Michał Włodarczyk
Treść zadania, Opracowanie

Igor Adamski
Program

Dostępna pamięć: 64 MB. *OI, etap I, 17.10–14.11.2011*

Studnia

Bajtazar wybrał się na wyprawę wzdłuż Suchej Rzeki, która przecina Pustynię Bajtocką. Niestety Sucha Rzeka wyschła, a Bajtazarowi skończyła się woda. Jedynym ratunkiem dla Bajtazara jest wykopanie studni na dnie wyschniętego koryta rzeki i dokopanie się do wody.

Bajtazar postanowił dobrze przemyśleć, co ma zrobić, zanim weźmie się za kopanie — wie, że jeśli opadnie z sił, a nie dokopie się do wody, to będzie miał skrajnie małe szanse na przetrwanie. Udało mu się określić, na jakiej głębokości pod dnem rzeki zalega woda. Wie też, na ile kopania starczy mu sił. Boi się tylko, żeby w czasie kopania nie osunęła się ziemia, gdyż może go pogrzebać żywcem. Bajtazar przesłał Ci (przez telefon satelitarny) opis topografii koryta rzeki. Poprosił Cię o wyznaczenie planu, gdzie ma kopać, tak aby dokopać się do wody, zanim opadnie z sił, a równocześnie, żeby zbocza w wykopie były jak najłagodniejsze. Bajtazar czeka na Twoją pomoc!

Wejście

W pierwszym wierszu standardowego wejścia są zapisane dwie dodatnie liczby całkowite n oraz m ($1 \leqslant n \leqslant 1\,000\,000$, $1 \leqslant m \leqslant 10^{18}$), oddzielone pojedynczym odstępem. W drugim wierszu znajduje się n dodatnich liczb całkowitych $x_1, x_2, \ldots, x_n$ ($1 \leqslant x_i \leqslant 10^9$), pooddzielanych pojedynczymi odstępami.

Bajtazarowi zostało sił na m ruchów łopatą. Liczby $x_1, x_2, \ldots, x_n$ stanowią opis topografii koryta Suchej Rzeki, zdatnego do kopania studni. Liczby te reprezentują grubość warstwy piasku ponad poziomem wody gruntowej, w kolejnych miejscach, co metr wzdłuż koryta rzeki. Jednym ruchem łopaty Bajtazar może wybrać tyle piachu, aby jedną z liczb x_i zmniejszyć o 1. Jeżeli którakolwiek z liczb x_i, powiedzmy x_k, zmniejszy się do 0, będzie to oznaczać, że Bajtazar dokopał się do wody. Poza dokopaniem się do wody w co najmniej jednym punkcie koryta rzeki, Bajtazarowi zależy na tym, aby na końcu następująca liczba z, charakteryzująca nachylenie piaszczystych zboczy:

$$z = \max_{i=1,2,\ldots,n-1} |x_i - x_{i+1}|,$$

była jak najmniejsza. Jeżeli istnieje wiele poprawnych wartości liczby k, reprezentującej miejsce, w którym Bajtazar powinien dokopać się do poziomu wody, Twój program powinien wypisać dowolną z nich. Możesz przyjąć, że poza miejscami $1, 2, \ldots, n$ na wszystkich głębokościach znajduje się lita skała oraz że Bajtazar zawsze będzie miał wystarczająco dużo siły, żeby w którymś miejscu dokopać się do wody.

W testach wartych co najmniej 35% punktów zachodzi dodatkowy warunek $n \leqslant 10\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście dwie liczby całkowite oddzielone pojedynczym odstępem: miejsce k, w którym Bajtazar powinien dokopać się do wody, oraz najmniejszą możliwą wartość liczby z.

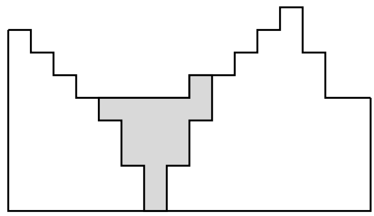
Przykład

Dla danych wejściowych:

```
16 15
8 7 6 5 5 5 5 5 6 6 7 8 9 7 5 5
```

poprawnym wynikiem jest:

```
7 2
```



Na powyższym rysunku prawidłowy wykop Bajtazara oznaczono szarym kolorem.

Rozwiązanie

Pierwsze podejście

Rozważania rozpoczniemy od skonstruowania najprostszego rozwiązania, które będziemy stopniowo ulepszać, aż dojdziemy do efektywnego algorytmu.

Na początku ustalmy k i z. Chcemy sprawdzić, czy można dokopać się do wody w punkcie k, tak aby nachylenie zbocza nie przekroczyło z. Dla tych parametrów, przez (y_i) oznaczmy ciąg opisujący topografię koryta rzeki po dokopaniu się do wody przy minimalnej liczbie ruchów łopatą (niedługo okaże się, że jest on wyznaczony jednoznacznie). Chcemy stwierdzić, czy:

$$\sum_{i=1}^{n}(x_i - y_i) \leqslant m.$$

Po pierwsze, wiemy, że $y_k = 0$, bo w tym miejscu dokopaliśmy się do wody. Dla $i \neq k$ pozostawmy $y_i = x_i$ i postarajmy się teraz wygładzić teren (tzn. usunąć część piasku tak, aby nachylenie nie przekraczało z). Wyobraźmy sobie, że idziemy w prawo i kiedy natrafiamy na próg o wysokości większej od z, pomniejszamy go. Następnie powtarzamy tę czynność, idąc w lewo. Poniżej przedstawiamy pseudokod tej procedury i dowód poprawności (ciągowi (y_i) odpowiada w kodzie tablica $y[]$):

for $i := 1$ **to** $n-1$ **do**
    if $y[i+1] > y[i] + z$ **then**
        $y[i+1] := y[i] + z;$
for $i := n$ **downto** 2 **do**
    if $y[i-1] > y[i] + z$ **then**
        $y[i-1] := y[i] + z;$

Lemat 1. Dla ciągu (y_i) po wykonaniu powyższej procedury zachodzi:

$$|y_i - y_{i+1}| \leqslant z \qquad \text{dla } i = 1, \ldots, n-1 \tag{1}$$

i każdy inny ciąg (v_i) o tej własności, $0 \leqslant v_i \leqslant x_i$, $v_k = 0$, spełnia:

$$v_i \leqslant y_i \qquad \text{dla } i = 1, \ldots, n.$$

Dowód: Załóżmy, że po wykonaniu podanego algorytmu nierówność (1) nie jest spełniona, tzn. dla pewnego j zachodzi $y_j > y_{j+1}+z$ lub $y_{j+1} > y_j+z$. Pierwsza możliwość jest wykluczona, ponieważ w drugiej części procedury dla $i = j + 1$ poprawiliśmy y_j, tak aby zachodziło $y_j \leqslant y_{j+1} + z$, a później żadna z tych liczb nie była zmieniana. W drugim przypadku natomiast wiemy, że po przejściu w prawo było $y_{j+1} \leqslant y_j + z$. Idąc w lewo, nie mogliśmy zwiększyć y_{j+1}. Jeśli zaś y_j zostało zmienione, to spełnia równość $y_j = y_{j+1} + z$, co jest sprzeczne z założeniem.

Drugą część tezy dowiedziemy indukcyjnie względem długości ciągów. Dokładniej, pokażemy, że dla dowolnego ciągu (x'_i) (odpowiadającego ciągowi (x_i) z ustawionym $x_k = 0$) i ciągów (y_i) i (v_i), ograniczonych z góry przez ciąg x'_i i spełniających nierówność typu (1), przy czym ciąg y_i jest skonstruowany za pomocą podanego wyżej algorytmu, dla każdego i zachodzi $v_i \leqslant y_i$. Baza indukcji (dla ciągów długości 1) jest trywialna. Przypuśćmy, że teza zachodzi dla wszystkich ciągów długości $n - 1$. Pokażemy, że stąd wynika teza dla ciągów długości n.

Zauważmy, że $v_2, y_2 \leqslant \min(x'_2, x'_1 + z)$. Co więcej, gdybyśmy w podanym wyżej algorytmie wystartowali od ciągu $\min(x'_2, x'_1 + z), x'_3, \ldots, x'_n$, skonstruowalibyśmy właśnie ciąg $y_2, y_3, \ldots, y_n$. Stosując w tym miejscu założenie indukcyjne, dostajemy $v_i \leqslant y_i$ dla $i > 1$. Wreszcie

$$v_1 \leqslant \min(x'_1, v_2 + z) \leqslant \min(x'_1, y_2 + z) = y_1,$$

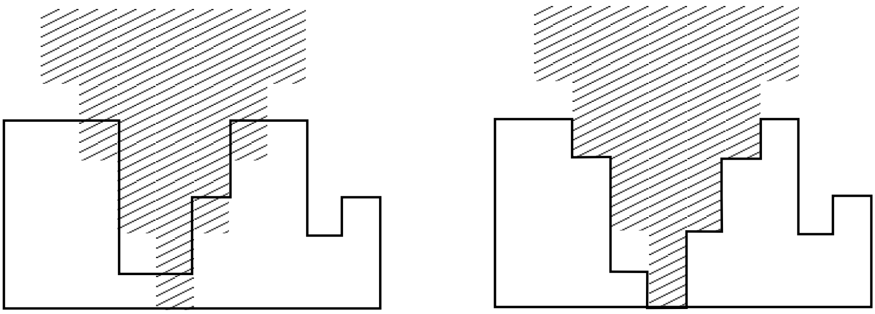
co dowodzi tezy indukcyjnej. ■

Z powyższego lematu wynika, że nasza procedura minimalizuje liczbę ruchów łopatą potrzebnych do wygładzenia terenu. Wiemy zatem, jak sprawdzać w czasie $O(n)$, czy możemy dokopać się do wody dla zadanych k oraz z. Oczywiście, jeśli możemy dokopać się do wody w jakimś punkcie z nachyleniem nie większym niż z, to dla $z+1$ również dostaniemy pozytywną odpowiedź. Ponadto dla $z = \max x_i$ nie trzeba nic wygładzać — wystarczy dokopać się do wody w najniższym punkcie, a w treści zadania zagwarantowano, że jest to możliwe. To pozwala znajdować najmniejsze z przy pomocy wyszukiwania binarnego (w każdym kroku wyszukiwania rozważamy wszystkie możliwe k). Takie rozwiązanie działa w czasie $O(n^2 \log(\max x_i))$ i pozwalało na zawodach zdobyć trochę punktów, ale niestety nie radziło sobie z największymi testami. Implementacje można znaleźć w plikach `stus0.cpp` i `stus1.pas`.

Rozwiązanie wzorcowe

Aby pozbyć się złożoności kwadratowej, musimy sprytniej obliczać liczbę potrzebnych ruchów łopatą. Cofnijmy się do miejsca, w którym przyjęliśmy $y_k = 0$, i załóżmy, że z jest ustalone. Wiemy, że $y_{k-1}, y_{k+1} \leqslant z$, bo inaczej przekroczylibyśmy dozwolone nachylenie. Indukcyjnie pokazujemy, że $y_{k-j}, y_{k+j} \leqslant jz$ dla $j = 1, 2, \ldots$ Z tego wynika, że ze studni musimy usunąć cały piasek zawarty w „odwróconej piramidzie” o środku w k i nachyleniu z (patrz rys. 1).

Czy to wystarczy do zapewnienia bezpieczeństwa? Otóż nie; na rysunku widać, że problem może pojawić się z dala od pozycji k lub zbocze w okolicy pozycji k może mieć nieregularny kształt.



Rys. 1: Zakreskowany teren to odwrócona piramida dla $z = 2$. Po lewej stronie widać pierwotną topografię, po prawej zaś teren po usunięciu piasku z piramidy.

Jednak po chwili zastanowienia można zaryzykować tezę, że trudności te nie miałyby miejsca, gdyby nachylenie studni przed rozpoczęciem kopania nie przekraczało z. Nasz wysiłek umysłowy z pierwszego rozdziału nie pójdzie na marne — dzięki lematowi 1 wiemy, jak optymalnie wygładzić ciąg (x_i) dla zadanego z, a potem spróbujemy sztuczki z piramidą. Najpierw jednak musimy przekonać się, czy to aby na pewno wystarczy.

Lemat 2. Wygładzenie ciągu (x_i) do nachylenia z, a następnie usunięcie piasku zawartego w piramidzie o środku w k i nachyleniu z, zadaje bezpieczny wykop do punktu k przy minimalnej liczbie ruchów łopatą.

Dowód: Przez (y_i) oznaczmy ciąg po wygładzeniu, zaś przez (u_i) — ciąg końcowy. Usunięcie piasku z piramidy oznacza, że na pozycji i otrzymamy wyraz $u_i = \min(y_i, |i-k| \cdot z)$. Upewnimy się teraz, że dla każdego i zachodzi $u_i \leqslant u_{i+1} + z$. Bez trudu dostajemy:

$$\begin{aligned} u_i &\leqslant y_i \leqslant y_{i+1} + z, \\ u_i &\leqslant |i-k| \cdot z \leqslant |(i+1)-k| \cdot z + z, \end{aligned}$$

a z połączenia tych dwóch nierówności mamy

$$u_i \leqslant \min(y_{i+1} + z,\ |(i+1)-k| \cdot z + z) = \min(y_{i+1},\ |(i+1)-k| \cdot z) + z = u_{i+1} + z.$$

Symetryczną nierówność $u_{i+1} \leqslant u_i + z$ dowodzimy w taki sam sposób. Pokazaliśmy zatem, że (u_i) zadaje bezpieczny wykop.

Niech teraz (v_i) będzie dowolnym ciągiem zadającym bezpieczny wykop przy założeniach lematu. Jest on, w szczególności, ciągiem wygładzającym (x_i), więc na mocy lematu 1 wiemy, że $v_i \leqslant y_i$ dla każdego i. W połączeniu z nierównością $v_i \leqslant |i-k| \cdot z$, otrzymujemy:

$$v_i \leqslant \min(y_i, |i-k| \cdot z) = u_i.$$

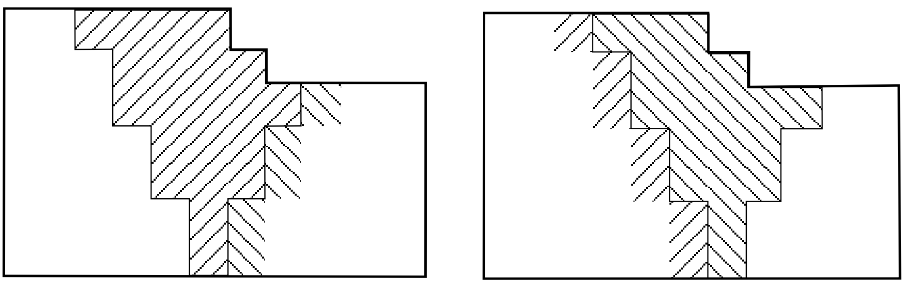
To dowodzi, że liczby ruchów łopatą nie da się poprawić. ■

Daje to następujący pomysł na lepszy algorytm: wyszukujemy binarnie z i dla ustalonego z wygładzamy teren, po czym obliczamy, ile piasku mieści się w odwróconych piramidach dla różnych k. Chcemy rozstrzygnąć, czy dla pewnego k łączna liczba ruchów łopatą nie przekracza m. Pozostaje tylko wymyślić sposób na szybkie obliczanie zawartości piramid.

Studnia pełna piramid

Na początek zauważmy, że dla $z = 0$ wystarczy sprawdzić, czy $\sum_{i=1}^{n} x_i \leqslant m$. Nieuwzględnienie tego przypadku może prowadzić do błędów w dalszej części rozważań.

Przyjmijmy teraz, że $z > 0$. Niech ciąg (y_i) oznacza wygładzony ciąg (x_i). Chcemy dla każdego k obliczyć sumę postaci $\sum_{i=1}^{n} \max(y_i - |i-k| \cdot z, 0)$. Warto pomyśleć, jak zmieni się taka suma, gdy k zwiększymy o 1. Na rysunku 2 widać dwa zbiory, w których leżą fragmenty gruntu odpowiednio wchodzące do piramidy i wychodzące z piramidy w trakcie jej przesunięcia. Nazwijmy te zbiory *prawą* i *lewą skośną*, a ich rozmiary oznaczmy przez p_i i l_i. Jasne jest, że znając liczbę pól każdej skośnej oraz zawartość piramidy dla $k = 1$, łatwo wyznaczymy odpowiedzi dla każdego k. Jako że pierwszą piramidę możemy zbadać w czasie $O(n)$, skupimy się teraz na obliczeniu ciągu p_i w czasie $O(n)$ (ciąg l_i można będzie wyznaczyć w ten sam sposób, wykonując obliczenia w odwrotnym kierunku).

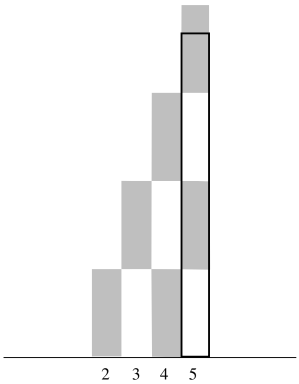


Rys. 2: Piramida o środku w k, przesuwając się o jednostkę w prawo, wchłania prawą skośną p_{k+1} i pozostawia po sobie lewą skośną l_k.

Rozważmy pewne $j \in \{1, \ldots, n\}$. Zastanówmy się, dla jakich i, i-ta prawa skośna zawiera piasek z pozycji j. Na pewno $y_j \bmod z$ najwyższych warstw trafi do skośnej o numerze $j - \left\lfloor \frac{y_j}{z} \right\rfloor$, oczywiście o ile skośna o takim numerze istnieje. Natomiast wszystkie skośne z przedziału $[\max(j + 1 - \left\lfloor \frac{y_j}{z} \right\rfloor, 1), j]$ dostaną po z warstw piasku, patrz także rys. 3.

Oczywiście, bezpośrednia aktualizacja skośnych doprowadziłaby nas z powrotem do czasu kwadratowego. Zamiast tego możemy jedynie zaznaczyć, gdzie zaczyna się i kończy przedział prawych skośnych, których rozmiary chcemy zwiększyć o z, a faktyczne sumowanie wykonać na samym końcu. Mówiąc dokładniej, jeśli w pomocniczej tablicy z licznikami $t[]$ na początku interesującego nas przedziału ustawimy 1, a tuż za jego końcem -1, to liczba przedziałów pokrywających skośną o numerze i będzie równa $t[1] + \ldots + t[i]$. Pozwala to wyznaczyć ciąg p_i następującym algorytmem (zakładamy, że tablice $t[]$ i $p[]$ są na początku wypełnione zerami).

```
for j := 1 to n do begin
   t[max(j + 1 − y[j] div z, 1)] += 1;
   t[j + 1] −= 1;
   if j − y[j] div z > 0 then
      p[j − y[j] div z] += y[j] mod z;
end
sum := 0;
for i := 1 to n do begin
   sum += t[i];
   p[i] += z · sum;
end
```



Rys. 3: Niech $j = 5$, $z = 3$ i $y_5 = 11$. Rysunek przedstawia, ile piasku z piątej pozycji trafi do poszczególnych prawych skośnych: 2 warstwy zasilą p_2, zaś do p_3, p_4, p_5 trafią po 3 warstwy.

Możemy w tym momencie podsumować nasze rozważania, prezentując pseudokod funkcji rozstrzygającej, czy można dokopać się do wody przy nachyleniu nieprzekraczającym z. Funkcja zwraca najmniejszy poprawny indeks k, jeśli bezpieczny wykop jest możliwy, a w przeciwnym razie zwraca -1.

```
function sprawdź_nachylenie(z, m)
begin
   y[] := wygładź(x[], z);
   p[] := oblicz_prawe_skośne(y[], z);
   l[] := oblicz_lewe_skośne(y[], z);
   koszt := 0;
   piramida := 0;
   for i := 1 to n do begin
      koszt += x[i] − y[i];
      piramida += max(y[i] − (i − 1) · z, 0);
   end
   if koszt + piramida ⩽ m then
      return 1;
```

```
    for k := 2 to n do begin
        piramida += p[k];
        piramida −= l[k − 1];
        if koszt + piramida ⩽ m then
            return k;
    end
    return −1;
end
```

Dołączając do powyższego pseudokodu wyszukiwanie binarne wartości z, uzyskujemy algorytm wykonujący $O(n \log(\max x_i))$ operacji i wykorzystujący liniową pamięć. Jego implementację można znaleźć w plikach `stu.cpp` i `stu1.pas`.

Rozwiązania niepoprawne

Autorzy testów przewidzieli takie błędy, jak wyrównywanie jedynie stoku wokół miejsca wykopu (plik `stub0.cpp`), pominięcie przypadku $z = 0$ (pliki `stub1.cpp` i `stub2.cpp`) czy też korzystanie ze zbyt małego typu do reprezentacji liczb całkowitych (plik `stub3.cpp`).

Inne niepoprawne rozwiązanie polega na próbie dokopania się do wody jedynie w miejscach o najmniejszej grubości piasku (plik `stub4.cpp`). Modyfikacja tej strategii, która wybiera np. 30 punktów o najmniejszej wysokości (plik `stub5.cpp`), działa nieco lepiej — jakkolwiek łatwo wskazać kontrprzykład na nią, trzeba przyznać, że nieźle radzi sobie z testami losowymi.

Rozwiązania wolniejsze

Oprócz wspomnianego na początku algorytmu, działającego w czasie $O(n^2 \log(\max x_i))$, można wymyślić szereg rozwiązań o złożoności czasowej $O(n \log n \log(\max x_i))$. Jednym z nich jest modyfikacja rozwiązania wzorcowego, korzystająca z tzw. drzewa licznikowego, zwanego też drzewem przedziałowym[1], do obliczenia zawartości skośnych (pliki `stus8.cpp`, `stus9.pas`, a także za pomocą drzewa potęgowego: `stus10.cpp` i `stus11.pas`). Innym przykładem jest zastosowanie wolniejszego algorytmu wygładzania terenu, w którym znajdujemy kolejno miejsca o najniższej wysokości gruntu, używając do tego kolejki priorytetowej. Wiadomo, że z najniższego miejsca nie warto usuwać piasku, więc można od razu zaktualizować sąsiednie pozycje i wyrzucić minimum z kolejki. Po wykonaniu tej operacji n razy teren zostanie wygładzony. Implementacje tego pomysłu można znaleźć w plikach `stus12.cpp`, `stus13.pas`, `stus14.cpp` i `stus15.pas`. Rozwiązania o takiej złożoności mogły zdobyć 60 punktów lub więcej, w zależności od jakości implementacji.

Rozwiązania działające co najmniej liniowo względem m lub $\max x_i$ (pliki `stus2.cpp` – `stus7.pas`) nie miały większych szans na osiągnięcie wyniku powyżej 30 punktów. Należały do nich w szczególności programy niekorzystające z wyszukiwania binarnego.

---

[1] Opis tej struktury danych można znaleźć np. w opracowaniu zadania *Tetris 3D* z XIII Olimpiady Informatycznej [13] czy opracowaniu zadania *Koleje* z IX Olimpiady Informatycznej [9].

Testy

Do tworzenia testów używane były funkcje generujące losowe ciągi liczb: rosnące, malejące i dowolne. W tabeli „dołek” oznacza ciąg liczb, który do pewnego miejsca jest malejący, a potem rosnący. Z kolei „nierówny dołek” powstaje z dołka przez podzielenie go na fragmenty równej długości i losowe poprzestawianie elementów w każdym fragmencie z osobna. Podobnie definiujemy „górkę” i „nierówną górkę”.

Nazwa	**n**	**m**	**Opis**
stu1.in	71	236	mały dołek z nierównościami na środku
stu2a.in	237	821	mały dołek z nierównościami na środku otoczony spadami
stu2b.in	842	681 835	niewielki losowy test, odpowiedź 0
stu3.in	2 042	4 423	niewielki nierówny dołek
stu4a.in	5 000	2 521	trzy niewielkie dołki, skrajne są nierówne
stu4b.in	10 000	5	małe liczby prócz jednej na środku
stu4c.in	10 000	10^{12}	duża liczba dołków, z których jeden jest szerszy i płytszy
stu5a.in	10 000	500 000	dołek otoczony losowymi wartościami
stu5b.in	10 000	$\approx 10^{14}$	dosyć duży losowy test, odpowiedź 0
stu5c.in	10 000	10^{11}	duża liczba losowych odcinków, z których jeden jest szerszy i średnio niższy
stu6a.in	486 000	2 414 746 423	54 średnie dołki
stu6b.in	75 000	811 178 223	dołek z nierównościami na środku
stu7a.in	100 000	828	test z jedną poprawną odpowiedzią
stu7b.in	140 000	411 651 544	dołek z nierównościami na środku
stu7c.in	450 000	898 889	duży losowy test
stu8a.in	150 000	2	test z jedną poprawną odpowiedzią
stu8b.in	90 000	1 059 865 233	dołek z nierównościami na środku
stu9a.in	400 000	352 247	duży dołek z nierównościami na środku
stu9b.in	550 000	$\approx 3 \cdot 10^{14}$	duży dołek z nierównościami na środku
stu10a.in	500 000	10^{18}	duża nierówna górka, odpowiedź 0
stu10b.in	700 000	$\approx 10^{12}$	duży dołek z nierównościami na środku
stu11a.in	1 000 000	15 486 352 148	150 nierównych dołków
stu11b.in	1 000 000	500 000 000	maksymalny dołek
stu11c.in	1 000 000	$\approx 5 \cdot 10^{12}$	duży dołek z nierównościami na środku
stu12a.in	1 000 000	$\approx 5 \cdot 10^{11}$	dołek z nierównościami na środku
stu12b.in	750 000	2	test z jedną poprawną odpowiedzią

Zawody II stopnia

opracowania zadań