©Ali Galip Bayrak - ODTÜ Bilgisayar Topluluğu

Oyun

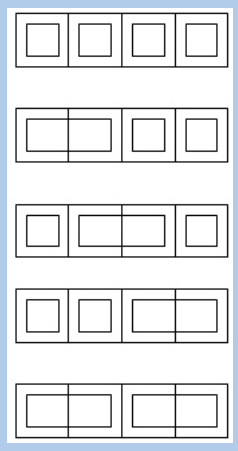
Berk ve Can eski eşyaları karıştırırken bir oyun bulurlar. Uzun uğraşlardan sonra ikili oyunu çözer. Oyun 1xn'lik bir tahtada oynanmaktadır. Eski bir kutuda n'er tane 1x1'lik ve 1x2'lik taşlar bulunmaktadır. Sırası gelen oyuncu boş kalan yerlerden istediği yere kutudaki taşlardan birisini alarak koyar. Son taşı koyan oyuncu oyunu kazanacaktır. Oyunu biraz daha karıştırırken bir kağıt bulurlar. Kağıtta "oyuna başla ve kazan" yazmaktadır. Hemen oyunu alarak size gelirler. Sizden istedikleri oyun sonunda kaç farklı tahta durumu olabileceğini belirlemeniz ve kazanan bir strateji bulmanız.

Girdi – Çıktı (standart girdi, çıktı):

• İlk olarak girdiden n sayısını ifade eden bir tamsayı okumalısınız ($1 \le n \le 100$).

• Daha sonra çıktıya tek satırda oyun sonunda kaç farklı tahta durumu olabileceğini basmalısınız.

• Takip eden satırlarda oyun sonuna kadar önce hamlenizi basmalı, sonra rakibin hamlesini okumalısınız. Her hamle iki sayıdan oluşmaktadır. İlk sayı 1 ya da 2, koyduğunuz taşın enini ifade edecek, ikinci sayı taşın solunun kaçıncı karede olduğunu ifade edecektir.



Örnek:

Girdiden okuduğunuz ilk sayı 4 olsun. İlk olarak n=4 için kaç farklı oyun sonu durumu olabileceğini basmalısınız. Aşağıda görebileceğiniz üzere cevap 5'tir (farklı oyun sonu durumlarını sayarken taşların koyulma sırasını önemsemeyeceksiniz).

Daha sonra hamlelerinizi basmalı ve rakibin hamlelerini okumalısınız. Örnek bir oyun (2. şekildeki) aşağıdaki gibi olabilir:

2 1 (ikilik taşı 1. kareye koydunuz, bunu standart çıktıya yazdınız)

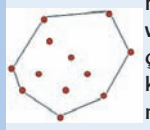
1 3 (rakip tekli taşı 3. kareye koydu, bunu standart girdiden okudunuz)

1 4 (tekli taşı 4. kareye koydunuz, bunu standart çıktıya yazdınız)

Bu durumda oyunu kazanacaksınız fakat farkedeceğiniz üzere rakip kendi ilk hamlesinde 3. kareye ikilik bir taş koysa idi kaybedecektiniz. Dolayısıyla ilk hamlede 2 1 oynamak hatalı bir seçim.

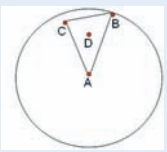
Geçen Sayımızdaki Soruların Çözümleri

Güvenlik: Daha önce, convex hull'dan bahsetmiştik. Tekrar değinecek olursak, verilen bütün noktaları içine alan çokgene convex hull denir. Soldaki şekilde bir convex hull görünmekte.



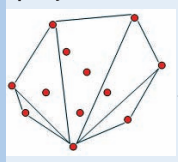
Şimdi birbirine en uzak olan iki noktanın convex hull üzerinde olması gerektiğini ispatlayalım.

Lemma 1: Bir ABC üçgeni ve bu üçgen içindeki bir D noktası verilsin. $|AD| < \max\{|AB|,|AC|\}$ (AB ve AC kenarlarından uzun olanın uzunluğu) olmalıdır.



Yukardaki şekle bakarak lemma'mızın doğruluğunu görebiliriz. AB kenarı büyük olan kenar olsun, A'yı merkez kabul eden ve B'den geçen çemberi çizersek, bu üçgen içerisindeki herhangi bir noktanın çember içinde kaldığını görebiliriz. Dolayısıyla üçgen içindeki hiçbir noktanın uzunluğu AB ve AC kenarlarından uzun olanın uzunluğundan daha fazla olamaz (A merkez olduğu için çember içindeki herhangi bir noktanın A'ya uzaklığı yarıçaptan yani AB'den küçüktür).

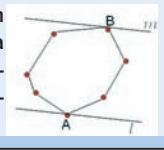
Convex hull'ı aşağıdaki gibi üçgenlere ayırarak ve Lemma 1'i kullanarak birbirine en uzak iki noktanın convex hull üzerinde olması gerektiğini ispatlayabiliriz.



Convex hull'ı O(n logn)'lik bir algoritma kullanarak bulabiliriz. Convex hull'ı bulduktan sonra birbirine en uzak iki noktayı O(n)'lik bir algoritma ile saptayabiliriz.

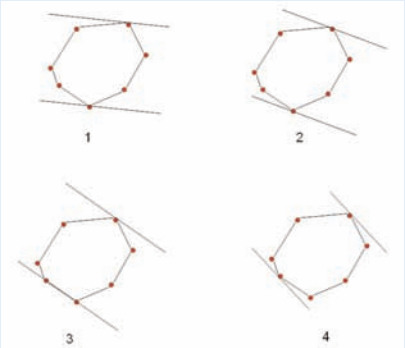
Çokgen üzerindeki bir noktadan geçen ve çokgenin tamamını bir tarafında bırakan doğruya o noktanın destek doğrusu diyelim.

Şekilde *l* doğrusuna A'nın destek doğrusu, *m* doğrusuna da B'nin destek doğrusu diyebiliriz. Bu şekilde, paralel destek doğruları arasındaki uzaklıklardan en büyüğünü bulursak çözüme ulaşmış oluruz. En büyüğe de şu şekilde ulaşabiliriz.



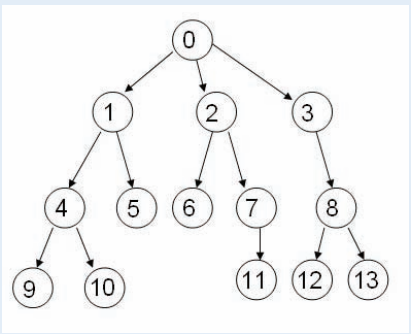
• Rastgele bir yerden başla (örneğin x'i en küçük olan nokta ile x'i en büyük olan noktaların paralel destek doğrularından)

• Destek doğrularını bir yöne doğru çevirmeye başla ve çokgenin kenarlarından birisine değdiği anda destek noktasını değilen kenarın diğer ucundaki nokta olarak değiştir. Ve aynı işlemi tekrar en baştaki duruma dönene kadar tekrarla.



Yukardaki şekilde döndürme işlemini devam ettirerek 1 nolu şekle tekrar dönersek birbirine en uzak olan iki noktayı saptamış oluruz.

Taşlar: Bu soruyu ağaç üzerinde arama metodlarından birisini kullanarak yapabiliriz. Ben size DFID (depth first iterative deepening, yani derinlik öncelikli aşamalı derinlikli) arama yönteminden bahsedeceğim. Bu yöntemi daha önce bahsetmiş olduğum DFS (depth first search, yani derinlik öncelikli arama)'yi kullanarak tanımlayabiliriz. DFS'yi hatırlayacak olursak: Herhangi bir durumda yapabileceğimiz hamlelerden birisini yapıp çözümü ararız, eğer o hamle hiçbir şekilde çözüm üretemiyor ise aynı durumda yapabileceğimiz başka bir hamleyi deneriz. Örnek bir şekille gösterecek olursak:



Her düğüm (çemberler ile gösterilen) bir durumu ifade etsin. 0 nolu durumdan başlarız. İlk olarak, 1 nolu duruma geçtiğimizde çözüm var mı diye 1 nolu düğümün altındaki ağacı aynı şekilde ararız. Çözüme ulaşamadıysak 2 nolu düğümün altındaki ağacı, onda da çözüme ulaşamadıysak 3 nolu düğümün altındaki ağacı ararız. Örneğin 11 nolu düğüm sonuç durumu olsun. Bu durumda ağaçtaki gezintimiz 0-1-4-9-10-5-2-6-7-11 şeklinde olacaktı.

DFS'ye derinlik kısıtı koyduğumuzu, yani verilen derinlikten daha derine inmeyi yasakladığımızı düşünelim (0 nolu düğüm 0. derinlikte kabul edelim). Yukardaki ağaçta derinlik kısıtı olarak 2 koyduğumuzda gezintimiz 0-1-4-5-2-6-7-3-8 şeklinde olacaktı. DFID'de yaptığımız şey ise 0'dan başlayarak derinlik kısıtı koymak ve aramayı yapmak, eğer sonuca ulaşamadıysak derinlik kısıtını 1 artırıp tekrar aramayı yapmak. Yukardaki şekilde DFID kullanarak 11'i bulmamız şu şekilde olacaktı:

0 (Derinlik kısıtı 0 iken)

0-1-2-3 (Derinlik kısıtı 1 iken)

0-1-4-5-2-6-7-3-8 (Derinlik kısıtı 2 iken)

0-1-4-9-10-5-2-6-7-11 ve sonuca ulaştık (Derinlik kısıtı 3 iken)

Düğümlerin her birisini tahtanın bir durumu, 0 nolu düğümü tahtanın ilk durumu ve bir düğümün çocuklarını o durumdayken yapılabilecek hamleler sonucunda oluşabilecek tahta durumları olarak düşünürsek problemi çözebiliriz.