 1 0 | 3 |

**Метки:** графы

Сложность: 131

***Описание решения:*** Дано число городов n, число городов с электростанциями k, номера городов с электростанциями, стоимости постройки линий электропередач между городами. Необходимо найти минимальную стоимость электрификации всех городов, с учетом постройки нескольких линий электропередач.

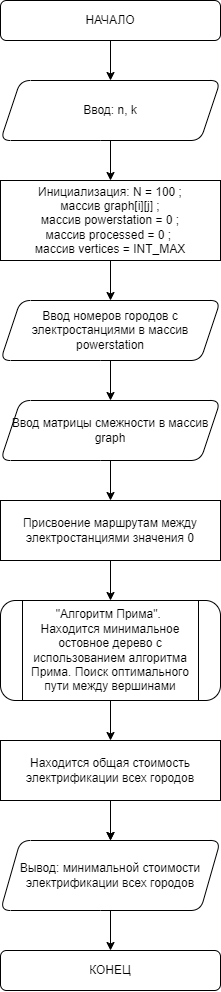
В коде используются следующие структуры данных. Целочисленные переменные n и k. Массив "powerstation" – хранит информацию о наличии электростанций в каждом городе. Двумерный массив "graph" – хранит веса связей между городами. Массив "vertices", хранящий минимальный вес вершин в остовном дереве. Массив "processed", для отслеживания обработанных вершин.

Код решает поставленную задачу по минимизации затрат на электрификацию городов, учитывая наличие некоторых уже существующих электростанций. Основная идея заключается в поиске минимального остовного дерева в графе связей между городами с использованием алгоритма Прима. Вначале вводятся данные о количестве городов и городах с электростанциями. Затем заполняется таблица связей между городами, учитывая электростанции. Далее находится минимальное остовное дерево с использованием алгоритма Прима, которое позволяет выяснить минимальную стоимость электрификации всех городов. В конце выводиться общая стоимость электрификации всех городов.

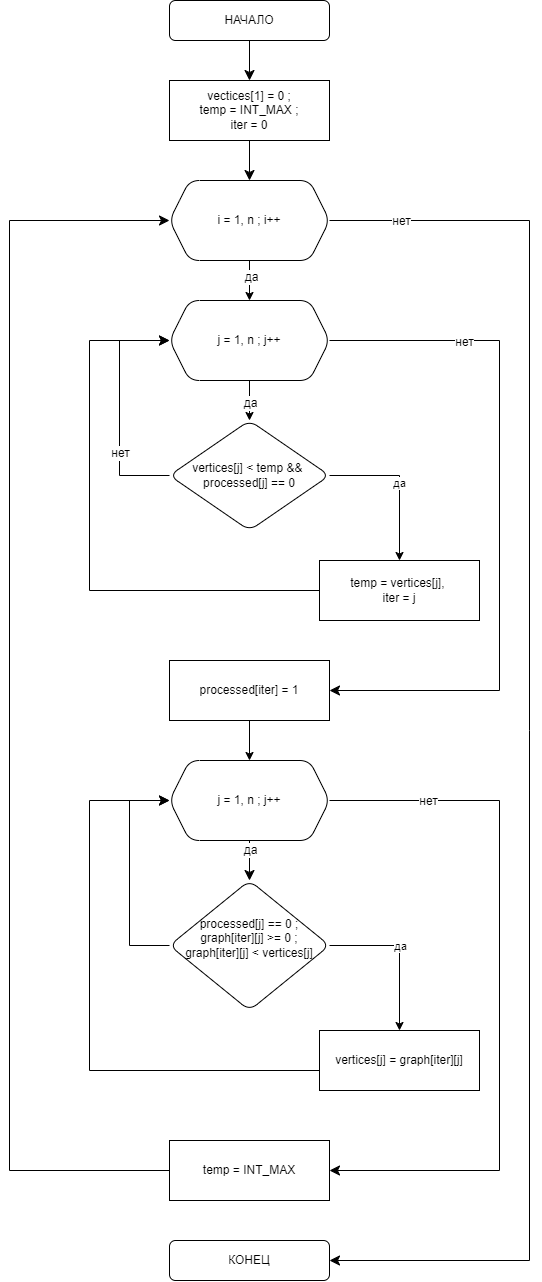
При решении задачи на языке C# считывается строка, с помощью метода Split() разбивается на элементы (строковый массив), они переводятся в тип int. Также в C# используется метод "Trim()" лишних символов из строки. В C++ считываемые значения "n" и "k" сразу записываются в переменные, без использования массива.

Листинги программы приведены в Приложении 15.

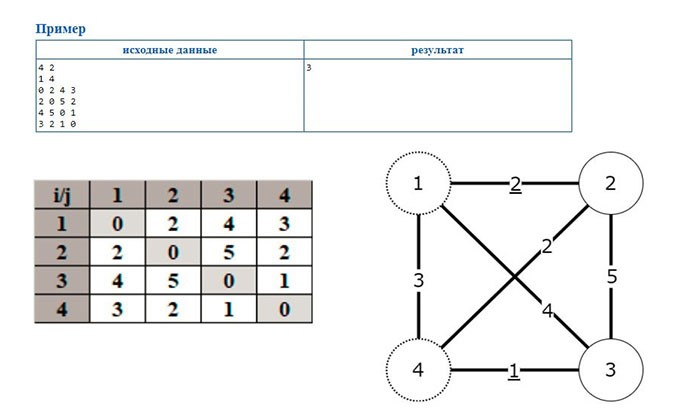
Блок схема решения задачи:



Блок схема подпрограммы – алгоритм Прима:



Пример выполнения программы:



Примечание к рисунку:

Строка №1: n=4; k=2 (n – число городов, k – число городов с электростанциями).

Строка №2: 1, 4 – номера городов с электростанциями.

Строки №3-6: Таблица: размер – n\*n; Cij = Cji; Cij > 0, для i != j; Cii = 0.

Результат: 3 – минимальная стоимость электрификации всех городов.

Вершины графа – города.

Вершины с пунктиром – города с электростанциями.

Числа на ребрах графа – стоимость электрификации.

Подчеркнутые числа на ребрах графа – выбранная минимальная стоимость электрификации.

В данном примере на ввод подается n – число городов и k – число городов с электростанциями. Далее предоставлены номера городов с электростанциями. Дальше предоставлена матрица смежности, которая показывает стоимость постройки линии электропередачи между двумя городами. На рисунке представлена матрица смежности и граф. На графе все вершины соединены друг с другом. Пунктирным контуром отмечены вершины графа, которые являются электрифицированными городами. Числа на ребрах графа обозначают стоимость постройки ЛЭП между двумя городами. А подчеркнутые числа на ребрах графа представляют собой минимально возможную стоимость электрификации всех городов. Чтобы найти минимальную стоимость электрификации, используется алгоритм Прима для поиска остовного дерева. В нашем примере это стоимость постройки ЛЭП между первым и вторым городом равная 2-м условным единицам. И стоимость постройки ЛЭП между 4-ым и 3-им равная 1-ой условной единице. В итоге сумма постройки будет 2 + 1 = 3. Для наших входных данных минимальная сумма постройки будет равна 3.

# Список консультаций

За период прохождения практики были посещены следующие консультации.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Дата | Содержание | Сданные задачи |
|  | 12.02.2024 | Установочная конференция |  |
|  | 28.02.2024 | Защита решения задач | 1877, 1352, 1457, 1737, 1562, 1142, 1247 |
|  | 28.03.2024 | Защита решения задач | 1002, 1072, 1107, 1212, 1422 |
|  | 11.04.2024 | Защита решения задач | 1282, 1982, 1317 |
|  | 13.04.2024 | Консультация по оформлению отчета |  |
|  | 29.04.2024 | Отправил отчет на проверку |  |
|  | 03.05.2024 | Отправил отчет на проверку |  |
|  | 04.05.2024 | Заключительная конференция |  |

# Заключение

Учебная практика способствовала закреплению и углублению теоретической подготовки, приобретению практических навыков и компетенций в сфере профессиональной деятельности путём самостоятельного решения задач.

В ходе практики было решено 15 задач, реализованных на языках программирования С++, C# и Python. Листинги всех составленных программ приведены в приложении.

Наиболее интересным оказалось решение задачи №1982, а именно разобраться как работает алгоритм Прима для нахождения минимального остовного дерева. Также было интересно находить решения задач, используя различные алгоритмы из теории графов, динамического программирования и геометрические формулы.

Наиболее сложным оказалось успевать сдавать в срок выданные задачи. Особенно завершающие, которые требовали для их решения большое количество времени и грамотной оптимизации алгоритма решения. Были сложности с ограничениями по времени в задачах. Часть задач на сайте «acm.timus» крайне тяжело решить, используя такие языки, как Python или C#. Данные задачи имеют серьезные ограничения по времени, которые удается соблюдать только, если Вы пишите код на языках C++ или Rust. Или у вас действительно максимально эффективный и оптимизированный алгоритм решения задачи. Из-за ограничений по времени пришлось отказаться от пары задач, хотя они были решены на нужных языках. Также возникли трудности с выявлением ошибок в коде. Сайт не дает возможности просмотреть тесты, которые использует для проверки программ. Вдобавок в обсуждениях задач нету необходимой информации по ошибкам и их причинам, а лично заданные вопросы остаются проигнорированы, поскольку количество активных пользователей сайта небольшое. Еще авторы задач часто переусложняют условия задач и пишут целые сочинения для них. Это создает дополнительные трудности в понимании постановки задачи из-за чего даже простые задачи кажутся довольно сложными.

# Список литературы

1. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. – М.: Вильямс, 2016. – 400 с.
2. Вирт Н. Алгоритмы и структуры данных. М.: ДМК, 2011. – 272 с.
3. Задачи по программированию: учебное пособие / С. М. Окулов, Т. В. Ашихмина, Н. А. Бушмелева, М. А. Корчёмкин. – 3-е изд. – М.: Лаборатория знаний, 2017. – 826 с.
4. Кнут Д. Искусство программирования. Том 1. Основные алгоритмы. – 3-е изд. – М.: Вильямс, 2017. – 720 с.
5. Кнут Д. Искусство программирования. Том 3. Сортировка и поиск. –   
   2-е изд. – М.: Вильямс, 2017. – 824 с.
6. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. – М.: Вильямс, 2019. – 1328 с.
7. Лутц М. Изучаем Python. Том 1. – М.: Издательство Диалектика, 2019. – 832 с.
8. Окулов С. М. Основы программирования: учебное пособие / С. М. Окулов. – 10-е изд. – М.: Лаборатория знаний, 2020. – 339 с.
9. Окулов С. М. Программирование в алгоритмах [Текст] / С. М. Окулов. – 6-е изд. – М.: Лаборатория знаний, 2017. – 386 с.
10. Окулов С. М. Алгоритмы обработки строк: учебное пособие / С. М. Окулов. – 4-е изд. – М.: Лаборатория знаний, 2020. – 258 с.
11. Окулов, С. М. Дискретная математика. Теория и практика решения задач по информатике: учеб. пособие / С. М. Окулов. – М.: Лаборатория знаний, 2020. – 422 с.
12. Окулов, С. М. Динамическое программирование: учебное пособие / С. М. Окулов, О. А. Пестов. – 3-е изд. – М.: Лаборатория знаний, 2020. – 299 с.

Приложения

**Приложение 1. Задача 1002. Телефонные номера.**

**Листинг на языке C++**

#include <iostream>

#include <string>

#include <map> // контейнер map

#include <vector>

#include <queue>

#include <stack>

using namespace std;

string num, word;

int n;

map <char, char> alphabet;

vector <int> par;

vector <int> d;

vector <string> dict;

vector <vector <int> > A;

queue <int> q;

stack <int> r;

// Инициализация алфавита

void init\_alphabet(void)

{

alphabet['a'] = '2';

alphabet['b'] = '2';

alphabet['c'] = '2';

alphabet['d'] = '3';

alphabet['e'] = '3';

alphabet['f'] = '3';

alphabet['g'] = '4';

alphabet['h'] = '4';

alphabet['k'] = '5';

alphabet['l'] = '5';

alphabet['m'] = '6';

alphabet['n'] = '6';

alphabet['p'] = '7';

alphabet['r'] = '7';

alphabet['s'] = '7';

alphabet['t'] = '8';

alphabet['u'] = '8';

alphabet['v'] = '8';

alphabet['w'] = '9';

alphabet['x'] = '9';

alphabet['y'] = '9';

alphabet['o'] = '0';

alphabet['q'] = '0';

alphabet['z'] = '0';

alphabet['i'] = '1';

alphabet['j'] = '1';

}

// Инициализация объектов программы

void init(void)

{

dict.clear(); // очистка вектора dict

dict.resize(n); // изменение размера вектора dict

par.clear();

par.resize(num.size() + 1);

d.clear();

d.resize(num.size() + 1);

A.clear();

A.resize(num.size() + 1);

for (int i = 0; i < A.size(); ++i)

{

A[i].resize(num.size() + 1);

}

}

// Проверка

void test(int k, int n)

{

for (int i = k; i < k + dict[n].size(); ++i) // цикл от k до k + размер слова

{

if (num[i] != alphabet[dict[n][i - k]]) // если символы не совпадают, выход из функции

return;

}

A[k][k + dict[n].size()] = n + 1; // заполнение матрицы A значением

}

// Поиск соответствия кратчайших слов номеру телефона. Алгоритм BFS(Breadth - First Search) — алгоритм поиска в ширину

void BFS(int u)

{

d[u] = 1; // инициализация d

q.push(u); // добавление u в очередь

while (!q.empty()) // пока очередь не пуста

{

u = q.front(); // получение элемента из очереди

q.pop(); // удаление элемента из очереди

for (int v = 0; v <= num.size(); ++v) // цикл от 0 до размера строки

{

if (!d[v] && A[u][v]) // если v не посещена и есть путь

{

q.push(v); // добавление v в очередь

d[v] = d[u] + 1; // увеличение расстояния

par[v] = u; // установка предыдущего узла

}

}

}

}

// Вывод значения или сообщения об отсутствии решения

void print(void)

{

int u = num.size();

if (d[u] != 0)

{

while (u != 0)

{

r.push(u);

u = par[u];

}

while (u != num.size())

{

int v = r.top();

r.pop();

cout << dict[A[u][v] - 1] << ' ';

u = v;

}

printf("\n");

}

else

printf("No solution.\n");

}

int main(void)

{

#ifndef ONLINE\_JUDGE

// Открытие файлов для ввода и вывода

freopen("T1002.in", "r", stdin);

freopen("T1002.out", "w", stdout);

#endif

init\_alphabet(); // инициализация алфавита

while (getline(cin, num) && num != "-1") // считывание значений, пока не встречено "-1"

{

scanf("%d\n", &n); // считывание значения n

init(); // инициализация объектов

for (int i = 1; i <= n; ++i) // цикл по словарю

{

getline(cin, word); // считываем строки

dict[i - 1] = word; // записываем строки в словарь

for (int j = 0; j < num.size(); ++j) // цикл по символам строки

{

if (num[j] == alphabet[word[0]] && j + word.size() - 1 < num.size()) // если символы совпадают и слово помещается

test(j, i - 1); // проверка слова

}

}

BFS(0); // поиск кратчайшего слова

print(); // вывод результата

}

return 0;

}

**Листинг на языке Python**

num = ""

word = ""

n = 0

alphabet = {}

par = []

d = []

dict = []

A = []

q = []

r = []

def init\_alphabet():

alphabet['a'] = '2'

alphabet['b'] = '2'

alphabet['c'] = '2'

alphabet['d'] = '3'

alphabet['e'] = '3'

alphabet['f'] = '3'

alphabet['g'] = '4'

alphabet['h'] = '4'

alphabet['k'] = '5'

alphabet['l'] = '5'

alphabet['m'] = '6'

alphabet['n'] = '6'

alphabet['p'] = '7'

alphabet['r'] = '7'

alphabet['s'] = '7'

alphabet['t'] = '8'

alphabet['u'] = '8'

alphabet['v'] = '8'

alphabet['w'] = '9'

alphabet['x'] = '9'

alphabet['y'] = '9'

alphabet['o'] = '0'

alphabet['q'] = '0'

alphabet['z'] = '0'

alphabet['i'] = '1'

alphabet['j'] = '1'

def init():

dict.clear()

dict.extend([None] \* n)

par.clear()

par.extend([None] \* (len(num) + 1))

d.clear()

d.extend([None] \* (len(num) + 1))

A.clear()

for i in range(len(num) + 1):

A.append([None] \* (len(num) + 1))

def test(k, n):

for i in range(k, k + len(dict[n])):

if num[i] != alphabet.get(dict[n][i - k]):

return

A[k][k + len(dict[n])] = n + 1

def BFS(u):

d[u] = 1

q.append(u)

while q:

u = q.pop(0)

for v in range(len(num) + 1):

if not d[v] and A[u][v]:

q.append(v)

d[v] = d[u] + 1

par[v] = u

def print\_result():

u = len(num)

if d[u]:

while u != 0:

r.append(u)

u = par[u]

while u != len(num):

v = r.pop()

print(dict[A[u][v] - 1], end=' ')

u = v

print()

else:

print("No solution.")

if \_\_name\_\_ == "\_\_main\_\_":

init\_alphabet()

while True:

num = input()

if num == "-1":

break

n = int(input())

init()

for i in range(1, n + 1):

word = input()

dict[i - 1] = word

for j in range(len(num)):

if num[j] == alphabet.get(word[0]) and j + len(word) - 1 < len(num):

test(j, i - 1)

BFS(0)

print\_result()

**Приложение 2. Задача 1072. Маршрутизация.**

**Листинг на языке C++**

#include <iostream>

#include <string>

#include <vector>

#include <queue>

#include <stack>

#include <cmath>

#include <algorithm>

#include <map>

#include <set>

using namespace std;

/\*

scanf\_s->scanf

#ifndef ONLINE\_JUDGE

freopen("T1072.in", "r", stdin);

freopen("T1072.out", "w", stdout);

#endif

\*/

// Объявление глобальных переменных

int n, k, start, end\_;

vector <vector <string>> inter;

vector <vector <int>> A;

vector <int> D, P;

queue <int> q;

stack <int> path;

// Функция для преобразования числа в двоичное представление

void trans(string& s, int n, int pos)

{

for (int i = 128; i >= 1; i /= 2)

{

s[pos++] += n / i;

n %= i;

}

}

// Функция для выполнения операции побитового "и" над двумя строками

void AND(string& s, string& m)

{

for (int i = m.size() - 1; i >= 0; --i)

{

if (m[i] == '0')

s[i] = '0';

else

return;

}

}

// Функция для ввода данных

//Функция input для ввода данных.Считываются данные о сетях и создается граф, где сети, имеющие общие узлы, соединены ребром.

void input(void)

{

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

{

scanf\_s("%d\n", &k);

for (int j = 0; j < k; ++j)

{

string s = "00000000000000000000000000000000";

string m = "00000000000000000000000000000000";

int t;

scanf\_s("%d.", &t);

trans(s, t, 0);

scanf\_s("%d.", &t);

trans(s, t, 8);

scanf\_s("%d.", &t);

trans(s, t, 16);

scanf\_s("%d ", &t);

trans(s, t, 24);

scanf\_s("%d.", &t);

trans(m, t, 0);

scanf\_s("%d.", &t);

trans(m, t, 8);

scanf\_s("%d.", &t);

trans(m, t, 16);

scanf\_s("%d\n", &t);

trans(m, t, 24);

AND(s, m);

inter[i].push\_back(s);

for (int j = i - 1; j >= 0; --j)

{

for (int l = 0; l < inter[j].size(); ++l)

{

if (s == inter[j][l])

{

A[i].push\_back(j);

A[j].push\_back(i);

}

}

}

}

}

scanf\_s("%d%d", &start, &end\_);

start--;

end\_--;

}

// Функция BFS для поиска кратчайшего пути в графе с помощью алгоритма обхода в ширину

void BFS(int u)

{

D[u] = 1;

q.push(u);

while (!q.empty())

{

u = q.front();

q.pop();

for (int v = 0; v < A[u].size(); ++v)

{

if (!D[A[u][v]])

{

D[A[u][v]] = D[u] + 1;

q.push(A[u][v]);

P[A[u][v]] = u;

}

}

}

}

// Функция восстановления пути - восстанавливает путь от начальной точки до конечной

void return\_path(void)

{

int u = end\_;

path.push(u);

do

{

u = P[u];

path.push(u);

} while (u != start);

}

int main(void)

{

scanf\_s("%d\n", &n); // считывается количество сетей

// Инициализация / выделение памяти для векторов

inter.resize(n);

A.resize(n);

D.resize(n);

P.resize(n);

input(); // ввод данных

BFS(start); // запуск алгоритма поиска кратчайшего пути

// Если путь существует, выводится сообщение "Yes" и путь от стартовой до конечной точки.

// Иначе выводится сообщение "No"

if (D[end\_]) // Если до нужной вершины можно дойти

{

return\_path();

printf("Yes\n");

while (!path.empty())

{

printf("%d ", path.top() + 1);

path.pop();

}

}

else // Если до нужной вершины нельзя дойти

{

printf("No");

}

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net.NetworkInformation;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace PhoneWordSearch

{

class Program

{

// Объявление глобальных переменных

static int n, k, start, end\_;

static List<List<string>> inter = new List<List<string>>();

static List<List<int>> A = new List<List<int>>();

static List<int> D = new List<int>();

static List<int> P = new List<int>();

static Queue<int> q = new Queue<int>();

static Stack<int> path = new Stack<int>();

// Функция для преобразования числа в двоичное представление

static void trans(ref string s, int n, int pos)

{

for (int i = 128; i >= 1; i /= 2)

{

s = s.Insert(pos++, (n / i).ToString());

n %= i;

}

}

// Функция для выполнения операции побитового "и" над двумя строками

static void AND(ref string s, string m)

{

for (int i = m.Length - 1; i >= 0; i--)

{

if (m[i] == '0')

s = s.Remove(i, 1).Insert(i, "0");

else

return;

}

}

// Функция для ввода данных

static void Input()

{

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

{

k = Convert.ToInt32(Console.ReadLine());

for (int j = 0; j < k; j++)

{

string s = "00000000000000000000000000000000";

string m = "00000000000000000000000000000000";

string[] parts; // Trim() - обрезает пробелы по краям

string input;

int part\_1, part\_2, part\_3, part\_4, part\_5, part\_6, part\_7, part\_8;

//int t;

input = Console.ReadLine();

parts = input.Split('.', ' ');

//for (int k = 0; k < parts.Length; k++)

//{

// Console.WriteLine(parts[k]);

//}

part\_1 = Convert.ToInt32(parts[0].Trim());

part\_2 = Convert.ToInt32(parts[1].Trim());

part\_3 = Convert.ToInt32(parts[2].Trim());

part\_4 = Convert.ToInt32(parts[3].Trim());

part\_5 = Convert.ToInt32(parts[4].Trim());

part\_6 = Convert.ToInt32(parts[5].Trim());

part\_7 = Convert.ToInt32(parts[6].Trim());

part\_8 = Convert.ToInt32(parts[7].Trim());

trans(ref s, part\_1, 0);

trans(ref s, part\_2, 8);

trans(ref s, part\_3, 16);

trans(ref s, part\_4, 24);

trans(ref m, part\_5, 0);

trans(ref m, part\_6, 8);

trans(ref m, part\_7, 16);

trans(ref m, part\_8, 24);

AND(ref s, m);

inter[i].Add(s);

for (int x = i - 1; x >= 0; x--)

{

for (int l = 0; l < inter[x].Count; l++)

{

if (s == inter[x][l])

{

A[i].Add(x);

A[x].Add(i);

}

}

}

}

}

string[] inputs = Console.ReadLine().Split(' ');

start = Convert.ToInt32(inputs[0]) - 1;

end\_ = Convert.ToInt32(inputs[1]) - 1;

}

// Функция BFS для поиска кратчайшего пути в графе

static void BFS(int u)

{

D[u] = 1;

q.Enqueue(u);

while (q.Count > 0)

{

u = q.Dequeue();

for (int v = 0; v < A[u].Count; v++)

{

if (D[A[u][v]] == 0)

{

D[A[u][v]] = D[u] + 1;

q.Enqueue(A[u][v]);

P[A[u][v]] = u;

}

}

}

}

// Функция восстановления пути

static void ReturnPath()

{

int u = end\_;

path.Push(u);

do

{

u = P[u];

path.Push(u);

} while (u != start);

}

static void Main()

{

n = Convert.ToInt32(Console.ReadLine());

// Инициализация / выделение памяти для векторов

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

{

inter.Add(new List<string>());

A.Add(new List<int>());

D.Add(0);

P.Add(0);

}

Input();

BFS(start);

if (D[end\_] > 0)

{

ReturnPath();

Console.WriteLine("Yes");

foreach (int node in path)

{

//Console.Write($"{path.Peek() + 1} ");

//path.Pop();

Console.Write($"{node + 1} ");

}

}

else // If there is no path to the end node

{

Console.WriteLine("No");

}

// string iiu = Console.ReadLine();

}

}

}

**Приложение 3. Задача 1107. Складская задача.**

**Листинг на языке C++**

#include <cstdio>

using namespace std;

int main()

{

size\_t n, k, m;

scanf\_s("%zu %zu %zu", &n, &k, &m); // zu - беззнакове целое число ; z - size\_t | u - unsigned

std::printf("YES\n");

for (size\_t i = 0; i < k; ++i) {

size\_t q;

scanf\_s("%zu", &q); // вводим количество видов товара

unsigned s = 0;

for (size\_t j = 0; j < q; ++j) {

unsigned x;

scanf\_s("%u", &x); // вводим товары

s += x;

}

std::printf("%u\n", 1 + s % m); // формула для решения задачи

}

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Net.NetworkInformation;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace PhoneWordSearch

{

class MainClass

{

public static void Main(string[] args)

{

string[] parts;

int n, k, m;

string str = Console.ReadLine();

parts = str.Split(' ');

n = Convert.ToInt32(parts[0].Trim());

k = Convert.ToInt32(parts[1].Trim());

m = Convert.ToInt32(parts[2].Trim());

//for (int k = 0; k < parts.Length; k++)

//{

// Console.WriteLine(parts[k]);

//}

Console.WriteLine("YES");

for (uint i = 0; i < k; ++i)

{

str = Console.ReadLine();

parts = str.Split(' ');

int q = Convert.ToInt32(parts[0].Trim());

int s = 0;

for (uint j = 1; j <= q; ++j)

{

int x = Convert.ToInt32(parts[j].Trim());

s += x;

}

Console.WriteLine(1 + s % m);

}

}

}

}

**Приложение 4. Задача 1142. Отношения.**

**Листинг на языке C++**

#include <iostream>

using namespace std;

int main()

{

int n;

const int MAXN = 12;

int A[MAXN][MAXN] = { 0 };

int ans[MAXN] = { 0 };

// Заполнение массива A, с использованием формулы для нахождения всех отношений порядка

// Сначала находим все отношения порядка для N переменных и записываем их по индексу i (кол-ву переменных), потом их суммирует и соотносим с N (кол-ву переменных)

A[1][1] = 1; // для одной переменой

for (int i = 1; i < MAXN - 1; ++i)

{

for (int j = 1; j <= i; ++j)

{

A[i + 1][j] += A[i][j] \* j;

A[i + 1][j + 1] += A[i][j] \* (j + 1);

}

}

for (int i = 1; i < MAXN - 1; ++i) // кол-во переменных в выражении т.е. N

for (int j = 1; j <= i; ++j) // кол-во различных отношений порядка между N объектами

ans[i] += A[i][j]; // суммирование кол-во различных отношений порядка для значение N

scanf("%d", &n); // scanf\_s - visual

while (n != -1)

{

printf("%d\n", ans[n]); // вывод значения по индексу

scanf("%d", &n); // scanf\_s - visual

}

return 0;

}

**Листинг на языке Python**

n = int(input())

MAXN = 12

# Заполнение массивов нулями

A = [[0 for i in range(MAXN)] for i in range(MAXN)]

ans = [0 for i in range(MAXN)]

A[1][1] = 1

# Заполнение массива A, с использованием формулы для нахождения всех отношений порядка

for i in range(1, MAXN - 1):

for j in range(1, i+1):

A[i + 1][j] += A[i][j] \* j

A[i + 1][j + 1] += A[i][j] \* (j + 1)

for i in range(1, MAXN - 1):

for j in range(1, i+1):

ans[i] += A[i][j]

while n != -1:

print(ans[n])

n = int(input())

**Приложение 5. Задача 1212. Морской бой.**

**Листинг на языке C++**

#include <algorithm>

#include <iostream>

#include <vector>

// Определение структуры Ship, которая содержит координаты (x, y), длину корабля (k) и тип (t).

struct Ship

{

unsigned x;

unsigned y;

unsigned k;

char t;

};

// Функция count принимает вектор позиций кораблей p и целое число k,

// сортирует вектор, удаляет дубликаты и вычисляет количество возможных

// расстановок кораблей с учетом расстояния k.

unsigned count(std::vector<unsigned>& p, unsigned k)

{

std::sort(p.begin(), p.end()); // сортировка элементов от начала до конца

p.erase(std::unique(p.begin(), p.end()), p.end()); // удаление дубликатов. unique - перемещает в конец ; erase - удаляет

unsigned q = 0; // для подсчета результатов

for (size\_t i = 1; i < p.size(); ++i) // перебор элементов вектора p

{

const unsigned d = p[i] - p[i - 1] - 1; // разница между текущим и предыдущим элементом вектора p

if (d >= k) // проверка, больше ли эта разница заданного значения

q += d - k + 1; // добавляем в q

}

return q;

}

// Выполняет подсчет возможных расстановок кораблей вдоль строк и столбцов поля

unsigned solve(unsigned n, unsigned m, const std::vector<Ship>& s, unsigned k)

{

unsigned v = 0;

for (unsigned y = 1; y <= n; ++y) // цикл по всем строкам игрового поля

{

std::vector<unsigned> p = { 0, m + 1 };

for (const Ship& q : s) // для каждого корабля

{

switch (q.t) {

case 'H': // горизонтальный

if (q.y >= y - 1 && q.y <= y + 1) // дипазон координаты у

{

p.push\_back(q.x - 1);

for (unsigned i = 0; i <= q.k; ++i)

p.push\_back(q.x + i);

}

break;

case 'V': // вертикальный

if (q.y <= y + 1 && q.y + q.k >= y) {

p.push\_back(q.x - 1);

p.push\_back(q.x);

p.push\_back(q.x + 1);

}

break;

}

}

v += count(p, k); // считаем количество кораблей в векторе p

}

for (unsigned x = 1; x <= m; ++x) // цикл по всем столбцам игрового поля

{

std::vector<unsigned> p = { 0, n + 1 };

for (const Ship& q : s) // для каждого корабля

{

switch (q.t) {

case 'H': // горизонтальный корабль

if (q.x <= x + 1 && q.x + q.k >= x) // координата находится в пределах

{

p.push\_back(q.y - 1);

p.push\_back(q.y);

p.push\_back(q.y + 1);

}

break;

case 'V': // вертикальный корабль

if (q.x >= x - 1 && q.x <= x + 1) {

p.push\_back(q.y - 1);

for (unsigned i = 0; i <= q.k; ++i)

p.push\_back(q.y + i);

}

break;

}

}

v += count(p, k); // считаем количество кораблей в векторе p

}

return k == 1 ? v / 2 : v; // если значение равно 1 -Ю v/2

}

int main()

{

std::cin.tie(nullptr)->sync\_with\_stdio(false);

unsigned n, m, q;

std::cin >> n >> m >> q; // Считывает значения n (количество строк поля), m (количество столбцов поля) и q (количество кораблей).

std::vector<Ship> s(q);

for (unsigned i = 0; i < q; ++i)

std::cin >> s[i].x >> s[i].y >> s[i].k >> s[i].t;

unsigned k;

std::cin >> k; // Считывает значение k(расстояние между кораблями).

std::cout << solve(n, m, s, k) << '\n';

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Linq;

public class Ship

{

public uint x;

public uint y;

public uint k;

public char t;

}

class Program

{

static uint Count(List<uint> p, uint k)

{

p.Sort();

p = p.Distinct().ToList();

uint q = 0;

for (int i = 1; i < p.Count; i++)

{

uint d = p[i] - p[i - 1] - 1;

if (d >= k)

q += d - k + 1;

}

return q;

}

static uint Solve(uint n, uint m, List<Ship> s, uint k)

{

uint v = 0;

for (uint y = 1; y <= n; y++)

{

List<uint> p = new List<uint> { 0, m + 1 };

foreach (Ship q in s)

{

switch (q.t)

{

case 'H':

if (q.y >= y - 1 && q.y <= y + 1)

{

p.Add(q.x - 1);

for (uint i = 0; i <= q.k; i++)

p.Add(q.x + i);

}

break;

case 'V':

if (q.y <= y + 1 && q.y + q.k >= y)

{

p.Add(q.x - 1);

p.Add(q.x);

p.Add(q.x + 1);

}

break;

}

}

v += Count(p, k);

}

for (uint x = 1; x <= m; x++)

{

List<uint> p = new List<uint> { 0, n + 1 };

foreach (Ship q in s)

{

switch (q.t)

{

case 'H':

if (q.x <= x + 1 && q.x + q.k >= x)

{

p.Add(q.y - 1);

p.Add(q.y);

p.Add(q.y + 1);

}

break;

case 'V':

if (q.x >= x - 1 && q.x <= x + 1)

{

p.Add(q.y - 1);

for (uint i = 0; i <= q.k; i++)

p.Add(q.y + i);

}

break;

}

}

v += Count(p, k);

}

return k == 1 ? v / 2 : v;

}

static void Main(string[] args)

{

string[] input = Console.ReadLine().Split();

uint n = uint.Parse(input[0]);

uint m = uint.Parse(input[1]);

uint q = uint.Parse(input[2]);

List<Ship> s = new List<Ship>();

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

{

input = Console.ReadLine().Split();

Ship ship = new Ship

{

x = uint.Parse(input[0]),

y = uint.Parse(input[1]),

k = uint.Parse(input[2]),

t = char.Parse(input[3])

};

s.Add(ship);

}

uint k = uint.Parse(Console.ReadLine());

Console.WriteLine(Solve(n, m, s, k));

}

}

**Приложение 6. Задача 1247. Проверка последовательности.**

**Листинг на языке C++**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Проверяем является ли заданная последовательность чисел корректной. Суммируем элементы по порядку в обе стороны и

// и проверяем, сумма < i+1 - если условие нарушено хотя бы для одного элемента -> NO

int main()

{

int s, n;

cin >> s >> n;

vector<int> data(s);

for (int i = 0; i < s; i++) // записываем элементы последовательности в вектор

cin >> data[i];

int sum = 0;

for (int i = 0; i < s; i++) // цикл для вычисления суммы элементов вектора

{

sum += data[i];

if (sum < (i + 1)) // если сумма на i-м шаге меньше, чем i + 1 -> выводим "NO"

{

cout << "NO" << endl;

return 0;

}

}

sum = 0; // обнуляем переменную sum

for (int i = 0; i < s; i++) // цикл для вычисления суммы элементов вектора в обратном порядке

{

sum += data[s - 1 - i];

if (sum < (i + 1))

{

cout << "NO" << endl;

return 0;

}

}

cout << "YES" << endl; // проверки пройдены успешно

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

//int s, n;

//s = Convert.ToInt32(Console.ReadLine());

//n = Convert.ToInt32(Console.ReadLine());

int[] A = Array.ConvertAll(Console.ReadLine().Split(), int.Parse); // A[0] - s; a[1] - n

int[] data = new int[A[0]];

for (int i = 0; i < A[0]; i++)

{

data[i] = Convert.ToInt32(Console.ReadLine());

}

int sum = 0;

for (int i = 0; i < A[0]; i++)

{

sum += data[i];

if (sum < (i + 1))

{

Console.WriteLine("NO");

return;

}

}

sum = 0;

for (int i = 0; i < A[0]; i++)

{

sum += data[A[0] - 1 - i];

if (sum < (i + 1))

{

Console.WriteLine("NO");

return;

}

}

Console.WriteLine("YES");

//string input = Console.ReadLine();

}

}

}

**Приложение 7. Задача 1282. Дерево игры.**

**Листинг на языке C++**

#include <iostream>

#include <vector>

using namespace std;

// Структура данных

struct Node

{

vector<size\_t> subnodes;

int value = -1;

};

// Объявление функции solve, которая рекурсивно обходит граф g, начиная с узла x, с учетом параметра m.

// Рекурсивно проходим по дереву, начиная с узла x, и находим оптимальное значение value с учетом переменной m.

// Узлы дерева представлены структурой Node, которая содержит значение узла и список подузлов subnodes.

int solve(vector<Node>& g, size\_t x, int m)

{

if (g[x].subnodes.empty()) // если у текущего узла нет подузлов

return g[x].value; // значение узла

// если у текущего узла есть подузлы - рекурсивно обрабатываем их

int value = -m;

for (const size\_t v : g[x].subnodes)

{

const int t = solve(g, v, -m);

value = m < 0 ? min(value, t) : max(value, t);

}

return value;

}

int main()

{

cin.tie(nullptr)->sync\_with\_stdio(false); // ускорение ввода-вывода

size\_t n;

cin >> n;

vector<Node> g(n); // создание вектора g размера n для хранения узлов графа

for (size\_t i = 1; i < n; ++i) // цикл для заполнения графа g и его узлов subnodes и value.

{

char t;

cin >> t;

unsigned p;

cin >> p;

if (t == 'L')

cin >> g[i].value;

g[p - 1].subnodes.push\_back(i);

}

constexpr const char\* s[3] = { "-1", "0", "+1" }; // constexpr позволяет оптимизировать выполнение программы

cout << s[solve(g, 0, 1) + 1] << '\n'; // вывод на экран значения из массива s в зависимости от результата solve.

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.Linq;

class Node

{

public List<int> subnodes = new List<int>();

public int value = -1;

}

class Program

{

static int Solve(List<Node> g, int x, int m)

{

if (g[x].subnodes.Count == 0)

return g[x-1].value;

int value = -m;

foreach (int v in g[x].subnodes)

{

int t = Solve(g, v, -m);

value = m < 0 ? Math.Min(value, t) : Math.Max(value, t);

}

return value;

}

static void Main()

{

int n = int.Parse(Console.ReadLine());

List<Node> g = new List<Node>();

for (int i = 1; i < n; ++i)

{

string[] input = Console.ReadLine().Split();

char t = char.Parse(input[0]);

int p = int.Parse(input[1]);

if (t == 'L')

g.Add(new Node() { value = int.Parse(input[2]) });

else

g.Add(new Node());

g[p - 1].subnodes.Add(i);

}

string[] s = { "-1", "0", "+1" };

g.Add(new Node());

Console.WriteLine(s[Solve(g, 0, 1) + 1]);

string str = Console.ReadLine();

}

}

**Приложение 8. Задача 1317. Град.**

**Листинг на языке C++**

#include <stdio.h>

#include <vector>

#include <cmath>

#include <numeric> // для использования accumulate()

using namespace std;

using ld = long double; // синоним типа

const ld eps = 1e-9; // очень маленькое значение

struct point

{

ld x, y; // координаты

point operator-(const point& a) const { return { x - a.x, y - a.y }; } // вычитание точек

bool operator==(const point& a) const { return x == a.x && y == a.y; } // сравнение точек

ld len() const { return sqrt(x \* x + y \* y); } // вычисление длины вектора

};

point f[10]; // столбы для забора

point v[100]; // градины

bool ans[100]; // отмечает доступные градины

// Для нахождения пересечения двух отрезков методом Крамера

// определения пересечения текущего сегмента забора с градиной

// это позволяет определить, падает ли градина в область действия лазера

// и уничтожается ли она

void intersect(ld& s, ld& t, point p1, point p2, point q1, point q2)

{

// Решает p1 + s\*p2 = q1 = t\*q2, используя правило Крамера.

point d = q1 - p1;

ld det = q2.x \* p2.y - p2.x \* q2.y;

s = (q2.x \* d.y - d.x \* q2.y) / det;

t = (p2.x \* d.y - d.x \* p2.y) / det;

}

int main()

{

int n, k;

ld h, d, lx, ly, s, t;

// Прочитать все введенные данные

scanf\_s("%d %Lf", &n, &h); // кол-во градин и высота забора

for (int i = 0; i < n; i++) // координаты вершин многоугольника

scanf\_s("%Lf %Lf", &f[i].x, &f[i].y);

scanf\_s("%Lf %Lf %Lf %d", &d, &lx, &ly, &k); // дальность лазера - его координаты - кол-во градин

for (int i = 0; i < k; i++) // координаты точек падения градин

scanf\_s("%Lf %Lf", &v[i].x, &v[i].y);

for (int i = 0; i < n; i++) // для каждого сегмента забора

{

point p1 = f[i], p2 = f[(i + 1) % n] - f[i]; // определение начала и конца вектора для текущего сегмента

for (int j = 0; j < k; j++) // За каждую градину

{

point q1 = { lx, ly }, q2 = (v[j] - q1); // определение начальной точки и направлющего вектора для текущей градины

intersect(s, t, p1, p2, q1, q2); // поиск пересечения текущего сегмента забора с градиной

// Если град прямо над нами, или лазер пересекает какой-то забор

if (q1 == v[j] || s >= -eps && s <= 1 + eps && t >= 0) // проверка совпадение точек и пересечения сегмента с градиной

// Определите минимальную высоту по подобным треугольникам и проверьте, находится ли она в пределах допустимого диапазона.

if (sqrt(h \* h / t / t + q2.len() \* q2.len()) < d + eps) // проверка минимальной высоты

ans[j] = true; // обновление массива ответов

}

}

// accumulate - используется для суммирования всех элементов в указанном диапазоне. Параметры: начальная итерация - конечная итерация - начальное значение

printf("%d", accumulate(ans, ans + 100, 0)); // вывод количества градин, попавших в область действия лазера и уничтоженных

}

**Листинг на языке C#**

using System;

struct Point

{

public double x, y; // координаты

public Point(double x, double y)

{

this.x = x;

this.y = y;

}

public static Point operator -(Point a, Point b)

{

return new Point(a.x - b.x, a.y - b.y); // вычитание точек

}

public static bool operator !=(Point a, Point b)

{

return a.x != b.x && a.y != b.y; // сравнение точек

}

public static bool operator ==(Point a, Point b)

{

return a.x == b.x && a.y == b.y; // сравнение точек

}

public double Length()

{

return Math.Sqrt(x \* x + y \* y); // вычисление длины вектора

}

}

class Program

{

static void Intersect(ref double s, ref double t, Point p1, Point p2, Point q1, Point q2)

{

Point d = new Point(q1.x - p1.x, q1.y - p1.y);

double det = q2.x \* p2.y - p2.x \* q2.y;

s = (q2.x \* d.y - d.x \* q2.y) / det;

t = (p2.x \* d.y - d.x \* p2.y) / det;

}

static void Main()

{

//decimal eps = 1e-9m;//

double eps = 0.000000001;

int n, k;

double h, d, lx, ly, s = 0, t = 0;

// Прочитать все введенные данные

string[] input = Console.ReadLine().Split();

n = int.Parse(input[0]);

h = double.Parse(input[1]);

Point[] f = new Point[n];

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

{

input = Console.ReadLine().Split();

f[i] = new Point(double.Parse(input[0]), double.Parse(input[1]));

}

input = Console.ReadLine().Split();

d = double.Parse(input[0]);

lx = double.Parse(input[1]);

ly = double.Parse(input[2]);

//k = int.Parse(input[3]);

k = int.Parse(Console.ReadLine());

Point[] v = new Point[k];

bool[] ans = new bool[k];

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

{

input = Console.ReadLine().Split();

v[i] = new Point(double.Parse(input[0]), double.Parse(input[1]));

}

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

{

Point p1 = f[i];

Point p2 = f[(i + 1) % n] - f[i];

for (int j = 0; j < k; j++)

{

Point q1 = new Point(lx, ly);

Point q2 = v[j] - q1;

Intersect(ref s ,ref t, p1, p2, q1, q2);

if (q1 == v[j] || (s >= -eps && s <= 1 + eps && t >= 0))

{

if (Math.Sqrt(h \* h / (t \* t) + q2.Length() \* q2.Length()) < d + eps)

{

ans[j] = true;

}

}

}

}

int result = 0;

foreach (bool val in ans)

{

if (val)

{

result++;

}

}

Console.WriteLine(result); // вывод количества градин, попавших в область действия лазера и уничтоженных

//string str = Console.ReadLine();

}

}

**Приложение 9. Задача 1352. Простые числа Мерсенна.**

**Листинг на языке C++**

#include <iostream>

#include <vector>

using namespace std;

int main()

{

// Кол-во тестов

int n;

cin >> n;

// Массив простых чисел Мерсенна

vector<int> data = { 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593, 13466917, 20996011, 24036583, 25964951, 30402457, 32582657, 37156667, 42643801, 43112609, 57885161 };

vector<int> result(n); // массив индексов

// Записываем индексы для теста

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

{

int index;

cin >> index;

result[i] = data[index - 1];

}

// Выводим числа Мерсенна соответствующие индексам

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

{

cout << result[i] << endl;

}

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Numerics;

using System.Threading.Tasks;

namespace T1

{

class Program

{

static void Main(string[] args)

{

int n = int.Parse (Console.ReadLine());

int[] data = new int[]{2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593, 13466917, 20996011, 24036583, 25964951, 30402457, 32582657, 37156667, 42643801, 43112609, 57885161};

int[] result = new int[n];

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

result [i] = data [int.Parse (Console.ReadLine ()) - 1];

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

Console.WriteLine (result [i]);

// string input = Console.ReadLine();

}

}

}

**Листинг на языке Python**

# Кол-во тестов

n = int(input())

# Массив простых чисел Мерсенна

data = [ 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107, 127, 521, 607,

1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209,

44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593, 13466917, 20996011, 24036583, 25964951, 30402457, 32582657, 37156667, 42643801, 43112609, 57885161 ]

result = [] # массив индексов

# Записываем индексы для теста

for i in range(n):

index = int(input())

result.append(data[index - 1])

# Выводим числа Мерсенна соответствующие индексам

for i in range(n):

print(result[i])

**Приложение 10. Задача 1422. Светлячки.**

**Листинг на языке C++**

#include <iostream>

#include <unordered\_map> // неупорядоченный контейнер

using namespace std;

// Рекурсивные функции для нахождения НОД (Наибольший Общий Делитель)

int gcd(int a, int b) { return a == 0 ? b : gcd(b % a, a); }

int gcd(int a, int b, int c) { return gcd(a, gcd(b, c)); }

// Определение структуры vector3D, представляющей трехмерный вектор

struct vector3D

{

// Координаты

int x, y, z;

// Перегрузка операторов вычитания, деления и сравнения для векторов.

vector3D operator-(const vector3D& v2) { return { x - v2.x, y - v2.y, z - v2.z }; };

vector3D operator/(int d) { return { x / d, y / d, z / d }; };

bool operator==(const vector3D& v2) const { return x == v2.x && y == v2.y && z == v2.z; };

// Определение внутренней структуры для хэширования векторов

// XOR = ^ ; 1010 XOR 1100 = 0110 т.е 1 - когда 2 бита различны и 0 - когда одинаковы

// Алгоритм работы функции хеширования заключается в том, что каждая координата вектора (x, y, z)

// сначала хешируется отдельно с использованием стандартной функции хеширования для целых чисел

// (hash<int>()), а затем результаты объединяются с использованием операций XOR (^) и побитового

// сдвига (<<). Результат функции хеширования для вектора v будет состоять из трех хэш-значений

// для каждой из координат, объединенных в одно и возвращенных в качестве общего хэш-значения.

struct Hash

{

// Перегруженные оператор принимает объект типа vector3D и возвращает хэш-значение

size\_t operator()(const vector3D& v) const

{

return hash<int>()(v.x) ^ (hash<int>()(v.y) << 11) ^ (hash<int>()(v.z) << 22);

}

};

};

int main()

{

// Оптимизация

ios::sync\_with\_stdio(false); // отключение синхронизации потоков ввода / вывода

cin.tie(nullptr); // установка связи между потоками ввода и вывода на nullptr

const int SIZE = 2000;

vector3D A[SIZE];

int n;

int r = 1;

cin >> n; // ввод кол-ва объектов в пространстве

// Ввод координат векторов в массив A

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

cin >> A[i].x >> A[i].y >> A[i].z;

for (int i = 0; i < n; i++) // цикл по всем векторам в массиве

{

// unordered\_map параметры (ключ типа "vector3D" - ключ типа int - хэш-функция типа "vector3D::Hash")

// ассоциативный контейнер хранить пары ключ-значение а хеш-функция используется для быстрого доступа к элементам в "unordered\_map" на основе ключей

// Хэш-функция в unordered\_map используется для определения индекса (хеша) каждого элемента в

// хэш-таблице. Она позволяет быстро перемещать или искать элементы в контейнере. При

// использовании хэш-функции скорость выполнения операции вставки, поиска и удаления О(1)

unordered\_map<vector3D, int, vector3D::Hash> m; // ассоциативный массив

// Цикл по всем остальным векторам для нахождения НОД и обновления максимального

// значения "r".

for (int j = 0; j < n; j++)

{

if (i == j)

continue;

vector3D v2 = A[j] - A[i]; // вычисление разности векторов

vector3D g = v2 / gcd(v2.x, v2.y, v2.z); // вычисление вектора с единичными координатами

r = max(r, 1 + ++m[g]); // обновление значения "r" с учетом нахождения одинаковых единичных векторов

}

}

cout << r; // вывод максимального количества точек

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

class Program

{

static int Gcd(int a, int b)

{

return a == 0 ? b : Gcd(b % a, a);

}

static int Gcd(int a, int b, int c)

{

return Gcd(a, Gcd(b, c));

}

// Определение структуры Vector3D, представляющей трехмерный вектор

public struct Vector3D

{

// Координаты

public int x, y, z;

// Перегрузка операторов вычитания, деления и сравнения для векторов.

public static Vector3D operator -(Vector3D v1, Vector3D v2)

{

return new Vector3D { x = v1.x - v2.x, y = v1.y - v2.y, z = v1.z - v2.z };

}

public static Vector3D operator /(Vector3D v, int d)

{

return new Vector3D { x = v.x / d, y = v.y / d, z = v.z / d };

}

public static bool operator ==(Vector3D v1, Vector3D v2)

{

return v1.x == v2.x && v1.y == v2.y && v1.z == v2.z;

}

public static bool operator !=(Vector3D v1, Vector3D v2)

{

return !(v1 == v2);

}

public override bool Equals(object obj)

{

if (obj is Vector3D)

{

return this == (Vector3D)obj;

}

return false;

}

// Хэш-функция для вектора

public override int GetHashCode()

{

return x.GetHashCode() ^ (y.GetHashCode() << 11) ^ (z.GetHashCode() << 22);

}

}

static void Main()

{

const int SIZE = 2000;

Vector3D[] A = new Vector3D[SIZE];

int n;

int r = 1;

n = int.Parse(Console.ReadLine()); // ввод кол-ва объектов в пространстве

// Ввод координат векторов в массив A

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

{

string[] inputs = Console.ReadLine().Split(' ');

A[i] = new Vector3D { x = int.Parse(inputs[0]), y = int.Parse(inputs[1]), z = int.Parse(inputs[2]) };

}

for (int i = 0; i < n; i++) // цикл по всем векторам в массиве

{

Dictionary<Vector3D, int> m = new Dictionary<Vector3D, int>();

// Цикл по всем остальным векторам для нахождения НОД и обновления максимального значения "r".

for (int j = 0; j < n; j++)

{

if (i == j)

continue;

Vector3D v2 = A[j] - A[i]; // вычисление разности векторов

Vector3D g = v2 / Gcd(v2.x, v2.y, v2.z); // вычисление вектора с единичными координатами

if (m.ContainsKey(g))

{

m[g]++;

r = Math.Max(r, 1 + m[g]); // обновление значения "r" с учетом нахождения одинаковых единичных векторов

}

else

{

m.Add(g, 1);

r = Math.Max(r, 2); // обновление значения "r" при добавлении нового единичного вектора

}

}

}

Console.WriteLine(r); // вывод максимального количества точек

}

}

**Приложение 11. Задача 1457. Теплотрасса.**

**Листинг на языке C++**

#include <iostream>

#include <cstdio>

using namespace std;

// scanf\_s -> scanf

int main()

{

int N; // кол-во цифр

int p; // число

double sum = 0;

scanf\_s("%d", &N);

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

{

scanf\_s("%d", &p);

sum += (double)p;

}

sum = sum / (double)N;

printf("%.6lf", sum); // выводим 6 знаков после запятой

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

int n = int.Parse(Console.ReadLine()); // вводим кол-во чисел

int[] p = Array.ConvertAll(Console.ReadLine().Split(), int.Parse); // вводим сами числа

Console.WriteLine((double)p.Sum() / n); // выводим среднее арифметическое

//string input = Console.ReadLine();

}

}

}

**Листинг на языке Python**

import sys

n = int(sys.stdin.readline()) # вводим кол-во чисел

p = [int(x) for x in sys.stdin.readline().split()] # вводим сами числа

print(sum(p) / n) # выводим среднее арифметическое

**Приложение 12. Задача 1562. Ананас.**

**Листинг на языке C++**

#define \_USE\_MATH\_DEFINES // Для включения математических констант (M\_PI - число ПИ)

#include <iostream>

#include <math.h> // <cmath>

using namespace std;

// Формулы для вычисления размера части ананаса

double pieceSize(double x1, double x2, double bh, double ah)

{

return M\_PI \* pow(ah, 2) \* ((x2 - x1) + 1 / (3.0 \* pow(bh, 2)) \* (pow(x1, 3) - pow(x2, 3)));

}

int main()

{

double a, b;

int n;

// a - ширина ананаса (см)

// b - высота ананаса (см)

// n - кол-во кусков

cin >> a >> b >> n;

// Вычисляем шаг и начальное значение x

double dx = b / n;

double x = -b / 2 + dx;

// Выводит вес каждого куска в граммах с точностью до микрограмма в порядке их отрезания

while (x <= b / 2 + 0.000001) // идем до середины ширины, поскольку дальше результаты симметричны

{

//cout << pieceSize(x - dx, x, b / 2, a / 2) << endl;

printf\_s("%.6f\n", pieceSize(x - dx, x, b / 2, a / 2)); // printf - visual ; printf\_s - site

x += dx;

}

return 0;

}

**Листинг на языке C#**

using System;

class Program

{

const double M\_PI = Math.PI; // Число ПИ

// Формулы для вычисления размера части ананаса

static double PieceSize(double x1, double x2, double bh, double ah)

{

return M\_PI \* Math.Pow(ah, 2) \* ((x2 - x1) + 1 / (3.0 \* Math.Pow(bh, 2)) \* (Math.Pow(x1, 3) - Math.Pow(x2, 3)));

}

static void Main()

{

double a, b;

int n;

// a - ширина ананаса (см)

// b - высота ананаса (см)

// n - кол-во кусков

string[] input = Console.ReadLine().Split(' ');

a = double.Parse(input[0]);

b = double.Parse(input[1]);

n = int.Parse(input[2]);

// Вычисляем шаг и начальное значение x

double dx = b / n;

double x = -b / 2 + dx;

// Выводит вес каждого куска в граммах с точностью до микрограмма в порядке их отрезания

while (x <= b / 2 + 0.000001) // идем до середины ширины, поскольку дальше результаты симметричны

{

//Console.WriteLine(pieceSize(x - dx, x, b / 2, a / 2));

Console.WriteLine("{0:F6}", PieceSize(x - dx, x, b / 2, a / 2));

x += dx;

}

string str = Console.ReadLine();

}

}

**Приложение 13. Задача 1737. Мнемоника и палиндромы 3.**

**Листинг на языке C++**

#include <iostream>

#include <string>

#include <queue> // Подключение библиотеки для работы с очередью

using namespace std;

int main()

{

// Создание очереди строк

queue<string> result;

// Кол-во букв

int n; cin >> n;

// Добавляем строки в очередь

result.push("a");

result.push("b");

result.push("c");

// Создание массива символов

char abc[] = { 'a', 'b', 'c' };

for (int i = 1; i < n; i++) // от 0 до n-1

{

while (true)

{

// Получение ссылки на первый элемент очереди

string& s = result.front(); // string s

// Если размер строки больше i выходим из цикла

if (s.size() > i)

break;

// Получение последнего символа строки

char last = s[s.size() - 1];

if (s.size() > 1)

{

// Получение предпоследнего символа строки

char lbo = s[s.size() - 2];

for (int k = 0; k < 3; k++)

{

// Получение k-ого символа из массива abc

char c = abc[k];

// Если последний и предпоследний символы не равны символу "с" добавляем новую строку в очередь

if (last != c && lbo != c)

result.push(s + c);

}

}

else

{

for (int k = 0; k < 3; k++) // Цикл с k от 0 до 2

{

// Получение k-ого символа из массива abc

char c = abc[k];

// Если последний символ не равен символу "c" добавляем новую строку в очередь

if (last != c)

result.push(s + c);

}

}

// Удаление первого элемента из очереди

result.pop();

}

// Если размер очереди умноженный на (i+1) больше 100000

if ((i + 1) \* result.size() > 100000)

{

cout << "TOO LONG";

return 0;

}

}

// Пока очередь не пуста, выводим первый элемент очереди и удаляем его

while (!result.empty())

{

cout << result.front() << endl;

result.pop();

}

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

// Создание очереди строк

Queue<string> result = new Queue<string>();

// Кол-во букв

int n = int.Parse(Console.ReadLine());

// Добавляем строки в очередь

result.Enqueue("a");

result.Enqueue("b");

result.Enqueue("c");

// Создание массива символов

char[] abc = { 'a', 'b', 'c' };

for (int i = 1; i < n; i++)

{

while (true)

{

// Получение ссылки на первый элемент очереди

string s = result.Peek();

// Если размер строки больше i выходим из цикла

if (s.Length > i)

break;

// Получение последнего символа строки

char last = s[s.Length - 1];

if (s.Length > 1)

{

// Получение предпоследнего символа строки

char lbo = s[s.Length - 2];

for (int k = 0; k < 3; k++)

{

// Получение k-ого символа из массива abc

char c = abc[k];

// Если последний и предпоследний символы не равны символу "с" добавляем новую строку в очередь

if (last != c && lbo != c)

result.Enqueue(s + c);

}

}

else

{

for (int k = 0; k < 3; k++)

{

// Получение k-ого символа из массива abc

char c = abc[k];

// Если последний символ не равен символу "c" добавляем новую строку в очередь

if (last != c)

result.Enqueue(s + c);

}

}

// Удаление первого элемента из очереди

result.Dequeue();

}

// Если размер очереди умноженный на (i+1) больше 100000

if ((i + 1) \* result.Count > 100000)

{

Console.WriteLine("TOO LONG");

return;

}

}

// Пока очередь не пуста, выводим первый элемент очереди и удаляем его

while (result.Count > 0)

{

Console.WriteLine(result.Peek());

result.Dequeue();

}

//string input = Console.ReadLine();

}

}

}

**Приложение 14. Задача 1877. Велосипедные коды.**

**Листинг на языке C++**

#include <iostream>

using namespace std;

int main()

{

int codeLock\_1, codeLock\_2;

cin >> codeLock\_1 >> codeLock\_2;

if (codeLock\_1 % 2 == 0 || codeLock\_2 % 2 != 0)

cout << "yes" << endl;

else

cout << "no" << endl;

return 0;

}

**Листинг на языке C#**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.InteropServices;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

internal class Program

{

static void Main(string[] args)

{

int codeLock\_1, codeLock\_2;

codeLock\_1 = Convert.ToInt32(Console.ReadLine());

codeLock\_2 = Convert.ToInt32(Console.ReadLine());

if (codeLock\_1 % 2 == 0 || codeLock\_2 % 2 != 0)

Console.WriteLine("yes");

else

Console.WriteLine("no");

// string input = Console.ReadLine();

}

}

}

**Листинг на языке Python**

codeLock\_1 = int(input())

codeLock\_2 = int(input())

if codeLock\_1 % 2 == 0 or codeLock\_2 % 2 != 0:

print("yes")

else:

print("no")

**Приложение 15. Задача 1982. План электрификации.**

**Листинг на языке C++**

#include <iostream>

#include <limits.h>

using namespace std;

int main()

{

int n, k; // n - количество городов, k - количество городов, в которых уже есть электростанции

cin >> n >> k;

const int N = 100;

int powerstation[N + 1]; // информация о наличии электростанций

int graph[N + 1][N + 1]; // граф связей между городами

int vertices[N + 1], processed[N + 1]; // обработанные вершины

// Предположим, что ни в одном городе нет электростанции

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

powerstation[i] = 0, vertices[i] = INT\_MAX, processed[i] = 0;

// Ввод номеров городов с электостанциями

for (int i = 0, temp = 0; i < k; i++)

{

cin >> temp;

powerstation[temp] = 1;

}

// Получение входных данных -> Заполнение таблицы {Cij} связей между городами

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= n; j++)

{

cin >> graph[i][j];

}

}

// Маршруты между электростанциями установлены на ноль, потому что на самом деле они

// не требуются, поэтому не увеличивают стоимость.

// Устанавливаются маршруты между электростанциями на ноль

for (int i = 1; i <= n; i++)

{

if (powerstation[i] == 1)

{

for (int j = 1; j <= n; j++)

{

if (powerstation[j] == 1)

graph[i][j] = 0;

if (i == j)

graph[i][j] = INT\_MIN;

}

}

}

// Находится минимальное остовное дерево (MST) с использованием алгоритма Прима.

// Строит остовное дерево с минимальным весом вершин. Остовное дерево используется для поиска

// оптимального пути или связи между вершинами.

vertices[1] = 0;

for (int i = 1, temp = INT\_MAX, iter = 0; i < n; i++)

{

// Нахождение вершины с минимальным весом

for (int j = 1; j <= n; j++)

{

if (vertices[j] < temp && processed[j] == 0)

temp = vertices[j], iter = j;

}

processed[iter] = 1;

// Обновление массива вершин

for (int j = 1; j <= n; j++)

{

if (processed[j] == 0)

if (graph[iter][j] >= 0 && graph[iter][j] < vertices[j])

vertices[j] = graph[iter][j];

}

temp = INT\_MAX;

}

// Находится общая стоимость электрификации всех городов и выводится на экран

int u = 0;

for (int i = 1; i <= n; i++)

u += vertices[i];

cout << u;

return 0;

}

**Листинг на языке C#**

using System;

public class Program

{

static void Main()

{

string [] input;

string str;

str = Console.ReadLine();

input = str.Split(' ');

int n, k;

n = Convert.ToInt32(input[0].Trim());

k = Convert.ToInt32(input[1].Trim());

const int N = 100;

int[] powerstation = new int[N + 1];

int[,] graph = new int[N + 1, N + 1];

int[] vertices = new int[N + 1];

int[] processed = new int[N + 1];

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

{

powerstation[i] = 0;

vertices[i] = int.MaxValue;

processed[i] = 0;

}

str = Console.ReadLine();

input = str.Split(' ');

for (int i = 0, temp = 0; i < k; i++)

{

//temp = Convert.ToInt32(Console.ReadLine());

temp = Convert.ToInt32(input[i]);

powerstation[temp] = 1;

}

for (int i = 1; i <= n; i++)

{

str = Console.ReadLine();

input = str.Split(' ');

for (int j = 1; j <= n; j++)

{

graph[i, j] = Convert.ToInt32(input[j-1]);

}

}

for (int i = 1; i <= n; i++)

{

if (powerstation[i] == 1)

{

for (int j = 1; j <= n; j++)

{

if (powerstation[j] == 1)

graph[i, j] = 0;

if (i == j)

graph[i, j] = int.MinValue;

}

}

}

vertices[1] = 0;

for (int i = 1, temp = int.MaxValue, iter = 0; i < n; i++)

{

for (int j = 1; j <= n; j++)

{

if (vertices[j] < temp && processed[j] == 0)

{

temp = vertices[j];

iter = j;

}

}

processed[iter] = 1;

for (int j = 1; j <= n; j++)

{

if (processed[j] == 0)

{

if (graph[iter, j] >= 0 && graph[iter, j] < vertices[j])

{

vertices[j] = graph[iter, j];

}

}

}

temp = int.MaxValue;

}

int u = 0;

for (int i = 1; i <= n; i++)

{

u += vertices[i];

}

Console.WriteLine(u);

}

}