Разрязване на торта

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2 seconds
Memory limit:	512 megabytes

Тази задача може да се реши само на език за програмиране C++.

Организаторите на Закретата ученическа олимпиада по програмиране решили да поръчат огромна торта за тържествената вечеря. За целта, те поръчали торта с n свещички, номерирани от 0 до $n-1$. Известно е, че свещичките се намират в различни точки от равнината, никои тре свещички не лежат на една права, Никои четири свещички не образуват трапец(четириъгълник с две успоредни страни).

За да има парче торта за всеки, организаторите искат предварително да опишат начин за разрязване на тортата на максимален брой парчета. За съжаление, вие не знаете разположението на свещичките върху тортата, а единствения достъпен начин да се разбере, как изглежда тортата е кореспонденция със сладкърницата по електронната поща.

В едно съобщение можете да поискате да се претеглят не повече от четири триъгълни парчета, с върхове в свещички:

- Предоставяте два списъка с триъгълници `left` и `right`, общо съдържащи не повече от четири триъгълника. Триъгълниците може да се пресичат, да имат общи свещи и даже да съвпадат.
- В сладкарницата ще изрежаттриъгълни парчета, съответстващи на изпратения списък от няколко копия на тортата и ще сложат парченцата от `left` на лявата чаша на везната, а парченцата от `right` на дясната чаша на везната.
- След това ви изпращат резултата от претеглянето. Приема се, че теглото на парчето е равно на неговата площ. В резултат на това вие ще разберете, в кой от двата списъка от триъгълници общата площ е по-голяма, или че площите съвпадат.

След няколко претегляния, вие трябва да можете да намерите начин за разрязване на тортата. Всяко парче трябва да бъде изпъкнал многоъгълник с върхове в свещички. Върховете на дадено парче трябва да бъдат изброени по посока на часовниковата стрелка или обратно на часовниковата стрелка. Никои две парчета не трябва да се пресичат във вътрешна точка. Трябва да върнете списък с парчета, чиято обща площ е възможно най-голяма. В някои групи може също да искате да увеличите максимално броя на парчетата.

Solution Format

Това е нестандартна задача. Формата на тестването и е с грейдър и затова вие трябва да реализирате само функция `solve` с решението. Тази функция ще бъде извиквана от тестващата програма (грейдър) на журито и стойността, която ще върне функцията се ще приема като решение на задачата.

В частност, това означава, че в изпратеният от вас код**не трябва да има вход и изход**. Във вашия код **не трябва** да има функция `main`. При необходимост може да реализирате произволен брой помощни функции, структури, класове и глобални променливи, но целият код на вашето решение трябва да бъде в един файл.

Вие трябва да реализирате следната функция:

```
std::vector<std::vector<int>> solve(int n);
```

Функцията `solve` приема като параметър единствено цяло число n — броя на свещичките.

При реализацията на функцията `solve` може да използвате функцията `compare`, която реализира грейдъра:

```
int compare(const std::vector<Triangle>& left, const std::vector<Triangle>& right);
```

Тази функция приема като параметри два непразни списъка от триъгълници, с общ размер не повече от 4. Като резултат тя връща -1, ако общата площ на триъгълниците от `left` е по-малка от общата площ на триъгълниците от `right`, 0, ако площите са равни и 1, в третия вариант. Ако е направена некоректна заявка или бъде достигнат максималния брой заявки, програмата автоматично завършва своята работа. За да получите достъп до функцията `compare` в решението, на първия ред на вашия код трябва да включите заглавния файл на грейдъра:

```
#include "triangles.h"
```

В този файл е декларирана и използваната във функцията `compare` структура `Triangle`:

```
struct Triangle {
    int i, j, k;

    Triangle() = default;
    Triangle(int i_, int j_, int k_) : i(i_), j(j_), k(k_) {}
};
```

Дадената структура описва едно триъгълно парче, като член-данните `i`, `j` и `k` описват номерата на свещичките, образуващи върховете на триъгълното парче. Номерата на свещичките трябва да са различни за едно триъгълно парче, но могът да се повтарят в няколко триъгълника.

В кода на изпращаното решение не трябва да включвате декларация на структурата `Triangle`, защото тя ще се вземе автоматично от заглавния файл.

Всички параметри (индексите на свещичките в заявките, а също и връщаният от вас резултат) се задава в **0-индексация**.

Вашата функция `solve` с помощта на извикването на функцията `compare` трябва да намери произволно разделяне на тортата на изпъкнали парчета, в които кои да е две парчета не се пресичат във вътрешна точка, Общата площ на парчетата е максимална, и, возможно е, броят на парчетата е максимален.

Функцията `solve` трябва да върне списък от парчетата, на които се разделя тортата. Всяко парче се описва със стуктурата `std::vector<int>`, който се състои от свещички, образуващи изпъкнал многоъгълник. Свещичките трябва да са изброени **подред по границите на многоъгълника** (по часовниковата стрелка или обратно на часовниковата стрелка). Кои зда е две парчета не трябва да се пресичат във вътрешна точка. Трябва да върнете като резултат списък с парчетата, общата площ,на които е максимално възможна, а в някои групи допълнително се изисква да се максимизира броя парчета, при това съхранявайки условието за максимална възможна обща площ.

При едно стартиране на грейдъра **журито прави точно едно обръщение към функцията `solve`**. **Грейдърът не е адаптивен**, т.е. положенията на свещичките се фиксират предварително и не зависят от реализацията на функцията `solve`.

Testing

Предоставен ви е шаблон на решението `triangles.cpp`, а също и заглавния файл `triangles.h`, съдържащ декларации на функциите `solve` и `compare`. За удобство при тестването ви е предоставен грейдър във файла `grader.cpp`.В този файл е реализирано въвеждане на входни данни от стандартния вход, извикване на функцията `solve` и извеждане на стандартния изход на върнатата от функцията `solve` стойност.В тестващата система грейдърът може да е различен.

За да компилирате вашия код `triangles.cpp` на езика C++, използвайте командата

```
g++ -std=c++20 grader.cpp triangles.cpp -o grader
```

След изпълнение на тази команда ще бъде създаден изпълним файл на грейдъра `grader` или `grader.exe`, в зависимост от вашата операционна система, след стартирането на който може да

се въведе тест във формата, указан по-долу.

Ако компилирането чрез команда ви е трудно, за локално тестване може да копирате функцията `solve` в файла `grader.cpp` (трябва да я поставите преди функцията `main`) и да стартирате файла `grader.cpp`. При това, преди да изпратите вашето решение към системата за тестване, в него трябва да остане само реализацията на функцията `solve`, **като не забравяте да включите заглавния файл в началото на кода** (това става с помощта на реда `#include "triangles.h"`).Ако използвате някакви библиотеки на езика C++, тяхното включване също трябва да бъде добавено в началото на изпращания код.

В случай, че получите съобщение за грешка при компилация, убедете се, че **в изпратеният от вас код не се съдържа функция `main`, нито декларации на функцията `compare` или структурата `Triangle`**. Функцията `compare` и структурата `Triangle` може просто да използвате в изпратения от вас код.

Input

Грейдърът чете тест в следния формат:

На първия ред се намира цяло число n ($4 \leq n \leq 10\,000$) — броя на свещичките.

На следващите n реда са зададени по две цели числа x_i, y_i ($-10^9 \leq x_i, y_i \leq 10^9$) — координатите на i-тата свещичка.

Output

Грейдърът извежда резултатът, върнат от функцията `solve` — списък от намерените парчета.

На първия ред се извежда броя на намерените многоъгълници (размера на вектора, който връща функцията `solve`).

На следващите редове за всеки многоъгълник (елемент на вектор от вектора, който връща функцията `solve`) в началото се извежда размера на многоъгълника, а на следващия ред номарата на свещичките, които са върхове на този многоъгълник (елементите на всеки от `std::vector<int>`, включен във върнатата от функцията `solve` стойност). Всеки многоъгълник трябва да бъде изпъкнал, неговите върхове трябва да са изброени последователно (по часовниковата стрелка или срещу часовниковата стрелка). Никои два многоъгълника не трабва да се пресичат във вътрешна точка.

Във файла `grader.cpp` има променлива `verbose`, която в началото е равна на 0. При увеличение на нейната стойност грейдърът ще извежда по-подробна информация за вашето решение и неговите заявки.

Examples

standard input	standard output
4 1 2 1 4 0 0 3 -1	3 3 0 1 2 3 0 2 3 3 0 1 3
5 -1 -1 4 4 4 -2 1 2 -2 2	1 4 0 4 1 2
6 2 2 0 -2 -1 3 -2 0 7 0 2 -3	4 3 2 4 3 3 3 4 5 3 1 3 5 3 0 2 4

Note

В първия пример възможна последователност на обръщения към функциите може да изглежда така:

Отначало се извиква функцията `solve(4)`.

От функцията `solve` се извиква функцията

`compare({Triangle(0, 1, 2)}, {Triangle(1, 2, 3)})}`

По време на изпълнението на функцията се прави сравнение на размерите на триъгълните парчета, образувани от триъгълник с върхове в свещичките с номера 0, 1 и 2 и триъгълник с върхове в свещичките с номера 1, 2, 3.

Тъй като първият триъгълник е по-малък от втория, функцията връща -1.

След това във функцията `solve` се извиква функцията

`compare({Triangle(1, 2, 3)}, {Triangle(0, 1, 2), Triangle(0, 1, 3)})}`

В отговор ще бъде върнато 1, тъй като площта на триъгълника с върхове в свещичките 1, 2 и 3 е по-голяма от общата площ на триъгълника с върхове в свещичките 0, 1 и 2 и на триъгълника с върхове в свещичките 0, 1 и 3.

След това във функцията `solve` се извиква функцията

`compare({Triangle(1, 2, 3)}, {Triangle(0, 1, 2), Triangle(2, 3, 0), Triangle(0, 1, 3)})}`

В отговор ще бъде върнато 0.

Накрая функцията `solve` връща`{{0, 1, 2}, {0, 2, 3}, {0, 1, 3}}` — описанието на разделянето на тортата на изпъкнали парчета, имащи максимално възможна обща площ и не пресичащи се във вътрешни точки.

В първия и третия пример, в намерения начин да се разреже тортата броят на парчетата е максималният възможен, затова тези отговори ще се приемат за коректни в групите с максимизация. Във втория тест броят на парчетата не е максималния възможен, затова този отговор ще се приеме за коректен в групите без максимизация, но ще получи 0 точки в групите с максимизация.

Обърнете внимание, че първият тест попада под допълнителните ограничения на групите 1 и 2, а третия тест попада под допълнителните ограничения на групите 5.

Scoring

Тестовете към тази задача се състоят от 11 групи. Правилата, по които се присъждат точките за групите са описани по-долу.Обърнете внимание, преминаването на тестовете от условието не е необходимо за някои от групите. Общия брой точки за всяка група е равен на максималния брой точки, получени за тази група тестове от всички събмити. Точките за решението за група тестове са равни на минималния брой точки, получени на всички тестове в групата.

- Ако вашето решение направи некоректно извикване на `compare` или върне отговор неудовлетворяващ описаните условия, то ще получи 0 точки за теста.
- Във всеки тест на вашето решение ще бъдат позволени не повече от $2 \cdot 10^6$ извиквания на функцията `compare`. В случай на превишаване на лимита, вашето решение ще получи 0 точки за този тест.
- В част от групите се изисква да се максимизира броя на парченцата в отговора. В тези групи, ако броят на парчетата не е максималния възможен, вашето решение ще получи 0 точки за този тест.

Ако нито едно от описаните условия в теста не е нарушено, отговорът на теста се приема за правилен. Нека пълният брой точки за група тестове е p. Част от групите се оценяват с използване на формули, а останалите се оценяват без използване на формули:

- Ако групата се оценява без използване на формули, точките за теста са p.
- Ако групата се оценява с използване на формули, нека q е броят на извиквания на функцията `compare`, което е направило вашето решение. Тогава точките за теста са $\left\lfloor p \cdot \frac{20n}{\max(q, 20n)} \right\rfloor$.

Група	Пълен брой точки	Оценка		Доп. ограничения	Необх. групи	Коментари
		Макс.	Формула	n		
0	0	не	не	–	–	Тестовете от условието.
1	13	не	не	$n = 4$	–	
2	4	да	не	$n = 4$	1	
3	13	не	не	–	–	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ има в множеството, останалите точки са случайни вътре в този триъгълник
4	8	да	не	–	3	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ има в множеството, останалите точки са случайни вътре в този триъгълник
5	11	да	да	–	–	точките са върхове на изпъкнал многоъгълник
6	9	не	не	$n \leq 100$	–	случайни точки
7	8	да	не	$n \leq 100$	6	случайни точки
8	9	не	да	–	–	случайни точки
9	8	да	да	–	–	случайни точки
10	9	не	да	–	–	
11	8	да	да	–	–	

В групите 6 – 9 всички точки са избрани случайно и независимо от квадрата $[-10^9, 10^9] \times [-10^9, 10^9]$.

В групите 3 – 4 точките, освен $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$, са избрани случайно и независимо от триъгълника с върхове $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$.

В тези групи от всички случайни тестове са оставени само тези, в които всички точки са различни, никои три не лежат на една права и никои четири не образуват трапец.