Jakub Radoszewski
Treść zadania, Opracowanie

Marcin Pilipczuk
Program

OI, Etap II, dzień pierwszy, 15.02.2006

Najazd

Stało się — Trójkątowie najechali Bajtocję! Bajtocja leży na wyspie i zajmuje całą jej powierzchnię. Wyspa ta ma kształt wielokąta wypukłego (tzn. takiego wielokąta, którego każdy kąt wewnętrzny jest mniejszy od 180°). W Bajtocji znajduje się pewna liczba fabryk oprogramowania. Każda fabryka przynosi pewne stałe zyski lub straty.

Trójkątowie postanowili opanować taki fragment Bajtocji, który:

- *ma kształt trójkąta, którego wierzchołkami są pewne trzy różne wierzchołki wielokąta-wyspy,*
- *przyniesie im jak największe zyski, tzn. suma zysków i strat przynoszonych przez fabryki znajdujące się na opanowanym terytorium będzie możliwie jak największa.*

Przyjmujemy, że jeżeli fabryka leży na brzegu lub w wierzchołku opanowanego terytorium, to należy do niego. Terytorium, które nie zawiera żadnej fabryki przynosi oczywiście zysk równy 0.

Król Bajtocji, Bajtazar, zastanawia się, jak duże straty dla gospodarki kraju może przynieść najazd Trójkątów. Pomóż mu i napisz program, który wyznaczy sumę zysków i strat przynoszonych przez fabryki, które chcą opanować Trójkątowie.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis kształtu Bajtocji i położenie fabryk,*
- *wyznaczy maksymalną sumę zysków i strat przynoszonych przez fabryki znajdujące się w obrębie trójkąta, którego wierzchołkami są pewne trzy różne wierzchołki wielokąta-wyspy.*
- *wypisze wynik na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n ($3 \leqslant n \leqslant 600$), oznaczającą liczbę wierzchołków wielokąta-wyspy. Kolejnych n wierszy wejścia zawiera po dwie liczby całkowite x_j i y_j ($-10\,000 \leqslant x_j, y_j \leqslant 10\,000$), oddzielone pojedynczym odstępem i oznaczające współrzędne x i y kolejnych wierzchołków wyspy, podane w kolejności zgodnej z kierunkiem ruchu wskazówek zegara. Wiersz $n+2$-gi zawiera jedną liczbę całkowitą m ($1 \leqslant m \leqslant 10\,000$), oznaczającą liczbę fabryk. Kolejne m wierszy zawiera po trzy liczby całkowite x'_i, y'_i i w_i ($-10\,000 \leqslant x'_i, y'_i \leqslant 10\,000$, $-100\,000 \leqslant w_i \leqslant 100\,000$), oddzielone pojedynczymi odstępami i oznaczające odpowiednio: współrzędne x i y i-tej fabryki, oraz zysk (dla $w_i \geqslant 0$) bądź stratę (dla $w_i < 0$), którą ta fabryka przynosi. Każda fabryka leży na wyspie-wielokącie, tzn. wewnątrz niej lub na jej brzegu. Kilka fabryk może leżeć w tym samym miejscu, tj. mieć te same współrzędne.

Wyjście

Pierwszy i jedyny wiersz wyjścia powinien zawierać jedną liczbę całkowitą, oznaczającą maksymalną sumę zysków i strat przynoszonych przez fabryki znajdujące się w obrębie trójkąta, którego wierzchołkami są pewne trzy różne wierzchołki wielokąta-wyspy. Może się zdarzyć, że liczba ta będzie ujemna.

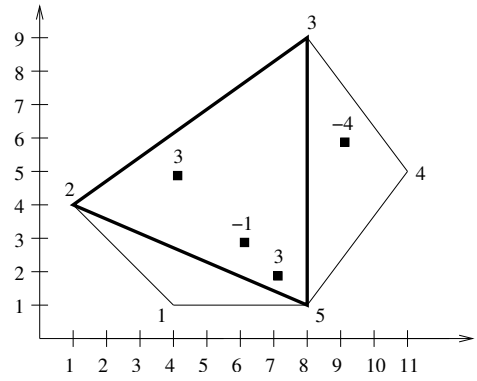
Przykład

Dla danych wejściowych:

```
5
4 1
1 4
8 9
11 5
8 1
4
7 2 3
6 3 -1
4 5 3
9 6 -4
```

poprawnym wynikiem jest:

```
5
```



Rozwiązanie

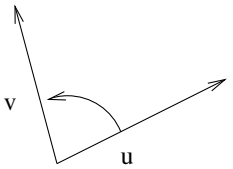
Rozwiązanie siłowe

Zacznijmy od konstrukcji jakiegokolwiek rozwiązania zadania. Najprostszy pomysł, jaki przychodzi na myśl, to przeanalizowanie każdego dopuszczalnego trójkąta (jest ich $O(n^3)$) i zsumowanie zysków/strat wszystkich fabryk w nim zawartych. Dostajemy w ten sposób algorytm, który można nazwać naiwnym, działający w czasie $O(n^3m)$, o ile ... umiemy w czasie stałym sprawdzić, czy dany punkt należy do danego trójkąta.

Jest to pytanie z zakresu geometrii obliczeniowej, czyli dziedziny matematyki i informatyki, która zajmuje się geometrią z algorytmicznego punktu widzenia. Zagadnienia z tej dziedziny są na ogół łatwe do rozwiązania dla człowieka (wystarczy narysować trójkąt oraz punkt i wynik widać „gołym okiem"), ale o wiele trudniejsze do zapisania w postaci algorytmu, gdy trzeba posługiwać się odpowiednimi wzorami matematycznymi. Ponadto często trudno tak zapisać algorytm, żeby działał poprawnie dla wszystkich przypadków szczególnych. Wiąże się to między innymi z ograniczoną dokładnością działań na typach zmiennoprzecinkowych na komputerze. Ich używanie może uniemożliwić na przykład bezbłędne sprawdzenie, czy punkt należy do danego odcinka (w tym celu trzeba porównać dwie liczby rzeczywiste, których reprezentacja w komputerze może być przybliżona), stąd zawsze warto próbować pozostać w dziedzinie liczb całkowitych, na których obliczenia są dokładne.

W geometrii obliczeniowej bardzo istotną rolę odgrywa *iloczyn wektorowy* (oznaczany $\vec{u} \times \vec{v}$). Przypomnijmy jego najważniejsze własności dla wektorów na płaszczyźnie (więcej o tym działaniu można przeczytać w dowolnej książce o geometrii obliczeniowej, np. [34], a także w [18]):

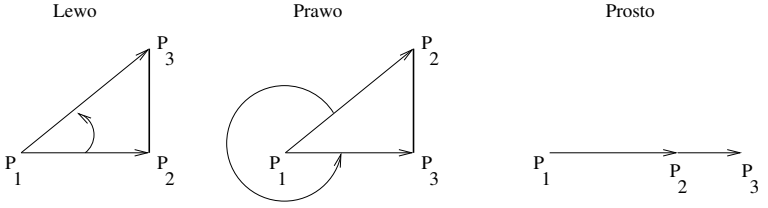
1. Iloczyn wektorowy dwóch wektorów jest także wektorem, którego kierunek jest prostopadły do płaszczyzny, w której zawarte są oba te wektory, zwrot jest wyznaczony z reguły prawej dłoni (śruby prawoskrętnej), a wartość równa jest $|\vec{u} \times \vec{v}| = |\vec{u}| \cdot |\vec{v}| \cdot \sin\alpha$, gdzie α jest kątem skierowanym między wektorami $\vec{u}$ i $\vec{v}$ (zauważmy, że tak zdefiniowana wartość jest liczbą ze znakiem, czyli uwzględnia zwrot — będziemy z tego korzystać w dalszej części opisu).



Rys. 1: Kąt skierowany

2. Z powyższej własności wynika, że wartość bezwzględna wartości iloczynu wektorowego jest równa dwukrotności pola trójkąta, zawartego między wektorami $\vec{u}$ i $\vec{v}$.

3. Na podstawie pierwszej własności możemy sprawdzić, czy przechodząc przez łamaną łączącą kolejno punkty P_1, P_2 i P_3 (będziemy ją oznaczać $P_1 \to P_2 \to P_3$) skręcamy w punkcie P_2 w prawo ($|\vec{P_1P_2} \times \vec{P_1P_3}| < 0$), w lewo ($|\vec{P_1P_2} \times \vec{P_1P_3}| > 0$) czy idziemy prosto ($|\vec{P_1P_2} \times \vec{P_1P_3}| = 0$). W pseudokodach w niniejszym opracowaniu przy badaniu kierunku skrętu będziemy posługiwać się funkcją Skręt(P_1, P_2, P_3), o wartościach ze zbioru $\{Lewo, Prawo, Prosto\}$.



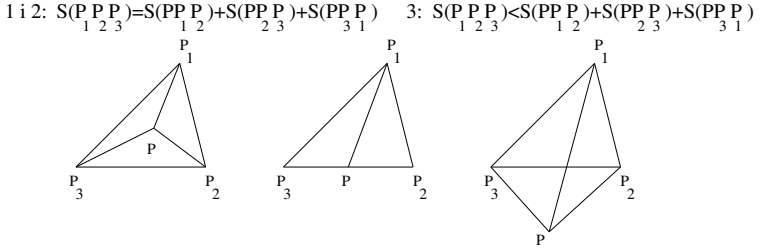
Rys. 2: Sprawdzanie kierunku skrętu

4. Jeżeli $\vec{u} = [x_1, y_1]$, a $\vec{v} = [x_2, y_2]$, to $|\vec{u} \times \vec{v}| = x_1y_2 - x_2y_1$. Jest to bardzo ważna własność, która pozwala łatwo obliczać wartość iloczynu wektorowego i dodatkowo upewnia nas, że liczba ta jest całkowita, o ile tylko współrzędne wektorów są całkowitoliczbowe.

Z użyciem iloczynu wektorowego możemy już łatwo określić, czy punkt P leży wewnątrz trójkąta $\triangle P_1P_2P_3$: wystarczy sprawdzić, czy

$$S(\triangle P_1P_2P_3) = S(\triangle PP_1P_2) + S(\triangle PP_2P_3) + S(\triangle PP_3P_1),$$

gdzie $S(\triangle ABC)$ oznacza pole trójkąta $\triangle ABC$.

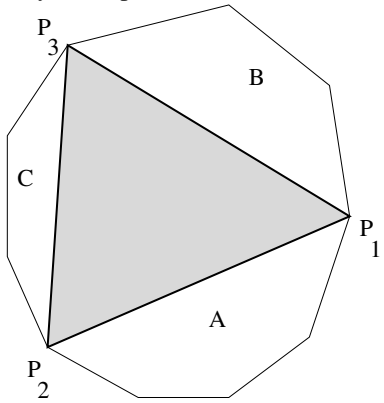


Rys. 3: Sprawdzanie, czy punkt należy do trójkąta

To kończy opis algorytmu naiwnego. Mimo nikłej trudności tego rozwiązania, można było za nie zdobyć w czasie zawodów 30 punktów, gdyż w pełni poprawna implementacja wymagała pewnej wiedzy z zakresu geometrii obliczeniowej.

Szybsze rozwiązanie

Złożoność czasowa powyższego rozwiązania nie jest wystarczająca przy ograniczeniach z zadania. W kolejnym rozwiązaniu najpierw wykonamy obliczenia, które potem umożliwią szybsze znajdowanie sumy zysków/strat fabryk zawartych w danym trójkącie. Najistotniejszym spostrzeżeniem, prowadzącym do tego rozwiązania jest przedstawienie wyniku dla dopuszczalnego trójkąta, jako różnicy sumy wartości wszystkich fabryk i sum wartości fabryk zawartych w trzech półpłaszczyznach otwartych, ograniczonych przez boki tego trójkąta (oznaczonych na rysunku przez A, B i C):



Rys. 4: Półpłaszczyzny A, B i C

Aby zrozumieć, dlaczego waga powyższego spostrzeżenia jest tak duża, zastanówmy się, ile jest różnych takich półpłaszczyzn. Zauważmy, że każda z półpłaszczyzn jest wyznaczona jednoznacznie przez wektor złożony z dwóch wierzchołków wielokąta-wyspy, a zatem różnych półpłaszczyzn jest $O(n^2)$. Dla ustalenia uwagi oznaczmy przez (P_1, P_2) półpłaszczyznę złożoną z punktów P takich, że przechodząc przez $P_1 \to P_2 \to P$ skręcamy w lewo (na przykład na powyższym rysunku $A = (P_1, P_2)$). Teraz możemy sformułować rozwiązanie:

1. Najpierw dla każdej półpłaszczyzny (P_1, P_2) wyznaczamy sumę wartości fabryk w niej zawartych — oznaczmy tę wartość przez $f[P_1, P_2]$. Możemy to uczynić obliczając

dla każdego punktu-fabryki odpowiedni iloczyn wektorowy. Ten krok ma złożoność czasową $O(n^2m)$, właśnie dzięki niewielkiej liczbie różnych półpłaszczyzn.

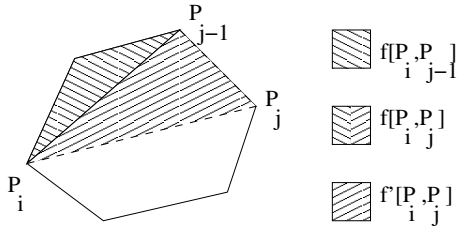
2. Następnie wynik dla trójkąta $\triangle P_1P_2P_3$ wyznaczamy w czasie stałym, jako $S - f[P_1,P_2] - f[P_2,P_3] - f[P_3,P_1]$, gdzie S jest sumą wartości wszystkich fabryk. Złożoność czasowa tego kroku to $O(n^3)$.

Całkowita złożoność czasowa tego rozwiązania to $O(n^2(n+m))$. Pozwalało ono uzyskać 70 punktów, jako że zawierało pewien pomysł natury algorytmicznej, który — jak się za chwilę okaże — ulepszony stanowi już rozwiązanie wzorcowe.

Rozwiązania wzorcowe

Kluczem do rozwiązania wzorcowego jest przyspieszenie pierwszej fazy wyżej opisanego algorytmu. Odtąd będziemy w opisie zakładali, że wierzchołki wielokąta są ponumerowane w kolejności zgodnej z kierunkiem ruchu wskazówek zegara: $P_1,P_2,\ldots,P_n$. Ponadto przyjmujemy, że numeracja wierzchołków jest cykliczna, to znaczy $P_0 = P_n$, $P_{n+1} = P_1$ itd.

Będziemy starali się policzyć wartości $f'[P_i,P_j]$, zdefiniowane jako $f'[P_i,P_j] = f[P_i,P_j] - f[P_i,P_{j-1}]$. Odpowiadają one sumom wartości fabryk, zawartych w jednostronnie otwartych trójkątach (trójkąt o wierzchołkach P_i, P_{j-1} oraz P_j, niezawierający boku $P_i\bar{P}_j$ w dalszej części będziemy oznaczać symbolem $\angle P_iP_{j-1}P_j$).



Rys. 5: Ilustracja f i f'

Na podstawie tych wartości będziemy w stanie w złożoności czasowej $O(n^2)$ policzyć wartości f — dla każdego punktu P_i wyznaczymy wszystkie wartości $f[P_i,*]$ na podstawie równości:

- $f[P_i,P_i] = f[P_i,P_{i+1}] = 0$,
- $f[P_i,P_j] = f[P_i,P_{j-1}] + f'[P_i,P_j]$ dla $j > i+1$,

Zastanówmy się więc jak policzyć wartości f'. Skupimy się na wyznaczeniu wszystkich wartości $f'[P_i,*]$ dla ustalonego punktu P_i. Opiszemy dwie metody wykonania tego podzadania.

Sposób pierwszy

W danym kroku przeanalizujemy wszystkie fabryki i dla każdej z nich stwierdzimy, w którym trójkącie $\angle P_iP_{j-1}P_j$ się znajduje — oczywiście każda fabryka należy do co najwyżej jednego takiego trójkąta. Zauważmy, że fabryka F położona na wyspie nie należy do żadnego

z podanych trójkątów wtedy i tylko wtedy, gdy należy do odcinka $P_{i-1}P_i$. Możemy to sprawdzić obliczając jeden iloczyn wektorowy. W pozostałych przypadkach możemy zidentyfikować odpowiedni trójkąt za pomocą wyszukiwania binarnego, w którym będziemy dla danej przekątnej P_iP_j sprawdzać, w jakim kierunku skręcamy przechodząc przez $P_i \to P_j \to F$.

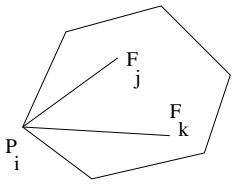
Powyższy krótki opis pozwala już na ułożenie pseudokodu tego rozwiązania:

```
1:  { Wyliczanie f'[P_i, *]. }
2:  for j := 1 to m do
3:  begin
4:    if (Skręt(P_i,P_{i-1},F_j)= Prosto) then
5:      { Fabryka należy do odcinka P_{i-1}P_i }
6:      continue;
7:    { Dla fabryki F_j szukamy a, takiego że F_j ∈ ∠ P_iP_{a-1}P_a. }
8:    a := i+1, b := i+n-1;
9:    while (a ≠ b) do
10:   begin
11:     c := ⌊(a+b)/2⌋;
12:     if (Skręt(P_i,P_c,F_j) ∈ {Prawo, Prosto}) then
13:       a := c+1
14:     else
15:       b := c;
16:   end
17:   f'[P_i,P_a] := f'[P_i,P_a]+wartość(F_j);
18: end
```

Złożoność czasowa powyższego kodu to oczywiście $O(m \log n)$. Wykorzystując tę metodę możemy skonstruować rozwiązanie całego zadania o złożoności $O(nm \log n + n^3)$. Jest to pierwsze z rozwiązań wzorcowych.

Sposób drugi

Zamiast stosować wyszukiwanie binarne, możemy dla danego punktu P_i posortować wszystkie fabryki po kącie nachylenia odcinków, których końcami są punkt P_i i dana fabryka. Dokładniej, powiemy że $F_j < F_k$, jeżeli przechodząc przez $F_j \to P_i \to F_k$ skręcamy w lewo.



Rys. 6: Kryterium sortowania

W tej samej kolejności mamy już posortowane wierzchołki wielokąta — kolejność zgodna z kierunkiem ruchu wskazówek zegara jest właśnie takim porządkiem. Teraz możemy połączyć te dwa posortowane zbiory, czyli wykonać krok podobny do fazy scalania w algorytmie sortowania przez scalanie (ang. Merge Sort). W trakcie tego kroku

kolejne fabryki są przydzielane do pierwszego trójkąta, a kiedy jakaś fabryka przestaje się w nim mieścić, to zaczynamy przydzielanie do drugiego trójkąta itd. Zauważmy, że do sprawdzenia przynależności punktu do trójkąta wystarczy nam tym razem jedno użycie iloczynu wektorowego.

Pseudokod tego rozwiązania jest jeszcze łatwiejszy niż poprzedniego:

```
1:  { Wyliczanie f'[P_i, *]. }
2:  Sortowanie zbioru {F_1, ..., F_m} w porządku kątowym;
3:  j := 1, a := i + 2;
4:  while (a ≠ i) do
5:  begin
6:    while (j ⩽ m and Skręt(P_i,P_a,F_j)= Lewo) do
7:    begin
8:      { Fabryka należy do trójkąta ∠ P_i P_{a-1} P_a]. }
9:      f'[P_i, P_a] := f'[P_i, P_a] + wartość(F_j);
10:     j := j + 1;
11:   end
12:   a := a + 1; { Przypominamy, że numeracja wierzchołków jest cykliczna. }
13: end
```

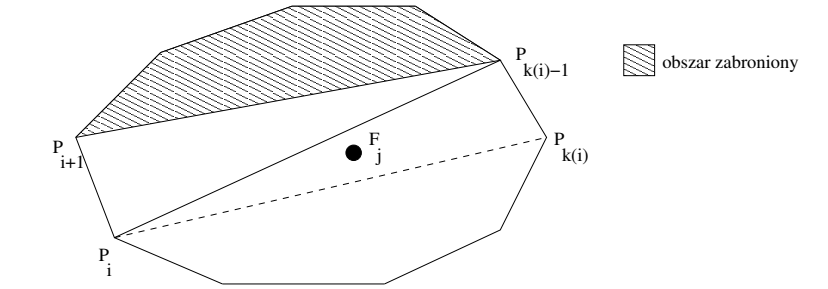
Sortowanie przy zastosowaniu szybkiego algorytmu ma złożoność $O(m \log m)$, faza scalania wymaga czasu liniowego względem sumy liczb fabryk i wierzchołków wielokąta. A zatem złożoność powyższej procedury to $O(n + m \log m)$, a złożoność czasowa całego algorytmu to $O(nm \log m + n^3)$. Otrzymujemy więc metodę nieznacznie gorszą od poprzedniej.

Ponieważ nie sposób odróżnić powyższe rozwiązania w procesie testowania, to oba zostały uznane za wzorcowe. Implementacje obu rozwiązań znajdują się na dysku dołączonym do książeczki.

Rozwiązanie szybsze od wzorcowego

Okazuje się, że istnieje szybsze rozwiązanie zadania — pomysł przedstawili sami zawodnicy olimpiady. Idea rozwiązania polega na szybszym wykonaniu fazy obliczania wartości f (a właściwie f'). Tym razem dla to danej fabryki F_j i wszystkich punktów P_i będziemy wyznaczali punkty $P_{k(i)}$, takie że F_j należy do $\angle P_i P_{k(i)-1} P_{k(i)}$.

Rozpoczynamy od wyznaczenia wartości $k(1)$, na przykład za pomocą najprostszego, liniowego algorytmu, analizując kolejne punkty $P_2, P_3, \ldots$ Następnie zauważamy, że dla kolejnych punktów P_i wartość $k(i)$ będzie tylko rosła (ponownie przypominamy o cyklicznej numeracji punktów). Na rysunku 7 pokazujemy, że rzeczywiście $P_{k(i+1)}$ nie może być wierzchołkiem, leżącym na brzegu wielokąta między wierzchołkami P_i oraz $P_{k(i)-1}$:



Rys. 7: Dowód stwierdzenia

A zatem możemy wyznaczać kolejne wartości $k(i)$ zwiększając o jeden wskaźnik, który na samym końcu tego procesu musi znaleźć się w punkcie $k(1)$, czyli wykonamy co najwyżej n ruchów. W ten sposób wszystkie wartości $k(i)$ dla fabryki F_j znajdujemy w czasie $O(n)$. Obliczenie wszystkich wartości f' ma złożoność równą $O(nm)$, czyli lepszą niż w przypadku rozwiązania wzorcowego, a całe rozwiązanie ma złożoność $O(nm+n^3) = O(n(m+n^2))$.

Na koniec zilustrujemy powyższy opis pseudokodem rozwiązania, który już tradycyjnie jest krótszy od pseudokodu poprzedniego rozwiązania:

```
1:  { Dla danej fabryki F_j wyznaczamy P_k(i) dla i = 1,2,...,n. }
2:  k := 1;
3:  for i := 1 to n do
4:  begin
5:    { W pętli 7-8 zarówno wyliczamy k(1) w złożoności O(n), }
6:    { jak i pozostałe wartości k(i) w łącznej złożoności liniowej. }
7:    while (k ≠ i and Skręt(P_i,P_k,F_j) ≠ Lewo) do
8:      k := k+1; { Przypominamy, że numeracja wierzchołków jest cykliczna. }
9:    if (k ≠ i) then
10:     f'[P_i,P_k] := f'[P_i,P_k]+wartość(F_j);
11: end
```

Testy

Zadanie testowane było na zestawie 10 danych testowych, z których pierwszych 7 było grupami złożonymi z dwóch testów. Większość stanowiły testy losowe, a pozostałe były testami „specyficznymi", które charakteryzowały się tym, że kolejne trójkąty były na przemian zyskowne i stratne, co pomagało eliminować rozwiązania, polegające na znajdowaniu pewnego maksimum lokalnego w zbiorze trójkątów.

Nazwa	n	m	Opis
naj1a.in	64	200	test losowy
naj1b.in	64	200	test specyficzny
naj2a.in	64	200	test losowy

Nazwa	n	m	Opis
naj2b.in	64	200	test specyficzny
naj3a.in	64	200	test losowy
naj3b.in	64	200	test specyficzny
naj4a.in	200	5 000	test losowy
naj4b.in	200	500	test specyficzny
naj5a.in	200	5 000	test losowy
naj5b.in	200	500	test specyficzny
naj6a.in	200	5 000	test losowy
naj6b.in	200	500	test specyficzny
naj7a.in	200	5 000	test losowy
naj7b.in	200	500	test specyficzny
naj8.in	600	10 000	test losowy
naj9.in	600	10 000	test losowy
naj10.in	600	10 000	test losowy