Marek Cygan, Jakub Radoszewski
Treść zadania, Opracowanie

Marek Cygan, Jakub Radoszewski
Program

OI, Etap II, dzień pierwszy, 07.02.2007

Skalniak

Wicehrabia de Bajteaux jest właścicielem znanej na całym świecie kolekcji głazów. Dotychczas utrzymywał ją w piwnicach swojego pałacu, lecz teraz postanowił wyeksponować kolekcję w swoim ogromnym ogrodzie.

Ogrody wicehrabiego mają kształt kwadratu o bokach długości 1 000 000 000 jednostek. Boki kwadratu leżą w kierunkach wschód-zachód i północ-południe. Dla każdego głazu wicehrabia wyznaczył współrzędne punktu, w którym chciałby, żeby został on umiejscowiony (współrzędne te to odległości od południowego i zachodniego boku ogrodu), a następnie przekazał te informacje służącym. Niestety jednak zapomniał im powiedzieć, w jakiej kolejności podał współrzędne poszczególnych punktów. Dokładniej, współrzędne niektórych punktów mógł podać w kolejności odcięta-rzędna, a niektórych rzędna-odcięta. Nieświadomi tego faktu służący rozmieścili głazy, zakładając, że kolejność podawania współrzędnych jest standardowa (odcięta-rzędna).

Wicehrabia postanowił zadbać o bezpieczeństwo swojej kolekcji i ogrodzić ją płotem, tworząc skalniak. Płot ze względów estetycznych powinien mieć kształt prostokąta o bokach równoległych do boków ogrodu. Wicehrabia tak rozplanował uporządkowania współrzędnych punktów, żeby łączna długość potrzebnego płotu była jak najmniejsza (czyli spośród wszystkich uporządkowań par współrzędnych podanych punktów, uporządkowanie wicehrabiego wymaga minimalnej łącznej długości płotu). Przyjmujemy przy tym, że prostokąt może mieć boki zerowej długości.

Aby nie wyszła na jaw omyłkowa realizacja planu wicehrabiego, służący muszą poprzestawiać głazy tak, by długość potrzebnego do ich ogrodzenia płotu była najmniejsza z możliwych. Każdy głaz mogą przestawić tylko tak, by jego nowe położenie miało takie same współrzędne, jak aktualne, tylko w odwrotnej kolejności. Chcą się przy tym jak najmniej napracować, gdyż głazy są ciężkie. Chcieliby więc zminimalizować łączny ciężar przestawianych głazów.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia aktualne rozmieszczenie głazów w ogrodach wicehrabiego oraz ich ciężar,*
- *wyznaczy taki sposób poprzestawiania głazów, który umożliwi ogrodzenie ich jak najkrótszym płotem, a dodatkowo zminimalizuje łączny ciężar przestawianych głazów,*
- *wypisze wynik na standardowe wyjście.*

Wejście

Pierwszy wiersz wejścia zawiera jedną liczbę całkowitą n ($2 \leqslant n \leqslant 1\,000\,000$), oznaczającą liczbę głazów w kolekcji wicehrabiego. Kolejnych n wierszy zawiera po trzy liczby całkowite

x_i, y_i oraz m_i ($0 \leqslant x_i, y_i \leqslant 1\,000\,000\,000$, $1 \leqslant m_i \leqslant 2\,000$), pooddzielane pojedynczymi odstępami i oznaczające współrzędne aktualnego położenia i ciężar i-tego głazu. Żadna para nieuporządkowana współrzędnych nie pojawia się na wejściu więcej niż raz.

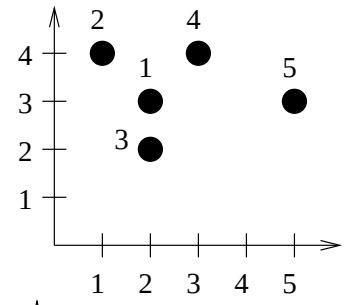
Wyjście

Pierwszy wiersz wyjścia powinien zawierać dwie liczby całkowite, oddzielone pojedynczym odstępem — najmniejszą możliwą do osiągnięcia długość płotu i minimalny ciężar kamieni, jakie trzeba przemieścić, by taki wynik osiągnąć. Drugi wiersz powinien zawierać ciąg n zer i/lub jedynek — i-ty znak powinien być jedynką, jeżeli w optymalnym rozwiązaniu przemieszczamy i-ty kamień, a zerem w przeciwnym przypadku. Jeżeli istnieje wiele poprawnych rozwiązań, Twój program powinien wypisać jedno z nich.

Przykład

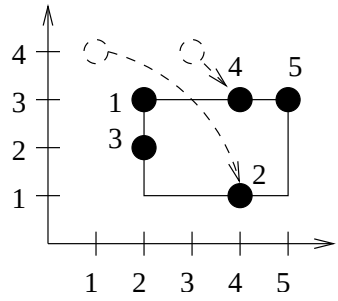
Dla danych wejściowych:

```
5
2 3 400
1 4 100
2 2 655
3 4 100
5 3 277
```



poprawnym wynikiem jest:

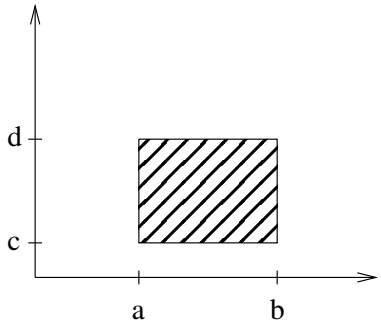
```
10 200
01010
```



Rozwiązanie

Test przydatności prostokąta

Zastanówmy się na początek nad łatwiejszym problemem niż postawiony w treści zadania: jak dla określonego położenia prostokątnego płotu sprawdzić, czy da się poprzestawiać głazy tak, by znalazły się wewnątrz ogrodzenia. Jeżeli takie przestawienie jest możliwe, to chcielibyśmy także umieć wyznaczyć jego minimalny koszt. Położenie płotu będziemy odtąd identyfikować z jego rzutami na każdą z osi, to znaczy czwórką liczb a, b, c, d, takich że prostokąt ogrodzony płotem jest iloczynem kartezjańskim $[a, b] \times [c, d]$.



Parametry opisujące prostokątny płot

Prostokąt $[a,b] \times [c,d]$ nazwiemy *przydatnym*, jeżeli da się w nim umieścić wszystkie głazy (przestawiając w razie konieczności pewne z nich w opisany w treści zadania sposób). Przydatność prostokąta można w prosty sposób sprawdzić, wyznaczając jednocześnie optymalny koszt przeniesienia głazów. Dla każdego głazu sprawdzamy mianowicie, czy istnieje jego położenie, mieszczące się w granicach badanego prostokąta. Jeżeli nie istnieje, to prostokąt nie jest przydatny. W przeciwnym wypadku trzeba wybrać mniej kosztowny sposób umieszczenia głazu w ogrodzeniu — jeżeli głaz bez przestawiania mieści się wewnątrz płotu, to oczywiście pozostawiamy go na miejscu; w przeciwnym razie jesteśmy zmuszeni go przestawić i do całkowitego kosztu operacji doliczyć jego ciężar. Ten prosty, ale bardzo przydatny w dalszej części, algorytm ma oczywiście złożoność czasową $O(n)$.

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

Wykorzystując algorytm z poprzedniego rozdziału możemy rozwiązać zadanie, sprawdzając wszystkie możliwe prostokątne płoty. Oczywiście bierzemy pod uwagę tylko prostokąty przydatne — spośród nich wybieramy prostokąt o najkrótszym obwodzie, a jeśli jest takich wiele, to wymagający najmniejszego kosztu przestawienia głazów. Jak już zauważyliśmy, każde położenie płotu może zostać scharakteryzowane przez cztery liczby a,b,c,d. Niech Z oznacza zbiór wszystkich liczb, które występują w danych jako współrzędne głazów, czyli $Z = \{x_i : 1 \leqslant i \leqslant n\} \cup \{y_i : 1 \leqslant i \leqslant n\}$. Bez straty ogólności możemy ograniczyć nasze rozważania do prostokątów, dla których $a,b,c,d \in Z$ — każdy inny prostokąt można bowiem zmniejszyć, nie zmieniając jego podstawowych własności: przydatności i kosztu przestawiania głazów.

Poczynione spostrzeżenie ogranicza liczbę potencjalnych położeń płotu do $O(n^4)$ ($|Z| \leqslant 2 \cdot n$) i, w połączeniu z liniowym algorytmem sprawdzania przydatności prostokąta, daje rozwiązanie zadania o złożoności czasowej $O(n^5)$. Aby przyspieszyć ten prosty algorytm, przyjrzymy się bliżej zbiorowi Z. Niech *min* oznacza najmniejszą liczbę w Z, a *max* — największą. Zauważmy, że wartość *min* musi wystąpić jako rzędna lub odcięta położenia jednego z głazów — stanowi ona tym samym najlepsze dolne ograniczenie rzutu prostokąta na odpowiednią oś. Analogiczny argument dotyczy wartości *max*. Stąd $min \in \{a,c\}$ oraz $max \in \{b,d\}$, co daje cztery odrębne przypadki do rozważenia. W każdym z nich tylko dwie spośród liczb a,b,c,d pozostają niewiadomymi; rozważając każdy przypadek osobno

uzyskujemy zawsze $O(n^2)$ możliwych położeń płotu, co przy liniowym względem n czasie sprawdzania przydatności prostokąta daje rozwiązanie o złożoności czasowej $O(n^3)$. Jest ono zaimplementowane w plikach `skas2.cpp` oraz `skas5.pas`. Rozwiązanie to pozwalało uzyskać na zawodach 30% punktów.

Rozwiązanie o złożoności czasowej $O(n^2)$

Powyższe rozwiązanie można jeszcze przyspieszyć, uzyskując złożoność czasową $O(n^2)$. W tym celu poszukiwanie najlepszego położenia płotu podzielimy na dwie fazy:

1. znajdowanie minimalnej długości obwodu prostokąta przydatnego;
2. wyznaczanie najmniejszego kosztu poprzestawiania wszystkich głazów, tak aby zmieściły się w pewnym prostokącie o minimalnym obwodzie.

Przypomnijmy, że dwie spośród czterech liczb opisujących płot mamy ustalone; dla każdego z czterech przypadków wynikających z tych ustaleń możemy przeanalizować wszystkie wartości (należące do zbioru Z) jednej z niewiadomych liczb. Dla ustalenia uwagi (choć nie bez straty ogólności!) przyjmijmy, że $a = min$, $b = max$ i za c podstawiamy kolejno wszystkie wartości ze zbioru Z. Wśród parametrów a,b,c,d pozostaje nam wówczas jedna niewiadoma — d. Przeglądamy teraz wszystkie głazy i dla każdego z nich wyznaczamy te spośród dwóch możliwych położeń, które są zgodne z przyjętymi wartościami a,b,c, czyli pierwsza współrzędna mieści się w przedziale $[a,b]$, a druga jest nie mniejsza od c (jeżeli żadne położenie głazu nie spełnia tego warunku, to odrzucamy trójkę a,b,c — nie da się jej uzupełnić do prostokąta przydatnego). Następnie spośród wyznaczonych położeń wybieramy to, dla którego uzyskujemy minimalną wartość drugiej współrzędnej głazu — takie położenie wymusza najmniejszy wzrost wartości parametru d, a tym samym długości płotu. Po przeanalizowaniu wszystkich głazów uzyskujemy zatem najmniejszą wartość liczby d, dla której da się umieścić wewnątrz płotu wszystkie głazy, a $2(b-a+d-c)$ jest długością tego płotu.

Rozważając pozostałe trzy przypadki wynikające z ograniczeń $min \in \{a,c\}$ i $max \in \{b,d\}$ i przeglądając w pętli wszystkie możliwe wartości jednego z pozostałych parametrów, wyznaczamy minimalny możliwy do osiągnięcia obwód prostokąta w każdym z tych przypadków. Najmniejsza z tych czterech wartości jest poszukiwaną, optymalną długością płotu. Złożoność czasową tej fazy algorytmu uzyskujemy, przemnażając liczbę sprawdzanych trójek parametrów a,b,c,d przez złożoność czasową analizy wszystkich głazów, czyli wynosi ona $4n \cdot O(n) = O(n^2)$.

Skoro wyznaczyliśmy już minimalną długość płotu, możemy przejść do drugiej fazy rozwiązania. Przeprowadzamy ją w sposób bardzo podobny do poprzedniej. Ponownie rozważamy $4n$ sposoby ustalenia wartości trzech spośród czterech liczb a,b,c,d, ale tym razem — znając optymalny obwód ℓ prostokąta — czwarty parametr wyznaczamy jednoznacznie z równości $2(b-a+d-c) = \ell$ w czasie $O(1)$. Mając wartości wszystkich parametrów a,b,c,d, sprawdzamy w czasie liniowym względem n, czy taki prostokąt jest przydatny i jeżeli tak, to wyznaczamy minimalny koszt przestawienia głazów, tak by znalazły się wewnątrz ogrodzenia.

Złożoność drugiej fazy algorytmu to $4n \cdot O(n) = O(n^2)$. Złożoność całego algorytmu także wynosi $O(n^2)$. Implementacje tego rozwiązania znajdują się w plikach `skas1.cpp` oraz `skas4.pas`. Rozwiązanie to pozwalało uzyskać na zawodach około 50% punktów.

Rozwiązanie wzorcowe

Wprowadzenie

Opisane powyżej rozwiązania wymagały kilku prostych spostrzeżeń, które pozwalały usprawnić pierwsze (proste, lecz czasochłonne) rozwiązanie do algorytmu działającego w czasie $O(n^2)$. Zaprezentowane przy tym techniki poprawiania złożoności czasowej można z powodzeniem wykorzystywać w konstrukcji wielu innych algorytmów. Tymczasem rozwiązanie wzorcowe, o złożoności czasowej $O(n)$, wymaga zupełnie innego podejścia opartego na kilku nietrywialnych do udowodnienia i momentami zaskakujących spostrzeżeniach. Zastosowaną w nim metodę prezentujemy w sposób, w jaki została ona wymyślona. Rozpoczniemy od rozwiązania maksymalnie uproszczonego wariantu problemu. Następnie, poprzez analizę ogólniejszej wersji, dojdziemy do rozwiązania problemu postawionego w treści zadania.

Rysunki w niniejszym rozdziale przedstawiają położenia prostokątnego płotu w dość nietypowy sposób — osie układu współrzędnych są narysowane *równolegle* do siebie. Taki sposób przedstawienia jest konieczny do właściwego zilustrowania zaprezentowanych dowodów.

Rozwiązanie uproszczonej wersji zadania

Spróbujmy na początek znaleźć *jakikolwiek* sposób poprzestawiania głazów, w którym minimalizujemy długość płotu koniecznego do ich ogrodzenia i nie zwracamy uwagi na ich ciężar. Okazuje się, że dobrym sposobem jest ustawienie głazów tak, by **każdy głaz miał pierwszą współrzędną nie większą niż drugą.** To oznacza, że dla głazu położonego na pozycji (x,y), jeśli $x > y$, to dokonujemy jego przestawienia, a w przeciwnym przypadku pozostawiamy go na miejscu. Oznaczmy uzyskane w ten sposób ustawienie głazów przez M.

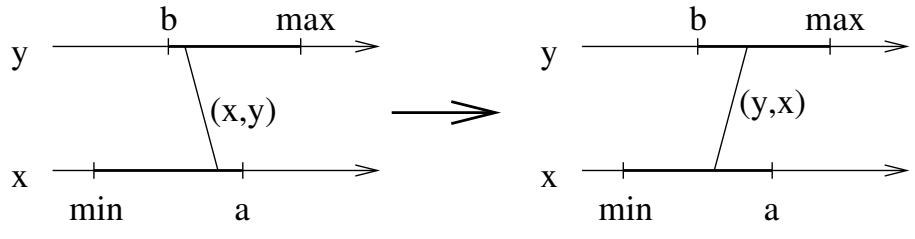
Niech U będzie dowolnym ustawieniem głazów, dla którego długość płotu potrzebnego do ich ogrodzenia jest minimalna. Pokażemy, że dla ustawienia M uzyskujemy taką samą długość płotu. Wykażemy to, przestawiając głazy w ustawieniu U w ten sposób, by nie zwiększyć długości otaczającego go płotu i doprowadzić do jego zrównania z ustawieniem M. Dowód przeprowadzimy odrębnie dla dwu przypadków:

1. w ustawieniu U wartości *max* i *min* występują na różnych osiach,
2. w ustawieniu U wartości *min* i *max* występują na tej samej osi,

gdzie *min* i *max* są zdefiniowane jak poprzednio.

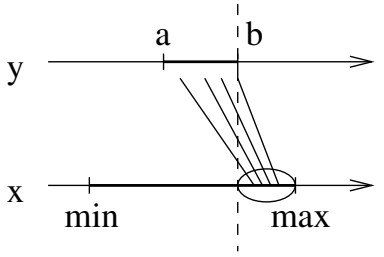
Rozważmy najpierw pierwszy przypadek. Przypuśćmy, że w ustawieniu U wartość *min* występuje jako odcięta pewnego punktu, a wartość *max* jako rzędna — w przeciwnym przypadku możemy przestawić wszystkie głazy w układzie U. Najkrótszy płot potrzebny do ogrodzenia układu U ma teraz postać $[min,a] \times [b,max]$ dla pewnych wartości parametrów a oraz b. Jeżeli jakiś głaz w U ma współrzędne (x,y), takie że $x > y$, to możemy

go przestawić, nie wychodząc poza ogrodzenie. Dzieje się tak dlatego, że skoro przed przestawieniem $x \leqslant a$ oraz $y \geqslant b$, to $y < x \leqslant a$ oraz $x > y \geqslant b$, a zatem przestawiony głaz mieści się w pierwotnym ogrodzeniu dla U. To kończy dowód w pierwszym przypadku.



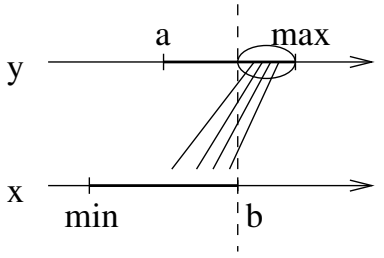
Pierwszy przypadek dowodu

Zajmijmy się teraz drugim przypadkiem. Jeżeli w ustawieniu U współrzędne *min* i *max* znajdują się na tej samej osi (możemy podobnie jak poprzednio doprowadzić do sytuacji, w której jest to oś odciętych), to na osi rzędnych rzut prostokąta stanowi pewien przedział $[a,b]$. Przyjrzyjmy się wszystkim głazom, których odcięta leży w przedziale $[b+1, max]$. Ich rzędne należą oczywiście do przedziału $[a,b]$.



Drugi przypadek dowodu — przed przestawieniem

Przestawiając wszystkie te głazy, otrzymujemy ustawienie U', którego rzut na oś odciętych mieści się w przedziale $[min, b]$, a rzut na oś rzędnych — w przedziale $[a, max]$.



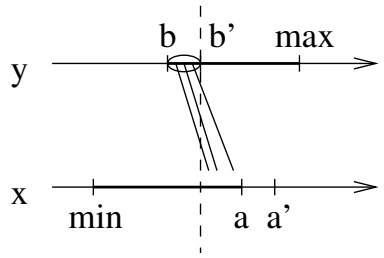
Drugi przypadek dowodu — po przestawieniu

Stąd ustawienie U' daje się ogrodzić płotem o długości nie większej niż $(b - min) + (max - a) = (max - min) + (b - a)$, czyli nie większej niż długość płotu otaczającego ustawienie U (z optymalności ustawienia U wynika ponadto, że musi to być

płot tej samej długości). Dodatkowo to ustawienie spełnia warunki przypadku pierwszego i możemy zastosować do niego opisaną powyżej procedurę sprowadzania do ustawienia M. To spostrzeżenie kończy uzasadnienie drugiego przypadku i zarazem cały dowód.

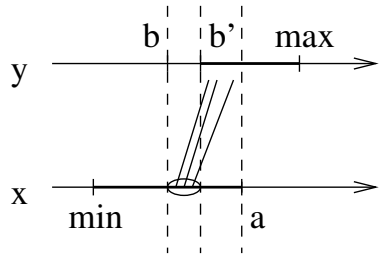
Optymalne prostokąty

Zastanówmy się teraz, ile jest (dla ustalonego zbioru głazów) *różnych ustawień płotu* o minimalnej długości obwodu. Ponownie rozważania podzielimy na dwa przypadki, analogiczne do powyższych. Pokażemy najpierw, że **istnieje dokładnie jedno (z dokładnością do przestawienia wszystkich głazów, co odpowiada zamianie osi miejscami) ustawienie płotu o minimalnej długości obwodu, w którym** *min* **i** *max* **są umieszczone na różnych osiach.** Dowód przeprowadzimy przez sprowadzenie do sprzeczności. Załóżmy, że istnieją dwa takie prostokąty: $[min,a]\times[b,max]$ oraz $[min,a']\times[b',max]$ (odpowiadające im ustawienia oznaczmy U_1 i U_2) i dla ustalenia uwagi przyjmijmy, że $a<a'$. Ponieważ prostokąty mają równe obwody, musi zachodzić $b<b'$. Oczywiście można tak dokonać przestawienia pewnych głazów, by z ustawienia U_1 uzyskać ustawienie U_2. Podczas przekształcenia głazy o rzędnych z przedziału $[b,b'-1]$ (występujące w ustawieniu U_1 i niemieszczące się w ogrodzeniu U_2) muszą zostać przestawione. Aby zmiana konfiguracji mogła się powieść, to odcięte wspomnianych głazów muszą zawierać się w przedziale $[b',max]$. Stąd możemy wnioskować, że $a\geqslant b'$, gdyż w przeciwnym przypadku zbiór możliwych wartości odciętych rozważanych głazów ($[min,a]\cap[b',max]$) byłby pusty (a to jest sprzeczne z założeniem o optymalności U_1).



Pierwszy przypadek — przed przestawieniem

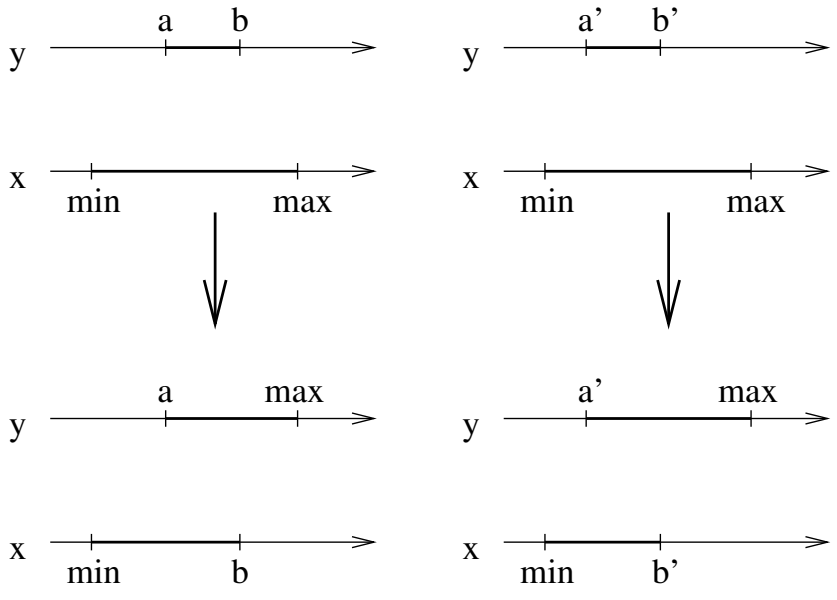
Można stąd wywnioskować, że $[b,b'-1]\subseteq[min,a]$, a zatem przestawiając *tylko* rozważane głazy, uzyskujemy prostokąt $[min,a]\times[b',max]$, czyli lepszy od obu wyjściowych.



Pierwszy przypadek — po przestawieniu

To daje nam żądaną sprzeczność.

Pokażemy teraz, że **z dokładnością do zamiany osi miejscami istnieje co najwyżej jedno optymalne ustawienie drugiego typu, w którym współrzędne** *min* **i** *max* **znajdują się na tej samej osi.** Gdyby istniały dwa takie prostokąty $[min,max]\times[a,b]$ oraz $[min,max]\times[a',b']$, to oba miałyby krótszy bok tej samej długości: $b-a=b'-a'$. Za pomocą opisanej w poprzednim rozdziale transformacji U do U', otrzymalibyśmy z nich dwa istotnie różne ustawienia pierwszego typu — $U_1=[min,b]\times[a,max]$ oraz $U_2=[min,b']\times[a',max]$ — co, jak już udowodniliśmy, nie jest możliwe.



Drugi przypadek — sprowadzenie do pierwszego

Algorytm

Przedstawione wyżej spostrzeżenia pozwalają skonstruować (zaskakująco proste) rozwiązanie zadania. Na początek przestawiamy każdy głaz, którego pierwsza współrzędna jest większa od drugiej. Dla tak otrzymanego ustawienia M wyznaczamy parametry a oraz b otaczającego je płotu $[min,a]\times[b,max]$. W ten sposób znamy już optymalną długość płotu i pozostaje ustalić, przy jakim ustawieniu osiągamy minimalny, sumaryczny ciężar przenoszonych głazów. Ponieważ mamy jedynie cztery możliwości ustawienia płotu o minimalnej długości obwodu:

- $[min,a]\times[b,max]$,
- $[b,max]\times[min,a]$,
- $[min,max]\times[b,a]$,
- $[b,a]\times[min,max]$,

to wystarczy dla każdej z nich, za pomocą opisanego na początku opracowania zadania testu przydatności prostokąta, sprawdzić, czy da się poprzestawiać głazy tak, by zostały ogrodzone tym płotem i jaki jest minimalny koszt takiego przestawienia. Najmniejszy z uzyskanych w ten sposób wyników jest poszukiwanym przez nas optymalnym sposobem poprzestawiania głazów.

Dzięki bardzo małej (stałej równej 4) liczbie przypadków do rozpatrzenia, rozwiązanie wzorcowe ma złożoność czasową $O(n)$. Jest ono zaimplementowane w plikach `ska.cpp` oraz `ska1.pas`.

Rozwiązania błędne

Warto zastanowić się, czy konieczne jest rozważanie wszystkich czterech przedstawionych wyżej przypadków. Okazuje się jednak, że dla każdego z nich istnieje taki zbiór głazów wraz z przypisanymi ciężarami, dla których optymalne ogrodzenie trzeba skonstruować tak, jak w tym przypadku. W plikach `skab1.cpp`, `skab2.cpp`, `skab3.cpp` oraz `skab4.cpp` znajdują się implementacje rozwiązania wzorcowego, w których pominięto rozważenie jednej z czterech możliwości ustawienia płotu.

Rozwiązania implementowane przez zawodników

Zaledwie kilku zawodników zaimplementowało poprawne rozwiązanie o złożoności czasowej liniowej; wszystkie te rozwiązania były podobne do wzorcowego. Duża grupa zawodników przedstawiła do oceny rozwiązania niepoprawne, w których pominięto niektóre spośród czterech wyżej opisanych przypadków; to niedopatrzenie zazwyczaj nie było przypadkowe, lecz było wynikiem nieudanej próby „zgadnięcia" zachłannej metody rozwiązania zadania. Rozwiązania tego typu uzyskiwały, wskutek grupowania testów, maksymalnie 10% punktów.

Pojawiło się także kilka rozwiązań o złożoności czasowej wykładniczej względem n, w których dla każdego sposobu poprzestawiania głazów (2^n możliwości) wyznaczano długość obwodu wynikowego płotu i koszt tego przestawienia. Łączna złożoność takiego algorytmu to $O(2^n \cdot n)$; jest on zaimplementowany w plikach `skas3.cpp` oraz `skas6.pas`. Rozwiązanie to zdobywało 10% punktów. Istniały także rozwiązania, których złożoności zależały od maksymalnej wartości współrzędnych głazów; zazwyczaj nie pozwalały one zdobyć żadnych punktów.

Zaplanowany sposób punktowania rozwiązań zadania *Skalniak* polegał na faworyzowaniu poprawnych rozwiązań wielomianowych (względem liczby głazów) w stosunku do prawie poprawnych liniowych rozwiązań zachłannych.

Testy

Rozwiązania zawodników były sprawdzane na 10 grupach testów. Prawie wszystkie grupy (poza pierwszą) składają się z 4 testów:

- testy *a* eliminują rozwiązanie `skab1.cpp`,

- testy *b* eliminują rozwiązanie `skab2.cpp`,
- testy *c* eliminują rozwiązanie `skab3.cpp`,
- testy *d* eliminują rozwiązanie `skab4.cpp`.

Wszystkie testy zostały wygenerowane w sposób losowy.

Poniżej zamieszczony jest opis każdej z grup testów. We wszystkich testach każdej grupy (poza pierwszą) wartość parametru *n* jest taka sama.

Nazwa	n	Opis
ska1a.in	3	bardzo mały test
ska1b.in	15	mały test
ska2(abcd).in	100	grupa losowych testów
ska3(abcd).in	500	grupa losowych testów
ska4(abcd).in	3 000	grupa losowych testów
ska5(abcd).in	10 000	grupa losowych testów
ska6(abcd).in	60 000	grupa losowych testów
ska7(abcd).in	300 000	grupa losowych testów
ska8(abcd).in	600 000	grupa losowych testów
ska9(abcd).in	1 000 000	grupa losowych testów
ska10(abcd).in	1 000 000	grupa losowych testów