Fonotaktika

Beseda mora vsebovati n samoglasnikov - te lahko izberemo na o^n načinov. Na začetek in konec besede ter med vsaka dva zaporedna samoglasnika lahko vstavimo med 0 in k soglasnikov. Vseh različnih besed je torej $o^n \cdot (1 + p + p^2 + \cdots + p^k)^{n+1}$.

Problem

Imamo štiri vhodne parametre: `n`, `k`, `o`, `p`, in naš cilj je izračunati določen matematični izraz, ki vključuje eksponentacijo in geometrijska zaporedja z uporabo modularne aritmetike. Specifična naloga je, da za vrednost `p` uporabimo geometrijsko zaporedje, če `p` ni enak 1, in enostavno eksponentacijo, če je `p` enak 1.

Rešitev vključuje dve ključni komponenti: 1. **Modularna eksponentacija:** za hitro izračunavanje velikih eksponentov. 2. **Geometrijska zaporedja:** za izračun zaporedja, ki vključuje parameter `p`.

Ključni koraki rešitve

1. **Modularna eksponentacija:**
   Uporabimo metodo **eksponentacije s kvadriranjem**, ki omogoča izračun velikih eksponentov v času $O(\log b)$, kar je optimizirano za velike vrednosti eksponentov. Ta metoda uporablja lastnosti, da je $a^{2k} = (a^k)^2$, kar omogoča hitro računanje brez prelivanja števil.

2. **Geometrijska vsota:**
   Geometrijska vsota za `p` je izračunana s formulo:

   $$S = 1 + p + p^2 + \cdots + p^k$$

   Ta vsota se izračuna iterativno, vsak člen pa je zmanjšan modulo $10^9 + 7$, da preprečimo prelivanje števil in ohranimo rezultate v obvladljivem območju.

3. **Modularna aritmetika:**
   Ker so številke lahko zelo velike, vse operacije potekajo z uporabo **modulo operacij** z modulo $M = 10^9 + 7$, ki je pogosto uporabljen v programerskih tekmovanjih.

Python 3.10

```
MOD = 10**9 + 7

def mod_inverz(x, mod=MOD):
    """Modularna inverzija s pomočjo Fermatovega malega izreka."""
    return pow(x, mod - 2, mod)

def geometrijska_vsota(p, k, mod=MOD):
    """Izračun geometrijske vsote: (1 - p^(k+1)) / (1 - p) % mod."""
    if p == 1:
        return (k + 1) % mod
    # Izračunaj (1 - p^(k+1)) / (1 - p) % mod
    stevnik = (1 - pow(p, k + 1, mod) + mod) % mod  # 1 - p^(k+1) % mod
    imenovalec = (1 - p + mod) % mod  # 1 - p % mod
    return (stevnik * mod_inverz(imenovalec, mod)) % mod

# Preberi vhodne podatke
n, k, o, p = map(int, input().split())
```

```
# Izračunaj o^n % MOD
rezultat = pow(o, n, MOD)

# Izračunaj geometrijsko vsoto in jo dvigni na moč n + 1
geometrija = geometrijska_vsota(p, k, MOD)
geometrija_povišana = pow(geometrija, n + 1, MOD)

# Pomnoži rezultate in izračunaj rezultat modulo MOD
rezultat = (rezultat * geometrija_povišana) % MOD

# Izpiši rezultat
print(rezultat)
```

C

```
#include <stdio.h>

typedef long long ll;
ll M = 1000000007;

ll mpow(ll a, ll b) {
    ll result = 1;
    while (b) {
        if (b % 2) {
            result *= a;
            result %= M;
        }
        a *= a;
        a %= M;
        b /= 2;
    }
    return result;
}

ll geometric_sum(ll p, ll k) {
    ll result = 0;
    ll term = 1;
    for (int i = 0; i <= k; ++i) {
        result += term;
        result %= M;
        term *= p;
        term %= M;
    }
    return result;
}

int main() {
    ll n, k, o, p;
    scanf("%lld %lld %lld %lld", &n, &k, &o, &p);
    ll result = mpow(o, n);
    result = (result * mpow(geometric_sum(p, k), n + 1)) % M;
    printf("%lld", result);
    return 0;
}
```

Pascal

```
program ModularExponentiation;
```

```
var
  n, k, o, p: Int64;
  M: Int64 = 1000000007;

function mpow(a, b: Int64): Int64;
var
  result: Int64;
begin
  result := 1;
  while b > 0 do
  begin
    if b mod 2 = 1 then
      result := (result * a) mod M;
    a := (a * a) mod M;
    b := b div 2;
  end;
  mpow := result;
end;

function mod_inverse(x: Int64): Int64;
begin
  mod_inverse := mpow(x, M - 2);
end;

function geometric_sum(p, k: Int64): Int64;
var
  numerator, denominator: Int64;
begin
  if p = 1 then
  begin
    // Če je p = 1, geometrijska vsota je preprosta: (k+1)
    geometric_sum := (k + 1) mod M;
  end
  else
  begin
    // Izračunaj geometrijsko vsoto (1 - p^(k+1)) / (1 - p)
    numerator := (1 - mpow(p, k + 1) + M) mod M;  // 1 - p^(k+1) mod M
    denominator := (1 - p + M) mod M;  // 1 - p mod M
    geometric_sum := (numerator * mod_inverse(denominator)) mod M;  // Modularno deljenje
  end;
end;

begin
  ReadLn(n, k, o, p);  // Preberi vhodne podatke
  // Izračunaj rezultat: (o^n) * (geometrijska vsota(p, k))^(n+1) % M
  WriteLn((mpow(o, n) * mpow(geometric_sum(p, k), n + 1)) mod M);
end.
```

Najmanjši nadkvadrat

Imamo tri točke v dvodimenzionalnem prostoru. Naš cilj je izračunati površino najmanjšega kvadrata, ki vsebuje vse te točke. Koordinate teh točk so podane in mi moramo izračunati, kakšna bo površina kvadrata, ki vsebuje vse tri točke.

Rešitev

Vsaka rešitev temelji na naslednjih ključnih korakih: 1. **Preberemo vhodne podatke.** 2. **Izračunamo mejne vrednosti koordinat (minimalne in maksimalne x in y vrednosti).** 3. **Izračunamo dolžino stranice kvadrata.** 4. **Izračunamo in izpišemo površino kvadrata.**

Python 3.10

```
import sys
input = sys.stdin.read
data = input().splitlines()

# Preberi koordinate točk A, B in C
points = [tuple(map(int, line.split())) for line in data]

# Najdi najmanjšo in največjo x ter y koordinato
min_x = min(point[0] for point in points)
max_x = max(point[0] for point in points)
min_y = min(point[1] for point in points)
max_y = max(point[1] for point in points)

# Dolžina stranice najmanjšega kvadrata, ki pokrije vse točke
side_length = max(max_x - min_x, max_y - min_y)

# Ploščina kvadrata
area = side_length ** 2

print(area)
```

C

```
#include <stdio.h>

int main() {
    long long x[3], y[3];
    long long min_x = 1000000001, max_x = -1000000001;
    long long min_y = 1000000001, max_y = -1000000001;

    // Preberi koordinate točk A, B in C
    for (int i = 0; i < 3; i++) {
        scanf("%lld %lld", &x[i], &y[i]);
        if (x[i] < min_x) min_x = x[i];
        if (x[i] > max_x) max_x = x[i];
        if (y[i] < min_y) min_y = y[i];
        if (y[i] > max_y) max_y = y[i];
    }

    // Dolžina stranice najmanjšega kvadrata, ki pokrije vse točke
    long long side_length = (max_x - min_x > max_y - min_y) ? (max_x - min_x) : (max_y - min_y)
;
```

```
    // Ploščina kvadrata
    printf("%lld\n", side_length * side_length);

    return 0;
}
```

C++

```
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    long long x[3], y[3];
    long long min_x = 1000000001, max_x = -1000000001;
    long long min_y = 1000000001, max_y = -1000000001;

    // Preberi koordinate točk A, B in C
    for (int i = 0; i < 3; i++) {
        cin >> x[i] >> y[i];
        min_x = min(min_x, x[i]);
        max_x = max(max_x, x[i]);
        min_y = min(min_y, y[i]);
        max_y = max(max_y, y[i]);
    }

    // Dolžina stranice najmanjšega kvadrata, ki pokrije vse točke
    long long side_length = max(max_x - min_x, max_y - min_y);

    // Ploščina kvadrata
    cout << side_length * side_length << endl;

    return 0;
}
```

Pascal

```
program SmallestSquare;


Uses math;
var
  x: array[1..3] of Int64;
  y: array[1..3] of Int64;
  i: Integer;
  min_x, max_x, min_y, max_y, side_length: Int64;

begin
  min_x := 1000000001; max_x := -1000000001;
  min_y := 1000000001; max_y := -1000000001;

  for i := 1 to 3 do
  begin
    ReadLn(x[i], y[i]);
    if x[i] < min_x then min_x := x[i];
    if x[i] > max_x then max_x := x[i];
```

```
    if y[i] < min_y then min_y := y[i];
    if y[i] > max_y then max_y := y[i];
  end;

  side_length := max(max_x - min_x, max_y - min_y);

  WriteLn(side_length * side_length);
end.
```

Reklamacije

Imamo **n nabiralnikov** in za vsak nabiralnik imamo: 1. **Začetno število zahtevkov**. 2. **Kapaciteto nabiralnika**.

Nato moramo obdelati **q reklamacij**, kjer vsaka reklamacijo poveča število zahtevkov v določenem nabiralniku za neko vrednost. Ko obdelamo vse reklamacije, moramo preveriti, ali so v kateri koli nabiralnikih preseženi njihovi kapaciteti. Če je to res, prenašamo presežek zahtevkov v naslednji nabiralnik, dokler ne dosežemo konca.

Na koncu želimo izpisati število zahtevkov v vsakem nabiralniku po vseh spremembah.

Ključni koraki rešitve

1. **Preberemo vhodne podatke**:
2. Preberemo število nabiralnikov, njihove začetne vrednosti in kapacitete.
3. Preberemo število reklamacij in obdelamo vsako reklamacijo.
4. **Obdelava reklamacij**:
5. Vsako reklamacijo povečamo v ustreznem nabiralniku za določeno vrednost.
6. **Prenos presežkov**:
7. Ko so vse reklamacije obdelane, preverimo vsak nabiralnik, ali presega svojo kapaciteto. Če presega, prenesemo presežek v naslednji nabiralnik, dokler ne dosežemo konca.
8. **Izpis končnega stanja**:
9. Na koncu izpišemo stanje vseh nabiralnikov.

Rešitve v različnih programskih jezikih

Tukaj so rešitve v Pythonu, C, C++ in Pascalu.

Python 3.10

```
import sys
input = sys.stdin.read
data = input().split()

# Preberi število zaposlenih
n = int(data[0])
idx = 1

# Preberi trenutne zahtevke v nabiralnikih
a = [0] * (n + 2)  # Indeksi od 1 do n (0 in n+1 sta bufferja)
for i in range(1, n + 1):
    a[i] = int(data[idx])
    idx += 1

# Preberi kapacitete nabiralnikov
c = [0] * (n + 2)
for i in range(1, n + 1):
    c[i] = int(data[idx])
    idx += 1

# Preberi število zahtevkov
q = int(data[idx])
idx += 1
```

```
# Obdelaj reklamacije
for _ in range(q):
    i = int(data[idx])
    x = int(data[idx + 1])
    idx += 2
    a[i] += x

# Prenos presežkov v naslednje nabiralnike
for i in range(1, n + 1):
    if a[i] > c[i]:
        a[i + 1] += (a[i] - c[i])
        a[i] = c[i]

# Izpiši končno stanje nabiralnikov
print(" ".join(map(str, a[1:n + 1])))
```

C

```
#include <stdio.h>

typedef long long ll;

ll M = 1000000007;

int main() {
    ll n, q, i, x;
    scanf("%lld", &n);
    ll a[n + 2], c[n + 2];

    // Preberi začetne zahtevke
    for (i = 1; i <= n; i++) {
        scanf("%lld", &a[i]);
    }

    // Preberi kapacitete nabiralnikov
    for (i = 1; i <= n; i++) {
        scanf("%lld", &c[i]);
    }

    // Preberi število reklamacij
    scanf("%lld", &q);

    // Obdelaj reklamacije
    for (i = 0; i < q; i++) {
        ll idx, val;
        scanf("%lld %lld", &idx, &val);
        a[idx] += val;
    }

    // Prenos presežkov v naslednje nabiralnike
    for (i = 1; i <= n; i++) {
        if (a[i] > c[i]) {
            a[i + 1] += (a[i] - c[i]);
            a[i] = c[i];
        }
    }

    // Izpiši končno stanje nabiralnikov
```

```
    for (i = 1; i <= n; i++) {
        printf("%lld ", a[i]);
    }
    return 0;
}
```

C++

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    long long n, q;
    cin >> n;

    vector<long long> a(n + 2, 0), c(n + 2, 0);

    // Preberi začetne zahtevke
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    // Preberi kapacitete nabiralnikov
    for (int i = 1; i <= n; i++) {
        cin >> c[i];
    }

    // Preberi število reklamacij
    cin >> q;

    // Obdelaj reklamacije
    for (int i = 0; i < q; i++) {
        int idx;
        long long x;
        cin >> idx >> x;
        a[idx] += x;
    }

    // Prenos presežkov v naslednje nabiralnike
    for (int i = 1; i <= n; i++) {
        if (a[i] > c[i]) {
            a[i + 1] += (a[i] - c[i]);
            a[i] = c[i];
        }
    }

    // Izpiši končno stanje nabiralnikov
    for (int i = 1; i <= n; i++) {
        cout << a[i] << " ";
    }

    return 0;
}
```

Pascal

```
program Reklamacije;

var
  n, q, i, x, y: Int64;
  a, c: array[1..200001] of Int64;

begin
  Read(n);

  // Preberi začetne zahtevke
  for i := 1 to n do
    Read(a[i]);

  // Preberi kapacitete nabiralnikov
  for i := 1 to n do
    Read(c[i]);

  // Preberi število reklamacij
  Read(q);

  // Obdelaj reklamacije
  for i := 1 to q do
  begin
    Read(x);
    Read(y);
    a[x] := a[x] + y;
  end;

  // Prenos presežkov v naslednje nabiralnike
  for i := 1 to n do
    if a[i] > c[i] then
    begin
      a[i+1] := a[i+1] + a[i] - c[i];
      a[i] := c[i];
    end;

  // Izpiši končno stanje nabiralnikov
  for i := 1 to n do
    Write(a[i], ' ');
end.
```

Superšampijon Janez

Imamo seznam a dolžine n, kjer za vsak element $a[i]$ iščemo število podzaporedij, v katerih se ta element pojavi, in nato izračunamo skupno vsoto teh elementov. Klasičen pristop bi bil generiranje vseh podzaporedij, vendar je to prepočasno za večje n. Namesto tega bomo uporabili **delitelje**.

Ključni koraki rešitve

1. **Pojavitev elementa v podzaporedjih**: Vsak element i se pojavi v podzaporedju, če in samo če obstaja nek izbran element z indeksom j, tako da j deli i. To pomeni, da bomo šteli, koliko deliteljev ima posamezen indeks i, in glede na to bomo izračunali, koliko podzaporedij vključuje ta element.

2. **Preštevanje deliteljev**: Število podzaporedij, v katerih ni prisoten element i, je enako 2^{n-k}, kjer je k število deliteljev števila i. To pomeni, da bo število podzaporedij, kjer je element prisoten, enako $2^n - 2^{n-k}$.

3. **Uporaba Eratostenovega sita**: Za učinkovito preštevanje deliteljev vsakega števila bomo uporabili **Eratostenovo sito**. S pomočjo tega sita bomo hitro dobili število deliteljev za vsako število od 1 do n.

4. **Skupni odgovor**: Končni odgovor je vsota:

$$\sum_{i=1}^{n}(2^n - 2^{n-\sigma(i)})a_i$$

   kjer $\sigma(i)$ predstavlja število deliteljev števila i.

5. **Časovna zahtevnost**: Časovna zahtevnost za vsak testni primer je $O(n \log n)$, ker bomo uporabili Eratostenovo sito za izračun števila deliteljev. Rešitev, ki temelji na preverjanju deliteljev do kvadratnega korena, bi imela časovno zahtevnost $O(n\sqrt{n})$, kar ni dovolj hitro za večje n.

Python 3.10

```
import sys

MOD = 10**9 + 7

def solve():
    n = int(sys.stdin.readline())
    arr = list(map(int, sys.stdin.readline().split()))

    divs = [0] * (n + 1)
    for i in range(1, n + 1):
        for j in range(i, n + 1, i):
            divs[j] += 1

    pw2 = [1] * (n + 1)
    for i in range(1, n + 1):
        pw2[i] = pw2[i - 1] * 2 % MOD

    res = 0
    for i in range(1, n + 1):
        res += (pw2[divs[i]] + MOD - 1) % MOD * pw2[n - divs[i]] % MOD * arr[i - 1] % MOD
        res %= MOD

    print(res)

# Preberemo število testov
t = int(sys.stdin.readline())
for _ in range(t):
```

```
    solve()
```

Ta rešitev prinaša 58 točk. Rešitve v Pythonu, ki bi prinesla 100 točk ni, saj je Python za to prepočasen.

C

```c
#include <stdio.h>
#include <math.h>

#define MOD 1000000007

void solve() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    int divs[n + 1];
    for (int i = 0; i <= n; i++) {
        divs[i] = 0;
    }

    for (int i = 1; i <= n; i++) {
        for (int j = i; j <= n; j += i) {
            divs[j]++;
        }
    }

    long long pw2[n + 1];
    pw2[0] = 1;
    for (int i = 1; i <= n; i++) {
        pw2[i] = pw2[i - 1] * 2 % MOD;
    }

    long long res = 0;
    for (int i = 1; i <= n; i++) {
        res = (res + (pw2[divs[i]] + MOD - 1) % MOD * pw2[n - divs[i]] % MOD * arr[i - 1] % MOD
) % MOD;
    }

    printf("%lld\n", res);
}

int main() {
    int t = 1;
    scanf("%d", &t);

    while (t--) {
        solve();
    }

    return 0;
}
```

C++

```cpp
#include <iostream>
#include <vector>
#define MOD 1000000007

using namespace std;

void solve() {
    int n;
    cin >> n;

    vector<int> arr(n);
    for (int& i : arr) {
        cin >> i;
    }

    vector<int> divs(n + 1, 0);
    for (int i = 1; i <= n; i++) {
        for (int j = i; j <= n; j += i) {
            divs[j]++;
        }
    }

    vector<long long> pw2(n + 1, 1);
    for (int i = 1; i <= n; i++) {
        pw2[i] = pw2[i - 1] * 2 % MOD;
    }

    long long res = 0;
    for (int i = 1; i <= n; i++) {
        res = (res + (pw2[divs[i]] + MOD - 1) % MOD * pw2[n - divs[i]] % MOD * arr[i - 1] % MOD
) % MOD;
    }

    cout << res << endl;
}

int main() {
    ios::sync_with_stdio(false);
    int t = 1;
    cin >> t;

    while (t--) {
        solve();
    }

    return 0;
}
```

Pascal

```pascal
program Solution;

const
  MODULO = 1000000007;

var
  t, n, i, j: Int64;
  arr, divs, pw2: array of Int64;
```

```
  res: Int64;

procedure Solve;
begin
  ReadLn(n);
  SetLength(arr, n);
  SetLength(divs, n + 1);
  SetLength(pw2, n + 1);

  for i := 0 to n - 1 do
    Read(arr[i]);

  for i := 1 to n do
    divs[i] := 0;

  for i := 1 to n do
    for j := i to n do
      if j mod i = 0 then
        Inc(divs[j]);

  pw2[0] := 1;
  for i := 1 to n do
    pw2[i] := (pw2[i - 1] * 2) mod MODULO;

  res := 0;
  for i := 1 to n do
  begin
    res := (res + ((pw2[divs[i]] + MODULO - 1) mod MODULO * pw2[n - divs[i]] mod MODULO * arr[i
 - 1] mod MODULO)) mod MODULO;
  end;

  WriteLn(res);
end;

begin
  ReadLn(t);
  while t > 0 do
  begin
    Solve;
    Dec(t);
  end;
end.
```