

Reconstruct

Zaman sınırı: 4 saniye

Bellek sınırı: 1024 MiB

Bu etkileşimli bir problemdir.

Bissy, EJOI delegasyonlarının şehir genelindeki farklı konumları ziyaret etmek için yarışacağı, gönüllüler tarafından düzenlenen *Final destination* adlı oryantiring oyununa katılmak üzeredir. Ancak Bissy, oyunu kazanmaktan çok oyunun *nasıl* tasarlandığıyla ilgilenmektedir.

Haritada N konum ve N takım bulunmaktadır. Her takım bütün konumları ziyaret etmek zorundadır ve takımların başlangıç konumları birbirinden farklıdır — i numaralı takım, i numaralı konumdan başlar. Her takıma kendisine özel bir rota verilmiştir.

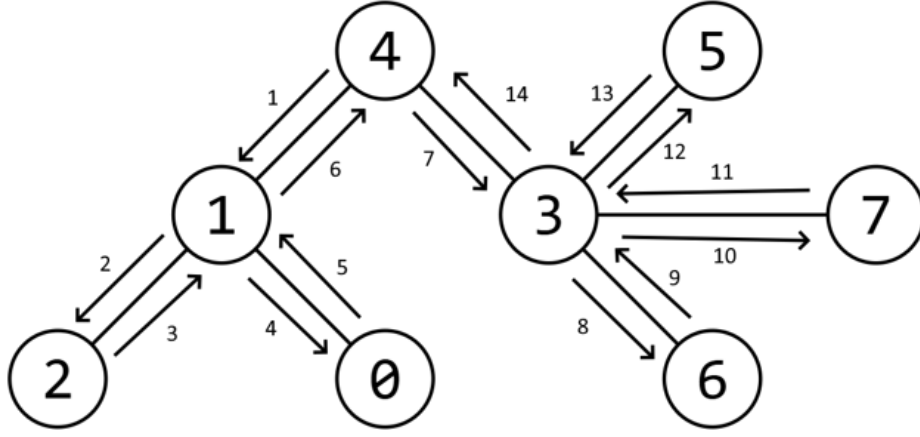
Bissy ayrıca bazı içeriden bilgilere sahiptir — rotalar, hızlı toplu taşıma hatlarından oluşan gizli bir yapıya dayanmaktadır. Daha açık ifade etmek gerekirse, yarışmanın jürisi her biri iki konumu birbirine bağlayan $N - 1$ hat seçmiştir ve bu hatlar kullanılarak herhangi bir konumdan başka herhangi bir konuma ulaşılabilir. Böyle bir yapıya ağaç da denir. Daha sonra jüri, her başlangıç konumu i için **herhangi bir geçerli DFS yürüyüşü** oluşturmuş ve bu yürüyüş i numaralı takımın rotası olarak belirlemiştir.

Bissy, jürinin seçtiği toplu taşıma ağacını bulmak istemektedir. Yalnızca aşağıdaki biçimde hızlı sorular sorabilmektedir:

- i numaralı takımın j numaralı hedefi nedir?

Gizli ağacı bulan `find_tree` programını yazarak Bissy'ye yardım edin.

Bir ağacın *DFS yürüyüşü*, derinlik öncelikli arama prosedürü sırasında düğümlerin ilk kez ziyaret edilme sırasıdır. Böyle bir prosedür, belirli bir v düğümünden başlar ve ziyaret edilmemiş komşularından birine özyinelemeli olarak gider. Ziyaret edilebilecek komşu kalmadığında bir önceki ziyaret edilen düğüme geri döner ve işlemi oradan sürdürür. Örnek:



Burada $v = 4$ ve kenarlar üzerindeki oklar, derinlik öncelikli aramanın adımlarını göstermektedir. Oluşturulan yürüyüş $[4, 1, 2, 0, 3, 6, 7, 5]$ şeklindedir. Komşuların ziyaret edilme sırasının önemli olduğuna dikkat edin; başka bir yürüyüş $[4, 3, 5, 7, 6, 1, 0, 2]$ şeklinde olabilir.

Ağaç yapısı ve N adet DFS yürüyüşü, programınız başlamadan önce sabitlenir ve sorduğunuz sorulara göre değişmez. Bu DFS işlemlerinde düğümlerin komşuları için kullanılan sıralamalar birbirinden farklı olabilir.

Uygulama detayları

Aşağıdaki fonksiyonu gerçeklemelisiniz:

```
std::vector<std::pair<int, int>> find_tree(int N)
```

- burada N , konumların sayısıdır;
- dönüş değeri, ağacın kenarlarının listesidir — kenarların ve kenar uçlarının sırası önemli değildir.

Her test için bu fonksiyon en fazla T kez çağrılabilir.

Jüri ile etkileşim kurmak için aşağıdaki fonksiyonu çağırabilirsiniz:

```
int guess(int i, int j)
```

Bu fonksiyon, ağacın i numaralı yürüyüşündeki j numaralı konumu döndürür. `guess(i, 0)` çağrısının i değerini döndüreceğine dikkat edin. Fonksiyonun, 6 numaralı alt göreve kadar olan bütün alt görevlerde sabit zamanda $O(1)$, 7 numaralı alt görevde ise logaritmik zamanda $O(\log N)$ cevap verdiğini varsayabilirsiniz. $0 \leq i, j \leq N - 1$ koşulu sağlanmalıdır; aksi hâlde çözümünüz `Output isn't correct: Invalid call` sonucunu alır.

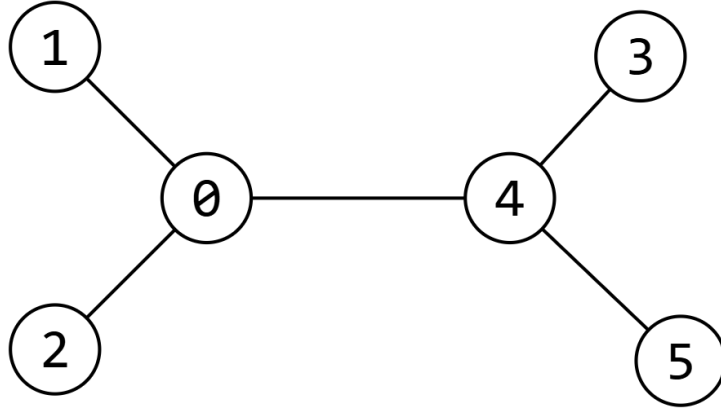
Kısıtlar

- $2 \leq N \leq 2^{16} + 1$
- T , bir testteki fonksiyon çağrıları arasında görülen en büyük N değerine göre belirlenen kısıtlara tabidir. Bu en büyük değer N_{max} ile gösterilsin:
 - eğer $N_{max} \leq 9$ ise $1 \leq T \leq 100$
 - eğer $N_{max} \leq 2^{10} + 1$ ise $1 \leq T \leq 10$
 - eğer $N_{max} \leq 2^{16} + 1$ ise $1 \leq T \leq 3$

Sistem değerlendiricisi en fazla 280 MiB bellek kullanabilir ve bu miktar çözümünüzün bellek sınırına dâhildir.

Örnek

Hızlı toplu taşıma hatlarının gizli yapısının aşağıdaki gibi olduğunu varsayalım:



Başlangıç konumu 0 olan takımın yürüyüşünün $[0, 1, 2, 4, 3, 5]$ olduğunu varsayalım. Aşağıda örnek bir etkileşim verilmiştir:

Yarışmacı programı	Jüri programı
	<code>find_tree(6)</code>
<code>guess(0, 0)</code>	0
<code>guess(0, 1)</code>	1
<code>guess(0, 2)</code>	2
<code>guess(0, 3)</code>	4
<code>guess(0, 4)</code>	3
<code>guess(0, 5)</code>	5
<code>return {{0,1},{0,2},{4,0},{5,4},{3,4}};</code>	

Sorgulardan elde edilen bilgilerin ağacı benzersiz biçimde belirlemek için yeterli olmadığına dikkat edin. Ancak bu cevap, aslında 0 numaralı alt görevdeki testin doğru cevabıdır.

Alt görevler

Alt görev	Puan	N	Ek kısıtlar
0	0	-	Örnek.
1	11	≤ 9	-
2	6	≤ 100	Her düğüm en fazla 2 başka düğüme bağlıdır.
3	13	≤ 100	Her yürüyüş, her adımda 0 numaralı düğümden uzaklaşmaya öncelik veren bir derinlik öncelikli arama tarafından oluşturulmuştur. Eğer 0 numaralı düğümden uzaklaşmak için birden fazla seçenek varsa bunlardan herhangi biri seçilir.
4	11	≤ 100	0 dışındaki her düğüm en fazla 2 başka düğüme bağlıdır.
5	10	≤ 100	-
6	31	$\leq 2^{10} + 1$	-
7	18	$\leq 2^{16} + 1$	-

Puanlama

5 numaralı alt göreve kadar olan bütün alt görevlerde, ağacı zaman sınırı içerisinde başarıyla bulursanız tam puan alırsınız. 6 ve 7 numaralı alt görevlerde, belirli bir testten alacağınız toplam puanın S oranı, o testin alt testlerinden herhangi birinde yaptığınız **en yüksek** sorgu sayısı olan Q_{max} değerine bağlıdır.

Bu alt görevlerde puanlama aşağıdaki gibidir:

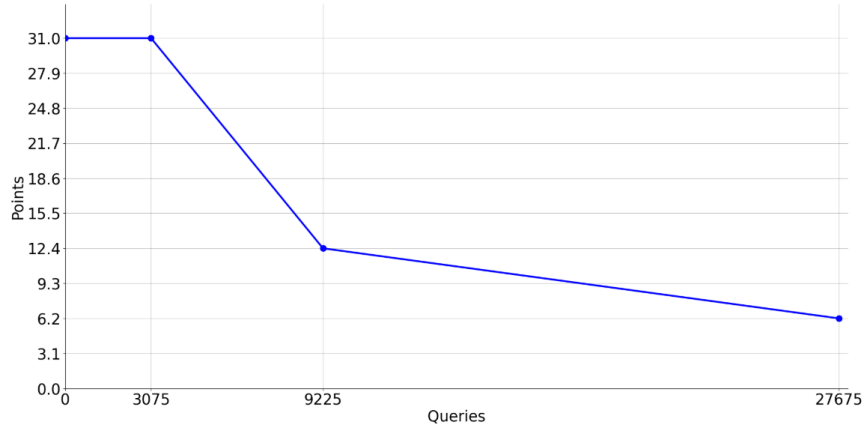
- eğer $Q_{max} \leq L_1$ ise $S = 1.0$;
- eğer $L_1 < Q_{max} \leq L_2$ ise $S = 0.4 + 0.6 \times \frac{L_2 - Q_{max}}{L_2 - L_1}$;
- eğer $L_2 < Q_{max} \leq 3L_2$ ise $S = 0.2 + 0.2 \times \frac{3L_2 - Q_{max}}{2L_2}$;
- eğer $3L_2 < Q_{max}$ ise $S = 0.2$;

burada 6 numaralı alt görev için $L_1 = 3 \times (2^{10} + 1) = 3075$ ve $L_2 = 9 \times (2^{10} + 1) = 9225$, 7 numaralı alt görev için ise $L_1 = 3 \times (2^{16} + 1) = 196611$ ve $L_2 = 15 \times (2^{16} + 1) = 983055$ değerleri kullanılır.

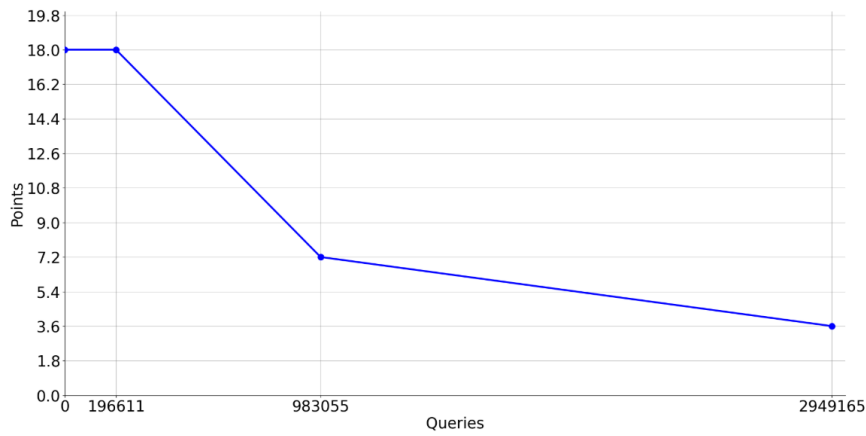
Alt görevin tamamından alacağınız puan oranı, bütün testlerde elde edilen en küçük S değeridir.

Aşağıdaki grafikler son iki alt görev için puanlama fonksiyonunu göstermektedir:

Alt görev 6



Alt görev 7



Örnek değerlendirici

İki adet örnek değerlendirici sağlanmaktadır.

Yerel testler için `Lgrader.cpp`, programınızla birlikte derlenerek kullanılabilir. Program ilk olarak test vakalarının sayısı olan T değerini okur. Daha sonra her test vakası için konum sayısı N ve ardından doğrudan bir hattı tanımlayan iki tam sayıdan oluşan $N - 1$ satır bekler. Bundan sonra her biri N tam sayı içeren N satır gelir – bunlar DFS yürüyüşleridir. i numaralı yürüyüşün i numaralı düğümden başlaması gerektiğine dikkat edin. `AUTO_GENERATE` sabitini `true` olarak ayarlayarak değerlendiricinin yürüyüşleri sizin için oluşturmasını sağlayabilirsiniz. Çıktı olarak değerlendirici, sonuç hatalıysa bir hata mesajı; aksi takdirde her test vakası için yapılan sorgu sayısını ve genel maksimum sorgu sayısını bildirir. Değerlendiricinin yeterince büyük N değerlerinde, daha açık ifade etmek gerekirse 7 numaralı alt görevin kısıtlarında çalışmadığına da dikkat edin. Bu durum için kendi yöntemlerinizi kullanabilirsiniz.

Sistemdeki kullanıcı testleri için `stub.cpp`, programınızla birlikte kullanılabilir. Girdi biçimi diğer değerlendiricinin girdi biçimiyle aynıdır. Kullanıcı testlerinde yürüyüşlerin otomatik olarak oluşturulmasını sağlayan yerleşik bir seçenek bulunmadığına dikkat edin.