ZIP ZIP KURBAĞA

“Zıp zıp kurbağa bir nehri karşıdan karşıya geçmek istemektedir. Nehrin iki kenarı arasında N adet taş vardır. Kurbağa sadece ileri doğru ve tek bir seferde rastgele olarak 1, 2 veya 3 sonraki taşa zıplamaktadır. N adet taş verildiğinde nehrin bir kenarından diğer kenarına kaç farklı şekilde gidebilir?”

Özet olarak bir kurbağamız var. Önündeki nehirde N adet taş var. Kurbağamız 1, 2 veya 3 taş zıplayabiliyor. N adet taştan oluşan bir yolu kaç farklı şekilde gidebilir?

1 için barizce 1.

2 için 1+1 ya da 2, toplam 2.

3 -> 1+1+1, 1+2, 2+1, 3; toplam 4.

4 -> 1+1+1+1, 1+2+1, 2+1+1, 3+1, 1+1+2, 2+2, 1+3; toplam 7.

Üşenmeyip yazılırsa 5 için 13 ve 6 için 24 bulunabildiği rahatlıkla görülebilir.

Biraz incelendiğinde kolaylıkla atılan her bir adımın aslında bizi başka bir duruma sürüklediği anlaşılıyor. Mesela 5 taş var önümüzde. 2 taş atlarsak önümüzde 3 taş kalır ve başta önümüzde 3 taş olsaydı yapabileceğimiz hamleleri yaparız. 2 taş atlamak yerine 1 atlasaydık önümüzde 4 taş senaryosu oluşurdu. Aynı şey ilk hamlemizi 3 olarak yaptığımızda da olur.

Yani 3’ten büyük hangi durumda olursak olalım, N sayımızdan 1, 2 ve 3 eksik olan durumları toplarsak N için olan durum sayısını buluruz.

Özyinelemeli fonksiyon olarak yazması o kadar da zor değil. 3 adet temel durum var. Diğerleri için 1, 2 ve 3 eksiğinin toplamı.

Yinelemeli olarak 4 değişken belirlenip sonraki adımda ilk 3 değişkenin kendisinden sonra gelen değişkenin değerini alması, sonuncu değişkenin ise kendisinden öncekilerin toplam değerini alması yeterli.

Matematik açısından bakarsak Fibonacci serisine benzediği aşikâr. Fakat burada önceki 3 eleman toplanarak bir sonraki elde ediliyor, önceki 2 değil.

İncelersek *f* (x) = *f* (x-1) + *f* (x-2) + *f* (x-3) şeklinde fonksiyon olarak ifade edebiliriz. Tanım kümesini [4,∞) olarak belirtmek oluşabilecek karışıklıkları engeller. Fonksiyonun davranışını incelemek için ifadeyi *f* (x-1)’e bölebiliriz. *f* (x) / *f* (x-1) ifadesine P der isek ifademiz P = 1 + *f* (x-2) / *f* (x-1) + *f* (x-3) / *f* (x-1) şeklini alır. *f* (x) / *f* (x-1) ile *f* (x-1) / *f* (x-2) ifadesi aynı şeyi ifade eder. Belirli bir elemanın kendisinden önce gelen elemana oranını. Bu durumda buna da P diyebiliriz. Elimizde tersi bulunduğu için ifademiz P = 1 + 1/P + *f* (x-3) / *f* (x-1) halini alır.

P’nin *f* (x) / *f* (x-1) ya da *f* (x-1) / *f* (x-2) şeklinde ifade edilebileceğini belirtmiştik. Aynı şekilde bu oran *f* (x-2) / *f* (x-3) için de geçerlidir. Yani *f* (x-1) / *f* (x-2) = *f* (x-2) / *f* (x-3) diyebiliriz. Burada içler dışlar çarpımı yaparsak *f* (x-2)’nin karesi *f* (x-1) \* *f* (x-3) ifadesine eşit olur. İki tarafı da *f* (x-1)’in karesine bölersek *f* (x-3) = (*f* (x-2))^2 / (*f* (x-1))^2 olacaktır. *f* (x-2) / *f* (x-1) ifadesi 1/P olur. Karesi alınıp asıl eşitlikte yerine yazılırsa P = 1 + 1/P + 1/P^2 olur. Paydalardan kurtulmak için ifadeyi P^2 ile çarparsak 0 = P^3 - P^2 - P -1 sonucuna ulaşırız. Bu ifadenin kökünü hesaplaması yeterince uğraştırıcı olduğu için yapma gereği duymuyorum. P = 1,83928675… gibi bir ifadeye eşit oluyor. Diğer kökleri reel değil. Elde ettiğimiz sayıların oranının da buna yakınsadığını test edebiliriz.

Buraya ulaştıktan sonra daha fazlası için elimdeki veriler ile internette araştırma yaptım. Buraya kadar sadece yıllar önce Numberphile kanalının [videosu](https://www.youtube.com/watch?v=dTWKKvlZB08) sayesinde geldim. Başta [gümüş oran](https://www.youtube.com/watch?v=7lRgeTmxnlg) gibi bir şey çıkacağını ummuştum ama öyle olmadı. Daha fazla arayınca elimizdeki Fibonacci serisine benzeyen seriye [Tribonacci serisi](http://oeis.org/A000073) dendiğini öğrendim.

Matematik kısmını düzgünce anlayınca kod kısmına pek bir şey kalmıyor. Biraz oyunlar kuramı bilgisi yapılan her seçenekten sonra başka bir senaryoya ulaşıldığını anlamak için yeterli.

Tek sorun kod kısmının yavaş olması. Bu yavaşlığının sebebi fonksiyonun tekrar tekrar çok sayıda fonksiyon çağırması. Temel durumlar için de hesap olsa taş başına yaklaşık 3^N adet fonksiyon çağırılıyor. Yinelemeli olanda ise her adımda 4 işlem yapılıyor, bu da taş başına 4\*N adet işlem demek. Diğerine kıyasla ne kadar küçük kaldığı bariz. Süre farkının sebebi de budur. Bu farkı azaltmak için bulunan her miktar global bir vector’e eklenebilirdi. Ya da global olmayan, parametre olarak alınan bir vector’e. Global dizi de olabilirdi ama bu sefer gideceğimiz adım sayısını belirtmemiz gerekirdi.

72 olma sebebi değişkenlerimiz signed long long. Yani bir biti işarete ayrılmış 8 byte. Bu 9 kentilyon (9.223.372.036.854.775.808‬) yapıyor. 72. adımda hesaplanan taş sayısı 7 kentilyon (7.015.254.043.203.144.209) civarı. Bundan sonraki adımın 9 kentilyonu geçeceği bariz. Bu durumda veri kaybı yaşanır.

Bu arada attığınız kod 72 değil, 71’e kadar yazdırıyor. Düzeltmeyin dediğiniz için ellemedim.