Разрезание торта

Имя входного файла:	`стандартный ввод`
Имя выходного файла:	`стандартный вывод`
Ограничение по времени:	2 секунды
Ограничение по памяти:	512 мегабайт

Данную задачу можно решать только на языке программирования C++.

Организаторы Закрытой олимпиады школьников по программированию решили заказать большой торт на вечерний банкет. Для этого они заказали торт с n свечками, пронумерованными от 0 до $n-1$. Известно, что свечки находятся в различных точках на плоскости, никакие три свечки не лежат на одной прямой, никакие четыре свечки не образуют трапецию (четырехугольник с двумя параллельными сторонами).

Чтобы всем достался кусочек торта, организаторы хотят заранее построить способ разрезать торт на максимальное число кусочков. К сожалению, вам неизвестно расположение свечек на торте, а единственный доступный способ узнать, как выглядит торт — переписка с кондитерской по электронной почте.

В одном сообщении вы можете попросить взвесить не более четырех треугольных кусочков, с вершинами в свечках:

- Вы предоставляете два списка треугольников `left` и `right`, суммарно содержащих не более четырех треугольников. Треугольники могут пересекаться, содержать общие свечки и даже совпадать.
- В кондитерской вырежут треугольные кусочки в соответствии с запрошенными списками из нескольких копий торта и положат кусочки из `left` на левую чашу весов, кусочки из `right` на правую чашу весов.
- Вам отправят результат взвешивания. Будем считать, что вес кусочка равен его площади. В результате вы узнаете, в каком из списков треугольников суммарная площадь больше, или что площади совпадают.

После нескольких взвешиваний вы должны найти способ разрезать торт. Каждый кусочек должен быть выпуклым многоугольником с вершинами в свечках. Вершины кусочка должны быть перечислены в порядке обхода по или против часовой стрелки. Любые два кусочка не должны пересекаться по внутренним точкам. Вы должны вернуть список кусочков, суммарная площадь которых максимально возможная. Также в некоторых группах требуется максимизировать число кусочков.

Формат решения

Это необычная задача. Она имеет формат тестирования с грейдером, где вам необходимо реализовать только функцию `solve` с решением. Эта функция будет вызвана тестирующей программой жюри (грейдером), и возвращаемое значение функции будет принято как решение задачи.

В частности, это означает, что в отправленном вами коде **не должно быть ввода или вывода**. Ваш код **не должен** содержать функцию `main`. При необходимости вы можете реализовать произвольное количество вспомогательных функций, структур, классов и глобальных переменных, но весь код вашего решения должен находиться в одном файле.

Вы должны реализовать следующую функцию:

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

Функция `solve` принимает на вход единственное число n — количество свечек.

В реализации функции `solve` вы можете использовать функцию `compare`, которую реализует грейдер:

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

Данная функция принимает на вход два непустых списка треугольников суммарного размера не более 4. В качестве результата она возвращает -1, если суммарная площадь треугольников в `left` меньше суммарной площади треугольников в `right`, 0, если площади равны и 1, иначе. Если сделать

некорректный запрос или исчерпать лимит на количество запросов, то программа автоматически завершит свою работу.

Для того, чтобы получить доступ к функции `compare` в решении, первой строчкой вашего кода нужно подключить заголовочный файл следующей строчкой:

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

В этом же файле определена используемая в функции `compare` структура `Triangle`:

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

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

Данная структура описывает один треугольный кусочек, параметры `i`, `j` и `k` описывают номера свечек, образующих вершины треугольного кусочка. Номера свечек должны быть различными для одного треугольного кусочка, но могут повторяться в нескольких треугольниках.

В код отправляемого решения вам не надо включать определение структуры `Triangle`, оно будет автоматически браться из заголовочного файла.

Все параметры (индексы свечек в запросах, а также в возвращаемом вами результате) задаются в **0-индексации**.

Ваша функция `solve` с помощью вызовов функции `compare` должна найти любое разбиение торта на выпуклые кусочки, в котором любые два кусочка не пересекаются по внутренним точкам, суммарная площадь кусочков максимальна, и, возможно, количество кусочков максимально.

Функция `solve` должна вернуть список кусочков, на которые разбивается торт. Каждый кусочек описывается структурой `std::vector<int>`, состоящей из свечек, образующих выпуклый многоугольник. Свечки должны быть перечислены **в порядке следования по границе многоугольника** (по часовой стрелке или против часовой стрелки). Любые два кусочка не должны пересекаться по внутренним точкам. Вы должны вернуть список кусочков, суммарная площадь которых максимально возможная, а в некоторых группах дополнительно требуется максимизировать количество кусочков, при этом сохраняя условие на максимальную возможную суммарную площадь.

За один запуск грейдера **жюри сделает ровно один вызов функции** `solve`. **Грейдер не является адаптивным**, то есть положения свечек фиксируются заранее и не зависят от реализации функции `solve`.

Тестирование

Вам предоставлен шаблон решения `triangles.cpp`, а так же заголовочный файл `triangles.h`, содержащий определение функций `solve` и `compare`. Для удобства тестирования вам предоставлен грейдер — файл `grader.cpp`. В этом файле реализован ввод входных данных со стандартного потока ввода, запуск функции `solve` и вывод на стандартный поток вывода возвращаемого значения функции `solve`. В тестирующей системе грейдер может отличаться.

Чтобы скомпилировать ваш код `triangles.cpp` на языке C++, используйте команду

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

После выполнения данной команды будет создан исполняемый файл грейдера `grader` или `grader.exe`, в зависимости от вашей операционной системы, запустив который можно ввести тест в формате, указанном далее.

Если компиляция через команды вызывает у вас трудности, для локального тестирования вы можете скопировать реализацию функции `solve` в файл `grader.cpp` (ее необходимо вставить перед функцией `main`) и запустить файл `grader.cpp`. При этом перед отправкой решения в тестирующую систему вам нужно будет оставить только реализацию функции `solve`, **не забыв подключить заголовочный файл в начале кода** (это делается с помощью добавления строки `#include "triangles.h"` в начало сдаваемого вами кода). Если вы используете какие либо библиотеки языка C++, их подключение также необходимо добавить в начало отправляемого кода.

В случае получения вердикта ошибка компиляции, убедитесь, что **в отправляемом вами коде не содержится функции `main`, а также не содержатся определения функции `compare` и структуры `Triangle`**. Функцию `compare` и структуру `Triangle` вы можете только использовать в отправляемом вами коде.

Формат входных данных

Грейдер читает тест в следующем формате:

В первой строке находятся целое число n ($4 \leqslant n \leqslant 10\,000$) — количество свечек.

В следующих n строках находится по два целых числа x_i, y_i ($-10^9 \leqslant x_i, y_i \leqslant 10^9$) — координаты i-й свечки.

Формат выходных данных

Грейдер выводит результаты функции `solve` — список найденных кусочков.

В первой строке выводится число найденных многоугольников (размер вектора, возвращаемого функцией `solve`).

В следующих строках для каждого многоугольника (элемента вектора векторов, возвращаемого функцией `solve`) в начале выводится размер многоугольника, а в следующей строке номера свечек, являющихся вершинами многоугольника (элементы каждого из `std::vector<int>`, входящих в возвращаемое функцией `solve` значение). Каждый многоугольник должен быть выпуклым, его вершины должны быть перечислены в порядке следования (по часовой стрелке или против часовой стрелки). Никакие два многоугольника не должны пересекаться во внутренних точках.

В файле `grader.cpp` есть переменная `verbose`, которая исходно равна 0. При увеличении ее значения грейдер будет более подробно писать информацию о вашем решении и его запросах.

Примеры

стандартный ввод	стандартный вывод
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

Замечание

В первом примере возможная последовательность вызовов функций могла выглядеть так:

В начале вызывается функция `solve(4)`.

Из функции `solve` вызывается функция

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

Во время выполнения функции происходит сравнения размеров треугольных кусочков, образованных треугольником с вершинами в свечках с номерами 0, 1 и 2 и треугольником с вершинами в свечках 1, 2, 3.

Так как первый треугольник меньше второго, то функция возвращает -1.

Далее из функции `solve` вызывается функция

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

В ответ будет возвращено 1, так как площадь треугольника с вершинами в свечках 1, 2 и 3 больше суммарной площади треугольника с вершинами в свечках 0, 1 и 2 и треугольника с вершинами в свечках 0, 1 и 3.

Далее из функции `solve` вызывается функция

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

В ответ будет возвращен 0.

В конце функция `solve` возвращает `{{0, 1, 2}, {0, 2, 3}, {0, 1, 3}}` — описание разделения торта на выпуклые кусочки, имеющие максимальную возможную суммарную площадь и не пересекающихся во внутренних точках.

В первом и третьем примере, в найденном способе разрезать торт количество кусочков максимально возможное, поэтому такие ответы считались бы корректными в группах с максимизацией. Во втором тесте количество кусочков не максимально возможное, поэтому такой ответ считался бы корректным в группах без максимизации, но получил бы 0 баллов в группах с максимизацией.

Обратите внимание, что первый тест подходит под дополнительные ограничения групп 1 и 2, а третий тест подходит под дополнительные ограничения группы 5.

Система оценки

Тесты к этой задаче состоят из 11 групп. Правила, по которым выставляются баллы за группы описаны ниже. Обратите внимание, что прохождение тестов из условия не требуется для некоторых групп. Итоговый балл за каждую группу равняется максимальному баллу, полученному за эту группу тестов по всем отправленным посылкам.

Балл решения за группу тестов равен минимальному баллу, полученному по всем тестам группы.

- Если ваше решение сделает некорректный вызов `compare` или вернет ответ, не удовлетворяющий обязательным условиям, оно получит 0 баллов за тест.
- В каждом тесте вашему решению разрешается выполнить не более $2 \cdot 10^6$ вызовов функции `compare`. В случае превышения лимита ваше решение получит 0 баллов за тест.
- В части групп требуется максимизировать число кусочков в ответе. В таких группах, если число кусочков будет не максимально возможным, ваше решение получит 0 баллов за тест.

Если ни одно из описанных условий для теста не нарушено, то ответ за тест считается корректным. Пусть полный балл за группу тестов равен p. Часть групп оценивается с использованием формулы, часть оценивается без использования формулы:

- Если группа оценивается без использования формулы, балл за тест равен p.
- Иначе, если группа оценивается с использованием формулы, пусть q это количество вызовов `compare`, которое сделало ваше решение. Тогда балл за тест равен $\left\lfloor p \cdot \frac{20n}{\max(q, 20n)} \right\rfloor$.

Группа	Полный балл	Оценка		Доп. ограничения	Необх. группы	Комментарий
		Макс.	Формула	n		
0	0	нет	нет	–	–	Тесты из условия.
1	13	нет	нет	$n = 4$	–	
2	4	да	нет	$n = 4$	1	
3	13	нет	нет	–	–	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ есть в множестве, остальные точки случайные внутри этого треугольника
4	8	да	нет	–	3	$(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$ есть в множестве, остальные точки случайные внутри этого треугольника
5	11	да	да	–	–	точки являются вершинами выпуклого многоугольника
6	9	нет	нет	$n \leqslant 100$	–	точки случайные
7	8	да	нет	$n \leqslant 100$	6	точки случайные
8	9	нет	да	–	–	точки случайные
9	8	да	да	–	–	точки случайные
10	9	нет	да	–	–	
11	8	да	да	–	–	

В группах 6 – 9 все точки были выбраны случайно и независимо из квадрата $[-10^9, 10^9] \times [-10^9, 10^9]$.

В группах 3 – 4 точки, кроме $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$, были выбраны случайно и независимо из треугольника с вершинами $(10^9, -10^9)$, $(-10^9, 10^9)$, $(10^9, 10^9)$.

В этих группах среди таких случайных тестов оставлены только те, в которых все точки различные, никакие три не лежат на одной прямой, никакие четыре не образуют трапецию.