Marcin Kubica
Treść zadania, Opracowanie

Jakub Pawlewicz
Program

OI, Etap III, dzień pierwszy, 02.04.2008

Kupno gruntu

Bajtazar planuje kupno działki przemysłowej. Jego majątek jest wyceniany na k bajtalarów. Tyle też zamierza on przeznaczyć na zakup gruntu. Jednak znalezienie działki, która kosztuje dokładnie k bajtalarów, jest kłopotliwe. W związku z tym, Bajtazar jest gotów kupić ewentualnie droższą działkę. Dodatkowe fundusze może uzyskać przez zaciągnięcie kredytu. Maksymalny rozmiar kredytu, jaki może mu udzielić Bajtocki Bank Kredytowy, wynosi tyle, ile majątek Bajtazara, czyli k bajtalarów. Innymi słowy, Bajtazar chciałby przeznaczyć na kupno działki kwotę w wysokości od k bajtalarów do 2k bajtalarów włącznie.

Teren, na którym Bajtazar zamierza kupić działkę, ma kształt kwadratu o boku długości n metrów. Aktualni właściciele ziemi wyznaczyli różne ceny w przeliczeniu na metr kwadratowy. Bajtazar przeprowadził dokładny wywiad i sporządził mapę cenową tego terenu. Mapa ta opisuje cenę każdego kwadratu o rozmiarze metr na metr. Takich kwadratów jest dokładnie n^2. Teraz pozostaje wyznaczyć wymarzoną działkę. Musi ona mieć kształt prostokąta, złożonego wyłącznie z całych kwadratów jednostkowych. Bajtazar zaczął szukać na mapie odpowiedniej działki, ale mimo wzmożonych wysiłków nie był w stanie znaleźć właściwego prostokąta. Pomóż Bajtazarowi.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia liczby k i n oraz mapę cenową terenu,*
- *wyznaczy działkę o cenie z przedziału $[k, 2k]$ lub stwierdzi, że taka działka nie istnieje,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby całkowite k i n oddzielone pojedynczym odstępem, $1 \leqslant k \leqslant 1\,000\,000\,000$, $1 \leqslant n \leqslant 2\,000$. Każdy z następnych n wierszy zawiera n liczb całkowitych nieujemnych, pooddzielanych pojedynczymi odstępami. Liczba i-ta w wierszu numer $j+1$ określa cenę jednego kwadratu metr na metr, którego współrzędne w terenie to (i, j). Cena jednego metra nie przekracza 2 000 000 000 bajtalarów.

Wyjście

Jeżeli nie istnieje działka o cenie z przedziału $[k, 2k]$, to program powinien wypisać jeden wiersz zawierający słowo `NIE`*. W przeciwnym przypadku powinien wypisać jeden wiersz zawierający cztery liczby całkowite dodatnie x_1, y_1, x_2, y_2 pooddzielane pojedynczymi odstępami, określające współrzędne prostokąta. Para (x_1, y_1) oznacza lewy górny róg prostokąta, a para (x_2, y_2) — prawy dolny róg prostokąta. Wtedy taki prostokąt określony jest przez zbiór*

współrzędnych kwadratów: $\{(x,y) \mid x_1 \leqslant x \leqslant x_2 \text{ i } y_1 \leqslant y \leqslant y_2\}$. *Suma cen kwadratów c leżących wewnątrz wskazanego prostokąta powinna spełniać nierówności* $k \leqslant c \leqslant 2k$. *Jeżeli jest wiele działek spełniających wymagany warunek, to należy wypisać dowolną z nich.*

Przykład

Dla danych wejściowych:

```
4 3
1 1 1
1 9 1
1 1 1
```

poprawnym wynikiem jest:

```
NIE
```

a dla danych wejściowych:

```
8 4
1 2 1 3
25 1 2 1
4 20 3 3
3 30 12 2
```

poprawnym wynikiem jest:

```
2 1 4 2
```

1	*2*	*1*	*3*
25	*1*	*2*	*1*
4	*20*	*3*	*3*
3	*30*	*12*	*2*

Cenowa mapa terenu i wyznaczona działka w drugim przykładzie

Rozwiązanie

Rozwiązania nieoptymalne

Dla uproszczenia, kwadraty jednostkowe, na które podzielona jest mapa, będziemy nazywać *polami*, a sumę cen pól zawartych w prostokącie — *ceną prostokąta*. Zadanie polega na znalezieniu prostokąta, którego cena mieści się w przedziale od k do $2k$.

Proste rozwiązania

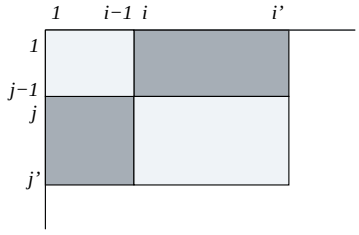
Najprostsze rozwiązanie polega na przejrzeniu wszystkich możliwych prostokątów i wyznaczeniu ceny każdego z nich przez proste zsumowanie cen pól. Jednak wszystkich prostokątów jest $O(n^4)$ i każdy zawiera średnio $O(n^2)$ pól. Prosty program złożony z sześciu zagnieżdżonych pętli **for** działa więc w czasie rzędu $O(n^6)$. Rozwiązanie takie zostało zaimplementowane w programach `kups3.cpp` i `kups6.java`.

W poszukiwaniu efektywniejszego podejścia możemy zajrzeć do różnych archiwów zadań — także z Olimpiady Informatycznej. Można znaleźć w nich wiele problemów opartych na schemacie podobnym jak „Kupno gruntu". Zazwyczaj jest w nich dana prostokątna „mapa" podzielona na jednostkowe pola, którym są przypisane pewne nieujemne

wagi, i należy znaleźć prostokąt o sumarycznej wadze spełniającej określone warunki (patrz, na przykład, zadania „Artemis" z XVI IOI [12] oraz „Piramida" z XVIII IOI [14]). Istnieją pewne standardowe techniki, które można zastosować przy rozwiązywaniu tego typu zadań. Jedna z nich polega na wstępnym przetworzeniu mapy tak, aby potem móc w stałym czasie określać dla dowolnego prostokąta sumę wag zawartych w nim pól. (Została ona opisana także w zadaniu „Mapa gęstości" w [8].)

Oznaczmy przez $c_{i,j}$ cenę pola o współrzędnych (i, j), a przez $K_{i,j}^{i',j'}$ cenę prostokąta, którego lewy górny róg ma współrzędne (i, j), a prawy dolny (i', j'):

$$K_{i,j}^{i',j'} = \sum_{x=i}^{i'} \sum_{y=j}^{j'} c_{x,y}.$$



Rys. 1: Wyznaczanie ceny prostokąta

Jeżeli znamy wartości $K_{1,1}^{x,y}$ dla wszystkich $x = 1, 2, \ldots, n$, $y = 1, 2, \ldots, n$, to możemy łatwo wyznaczyć każdą inną wartość $K_{i,j}^{i',j'}$, korzystając z tożsamości (patrz też rys. 1):

$$K_{i,j}^{i',j'} = K_{1,1}^{i',j'} - K_{1,1}^{i-1,j'} - K_{1,1}^{i',j-1} + K_{1,1}^{i-1,j-1}.$$

Z kolei, aby wyznaczyć wartości $K_{1,1}^{x,y}$, wystarczy przejrzeć kolejne wiersze mapy, idąc z góry na dół, i skorzystać z tożsamości:

$$K_{1,1}^{x,y} = K_{1,1}^{x-1,y} + K_{1,1}^{x,y-1} - K_{1,1}^{x-1,y-1} + c_{x,y}$$

lub sumować na bieżąco wagi pól w danym wierszu i skorzystać z tożsamości:

$$K_{1,1}^{x,y} = K_{1,1}^{x,y-1} + (c_{1,y} + c_{2,y} + \cdots + c_{x,y}).$$

Widzimy więc, że możemy wyznaczyć wszystkie wartości $K_{1,1}^{x,y}$ w czasie i pamięci $O(n^2)$. Mając te wartości, wystarczy przejrzeć wszystkie możliwe prostokąty (jest ich $O(n^4)$) i sprawdzić ich ceny (każdą obliczamy w czasie $O(1)$). W ten sposób dostajemy algorytm działający w czasie rzędu $O(n^4)$ i pamięci rzędu $O(n^2)$. Jest on zaimplementowany w plikach `kups2.cpp` i `kups5.java`.

Poszukiwanie w przedziale

Jak dotąd przedstawiliśmy rozwiązania, w których analizujemy ceny wszystkich prostokątów. Nie wykorzystaliśmy faktu, że szukana suma musi być liczbą z podanego przedziału — najwyższa pora uwzględnić ten fakt.

Obserwacja 1. Załóżmy, że mamy dwa prostokąty: jeden większy, a drugi mniejszy, zawarty w większym. Ponieważ ceny pól są nieujemne, więc cena większego prostokąta nie może być mniejsza niż cena mniejszego prostokąta. Inaczej mówiąc, jeśli $1 \leqslant i_1 \leqslant i_2 \leqslant i_3 \leqslant i_4 \leqslant n$, oraz $1 \leqslant j_1 \leqslant j_2 \leqslant j_3 \leqslant j_4 \leqslant n$, to:

$$K_{i_2,j_2}^{i_3,j_3} \leqslant K_{i_1,j_1}^{i_4,j_4}.$$

Korzystając z tej obserwacji, możemy pominąć niektóre prostokąty. Jeśli trafimy na prostokąt o cenie większej niż $2k$, to możemy pominąć wszystkie większe, zawierające go prostokąty. Analogicznie, jeśli mamy prostokąt o cenie mniejszej niż k, to możemy pominąć wszystkie zawarte w nim prostokąty.

Ustalmy na chwilę pewne wartości współrzędnych pionowych $1 \leqslant j \leqslant j' \leqslant n$. Możemy przejrzeć wszystkie prostokąty o górnej krawędzi na wysokości j i dolnej na wysokości j' tzw. metodą „gąsienicy". Prostokąty rozważamy w kolejności od lewej do prawej, przy czym, jak zobaczymy, wystarczy, że sprawdzimy tylko niektóre z nich. Zaczynamy od prostokąta o lewym górnym rogu w $(1, j)$ i prawym dolnym rogu w $(1, j')$.

Powiedzmy, że lewy górny róg aktualnie rozważanego prostokąta ma współrzędne (i, j), a prawy dolny — (i', j'). W zależności od wartości $K_{i,j}^{i',j'}$ rozróżniamy trzy przypadki:

- Jeśli $K_{i,j}^{i',j'} < k$, to znaczy, że ani nasz prostokąt, ani żaden w nim zawarty nie spełnia warunków zadania. W szczególności, możemy pominąć prostokąty o prawym dolnym rogu w (i', j') i lewym górnym na prawo od (i, j). Możemy więc powiększyć nasz prostokąt, zwiększając i' o 1.

- Jeśli $K_{i,j}^{i',j'} > 2k$, to znaczy, że ani nasz prostokąt, ani żaden zawierający go większy prostokąt nie spełniają warunków zadania. W szczególności, możemy pominąć prostokąty o lewym górnym rogu w (i, j) i prawym dolnym na prawo od (i', j'). Możemy więc zmniejszyć nasz prostokąt, zwiększając i o 1.

- Jeżeli żaden z poprzednich przypadków nie zachodzi, to znaczy, że $k \leqslant K_{i,j}^{i',j'} \leqslant 2k$ i nasz prostokąt spełnia warunki zadania.

Badany prostokąt jest raz węższy, raz szerszy — jak gąsienica pełznąca z lewa na prawo. Ile kroków może wykonać taka „gąsienica"? Co najwyżej $2n$, gdyż w każdym kroku zwiększamy o 1 współrzędną lewego lub prawego boku prostokąta. Jeżeli weźmiemy pod uwagę, że możliwych par j i j' jest $O(n^2)$, otrzymujemy algorytm działający w czasie $O(n^3)$ i pamięci $O(n^2)$. Rozwiązanie takie jest zaimplementowane w plikach `kups1.cpp` i `kups4.java`.

Opisany algorytm możemy stosować wszędzie tam, gdzie cena szukanego prostokąta powinna należeć do zadanego przedziału. Jednak w naszym przypadku nie jest to dowolny przedział, ale przedział postaci: od k do $2k$. Wykorzystując ten fakt, można skonstruować jeszcze szybsze rozwiązanie.

Rozwiązanie optymalne

Zacznijmy od kilku prostych obserwacji.

Obserwacja 2. Jeżeli, dla pewnych i i j, $c_{i,j} > 2k$, to pole (i, j) nie może być zawarte w żadnym prostokącie spełniającym warunki zadania.

Jest to dosyć oczywisty fakt: cena dowolnego prostokąta zawierającego pole (i, j) musi być większa niż $2k$. W związku z tym pola o cenach większych niż $2k$ nazwiemy *zabronionymi*. Pozostałe pola nazwiemy *dozwolonymi*. Cały prostokąt nazwiemy *dozwolonym*, jeśli wszystkie pola w nim zawarte są dozwolone. Dla uproszczenia przyjmiemy, że wszystkie pola leżące poza mapą są zabronione.

Fakt 1. *Jeżeli prostokąt jest dozwolony i jego cena jest nie mniejsza niż k, to istnieje zawarty w nim prostokąt, którego cena jest między k a 2k.*

Dowód: Dowód przeprowadzimy przez indukcję ze względu na wielkość prostokąta. Oznaczmy przez P dozwolony prostokąt o cenie $S \geqslant k$.

1. Jeśli prostokąt P składa się tylko z jednego pola, to musi ono być dozwolone, czyli $S \leqslant 2k$. Tym samym P stanowi szukane rozwiązanie.

2. Załóżmy, że prostokąt P składa się z więcej niż jednego pola. Jeżeli $S \leqslant 2k$, to stanowi on szukane rozwiązanie.

   Załóżmy przeciwnie, że $S > 2k$. Wówczas można podzielić P, w pionie lub w poziomie, na dwa mniejsze prostokąty P_1 i P_2. Oznaczmy ich ceny, odpowiednio, przez S_1 i S_2. Bez straty ogólności możemy założyć, że $S_1 \leqslant S_2$. Mamy więc $S_1 + S_2 = S > 2k$. Tak więc $S_2 > k$. Z założenia indukcyjnego, prostokąt P_2 zawiera szukane rozwiązanie.

■

Pokazany fakt pozwala nam zredukować problem z zadania do znalezienia *prostokąta dozwolonego o maksymalnej cenie*:

- Jeśli jego cena jest mniejsza od k, to prostokąt będący rozwiązaniem nie istnieje.
- Jeśli zaś jego cena jest większa lub równa k, to zawiera on szukany prostokąt. Dowód Faktu 1 stanowi jednocześnie przepis na konstrukcję rozwiązania zadania w obrębie znalezionego maksymalnego prostokąta. Jeżeli w każdym kroku będziemy dzielić dany prostokąt mniej więcej w połowie, to po co najwyżej $O(\log n)$ krokach otrzymamy rozwiązanie.

Maksymalny dozwolony prostokąt

Zastanówmy się, jak znaleźć prostokąt dozwolony o maksymalnej cenie. Jak w przypadku rozwiązań nieoptymalnych, okazuje się, że pomocne może być poszperanie w olimpijskich archiwach — rozwiązanie tego problemu zostało opisane w rozdziale „Działka” w [9]. Opiszemy je jednak również tutaj.

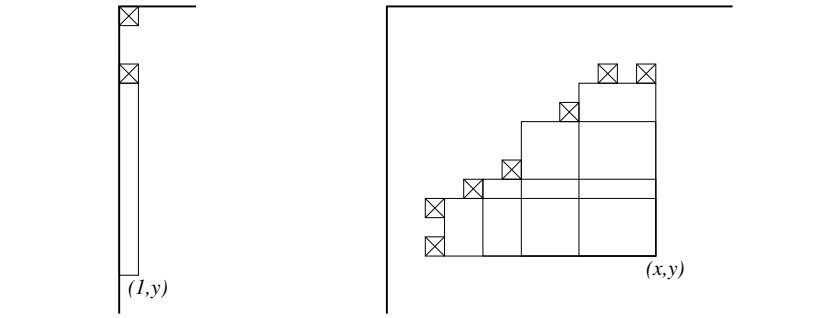
Rozwiązanie polega na sukcesywnym przeglądaniu prostokątów dozwolonych, które mogą być maksymalne ze względu na cenę. Zauważmy, że pewne prostokąty możemy pominąć — jeśli mamy dwa dozwolone prostokąty A i B, gdzie A zawiera się w B, to wystarczy rozważyć B (A nie może mieć ceny większej niż B). Przypomnijmy, że naszą mapę

zorientowaliśmy tak, że lewy górny róg ma współrzędne $(0,0)$, pierwsza współrzędna (x) rośnie w prawo wzdłuż osi poziomej, a druga (y) — w dół wzdłuż osi pionowej. Oznaczmy przez $M_{x,y}$ zbiór lewych górnych rogów prostokątów dozwolonych, których prawy dolny róg ma współrzędne (x,y). Dodatkowo będziemy wymagać, by prostokątów umieszczonych w zbiorze $M_{x,y}$ nie dało się rozszerzyć w górę lub w lewo do prostokąta dozwolonego — takie prostokąty nazwiemy *nierozszerzalnymi*. Punkty należące do zbiorów $M_{x,y}$ będziemy utożsamiać z prostokątami, które one (wraz z punktem (x,y)) wyznaczają, i dla uproszczenia będziemy mówić o prostokątach należących do $M_{x,y}$.

Oczywiście dla każdego pola zabronionego (x,y) (w tym dla każdego pola spoza mapy) mamy $M_{x,y} = \emptyset$. Rozważmy pole dozwolone (x,y). Jeśli $x = 1$, to:

$$M_{1,y} = \{(1, \max\{j : 0 \leqslant j \leqslant y,\ (1,j) \text{ jest zabronione}\} + 1)\}.$$

Dla $x > 1$ zbiór $M_{x,y}$ składa się z prostokątów o takich lewych górnych rogach (x',y'), że zarówno wśród $(x',y'-1),(x'+1,y'-1),\ldots,(x,y'-1)$, jak i wśród $(x'-1,y'),(x'-1,y'+1),\ldots,(x'-1,y)$ muszą występować pola zabronione. Ilustruje to rysunek 2.



Rys. 2: Prostokąty tworzące zbiory $M_{1,y}$ i $M_{x,y}$. Pola zabronione oznaczono przekreślonymi kwadracikami.

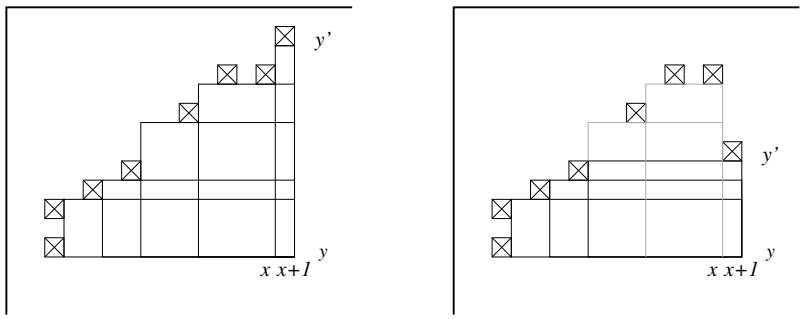
Niech $M_{x,y} = \{(x_1,y_1),(x_2,y_2),\ldots,(x_k,y_k)\}$, przy czym $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_k$. Z tego, że prostokąty ze zbioru $M_{x,y}$ są nierozszerzalne, wynika, że $x_1 < x_2 < \cdots < x_k$ oraz $y_1 > y_2 > \cdots > y_k$. Zastanówmy się teraz, jak zmieni się zbiór $M_{x,y}$, gdy zwiększymy x o jeden, czyli jak mają się do siebie $M_{x,y}$ i $M_{x+1,y}$? Zależy to od tego, na jakiej wysokości znajduje się najniższe pole zabronione leżące nad polem $(x+1,y)$ — oznaczmy je przez $(x+1,y')$:

$$y' = \max\{j : 0 \leqslant j \leqslant y,\ (x+1,j) \text{ jest zabronione}\}.$$

Możliwe są cztery przypadki (dwa najważniejsze z nich przedstawiono na rys. 3):

- jeśli $y'+1 < y_k$, to $M_{x+1,y} = M_{x,y} \cup \{(x+1,y'+1)\}$,
- jeśli $y'+1 = y_k$, to $M_{x+1,y} = M_{x,y}$,
- jeśli $y_k < y'+1 \leqslant y$, to:

$$M_{x+1,y} = \{(x_i,y_i) : y_i > y', i = 1,2,\ldots,k\} \cup \{(\min\{x_i : 1 \leqslant i \leqslant k,\ y_i \leqslant y'+1\}, y'+1)\},$$



Rys. 3: Zależność między $M_{x,y}$ i $M_{x+1,y}$.

- jeśli $y' = y$, to $M_{x+1,y} = \emptyset$.

Zbiory $M_{x,y}$ konstruujemy wiersz po wierszu, dla kolejnych wartości y. Zauważmy, że poszukując prostokąta dozwolonego o maksymalnej cenie, nie musimy brać pod uwagę wszystkich prostokątów ze zbioru $M_{x,y}$. Jeśli punkt $(x',y') \in M_{x,y} \cap M_{x+1,y}$, to wyznaczony przez niego prostokąt ze zbioru $M_{x,y}$ zawiera się w całości w wyznaczonym przez ten punkt prostokącie ze zbioru $M_{x+1,y}$. Jeśli więc będziemy konstruować kolejne zbiory $M_{x,y}$ (dla danego y) w kolejności od lewej do prawej (dla $x = 1,2,\ldots$), to wystarczy, że będziemy uwzględniać tylko ceny prostokątów o prawym dolnym rogu w (x,y) i lewych górnych rogach ze zbioru $M_{x,y} \setminus M_{x+1,y}$ (czyli „odpadających" przy przejściu od zbioru $M_{x,y}$ do zbioru $M_{x+1,y}$).

Implementacja

Zastanówmy się, jakiej struktury danych użyć do reprezentowania zbiorów $M_{x,y}$, tak aby móc je efektywnie przeglądać. Zauważmy, że przekształcając $M_{x,y}$ w $M_{x+1,y}$, usuwamy pewną liczbę najwyżej położonych punktów i ewentualnie dodajemy jeden, położony wyżej niż wszystkie pozostałe. Odpowiednią strukturą będzie więc stos — pola $(x_1,y_1),(x_2,y_2),\ldots,(x_k,y_k)$ będą na nim ułożone w kolejności od spodu do wierzchu. Oto pseudokod programu, który znajduje prostokąt dozwolony o maksymalnej cenie:

```
1:  s := −1;  { maksymalna dotychczas uzyskana cena }
2:  w := NULL;  { najlepszy znaleziony prostokąt dozwolony }
3:  for y := 1 to n do begin
4:    M := ∅;  { stos }
5:    for x := 1 to n+1 do begin
6:      y' := max{j : 0 ⩽ j ⩽ y, (x, j) jest zabronione};
7:      x' := x;
8:      while (M ≠ ∅) and (M.top.y < y' + 1) do begin
9:        if K^{x−1,y}_{M.top} > s then begin
10:         s := K^{x−1,y}_{M.top};
11:         w := prostokąt o rogach w M.top i (x − 1, y);
12:       end;
13:       x' := M.top.x;
```

```
14:          M.pop;
15:        end;
16:        if (M = ∅) ∨ (M.top.y > y' + 1) then M.push(x', y' + 1);
17:      end;
18:    end;
19:    return w;
```

Złożoność rozwiązania

Jaka jest złożoność czasowa przedstawionego rozwiązania? Zawartość wewnętrznej pętli **for** (linie 6–16) jest powtarzana $O(n^2)$ razy. Pokażemy, że jest ona wykonywana w stałym czasie (zamortyzowanym). Musimy przyjrzeć się dokładniej dwóm instrukcjom: obliczeniu wartości y' (linia 6) i pętli **while** (linie 8–15).

Linia 6 — maksimum. Nie musimy obliczać najbliższych pól zabronionych za każdym razem od nowa. Dysponując wartościami wyznaczonymi dla poprzedniego wiersza $(y-1)$, wystarczy poprawić je jedynie w tych kolumnach, gdzie w wierszu y pojawiło się pole zabronione. W ten sposób, potrzebne maksimum obliczamy tak naprawdę w czasie stałym.

Linie 8–15 — pętla while. Zauważmy, że w każdym kroku pętli ze stosu jest zdejmowany jeden element. Jednak elementy są wkładane na stos poza pętlą **while**, a każdy włożony element może być zdjęty tylko raz. To oznacza, że sumaryczna liczba wykonań tej pętli w czasie rozważania jednego wiersza jest rzędu $O(n)$, czyli pętla jest wykonywana w stałym czasie zamortyzowanym.

Widzimy więc, że przedstawiona procedura działa w czasie $O(n^2)$. Dodatkowe operacje, pozwalające na podstawie prostokąta dozwolonego o maksymalnej cenie wyznaczyć poszukiwaną działkę o cenie z przedziału $[k,2k]$, zajmują czas $O(n^2+\log n)$. Tak więc, cały algorytm działa w czasie $O(n^2)$ i pamięci $O(n)$. Jest to optymalne rozwiązanie, gdyż samo wczytanie danych wymaga wykonania n^2 operacji. Zostało ono zaimplementowane w plikach `kup.cpp` i `kup1.java`.

Testy

Testy zostały tak dobrane, że programy o złożoności czasowej $O(n^6)$ nie dostają żadnych punktów. Rozwiązania działające w czasie $O(n^4)$ uzyskują 20 %, a działające w czasie $O(n^3)$ — 40 % punktów.
W poniższej tabeli przedstawiono charakterystykę testów.

Nazwa	n	Opis
kup0 .in	3	test przykładowy, brak rozwiązania
kup0a.in	4	test przykładowy
kup1a.in	5	test losowy, poprawnościowy
kup1b.in	50	test strukturalny, wydajnościowy

Nazwa	n	Opis
kup1c.in	10	test losowy, poprawnościowy, brak rozwiązania
kup2a.in	15	test strukturalny, poprawnościowy
kup2b.in	60	test losowy, wydajnościowy i poprawnościowy
kup3a.in	200	test losowy, poprawnościowy
kup3b.in	200	test losowy, wydajnościowy i poprawnościowy
kup4a.in	500	test losowy, poprawnościowy
kup4b.in	500	test losowy, wydajnościowy i poprawnościowy
kup5 .in	1000	test losowy, wydajnościowy i poprawnościowy
kup6 .in	2000	test losowy, wydajnościowy i poprawnościowy
kup7 .in	500	test z polami zabronionymi na obu przekątnych, wydajnościowy
kup8 .in	2000	test z polami zabronionymi na obu przekątnych, wydajnościowy
kup9a.in	500	test z polami zabronionymi w kształcie konturu „karo”, wydajnościowy
kup9b.in	500	test losowy, wydajnościowy, brak rozwiązania
kup10a.in	2000	test z polami zabronionymi w kształcie konturu „karo”, wydajnościowy
kup10b.in	2000	test losowy, wydajnościowy, brak rozwiązania