

2023-2024

Soluzioni alla 1° gara





Syllabus

Syllabus

Olimpiadi di Informatica a Squadre

Versione 2 --- 19 maggio 2019

Livello 1.

- Tipi di dato primitivo (e.g. int, char, double, bool)
- Array mono-dimensionali (e.g. int[], char[], double[], bool[])
- Branching (costrutti if/else)
- Cicli limitati (costrutti for semplici)

Livello 2.

- Array multi-dimensionali (e.g. int[][])
- Strutture dati coda e pila
- Cicli illimitati (costrutto while)
- Funzioni e ricorsione esaustiva (e.g. elencare le permutazioni)
- Ricerca binaria
- Algoritmi di ordinamento quadratici (e.g. bubble sort)
- Stringhe e ricerca quadratica di una sottostringa in una stringa
- Algoritmo di Euclide per il massimo comun divisore
- Concetti base di geometria e aritmetica



Syllabus

Livello 3.

- Contenitori standard (e.g. vector, set, map)
- Divide et impera
- Programmazione dinamica su array mono- o multi-dimensionali
- Visite di grafi (BFS, DFS)
- Algoritmi di ordinamento efficienti (sort o qsort)
- Ottimizzazione approssimata tramite tecniche euristiche

Livello 4.

- Struttura dati union-find per insiemi disgiunti
- Strutture dati per query su range
- Minimo antenato comune su alberi
- Algoritmi su grafi: cammini minimi, albero ricoprente
- Programmazione dinamica su grafi aciclici
- Backtracking (algoritmi branch and bound)

Livello 5.

- Tutti gli argomenti non menzionati nei livelli precedenti



01

Soluzioni Gara 1



L1: Excellent

Excellent Numbers (excellent)

Valerio was recently introduced to the concept of excellent numbers: a positive integer is considered *excellent* if its decimal representation only contains the digits 1 and 5, and it is divisible by 3.

For example, **15** and **111** are *excellent* numbers ($15 = 5 \cdot 3 + 0$ and $111 = 37 \cdot 3 + 0$), while **151** is not ($151 = 50 \cdot 3 + 1$).



Figure 1: 1515 is considered by many an *Angel Number*¹ and also happens to be an *excellent* number!

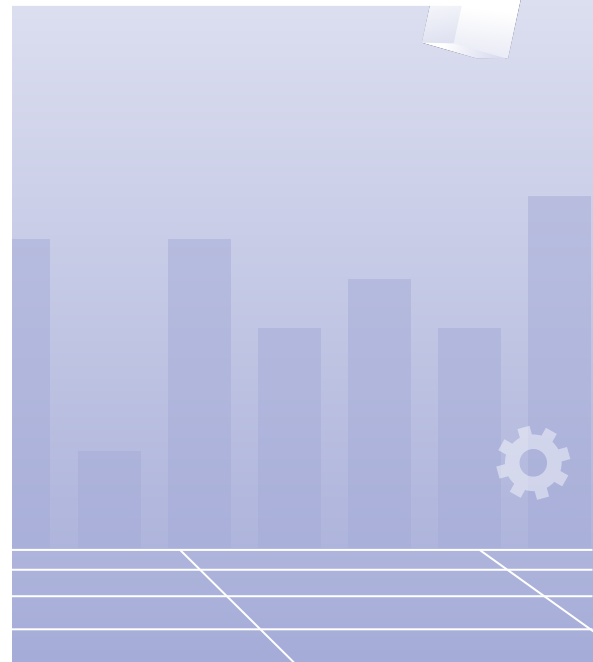
Valerio is wondering if there exists at least one *excellent* number with exactly N digits. Help him by **finding one**, or by determining that there are no *excellent* numbers with that number of digits!

Esempio:

Input: 2

Possibile output (1): 15

Possibile output (2): 51





L1: Excellent

Approccio 1: Provo con tutti i numeri di N cifre e vedo se sono eccellenti... un po' brutto...



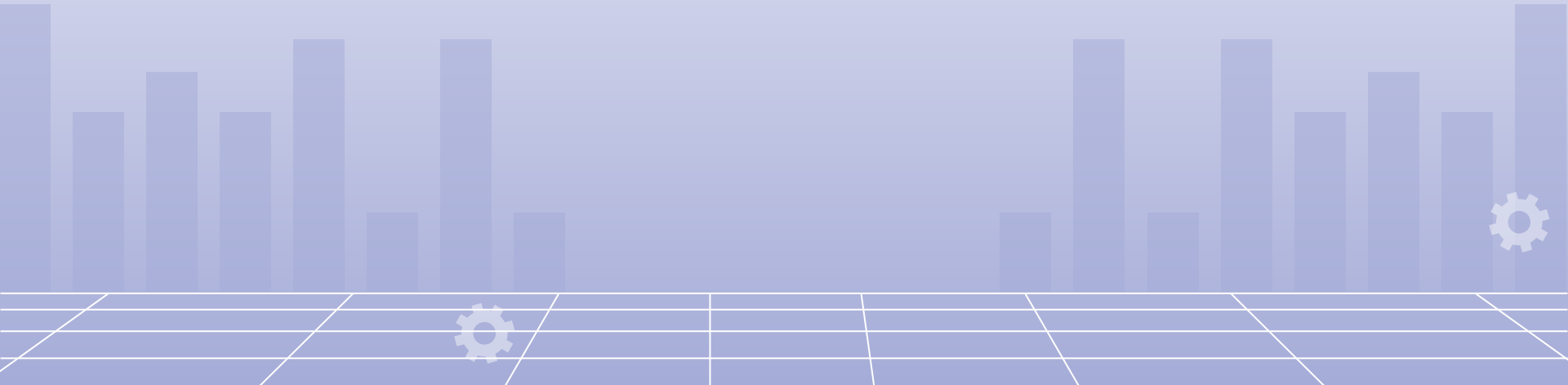
Approccio 2: Genero tutti i numeri contenenti 1 e 5 di N cifre e poi vedo se sono eccellenti... troppo complicato...





L1: Excellent

Aspetta... quand'è che un numero è divisibile per 3?





L1: Excellent

Aspetta... quand'è che un numero è divisibile per 3?



Se la somma delle cifre è multipla di 3!



L1: Excellent

N è pari

Spammiamo un multiplo di 3 di 2 cifre (e contenente solo 1 e 5), tanto la somma sarà comunque multipla di 3

N = 4 -> 1515
N = 6 -> 151515

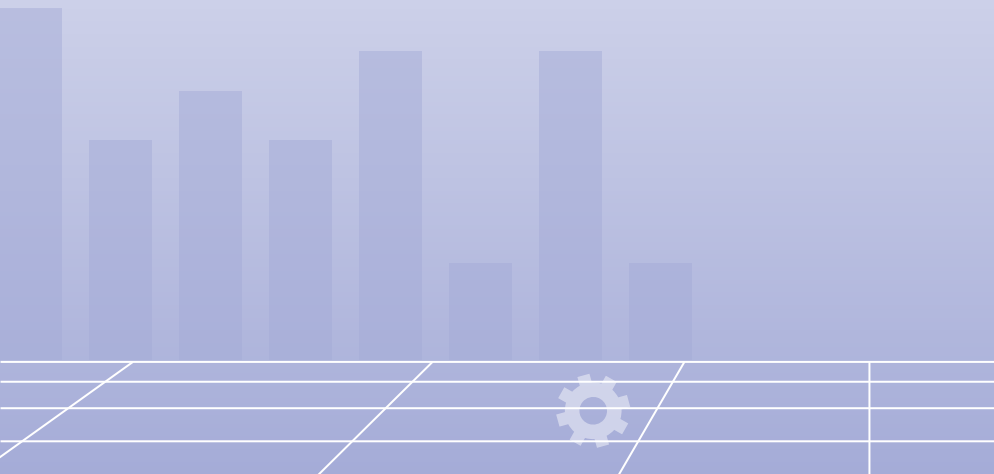
N è dispari

Attacchiamo prima un numero di 3 cifre multiplo di 3 (e contenente solo 1 e 5) e poi spammiamo un multiplo di 3 di 2 cifre finchè riusciamo

N = 3 -> 11115
N = 7 -> 1111515



L1: Excellent



Nota: // è la divisione con risultato intero, quindi arrotonda automaticamente per difetto

input data

```
N = int(input().strip())
```

I numeri da 1 a 9 non sono eccellenti

```
if N<2:
```

```
    print('-1')
```

```
else:
```

Se N è pari possiamo spammare 15 (o 51), visto che la somma dei numeri sarà sempre multipla di 6 e quindi divisibile per 3 :)

```
    if N%2==0:
```

```
        print('15'*(N//2))
```

Se N è dispari anteponiamo 111 (o 555) e poi spammiamo 15, visto che la somma dei numeri sarà sempre divisibile per 3 :)

```
    else:
```

```
        print('111'+ '15'*((N-3)//2))
```



L1: Ping pong

| Input: | Possible Output: |
|--------|------------------|
| 5 | 11 10 |
| 33 15 | 11 5 |
| 40 29 | 11 0 |
| 55 55 | 11 10 |
| 39 16 | 11 8 |
| 29 54 | 7 11 |
| | 11 0 |
| | -1 -1 |
| | 11 5 |
| | 11 0 |
| | 6 11 |
| | 11 0 |
| | -1 -1 |

Ping-pong (pingpong)

You are attending a table tennis tournament and you decided to note down the results the players achieved. However, a bad misfortune happened and you lost your detailed list of results so you don't know the way the games went anymore.

There is good news though: In order to avoid a complete loss of data, you also noted how many points each player has scored in total during all the sets they played. In addition, you also know that the first player always won in the end.



Figure 1: The games were already played. Can you reconstruct the results?

Now your goal is to reconstruct the way T games turned out, if you know the following details:

- The games have been played using normal table tennis rules, but with a key difference. The set always ends when a player reaches 11 points (even when the score is 11-10, so no tiebreaks exist).
- The match is played using a best of 5 system (the first player to win 3 sets wins the game).
- The first player always won in the end.

Since the games can turn out in many different ways, any such sequence of set scores is acceptable.



L1: Ping pong

Siccome il gioco è al meglio di 5, e il primo giocatore a vincere 3 set vince (tenendo conto che vince sempre quello a sinistra nell'input), possono accadere 3 scenari:



- 1) **Si svolgono 3 match:** il giocatore 1 ha vinto 3 volte e perso 0 volte
- 2) **Si svolgono 4 match:** il giocatore 1 ha vinto 3 volte e perso 1 volta
- 3) **Si svolgono 5 match:** il giocatore 1 ha vinto 3 volte e perso 2 volte





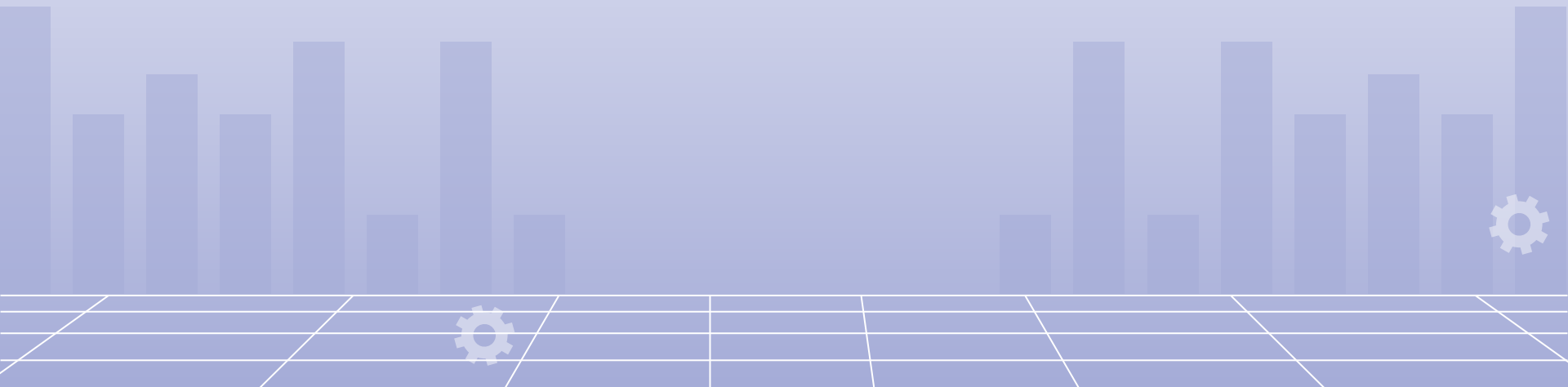
L1: Ping pong

1) Si svolgono 3 match:

Quanti punti deve fare il giocatore 1 se vince i primi 3 match?



Quanti punti può fare al massimo il giocatore 2 visto che perde i primi 3 match?





L1: Ping pong

1) Si svolgono 3 match:

Quanti punti deve fare il giocatore 1 se vince i primi 3 match? **33**



Quanti punti può fare al massimo il giocatore 2 visto che perde i primi 3 match? **30**



Allora il G1 deve vincere tutti e 3 i match facendo 11 punti per ciascuno, mentre G2 può fare quanto vuole (meno di 11)

Per esempio, dato il risultato 33 30, un possibile output è:

11 10

11 10

11 10





L1: Ping pong

1) Si svolgono 3 match:

Facciamo fare a G2 $\frac{1}{3}$ dei suoi punti totali ad ogni match, per semplicità!

Tradotto in codice:

```
# 3 match: G1 perde 0 partite
if a==33 and b≤30:
    print(11, b // 3)
    print(11, b // 3)
    print(11, b - (b // 3) * 2)
```

Nel caso G1 o G2 facciamo più punti, 3 match non bastano più!



L1: Ping pong

2) Si svolgono 4 match:

Quanti punti può fare il giocatore 1 se vince 3 match e ne perde 1?

Quanti punti può fare il giocatore 2 visto che perde 3 match e ne vince 1?





L1: Ping pong

2) Si svolgono 4 match:

Quanti punti può fare il giocatore 1 se vince 3 match e ne perde 1?

Minimo **33** (se ne fa meno non ne vince più 3 match)

Massimo **43** (se ne fa di più vincerebbe più di 3 match)

Quanti punti può fare il giocatore 2 visto che perde 3 match e ne vince 1?

Minimo **11** (deve vincere almeno un match)

Massimo **41** (altrimenti vincerebbe più di un match)



L1: Ping pong

2) Si svolgono 4 match:

Facciamo fare a G2 una partita (vanno via 11 punti) e poi usiamo $\frac{1}{3}$ dei suoi punti totali rimanenti per i successivi match, per semplicità!

A G1 servono 33 punti per vincere 3 match, quindi usiamo i rimanenti nel match perso.

Tradotto in codice:

```
elif 33 ≤ a ≤ 43 and 11 ≤ b ≤ 41:  
    print(a-33, 11)  
    b-=11  
    print(11, b//3)  
    print(11, b//3)  
    print(11, b - (b // 3) * 2)
```

Nel caso G1 o G2 facciamo più punti, 4 match non bastano più!



L1: Ping pong

3) Si svolgono 5 match:

Quanti punti può fare il giocatore 1 se vince 3 match e ne perde 2?

Quanti punti può fare il giocatore 2 visto che perde 3 match e ne vince 2?





L1: Ping pong

3) Si svolgono 5 match:

Quanti punti può fare il giocatore 1 se vince 3 match e ne perde 2?

Minimo **33** (se ne fa meno non ne vince più 3 match)

Massimo **53** (se ne fa di più vincerebbe più di 3 match)

Quanti punti può fare il giocatore 2 visto che perde 3 match e ne vince 2?

Minimo **22** (deve vincere almeno 2 match)

Massimo **52** (altrimenti vincerebbe più di 2 match)



L1: Ping pong

3) Si svolgono 5 match:

Facciamo fare a G2 due partite (vanno via 22 punti) e poi usiamo $\frac{1}{3}$ dei suoi punti totali rimanenti per i successivi match, per semplicità!

A G1 servono 33 punti per vincere 3 match, quindi usiamo i rimanenti nel match perso.

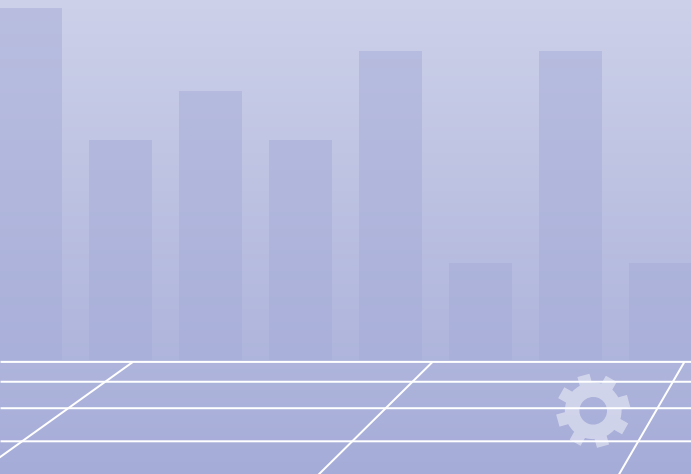
Tradotto in codice:

```
elif 33 ≤ a ≤ 53 and 22 ≤ b ≤ 52:
    # Tanto gli facciamo vincere 3 partite dopo
    a -= 33
    print(a // 2, 11)
    print(a - a // 2, 11)
    # G2 ha già vinto 2 match
    b -= 22
    print(11, b // 3)
    print(11, b // 3)
    print(11, b - (b // 3) * 2)
```

Nel caso G1 o G2 facciamo più punti o addirittura meno, stampiamo -1 -1



L1: Ping Ping



```
def solv(a,b):  
    # 3 match: G1 perde 0 partite  
    if a==33 and b<=30:  
        print(11,b//3)  
        print(11,b//3)  
        print(11,b - (b // 3) * 2)  
    # 4 match: G1 perde 1 partita  
    elif 33<=a<=43 and 11<=b<=41:  
        print(a-33,11)  
        b-=11  
        print(11,b//3)  
        print(11,b//3)  
        print(11,b - (b // 3) * 2)  
    # 5 match: G1 perde 2 partite  
    elif 33<=a<=53 and 22<=b<=52:  
        # Tanto gli facciamo vincere 3 partite dopo  
        a-=33  
        print(a//2, 11)  
        print(a - a//2, 11)  
        # G2 ha già vinto 2 match  
        b-=22  
        print(11, b//3)  
        print(11, b//3)  
        print(11, b - (b // 3) * 2)  
    else:  
        print('-1 -1')  
  
t = int(input().strip())  
for i in range(t):  
    a,b = list(map(int, input().strip().split()))  
    solv(a,b)
```



L5:Area Under Path

Input:
2 2 3 1

Possible Output:
2

Area Under Path (areaunderpath)

Peter is standing at the origin of the Cartesian plane, and he decided to take a *regular* walk to point (N, M) for some positive integers N and M . In each step of a *regular* walk, Peter must move parallel to one of the axes. During a move, he can go either one unit to the right or one unit up.



Figure 1: Peter on his morning walk on the Cartesian plane.

Formally, a regular walk from $(0, 0)$ to (N, M) is a sequence of points (x_i, y_i) ($0 \leq i \leq N + M$) such that

- $(x_0, y_0) = (0, 0)$ and $(x_{N+M}, y_{N+M}) = (N, M)$, and
- for each $i = 1, \dots, N + M$, either $(x_i, y_i) = (x_{i-1} + 1, y_{i-1})$ or $(x_i, y_i) = (x_{i-1}, y_{i-1} + 1)$.

We define the area under a regular walk as the area of the polygon whose vertices in clockwise order are $(0, 0) = (x_0, y_0), (x_1, y_1), \dots, (x_{N+M}, y_{N+M}) = (N, M)$ and $(N, 0)$.

Given a prime number P and a remainder R , you have to find the number W of *regular* walks from $(0, 0)$ to (N, M) under which the area is congruent to R modulo P . Since the answer can be very large, you have to compute it modulo $10^9 + 7$.



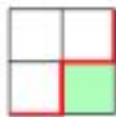
L5: Area Under Path

Explanation

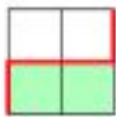
In the **first sample case**, there are six possible *regular* walks from $(0,0)$ to (N,M) , as shown in the figure below.



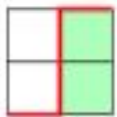
area=0



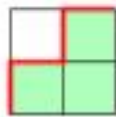
area=1



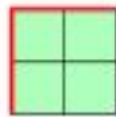
area=2



area=2



area=3



area=4

The area under the second and the sixth paths are 1 and 4 respectively, both of which give a remainder of 1 when divided by 3.



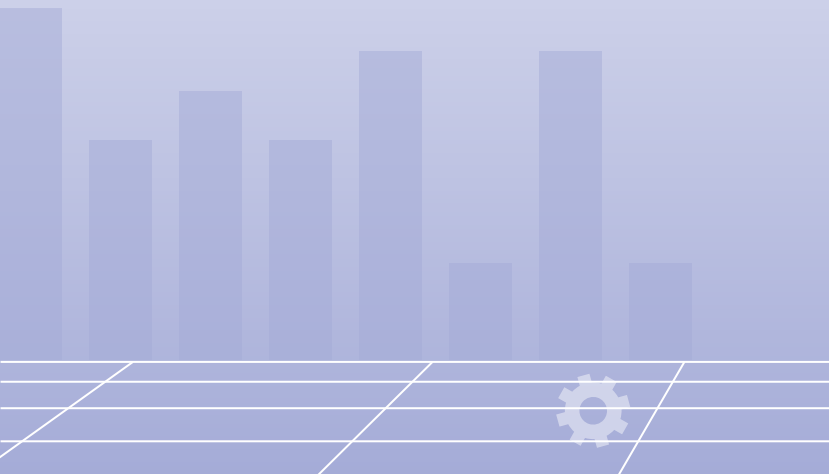


L5:Area Under Path



Un bel 16/100 su

https://training.olinfo.it/#/task/ois_area_under_path, non male!



```
MOD = 10**9 + 7
```

```
n, m, p, r = map(int, input().split())
```

```
result = 0
```

```
def print_all_paths_with_area(x, y, current_area = 0):
```

```
    global result
```

```
    # Caso base: raggiunto l'angolo in alto a destra
```

```
    if x == n or y == m:
```

```
        #print(str(current_area))
```

```
        if current_area % p == r:
```

```
            result = ((result + 1) % MOD)
```

```
        return
```

```
    # Muoviti verso destra se possibile
```

```
    if x < n:
```

```
        print_all_paths_with_area(x + 1, y, current_area)
```

```
    # Muoviti verso l'alto se possibile
```

```
    if y < m:
```

```
        print_all_paths_with_area(x, y + 1, current_area + n - x)
```

```
def solv():
```

```
    print_all_paths_with_area(0, 0)
```

```
solv()
```

```
print(result)
```



L2: barbarian



barbarian.py ×

barbarian.py

```
1  #!/usr/bin/env python3
2  # @check-accepted: examples Nsmall N1000 N10000
3  # @check-time-limit-exceeded: no-limits
4  # insert brief description of the solution here
5
6
7  # input data
8  n = int(input().strip())
9  v = list(map(int, input().strip().split()))
10
11  if n == 1:
12      print(0)
13      exit
14
15  step = [0 for i in range(n)]
16  step[n - 1] = n - 1
17
18  stack = [0]
19
20  for i in range(1, n-1):
21      if v[i] - v[i-1] <= v[i+1] - v[i]:
22          while stack[-1] > 0 and 2*v[stack[-1]] - v[stack[-1]-1] <= v[i+1]:
23              stack.pop()
24              step[i] = stack[-1]
25          stack.append(i)
26
27  stack = [n-1]
28  for i in range(n-2, 0, -1):
29      if v[i+1] - 2*v[i] + v[i-1] < 0:
30          while stack[-1] < n-1 and v[stack[-1] + 1] - 2*v[stack[-1]] + v[i-1] < 0:
31              stack.pop()
32              step[i] = stack[-1]
33          stack.append(i)
34
35  def find(i):
36      if step[i] == i:
37          return i
38      step[i] = find(step[i])
39      return step[i]
40
41  for i in range(n):
42      print (n-1-find(i), end = ' ')
43  print()
44
```



L3: avg



| | |
|-------------------------|--------------|
| Input: 2 3 10 6 | Output: 2 |
| Input: 3 9 2 10 1 | Output: 1 |

Precise Average (avg)

Your friend John works at a shop that sells N types of products, numbered from 0 to $N - 1$. Product i has a price of P_i bytedollars, where P_i is a positive integer.

The government of Byteland has created a new law for shops. The law states that the average of the prices of all products in a shop should be exactly K , where K is a positive integer. John's boss gave him the task to change the prices of the products so the shop would comply with the new law.



Figure 1: It takes a lot of time to keep the shop tidy! ¹

He has lots of other stuff to do, so he asked you for help: what is the minimum number of products whose prices should be changed? A product's price can be changed to any **positive** integer amount of bytedollars.

Among the attachments of this task you may find a template file `avg.*` with a sample incomplete implementation.

Input

The first line of the input contains two integers N and K . The next line contains N positive integers, P_0, \dots, P_{N-1} .

Output


Output a single integer between 0 and N , inclusive: the answer to the question. It can be proven that it is always possible to change the prices of some products so that the new prices comply with the law.





L3: avg




- ★ Se la media è già a posto possiamo stampare direttamente 0
- ★ Se la media dei prodotti è minore possiamo aumentare il prezzo del primo prodotto potenzialmente all'infinito, fino a farla coincidere 
- ★ Altrimenti, parte l'algoritmo vero e proprio!
Per modificare il prezzo di meno prodotti possibili è logico pensare che devo abbassare la media del più possibile, usando quindi i prodotti più costosi (più il numero è alto più velocemente posso abbassare la media. Stiamo facendo quindi una scelta greedy (ingorda)

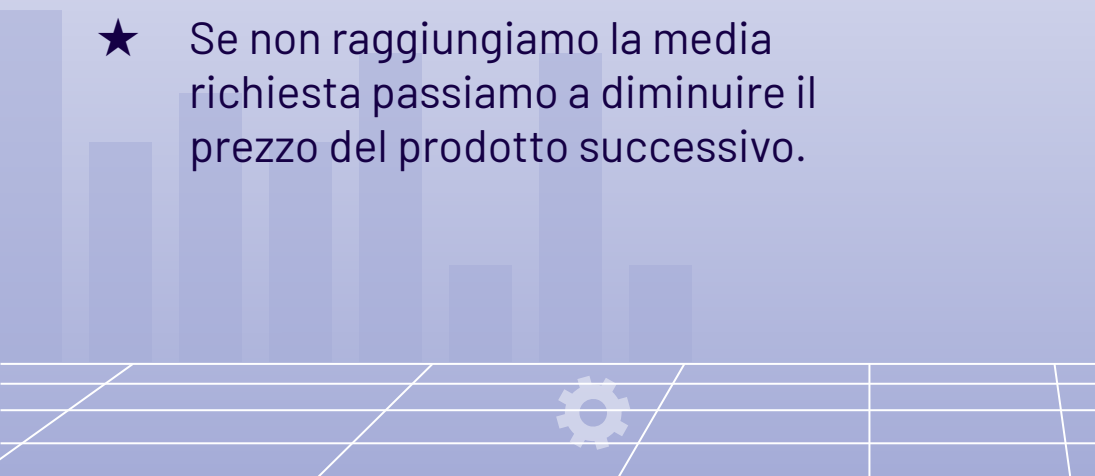




L3: avg



★ Ordiniamo quindi in ordine decrescente e, partendo appunto dai prodotti più costosi, iniziamo a diminuire il prezzo più che possiamo (lasciando minimo il prezzo a 1€).



★ Se non raggiungiamo la media richiesta passiamo a diminuire il prezzo del prodotto successivo.

```
n, k = map(int, input().split())
p = list(map(int, input().split()))
somma = sum(p)
# se la media è già uguale non dobbiamo modificare
prodotti
if somma == n * k:
    print(0)
# se la media è minore possiamo aumentare il prezzo del
primo prodotto all'infinito
elif somma < n * k:
    print(1)
# se la media è maggiore dobbiamo togliere prodotti
else:
    # ordiniamo in ordine decrescente
    p = sorted(p, reverse=True)
    ris = 0
    rimanente = somma - n * k
    # iniziamo a ridurre il più possibile, partendo dai
prodotti più costosi
    for i in range(n):
        da_togliere = min(p[i] - 1, rimanente)
        if da_togliere > 0:
            ris += 1
            rimanente -= da_togliere

    print(ris)
```



L3: brackets2



★ Testo del problema



★ Soluzione





L4: binarychess

binarychess.py

```
1  #!/usr/bin/env python3
2  from sys import stdin, stdout
3
4  def rd(): return stdin.readline().strip()
5  def rdl(x): return map(x, rd().split())
6  def wt(x): stdout.write(str(x))
7  def wtl(x): wt(str(x) + '\n')
8
9  R, C, n = rdl(int)
10 mp, ans = [dict() for i in range(4)], 1
11 p, ncc = [-1 for i in range(n)], n
12
13 def get(x):
14     global p
15     if p[x] < 0:
16         return x
17     p[x] = get(p[x])
18     return p[x]
19
20 def unite(x, y):
21     global p, ncc
22     x, y = get(x), get(y)
23     if x != y:
24         if p[x] < p[y]:
25             x, y = y, x
26             ncc = ncc - 1
27             p[y] += p[x]
28             p[x] = y;
29
30 for i in range(n):
31     r, c = rdl(int)
32     if r in mp[0]:
33         mp[0][r].append(i);
34     else:
35         mp[0][r] = [i]
36     if c in mp[1]:
37         mp[1][c].append(i);
38     else:
39         mp[1][c] = [i]
40     if r - c in mp[2]:
41         mp[2][r - c].append(i);
42     else:
43         mp[2][r - c] = [i]
44     if r + c in mp[3]:
45         mp[3][r + c].append(i);
46     else:
47         mp[3][r + c] = [i]
48 for vv in mp:
49     for v in vv.values():
50         for i in range(len(v) - 1):
51             unite(v[i], v[i + 1])
52 wtl(pow(2, ncc, 1000000007))
```



L5: areaunderpath



areaunderpath.py X

areaunderpath.py > ...

```
1  |#!/usr/bin/env python3
2  import sys
3
4  sys.setrecursionlimit(10**6)
5  lp=10**9+7
6  mem=[1,1,2]
7
8  def fac(n):
9      while len(mem)<n+1:
10         mem.append((mem[-1]*len(mem))%lp)
11         return mem[n]
12
13  def choose(n,k):
14      if k>n:
15         return 0
16      return (fac(n)*pow(fac(n-k),-1,lp)*pow(fac(k),-1,lp))%lp
17
18  memo={}
19
20  def dp(n,m,r,p):
21      if n==0:
22         if r==0:
23             return 1
24         else:
25             return 0
26      if m==0:
27         if r==0:
28             return 1
29         else:
30             return 0
31      if (n,m,r,p) in memo:
32         return memo[(n,m,r,p)]
33      res=dp(n,m-1,r,p)
34      res+=dp(n-1,m,(r-m)%p,p)
35      res%=lp
36      memo[(n,m,r,p)]=res
37      return res
38
39  def solv(n,m,p,r):
40      if n%p==0 and m%p==0:
41         all=choose(n+m,n)
42         err=choose((n+m)//p,n//p)
43         if r==0:
44             return ((all-err)*pow(p,-1,lp)+err)%lp
45         else:
46             return ((all-err)*pow(p,-1,lp))%lp
47
48      all=choose(n+m,n)
49      err=choose((n+m)//p,n//p)*choose(n%p+m%p,n%p)
50      res= ((all-err)*pow(p,-1,lp))%lp
51      return (res+choose((n+m)//p,n//p)*dp(n%p,m%p,r,p))%lp
52
53  n, m, p, r = map(int, input().split())
54  print(solv(n, m, p, r))
55
```