Piotr Stańczyk
Treść zadania, Opracowanie

Piotr Stańczyk
Program

OI, Etap I

Żaby

W Bajtocji zapanowała klęska żab — niszczą one wszystkie uprawy. Rolnik Bajtazar postanowił walczyć z żabami za pomocą specjalnych odstraszaczy, które rozmieścił w wybranych miejscach na swoim polu. Każda żaba, przemieszczając się z jednego miejsca do drugiego, stara się omijać odstraszacze w jak największej odległości, tzn. maksymalizuje odległość od najbliższego odstraszacza.

Pole Bajtazara ma kształt prostokąta. Żaby skaczą po polu równolegle do boków pola, przemieszczając się w każdym pojedynczym skoku o jedną jednostkę. Odległość od odstraszaczy jest rozumiana jako minimum z odległości od poszczególnych odstraszaczy dla wszystkich pozycji, na których żaba się znajdowała.

Bajtazar wie skąd i dokąd żaby się najczęściej przemieszczają i eksperymentuje z różnymi rozmieszczeniami odstraszaczy. Poprosił Cię o pomoc, chce abyś napisał program obliczający dla zadanego rozstawienia odstraszaczy, w jakiej odległości od odstraszaczy żaby mogą przedostawać się po jego polu z jednego miejsca do drugiego.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia rozmiar pola, rozmieszczenie odstraszaczy oraz pozycję początkową i końcową żaby,*
- *wyznaczy maksymalną odległość od najbliższego odstraszacza, przy jakim żaba będzie musiała przejść na swojej drodze,*
- *wypisze kwadrat znalezionej odległości na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera dwie liczby całkowite: w_x i w_y oddzielone pojedynczym odstępem — szerokość i długość pola ($2 \leqslant w_x, w_y \leqslant 1\,000$). Drugi wiersz wejścia zawiera cztery liczby całkowite: p_x, p_y, k_x i k_y oddzielone pojedynczymi odstępami; (p_x, p_y) to początkowe położenie żaby, (k_x, k_y) to końcowe położenie żaby ($1 \leqslant p_x, k_x \leqslant w_x$, $1 \leqslant p_y, k_y \leqslant w_y$). Trzeci wiersz wejścia zawiera jedną liczbę całkowitą n — liczbę odstraszaczy rozmieszczonych na polu ($1 \leqslant n \leqslant w_x \cdot w_y$). Kolejne n wierszy zawiera współrzędne kolejnych odstraszaczy. Wiersz $i+3$ dla $1 \leqslant i \leqslant n$ zawiera dwie liczby całkowite x_i i y_i oddzielone pojedynczym odstępem — są to współrzędne i-tego odstraszacza ($1 \leqslant x_i \leqslant w_x$, $1 \leqslant y_i \leqslant w_y$). Każdy odstraszacz znajduje się w innym miejscu i żaden z nich nie znajduje się w punkcie (p_x, p_y) ani (k_x, k_y).

Wyjście

W pierwszym i jedynym wierszu wyjścia powinna znaleźć się jedna liczba całkowita, kwadrat maksymalnej odległości, na jaką żaba musi zbliżyć się do najbliższego odstraszacza. Jeżeli żaba nie może uniknąć wskoczenia bezpośrednio na odstraszacz, to wynikiem jest 0.

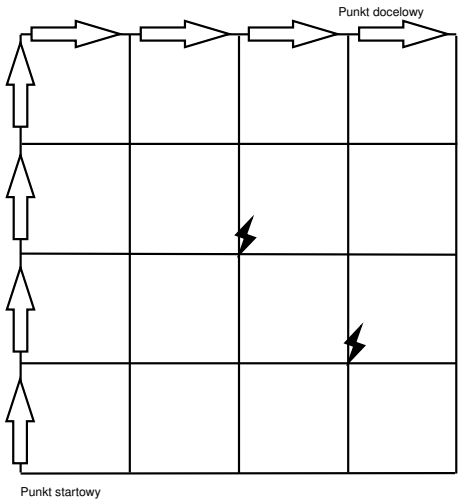
Przykład

Dla danych wejściowych:

```
5 5
1 1 5 5
2
3 3
4 2
```

poprawnym wynikiem jest:

```
4
```



Optymalna droga żaby.

Rozwiązanie

Wprowadzenie

Rozpocznijmy od wprowadzenia terminologii, którą będziemy stosować w opisie rozwiązania. *Pozycją* nazwiemy dowolne miejsce na polu Bajtazara, na którym może znaleźć się żaba (na polu o wymiarach w_x na w_y jest dokładnie $w_x \cdot w_y$ pozycji). Poprzez *odległość danej pozycji od odstraszaczy* rozumieć będziemy minimum z odległości tej pozycji od wszystkich odstraszaczy.

Rozwiązanie zadania podzielimy na dwie niezależne części:

- wyznaczenie dla każdej pozycji odległości od odstraszaczy (faza I),
- wyznaczenie drogi żaby maksymalizującej odległość od odstraszaczy na podstawie odległości pozycji od odstraszaczy (faza II).

Powstaje pytanie, czy taki podział pracy nie spowoduje, że obliczenia będą trwały dłużej niż to konieczne. Jeżeli uda nam się skonstruować algorytm działający w czasie zbliżonym do $w_x \cdot w_y$, to na pewno nie będzie to algorytm zły. Wynika to stąd, że już same dane dla zadania mogą mieć rozmiar $w_x \cdot w_y$ (Bajtazar mógł rozstawić aż tyle odstraszaczy), a każdy — nawet najlepszy — algorytm wymaga przynajmniej przeczytania tych danych.

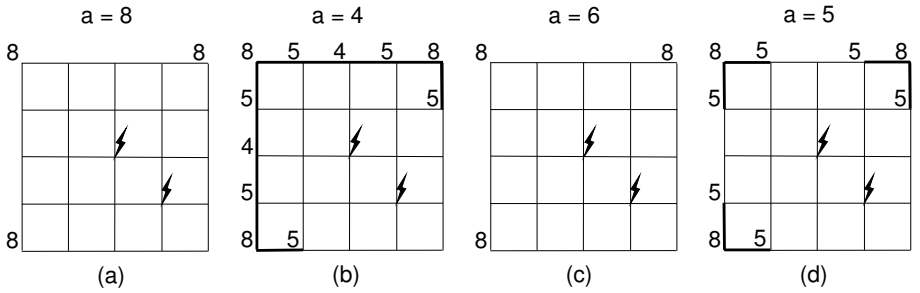
W kolejnych rozdziałach przedstawimy możliwe sposoby realizacji obu faz. Rozpoczniemy dość nietypowo — od fazy drugiej.

Faza II

Ścieżką będziemy nazywali sekwencję sąsiednich pozycji, pomiędzy którymi żaba może bezpośrednio przeskakiwać, z których pierwsza jest pozycją początkową żaby, natomiast ostatnia — pozycją końcową. Przez *odległość ścieżki od odstraszaczy* będziemy rozumieli najmniejszą odległość pozycji należącej do ścieżki od odstraszaczy. Zakładając, że dla każdej pozycji jest już policzona odległość od odstraszaczy, należy wyznaczyć ścieżkę, której odległość od odstraszaczy jest możliwie największa — kwadrat tej odległości, oznaczmy go s_{max}, jest poszukiwanym wynikiem zadania.

Wartość s_{max} jest liczbą całkowitą niemniejszą od 0 (wtedy najlepsza ścieżka musi przejść przez pozycję zawierającą odstraszacz) oraz nie większą od $w_x^2 + w_y^2$ (Bajtazar postawił na polu co najmniej jeden odstraszacz). Na pytanie, czy wartość s_{max} jest niemniejsza od zadanej wartości a można udzielić odpowiedzi w czasie $O(w_x \cdot w_y)$. Jeśli bowiem taka ścieżka istnieje, to musi ona składać się z pól, których odległość od odstraszaczy jest niemniejsza od a. Aby ją znaleźć (lub stwierdzić, że nie istnieje), skonstruujmy graf G_a o zbiorze wierzchołków odpowiadającym pozycjom o odległościach od odstraszaczy niemniejszych niż a. Krawędziami w grafie G_a połączymy takie pozycje, pomiędzy którymi żaba może bezpośrednio przeskoczyć. Ścieżka o odległości co najmniej a od odstraszaczy istnieje wtedy i tylko wtedy, gdy wierzchołki reprezentujące pozycję początkową oraz końcową żaby są połączone ścieżką w grafie G_a (lub inaczej — należą do tej samej składowej spójności). Sprawdzenia tego można dokonać przy użyciu prostego algorytmu przeszukiwania grafu wszerz bądź w głąb. Przeszukiwanie grafu o $O(w_x \cdot w_y)$ wierzchołkach i krawędziach wykonujemy właśnie w takim czasie.

Zapewne Czytelnik spotkał się z grą, której celem jest odgadnięcie liczby naturalnej x wymyślonej przez przeciwnika, przy użyciu pytań postaci „czy x jest mniejsze od a?" (wśród informatyków grę tę nazywa się zazwyczaj *wyszukiwaniem binarnym*). Wykorzystując opisany powyżej algorytm, można wyznaczyć poszukiwaną wartość s_{max} w analogiczny sposób. Początkowo wiemy, że wartość ta znajduje się w przedziale $[0, w_x \cdot w_y]$. Następnie, sprawdzając jak ma się s_{max} do środka przedziału, zawężamy dalsze poszukiwania do przedziału o połowę mniejszego. W ten sposób, zadając logarytmiczną liczbę pytań, można zrealizować drugą fazę algorytmu w czasie $O(w_x \cdot w_y \cdot \log(w_x \cdot w_y))$. Przykład procesu wyszukiwania został przedstawiony na rysunku 1.



Rys. 1: Wyszukiwanie binarne wartości s_{max} dla przykładowych danych z treści zadania (kwadraty odległości poszczególnych pozycji zostały umieszczone przy pozycjach, o ile są niemniejsze od rozpatrywanego w kolejnym kroku ograniczenia a). **(a)** Maksymalna odległość dwóch pól wynosi 16, zatem wyszukiwanie binarne wykonywane jest na przedziale $[0\ldots16]$. Dla ograniczenia 8, pozycja początkowa (lewa-dolna) oraz końcowa (prawa-górna) nie są połączone — rozwiązanie musi znajdować się w takim razie w przedziale $[0\ldots7]$ **(b)** Dla kolejnej testowanej wartości $a=4$, pozycja początkowa i końcowa są połączone, więc rozwiązanie musi mieścić się w przedziale $[4\ldots7]$. **(c)** Dla $a=6$ pozycje znowu nie są połączone — rozwiązanie znajduje się w przedziale $[4\ldots5]$. **(d)** Dla $a=5$ po raz kolejny pozycje nie są połączone, zatem wynikiem końcowym jest 4.

Inne możliwe rozwiązania

Istnieją również inne sposoby realizacji drugiej fazy algorytmu. Dla przykładu przedstawimy dwie z nich — obie sprowadzają analizowany problem do zagadnienia grafowego.

Pierwszym ze sposobów — dość często implementowanym przez uczestników XIII Olimpiady Informatycznej — było przedstawienie problemu w postaci grafu G, którego wierzchołkami są pozycje i krawędzie łączą te pozycje, pomiędzy którymi żaba może bezpośrednio przeskoczyć. Po skonstruowaniu grafu G, do wyznaczenia poszukiwanej ścieżki można wykorzystać modyfikację algorytmu Dijkstry (patrz [18]). W oryginalnej wersji tego algorytmu mamy graf, którego krawędziom są przypisane długości, i wyliczamy długości najkrótszych ścieżek do wszystkich wierzchołków grafu z zadanego źródła (długość ścieżki rozumiana jest jako suma długości krawędzi na niej leżących). W naszym problemie wagi są przypisane wierzchołkom, a nie krawędziom, natomiast poszukiwanym wynikiem jest ścieżka między wierzchołkiem początkowym a docelowym, dla której najmniejsza waga wierzchołka do niej należącego jest maksymalna. Zadanie możemy rozwiązać modyfikując algorytm Dijkstry tak, by obliczać dla każdego wierzchołka v najmniejszą wagę w wierzchołka na ścieżce prowadzącej z wierzchołka początkowego do v. Wówczas wierzchołki musimy odwiedzać w kolejności nierosnących wartości w. Implementacja tego algorytmu (przy użyciu kopca jako kolejki priorytetowej dla wierzchołków) ma złożoność proporcjonalną $O(w_x \cdot w_y \cdot \log(w_x \cdot w_y))$.

Drugi sposób jest modyfikacją algorytmu Kruskala wyznaczania najlżejszych drzew rozpinających w grafie (patrz [18]). W oryginalnym algorytmie rozpoczynamy działanie od grafu zawierającego jedynie wierzchołki, a następnie przeglądamy krawędzie w kolejności niemalejących wag, dodając do grafu te, które łączą różne spójne składowe grafu. Kończymy w momencie, gdy graf zawiera dokładnie jedną spójną składową. Algorytm możemy zmodyfikować tak, by budować graf stopniowo dodając nie krawędzie, lecz wierzchołki

(wraz ze wszystkimi incydentnymi do nich krawędziami) w kolejności nierosnących odległości od odstraszaczy. Ponadto obliczenia kończymy w chwili, gdy pozycja początkowa i końcowa żaby znajdą się w tej samej składowej. Kwadrat odległości od odstraszaczy ostatniego dodanego do grafu wierzchołka jest wówczas poszukiwaną wartością s_{max}. Czas działania tego algorytmu (ze względu na konieczność posortowania wierzchołków) to ponownie $O(w_x \cdot w_y \cdot \log(w_x \cdot w_y))$.

Faza I

Prostym i nasuwającym się od razu sposobem wyznaczenia wyniku dla pierwszej fazy jest obliczenie dla każdej pozycji odległości od wszystkich odstraszaczy, a następnie wybranie najmniejszej z tych wartości. Podejście takie jest bardzo proste w implementacji, jednak wyjątkowo nieefektywne — przy maksymalnej możliwej liczbie odstraszaczy jego złożoność czasowa to $O(w_x^2 \cdot w_y^2)$. Wielu zawodników zauważyło, że taki algorytm nie jest wystarczający i starało się wymyślić efektywniejszą metodę. My także, mając na względzie fakt, iż druga faza algorytmu została zrealizowana w czasie $O(w_x \cdot w_y \cdot \log(w_x \cdot w_y))$, skupimy się na poszukiwaniu rozwiązania o niegorszej złożoności.

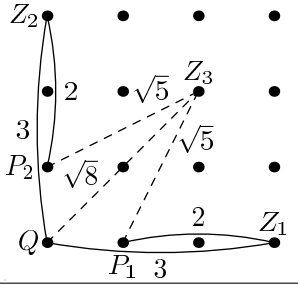
Rozwiązania błędne

Jednym z możliwych pomysłów jest, podobnie jak to zostało zrobione w drugiej fazie, potraktowanie pola Bajtazara jako grafu G, w którym wierzchołki oznaczają pozycje na polu, a krawędzie — skoki żaby. Na takim grafie możemy wykonać algorytm przeszukiwania wszerz, rozpoczynając od wszystkich pozycji odstraszaczy jednocześnie. Pozycjom tym przypisujemy odległość zero, a następnie przeglądając kolejne wierzchołki przypisujemy im obliczone minimalne odległości od odstraszaczy. Algorytm taki ma złożoność czasową $O(w_x \cdot w_y)$ i jest poprawny w przypadku stosowania metryki miejskiej do wyznaczania odległości między pozycjami, jednak w przypadku metryki Euklidesowej, z jaką mamy do czynienia w zadaniu, podejście takie nie jest poprawne (w metryce miejskiej definiujemy odległość między pozycjami a i b jako $|a_x - b_x| + |a_y - b_y|$, a w metryce Euklidesowej, jako $(a_x - b_x)^2 + (a_y - b_y)^2$).

Kolejnym podejściem może być zastąpienie przeszukiwania wszerz algorytmem Dijkstry. Dla każdej pozycji obliczymy i zapiszemy odległość od najbliższego odstraszacza oraz pozycję najbliższego odstraszacza (pozycja źródłowa).

- Na początku dla każdej pozycji P zawierającej odstraszacz zapisujemy odległość równą 0 i pozycję źródłową równą P; pozostałym pozycjom przypisujemy wstępnie odległości $+\infty$.
- Wszystkie wierzchołki z wyznaczoną odległością 0 wstawiamy do kolejki priorytetowej,
- Przetwarzamy pozycje z kolejki, w kolejności niemalejących odległości. Dla pozycji P, której źródłem jest pozycja Z, odwiedzamy każdego sąsiada S pozycji P (czyli pozycje, na które żaba może bezpośrednio przeskoczyć z P). Jeśli odległość pozycji S od Z jest mniejsza niż dotychczasowa odległość S od jej źródła, to przypisujemy S nowe źródło Z i odpowiednio zmieniamy odległość.

Powyższy algorytm został zaprogramowany przez wielu zawodników. Nie jest on jednak poprawny, co można łatwo wykazać. Niech $Z_1(4,1)$, $Z_2(1,4)$, $Z_3(3,3)$ będą pozycjami odstraszaczy na polu Bajtazara. Przedstawiony algorytm dla pozycji $P_1(2,1)$ wyznaczy źródło Z_1 i odległość 2, dla pozycji $P_2(1,2)$ źródłem będzie Z_2, a odległością 2. Dla obu pól P_1 i P_2 odległość do odstraszacza Z_3 wynosi $\sqrt{5}$ i jest większa od 2. Pozycja Q jest sąsiadem P_1 i P_2, zatem źródło dla pozycji Q zostanie ustalone na Z_1 lub na Z_2, a co za tym idzie odległość od odstraszaczy będzie ustalona na 3. Tymczasem odległość Q od Z_3 jest mniejsza i wynosi $\sqrt{8}$.



Rys. 2: Przykładowa sytuacja, dla której zmodyfikowany algorytm Dijkstry daje złą odpowiedź

Dziwna funkcja g

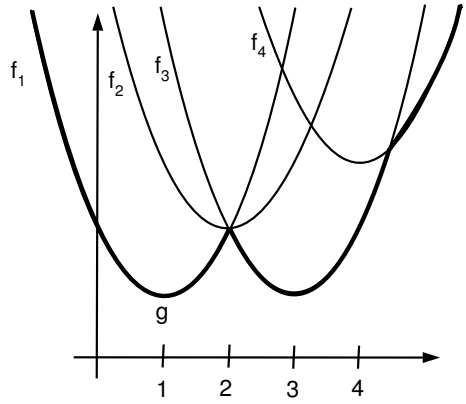
Wszystkie powyższe kłopoty wynikały z nietypowości metryki, która jest zastosowana w zadaniu. Okazuje się jednak, że istnieje stosunkowo prosty (implementacyjnie, acz nie koniecznie koncepcyjnie) sposób wyznaczenia wyniku dla fazy pierwszej w złożoności czasowej $O(w_x \cdot w_y)$.

Rozpatrzmy trochę inny problem od tego, który musimy rozwiązać. Niech dana będzie funkcja $b : \{1,2,\ldots,N\} \to R$. Naszym zadaniem jest wyznaczenie funkcji $g : \{1,2,\ldots,N\} \to R$, zadanej wzorem:

$$g(x) = \min_{n \in \{1,2,\ldots,N\}} ((x-n)^2 + b(n))$$

Przykładowo, dla $N = 4$, $b(1) = 1$, $b(2) = 2$, $b(3) = 1$ i $b(4) = 3$ wartości poszukiwanej funkcji g są następujące: $g(1) = 1$, $g(2) = 2$, $g(3) = 1$, $g(4) = 2$

Po rozszerzeniu dziedziny funkcji g do zbioru liczb rzeczywistych, wyznaczanie wartości $g(x)$ można traktować jako znajdowanie najmniejszej wartości rodziny funkcji kwadratowych. Wykres funkcji g powstaje z fragmentów paraboli rozpatrywanych funkcji kwadratowych. Wykresy dwóch funkcji kwadratowych $f_1(x) = (x-a_1)^2 + b_1$ oraz $f_2(x) = (x-a_2)^2 + b_2$, $a_1 < a_2$ mają dokładnie jeden punkt przecięcia $c_{(1,2)} = (c_x, c_y)$ (łatwo go wyznaczyć z równania $(x-a_1)^2 + b_1 = (x-a_2)^2 + b_2$). Funkcja f_1 ma mniejsze wartości od funkcji f_2 dla $x < c_x$, natomiast dla $x > c_x$ sytuacja jest odwrotna. Zatem wykres funkcji g zdefiniowanej jako minimum dwóch funkcji f_1 i f_2 składa się z fragmentu paraboli funkcji f_1 położonego na lewo od punktu $c_{1,2}$ (czyli $g(x) = f_1(x)$ dla $x \leqslant c_x$) oraz z fragmentu paraboli f_2 położonego na prawo od tego punktu (czyli $g(x) = f_2(x)$ dla $x \geqslant c_x$). Łatwo zauważyć, że funkcja g wyznaczona przez k funkcji f_i ($1 \leqslant i \leqslant k$) składa się z co najwyżej k fragmentów paraboli (niektóre parabole mogą wcale nie wchodzić w skład wykresu g). Przykład funkcji g wyznaczonej przez rodzinę czterech funkcji kwadratowych jest przedstawiony na rysunku 3).



Rys. 3: Postać poszukiwanej funkcji g wyznaczonej przez rodzinę funkcji f_1, f_2, f_3 i f_4

Załóżmy, że mamy wyznaczoną funkcję $g = min(f_1, f_2, \ldots, f_k)$, gdzie $f_1(x) = (x-a_1)^2 + b_1$, $f_2(x) = (x-a_2)^2 + b_2$, ..., $f_k(x) = (x-a_k)^2 + b_k$, $a_1 < a_2 < \ldots < a_k$. Funkcję g możemy przedstawić jako ciąg funkcji ją wyznaczających, przeplatanych punktami przecięcia sąsiednich funkcji — reprezentację tę oznaczymy $R(g)$:

$$R(g) = [f_{w_1}, c_{(w_1,w_2)}, f_{w_2}, c_{(w_2,w_3)}, \ldots, c_{(w_{l-1},w_l)}, f_{w_l}]$$

Przykładowo, reprezentacja funkcji g z rysunku 3 wygląda następująco:

$$R(g) = [f_1, (2,2), f_3, (4.5, 3.25), f_4]$$

Tak zapisaną funkcję g możemy łatwo modyfikować dodając kolejne funkcje z rodziny f. Aby dodać funkcję $f_{l+1}(x) = (x - a_{l+1})^2 + b_{l+1}$, gdzie $a_l < a_{l+1}$ (dla uproszczenia oznaczeń załóżmy przez chwilę, że $R(g) = [f_1, c_{1,2}, f_2, \ldots, f_{l-1}, c_{l-1,l}, f_l]$), wystarczy usuwać z reprezentacji $R(g)$ od końca kolejne funkcje $f_i = f_l, f_{l-1}, \ldots$ tak długo, dopóki punkt przecięcia usuwanej funkcji f_i z funkcją f_{l+1} znajduje się na lewo od punktu przecięcia f_i z funkcją f_{i-1}. Następnie należy dodać na końcu reprezentacji punkt przecięcia f_i z f_{l+1} oraz funkcję f_{l+1}.

Wyznaczenie reprezentacji funkcji g zdefiniowanej przez k funkcji z rodziny f zajmuje czas $O(k)$, gdyż każda funkcja z rodziny f jest raz wstawiana do reprezentacji i co najwyżej raz z niej usuwana (oczywiście przy założeniu, że funkcje z rodziny f mamy uporządkowane według wartości a_i). Także wyznaczenie wartości funkcji g dla kolejnych argumentów $1, 2, \ldots, k$ może zostać w prosty sposób wykonane w czasie $O(k)$ przy użyciu jej reprezentacji $R(g)$.

Poprawne rozwiązanie

Rozpatrzmy teraz pole Bajtazara, którego wymiary wynoszą $w_x \times 1$. Zauważmy, że zdefiniowaną niżej funkcję f_i możemy interpretować, jako kwadrat odległości od odstraszacza stojącego na pozycji $(i,1)$ (jeśli tam jest jakiś odstraszacz) dla wszystkich pozycji $(x,1)$, gdzie $1 \leqslant x \leqslant w_x$,

$$f_i(x) = (x-1)^2 + o_i,$$

o ile zdefiniujemy $o_i = 0$, jeśli na pozycji $(i,1)$ stoi odstraszacz, oraz $o_i = \infty$ w przeciwnym wypadku.
Kwadrat odległości pozycji (x,1) od najbliższego odstraszacza, to z kolei wartość funkcji:

$$g(x) = min(f_1(x), f_2(x), \ldots, f_{w_x}(x)).$$

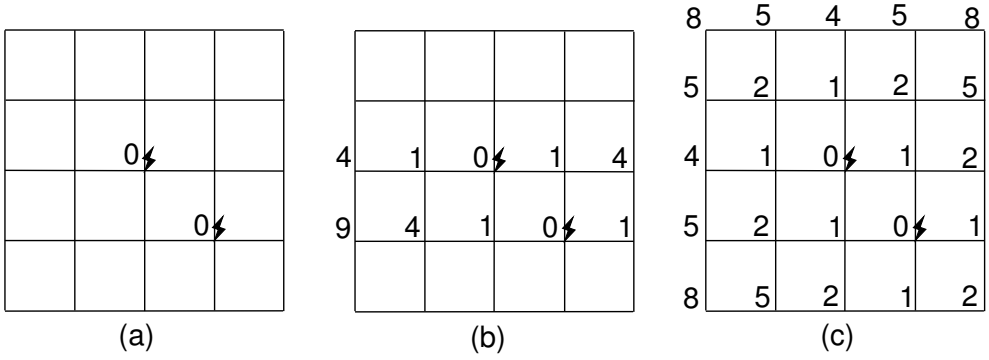
Z poprzedniego rozdziału wiemy, że wartości te dla argumentów $x = 1,2,\ldots,w_x$ potrafimy wyznaczyć w czasie $O(w_x)$.

Obliczenie wyniku w przypadku pola dwuwymiarowego nie wymaga już dużo pracy. Wystarczy najpierw dla każdego wiersza pola Bajtazara wyznaczyć powyższą metodą odległości od odstraszaczy znajdujących się w tym wierszu. W następnym kroku powtórzymy takie same obliczenia dla kolumn, wychodząc od wartości wyliczonych w poprzednim kroku. Jeśli dla pewnej kolumny wartości wyliczone dla pozycji w niej występujących podczas pierwszego przebiegu będą równe $c_1, c_2, \ldots, c_{w_y}$, to rodzinę funkcji f definiujemy jako $f_i = (x-i)^2 + c_i$ (dla $1 \leqslant i \leqslant w_y$) i dalej:

$$g(x) = min(f_1(x), f_2(x), \ldots, f_{w_y})$$

Na rysunku 4 zaprezentowane jest działanie algorytmu na przykładzie z treści zadania. Wyznaczenie odległości wszystkich pozycji na polu Bajtazara od odstraszaczy poprzez wyznaczenie funkcji g jest realizowane w łącznym czasie $O(w_x \cdot w_y)$.

Powyższa metoda z pewnością wymaga dowodu poprawności. Weźmy zatem dowolną pozycję (p_x, p_y), dla której najbliższy odstraszacz jest umieszczony na pozycji (o_x, o_y). W pierwszym kroku algorytmu, przetwarzając wiersze pola, wyznaczymy dla pozycji (p_x, o_y) wartość $(o_x - p_x)^2$ (gdyby wyznaczona wartość była mniejsza, to oznaczałoby to istnienie bliższego pozycji (p_x, o_y) odstraszacza w wierszu o_y, ale taki odstraszacz byłby bliższy pozycji (p_x, p_y) niż odstraszacz (o_x, o_y)). W drugim kroku, przetwarzając kolumny, dla pozycji (p_x, p_y) wyznaczymy (na podstawie wartości $(o_x - p_x)^2$ dla pozycji (p_x, o_y)) wartość $(o_x - p_x)^2 + (o_y - p_y)^2$, czyli faktycznie kwadrat odległości między pozycją (p_x, p_y) a najbliższym odstraszaczem (znów mniejszy wynik oznaczałby istnienie bliższego niż (o_x, o_y) odstraszacza).



Rys. 4: Proces wyznaczania odległości pól od odstraszaczy. **(a)** Początkowe wartości (nieskończoności nie zostały zaznaczone). **(b)** Wyznaczenie odległości od odstraszaczy w wierszach. **(c)** Wyznaczenie odległości od odstraszaczy w kolumnach

Testy

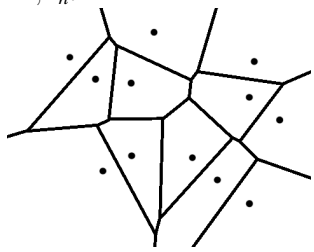
Zadanie było testowane na zestawie 20 danych testowych. Połowa testów to testy poprawnościowe, mające na celu zweryfikowanie poprawności zastosowanych przez zawodników algorytmów. Reszta testów to testy poprawnościowo-wydajnościowe. Poniższa tabela zawiera listę testów wraz z ich podstawową charakterystyką — wymiarami pola w_x i w_y oraz liczbą odstraszaczy n:

Nazwa	w_x	w_y	n	Opis
zab1.in	4	4	3	test poprawnościowy
zab2.in	10	10	6	test poprawnościowy
zab3.in	10	10	15	test poprawnościowy
zab4.in	50	10	20	test poprawnościowy
zab5.in	11	50	15	test poprawnościowy
zab6.in	50	50	160	test poprawnościowy
zab7.in	50	50	199	test poprawnościowy
zab8.in	50	39	334	test poprawnościowy
zab9.in	50	50	350	test poprawnościowy
zab10.in	31	32	213	test poprawnościowy
zab11.in	400	400	120	test wydajnościowo-poprawnościowy
zab12.in	1000	200	100	test wydajnościowy
zab13.in	200	1000	50	test wydajnościowy
zab14.in	1000	400	20000	test wydajnościowy
zab15.in	800	200	200	test wydajnościowy
zab16.in	1000	1000	23	test wydajnościowo-poprawnościowy
zab17.in	999	999	10016	test wydajnościowo-poprawnościowy
zab18.in	999	999	76151	test wydajnościowo-poprawnościowy
zab19.in	1000	1000	50000	test wydajnościowo-poprawnościowy
zab20.in	1000	1000	200003	test wydajnościowo-poprawnościowy

Ciekawostka

Gdyby rozpatrywać poruszanie się żaby w zadaniu w sposób ciągły (żaba nie wykonuje skoków, lecz przesuwa się płynnie pomiędzy punktem startowym a docelowym), to zadanie można by rozwiązać w złożoności $O(n \cdot \log n)$, gdzie n jest liczbą odstraszaczy. Rozwiązanie

takie jest możliwe dzięki zastosowaniu tzw. *Diagramów Voronoia* (można przeczytać o nich więcej w książce [33] lub [34]). Diagram taki można wyznaczyć dla dowolnego zbioru n punktów na płaszczyźnie $P_1, \ldots, P_n$ i stanowi on podział płaszczyzny na n rozłącznych obszarów o kształcie wielokątów wypukłych. Obszar zawierający punkt P_i składa się ze wszystkich takich punktów płaszczyzny, których odległość od P_i jest niewiększa od ich odległości od $P_1, \ldots, P_{i-1}, P_{i+1}, \ldots, P_n$.



Rys. 5: Przykładowy *Diagram Voronoia* wyznaczony dla zbioru 12 punktów.

Wyznaczenie *Diagramu Voronoia* jest możliwe w czasie $O(n \cdot \log n)$, jednak implementacja algorytmu jest bardzo skomplikowana. Po wyznaczeniu diagramu można potraktować go jako graf, w którym wierzchołkami są punkty przecięcia odcinków diagramu, natomiast krawędziami grafu — odcinki diagramu. Każdej krawędzi należy przydzielić wagę równą odległości odpowiadającego odcinka od najbliższego odstraszacza (jest to zadanie proste, gdyż najbliższym punktem każdego odcinka jest ten punkt, którego obszar odcinek wyznacza), a następnie postępować podobnie, jak w drugiej fazie przedstawionego wcześniej algorytmu — zastosować algorytm Dijkstry do wyznaczenia końcowego wyniku. Jako początkowe (końcowe) wierzchołki działania algorytmu można przyjąć na przykład wszystkie wierzchołki wielokąta diagramu, w którym zawarta jest pozycja początkowa (końcowa) ruchu żabki; uprzednio należy ustawić wagi przypisane tym wierzchołkom, wynikające z konieczności przejścia do nich z pozycji początkowej (końcowej). Diagram Voronoia ma co najwyżej $2n-5$ wierzchołków i $3n-6$ krawędzi, zatem cały algorytm (wliczając czas konstrukcji diagramu oraz wykonania algorytmu Dijkstry) działa w czasie $O(n \cdot \log n)$.

Niniejszy skrótowy opis stanowi tylko zarys algorytmu; po dowody przedstawionych w nim faktów i algorytm konstrukcji diagramu Voronoi odsyłamy do wymienionej powyżej literatury.

Zawody II stopnia

opracowania zadań