Zadanie: ZEK
Radary

polish

ONTAK 2023, dzień 6. Dostępna pamięć: 256 MB. Limit czasu: 10 s. *06.07.2023*

Imperium Galaktyczne nie jest z Ciebie zadowolone. Z tajnego kompleksu więziennego na zarządzanej przez Ciebie planecie *Narkina 5* uciekło q więźniów. Wieść głosi, że pracowali oni, w warunkach nadzwyczajnej tajności oraz efektywnego zarządzania produkcją, przy konstrukcji nowego bojowego projektu Imperium*. Jeśli szybko się nie odnajdą, może się Tobą zainteresować Imperialna Służba Bezpieczeństwa, a oni uczynią Twoje życie bogatym w mocne wrażenia (oraz znacznie krótszym).

Na szczęście, masz jeszcze jedną kartę w zanadrzu: więźniowie wciąż noszą ślady rzadkich metali używanych przy produkcji, po których można ich namierzyć. Do tego celu posiadasz na planecie system n nowoczesnych radarów. Pojedyncze użycie systemu polega na wybraniu pewnego podzbioru radarów będących wierzchołkami wielokąta wypukłego. Następnie, dla takiego wielokąta, radary określają, czy więzień jest wewnątrz wielokąta, czy nie. Budżet na radary nie jest jednak nieograniczony. Dla każdego więźnia, po wykonaniu co najwyżej k takich zapytań, musisz podać grupie pościgowej odpowiedź – namiary na więźnia. Będzie to pewien końcowy zbiór radarów, również będący wielokątem wypukłym takim, że pozycja więźnia na pewno jest wewnątrz wielokąta, a dodatkowo pozycja żadnego innego radaru nie jest wewnątrz wielokąta.

Do dzieła! Działaj na chwałę Galaktycznego Imperium (a także, aby zachować życie).

Komunikacja

To zadanie jest interaktywne. Należy napisać program, który będzie zarządzał systemem radarowym i zlokalizuje każdego z więźniów. W Twoim programie **nie może być funkcji `main()`**. Działaniem programu będzie sterowała biblioteka, która będzie się komunikowała z Twoim programem. Na początku programu musisz dołączyć nagłówek `zeklib.h` za pomocą dyrektywy:

```
#include "zeklib.h"
```

W Twoim programie muszą być zaimplementowane następujące dwie funkcje:

- `void setRadars(std::vector<std::pair<int, int> > radars, int k, int q);` – Biblioteka wywoła tę funkcję dokładnie raz, na początku działania (czyli przed wszystkimi innymi wywołaniami). Argumentami funkcji są pozycje wszystkich radarów (podane jako lista n par liczb całkowitych; każda para to współrzędne jednego radaru), oraz dozwolona liczba zapytań dla jednego więźnia k oraz liczba więźniów q. Użyj tej funkcji, aby (między innymi) zapamiętać pozycje radarów.
- `std::vector<int> localize();` – Ta funkcja zostanie wywołana q razy, raz dla każdego więźnia. Powinna zwrócić wektor liczb całkowitych oznaczających indeksy radarów (na liście radarów podanej wyżej), które tworzą wielokąt wypukły zawierający we wnętrzu pozycję więźnia (ale nie zawierający żadnego innego radaru). Wielokąt musi być zakreślony zgodnie lub przeciwnie do ruchu wskazówek zegara. Radary są ponumerowane od 1 do n. W implementacji tej funkcji możesz „zapytać” system radarowy za pomocą funkcji `isInside()` opisanej niżej.

Biblioteka udostępnia Ci jedną funkcję:

- bool `isInside(std::vector<int> pol);` – Tę funkcję możesz wykonać k razy dla każdego więźnia, czyli wewnątrz jednego `localize()`. Podajesz jako argument listę indeksów radarów, które chcesz zapytać o pozycję więźnia. Radary powinny tworzyć wielokąt wypukły i wielokąt musi być zakreślony zgodnie lub przeciwnie do ruchu wskazówek zegara, a funkcja zwróci **true** wtedy i tylko wtedy, gdy więzień jest wewnątrz tego wielokąta.

Wszelkie niezgodności ze specyfikacją komunikacji będą skutkowały złą odpowiedzią. Twój program nie powinien czytać ze standardowego wejścia, a tym bardziej starać się ingerować w działanie biblioteczki. Grozi to co najmniej złą odpowiedzią.

Biblioteka

Możesz założyć, że:

- żadne trzy radary nie leżą na jednej prostej;

*O projekcie wiesz tylko tyle, że jest duży, okrągły i nie jest księżycem.

- pozycja więźnia została ustalona z góry (biblioteka nie będzie adaptować się do działania Twojego programu);
- pozycja więźnia nie leży na jednej prostej z żadnymi dwoma spośród radarów.
- każdy więzień jest z pewnością osiągalny, czyli gwarantowano jest że odpowiedź istnieje
- wszystkie współrzędne (więźniów i radarów) leżą w zakresie od -10^9 do 10^9.

Ocenianie

Twoje rozwiązanie musi dla każdego testu zmieścić sie w limicie k zapytań `isInside()` dla każdego więźnia, oraz zwrócić prawidłowy wielokąt wypukły jako wynik funkcji `localize()`. Jeśli zrobi to we wszystkich testach w danym podzadaniu, otrzyma za nie pełną liczbę punktów, a w przeciwnym razie – 0 punktów. We wszystkich testach zachodzi $n \geq 3$ oraz $q \geq 1$. Zestaw testów dzieli się na następujące podzadania:

Podzadanie	Ograniczenia	Punkty
1	$n \leq 6$, $q \leq 50$, $k = 4\,000$	15
2	$n \leq 20$, $q \leq 50$, $k = 4\,000$	17
3	$n \leq 60$, $q \leq 400$, $k = 4\,000$	9
4	$n \leq 300$, $q \leq 1\,000$, $k = 600$	9
5	$n \leq 5\,000$, $q \leq 10$, $k = 10\,000$	10
6	$n \leq 300$, $q \leq 1\,000$, $k = 250$	10
7	$n \leq 1\,000$, $q \leq 1\,000$, $k = 200$	10
8	$n \leq 1\,000$, $q \leq 2\,000$, $k = 60$	10
9	$n \leq 2\,500$, $q \leq 2\,000$, $k = 40$	10

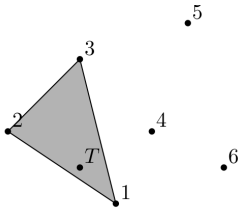
Przykładowe działanie programu

W opisanym poniżej teście jest $n = 6$ radarów, $q = 4$ więźniów, a limit zapytań $k = 5$. Wiersze tabelki z symbolem „→” oznaczają, że biblioteka wywołuje funkcje Twojego programu, natomiast „←” oznacza, że Twój program komunikuje się z biblioteką. Cała sytuacja przedstawiona jest na rysunkach poniżej.

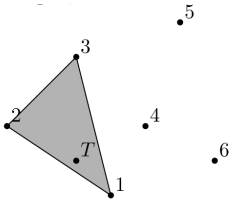
Wywołana funkcja	Wynik	Opis
→ `set_radars({(3,0), (0,2), (2,4), (4,2), (5,5), (6,1)}, 10, 4)`	—	Biblioteka podała listę radarów oraz wartości q i k.
→ `localize()`	...	Biblioteka wywołała `localize()` i czeka, aż program zwróci wartość.
← `isInside({1,2,3})`	**true**	Więzień znajduje się w środku trójkąta $(1, 2, 3)$.
← *return* `{1,2,3}`;	—	Program zwraca odpowiedź $(1, 2, 3)$ jako wynik `localize()`.
→ `localize()`	...	Drugie wywołanie `localize()` dla drugiego więźnia.
← `isInside({1,2,3})`	**false**	Więźnia nie ma w trójkącie $(1, 2, 3)$.
← `isInside({1,2,3,4})`	**true**	Więzień jest w czworokącie $(1, 2, 3, 4)$.
← *return* `{1,3,4}`;	—	Program zwraca odpowiedź $(1, 3, 4)$ dla drugiego więźnia.
→ `localize()`	...	Trzeci więzień.
← `isInside({2,3,5,4,1})`	**false**	
← `isInside({4,5,6})`	**false**	
← *return* `{1,4,6}`;	—	
→ `localize()`	...	Czwarty więzień.
← `isInside({1,3,5,6})`	**false**	
← `isInside({4,3,5,6})`	**true**	
← *return* `{4,3,5,6}`;	—	

Na rysunkach zilustrowana jest powyższa interakcja programu i biblioteki. Kolejne wiersze oznaczają kolejnych więźniów, zaznaczonych na rysunku jako punkt T. Ostatni wielokąt w każdym wierszu to odpowiedź programu.

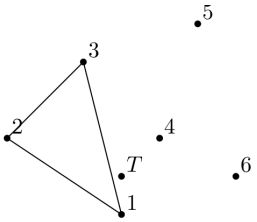
Więzień 1, pytanie 1



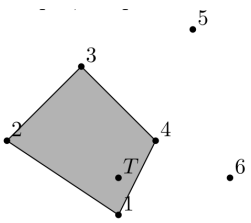
Więzień 1, odpowiedź



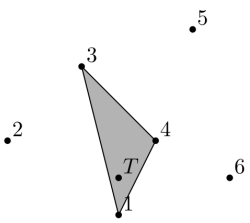
Więzień 2, pytanie 1



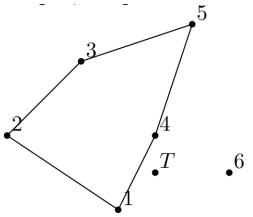
Więzień 2, pytanie 2



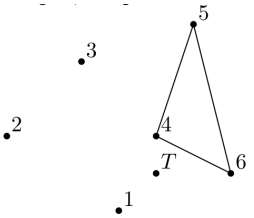
Więzień 2, odpowiedź



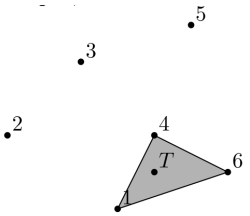
Więzień 3, pytanie 1



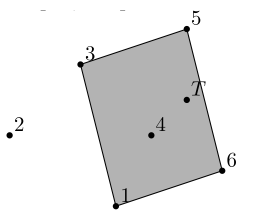
Więzień 3, pytanie 2



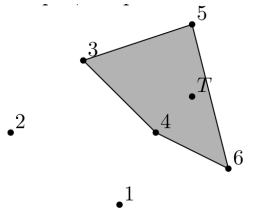
Więzień 3, odpowiedź



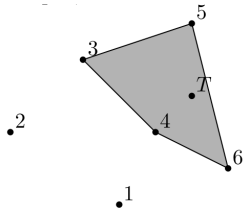
Więzień 4, pytanie 1



Więzień 4, pytanie 2



Więzień 4, odpowiedź



Eksperymenty

Przykład **nieprawidłowego** rozwiązania oraz przykład biblioteki można znaleźć w archiwum w sekcji *Pliki* na SIO_2. Zachowanie biblioteki może różnić się od tego, które będzie używane do oceny rozwiązań, i może nie spełniać warunków problemu. Powinno tylko pokazać sposób interakcji z programem.

Rozwiązanie `zek.cpp` można skompilować w następujący sposób:

```
g++ -O3 -static -o zek zek.cpp zeklib.cpp -std=c++17
```

Należy upewnić się, że pliki `zeklib.h` i `zeklib.cpp` znajdują się w tym samym folderze co rozwiązanie.
Po kompilacji, biblioteka przyjmuje na standardowym wejściu opis testu:

$n\ k$
$x_1\ y_1$
...
$x_n\ y_n$
q

Biblioteka wypisuje na ekran pytania zadawane przez program użytkownika `zek.cpp` w formacie:

? $m\ ind_1,\ ind_2,\ \ldots ind_m$

gdzie m to długość listy radarów, a ind_j – indeksy radarów. Jest to pytanie `isInside`$(\{ind_1, ind_2, \ldots, ind_m\})$, na które należy wpisać z klawiatury `Yes` lub `No`.
Odpowiedź, którą podał program `zek.cpp` będzie zapisana w podobnym formacie ! $m\ ind_1,\ ind_2,\ \ldots ind_m$.
Plik wejściowy dla przykładowego testu można znaleźć w archiwum w pliku `zek0.in`.

Завдання: ZEK
Radary

ukrainian

ONTAK 2023, день 6. Обмеження пам'яті: 256 MB. Ліміт часу: 10 s. *06.07.2023*

Галактична Імперія незадоволена тобою. З секретного тюремного комплексу на планеті *Зона номер 5*, що керується тобою, втекло q зеків. Ходять чутки, що вони працювали, за умови надзвичайної секретності та ефективного управління виробництвом, над конструкцією нового бойового проекту Імперії*. Якщо їх швидко не знайти, тобою може зацікавитися Служба Безпеки Універсума, і вони зроблять твоє життя багатим на сильні враження (та значно коротшим).

На щастя, у тебе ще є одна карта в рукаві: зеки все ще мають в крові сліди рідкісних речовин, які використовувались на зоні, і за якими зеків можна відстежити. Для цієї мети у тебе на планеті є система з n сучасних радарів. Одиничне використання системи полягає у виборі певного підмножини радарів, які є вершинами випуклого багатокутника. Потім, для такого багатокутника, радари визначають, чи є зек всередині багатокутника, чи ні. Бюджет на радари, однак, не є необмеженим. Для кожного зека, після виконання максимум k таких запитів, ти повинен надати відповідь групі переслідувачів – геопозиція зека. Це буде певна множина радарів, яка також є випуклим багатокутником таким, що позиція зека обов'язково є всередині багатокутника, а також позиція жодного іншого радара не є всередині багатокутника.

До роботи! Дій на славу Галактичної Імперії (а також, щоб зберегти життя).

Комунікація

Це завдання є інтерактивним. Треба написати програму, яка буде керувати системою радарів і знаходити кожного з зеків. У вашій програмі **не може бути функції** `main()`. Роботою програми буде керувати бібліотека, яка буде спілкуватися з вашою програмою. На початку програми ви повинні додати заголовок `zeklib.h` за допомогою директиви: `#include "zeklib.h"`

У вашій програмі повинні бути реалізовані наступні дві функції:

- `void setRadars(std::vector<std::pair<int, int> > radars, int k, int q);` – Бібліотека викличе цю функцію лише один раз, на початку роботи (тобто перед усіма іншими викликами). Аргументами функції є позиції всіх радарів (подані як список n пар цілих чисел; кожна пара - це координати одного радара), а також дозволена кількість запитів для одного зека k та кількість зеків що втекли q. Використайте цю функцію, щоб (між іншим) запам'ятати позиції радарів.
- `std::vector<int> localize();` – Ця функція буде викликана q разів, раз для кожного зека. Вона повинна повернути вектор цілих чисел, що позначають індекси радарів (у списку радарів, наведених вище), які створюють випуклий багатокутник, що містить всередині позицію зека (але не містить жодного іншого радара). В багатокутнику радари мають йти за або проти годинникової стрілки. Радари пронумеровані від 1 до n. У реалізації цієї функції ви можете „запитати" систему радарів за допомогою функції `isInside()`, описаної нижче.

Бібліотека надає вам одну функцію:

- bool `isInside(std::vector<int> pol);` – Цю функцію ви можете виконати k разів для кожного зека, тобто всередині одного `localize()`. Ви подаєте як аргумент список індексів радарів, які ви хочете запитати про позицію зека. Радари повинні формувати випуклий багатокутник, обійдений за або проти годинникової стрілки, а функція поверне **true** тоді і тільки тоді, коли зек знаходиться всередині цього багатокутника.

Будь-які невідповідності із специфікацією комунікації призведуть до неправильної відповіді. Ваша програма не повинна читати зі стандартного входу, а тим більше намагатися втручатися в роботу бібліотеки. Це загрожує принаймні неправильною відповіддю.

Бібліотека

Ви можете припустити, що:

- жодні три радари не лежать на одній прямій;
- позиція зека була встановлена заздалегідь (бібліотека не буде адаптуватися до дій вашої програми);

*Про проект ти знаєш тільки те, що він великий, круглий і не є місяцем.

- позиція зека не лежить на одній прямій з будь-якими двома радарами;
- кожний зек є досяжний радарами, тобто гарантовано що відповідь існує;
- всі координати (зеків і радарів) лежать в межах від -10^9 до 10^9.

Оцінювання

Ваше рішення має вписуватися в ліміт k запитів `isInside()` для кожного зека, а також повертати правильний випуклий багатокутник як результат функції `localize()`. Якщо воно зробить це в усіх тестах у даному підзавданні, воно отримає за них повну кількість балів, в протилежному випадку – 0 балів. В усіх тестах виконується $n \geq 3$ та $q \geq 1$. Набір тестів поділяється на наступні підзавдання:

Підзавдання	Обмеження	Пункти
1	$n \leq 6$, $q \leq 50$, $k = 4\,000$	15
2	$n \leq 20$, $q \leq 50$, $k = 4\,000$	17
3	$n \leq 60$, $q \leq 400$, $k = 4\,000$	9
4	$n \leq 300$, $q \leq 1\,000$, $k = 600$	9
5	$n \leq 5\,000$, $q \leq 10$, $k = 10\,000$	10
6	$n \leq 300$, $q \leq 1\,000$, $k = 250$	10
7	$n \leq 1\,000$, $q \leq 1\,000$, $k = 200$	10
8	$n \leq 1\,000$, $q \leq 2\,000$, $k = 60$	10
9	$n \leq 2\,500$, $q \leq 2\,000$, $k = 40$	10

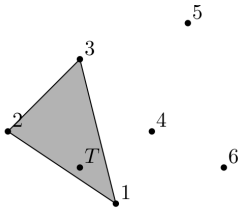
Приклад роботи програми

У описаному нижче тесті є $n = 6$ радарів, $q = 4$ зеків, а ліміт запитів $k = 5$. Рядки таблиці з символом „→” означають, що бібліотека викликає функції вашої програми, тоді як „←” означає, що ваша програма комунікує з бібліотекою. Вся ситуація представлена на малюнках нижче.

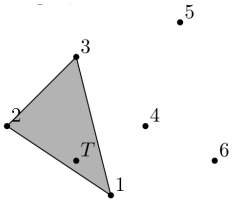
Викликана функція	Результат	Опис
→ `set_radars((3,0), (0,2), (2,4), (4,2), (5,5), (6,1), 10, 4)`	—	Бібліотека подала список радарів та значення q і k.
→ `localize()`	...	Бібліотека викликала `localize()` і чекає, коли програма поверне значення.
← `isInside(1,2,3)`	**true**	В'язень знаходиться всередині трикутника $(1, 2, 3)$.
← *return* `1,2,3;`	—	Програма повертає відповідь $(1, 2, 3)$ як результат `localize()`.
→ `localize()`	...	Другий виклик `localize()` для другого зека.
← `isInside(1,2,3)`	**false**	В'язня немає у трикутнику $(1, 2, 3)$.
← `isInside(1,2,3,4)`	**true**	В'язень знаходиться в чотирикутнику $(1, 2, 3, 4)$.
← *return* `1,3,4;`	—	Програма повертає відповідь $(1, 3, 4)$ для другого зека.
→ `localize()`	...	Третій зек.
← `isInside(2,3,5,4,1)`	**false**	
← `isInside(4,5,6)`	**false**	
← *return* `1,4,6;`	—	
→ `localize()`	...	Четвертий зек.
← `isInside(1,3,5,6)`	**false**	
← `isInside(4,3,5,6)`	**true**	
← *return* `4,3,5,6;`	—	

На малюнках зображено взаємодію програми і бібліотеки, що була описана вище. Кожен рядок позначає відповідного зека, позначеного на малюнку як точка T. Останній багатокутник у кожному рядку є відповіддю програми.

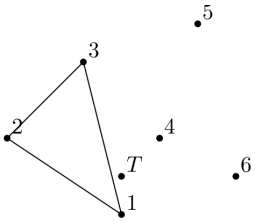
Зек 1, питання 1



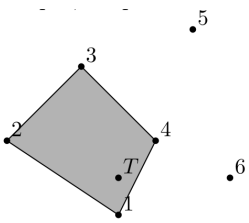
Зек 1, відповідь



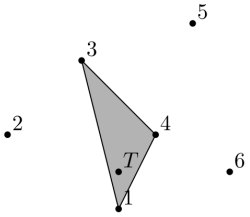
Зек 2, питання 1



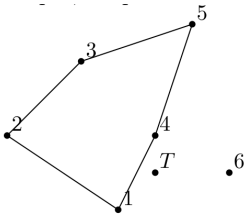
Зек 2, питання 2



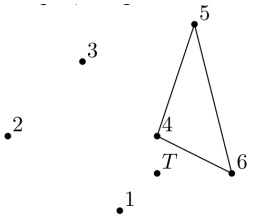
Зек 2, відповідь



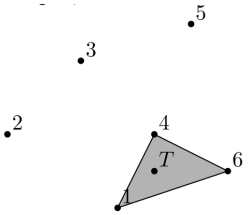
Зек 3, питання 1



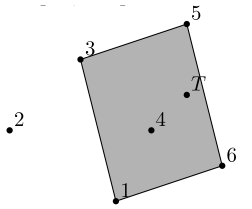
Зек 3, питання 2



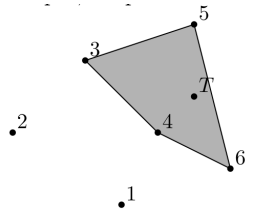
Зек 3, відповідь



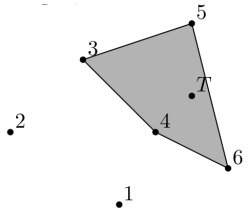
Зек 4, питання 1



Зек 4, питання 2



Зек 4, відповідь



Експерименти

Приклад **неправильного** розв'язку і приклад бібліотеки можна знайти у архіві в розділі *Файли* на SIO$_2$. Поведінка бібліотеки може відрізнятися від тієї, що буде використана для оцінки розв'язків, і не задовольняти умовам задачі. Вона має лише показати спосіб взаємодії з програмою.

Розв'язок `zek.cpp` можна скомпілювати наступним чином:

```
g++ -O3 -static -o zek zek.cpp zeklib.cpp -std=c++17
```

Потрібно забезпечити, щоб файли `zeklib.h` та `zeklib.cpp` знаходилися у тій самій папці, що і розв'язок.

Після компіляції, бібліотека приймає на стандартному вході опис тесту:

n k

x_1 y_1

...

x_n y_n

q

Бібліотека виводимите питання, які задає рішення `zek.cpp` у форматі

? m ind_1, ind_2, ... ind_m.

Ви маєте дати відповідь на питання $isInside(ind_1, ind_2, \ldots, ind_m)$(Yes або No).

Відповідь яку видала програма `zek.cpp` буде виписана у форматі ! m ind_1, ind_2, ... ind_m. Файл вхідних даних для прикладового тесту можна знайти у архіві у файлі `zek0.in`.