Marcin Kubica
Treść zadania, Opracowanie

Marek Cygan
Program

OI, Etap II, dzień drugi, 16.02.2006

Orka

Rolnik Bajtazar chce zaorać pole w kształcie prostokąta. Bajtazar może zacząć od zaorania jednej skiby z dowolnego boku pola, potem może zaorać jedną skibę z dowolnego boku niezaoranej części pola itd., aż całe pole będzie zaorane. Po zaoraniu każdej kolejnej skiby, niezaorana część pola ma kształt prostokąta. Skiby mają szerokość 1, a długość i szerokość pola wyrażają się liczbami całkowitymi m i n.

Niestety Bajtazar do orki ma tylko jedną słabowitą szkapę. Gdy szkapa zacznie orać skibę, to nie zatrzymuje się, aż zaorze ją do końca. Bajtazar musi uważać, jeżeli zaoranie skiby będzie dla szkapy zbyt wielkim wysiłkiem, to szkapa padnie. Po zaoraniu każdej kolejnej skiby szkapa może odpocząć i nabrać sił. Nie wszystkie miejsca na polu są tak samo trudne do zaorania. Bajtazar dokładnie zna swoje pole i dokładnie wie jak trudno się orze w każdym miejscu.

Podzielmy pole na $m \times n$ kwadratów jednostkowych. Kwadraty będziemy identyfikować za pomocą ich współrzędnych (i,j), dla $1 \leqslant i \leqslant m$ i $1 \leqslant j \leqslant n$. Każdemu z kwadratów jest przypisany jego współczynnik trudności orki — nieujemna liczba całkowita. Współczynnik trudności orki kwadratu o współrzędnych (i,j) będziemy oznaczać przez $t_{i,j}$. Dla każdej skiby suma współczynników trudności orki kwadratów tworzących skibę nie może przekroczyć pewnej ustalonej stałej k — w przeciwnym przypadku szkapa padnie.

Bajtazar stoi przed trudnym zadaniem. Przed zaoraniem każdej skiby musi zdecydować z którego boku niezaoranej części pola ją zaorać, tak żeby szkapa nie padła. Z drugiej strony, chciałby żeby było jak najmniej skib.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia liczby k, m i n, oraz współczynniki trudności orki,*
- *wyznaczy, w jaki sposób Bajtazar powinien zaorać pole,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu standardowego wejścia znajdują się trzy dodatnie liczby całkowite: k, m i n, pooddzielane pojedynczymi odstępami, $1 \leqslant k \leqslant 200\,000\,000$, $1 \leqslant m \leqslant 2\,000$, $1 \leqslant n \leqslant 2\,000$. W kolejnych n wierszach znajdują się współczynniki trudności orki. Wiersz $j+1$ zawiera współczynniki $t_{1,j}, t_{2,j}, \ldots, t_{m,j}$, pooddzielane pojedynczymi odstępami, $0 \leqslant t_{i,j} \leqslant 100\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście jedną liczbę całkowitą: minimalną liczbę skib powstałych po zaoraniu pola zgodnie z podanymi warunkami. Możesz założyć, że dla danych wejściowych, pole zawsze da się zaorać zgodnie z warunkami podanymi w zadaniu.

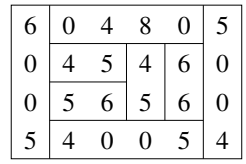
Przykład

Dla danych wejściowych:

```
12 6 4
6 0 4 8 0 5
0 4 5 4 6 0
0 5 6 5 6 0
5 4 0 0 5 4
```

poprawnym wynikiem jest:

```
8
```



Powyższy rysunek przedstawia przykładowy sposób zaorania pola.

Rozwiązanie

Rozwiązanie wykładnicze

Jako pierwsze przedstawimy rozwiązanie proste, lecz nieefektywne. Zauważmy, że po zaoraniu każdej kolejnej skiby pole ma kształt prostokąta złożonego z całych kwadratów jednostkowych. Zaoranie każdej skiby to zmniejszenie tego prostokąta przez usunięcie skrajnej kolumny lub wiersza. Widać więc, że problem ma charakter rekurencyjny: zaoranie całego pola sprowadza się do właściwego wyboru pierwszej skiby i zaorania pozostałego pola w optymalny sposób.

Niemniej jednak, rozwiązywanie zadania poprzez zwykłe rekurencyjne przeszukiwanie z nawrotami (ang. *back-tracking*) możliwych sekwencji skib byłoby błędem. Złożoność czasowa takiego algorytmu jest wykładnicza — wystarczy rzut oka na ograniczenia wielkości danych, żeby przekonać się, że takie rozwiązanie nie mogłoby działać w sensownym czasie.

Rozwiązanie o złożoności czasowej $O(n^2 \cdot m^2)$

Proste spostrzeżenie pozwala znacznie ulepszyć poprzedni algorytm. Wystarczy zauważyć, że wiele różnych początkowych sekwencji skib prowadzi do takich samych pozostałych niezaoranych fragmentów pola i jest tylko $O(n^2 \cdot m^2)$ możliwych różnych niezaoranych prostokątów. Możemy więc rozwiązać zadanie stosując programowanie dynamiczne: dla każdego prostokąta zawartego w polu obliczamy minimalną liczbę skib potrzebnych do jego zaorania. Prostokąty przeglądamy w kolejności od mniejszych do większych i dla każdego rozważamy wszystkie skrajne skiby. Wybieramy taką, którą szkapa może zaorać i po zaoraniu której pozostaje prostokąt wymagający zaorania najmniejszej liczby skib.

Rozwiązanie problemu uzyskujemy po wyznaczeniu wyniku dla największego prostokąta, czyli dla całego pola.

Opiszmy przedstawiony algorytm bardziej formalnie. Przyjmijmy najpierw, że dla $i < 1$, $i > m$, $j < 1$ lub $j > n$ mamy $t_{i,j} = 0$. Oznaczmy przez $S[i,j,i',j']$ (dla $1 \leqslant i \leqslant m$, $1 \leqslant j \leqslant n$) minimalną liczbę skib potrzebnych do zaorania prostokąta, który w dolnym lewym rogu ma kwadrat o współrzędnych (i,j), a w górnym prawym rogu kwadrat o współrzędnych (i',j'). Szukana przez nas wartość to $S[1,1,m,n]$. Liczba skib potrzebna do zaorania danego prostokąta jest określona następująco. Jeśli $i > i'$ lub $j > j'$, to przyjmujemy, że $S[i,j,i',j'] = 0$, natomiast w przeciwnym przypadku:

$$S[i,j,i'j'] = 1 + \min \begin{cases} S[i+1,j,i',j'] & \text{jeśli} \quad t_{i,j} + t_{i,j+1} + \cdots + t_{i,j'} \leqslant k \\ S[i,j+1,i',j'] & \text{jeśli} \quad t_{i,j} + t_{i+1,j} + \cdots + t_{i',j} \leqslant k \\ S[i,j,i'-1,j'] & \text{jeśli} \quad t_{i',j} + t_{i',j+1} + \cdots + t_{i',j'} \leqslant k \\ S[i,j,i',j'-1] & \text{jeśli} \quad t_{i,j'} + t_{i+1,j'} + \cdots + t_{i',j'} \leqslant k \end{cases}$$

Duże znaczenie dla efektywności tego algorytmu ma sposób liczenia sum współczynników trudności dla określonych skib prostokąta. Dla uproszenia sumę taką będziemy nazywać *wagą* skiby. Jeżeli będziemy liczyć wagi wprost, sumując współczynniki trudności kolejnych kwadratów jednostkowych, to obliczenie pojedynczej wartości $S[i,j,i'j']$ będzie wymagać czasu rzędu $O(m+n)$. W ten sposób otrzymamy algorytm o łącznej złożoności czasowej rzędu $O(m^2 \cdot n^2 \cdot (m+n))$ i złożoności pamięciowej rzędu $O(m^2 \cdot n^2)$.

Wagi możemy jednak obliczać znacznie efektywniej. Wystarczy, że dla każdej pary współrzędnych $0 \leqslant i \leqslant m$, $0 \leqslant j \leqslant n$ wyznaczymy wartości:

$$W[i,j] = t_{1,j} + t_{2,j} + \cdots + t_{i,j}$$

$$K[i,j] = t_{i,1} + t_{i,2} + \cdots + t_{i,j},$$

co można zrobić w czasie rzędu $O(n \cdot m)$, przeglądając współczynniki trudności kwadratów jednostkowych w wierszach oraz w kolumnach, jak w poniższej procedurze.

```
1:  var W,K : array [0..m,0..n] of integer;
2:  begin
3:    for i = 0 to m do begin
4:      W[i,0] := 0;
5:      K[i,0] := 0
6:    end;
7:    for j = 0 to n do begin
8:      W[0,j] := 0;
9:      K[0,j] := 0
10:   end;
11:   for i = 1 to m do
12:     for j = 1 to n do begin
13:       W[i,j] := W[i-1,j] + t_{i,j};
14:       K[i,j] := K[i,j-1] + t_{i,j}
15:     end
16: end
```

Wyliczone wartości W i K pozwalają nam w czasie stałym wyznaczyć wagę każdej skiby i do obliczenia tablicy S możemy zastosować następujący wzór:

$$S[i,j,i'j'] = 1 + \min \begin{cases} S[i+1,j,i',j'] & \text{jeśli} \quad K[i,j'] - K[i,j-1] \leqslant k \\ S[i,j+1,i',j'] & \text{jeśli} \quad W[i',j] - W[i-1,j] \leqslant k \\ S[i,j,i'-1,j'] & \text{jeśli} \quad K[i',j'] - K[i',j-1] \leqslant k \\ S[i,j,i',j'-1] & \text{jeśli} \quad W[i',j'] - W[i-1,j'] \leqslant k \end{cases}$$

Wykorzystanie tego wzoru w metodzie programowania dynamicznego daje w efekcie algorytm, którego pseudokod wygląda następująco:

```
1:  var S: array [1..m,1..n] of integer;
2:  begin
3:    for i = m downto 1 do
4:      for i' = 1 to m do
5:        for j = n downto 1 do
6:          for j' = 1 to n do begin
7:            if i > i' or j > j' then S[i,j,i',j'] := 0
8:            else begin
9:              S[i,j,i',j'] := ∞;
10:             if K[i,j'] - K[i,j-1] ⩽ k then
11:               S[i,j,i',j'] := min(S[i,j,i',j'], S[i+1,j,i',j'] + 1);
12:             if W[i',j] - W[i-1,j] ⩽ k then
13:               S[i,j,i',j'] := min(S[i,j,i',j'], S[i,j+1,i',j'] + 1);
14:             if K[i',j'] - K[i',j-1] ⩽ k then
15:               S[i,j,i',j'] := min(S[i,j,i',j'], S[i,j,i'-1,j'] + 1);
16:             if W[i',j'] - W[i-1,j'] ⩽ k then
17:               S[i,j,i',j'] := min(S[i,j,i',j'], S[i,j,i',j'-1] + 1);
18:           end
19:         end
20: end
```

W rezultacie możemy rozwiązać zadanie w czasie $O(m^2 \cdot n^2)$, wykorzystując pamięć tego samego rzędu. Powyższy algorytm został zaimplementowany w programach `orkb1.cpp` i `orkb2.pas`. Taką samą złożoność możemy uzyskać udoskonalając przeszukiwanie z nawrotami poprzez spamiętywanie wyników podproblemów.

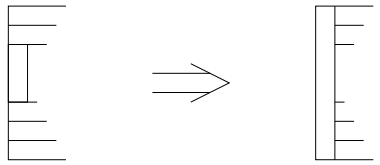
Rozwiązanie o złożoności czasowej $O((n+m)^3)$

Zadanie można rozwiązać jeszcze lepiej. W tym celu musimy dokładniej przeanalizować możliwe układy skib. Zauważmy, że w dowolnym zaoraniu pola mamy n skib poziomych lub m skib pionowych. Wynika to stąd, że każde odcięcie skiby zmniejsza długość jednej pary boków pola o 1, a ostatnia skiba musi mieć przynajmniej jedną parę boków długości 1. Jeśli są to boki pionowe (czyli skiba jest pozioma), to trzeba zaorać n skib poziomych, by doprowadzić do tej sytuacji — takie zaoranie nazwiemy *poziomym*. Analogicznie, jeśli ostatnia skiba jest pionowa, to również zaoranie będziemy nazywać *pionowym*. Gdy ostatnia skiba ma wymiary 1×1, zaoranie możemy nazwać zarówno poziomym, jak i pionowym. Co

więcej, jeżeli pole w ogóle da się zaorać (a w zadaniu rozważamy tylko takie pola), to istnieje dla niego zarówno zaoranie poziome, jak i pionowe. Wynika to stąd, że zaoranie poziome można przekształcić w pionowe dzieląc ostatnią skibę na skiby pionowe o wymiarach 1×1. Analogicznie można uzupełnić zaoranie pionowe do poziomego.

Możemy więc poszukać oddzielnie optymalnego zaorania poziomego oraz optymalnego zaorania pionowego i wybrać ten z wyników, który zawiera mniej skib. W dalszej części skupimy się na poszukiwaniu optymalnego zaorania pionowego, czyli zawierającego m skib pionowych i minimalną liczbę skib poziomych. Dla wygody skibę, której waga nie przekracza k, a więc szkapa może ją zaorać, nazwiemy *dopuszczalną*. Dodatkowo pisząc o boku pola (lewym, prawym, górnym lub dolnym), będziemy mieli na myśli skrajną skibę przylegającą do tego boku prostokąta.

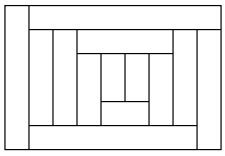
Fakt 1 *Jeżeli lewy bok pola jest skibą dopuszczalną, to istnieje optymalne zaoranie pola zaczynające się od tej skiby. Analogiczne stwierdzenie jest prawdziwe dla prawego boku pola.*



Rys. 1: Zaoranie skrajnie lewej skiby w rozwiązaniu optymalnym.

Dowód W rozwiązaniu optymalnym mamy m skib pionowych, więc istnieje wśród nich skiba pionowa zawarta w lewym boku pola. Skoro cała skrajnie lewa skiba pola jest dopuszczalna, to możemy od niej rozpocząć oranie pola. Dalszą część orki prowadzimy jak poprzednio, z tym że część skib poziomych będzie krótsza o jeden kwadrat. Ilustruje to rys. 1. Łączna liczba skib nie zwiększy się. ■

Korzystając z powyższej własności rozwiązania optymalnego możemy część skib wybierać w sposób zachłanny. Jeżeli lewy lub prawy bok pola jest skibą dopuszczalną, to orkę rozpoczynamy odpowiednio od lewego lub prawego boku. Co jednak należy zrobić, jeżeli nie możemy zaorać (nie zamęczając szkapy) ani lewego, ani prawego boku pola, ale zarówno górny, jak i dolny bok pola są skibami dopuszczalnymi? Który z nich wybrać?



Rys. 2: Rozłączne obszary górnych i dolnych skib poziomych.

W zaoraniu pionowym, skiby poziome tworzą dwa rozłączne obszary: jeden złożony z górnych, a drugi z dolnych skib — ilustruje to rys. 2. Rozważmy na chwilę decyzyjną wersję problemu z zadania: dla danych liczb g i d interesuje nas, czy można zaorać pole

pionowo tak, żeby było co najwyżej g górnych skib poziomych i co najwyżej d dolnych skib poziomych. Następujący fakt pozwala znaleźć odpowiedź na tak postawione pytanie metodą zachłanną.

Fakt 2 *Jeżeli istnieje pionowe zaoranie pola złożone z g górnych skib poziomych i d dolnych skib poziomych, $g > 0$ oraz górny bok pola jest skibą dopuszczalną, to istnieje zaoranie pionowe spełniające podane ograniczenia, zaczynające się od zaorania górnej skiby. Analogiczne stwierdzenie dla dolnej skiby zachodzi, gdy $d > 0$ oraz dolny bok pola jest skibą dopuszczalną.*

Dowód Rozważmy zaoranie spełniające założenia pierwszej części faktu. Ponieważ $g > 0$, więc istnieje przynajmniej jedna skiba pozioma górna, czyli istnieje też skiba pozioma zawarta w górnym boku prostokąta. Skoro bok ten jest skibą dopuszczalną, to możemy zmodyfikować rozważane zaoranie wybierając jako pierwszą skibę górny bok. Resztę pola możemy zaorać jak poprzednio, więc liczba skib poziomych górnych i dolnych pozostanie niezmieniona, a jedynie niektóre skiby pionowe będą krótsze.

Analogiczną modyfikację można przeprowadzić w drugim przypadku, gdy $d > 0$ i dolny bok jest skibą dopuszczalną.

■

Korzystając z wykazanych własności możemy zaproponować następujący algorytm sprawdzania, czy istnieje zaoranie spełniające zadane ograniczenia g i d. Kolejne skiby do zaorania będziemy wybierać zgodnie z następującymi zasadami:

- jeżeli lewy lub prawy bok pola jest skibą dopuszczalną, to wybieramy tę skibę, a w przeciwnym przypadku
- jeżeli dolny bok pola jest skibą dopuszczalną i nie przekroczyliśmy jeszcze limitu d, to wybieramy tę skibę, w przeciwnym przypadku
- jeżeli górny bok pola jest skibą dopuszczalną i nie przekroczyliśmy limitu g, to wybieramy dolny bok, w pozostałym przypadku
- widzimy, że nie da się zaorać pola przy danych ograniczeniach g i d.

Żeby móc szybko stwierdzić, czy określony bok pola jest skibą dopuszczalną, możemy posłużyć się opisanymi wcześniej tablicami K i W. Wówczas sprawdzenie, czy istnieje sposób zaorania pola przy zadanych ograniczeniach g i d, przedstawione w poniższej procedurze *test*, wymaga czasu rzędu $O(n+m)$.

```
1: function test(g,d : integer): boolean;
2: var i,i',j,j' : integer;
3: begin
4:   i := 1; j := 1; i' := m; j' := n;
5:   while i ⩽ i' and j ⩽ j' do begin
6:     { lewa skiba }
7:     if K[i,j'] - K[i,j-1] ⩽ k then i := i+1 else
8:     { prawa skiba }
9:     if K[i',j'] - K[i',j-1] ⩽ k then i' := i'-1 else
```

```
10:       { dolna skiba }
11:       if d > 0 and W[i', j'] − W[i − 1, j'] ⩽ k then begin
12:         j' := j' − 1; d := d − 1
13:       { górna skiba }
14:       end else if g > 0 and W[i', j] − W[i − 1, j] ⩽ k then begin
15:         j := j + 1; g := g − 1
16:       end else begin
17:         { porażka }
18:         test := false; exit
19:       end
20:     end;
21:     test := true
22: end
```

Przypomnijmy, że naszym celem jest znalezienie takich wartości g i d, dla których zaoranie pola jest możliwe, a ich suma jest minimalna. Najprościej jest sprawdzić wszystkie możliwe pary tych liczb. Biorąc pod uwagę, że jest ich $O(n^2)$, obliczenia zajmą czas $O(n^2(n+m))$. Dodatkowo musimy jeszcze znaleźć optymalne zaoranie poziome, czyli uzyskujemy rozwiązanie o złożoności czasowej $O((m+n)^3)$. Algorytm ten został zaimplementowany w programach `orks3.cpp` i `orks4.pas`.

Rozwiązanie wzorcowe o złożoności czasowej $O((n+m)^2)$

Poszukując optymalnego zaorania pionowego nie musimy badać wszystkich par g i d! Oznaczmy przez $f(g)$ najmniejszą wartość d, dla której jest możliwe zaoranie pola przy ustalonym ograniczeniu g na liczbę skib poziomych górnych:

$$f(g) = \min\{d : \mathit{test}(g,d)\}$$

Wówczas szukany wynik to:

$$m + \min_{g \in \{1,\ldots,n-1\}} (g + f(g))$$

Zauważmy, że funkcja f jest nierosnąca. Pozwala nam to policzyć powyższy wynik zadając $O(n+m)$ zapytań o wartość $\mathit{test}(x,y)$.

```
1: var wyn, g, d : integer;
2: begin
3:   wyn := ∞; g := 0; d := n − 1;
4:   while g < n and d ⩾ 0 do
5:     if test(g, d) then begin
6:       wyn := min(wyn, m + g + d);
7:       d := d − 1
8:     end else g := g + 1;
9:   return wyn
10: end
```

W podanym algorytmie każdy ze wskaźników d i g zostanie zmieniony co najwyżej n razy, więc otrzymaliśmy rozwiązanie zadania o złożoności czasowej $O((n+m)^2)$ i pamięciowej $O(n \cdot m)$.

Testy

Testy użyte do oceny rozwiązań były podzielone na 10 grup. Grupy 1–5 zawierają po jednym teście. Grupy 6–10 zawierają po dwa testy: test wydajnościowy przedstawiający pole, na którym znajduje się trudny do zaorania teren w kształcie litery „H” (lub obróconej litery „H”), oraz test losowy. Ów trudny do zaorania teren w kształcie litery „H”, wraz z odpowiednio dobraną wytrzymałością szkapy, wymagał dużej liczby skib. Wielkości testów przedstawiono w poniższej tabelce:

Nazwa	m	n	k	**Opis**
ork1.in	9	10	163	prosty test poprawnościowy
ork2.in	36	32	8 723	prosty test poprawnościowy
ork3.in	50	50	2 547	prosty test poprawnościowy
ork4.in	450	400	2 100 678	test losowy
ork5.in	614	590	3 672 684	test losowy
ork6a.in	1 300	1 305	200 000	test wydajnościowy „H”
ork6b.in	700	700	35 531 618	test losowy
ork7a.in	1 220	1 190	290 000	test wydajnościowy „H”
ork7b.in	1 300	1 250	64 048 256	test losowy
ork8a.in	1 611	1 613	100 000	test wydajnościowy „H”
ork8b.in	1 599	1 601	72 709 332	test losowy
ork9a.in	2 000	2 000	270 000	test wydajnościowy „H”
ork9b.in	2 000	2 000	100 771 608	test losowy
ork10a.in	1 998	2 000	250 000	test wydajnościowy „H”
ork10b.in	1 999	2 000	100 328 366	test losowy

Zawody III stopnia

opracowania zadań