Tymoteusz Perka 300243
Jan Wasilewski 283649

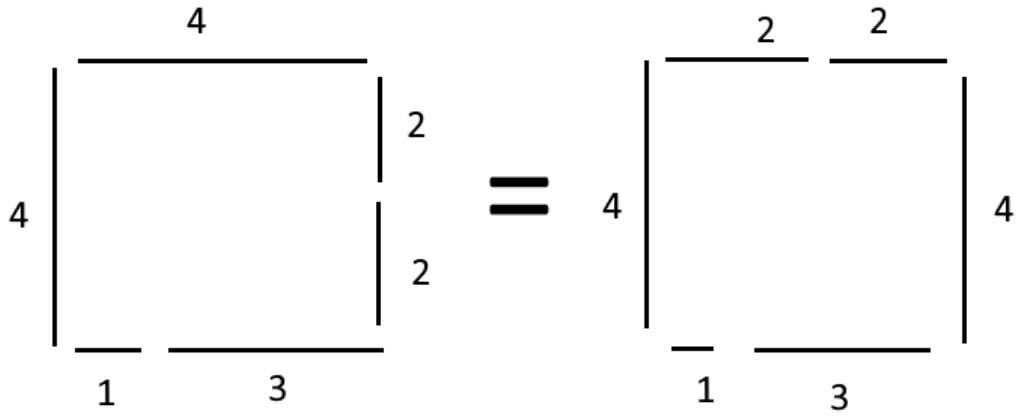
Analiza Algorytmów

Treść zadania

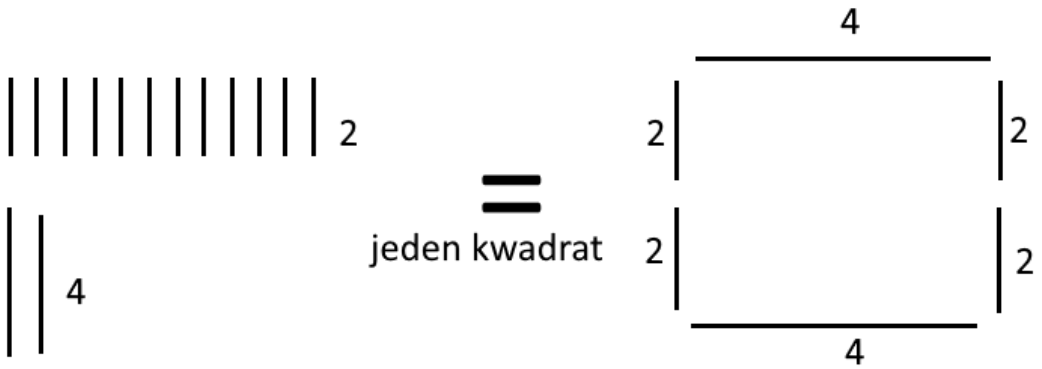
Mamy zestaw S niełamalnych patyków o długości s_i, $i \in (1, 2, ..., S)$. Zaproponuj algorytm wyliczający na ile sposobów można zbudować kwadrat przy użyciu sześciu z tych patyków i wyznaczy, które patyki należy użyć.

Interpretacja

Kolejność patyków w kwadracie nie ma znaczenia.



Kwadrat jest jednoznacznie identyfikowany jedynie przez długości patyków, z których się składa.



Rozwiązanie naiwne

Najprostszym rozwiązaniem byłoby utworzenie wszystkich możliwych 6 elementowych podzbiorów S, a następnie sprawdzenie które z nich mają szansę złożyć kwadrat. W związku z tym asymptotyczna złożoność jest klasy $O(n^6)$, gdyż chcąc utworzyć każdą możliwą szóstkę, musimy zagnieździć pętle 6 razy. Stwierdzenie, czy dany podzbiór utworzy kwadrat jest realizowane w czasie stałym, bo rozmiar każdego podzbioru jest równy 6. Utworzenie wszystkich możliwych podzbiorów jest bardziej kosztowne od sprawdzenia utworzonych podzbiorów, ponieważ:

$$\binom{n}{6} < n^6 \quad \forall n > 0$$

Koncepcja rozwiązania

Na początku działania algorytmu zbiór patyków zostaje posortowany rosnąco według długości.

Zauważmy, że aby możliwe było utworzenie kwadratu z 6 patyków, co najmniej dwa (a co najwyżej trzy) boki kwadratu muszą się składać z jednego patyka, co oznacza, że potrzebujemy, aby zbiór długości naszych patyków miał elementy powtarzające się. W tym układzie mamy dwa przypadki do rozpatrzenia:

1) Dwa boki składają się z patyków o długości X, a dwa pozostałe z dwóch patyków każdy, oba sumują się do X.

2) Trzy boki składają się z patyków o długości X, a czwarty składa się z trzech patyków o łącznej długości X.

W dalszej części opisu używamy terminów “boki samodzielne” i “boki kompozytowe”.

bok samodzielny - bok złożony z 1 patyka
bok kompozytowy - bok złożony z 2 lub 3 patyków.

Algorytm szuka kandydatów na boki samodzielne “od prawej”, czyli od największego (zbiór został posortowany). Jeżeli znajdzie co najmniej 2 kandydatów (patyki równej długości), przeszuka też zbiór “od lewej”, czyli od najmniejszego, szukając pasujących boków kompozytowych (dwupatykowych). Następnie, jeżeli liczba znalezionych kandydatów na bok samodzielny to co najmniej 3, to przeszuka zbiór “od lewej” szukając pasujących boków kompozytowych (trójpatykowych).

1. posortuj zbiór
2. szukaj kandydatów na boki samodzielne
   2.1. szukaj boków kompozytowych
3. wróć do punktu 2 aż skończysz szukanie

Działanie usprawnią warunki zapobiegające niepotrzebnym operacjom:

- jeżeli zbiór S ma mniej niż 6 elementów, to nie ma żadnych możliwych kwadratów

- jeżeli najkrótszy patyk w zbiorze jest dłuższy niż ⅓ obecnie rozpatrywanego kandydata na bok samodzielny, nie da się już utworzyć więcej kwadratów z 3 bokami samodzielnymi.

- jeżeli najkrótszy patyk w zbiorze jest dłuższy niż ½ obecnie rozpatrywanego kandydata na bok samodzielny, nie da się już utworzyć więcej kwadratów z 2 bokami samodzielnymi.

- jeżeli najkrótszy patyk spośród patyków pozostałych do sprawdzenia możliwości boku kompozytowego(3) jest dłuższy niż ⅓ kandydata na bok samodzielny, nie da się już utworzyć więcej boków kompozytowych dla tego kandydata.

- jeżeli najkrótszy patyk spośród patyków pozostałych do sprawdzenia możliwości boku kompozytowego(2) jest dłuższy niż ½ kandydata na bok samodzielny, nie da się już utworzyć więcej boków kompozytowych dla tego kandydata.

Ocena złożoności

Na podstawie operacji wykonywanych w opisanym wyżej algorytmie szacujemy teoretyczną złożoność pesymistyczną rzędu $O(n^4)$.

W praktyce z powodu usprawnień opisanych w dokumencie spodziewamy się przynajmniej trochę lepszej wydajności (jednak nadal rzędu n^4), z wyjątkiem dla danych specjalnie spreparowanych.

Przykładowo dla N patyków o długościach rozłożonych w sposób w przybliżeniu jednostajny(i w większości pozwalających zbudować kwadraty, z uwzględnieniem usprawnień):

$N * (N/2)^2$ + (N po 2)[symbol newtona] + $N * (N/3)^3$

Pierwsza połowa równania to szukanie kwadratów z 2 bokami kompozytowymi. Druga połowa to szukanie kwadratów z 1 bokiem kompozytowym.

Dane wejściowe i wyjściowe

Dane do programu można przekazać ze standardowego wejścia lub z pliku tekstowego (za pomocą przekierowania wejścia). Wyniki zostają wypisane na standardowe wyjście. Istnieje także możliwość losowej generacji danych wejściowych.

Format danych wejściowych

- n - dodatnia liczba całkowita oznaczająca liczbę patyków
- s_1, s_2, ..., s_n - dodatnie liczby całkowite oddzielone spacjami, oznaczające długości kolejnych patyków

Format danych wyjściowych

- m - liczba całkowita oznaczająca liczbę sposobów na ile można ułożyć kwadrat z danego zbioru patyków.
- W kolejnych m wierszach będą wymienione oddzielone spacjami liczby całkowite będące długościami patyków, z których został skonstruowany kwadrat.

Kompilacja i uruchamianie

Program nasz nie wymaga żadnych dodatkowych bibliotek poza standardową biblioteką C++. Kompilacja następuję automatycznie za pomocą narzędzia make. W celu skompilowania należy z poziomu konsoli wejść do folderu zawierającego projekt i wykonać polecenie:

make

W celu wyczyszczenia efektów kompilacji należy wykonać:

make clean

Stworzony przez nas program posiada trzy główne tryby uruchamiania:

1 Tryb manualny, służący do ręcznego wprowadzania danych za pomocą standardowego wyjścia, opcjonalnie pliku tekstowego. Wyniki można także przekierować do pliku wyjściowego.

./runner.o -m 1 [< input.txt] [> output.txt]

2. Tryb automatycznej generacji danych i rozwiązania problemu.

./runner.o -m 2 [-n liczba patyków] [-r maksymalna długość patyka] [-d trudność generowanych danych (może być 'hard' lub 'normal')]

3. Tryb automatycznego testowania:

./runner.o -m 3 [-n początkowa liczba patyków] [-s inkrementacja pomiędzy testami] [-k liczba testów] [-r liczba powtórzeń testu] [-f nazwa pliku]

Tryb automatycznego testowania ma ustaloną maksymalną długość patyka na n/4, aby było możliwie dużo potencjalnych kwadratów.

Przykład

Wejście:
10
1 2 4 2 5 2 5 5 2 4

Wyjście:
2
5 5 5 2 2 1
4 4 2 2 2 2

Chcąc utworzyć kwadrat zaczynamy od sprawdzenia najdłuższych patyków, dla dwóch 5 jesteśmy w stanie skleić jeszcze bok z 4 i 1, ale z żadnych pozostałych dwóch patyków nie jesteśmy w stanie otrzymać boku długości 5. Jesteśmy w stanie za to dołożyć trzeci patyk o długości 5, a ostatni bok otrzymać z patyków o długości 2, 2 oraz 1. Następnie próbujemy skonstruować dwa "całe" boki z patyków o długości 4, a następnie próbujemy skleić trzeci bok, co udaje się z pomocą dwóch patyków o długości 2. W ten sam sposób można skonstruować czwarty bok, otrzymując drugi kwadrat. Następnie możemy spróbować jeszcze złożyć trzy "całe" boki z patyków o długości 2, ale 2 jest mniejsze niż trzykrotność długości najmniejszego patyka, co oznacza, że nie skonstruujemy w ten sposób kwadratu. W ten sposób wyczerpaliśmy wszystkie opcje.

Testowanie

Przewidujemy dwa rodzaje danych wejściowych do testowania:

1) zbiór **n** patyków o losowych długościach z zakresu od **1** do **k**
2) zbiór **n = 4*i** patyków o długościach losowanych ze zbioru **(1, 2, 3, 4, … n-1, n)**, każda długość patyka występuje **4** razy.

przypadek **2** to dane "trudne" - dla takich danych kandydatom na boki samodzielne można dopasować maksymalnie dużo kombinacji boków kompozytowych.

Wyniki testów oraz wnioski

Wykonaliśmy dwa testy, różniące się trudnością generowanych danych. Na podstawie trybu testowania nr 1 mogliśmy sprawdzić nasze oszacowanie typowej złożoności, a na podstawie trybu testowania nr 2 mogliśmy zobaczyć, jak dobrze oszacowaliśmy pesymistyczną złożoność naszego algorytmu. Zrobiliśmy po 30 testów, każdy powtórzony 10 razy, za t(n) wzięliśmy średnią z przebiegów danego przypadku testowego. Za krok przyjęliśmy 20, a za początkowy rozmiar problemu 300. Wyniki testów razem z obliczonym q(n) znajdują się w poniższych tabelach:

Tabela 1. Przypadek normalny

n	t(n)	q(n)
300	64,5	0,8722846759
320	113	1,180489361
340	117,7	0,9648156554
360	151,7	0,9893708827
380	205,7	1,080646005
400	267,3	1,143780112
420	275,8	0,9709137407
440	331,1	0,9676805723
460	377	0,9223453671
480	500,8	1,03343441
500	596,6	1,045651052
520	698	1,045744515
540	748	0,9636295421
560	853	0,9501243979
580	951,9	0,9214311707
600	1183,1	1
620	1416,4	1,050032418
640	1512,5	0,9875498664
660	1716,4	0,9908919743
680	2011	1,030291144
700	2266,4	1,034017642
720	2627,3	1,070935284
740	2695,1	0,9845358897
760	2978,1	0,9778414768
780	3227,7	0,9552087645
800	3629,6	0,9706940453
820	3883,8	0,9409899716
840	4687,1	1,031265816
860	4913,4	0,9839473575
880	5331,6	0,9738911468
900	5628	0,9396532024

Tabela 2. Przypadek trudny:

n	t(n)	q(n)

300	61,1	0,9357710347
320	121,3	1,435073994
340	85,1	0,7900003774
360	117,8	0,8700589808
380	146,6	0,872193791
400	132,1	0,6401419068
420	190	0,7574776902
440	253,6	0,8393671146
460	319,6	0,8855005815
480	262,3	0,6129806254
500	386,9	0,7679485403
520	535,3	0,9082329959
540	624	0,910380522
560	635,9	0,8021399079
580	756,7	0,8295167626
600	1044,7	1
620	939,2	0,7885055105
640	1134,3	0,8387285818
660	1212,7	0,7928500214
680	1654,2	0,9597669097
700	1952,5	1,008816761
720	2012	0,9287768832
740	2242,3	0,9276416609
760	2675,9	0,995013372
780	2596,2	0,8701079598
800	2980,9	0,902819365
820	2897	0,7948890099
840	3292,2	0,820318438
860	3831,2	0,8688692469
880	4175	0,8636528252
900	4460,5	0,84338702

Jak widać, nie istnieje żaden trend w wartości q(n) w żadnym przypadku. Wartości oscylują wokół jedynki w przypadku normalnych danych oraz nieco poniżej jedynki dla danych trudnych, jednak biorąc za normalizujący wiersz np. n = 580 już

uzyskamy dużo bardziej zbliżone do 1 wyniki. Potwierdza to nasze przypuszczenia o złożoności algorytmu rzędu O(n^4).

Zaskakujący może wydawać fakt, że dla teoretycznie trudnych danych algorytm uzyskuje lepsze wyniki, niż dla generacji normalnej. Wynika to z dodanych licznych optymalizacji pod przypadki pesymistyczne w kodzie algorytmu.