подтверждение их участия в финале.

§3 Заключительный этап: индивидуальная часть

Заключительный этап олимпиады состоит из двух частей: индивидуальное решение задач по предметам (математика, информатика) и командное решение инженерное задачи. Для каждой из параллелей (9 класс или 10-11 класс) предлагается свой набор задач по математике, информатика общая для всех классов. На индивидуальное решение задач дается по 2 часа на один предмет.
Решение каждой задачи дает определенное количество баллов (см. критерии оценки). По математике за каждую задачу можно получить от 0 до указанного количества баллов. Баллы по информатике зачисляются в полном объеме за правильное решение задачи. Решение задачи по информатике подразумевает написание программы на языке Python или C++. Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 200 баллов.

3.1 Первая попытка
Задачи по математике (9 класс)

Задача 3.1.1 (15 баллов)
Условие:
Найти все значения x и y, удовлетворяющие равенству

$$x - xy + y = 1$$

Решение:
Данное равенство эквивалентно следующему: $(x - 1)(y - 1) = 0$.

Ответ:
одно из чисел равно 1, а другое произвольно.

Критерии оценки:
0 баллов - нет ответа, не получено равенство, неправильная логика рассуждений;
1-7 баллов - неправильный ответ, получено неверное эквивалентное равенство, верный ход рассуждений;
8-14 баллов - неправильный ответ, верный ход рассуждений, получено верное эквивалентное равенство; правильный ответ, верный ход рассуждений, не получено равенство
15 баллов - правильный ответ, получено эквивалентное равенство.

Задача 3.1.2 (20 баллов)
Условие:

Разработчики переходят из проекта в проект, причем работа в каждом следующем проекте либо интереснее, либо лучше оплачивается.
Одному разработчику повезло - в его новом проекте одновременно более интересные задачи и более высокая зарплата.
Могло ли такое быть, если на одно место в проекте претендует один кандидат? Ответ объясните.

Решение:

Пусть есть три проекта — А, Б и В. Пусть по сложности задач проекты расположены в порядке В, А, Б (Б — самый сложный), а по зарплате — в порядке Б, В, А (А — самый дорогой). Предположим, что разработчик, работавший в А, переходит в Б (более сложный проект), работавший в Б — в В (более дорогой), работавший в В — в А (одновременно более сложный и более дорогой). Как видим, описанная в задаче ситуация могла случиться.

Ответ:

Такое могло быть.

Критери оценки:

0 баллов - нет ответа, неправильная логика рассуждений;
1-6 баллов - правильный ответ, нет объяснений и/или примера; неправильный ответ, есть верные рассуждения
7-13 баллов - правильный ответ, верный пример;
14-19 баллов - правильный ответ, есть верное объяснение, нет примеров;
20 баллов - правильный ответ, есть верное объяснение и примеры.

Задача 3.1.3 (30 баллов)

Условие:

Сравните дроби $\frac{11110}{11111}$, $\frac{33331}{33334}$, $\frac{44441}{44445}$ и расположите их в порядке возрастания.

Решение:

Рассмотрим числа
1-x=1/11111,
1-y=3/33334,
1-z=4/44445,
а также обратные к ним
1/(1-x)=11111,
1/(1-y)=11111+1/3,
1/(1-z)=11111+1/4.
Мы видим, что 1/(1-x) < 1/(1-z) < 1/(1-y).
Поскольку все рассматриваемые числа положительны, 1-x > 1-z > 1-y. Следовательно, x<z<y.

Ответ:

$\frac{11110}{11111}, \frac{44441}{44445}, \frac{33331}{33334}$

Критерии:

0 баллов - нет ответа/неправильный ответ, неправильная логика рассуждений;
1-10 балл - правильный ответ, нет объяснения или неправильная логика рассуждений;
11-20 балла - нет ответа/неправильный ответ, правильная логика рассуждений, решение не доведено до конца;

21-29 - правильный ответ, решение не доведено до конца или допущена арифметическая ошибка;
30 баллов - правильный ответ, верная логика рассуждений, решение доведено до конца.

Задача 3.1.4 (35 баллов)

Условие:

В городе 15 остановок, некоторые из них соединены маршрутами общественного транспорта: автобусными, трамвайными и троллейбусными.
Маршруты разработаны так, что в выходной для автобусного депо день, попасть с любой остановки на любую другую остановку (возможно, с пересадками) можно на трамвае и/или троллейбусе. Аналогично для выходных в трамвайном и троллейбусном депо.
Какое наименьшее число маршрутов может быть в городе?

Решение:

1) Найдем минимальное число маршрутов двух видов транспорта.
Обозначим число маршрутов через n. Если мы уберем один из них, транспортная сеть города может распасться на две разрозненные сети, но может и не распасться. Если убрать еще один маршрут, не более одной трансортной сети может разделиться ещё на две, количество сетей увеличится не более чем на одну. Продолжая аналогичные рассуждения, получим, что после отмены последнего маршрута количество трансортных сетей не может превысить $n+1$. Поскольку 15 остановок без маршрутов между ними - это 15 разрозненых сетей, то $n \geq 14$ *(1).*
Пусть количество маршрутов автобусов, трамваев и троллейбусов будет a, b и c. Исходя из найденного минимального числа маршрутов двух видов транспорта n, получаем $a+b \geq 14$, $b+c \geq 14$, $c+a \geq 14$. Складывая эти неравенства, получаем: $2(a+b+c) \geq 42$, т. е. у трёх видов транспорта в сумме как минимум 21 маршрут.

Ответ:

21 маршрут.

Критерии оценки:

0 баллов - нет ответа/неправильный ответ, неправильная логика рассуждений;
1-10 баллов - правильный ответ, нет объяснения или неправильная логика рассуждений;
11-20 баллов - нет ответа/неправильный ответ, правильная логика рассуждений, решение не доведено до конца;
21-30 баллов - верная логика рассуждений, найдено минимальное число маршрутов двух видов транспорта, решение не доведено до конца, то есть не получена система неравенств или получена неправильная;
31-34 балла - верная логика рассуждений, найдено минимальное число маршрутов двух видов транспорта, получена система неравенств, допущена арифметическая ошибка в решении системы неравенств;
35 баллов - правильный ответ, верная логика рассуждений, найдено минимальное число маршрутов двух видов транспорта, получена система неравенств.

3.2 Первая попытка
Задачи по математике (10-11 класс)

Задача 3.2.1 (20 баллов)

Условие:

Решить систему уравнений:

$$x + y + z = -1$$
$$x^2 + y^2 + z^2 = 1$$

$$x^3 + y^3 + z^3 = -1$$

Решение:

Возводим (1) в квадрат и вычитаем (2), получаем:

$(x + y + z)^2 - (x^2 + y^2 + z^2) = 2(xy + yz + xz) \Rightarrow xy + xz + yz = 0$(•)

Возводим (1) в куб и вычитаем (3), получаем:

$(x + y + z)^3 - (x^3 + y^3 + z^3) = 3(x + y)(x + z)(y + z) \Rightarrow 3xyz = 0$(••)

Анализируем (••):

Пусть $x = 0$, то выражение (•) показывает, что $yz = 0$. Поэтому либо $y = 0$ и $z = -1$, либо $z = 0$ и $y = -1$.

Аналогично для y=0 и z=0. В результате получаем: (0, 0, -1), (0, -1, 0) и (-1, 0, 0).

Ответ:

(-1,0,0), (0,-1,0), (0,0,-1)

Критерии:

0 баллов - нет ответа/неправильный ответ, неправильная логика рассуждений;

1-7 баллов - неправильный ответ, получено неверное эквивалентное равенство, верный ход рассуждений; правильный ответ, но нет решения

8-14 баллов - правильная логика рассуждений, решение не доведено до конца или допущена арифметическая ошибка

15 баллов - правильный ответ, верная логика рассуждений, решение доведено до конца.

Задача 3.2.2 (20 баллов)

Условие:

Разработчики переходят из проекта в проект, причем работа в каждом следующем проекте либо интереснее, либо лучше оплачивается.

Одному разработчику повезло - в его новом проекте одновременно более интересные задачи и более высокая зарплата.

Могло ли такое быть, если на одно место в проекте претендует один кандидат? Ответ объясните.

Решение:

Пусть есть три проекта — А, Б и В. Пусть по сложности задач проекты расположены в порядке В, А, Б (Б — самый сложный), а по зарплате — в порядке Б, В, А (А — самый дорогой). Предположим, что разработчик, работавший в А, переходит в Б (более сложный проект), работавший в Б — в В (более дорогой), работавший в В — в А (одновременно более сложный и более дорогой). Как видим, описанная в задаче ситуация могла случиться.

Ответ:

Такое могло быть.

Критерии оценки:

0 баллов - нет ответа, неправильная логика рассуждений;

1-6 баллов - правильный ответ, нет объяснений и/или примера;

7-13 баллов - правильный ответ, верный пример;

14-19 баллов - правильный ответ, есть верное объяснение, нет примеров;

20 баллов - правильный ответ, есть верное объяснение и примеры.

Задача 3.2.3 (30 баллов)

Условие:

Три стартапера выбирают место для работы в своем городе. Один из них планирует добираться до работы пешком, второй - ехать на велосипеде, третий - на автомобиле. Их скорости соотносятся как 1:2:3. Какое место выбрать стартаперам, чтобы суммарное время, затрачиваемое на дорогу всеми участниками, было наименьшим?

Все трое живут не на одной улице. Каждый выбирает кратчайший путь до места встречи.

Решение:

Этим местом для работы является дом первого стартапера, планирующего ходить на работу пешком. Для доказательства этого обозначим искомое место для работы буквой *A*, а дома стартаперов - цифрами 1, 2, 3 в соответствии с соотношением скоростей. Расстояния между домом 1 и домами 2 и 3 обозначим через a и *b*, а расстояния от точки *A* до домов 1, 2, 3 — через *x*, *y* и *z* соответственно. Тогда

$\frac{a}{2} \leq \frac{x}{2} + \frac{y}{2}, \frac{b}{3} \leq \frac{x}{3} + \frac{z}{3},$

откуда

$\frac{a}{2} + \frac{b}{3} \leq \frac{x}{2} + \frac{y}{2} + \frac{x}{3} + \frac{z}{3} \leq \frac{y}{2} + \frac{z}{3}$, причём равенство достигается при x = 0, т. е. когда точка *A* совпадает с домом 1.

Ответ:

Дома у того, кто планирует добираться до работы пешком.

Критерии оценки:

0 баллов - нет ответа/неправильный ответ, неправильная логика рассуждений;
1-10 балл - правильный ответ, нет объяснения или неправильная логика рассуждений;
11-20 балла - нет ответа/неправильный ответ, правильная логика рассуждений, решение не доведено до конца;
21-29 - правильный ответ, решение не доведено до конца или допущена арифметическая ошибка;
30 баллов - правильный ответ, верная логика рассуждений, решение доведено до конца.

Задача 3.2.4 (35 баллов)

Условие:

Компьютеры компании объединены в несколько сетей, при этом: 1) любой компьютер может напрямую обратиться к любому другому компьютеру; 2) для любой пары сетей найдется, и притом единственное, общее устройство; 3) в каждой сети ровно три компьютера. Сколько всего сетей?

Решение:

Докажем, что если выполняются указанные условия, то число устройств *n* и число сетей *N* связаны соотношением $N = n(n - 1) + 1$. Пусть *a* — одна из сетей, *B* — компьютер, не подключенный к сети *a*. Каждая сеть, к которой подключен *B*, имеет общее устройство с сетью *a*. Поэтому *B* подключен ровно к *n* шинам. Аналогично доказывается, что через каждый компьютер сети *a* подключен к *n* - 1 сети, отличной от *a*. Всего получаем $n(n - 1)$ разных сетей и ещё сама *a*. Итого 7 сетей.

Ответ: 7

Критерии оценки:

0 баллов - нет ответа/неправильный ответ, неправильная логика рассуждений;
1-10 баллов - правильный ответ, нет объяснения или неправильная логика рассуждений;

11-20 баллов - нет ответа/неправильный ответ, правильная логика рассуждений, решение не доведено до конца;
21-30 баллов - верная логика рассуждений, решение не доведено до конца, то есть не получено соотношение между числом устройств и числом сетей или получено неправильное;
31-34 балла - верная логика рассуждений, получено соотношение между числом устройств и числом сетей, допущена арифметическая ошибка;
35 баллов - правильный ответ, верная логика рассуждений, получено соотношение между числом устройства, получена система неравенств.

3.3 Первая попытка
Задачи по информатике (9-11 класс)

Задача 3.3.1 (7 баллов)

Условия:

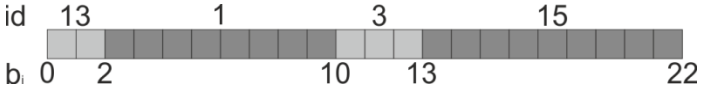
Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

При управлении сложными системами центральному серверу постоянно приходится отслеживать состояние множества компонент – будь то датчики, исполнительные устройства или другое оборудование. Рассмотрим систему, состоящую из N различных устройств. Каждое из этих устройств имеет свой уникальный целочисленный идентификатор id. Для хранения информации об -ом устройстве в памяти центрального сервера выделяется w_i байт. Память под все эти устройства выделяется одним целым блоком, размера

$$W = \sum_i w_i.$$

В любой момент времени серверу известен адрес в памяти на начало этого блока. Отсупом -го устройства будем называть количество байт b_i, которое нужно прибавить к адресу начала блока, чтобы получить адрес i-го устройства. Отступ первого устройства b_1 всегда равен 0. Отступ каждого последующего устройства рассчитывается по формуле:
$b_i = b_{i-1} + w_{i-1}$.

Каждый раз, когда серверу необходимо обратиться к устройству с определенным id, ему необходимо определить место в памяти, где хранится информация об этом устройстве. Ваша задача написать программу, которая будет рассчитывать отступ для устройств по запрашиваемым id.



Формат входных данных

В первой строке даны два целых числа: N – количество различных устройств, и K – количество запросов ($2 \le N \le 10^3, 1 \le K \le 10^3$).
В следующих N строках заданы по два целых числа: id_i и w_i – идентификатор и объём выделяемой памяти для -го устройства соответственно ($1 \le id_i \le 10^5, 1 \le w_i \le 10^5$). Устройства даны в том порядке, в котором они хранятся в выделенном блоке памяти.
Следующие K строк содержат по единственному целому числу id_k – идентификатор

устройства, для которого необходимо посчитать отступ b_k.

Формат выходных данных
Для каждого из K запросов выведите число b_k в отдельной строке.

Пример

Входные данные	**Выходные данные**
4 6 13 2 1 8 3 3 15 9 1 13 3 13 1 15	2 0 10 0 2 13

Решение:
Для заданных ограничений задачу можно решить, выделив массив A размером $10^5 + 1$ элементов, где индексом будет являться номер id_i устройства, а значением – накопленная сумма $W_i = \sum_{j=1}^{i-1} w_j$. Тогда для каждого id_k ответом будет являться значение $A[id_k]$.
Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

typedef struct {
    int id;
    long long w_cum;
} Device;
bool cmp1(Device &a, Device &b)
{
    return a.id > b.id;
}
int main()
{
    int n, k;
```

```
    scanf("%d%d\n", &n, &k);
    vector <Device> devs(n);

    int w_cum = 0;
    int max_id = -1;
    for (int i = 0; i < n; i++)
    {
        devs[i].w_cum = w_cum;
        if (i)
        {
            devs[i].w_cum += devs[i - 1].w_cum;
        }
        scanf("%d%d\n", &devs[i].id, &w_cum);
        max_id = max(devs[i].id, max_id);
    }
    vector <int> id_map(max_id + 1);
    for (int i = 0; i < n; i++)
    {
        id_map[devs[i].id] = i;
    }

    for (int i = 0; i < k; i++)
    {
        int tmp;
        scanf("%d\n", &tmp);
        printf("%d\n", devs[id_map[tmp]].w_cum);
    }
    return 0;
}
```

Код проверки:

```
import operator

def generate():

      tests = []

      return tests

def solve(dataset):

      input = dataset.split();

      cnt = 0;

      n = int(input[cnt]);

      cnt += 1;

      k = int(input[cnt]);

      cnt += 1;
```

```
	devs = [];
	w_cum = 0;
	for i in range(n):
		dev = {'id':0, 'w_cum':w_cum};
		if i > 0:
			dev['w_cum'] += devs[i - 1]['w_cum'];
		dev['id'] = int(input[cnt]);
		cnt += 1;
		w_cum = int(input[cnt]);
		cnt += 1;
		devs.append(dev);

	devs.sort(key=operator.itemgetter('id'));
	res = "";
	for i in range(k):
		id = int(input[cnt]);
		cnt += 1;
		idx = bin_search(devs, id);
		res += str(devs[idx]['w_cum']) + '\n';

	return res
def bin_search(ar, x):
	l = 0;
	r = len(ar) - 1;
	while (l <= r):
		m = int((l + r) / 2);
		if (ar[m]['id'] == x):
			return m;
		elif (ar[m]['id'] < x):
			l = m + 1;
		else:
			r = m - 1;
```

```
        return -1;
def check(reply, clue):
        reply = [str(i) for i in reply.strip().split()]
        clue = [str(i) for i in clue.strip().split()]
        return reply == clue
```

Тесты решений доступны по ссылке:

https://drive.google.com/drive/folders/0B6WHh6ex6PMtZy1RWmVJQmdZWWc

Задача 3.3.2 (13 баллов)

Условия:

Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

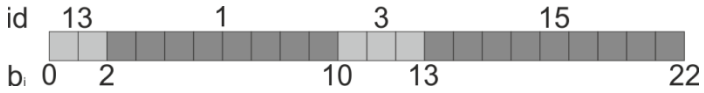
При управлении сложными системами центральному серверу постоянно приходится отслеживать состояние множества компонент – будь то датчики, исполнительные устройства или другое оборудование. Рассмотрим систему, состоящую из N различных устройств. Каждое из этих устройств имеет свой уникальный целочисленный идентификатор id. Для хранения информации об -ом устройстве в памяти центрального сервера выделяется w_i байт. Память под все эти устройства выделяется одним целым блоком, размера

$$W = \sum_i w_i.$$

В любой момент времени серверу известен адрес в памяти на начало этого блока. Отсупом -го устройства будем называть количество байт b_i, которое нужно прибавить к адресу начала блока, чтобы получить адрес i-го устройства. Отступ первого устройства b_1 всегда равен 0. Отступ каждого последующего устройства рассчитывается по формуле:

$$b_i = b_{i-1} + w_{i-1}.$$

Каждый раз, когда серверу необходимо обратиться к устройству с определенным id, ему необходимо определить место в памяти, где хранится информация об этом устройстве. Ваша задача написать программу, которая будет рассчитывать отступ для устройств по запрашиваемым id.



Формат входных данных

В первой строке даны два целых числа: N – количество различных устройств, и K – количество запросов ($2 \le N \le 10^5, 1 \le K \le 3 \cdot 10^5$).

В следующих N строках заданы по два целых числа: id_i и w_i – идентификатор и объём

выделяемой памяти для -го устройства соответственно ($1 \le id_i \le 10^9, 1 \le w_i \le 10^5$). Устройства даны в том порядке, в котором они хранятся в выделенном блоке памяти.
Следующие K строк содержат по единственному целому числу id_k – идентификатор устройства, для которого необходимо посчитать отступ b_k.

Формат выходных данных
Для каждого из K запросов выведите число b_k в отдельной строке.

Пример

Входные данные	**Выходные данные**
4 6 13 2 1 8 3 3 15 9 1 13 3 13 1 15	2 0 10 0 2 13

Решение:
Для заданных ограничений задачу можно решить, используя любой алгоритм поиска со сложностью$O(logN)$, где N – количество различных устройств. Например, метод дихотомии. Данный метод находит элементы в отсортированном массиве, поэтому необходимо предварительно отсортировать элементы по возрастанию величины id любым алгоритмом сортировки со сложностью$O(N \cdot logN)$. Таким образом, общая сложность решения $O((K+N) \cdot logN)$, что укладывается в лимит по времени для заданных ограничений.
Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

typedef struct {
    int id;
    long long w_cum;
```

```
} Device;

int cmp1(Device &a, Device &b)
{
    return a.id < b.id;
}

int bin_search(vector <Device> &ar, int x)
{
    int l = 0;
    int r = ar.size() - 1;
    while (l <= r)
    {
        int m = (l + r) >> 1;
        if (ar[m].id == x)
        {
            return m;
        }
        else if (ar[m].id < x)
        {
            l = m + 1;
        }
        else // if(ar[m].id > x)
        {
            r = m - 1;
        }
    }

    return -1;
}

int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, k;
    fscanf(f_in, "%d%d\n", &n, &k);

    vector <Device> devs(n);
    int w_cum = 0;
    for (int i = 0; i < n; i++)
    {
        devs[i].w_cum = w_cum;
        if (i)
        {
            devs[i].w_cum += devs[i - 1].w_cum;
        }

        fscanf(f_in, "%d%d\n", &devs[i].id, &w_cum);
```

```
    }

    sort(devs.begin(), devs.end(), cmp1);
    for (int i = 0; i < k; i++)
    {
        int id;
        fscanf(f_in, "%d\n", &id);
        int idx = bin_search(devs, id);
        fprintf(f_out, "%lld\n", devs[idx].w_cum);
    }

    return 0;
     }
```

Код проверки:

```
import operator

def generate():

     tests = []

     return tests

def solve(dataset):

     input = dataset.split();

     cnt = 0;

     n = int(input[cnt]);

     cnt += 1;

     k = int(input[cnt]);

     cnt += 1;


     devs = [];

     w_cum = 0;

     for i in range(n):

          dev = {'id':0, 'w_cum':w_cum};

          if i > 0:

               dev['w_cum'] += devs[i - 1]['w_cum'];

          dev['id'] = int(input[cnt]);

          cnt += 1;

          w_cum = int(input[cnt]);

          cnt += 1;
```

```
            devs.append(dev);

      devs.sort(key=operator.itemgetter('id'));

      res = "";
      for i in range(k):
            id = int(input[cnt]);
            cnt += 1;
            idx = bin_search(devs, id);
            res += str(devs[idx]['w_cum']) + '\n';

      return res

def bin_search(ar, x):
      l = 0;
      r = len(ar) - 1;
      while (l <= r):
            m = int((l + r) / 2);
            if (ar[m]['id'] == x):
                  return m;
            elif (ar[m]['id'] < x):
                  l = m + 1;
            else:
                  r = m - 1;

      return -1;

def check(reply, clue):
      reply = [str(i) for i in reply.strip().split()]
      clue = [str(i) for i in clue.strip().split()]
      return reply == clue
```

Тесты решений доступны по ссылке:
https://drive.google.com/drive/folders/0B6WHh6ex6PMtZy1RWmVJQmdZWWc

Задача 3.3.3 (7 баллов)

Условия:

Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

В системе управления иногда возникают простои, когда никаких важных задач выполнять не надо. Это свободное время используется для выполнения некоторых второстепенных задач. Время, которое система гарантировано может потратить на второстепенные задачи, всегда известно и равно T. Всего возможных второстепенных задач N. Для каждой из них известно время выполнения t и некоторая целочисленная величина b, характеризующая пользу, которую несет с собой выполнение этой задачи.
Вам необходимо по известному списку задач определить, какую максимальную суммарную пользу может принести система, оптимальным образом выбрав задачи, которые можно выполнить за суммарное время, не превосходящее T.
Польза от выполнения задачи засчитывается только для полностью завершенных задач. Задачи выполняются последовательно, после завершения одной – сразу начинается выполнение второй. Каждая задача может быть выполнена только один раз.

Формат входных данных

В первой строке даны два целых числа: N и T – количество доступных задач и время на выполнение второстепенных задач ($1 \le N \le 20, 1 \le T \le 10^3$).
В следующих N строках заданы по два целых числа: t_i и b_i – время выполнения и характеристика пользы -ой задачи ($1 \le t_i \le 10^3, 1 \le b_i \le 10^5$).

Формат выходных данных

В ответ выведите единственное число B – максимальную возможную суммарную пользу, которую можно получить, выполнив оптимальный набор задач из списка за суммарное время не превосходящее T.

Пример

Входные данные	**Выходные данные**
5 7 2 8 3 10 3 8 5 5 3 7	18

Решение:

Учитывая, что количество доступных задач N не превосходит 20, различных комбинаций задач, которые можно выполнить без учёта ограничения времени T, равно $2^{20} = 1048576 \sim 10^6$. Такое количество комбинаций можно полностью перебрать за отведенное время (1 секунду). В процессе перебора среди всех наборов задач, суммарное время на выполнение которых не превышает T, необходимо выбрать тот, который даёт наибольшую среди всех величину суммарной пользы B. Это и будет ответом.
Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

int main()
{
    int n, t;
    scanf("%d%d\n", &n, &t);
    vector<pair<int, int> > ar(n);
    for (int i = 0; i < n; i++)
    {
        scanf("%d%d\n", &ar[i].second, &ar[i].first);
    }

    int total_b = 0;

    for (int i = 1; i < (1 << n); i++)
    {
        int w = 0, b = 0;
        for (int j = 0; (1 << j) < i; j++)
        {
            if (i & (1 << j))
            {
                w += ar[j].second;
                b += ar[j].first;
            }
        }
        if (w <= t)
            total_b = max(total_b, b);
    }

    printf("%d\n", total_b);

    return 0;
     }
```

Код проверки:

```
def generate():
     tests = []
     return tests
```

```
def solve(dataset):
      input = dataset.split();

      cnt = 0;
      n = int(input[cnt]);
      cnt += 1;
      T = int(input[cnt]);
      cnt += 1;

      t = [];
      b = [];
      for i in range(n):
            t.append(int(input[cnt]));
            cnt += 1;
            b.append(int(input[cnt]));
            cnt += 1;
      dp = [[0 for i in range(T + 1)] for i in range(n + 1)];
      for i in range(1, n + 1):
            for j in range(T + 1):
                  if (t[i - 1] > j):
                        dp[i][j] = dp[i - 1][j];
                  else:
                        dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - t[i -
1]] + b[i - 1]);

      res = str(dp[n][T]) + '\n';

      return res


def check(reply, clue):
      reply = [str(i) for i in reply.strip().split()]
      clue = [str(i) for i in clue.strip().split()]
      return reply == clue
```

Тесты решений доступны по ссылке:
https://drive.google.com/drive/folders/0B6WHh6ex6PMtZy1RWmVJQmdZWWc

Задача 3.3.4 (13 баллов)

Условия:

Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

В системе управления иногда возникают простои, когда никаких важных задач выполнять не надо. Это свободное время используется для выполнения некоторых второстепенных задач. Время, которое система гарантировано может потратить на второстепенные задачи, всегда известно и равно T. Всего возможных второстепенных задач N. Для каждой из них известно время выполнения t и некоторая целочисленная величина b, характеризующая пользу, которую несет с собой выполнение этой задачи.
Вам необходимо по известному списку задач определить, какую максимальную суммарную пользу может принести система, оптимальным образом выбрав задачи, которые можно выполнить за суммарное время, не превосходящее T.
Польза от выполнения задачи засчитывается только для полностью завершенных задач. Задачи выполняются последовательно, после завершения одной – сразу начинается

выполнение второй. Каждая задача может быть выполнена только один раз.

Формат входных данных

В первой строке даны два целых числа: N и T – количество доступных задач и время на выполнение второстепенных задач ($1 \le N \le 10^3, 1 \le T \le 10^3$).

В следующих N строках заданы по два целых числа: t_i и b_i – время выполнения и характеристика пользы -ой задачи ($1 \le t_i \le 10^3, 1 \le b_i \le 10^5$).

Формат выходных данных

В ответ выведите единственное число B – максимальную возможную суммарную пользу, которую можно получить, выполнив оптимальный набор задач из списка за суммарное время не превосходящее T.

Пример

Входные данные	**Выходные данные**
5 7 2 8 3 10 3 8 5 5 3 7	18

Решение:

Данная задача является классической задачей динамического программирования – задача о рюкзаке. Пусть $B[i][t]$ – максимальная польза, которую можно получить, выполнив какое-либо подмножество из i-первых доступных задач за время, не превышающее t. Запишем рекуррентные соотношения, справедливые для $B[i][t]$:

- $B[0][t] = 0$
- $B[i][t] = B[i-1][t]$, если $t_i > t$
- $B[i][t] = max(B[i-1][t], B[i-1][t-t_i] + b_i)$, если $t_i < t$.

Ответом на задачу, таким образом, будет являться значение $B[N][T]$. Сложность расчёта этой величины есть $O(N \cdot T)$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

int main()
{
```

```
    int T, n;
    scanf("%d%d", &n, &T);
    vector <int> t(n);
    vector <int> b(n);
    vector <vector <int> > dp(n + 1, vector<int>(T + 1, 0));

    for (int i = 0; i < n; i++)
    {
        scanf("%d%d", &t[i], &b[i]);
    }

    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; j <= T; j++)
        {
            if (t[i - 1] > j)
            {
                dp[i][j] = dp[i - 1][j];
            }
            else
            {
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - t[i - 1]] +
b[i - 1]);
            }
        }
    }

    printf("%d\n", dp[n][T]);

    return 0;
}
```

Код проверки:

```
def generate():
     tests = []
     return tests

def solve(dataset):
     input = dataset.split();

     cnt = 0;
     n = int(input[cnt]);
     cnt += 1;
     T = int(input[cnt]);
     cnt += 1;

     t = [];
     b = [];
     for i in range(n):
          t.append(int(input[cnt]));
          cnt += 1;
          b.append(int(input[cnt]));
```

```
            cnt += 1;
      dp = [[0 for i in range(T + 1)] for i in range(n + 1)];
      for i in range(1, n + 1):
            for j in range(T + 1):
                  if (t[i - 1] > j):
                        dp[i][j] = dp[i - 1][j];
                  else:
                        dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - t[i -
1]] + b[i - 1]);

      res = str(dp[n][T]) + '\n';

      return res


def check(reply, clue):
      reply = [str(i) for i in reply.strip().split()]
      clue = [str(i) for i in clue.strip().split()]
      return reply == clue
```

Тесты решений доступны по ссылке:
https://drive.google.com/drive/folders/0B6WHh6ex6PMtZy1RWmVJQmdZWWc

Задача 3.3.5 (7 баллов)

Условия:

Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

В данной задаче рассматривается история выполнения различных операций некой системы управления за некоторый промежуток времени. Известно, что за рассматриваемое время система успела N раз выполнить определенные операции. Большинство из них были выполнены четное число раз, но K операций были выполнены нечетное количество раз.
Каждая операция имеет свой целочисленный идентификационный номер id.
Ваша задача по истории выполненных операций определить K идентификационных номеров id_k тех операций, которые выполнились нечетное количество раз.

Формат входных данных

В первой строке даны два целых числа N и K – общее количество выполненных за день операций и количество операций, выполненных нечетное число раз соответственно ($2 \leq N \leq 10^7, K = 1$).
В следующих N строках содержится по одному целому числу id_i – идентификатор выполненной операции ($1 \leq id_i \leq 10^9$).

Формат выходных данных

В единственной строке через пробел выведите K чисел id_k в порядке возрастания – идентификаторы операций, выполненных нечетное число раз.

Пример

Входные данные	**Выходные данные**
9 1	7

7 3 5 5 3 7 2 2 7	

Решение:

Обратим внимание на ограничение по памяти в 32 МБ при количестве входных данных порядка 10^7. Учитывая, что идентификаторы id могут быть в лучшем случае представлены в виде 32-битного целого числа (4 байта), для хранения всех идентификаторов потребуется порядка 40 МБ памяти.

Для решения данной задачи воспользуемся следующими свойствами операции суммирования по модулю 2 ($\oplus$):

- $a \oplus a = 0$
- $0 \oplus a = a$
- $a \oplus b = b \oplus a$
- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

Эти же свойства справедливы для эквивалентной битовой операции «исключающего ИЛИ». Из этих свойств, в частности, следует, что побитовая сумма по модулю 2 для всех входных идентификаторов id_i будет равно сумме по модулю два только тех идентификаторов, которые встретились нечётное число раз. Учитывая, что в условиях данной задачи такой идентификатор единственен, эта сумма по модулю 2 всех идентификаторов и будет являться ответом на задачу. При этом для её вычисления нет необходимости хранить все входящие числа; размер необходимой памяти есть $O(1)$, сложность решения есть $O(N)$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, k;
    fscanf(f_in, "%d%d", &n, &k);

    int x = 0;
    for (int i = 0; i < n; i++)
    {
```

```
        int tmp;
        fscanf(f_in, "%d", &tmp);
        x ^= tmp;
    }

    fprintf(f_out, "%d\n", x);

    return 0;
}
```

Код проверки:

```
def generate():
     tests = []
     return tests

def get_next(dataset, ptr, slen):
     while (ptr < slen and not((dataset[ptr] >= '0' and dataset[ptr]
<= '9') or dataset[ptr] == '-')):
          ptr += 1;
     if ptr == slen:
          return 0;
     mult = 1;
     if (dataset[ptr] == '-'):
          mult = -1;
          ptr += 1;

     n = 0;
     while (dataset[ptr] >= '0' and dataset[ptr] <= '9'):
          n = n * 10 + int(dataset[ptr]);
          ptr += 1;

     return n, ptr + 1

def solve(dataset):
     ptr = 0;
     slen = len(dataset);

     n, ptr = get_next(dataset, ptr, slen)
     k, ptr = get_next(dataset, ptr, slen)

     res = "";
     if (k == 1):
          res = sol_3_1(n, dataset, ptr, slen);
     elif (k == 2):
          res = sol_3_2(n, dataset, ptr, slen);

     res += '\n';

     return res

def sol_3_1(n, dataset, ptr, slen):
     x = 0;
     for i in range(n):
          tmp, ptr = get_next(dataset, ptr, slen);
          x ^= tmp;
```

```
      return str(x);


def sol_3_2(n, dataset, ptr, slen):
      x = 0;
      ar = [0] * 31;
      for i in range(n):
            tmp, ptr = get_next(dataset, ptr, slen);
            x ^= tmp;
            for shift in range(31):
                  if (tmp & (1 << shift)):
                        ar[shift] ^= tmp;

            a = -1;
            b = -1;
            for i in range(31):
                  if (ar[i] != 0):
                        if (a == -1):
                              a = ar[i];
                        elif (a != ar[i]):
                              b = ar[i];
                              break;

            if (a == x):
                  a ^= b;
            elif (b == x):
                  b ^= a;
            if (b < a):
                  a, b = b, a;

      return str(a) + ' ' + str(b);

def check(reply, clue):
      reply = [str(i) for i in reply.strip().split()]
      clue = [str(i) for i in clue.strip().split()]
      return reply == clue
```

Тесты решений доступны по ссылке:
https://drive.google.com/drive/folders/0B6WHh6ex6PMtZy1RWmVJQmdZWWc

Задача 3.3.6 (20 баллов)

Условия:

Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

В данной задаче рассматривается история выполнения различных операций некой системы управления за некоторый промежуток времени. Известно, что за рассматриваемое время система успела N раз выполнить определенные операции. Большинство из них были выполнены четное число раз, но K операций были выполнены нечетное количество раз.

Каждая операция имеет свой целочисленный идентификационный номер id.

Ваша задача по истории выполненных операций определить K идентификационных номеров id_k тех операций, которые выполнились нечетное количество раз.

Формат входных данных
В первой строке даны два целых числа N и K – общее количество выполненных за день операций и количество операций, выполненных нечетное число раз соответственно ($2 \le N \le 10^7, K = 2$).
В следующих N строках содержится по одному целому числу id_i – идентификатор выполненной операции ($1 \le id_i \le 10^9$).

Формат выходных данных: В единственной строке через пробел выведите K чисел id_k в порядке возрастания – идентификаторы операций, выполненных нечетное число раз.

Пример:

Входные данные	**Выходные данные**
8 2 3 2 8 2 2 8 8 3	2 8

Решение:
Обратим внимание на ограничение по памяти в 32 МБ при количестве входных данных порядка 10^7. Учитывая, что идентификаторы id могут быть в лучшем случае представлены в виде 32-битного целого числа (4 байта), для хранения всех идентификаторов потребуется порядка 40 МБ памяти.
Для решения данной задачи воспользуемся следующими свойствами операции суммирования по модулю 2 ($\oplus$):

- $a \oplus a = 0$
- $0 \oplus a = a$
- $a \oplus b = b \oplus a$
- $(a \oplus b) \oplus c = a \oplus (b \oplus c)$

Эти же свойства справедливы для эквивалентной битовой операции «исключающего ИЛИ». Из этих свойств, в частности, следует, что побитовая сумма по модулю 2 для всех входных идентификаторов id_i будет равно сумме по модулю два только тех идентификаторов, которые встретились нечётное число раз. В условиях данной задачи таких идентификаторов ровно два.

Обратим внимание, что если два числа не равны друг другу, то в битовом представлении они так же будут отличаться как минимум в одном бите. Кроме общей суммы по модулю 2 будем так же собирать 30 (количество бит, необходимых для представления числа 10^9) дополнительных сумм в массив S, где каждый элемент $S[j]$ будет хранить сумму по модулю 2 только тех элементов, которые имеют ненулевой j-ый бит в двоичном представлении. Так как два искомых идентификатора будут отличаться хотя бы в одном из разрядов битового представления, хотя бы один элемент массива S после выполнения всех суммирований будет отличаться от общей суммы по модулю 2 всех идентификаторов. Этот элемент и будет являться одним из искомых чисел. Второй число получается как результат суммирования по модулю два числа, найденного в массиве S и общей суммы по модулю 2 всех идентификаторов.

Для решения задачи нет необходимости хранить все входящие числа; размер необходимой памяти есть $O(\lceil log_2 max(id) \rceil)$, сложность решения есть $O(N \cdot \lceil log_2 max(id) \rceil)$.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>

using namespace std;

int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, k;
    fscanf(f_in, "%d%d", &n, &k);

    int ar[31] = {};
    int x = 0;
    for (int i = 0; i < n; i++)
    {
        int tmp;
        fscanf(f_in, "%d", &tmp);
        x ^= tmp;
        for (int shift = 0; shift < 31; shift++)
        {
            if (tmp & (1 << shift))
            {
                ar[shift] ^= tmp;
            }
        }
    }

    int a = -1, b = -1;
    for (int i = 0; i < 31; i++)
    {
        if (ar[i])
        {
            if (a == -1)
            {
                a = ar[i];
            }
            else if (a != ar[i])
            {
                b = ar[i];
```

```
                break;
            }
        }
    }

    if (a == x)
    {
        a ^= b;
    }
    else if (b == x)
    {
        b ^= a;
    }
    if (b < a)
    {
        swap(a, b);
    }

    fprintf(f_out, "%d %d\n", a, b);

    return 0;
}
```

Код проверки:

```
def generate():
     tests = []
     return tests

def get_next(dataset, ptr, slen):
     while (ptr < slen and not((dataset[ptr] >= '0' and dataset[ptr]
<= '9') or dataset[ptr] == '-')):
          ptr += 1;
     if ptr == slen:
          return 0;
     mult = 1;
     if (dataset[ptr] == '-'):
          mult = -1;
          ptr += 1;

     n = 0;
     while (dataset[ptr] >= '0' and dataset[ptr] <= '9'):
          n = n * 10 + int(dataset[ptr]);
          ptr += 1;

     return n, ptr + 1

def solve(dataset):
     ptr = 0;
     slen = len(dataset);

     n, ptr = get_next(dataset, ptr, slen)
     k, ptr = get_next(dataset, ptr, slen)
```

```
      res = "";
      if (k == 1):
            res = sol_3_1(n, dataset, ptr, slen);
      elif (k == 2):
            res = sol_3_2(n, dataset, ptr, slen);

      res += '\n';

      return res

def sol_3_1(n, dataset, ptr, slen):
      x = 0;
      for i in range(n):
            tmp, ptr = get_next(dataset, ptr, slen);
            x ^= tmp;

      return str(x);


def sol_3_2(n, dataset, ptr, slen):
      x = 0;
      ar = [0] * 31;
      for i in range(n):
            tmp, ptr = get_next(dataset, ptr, slen);
            x ^= tmp;
            for shift in range(31):
                  if (tmp & (1 << shift)):
                        ar[shift] ^= tmp;

            a = -1;
            b = -1;
            for i in range(31):
                  if (ar[i] != 0):
                        if (a == -1):
                              a = ar[i];
                        elif (a != ar[i]):
                              b = ar[i];
                              break;

            if (a == x):
                  a ^= b;
            elif (b == x):
                  b ^= a;
            if (b < a):
                  a, b = b, a;

      return str(a) + ' ' + str(b);

def check(reply, clue):
      reply = [str(i) for i in reply.strip().split()]
      clue = [str(i) for i in clue.strip().split()]
      return reply == clue
```

Тесты решений доступны по ссылке:
https://drive.google.com/drive/folders/0B6WHh6ex6PMtZy1RWmVJQmdZWWc

Задача 3.3.7 (33 балла)

Условия:

Ограничение по времени: 1 секунда
Ограничение по памяти: 32 мегабайта

Умный беспилотный робот пылесос, убирающий грязь с пола автономного космического транспорта, всегда передвигается по заданной последовательности из N векторов. Сначала пылесос находиться в точке $(0; 0)$, соответствующей положению его док-станции. Первым шагом он перемещается по вектору v_1. Его координаты становятся равны $(v_{1,x}, v_{1,y})$. Отсюда он следует по вектору v_2 в точку $(v_{1,x} + v_{2,x}, v_{1,y} + v_{2,y})$. Далее по вектору v_3 и т.д. до вектора v_N. После этого робот возвращается на док-станцию. «Почему же этот пылесос называется умным, если ездит он исключительно по заданной последовательности векторов?» – удивитесь вы. Оказывается, когда робот собирается начать движение по очередному вектору, он может выбрать двигаться ему по вектору v_i или по вектору $-v_i$ (в противоположном направлении). Известно, что длина всех векторов v_i не превосходит D.

Переместившись по всем доступным векторам умный пылесос должен иметь связь (конечно же, беспроводную) со своей док-станцией, чтобы по окончании обхода знать, куда вернуться на зарядку. Для обеспечения связи между обоими устройствами, пылесос должен находится на расстоянии не большем чем $\sqrt{2} \cdot D$ от док-станции.

Ваша задача, написать программу, которая должна определить направление движения пылесоса вдоль каждого вектора таким образом, чтобы в конце пути он не отдалился от док-станции с координатами $(0; 0)$, на расстояние большее чем $\sqrt{2} \cdot D$. Обратите внимание, что это условие должно выполниться только после прохода по всем N векторам. Промежуточные координаты пылесоса могут превосходить эту величину.

Формат входных данных

В первой строке даны два целых числа: количество векторов N, и расстояние D ($2 \leq N \leq 10^4, 1 \leq D \leq 100$).

В следующих N строках содержится по два целых числа x и y – координаты очередного вектора ($0 \leq \sqrt{x^2 + y^2} \leq D$).

Формат выходных данных

Если выбрать правильную последовательность направлений векторов таким образом, чтобы пылесос всегда был на связи с док-станцией невозможно, выведите единственное слово “Impossible” (без кавычек).

Если найти правильную последовательность направлений возможно, в первой строке выведите слово “Possible” (без кавычек), а во второй строке последовательность символов “F” и “B” (без кавычек) по следующему правилу. Если для очередного вектора выбрано направление совпадающее с направлением вектора, выводится символ “F” (от англ. – Forward). Если же выбрано направление противоположное направлению вектора, выводится символ “B” (от англ. – Backward).

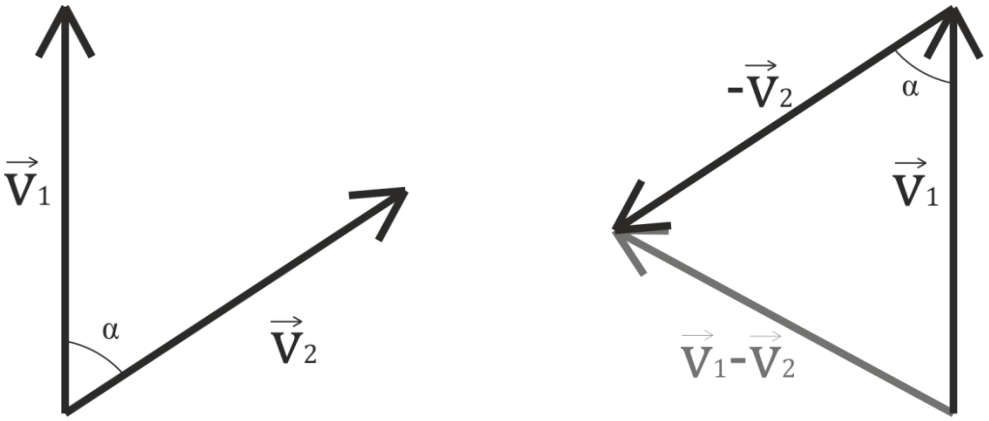
Пример

Входные данные	**Выходные данные**
5 7 0 7 -7 0 1 -1	Possible FFFFB

-1 1 -6 3	

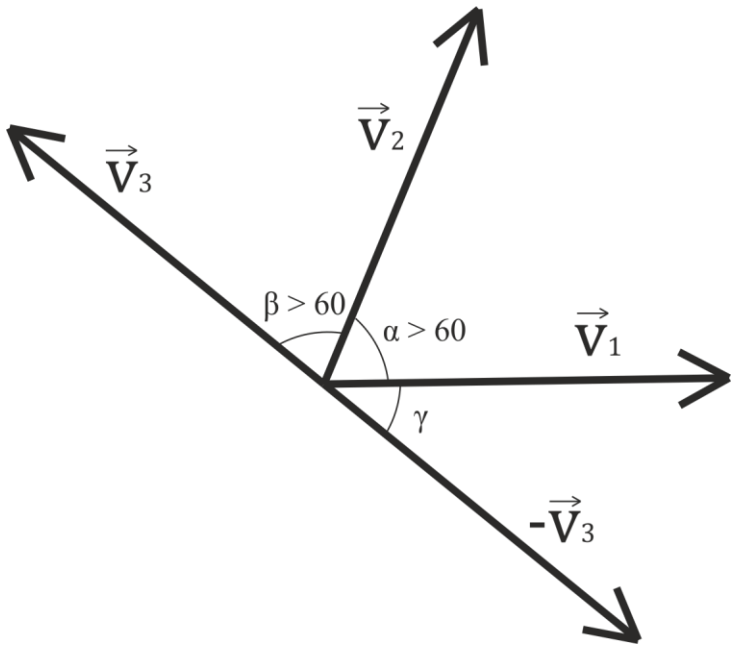
Решение:

Рассмотрим пары векторов, угол между которыми не превосходит 90°. Если угол между векторами больше 90°, приведём их к необходимому нам условию, взяв один из векторов со знаком минус. Пусть длина всех рассматриваемых векторов равна D. Если угол между парой векторов длины D равен 60°, то длина вектора, являющегося разностью этих векторов, будет равен D. Если угол меньше 60°, то длина вектора их разности будет меньше D.



С увеличением угла между парой векторов (вплоть до 90°), разность этих векторов так же будет увеличиваться. Когда угол между векторами равен 90°, с какими бы знаками не были взяты векторы, длина вектора их суммы или разности будет равна $\sqrt{2} \cdot D$ (как гипотенуза равнобедренного прямоугольного треугольника). Таким образом, из двух векторов длины D гарантировано можно получить суммарный вектор длины не больше $\sqrt{2} \cdot D$, правильно расставив знаки плюс или минус (**утверждение 1**).

Пусть угол между парой векторов длины D больше 60°, но не превосходит 90°. Невозможно найти третий вектор длины D такой, что наименьший угол между ним и каждым из двух ранее рассмотренных векторов был больше 60° (при условии, что третий вектор может быть взять как со знаком +, так и со знаком -).



На рисунке проиллюстрирован возможный выбор третьего вектора v_3 таким образом, чтобы углы между каждой парой векторов были больше 60°. Учитывая, что $\alpha > 60°$ и $\beta > 60°$, взяв вектор $-v_3$(со знаком минус), можно получить угол γ между векторами v_1и $-v_3$ равный

$\gamma = 180° - (\alpha + \beta) < 60°$.

Таким образом, среди трёх векторов длины D всегда можно выбрать 2 вектора таким образом, чтобы получить суммарный вектор (с правильно расставленными знаками плюс и минус), длина которого не превысит D (**утверждение 2**).

Рассмотренный случай, где все вектора имеют длину D, является граничным случаем для заданных условий. В случае, когда длина векторов произвольная, но не превосходит D, утверждения 1 и 2 остаются справедливыми.

Воспользовавшись утверждением 2, переберём тройки векторов из входных данных. В каждой тройке векторов найдём два вектора, сумма (с правильно расставленными знаками) которых даст вектор длины не больше D. Объединим эти два вектора в один. Будем проделывать это до тех пор, пока не останется только два вектора. Согласно утверждению 1, любые два вектора длины не больше D можно сложить таким образом, чтобы длина полученного вектора не превосходила $\sqrt{2} \cdot D$, что и требуется по условию задачи.

Код решения (C++)

```
#define _CRT_SECURE_NO_WARNINGS
#pragma comment(linker, "/STACK:4000000")

#include <iostream>
#include <cstdio>
#include <string>
#include <vector>
#include <utility>
#include <algorithm>
#include <ctime>
#include <set>
```

```
using namespace std;

typedef struct vec2d {
    int x;
    int y;
    int length()
    {
        return x * x + y * y;
    }
} vec2d;

typedef struct node {
    bool r_sign;
    node *left;
    node *right;
    vec2d vec;
    int idx;

    node()
    {
        r_sign = true;
        left = 0;
        right = 0;
        vec.x = 0;
        vec.y = 0;
    }
} node;

vec2d sum_vec(const vec2d &a, const vec2d &b)
{
    vec2d vec;
    vec.x = a.x + b.x;
    vec.y = a.y + b.y;

    return vec;
}

vec2d dif_vec(const vec2d &a, const vec2d &b)
{
    vec2d vec;
    vec.x = a.x - b.x;
    vec.y = a.y - b.y;

    return vec;
}

bool sign;
bool signs[1000000];
```

```
void dfs(const node *nd)
{
    if (nd->left == 0 && nd->right == 0)
    {
        signs[nd->idx] = sign;
        return;
    }
    if (nd->left != 0)
    {
        dfs(nd->left);
    }
    if (nd->right != 0)
    {
        bool old_sign = sign;
        if (!sign)
            sign = !nd->r_sign;
        else
            sign = nd->r_sign;
        dfs(nd->right);
        sign = old_sign;
    }
}

int main()
{
    FILE *f_in = stdin;
    FILE *f_out = stdout;

    int n, l;
    fscanf(f_in, "%d\n%d", &n, &l);

    int maxl = l * l * 2;
    l *= l;

    vector <node *> tree;
    vector <node *> tree_org;
    tree.reserve(n);

    for (int i = 0; i < n; ++i)
    {
        vec2d v = {};
        fscanf(f_in, "%d%d\n", &v.x, &v.y);
        node *nd = new node;
        nd->vec = v;
        nd->idx = i;
        tree.push_back(nd);
        tree_org.push_back(nd);
    }

    while (tree.size() > 2)
```

```
    {
        vector <node *> tree_tmp;
        tree_tmp.reserve(tree.size());
        for (int i = 0; i < tree.size() - 1; ++i)
        {
            node *nd_cur = tree[i];
            for (int j = i + 1; j < tree.size(); j++)
            {
                node *nd_next = tree[j];

                if (sum_vec(nd_cur->vec, nd_next->vec).length() <= l)
                {
                    node *nd = new node;
                    nd->vec = sum_vec(nd_cur->vec, nd_next->vec);
                    nd->left = nd_cur;
                    nd->right = nd_next;
                    nd->r_sign = true;
                    nd_cur = nd;
                }
                else if (dif_vec(nd_cur->vec, nd_next->vec).length()
<= l)
                {
                    node *nd = new node;
                    nd->vec = dif_vec(nd_cur->vec, nd_next->vec);
                    nd->left = nd_cur;
                    nd->right = nd_next;
                    nd->r_sign = false;
                    nd_cur = nd;
                }
                else
                {
                    tree_tmp.push_back(nd_next);
                }
            }
            tree_tmp.push_back(nd_cur);
            if (tree_tmp.size() != tree.size())
            {
                break;
            }
        }
        tree.swap(tree_tmp);
    }

    node root;
    if (tree.size() > 1)
    {
        node *nd_cur = tree[0];
        node *nd_next = tree[1];

        if (sum_vec(nd_cur->vec, nd_next->vec).length() <=
```

```
dif_vec(nd_cur->vec, nd_next->vec).length())
        {
            root.vec = sum_vec(nd_cur->vec, nd_next->vec);
            root.left = nd_cur;
            root.right = nd_next;
            root.r_sign = true;
        }
        else
        {
            root.vec = dif_vec(nd_cur->vec, nd_next->vec);
            root.left = nd_cur;
            root.right = nd_next;
            root.r_sign = false;
        }
    }
    else
    {
        root = *tree[0];
    }

    fprintf(f_out, "Possible\n");

    sign = true;
    dfs(&root);
    for (int i = 0; i < n; i++)
    {
        fprintf(f_out, signs[i] ? "F" : "B");
    }

    return 0;
}
```

Код проверки:

```
import sys

sys.setrecursionlimit(20000)


def generate():
     tests = []
     return tests

def get_next(dataset, ptr, slen):
     while (ptr < slen and not((dataset[ptr] >= '0' and dataset[ptr]
<= '9') or dataset[ptr] == '-')):
          ptr += 1;
     if ptr == slen:
          return 0;
     mult = 1;
     if (dataset[ptr] == '-'):
          mult = -1;
          ptr += 1;
```

```
      n = 0;
      while (ptr < slen and dataset[ptr] >= '0' and dataset[ptr] <=
'9'):
            n = n * 10 + int(dataset[ptr]);
            ptr += 1;

      return n * mult, ptr + 1

def sum_vec(a, b):
      return {'x' : a['x'] + b['x'], 'y' : a['y'] + b['y']};

def dif_vec(a, b):
      return {'x' : a['x'] - b['x'], 'y' : a['y'] - b['y']};

def len_vec(a):
      return a['x'] * a['x'] + a['y'] * a['y'];

def node_init():
      return {
            'r_sign' : True,
            'left'   : None,
            'right'  : None,
            'vec'    : { 'x' : 0, 'y' : 0 },
            'idx'    : 0
      }

signs = [];
sign = False;

def dfs(nd):
      global signs;
      global sign;
      if (nd['left'] == None and nd['right'] == None):
            signs[nd['idx']] = sign;
      if (nd['left'] != None):
            dfs(nd['left']);
      if (nd['right'] != None):
            old_sign = sign;
            if (not sign):
                  sign = not nd['r_sign'];
            else:
                  sign = nd['r_sign'];
            dfs(nd['right']);
            sign = old_sign;

def solve(dataset):
      global sign;
      global signs;

      ptr = 0;
      slen = len(dataset);

      n, ptr = get_next(dataset, ptr, slen);
      l, ptr = get_next(dataset, ptr, slen);

      signs = [0] * n;
```

```
      maxl = l * l * 2;
      l *= l;

      tree = [];
      tree_org = [];

      for i in range(n):
            vx, ptr = get_next(dataset, ptr, slen);
            vy, ptr = get_next(dataset, ptr, slen);
            nd = node_init();
            nd['vec']['x'] = vx;
            nd['vec']['y'] = vy;
            nd['idx'] = i;
            tree.append(nd);
            tree_org.append(nd);

      while (len(tree) > 2):
            tree_tmp = [];
            len_tree = len(tree);
            for i in range(len_tree - 1):
                  nd_cur = tree[i];
                  for j in range(i + 1, len_tree):
                        nd_next = tree[j];
                        if len_vec(sum_vec(nd_cur['vec'],
nd_next['vec'])) <= l:
                              nd = node_init();
                              nd['vec'] = sum_vec(nd_cur['vec'],
nd_next['vec']);
                              nd['left'] = nd_cur;
                              nd['right'] = nd_next;
                              nd['r_sign'] = True;
                              nd_cur = nd;
                        elif len_vec(dif_vec(nd_cur['vec'],
nd_next['vec'])) <= l:
                              nd = node_init();
                              nd['vec'] = dif_vec(nd_cur['vec'],
nd_next['vec']);
                              nd['left'] = nd_cur;
                              nd['right'] = nd_next;
                              nd['r_sign'] = False;
                              nd_cur = nd;
                        else:
                              tree_tmp.append(nd_next);
                  tree_tmp.append(nd_cur);
                  if len(tree_tmp) != len(tree):
                        break;

            tree, tree_tmp = tree_tmp, tree;

      root = {};
      if (len(tree) > 1):
            nd_cur = tree[0];
            nd_next = tree[1];

            if len_vec(sum_vec(nd_cur['vec'], nd_next['vec'])) <=
len_vec(dif_vec(nd_cur['vec'], nd_next['vec'])):
                  root['vec'] = sum_vec(nd_cur['vec'], nd_next['vec']);
                  root['left'] = nd_cur;
```

```
                root['right'] = nd_next;
                root['r_sign'] = True;
          else:
                root['vec'] = dif_vec(nd_cur['vec'], nd_next['vec']);
                root['left'] = nd_cur;
                root['right'] = nd_next;
                root['r_sign'] = False;
     else:
          root = tree[0];

     sign = True;
     dfs(root);

     res = "Possible\n";
     for i in range(n):
          if (signs[i]):
                res += 'F';
          else:
                res += 'B';
     res += '\n';

     return res

def check(reply, clue):
     reply = [str(i) for i in reply.strip().split()]
     clue_ = [str(i) for i in clue.strip().split()]
     if (clue_[0] != 'HACK'):
          clue = '''5 7
0 7
-7 0
1 -1
-1 1
-6 3
'''

     if (reply[0] != 'Possible' or len(reply) != 2):
          return False;

     dirs = reply[1];

     ptr = 0;
     slen = len(clue);

     n, ptr = get_next(clue, ptr, slen);
     l, ptr = get_next(clue, ptr, slen);
     maxl = l * l * 2;

     if len(dirs) != n:
          return False;

     x = 0;
     y = 0;

     for i in range(n):
          if (dirs[i] != 'F' and dirs[i] != 'B'):
                return False;
          x_cur, ptr = get_next(clue, ptr, slen);
          y_cur, ptr = get_next(clue, ptr, slen);
```

```
        if (dirs[i] == 'F'):
              x += x_cur;
              y += y_cur;
        else:
              x -= x_cur;
              y -= y_cur;


return x * x + y * y <= maxl;
```