Michał Włodarczyk
Treść zadania, Opracowanie

Adam Gawarkiewicz
Program

Dostępna pamięć: 32 MB.

OI, Etap II, dzień drugi, 11.02.2010

Owce

Mieszkańcy Wyżyny Bajtockiej z dziada-pradziada trudnią się hodowlą owiec. Każdy szanujący się hodowca posiada ogrodzone pastwisko w kształcie wielokąta wypukłego[1]*, na którym wypasa swoje owce. Każda szanująca się owca ma swoje ulubione miejsce na pastwisku, gdzie codziennie skubie trawę. Od czasu do czasu owce lubią się też pobawić. Ponieważ owce bawią się parami, każdy pastuch posiada parzystą liczbę owiec, tak żeby każda owca miała zawsze partnera do zabawy.*

Niepokój pastuchów wzbudziło rozporządzenie wydane przez bajtogrodzkiego komisarza ds. rolnictwa. Głosi ono, że od nowego roku owce można wypasać tylko na pastwiskach w kształcie trójkątów. Każdy hodowca, którego pastwisko jest n-kątem dla $n > 3$, musi je podzielić na trójkąty, stawiając na jego terenie $n - 3$ płotów. Pojedynczy płot musi mieć kształt odcinka łączącego dwa wierzchołki wielokąta stanowiącego całe pastwisko. Płoty nie mogą przecinać się poza wierzchołkami wielokąta. Bez spełnienia warunku komisarza hodowcy nie otrzymają dopłaty do hodowli posiadanych owiec.

Bajtazar, który jest hodowcą owiec, musi zadecydować, jak podzielić swoje pastwisko. Głowi się teraz, ile jest w ogóle sposobów, na jakie można podzielić jego pastwisko tak, aby żaden płot nie przechodził przez ulubione miejsce żadnej owcy i wszystkie owce nadal mogły się bawić, czyli aby w każdym trójkącie znajdowały się ulubione miejsca wypasu parzystej liczby owiec. Pomóż mu i napisz odpowiedni program!

Wejście

Pierwszy wiersz standardowego wejścia zawiera trzy liczby całkowite n, k oraz m ($4 \leqslant n \leqslant 600$, $2 \leqslant k \leqslant 20\,000$, $2 \mid k$, $2 \leqslant m \leqslant 20\,000$) pooddzielane pojedynczymi odstępami, oznaczające odpowiednio: liczbę wierzchołków wielokąta stanowiącego pastwisko, liczbę owiec oraz pewną dodatnią liczbę całkowitą m. W kolejnych n wierszach znajdują się po dwie liczby całkowite x_i i y_i ($-15\,000 \leqslant x_i, y_i \leqslant 15\,000$) oddzielone pojedynczym odstępem, oznaczające współrzędne i-tego wierzchołka pastwiska. Wierzchołki te są podane w kolejności zgodnej z ruchem wskazówek zegara. W kolejnych k wierszach znajdują się po dwie liczby całkowite p_j, q_j ($-15\,000 \leqslant p_j, q_j \leqslant 15\,000$) oddzielone pojedynczym odstępem, oznaczające współrzędne ulubionego miejsca j-tej owcy. Ulubione miejsce każdej owcy znajduje się wewnątrz (tj. nie na brzegu) pastwiska.

Wyjście

Twój program powinien wypisać na standardowe wyjście jedną liczbę całkowitą, oznaczającą resztę z dzielenia przez m liczby takich podziałów pastwiska na trójkąty, aby żaden płot nie przechodził przez ulubione miejsce żadnej owcy i w każdym powstałym trójkącie znajdowały się ulubione miejsca parzystej liczby owiec.

[1] Wielokąt wypukły to taki wielokąt prosty (bez samoprzecięć), w którym każdy kąt wewnętrzny jest mniejszy niż 180°.

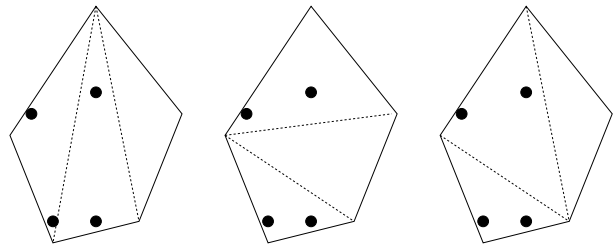
Przykład

Dla danych wejściowych:

```
5 4 10
5 5
3 0
-1 -1
-3 4
1 10
1 0
-1 0
1 6
-2 5
```

poprawnym wynikiem jest:

```
3
```



Na rysunku przedstawiono trzy możliwe sposoby podziału pastwiska na trójkąty. Kropkami zaznaczono ulubione miejsca owiec.

Rozwiązanie

Wprowadzenie

W zadaniu mamy do czynienia z problemem triangulacji wielokąta wypukłego — musimy ustalić liczbę jego podziałów na trójkąty, które spełniają określone warunki. Zanim zaczniemy myśleć o naszym problemie, spróbujmy zastanowić się nad czymś znacznie prostszym: ile jest podziałów pastwiska, na którym nie ma owiec?

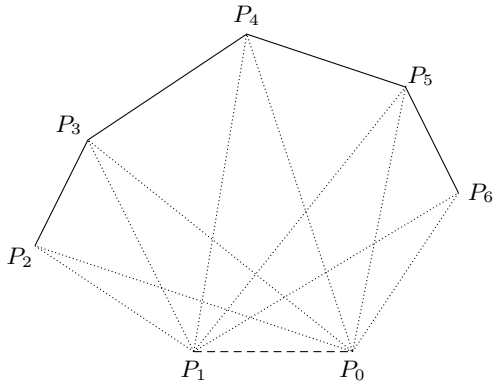
Łatwo zauważyć, że liczba ta nie zależy od kształtu pastwiska, a jedynie od liczby jego boków — wszelkie deformacje pastwiska zachowujące jego wypukłość zachowują także poprawne podziały. Oznaczmy odpowiedź na powyższe pytanie przez A_n, gdzie n to liczba boków wielokąta. Oczywiście $A_n = 1$ dla $n \leqslant 3$ (podział pusty jest w pełni poprawnym podziałem). Dla większych n wyróżniamy jeden bok n-kąta; bok ten musi stanowić bok pewnego trójkąta z podziału. Trzeci wierzchołek takiego trójkąta możemy wybrać na $n-2$ sposobów. Każdy z nich daje podział typu wielokąt–trójkąt–wielokąt, gdzie „wielokąt" może być w szczególności trójkątem lub nawet odcinkiem.

Daje to następujący wzór rekurencyjny:

$$A_n = \sum_{i=2}^{n-1} A_i \cdot A_{n+1-i}.$$

Okazuje się, że $A_n = C_{n-2}$, gdzie C_n to n-ta liczba Catalana[1]. Przy okazji można dostrzec, że przetwarzane w zadaniu liczby mogą rosnąć wykładniczo. Będziemy zatem musieli pamiętać, by w trakcie obliczeń przechowywać tylko reszty z dzielenia

[1]Istnieje wzór jawny na C_n, lecz nie będzie nam potrzebny. Zainteresowanych odsyłamy do [23].



Rys. 1: Możliwe trójkąty dla wyróżnionego boku $[P_0, P_1]$.

modulo m, aby nie wyjść poza zakres typów całkowitych 64-bitowych (jeśli będziemy zawsze wykonywali działania modulo m, to wystarczą nam tak naprawdę typy całkowite 32-bitowe).

Pierwsze rozwiązanie

Sposób, w jaki otrzymaliśmy wzór na A_n, daje się łatwo zaadaptować do potrzeb naszego zadania w pełnej jego ogólności. Tym razem dla pewnego wielokąta zawartego w pastwisku, zamiast sumować po wszystkich możliwych trójkątach o wyróżnionym boku, ograniczamy się do tych, które zawierają parzystą liczbę owiec i których boki nie przecinają owczych pozycji.

Będziemy chcieli oprzeć nasze rozwiązanie o programowanie dynamiczne, więc zastanówmy się, dla których wielokątów musimy znać wynik. Wydawać by się mogło, że dla wszystkich, których wierzchołki są także wierzchołkami pastwiska. Nie byłaby to dobra wiadomość, gdyż jest ich $\Theta(2^n)$. Zauważmy jednak, że wielokąty, których używamy bezpośrednio do obliczenia wyniku dla całego pastwiska, mają tylko jeden bok będący przekątną pastwiska. Co więcej, jeśli przy rozważaniu takich wielokątów ów właśnie bok potraktujemy jako wyróżniony, to dalej wielokąty, które otrzymamy w wyniku podziałów, będą miały tylko jeden bok stanowiący przekątną pastwiska. Wystarczy zatem rozważać tego rodzaju wielokąty. Jest ich oczywiście kwadratowo wiele.

Niech $P_0, P_1, \ldots, P_{n-1}$ oznaczają wierzchołki wyjściowego wielokąta. Oznaczmy przez $S_{x,y}$ wielokąt utworzony przez przecięcie pastwiska prostą przechodzącą przez wierzchołki o numerach x oraz y. Jako że takie wielokąty są zazwyczaj dwa, sprecyzujmy definicję. Jeśli $x < y$, to $S_{x,y}$ składa się z wierzchołków $\{P_x, P_{x+1}, \ldots, P_y\}$. Jeśli zaś $x > y$, to $S_{x,y}$ zawiera wierzchołki $\{P_x, P_{x+1}, \ldots, P_{n-1}, P_0, \ldots, P_y\}$. Przez $T_{x,y}$ oznaczmy liczbę poprawnych triangulacji wielokąta $S_{x,y}$. W poniższym pseudokodzie zaczynamy od obliczenia $T_{x,y}$ dla $y \equiv x+1 \pmod{n}$ oraz dla $y \equiv x+2 \pmod{n}$, tzn. dla boków wielokąta oraz dla trójkątów, a następnie obliczamy dla coraz większych wielokątów. Dla uproszczenia przyjmujemy, że wierzchołki numerowane są cyklicznie,

czyli $P_x = P_{x+n}$, oraz że dotyczy to także odwołań do tablicy, tzn. zapis $T[x][y]$ oznacza tak naprawdę $T[x \bmod n][y \bmod n]$.

```
1:  { obliczenia dla pojedynczych boków i trójkątów }
2:  for x := 0 to n − 1 do begin
3:    T[x][x + 1] := 1;
4:    if trójkąt Δ(P_x, P_{x+1}, P_{x+2}) zawiera parzyście wiele owiec
5:          and odcinek [P_x, P_{x+2}] nie zawiera żadnej owcy then
6:      T[x][x + 2] := 1
7:    else
8:      T[x][x + 2] := 0;
9:  end
10:
11: { główna pętla }
12: for d := 3 to n − 1 do { wielkość rozważanego aktualnie wielokąta to d + 1 }
13:   for x := 0 to n − 1 do
14:     for y := 1 to d − 1 do
15:       if trójkąt Δ(P_x, P_{x+y}, P_{x+d}) zawiera parzyście wiele owiec
16:             and jego boki nie zawierają żadnej owcy then
17:         T[x][x + d] := (T[x][x + d] + T[x][x + y] · T[x + y][x + d]) mod m;
18: return T[0][n − 1];
```

Nie sprecyzowaliśmy jeszcze, jak zliczać owce w trójkątach, lecz można przypuszczać, że wymagać to będzie co najwyżej $O(k)$ operacji na każdy trójkąt. Powyższy algorytm działa więc w czasie $O(n^3k)$ — przeglądamy $O(n^2)$ wielokątów, w każdym wykonujemy $O(nk)$ operacji.

Odrobina geometrii

Należy jeszcze wyjaśnić jedną rzecz: jak rozstrzygnąć, czy owca znajduje się wewnątrz trójkąta?

Najprostszą metodą jest skorzystanie z *iloczynu wektorowego*, który wektorom $\vec{u}, \vec{v}$ zawartym w przestrzeni trójwymiarowej przypisuje prostopadły do nich wektor $\vec{u} \times \vec{v}$ o długości równej podwojonemu polu trójkąta rozpinanego przez $\vec{u}, \vec{v}$. W naszym przypadku, leżące na płaszczyźnie $\vec{u} = \langle u_1, u_2 \rangle, \vec{v} = \langle v_1, v_2 \rangle$ zanurzamy w przestrzeń trójwymiarową i kojarzymy z wektorami $\langle u_1, u_2, 0 \rangle, \langle v_1, v_2, 0 \rangle$. Wtedy iloczyn wektorowy działa następująco:

$$\langle u_1, u_2, 0 \rangle \times \langle v_1, v_2, 0 \rangle = \langle 0, 0, u_1 v_2 - u_2 v_1 \rangle.$$

Interesującym dla nas faktem jest, że wektor $\vec{u} \times \vec{v}$ jest skierowany do góry (tzn. $u_1 v_2 - u_2 v_1 > 0$), jeśli $\vec{v}$ jest „na lewo" od $\vec{u}$. Oczywiście przeciwna nierówność pociąga odwrotne położenie, zaś $u_1 v_2 - u_2 v_1 = 0$ oznacza współliniowość wektorów.

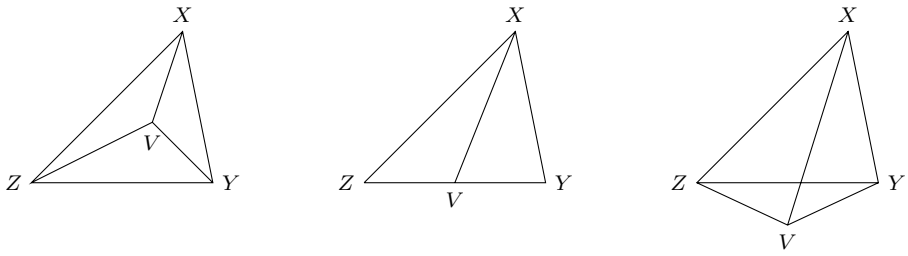
Aby zrozumieć powyższą zależność, połóżmy $\vec{u} = \langle u_1, 0 \rangle$ i niech $u_1 > 0$. Wówczas iloczyn wektorowy $\vec{u} \times \vec{v}$ ma zwrot dodatni wtedy i tylko wtedy, gdy $v_2 > 0$, czyli koniec wektora $\vec{v}$ zaczepionego w 0 znajduje się nad osią X. Ponadto $|\vec{u} \times \vec{v}| = |u_1 v_2|$ to

wysokość trójkąta $(\vec{u}, \vec{v})$ przemnożona przez długość jego podstawy, czyli podwojone pole tego trójkąta. Teraz wystarczy zauważyć, że iloczyn wektorowy nie zmienia się, jeśli oba wektory przekręcimy o dowolny kąt względem 0. Warto sprawdzić to samemu, pamiętając, jak działa obrót o kąt α: $\phi_\alpha(\langle x, y\rangle) = \langle x\cos\alpha - y\sin\alpha,\ x\sin\alpha + y\cos\alpha\rangle$.

Aby zbadać położenie owcy względem trójkąta, możemy zbadać jej położenie względem każdego boku (np. czy jest nie na prawo od żadnego z nich, jeśli będziemy je przeglądać w kierunku przeciwnym do ruchu wskazówek zegara). Można też zauważyć, że punkt V należy do trójkąta $\Delta(X, Y, Z)$, tylko gdy

$$P(X, Y, Z) = P(X, V, Y) + P(Y, V, Z) + P(Z, V, X), \tag{1}$$

gdzie $P(X, Y, Z)$ to pole trójkąta $\Delta(X, Y, Z)$. Pola trójkątów wyznaczamy oczywiście przy pomocy iloczynu wektorowego.



Rys. 2: W dwóch pierwszych przypadkach kryterium (1) da odpowiedź pozytywną, w trzecim zaś nie.

Więcej o iloczynie wektorowym i geometrii obliczeniowej można dowiedzieć się np. z książek [21] oraz [22].

Rozwiązanie wzorcowe

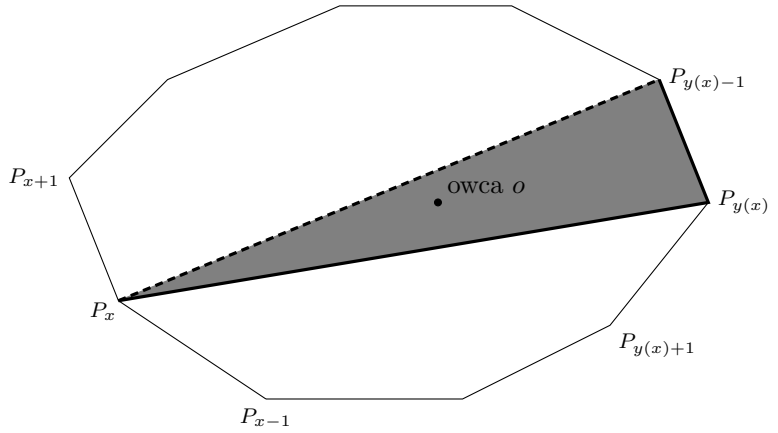
W pierwszym rozwiązaniu widoczne jest miejsce, w którym wiele razy wykonujemy podobne obliczenia. Chodzi o sprawdzanie parzystości liczby owiec w trójkątach i wyszukiwanie „złych" przekątnych. W dalszych rozważaniach trójkąt zawierający parzyście wiele owiec nazwiemy *dobrym* trójkątem. *Dobra* przekątna to zaś taka, która nie zawiera pozycji żadnej owcy.

Zauważmy, że trójkąt $\Delta(P_x, P_y, P_z)$ (gdzie $x < y < z$) jest dobry, gdy jego boki są dobrymi przekątnymi oraz wielokąty $S_{x,y}$, $S_{y,z}$ i $S_{z,x}$ zawierają w sumie parzyście wiele owiec. Istnieją proste sposoby pozwalające w czasie $O(nk \log k)$ znaleźć liczbę owiec we wszystkich wielokątach postaci $S_{x,y}$, polegające na sortowaniu owiec kątowo po kolei względem każdego wierzchołka. Jednak nie będą one dla nas satysfakcjonujące i skorzystamy ze sprytnego rozwiązania działającego w czasie $O(nk)$.

Oznaczmy przez $\Delta_{x,y}$ trójkąt $\Delta(P_x, P_{y-1}, P_y)$, z bokiem $[P_x, P_y]$, ale bez $[P_x, P_{y-1}]$ (przypomnijmy, że o $[P_{y-1}, P_y]$ wiadomo, że nie ma na nim owiec). Dla każdej owcy o i wierzchołka x istnieje dokładnie jeden wierzchołek $y(x)$ taki, że o znajduje się

w $\Delta_{x,y(x)}$. Dla ustalonej owcy obliczymy wartość $y(x)$ dla wszystkich x w łącznym czasie $O(n)$. Kluczowy jest tu następujący fakt, którego formalny dowód pozostawiamy Czytelnikowi.

Fakt 1. *Wierzchołek $P_{y(x+1)}$ nigdy nie leży „na lewo" od wierzchołka $P_{y(x)}$, tzn.* $y(x+1) \in \{y(x), y(x)+1, \ldots, x\}$.



Rys. 3: Ilustracja Faktu 1. Zamalowany trójkąt to $\Delta_{x,y} = \Delta(P_x, P_{y-1}, P_y)$.

Dzięki tej obserwacji wiemy, jak efektywnie zliczać owce w trójkątach. Dla danej owcy o i każdego wierzchołka x znajdujemy wartość $y(x)$ w czasie $O(n)$: najpierw liniowo obliczamy $y(0)$, a dla kolejnych x zwiększamy y (modulo n), aż dojdziemy do $y(x)$. Fakt 1 gwarantuje, że wartość y zmieni się nie więcej niż $2n$ razy.

Dla każdego $\Delta_{x,y}$ wiemy już, ile owiec się w nim znajduje. Możemy teraz w łącznym czasie $O(n^2)$ obliczyć te wartości dla każdego wielokąta $S_{x,y}$ (z brzegiem). W poniższym pseudokodzie spamiętywane są liczby owiec w $\Delta_{x,y}$ i $S_{x,y}$, ale właściwie wystarczy znać tylko ich parzystość. Jednocześnie, przy zliczaniu owiec w trójkątach zostają zapamiętane złe przekątne. Łatwo dostrzec, że każda przekątna zawierająca pozycję pewnej owcy zostanie zauważona. A opracowanie metody sprawdzenia, czy pozycja owcy znajduje się na danej przekątnej, która to metoda jest podobna do tych opisanych w poprzedniej sekcji, pozostawiamy Czytelnikowi.

W pseudokodzie zakładamy, że używane tablice są zainicjowane w następujący sposób: wszystkie komórki tablic ile_owiec_w_trójkącie oraz ile_owiec_w_S są wyzerowane, a wszystkie komórki tablicy dobra_przekątna ustawione na **true**.

```
1: for o := 1 to k do begin
2:   y := 2;
3:   for x := 0 to n − 1 do begin
4:     while owca o nie leży w Δ_{x,y} do
5:       y := (y + 1) mod n;
6:     if owca leży na odcinku [P_x, P_y] then
7:       dobra_przekątna[x][y] := false;
```

```
8:        ile_owiec_w_trójkącie[x][y] := ile_owiec_w_trójkącie[x][y] + 1;
9:     end
10:  end
11:  for x := 0 to n − 1 do
12:     for d := 2 to n − 1 do
13:        ile_owiec_w_S[x][(x + d) mod n] :=
14:              ile_owiec_w_S[x][(x + d − 1) mod n]
15:              +ile_owiec_w_trójkącie[x][(x + d) mod n];
```

Powyższe przetwarzanie niewiele różni się od wstępnych obliczeń w zadaniu *Najazd* z XIII Olimpiady Informatycznej. Warto dodać, że wówczas rozwiązanie wzorcowe używało metod o złożoności czasowej $O(nk \log k)$, a algorytm działający w czasie $O(nk)$ został wymyślony przez zawodników. W książeczce [13] znajduje się wyczerpujące opracowanie zadania *Najazd*, w którym dokładnie opisano także wolniejsze algorytmy zliczania punktów w trójkątach. Sporo miejsca poświęcono również iloczynowi wektorowemu.

Skonstruowane przez nas rozwiązanie działa w czasie $O(nk + n^3)$ (przetwarzanie wstępne oraz przeglądanie n^2 wielokątów) i przechodzi wszystkie testy. Można znaleźć je w plikach `owc.cpp`, `owc0.pas`, ..., `owc4.pas`.

Rozwiązania wolniejsze i błędne

W plikach `owcs1.cpp` oraz `owcs2.pas` zaimplementowano proste rozwiązanie siłowe działające w czasie wykładniczym. W kolejnych parach plików aż do `owcs7.cpp` i `owcs8.pas` można znaleźć kolejne ulepszenia rozwiązania naiwnego aż do algorytmu opisanego na początku opracowania, który zdobywał na zawodach do 60 punktów.

Rozwiązania korzystające z wolniejszego, wymagającego czasu $O(nk \log k)$ zliczania owiec w trójkątach zdobywały co najmniej 80 punktów (często było to ponad 90 punktów, zdarzało się nawet 100). Zaimplementowano je w plikach `owcs[9-14].[cpp|pas]`.

W pliku `owcb1.cpp` zaimplementowano rozwiązanie niesprawdzające, czy dana przekątna jest zła. Otrzymywało ono 10 punktów.

Testy

Wszystkie testy zostały wygenerowane losowo.

Nazwa	n	k	m
owc1.in	14	6	19 387
owc2.in	14	8	13 787
owc3.in	22	12	12 385
owc4.in	50	30	9 987
owc5.in	100	1 000	1 987

Nazwa	n	k	m
owc6.in	200	5 000	12 987
owc7.in	300	5 000	13 581
owc8.in	400	10 000	14 817
owc9.in	500	20 000	19 717
owc10.in	600	20 000	17 679