**Nama : Fajar Satria**

**NRP : 05111940000083**

**Kelas : B**

**Resume Kuliah PAA – 2 April 2021**

Aspek yang paling utama dari sebuah algoritma adalah aspek kebenaran. Dengan tidak melupakan aspek aspek lain seperti aspek waktu dan memori. Aspek kebenaran bisa didapatkan dengan observasi kreatif yang dihubungkan dengan referensi atau teori teori benar. Bahkan, sebuah persoalan sangat mungkin memerlukan solusi yang merupakan gabungan dari beberapa teori sekaligus.

**a. A simple equation**

Persoalan ini terlihat mudah. Akan tetapi akan menjadi problem yang agak sulit apabila ditambahkan suatu batasan yaitu tidak boleh menggunakan *looping*, kecuali pada bagian menerima input testcase. Persoalan ini bisa dikembangkan dengan menggunakan prinsip "kombinasi dengan repetisi", dengan tambahan konsep lain dan observasi cerdas.

TEOREMA 2 : There are C(n+r−1,r) = C(n+r−1,n−1) r-combinations from a set with n elements when repetition of elements is allowed.

Contoh :

How many solutions does the equation

$$x_1 + x_2 + x_3 = 11$$

have, where $x_1$, $x_2$, and $x_3$ are nonnegative integers?

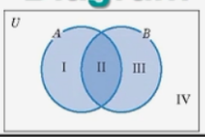
*Solution:* To count the number of solutions, we note that a solution corresponds to a way of selecting 11 items from a set with three elements so that $x_1$ items of type one, $x_2$ items of type two, and $x_3$ items of type three are chosen. Hence, the number of solutions is equal to the number of 11-combinations with repetition allowed from a set with three elements. From Theorem 2 it follows that there are

$$C(3 + 11 - 1, 11) = C(13, 11) = C(13, 2) = \frac{13 \cdot 12}{1 \cdot 2} = 78$$

solutions.

Konsep "kombinasi dengan repetisi" tidaklah cukup untuk menyelesaikan permasalahan ini. Karena konsep "kombinasi dengan repetisi" hanya mampu menghitung kombinasi **a + b + c = n** bukan **a + b + c <= n** . Oleh karena itu, konsep itu harus dimodifikasi dan dengan menambahkan konsep Inklusi-Eksklusi.

## Determining Sets from a Venn Diagram




| Set to Determine | Description of Set | Regions in Venn Diagram |
|---|---|---|
| a. A ∪ B | set of elements in A or B or Both | I,II,III |
| b. (A ∪ B)' | set of elements in U that are not in A ∪ B | IV |
| c. A ∩ B | set of elements in both A and B | II |
| d. (A ∩ B)' | set of elements in U that are not in A ∩ B | I, III, IV |
| e. A' ∩ B | set of elements that are not in A and are in B | III |
| f. A ∪ B' | set of elements that are in A or not in B or both | I,II, IV |

Konsep “kombinasi dengan repetisi” bisa dimanipulasi dari yang semula **x+y+z<=n** menjadi **w+x+y+z=n** . Hal ini bisa dilakukan karena w, x, y, z, dan n adalah *non-negative integer.* Namun permasalahannya adalah nilai yang digunakan pada x, y, dan z mempunyai batasan 0<=x<=A , 0<=x<=B , 0<=x<=C .

Permasalahan tersebut bisa diselesaikan dengan menggunakan konsep inklusi-eksklusi. Yakni untuk menghitung x>A diperlukan rumus baru **w + x + y + z = N - A – 1.** Untuk mempermudah perhitungan pada y>B dan z>C diperlukan sebuah tabel sebagai berikut.

| | | x>A | y>B | z>C | x>A,y>B | x>A,z>C | y>B,z>C | x>A,y>B,z>C |
|---|---|---|---|---|---|---|---|---|
| Inisial | | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| x>A | - | -1 | 0 | 0 | -1 | -1 | 0 | -1 |
| y>B | - | -1 | -1 | 0 | -2 | -1 | -1 | -2 |
| z>C | - | -1 | -1 | -1 | -2 | -2 | -2 | -3 |
| x>A,y>B | + | -1 | -1 | -1 | -1 | -2 | -2 | -2 |
| x>A,z>C | + | -1 | -1 | -1 | -1 | -1 | -2 | -1 |
| y>B,z>C | + | -1 | -1 | -1 | -1 | -1 | -1 | 0 |
| x>A,y>B,z>C | - | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Prinsip yang digunakan adalah setiap daerah hanya boleh dikurangi 1x. Sebagai contoh untuk mencari x<=A maka harus dikurangi dengan x>A. Akan tetapi akan muncul permasalahan baru jika x>A, y>B sama sama dikurangkan, sedangkan y>B sudah dikurangkan untuk mencari y<=B. Oleh karena itu setelah melakukan pengurangan dengan x>A, dan y>B, maka harus ditambah dengan x>A, y>B.

Kembali ke konsep kombinasi bahwa untuk menghitung kombinasi dari (N+3) diambil 3 maka rumusnya adalah

$$\frac{(N+3)!}{N!3!} = \frac{(N+3)(N+2)(N+1)N!}{3!N!} = \frac{(N+3)(N+2)(N+1)}{6}$$

**PSEUDOCODE**

```
COUNT(n)
1. if n<0
2.       return 0
3. else return (n+3) * (n+2) * (n+1) / 6;
```

```
SOLVE (a,b,c,n)

1. ans = 0
2. ans = ans + COUNT(n)
3. ans = ans - COUNT(n-a-1)
4. ans = ans - COUNT(n-b-1)
5. ans = ans - COUNT(n-c-1)
6. ans = ans + COUNT(n-a-b-2)
7. ans = ans + COUNT(n-b-c-2)
8. ans = ans + COUNT(n-a-c-2)
9. ans = ans - COUNT(n-a-b-c-3)
10. return ans
```

**SOURCECODE**

```
#include <cstdio>
using namespace std;
typedef long long LL;
LL count(LL val){
    if(val < 0) return 0;
    return (val+3)*(val+2)*(val+1)/6;
}
int main(){
    int t;scanf("%d",&t);
    while(t--){
        int n,a,b,c;
        scanf("%d %d %d %d",&n,&a,&b,&c);
        LL ans = 0;
        ans += count(n);
        ans -= count(n-a-1);
        ans -= count(n-b-1);
        ans -= count(n-c-1);
        ans += count(n-a-b-2);
        ans += count(n-b-c-2);
        ans += count(n-a-c-2);
        ans -= count(n-a-b-c-3);
        printf("%lld\n",ans);
    }
    return 0;
}
```

**SUBMISSION**

Fajar: submissions
A simple equation

| ID | | DATE | PROBLEM | RESULT | TIME | MEM | LANG |
|---|---|---|---|---|---|---|---|
| 27638422 | ☐ | 2021-04-02 03:44:37 | A simple equation | **accepted** edit ideone it | 0.00 | 4.8M | CPP |

**b. Amazing Factor Sequence**

Pada persoalan ini dapat menggunakan relasi rekurensi. Relasi rekurensi tidak perlu menggunakan solusi rekursif. Jadi rekursif pasti memiliki sifat rekurensi, akan tetapi rekurensi tidak harus diselesaikan dengan rekursif.

Solusi yang dapat digunakan adalah dengan memanfaatkan konsep bilangan bulat. Dalam konsep bilangan bulat, jika *n* adalah komposit maka *n* memiliki pembagi dibawah atau sama dengan $\sqrt{n}$ . Jadi untuk mencari factor prima dari 19, cukup dilakukan pengecekan sampai akar 19.

Konsep bilangan bulat belum sepenuhnya bisa menjadi solusi untuk permasalahan ini. Masih diperlukan observasi lebih lanjut mengenai hubungan antara *n* dan faktor-faktornya.

| *n* | *faktor* | | | | |
|---|---|---|---|---|---|
| 2 | 1 | | | | |
| 3 | 1 | | | | |
| 4 | 1 | 2 | | | |
| 5 | 1 | | | | |
| 6 | 1 | 2 | 3 | | |
| 7 | 1 | | | | |
| 8 | 1 | 2 | 4 | | |
| 9 | 1 | | | 3 | |
| 10 | 1 | 2 | 5 | | |

| *n* | *faktor* | | | | | |
|---|---|---|---|---|---|---|
| 11 | 1 | | | | | |
| 12 | 1 | 2 | 6 | 3 | 4 | |
| 13 | 1 | | | | | |
| 14 | 1 | 2 | 7 | | | |
| 15 | 1 | | | 3 | 5 | |
| 16 | 1 | 2 | 8 | | | 4 |
| 17 | 1 | | | | | |
| 18 | 1 | 2 | 9 | 3 | 6 | |
| 19 | 1 | | | | | |

Setelah melakukan observasi didapatkan bilangan yang selalu muncul yaitu angka 1, bilangan yang muncul setelah angka 2 adalah 3 sampai 9, bilangan yang

muncul setelah angka 3 adalah 4 sampai 6, bilangan yang muncul selanjutnya adalah 4. Angka 1 dapat disimpan pada operasi lain, karena angka 1 selalu muncul pada tiap factor *n*. Bilangan yang muncul setelah angka 2 dimulai dari 3 yaitu 2+1 sebagai batas bawah sampai dengan 9 yaitu 19/2 sebagai batas atas. Bilangan yang muncul setelah angka 3 dimulai dari 4 yaitu 3+1 sebagai batas bawah sampai dengan 6 yaitu 19/3 sebagai batas atas. Bilangan yang muncul selanjutnya adalah 4, yaitu akar dari 19.

Dengan demikian, dapat di simpulkan bahwa konsep deret bilangan dapat digunakan. Konsep deret bilangan dapat digunakan jika memenuhi batas bawah kurang dari sama dengan batas atas. Akan tetapi konsep deret bilangan tersebut tetap harus dimodifikasi dengan konsep inklusi-eksklusi.

Sebagai contoh untuk mencari jumlah 1-n maka digunakan rumus $\frac{n(n+1)}{2}$, akan tetapi pada permasalahan soal ini hanya mencari dari batas bawah a1 = (a+1) sampai batas atas $a2 = (n/a)$. Maka solusinya adalah dengan mencari jumlah dari 1 sampai batas atas $a2$ dikurangi dengan jumlah dari 1 sampai a1-1.

$$sum = \frac{a2(a2+1)}{2} - \frac{a1(a1-1)}{2}$$

Pada saat n=19, setelah selesai dengan deret bilangan masih perlu ditambahkan lagi dengan angka yang selalu muncul yaitu angka 2 dan angka 3 dapat dicari dengan manambahkan $a * (a2 - a1 + 1)$ . Setelah selesai solusi masih perlu ditambah dengan angka 2 dan 3 itu sendiri dan juga ditambahkan angka 1 sebanyak $n - 1$.

**PSEUDOCODE**

```
SOLVE(n)
1.  ans =0
2.  for i=2 to i*i = n
3.      a1 = (i + 1)
4.      a2 = n / i
5.      if a1 <= a2
6.          sum = (a2 * (a2 + 1)) / 2 – (a1 * (a1 – 1)) / 2
7.          ans = ans + sum
8.          ans = ans + (a * (a2 – a1 + 1))
9.      ans = ans + i
10. ans = ans + (n – 1)
11. return ans
```

**SOURCECODE**

```
#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long LL;
template<typename T>
T getNum(){
    T res=0;
    char c;
    while(1){
        c = getchar_unlocked();
        if(c==' '||c=='\n') continue;
        else break;
    }
    res=c-'0';
    while(1){
        c = getchar_unlocked();
        if(c>='0' && c<='9') res=10*res+c-'0';
        else break;
    }
    return res;
}
int main(){
    int T;
    LL n, a, a1, a2, ans, sum;
    T = getNum<int>();
    while(T--){
        n = getNum<LL>();
        ans = 0;
        for(a=2; a*a<=n; ++a){
            a1 = (a+1);
            a2 = n/a;
            if(a1<=a2){
                sum = (a2*(a2+1))/2 - (a1*(a1-1))/2;
                ans += sum;
                ans += a*(a2-a1+1);
            }
            ans += a;
        }
        ans += n-1;
        printf("%lld\n",ans);
    }
    return 0;
}
```

**SUBMISSION**

Fajar: submissions
Amazing Factor Sequence

| ID | | DATE | PROBLEM | RESULT | TIME | MEM | LANG |
|---|---|---|---|---|---|---|---|
| 27638732 | ☐ | 2021-04-02 04:58:33 | Amazing Factor Sequence | **accepted** edit ideone it | 0.00 | 4.6M | C |
| 27638612 | ☐ | 2021-04-02 04:48:24 | Amazing Factor Sequence | **accepted** edit ideone it | 0.00 | 4.6M | CPP |

**FAST Input**

Selain algoritma ada juga hal lain yang membuat solusi pada online judge menjadi tidak optimal, yaitu proses input dan output. Proses input dan output pasti memerlukan waktu yang paling panjang karena berhubungan erat dengan eksternal device. Contoh realnya adalah kecepatan mengetik pasti tidak bisa melebihi kecepatan processor dalam menghitung. Untuk mengimbangi kesenjangan tersebut, microprocessor memerlukan sebuah mekanisme untuk mempermudah komunikasi antara proses cepat dengan proses lambat, yakni menggunakan interupsi.

Namun dalam sebuah platform online judge, tidak diperlukan interupsi tersebut. Hal ini dikarenakan online judge membaca bukan melalui pengetikan keyboard, melainkan langsung dengan membaca file. Sehingga kita dapat menghindari proses interupsi tersebut.

Pada kasus normal, fungsi input yang digunakan adalah *getchar*(). Sedangkan pada saat ingin menonaktifkan proses interupsi dapat menggunakan fungsi input *getchar_unlocked*().

Dengan sedikit penggabungan antara looping dan *getchar_unlocked*(). Maka fungsi tersebut dapat digunakan untuk membaca integer dari yang semula hanya 1 digit char. Fungsi getNum dibawah ini hanya dapat digunakan untuk membaca integer dan long integer. Untuk membaca float dan string, masih diperlukan beberapa modifikasi lain.

**SOURCECODE FASTINPUT**

```
#include <cstdio>
#include <algorithm>
using namespace std;
typedef long long LL;
template<typename T>
T getNum(){
    T res=0;
    char c;
    while(1){
        c = getchar_unlocked();
        if(c==' '||c=='\n') continue;
        else break;
    }
    res=c-'0';
    while(1){
        c = getchar_unlocked();
        c = getchar();
        if(c>='0' && c<='9') res=10*res+c-'0';
        else break;
    }
    return res;
}
```