**Министерство цифрового развития, связи и массовых**

**коммуникаций Российской Федерации**

**Ордена Трудового Красного Знамени**

**федеральное государственное бюджетное**

**учреждение высшего образования**

**«Московский Технический Университет Связи и Информатики»**

Кафедра «Математическая кибернетика и

информационные технологии»

**КУРСОВАЯ РАБОТА №6**

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент группы БСТ2001

Савкин Д. И.

Проверил: асп. каф. «Математическая кибернетика и

информационные технологии»

Чайка А.Д.

Москва 2022

СОДЕРЖАНИЕ

Цели и задачи 3

Ход выполнения работы 4

«Упорядочивание соседей» (3500 баллов) 4

«Булева функция» (3000 баллов) 5

«Взламываем!» (2500 баллов) 6

«Пара прямых» (2000 баллов) 6

«Заседание жюри» (1500 баллов) 7

«Отопление» (1000 баллов) 8

ПРИЛОЖЕНИЕ А 10

# Цели и задачи

В текущей работе необходимо выбрать задания из банка задач по олимпиадному программированию с Интернет-ресурса Codeforces. Необходимо привести условие каждой задачи, её входные и выходные данные, а так же листинг реализованного программного кода.

# Ход выполнения работы

## «Упорядочивание соседей» (3500 баллов)

**Условие**. Для неориентированного графа назовём порядком соседей упорядоченный список всех соседей вершины для каждой из вершин . Рассмотрим некоторый порядок соседей в графе и три вершины , и такие, что является соседом и . Мы будет использовать запись , если стоит после в списке соседей вершины .

Порядок соседей называется хорошим, если для каждого простого цикла графа выполняется одно из следующих условий:

,

.

Для заданного графа определите, существует ли для него порядок хороших соседей, и постройте его, если он существует.

**Входные данные** состоят из нескольких наборов входных данных. В первой строке записано единственное целое число — количество наборов входных данных. Далее следуют описания наборов входных данных. Первая строка каждого набора входных данных содержит два целых числа и , количество вершин и количество рёбер графа. Следующие строк содержат по два целых числа , , обозначающих, что существует ребро, соединяющее вершины и . Гарантируется, что граф связный и в нем нет петель и кратных рёбер. Сумма и сумма для всех тестовых случаев не превосходят .

**Выходные данные**. Для каждого набора входных данных выведите одну строку с «YES», если граф допускает хороший порядок соседей, иначе выведите одну строку с «NO». Вы можете печатать каждую букву в любом регистре (верхнем или нижнем). Если ответ «YES», дополнительно выведите строк, описывающих порядок хороших соседей. В -й строке выведите соседей вершины по порядку.

## «Булева функция» (3000 баллов)

**Условие**. В данной задаче рассматриваются булевы функции от четырёх переменных , , , . Переменные , ,  и являются логическими и могут принимать значения 0 или 1. Будем задавать функцию с помощью следующей грамматики:

<выражение> ::= <переменная> | (<выражение>) <оператор> (<выражение>)

<переменная> ::= 'A' | 'B' | 'C' | 'D' | 'a' | 'b' | 'c' | 'd'

<оператор> ::= '&' | '|'

Здесь заглавными буквами , , ,  обозначаются переменные, а строчными — их отрицания. Например, если , то символу соответствует значение 1, а символу — значение 0. Символ здесь соответствует операции логического , символ — операции логического .

Вам дано выражение , задающее функцию , в котором некоторые операторы и переменные пропущены. Так же вам известны значения функции для некоторых различных наборов значений переменных. Посчитайте количество способов восстановить пропущенные в выражении элементы так, чтобы получившееся выражение соответствовало известной информации о функии в заданных наборах переменных. Так как величина результата может быть большой, выведите её остаток от деления на .

**Входные данные**. В первой строке содержится выражение , в котором некоторые символы операторов и/или переменных заменены знаком . Во второй строке содержится число — количество наборов переменных, для которых известно значение функции . В следующих строках содержатся описания наборов: в -й из них содержатся пять целых чисел , , , ,  , разделённых пробелами и означающих, что . Гарантируется, что все четвёрки различны.

**Выходные данные**. В единственной строке выведите ответ на задачу.

## «Взламываем!» (2500 баллов)

**Условие**. Little X недавно наткнулся на следующую задачу. Определим как сумму цифр в десятичной записи числа (например, ). Ваша задача — посчитать . Конечно, Little X быстро решил эту задачу, заблокировал решение и стал взламывать чужие решения. Открыв решение участника, он увидел следующий C++ код:



Этот код будет работать неправильно только на тесте с . Вам дано число , помогите Little X найти подходящий тест для взлома.

**Входные данные**. В первой строке записано единственное целое число .

**Выходные данные**. Выведите два целых числа: ,  — данные для взлома. Не выводите ведущих нулей. Гарантируется, что решение существует.

## «Пара прямых» (2000 баллов)

**Условие**. Задано точек на плоскости, все точки имеют целочисленные координаты. Все точки различны. Необходимо узнать, существуют ли две такие прямые, что каждая из заданных точек принадлежит хотя бы одной из них. Прямые могут совпадать.

**Входные данные**. В первой строке задано одно целое число — количество точек на плоскости. Далее в строках заданы по два целых числа и — координаты -й точки. Все точек различны.

**Выходные данные**. Если возможно провести две прямые так, что каждая точка лежит хотя бы на одной из них, то выведите YES, иначе NO.

## «Заседание жюри» (1500 баллов)

**Условие**. человек собрались, чтобы провести заседание жюри предстоящего соревнования, -й член жюри придумал задач, которыми он хочет поделиться с другими. Сначала жюри выбирает порядок, которому они будут следовать при обсуждении задач. Пусть это будет перестановка чисел от 1 до (массив размера , в котором каждое число от 1 до встречается ровно один раз). Затем обсуждение происходит следующим образом:

* если у члена жюри остались задачи, которые нужно рассказать, то он рассказывает одну задачу. В противном случае он будет пропущен.
* если у члена жюри остались задачи, которые нужно рассказать, то он рассказывает одну задачу. В противном случае он будет пропущен.
* …
* если у члена жюри остались задачи, которые нужно рассказать, то он рассказывает одну задачу. В противном случае он будет пропущен.
* если остались члены жюри, которые ещё не рассказывали все свои задачи, то процесс повторяется с самого начала. В противном случае обсуждение заканчивается.

Перестановка хорошая, если никто из членов жюри не будет рассказывать две или более задач подряд. Подсчитайте количество хороших перестановок. Ответ может быть очень большим, поэтому выведите его по модулю 998244353.

**Входные данные**. Первая строка содержит одно целое число — количество наборов входных данных. Первая строка набора входных данных содержит одно целое число — количество членов жюри. Вторая строка содержит n целых чисел — количество задач, которые придумал -й член жюри. Сумма по всем наборам входных данных не превосходит .

**Выходные данные**. Для каждого набора входных данных выведите одно целое число — количество хороших перестановок, взятое по модулю 998244353.

## «Отопление» (1000 баллов)

**Условие**. Несколько дней назад вы приобрели новый дом и сейчас планируете провести в нем капитальный ремонт. Так как зимы в вашем регионе могут быть очень суровыми, вам нужно определиться с системой отопления в каждой комнате. У вас в доме комнат. В -й комнате вы можете установить не более радиаторов системы отопления. Каждый радиатор может иметь несколько секций, но стоимость радиатора из секций равна бурлей. Так как комнаты имеют разные размеры, то вы рассчитали, что на обогрев -й комнаты вам понадобится как минимум секций суммарно по всем радиаторам. Для каждой комнаты посчитайте минимально возможную стоимость установки не более радиаторов и суммарным количеством секций не менее .

**Входные данные**. В первой строке задано единственное число n — количество комнат. В каждой из следующих строк заданы описания комнат. В -й строке задано два целых числа и — максимальное количество радиаторов и минимальное необходимое количество секций в -й комнате, соответственно.

**Выходные данные**. Для каждой комнаты выведите одно число — минимально возможная стоимость установки не более радиаторов с суммарным количеством секций не менее чем .

# ПРИЛОЖЕНИЕ А

Программный код решения задачи «Упорядочивание соседей»:

import functools  
  
  
def task\_01():  
 # https://codeforces.com/problemset/problem/1656/I  
 print("=== TASK 1 ===")  
 read = input()  
 for \_ in range(int(read)):  
 input\_graph = read\_graph()  
 ans: list[list[int]] = solve(input\_graph)  
 print\_ans(ans)  
  
  
class Graph:  
 def \_\_init\_\_(self):  
 self.n: int = 0  
 self.m: int = 0  
 self.al: list[list[int]] = []  
 self.morphism: list[int] = []  
 self.dfs\_children: list[list[int]] = []  
 self.dfs\_parent: list[int] = []  
 self.dfs\_num: list[int] = []  
 self.dfs\_low: list[int] = []  
 self.dfs\_count: int = 0  
 self.is\_root\_ac: bool = False  
 self.bad\_biccon: bool = False  
 self.repr\_edge: list[tuple[int, int]] = []  
  
 def dt\_dfs(self, v: int, par: int):  
 self.dfs\_parent[v] = par  
 self.dfs\_low[v] = self.dfs\_count  
 self.dfs\_num[v] = self.dfs\_count  
 self.dfs\_count += 1  
 for u in self.al[v]:  
 if u == par:  
 pass  
 elif self.dfs\_num[u] == -1:  
 self.dfs\_children[v].append(u)  
 self.dt\_dfs(u, v)  
 self.dfs\_low[v] = min(self.dfs\_low[v], self.dfs\_low[u])  
 else:  
 self.dfs\_low[v] = min(self.dfs\_low[v], self.dfs\_num[u])  
  
 def min\_repr\_edge(self, a: tuple[int, int], b: tuple[int, int]) -> tuple[int, int]:  
 if a[0] == -1:  
 return b  
 if b[0] == -1:  
 return a  
 if self.dfs\_num[a[1]] < self.dfs\_num[b[1]]:  
 return a  
 if self.dfs\_num[a[1]] > self.dfs\_num[b[1]]:  
 return b  
 if self.dfs\_num[a[0]] < self.dfs\_num[b[0]]:  
 return a  
 if self.dfs\_num[a[0]] < self.dfs\_num[b[0]]:  
 return a  
 return a  
  
 def dt\_dfs\_hamil(self, v: int):  
 self.repr\_edge[v] = (-1, -1)  
 for u in self.dfs\_children[v]:  
 self.dt\_dfs\_hamil(u)  
 self.repr\_edge[v] = self.min\_repr\_edge(self.repr\_edge[v], self.repr\_edge[u])  
  
 for u in self.al[v]:  
 if u == self.dfs\_parent[v]:  
 pass  
 elif self.dfs\_num[u] < self.dfs\_num[v]:  
 self.repr\_edge[v] = self.min\_repr\_edge(self.repr\_edge[v], (v, u))  
  
 if self.dfs\_parent[v] == -1:  
 pass  
 else:  
 if len(self.dfs\_children[v]) == 0:  
 pass  
 elif len(self.dfs\_children[v]) == 1:  
 if self.dfs\_low[self.dfs\_children[v][0]] != self.dfs\_low[v] and self.dfs\_low[self.dfs\_children[v][0]] != \  
 self.dfs\_num[self.dfs\_parent[v]]:  
 self.bad\_biccon = True  
 elif len(self.dfs\_children[v]) == 2:  
 if self.dfs\_parent[self.dfs\_parent[v]] != -1:  
 if not ((self.dfs\_low[self.dfs\_children[v][0]] == self.dfs\_num[self.dfs\_parent[v]]) ^ (  
 self.dfs\_low[self.dfs\_children[v][1]] == self.dfs\_num[self.dfs\_parent[v]])):  
 self.bad\_biccon = True  
 if self.dfs\_low[v] < min(self.dfs\_low[self.dfs\_children[v][0]],  
 self.dfs\_low[self.dfs\_children[v][1]]):  
 self.bad\_biccon = True  
 else:  
 self.bad\_biccon = True  
  
 def dt\_dfs\_cedges(self, cedges: list[list[tuple[int, int]]], edge\_stack: list[tuple[int, int]], v: int, par: int):  
 self.dfs\_parent[v] = par  
 self.dfs\_low[v] = self.dfs\_count  
 self.dfs\_num[v] = self.dfs\_count  
 self.dfs\_count += 1  
 for u in self.al[v]:  
 if u == par:  
 pass  
 elif self.dfs\_num[u] == -1:  
 self.dfs\_children[v].append(u)  
 edge\_stack.append((v, u))  
 self.dt\_dfs\_cedges(cedges, edge\_stack, u, v)  
 self.dfs\_low[v] = min(self.dfs\_low[v], self.dfs\_low[u])  
 if (par == -1 and self.is\_root\_ac) or (par != -1 and self.dfs\_low[u] >= self.dfs\_num[v]):  
 comp: list[tuple[int, int]] = []  
 while edge\_stack[-1][0] != v or edge\_stack[-1][1] != u:  
 comp.append(edge\_stack[-1])  
 edge\_stack.pop()  
 comp.append(edge\_stack[-1])  
 edge\_stack.pop()  
 cedges.append(comp)  
 else:  
 self.dfs\_low[v] = min(self.dfs\_low[v], self.dfs\_num[u])  
 if self.dfs\_num[u] < self.dfs\_num[v]:  
 edge\_stack.append((v, u))  
  
 def generate\_dfs\_tree(self, root: int):  
 self.dfs\_children = [[] for \_ in range(self.n)]  
 self.dfs\_parent = [0 for \_ in range(self.n)]  
 self.dfs\_count = 0  
 self.dfs\_num = [-1 for \_ in range(self.n)]  
 self.dfs\_low = [-1 for \_ in range(self.n)]  
 self.dt\_dfs(root, -1)  
  
 def generate\_edge\_partition(self, cedges: list[list[tuple[int, int]]], edge\_stack: list[tuple[int, int]],  
 root: int):  
 self.generate\_dfs\_tree(root)  
 self.is\_root\_ac = len(self.dfs\_children[root]) > 1  
 self.dfs\_children = [[] for \_ in range(self.n)]  
 self.dfs\_parent = [0 for \_ in range(self.n)]  
 self.dfs\_count = 0  
 self.dfs\_num = [-1 for \_ in range(self.n)]  
 self.dfs\_low = [-1 for \_ in range(self.n)]  
 self.dt\_dfs\_cedges(cedges, edge\_stack, root, -1)  
 if edge\_stack:  
 cedges.append(edge\_stack)  
  
 def generate\_hamil\_dfs\_tree(self, root: int):  
 self.generate\_dfs\_tree(root)  
 self.repr\_edge = [(0, 0) for \_ in range(self.n)]  
 self.dt\_dfs\_hamil(root)  
  
  
def partition\_biconnected(g: Graph) -> list[Graph]:  
 cedges: list[list[tuple[int, int]]] = []  
 edge\_stack: list[tuple[int, int]] = []  
 g.generate\_edge\_partition(cedges, edge\_stack, 0)  
 comp: list[Graph] = []  
 for vec in cedges:  
 h = Graph()  
 h.n = 0  
 h.m = len(vec)  
 rmorph: dict[int, int] = {}  
 for e in vec:  
 if rmorph.get(e[0]) is None:  
 rmorph[e[0]] = h.n  
 h.n += 1  
 if rmorph.get(e[1]) is None:  
 rmorph[e[1]] = h.n  
 h.n += 1  
 h.morphism = [0 for \_ in range(h.n)]  
 h.al = [[] for \_ in range(h.n)]  
 for e in vec:  
 u = rmorph[e[0]]  
 v = rmorph[e[1]]  
 h.morphism[u] = g.morphism[e[0]]  
 h.morphism[v] = g.morphism[e[1]]  
 h.al[u].append(v)  
 h.al[v].append(u)  
 comp.append(h)  
 return comp  
  
  
def upwards\_path(g: Graph, hc: list[int], v: int, tar: int):  
 if len(g.dfs\_children[v]) == 2:  
 u1: int = g.dfs\_children[v][0]  
 u2: int = g.dfs\_children[v][1]  
 if g.repr\_edge[u1] == g.repr\_edge[v]:  
 u1, u2 = u2, u1  
 hc.append(v)  
 downwards\_path(g, hc, u1, g.repr\_edge[u1][0])  
 if v != tar:  
 upwards\_path(g, hc, g.dfs\_parent[v], tar)  
 elif len(g.dfs\_children[v]) == 1:  
 u: int = g.dfs\_children[v][0]  
 hc.append(v)  
 if g.repr\_edge[u] != g.repr\_edge[v]:  
 downwards\_path(g, hc, u, g.repr\_edge[u][0])  
 if v != tar:  
 upwards\_path(g, hc, g.dfs\_parent[v], tar)  
 else:  
 hc.append(v)  
 if v != tar:  
 upwards\_path(g, hc, g.dfs\_parent[v], tar)  
  
  
def downwards\_path(g: Graph, hc: list[int], v: int, tar: int):  
 if len(g.dfs\_children[v]) == 2:  
 u1: int = g.dfs\_children[v][0]  
 u2: int = g.dfs\_children[v][1]  
 if g.repr\_edge[u1] == g.repr\_edge[v]:  
 u1, u2 = u2, u1  
 upwards\_path(g, hc, g.repr\_edge[u1][0], u1)  
 hc.append(v)  
 downwards\_path(g, hc, u2, tar)  
 elif len(g.dfs\_children[v]) == 1:  
 u: int = g.dfs\_children[v][0]  
 if v == tar:  
 upwards\_path(g, hc, g.repr\_edge[u][0], u)  
 hc.append(v)  
 else:  
 hc.append(v)  
 downwards\_path(g, hc, u, tar)  
 else:  
 hc.append(v)  
  
  
def hamiltonian\_cycle(g: Graph) -> list[int]:  
 g.generate\_hamil\_dfs\_tree(0)  
 if g.bad\_biccon:  
 return []  
 hc: list[int] = []  
 downwards\_path(g, hc, 0, g.repr\_edge[0][0])  
 assert len(hc) == g.n  
 return hc  
  
  
comp\_index = 0  
  
  
def cyclic\_comparator(a: int, b: int) -> bool:  
 if a < comp\_index:  
 a += 1e7  
 if b < comp\_index:  
 b += 1e7  
 return a < b  
  
  
def sort\_graph(g: Graph, hc: list[int]) -> Graph:  
 h = Graph()  
 h.n = g.n  
 h.m = g.m  
 h.morphism = [0 for \_ in range(g.n)]  
 h.al = [[] for \_ in range(g.n)]  
 rgc: list[int] = [0 for \_ in range(g.n)]  
 for i in range(g.n):  
 h.morphism[i] = g.morphism[hc[i]]  
 rgc[hc[i]] = i  
 for i in range(g.n):  
 for j in g.al[hc[i]]:  
 h.al[i].append(rgc[j])  
 h.al[i].sort(key=functools.cmp\_to\_key(cyclic\_comparator))  
 return h  
  
  
def has\_crossing(g: Graph) -> bool:  
 bad\_stack: list[tuple[int, int]] = []  
 for i in range(g.n):  
 global comp\_index  
 comp\_index = i  
 while bad\_stack and bad\_stack[-1][0] == i:  
 bad\_stack.pop()  
 j = len(g.al[i]) - 2  
 while j:  
 u = g.al[i][j]  
 if bad\_stack and cyclic\_comparator(bad\_stack[-1][0], u) and cyclic\_comparator(u, bad\_stack[-1][1]):  
 return True  
 if u > i and (not bad\_stack or u != bad\_stack[-1][0]):  
 bad\_stack.append((u, i))  
 j -= 1  
 return False  
  
  
def merge\_ans(g: Graph, ans: list[list[int]]):  
 for i in range(g.n):  
 for j in g.al[i]:  
 ans[g.morphism[i]].append(g.morphism[j])  
  
  
def merge\_single\_edge\_ans(g: Graph, ans: list[list[int]]):  
 u = g.morphism[0]  
 v = g.morphism[1]  
 ans[u].append(v)  
 ans[v].append(u)  
  
  
def solve(input\_graph: Graph) -> list[list[int]]:  
 ans: list[list[int]] = [[] for \_ in range(input\_graph.n)]  
 components: list[Graph] = partition\_biconnected(input\_graph)  
 for g in components:  
 if g.n == 1:  
 pass  
 elif g.n == 2:  
 merge\_single\_edge\_ans(g, ans)  
 else:  
 hc: list[int] = hamiltonian\_cycle(g)  
 if not hc:  
 return []  
 g2: Graph = sort\_graph(g, hc)  
 if has\_crossing(g2):  
 return []  
 merge\_ans(g2, ans)  
 return ans  
  
  
def read\_graph() -> Graph:  
 read = input().split()  
 n = int(read[0])  
 m = int(read[1])  
 g = Graph()  
 g.n = n  
 g.m = m  
 g.al = [[] for \_ in range(n)]  
 g.morphism = [0 for \_ in range(n)]  
 for i in range(n):  
 g.morphism[i] = i  
 for i in range(m):  
 read = input().split()  
 u = int(read[0])  
 v = int(read[1])  
 g.al[u].append(v)  
 g.al[v].append(u)  
 return g  
  
  
def print\_ans(ans: list[list[int]]):  
 n = len(ans)  
 if n == 0:  
 print('NO')  
 else:  
 print('YES')  
 for i in range(n):  
 for x in ans[i]:  
 print(f'{x} ', end='')  
 print()

Программный код решения задачи «Булева функция»:

def task\_02():  
 # https://codeforces.com/problemset/problem/582/E  
 print("=== TASK 2 ===")  
 task\_02\_main()  
  
  
MOD = 1e9 + 7  
N = 200 + 5  
V = 4  
S = 1 << V  
M = 1 << S  
  
s: list[str]  
n = 0  
d: list[list[int]] = [[0 for \_ in range(M)] for \_ in range(N)]  
  
variables = 'ABCDabcd'  
operations = '&|'  
  
req: list[int] = [0 for \_ in range(1 << V)]  
  
  
def add(a: int, b: int) -> int:  
 return a + b - MOD if a + b >= MOD else a + b  
  
  
def sub(a: int, b: int) -> int:  
 return a - b + MOD if a - b < 0 else a - b  
  
  
def mul(a: int, b: int) -> int:  
 return int((a \* b) % MOD)  
  
  
def mask\_one(v: int) -> int:  
 mask = 0  
 for i in range(S):  
 if (i >> v) & 1:  
 mask |= 1 << i  
 return mask  
  
  
def negate(a: list[int]):  
 a[0:M] = a[0:M][::-1]  
  
  
def copy(a: list[int], b: list[int]):  
 for i in range(M):  
 b[i] = a[i]  
  
  
def add1(a: list[int], b: list[int]):  
 for i in range(M):  
 b[i] = add(b[i], a[i])  
  
  
def sum\_sub(a: list[int], b: list[int], d: int):  
 copy(a, b)  
 for i in range(S):  
 for mask in range(M):  
 if not (mask & (1 << i)):  
 if d == +1:  
 b[mask ^ (1 << i)] = add(b[mask ^ (1 << i)], b[mask])  
 elif d == -1:  
 b[mask ^ (1 << i)] = sub(b[mask ^ (1 << i)], b[mask])  
 else:  
 raise ValueError()  
  
  
t1: list[int] = [0 for \_ in range(M)]  
t2: list[int] = [0 for \_ in range(M)]  
tops: list[list[int]] = [[0 for \_ in range(M)] for \_ in range(2)]  
  
  
def door(l: list[int], r: list[int], v: list[int]):  
 sum\_sub(l, t1, +1)  
 sum\_sub(r, t2, +1)  
 for i in range(M):  
 t1[i] = mul(t1[i], t2[i])  
 sum\_sub(t1, v, -1)  
  
  
def parse(pos: int) -> tuple[int, int]:  
 global n  
 if s[pos] != '(':  
 v = variables.find(s[pos])  
 pos += 1  
 if v == -1:  
 for i in range(V):  
 d[n][mask\_one(i)] = 1  
 d[n][(M - 1) ^ mask\_one(i)] = 1  
 else:  
 if v < V:  
 d[n][mask\_one(v)] = 1  
 else:  
 d[n][(M - 1) ^ mask\_one(v - V)] = 1  
 n += 1  
 return pos, n - 1  
 me = n  
 n += 1  
 pos += 1  
 pos, l = parse(pos)  
 assert s[pos] == ')'  
 pos += 1  
 v = operations.find(s[pos])  
 pos += 2  
 pos, r = parse(pos)  
 assert s[pos] == ')'  
 pos += 1  
  
 door(d[l], d[r], tops[1])  
  
 negate(d[l])  
 negate(d[r])  
 door(d[l], d[r], tops[0])  
 negate(tops[0])  
 negate(d[r])  
 negate(d[l])  
  
 if v == -1:  
 for i in range(len(operations)):  
 add1(tops[v], d[me])  
 else:  
 add1(tops[v], d[me])  
 return pos, me  
  
  
def task\_02\_main():  
 global s, n, req  
 read = input()  
 s = [char for char in read]  
 pos = 0  
 pos, root = parse(pos)  
 n = int(input())  
 for i in range(S):  
 req[i] = -1  
 for i in range(n):  
 mask = 0  
 read = input().split()  
 for j in range(V):  
 a = int(read[j])  
 mask |= (a << j)  
 req[mask] = int(read[4])  
 res = 0  
 for i in range(M):  
 good = True  
 for j in range(S):  
 good &= (req[j] == -1 or req[j] == ((i >> j) & 1))  
 if good:  
 res = add(res, d[root][i])  
 print(res)

Программный код решения задачи «Взламываем!»:

def task\_03():  
 # https://codeforces.com/problemset/problem/468/C  
 print("=== TASK 3 ===")  
 m = int(input())  
 x, t = 10 \*\* 100 - 1, m - 100 \* 45 \* 10 \*\* 99 % m  
 print(t, t + x)

Программный код решения задачи «Пара прямых»:

def task\_04():  
 # https://codeforces.com/problemset/problem/961/D  
 print("=== TASK 4 ===")  
 task\_04\_main()  
  
  
N = 200 \* 1000 + 555  
n = 0  
p: list[tuple[int, int]] = [(0, 0) for \_ in range(N)]  
used: list[bool] = [False for \_ in range(N)]  
  
  
def sub(a: tuple[int, int], b: tuple[int, int]) -> tuple[int, int]:  
 return a[0] - b[0], a[1] - b[1]  
  
  
def cross(a: tuple[int, int], b: tuple[int, int]) -> int:  
 return a[0] \* b[1] - a[1] \* b[0]  
  
  
def check() -> bool:  
 i1, i2 = -1, -1  
 for i in range(n):  
 if used[i]:  
 continue  
 if i1 == -1:  
 i1 = i  
 elif i2 == -1:  
 i2 = i  
 if i2 == -1:  
 return True  
 for i in range(n):  
 if used[i]:  
 continue  
 if cross(sub(p[i2], p[i1]), sub(p[i], p[i1])) != 0:  
 return False  
 return True  
  
  
def check2(a: tuple[int, int], b: tuple[int, int]) -> bool:  
 global used  
 used = [False for \_ in range(N)]  
 for i in range(n):  
 if cross(sub(b, a), sub(p[i], a)) == 0:  
 used[i] = True  
 return check()  
  
  
def task\_04\_main():  
 global n  
 n = int(input())  
 for i in range(n):  
 read = input().split()  
 p[i] = (int(read[0]), int(read[1]))  
 if n <= 2:  
 print('YES')  
 return  
 if check2(p[0], p[1]) or check2(p[0], p[2]) or check2(p[1], p[2]):  
 print('YES')  
 else:  
 print('NO')

Программный код решения задачи «Заседание жюри»:

MOD = 998244353  
  
  
def task\_05():  
 # https://codeforces.com/problemset/problem/1569/C  
 print("=== TASK 5 ===")  
 t = int(input())  
 for \_ in range(t):  
 n = int(input())  
 read = input().split()  
 a = [int(x) for x in read]  
 mx = max(a)  
 cmx = a.count(mx)  
 k = a.count(mx - 1)  
 ans = 1  
 sub = 1  
 for i in range(1, n + 1):  
 ans = ans \* i % MOD  
 if i != k + 1:  
 sub = sub \* i % MOD  
 if cmx == 1:  
 ans = (ans - sub + MOD) % MOD  
 print(ans)

Программный код решения задачи «Отопление»:

def task\_06():  
 # https://codeforces.com/problemset/problem/1260/A  
 print("=== TASK 6 ===")  
 n = int(input())  
 for \_ in range(n):  
 read = input().split()  
 c, summary = int(read[0]), int(read[1])  
 d = summary // c  
 rem = summary % c  
 result = rem \* (d + 1) \* (d + 1) + (c - rem) \* d \* d  
 print(result)