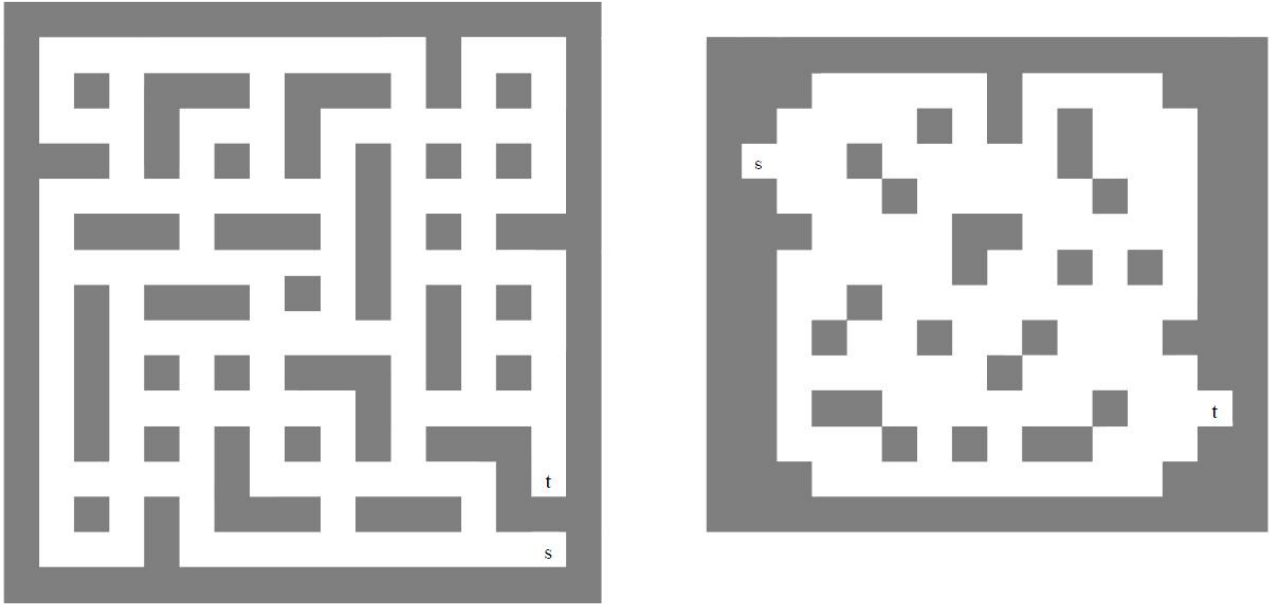


Problem Seti 8 Çözümleri

Problem 8-1. Sola Dönüş yok

Yeni bir Nexus Nano çaldınız, ancak arabanın sahibi, arabayı, The Spade denilen bir mekanik kilitle sola dönmelerini engelleyecek şekilde kitlemiş. Kaza yapmamak ve dikkat çekmemek için herhangi bir U dönüş de yapmamanız gerekiyor. Düz giderken hızlı gidebildiğinizden, sağa dönüşleri en aza indirerek, olabildiğince çabuk şekilde, aracın parçalarını satacağınız garajınıza ulaşmanız gerekiyor. Ancak eğer iki farklı yoldaki sağa dönüş sayısı aynı ise, düz gittiğiniz mesafeyi de azaltmak istiyorsunuz.

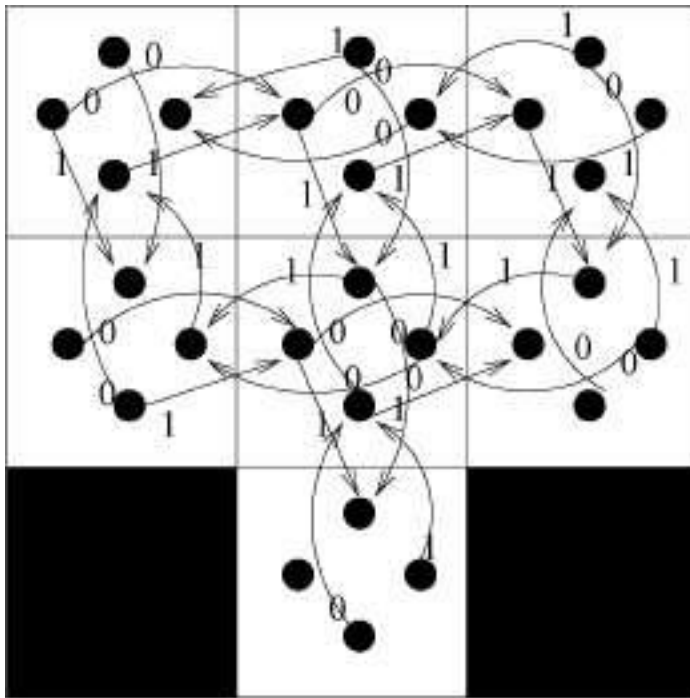
İyi ki elinizde şehrin haritası var, böylece sola dönmeden ve U dönüşü yapmadan bir yol planlayabilirsiniz. Daha da iyisi, haritanın $m \times n$ hücreli elektronik sürümü de mevcut. Bu hücreler ya boş(geçilebilir) ya dolu(geçilemez) şeklindedir. Her bir hücrede geldiğiniz yönde devam edebilir ya da sağa dönebilirsiniz. Sola dönmeye ihtiyacınız olmayan 2 örnek harita Şekil 1'de gösterilmiştir.



Şekil 1: 2 tane sola dönüşsüz harita. s'den t'ye olabildiğince az sağa dönüşlü, sola dönüşsüz ve U dönüşsüz yol bulun. Bu haritalar ile www.clickmazes.com/noleft/ixnoleft.htm adresindeki Java uygulamaları ile oynayabilirsiniz.

(a) $m \times n$ büyüklüğünde bir haritada, sola dönüşsüz, olabildiğince az sağa dönüşlü bir yol bulmak, veya böyle bir yol olmadığını belirlemek için verimli bir algoritma bulun. Algoritmanız aynı zamanda, eşit sağa dönüşlü yollar arasında, düz yol uzunluğunu da en aza indirmelidir. (*İpucu:* Soruyla ilgili bir sezgi edinebilmek için sol taraftaki harita için soruyu çözün.)

Çözüm: Problemi çözmek için temel fikir, problemi bir grafik olarak simgelemektir. Daha sonra en uygun sola dönüşsüz yolu bulmak için en kısa yol algoritmasından faydalanılır. Grafikteki her (u,v) kenarı, $w_1(u, v)$ ve $w_2(u, v)$ gibi 2 tane ağırlığa sahiptir. Eğer kenar bir sağa dönüşü simgeliyor ise, $w_1(u, v)=1$ atamasını yaparız, buna karşılık eğer kenar bir düz gidişi simgeliyorsa $w_2(u, v) = 1$ atamasını bir mesafe ile yaparız.



Şekil 2: Grafik Gösterim.

Bu problemi bir G grafiğine çevirmenin bir yolunu Şekil 2' de görüyorsunuz. Hücrelerden oluşan haritamız, ağırlıklı yönlendirilmiş bir grafiğe çevrildi. Haritadaki her mn hücre için, 4 düğüm oluşturuyoruz. Eğer araba, belirli bir hücreye onun altındaki başka bir hücreden geliyorsa, burada “güney” düğümüne girer.

Düğümleri yarattıktan sonra, onlara yönlendirilmiş kenarları ekleriz. Sağa dönüşü denk gelen bir kenar $w_1(u, v)=1$ ve $w_2(u, v)=0$ olurken, düz giderken $w_1(u, v)=0$ ve $w_2(u, v)=1$ oluyor. Sola dönüşü veya u dönüşü simgeleyen kenarlar grafikte gösterilmez. Bir yönden başka bir yöne giden kenar (güneyden batıya) sağa dönüşü simgeler. Bu grafiği yaratmak $O(mn)$ süre alacaktır.

Şimdi grafik gösterimimiz hazır. Sağa dönüşleri en aza indirmek ve daha sonra düz gidiş mesafesini kısaltmak durumundayız. Bir kenar için tek bir gerçek değerdeki ağırlık tutmak yerine, $(w_1(u, v), w_2(u, v))$ sayı çiftlerini tutuyoruz. İki ağırlığı, ilk önce ilk ağırlığa, daha sonra da 2. ağırlıklara bakarak kıyaslıyoruz. Başka bir deyişle, $w = (w_1, w_2)$ ve $w' = (w'_1, w'_2)$ için $w < w'$ olması için ya $w_1 < w'_1$ olmalı ya da $w_1 = w'_1$ ve $w_2 < w'_2$ olmalıdır.

Alternatif olarak, her bir kenarda tek ağırlıkla bu problemi ele alabiliriz : Sağa dönüşler $w > |E|$ (örnek, $4mn + 1$) ve düz gidilen köşeler 1 ağırlığında alırız. Grafikteki en kısa yol, her zaman en az sayıda sağa dönüşü sahip olacaktır. Çünkü sağa dönmek oldukça pahalı.

Bu w , yol ve kenar ağırlıkları gerçek sayılar yerine ikili olsa da, kenar ağırlıkları artı değerli olduğu için, Dijkstra'nın algoritmasını kullanabiliriz. Diğer bir deyişle, s 'den u 'ya ağırlığı $w(p)$ olan bir p yolu verildiğinde, bu p 'ye bir (u, v) kenarı ekleyerek elde edilecek p' yolunun ağırlığı hiçbir zaman daha küçük değildir; yani, $w(p') \geq w(p) + w(u, v)$. Dijkstra'nın algoritması $O(V \lg V + E)$ sürede çalıştığından, koşma süremiz $O(mn \lg mn)$ olacaktır.

Eğer düz gidilen yolları kısaltmak istemiyorsak ve sadece sağa dönüş sayısını azaltmak istiyorsak, öncelikli sıra işlemlerini $O(1)$ sürede desteklemesi için Dijkstra'nın algoritması için öncelikli sırayı düzenlemek mümkündür. Bu koşma süresini $O(mn)$ 'e azaltacaktır.

Bunu yapabilmek için basit bir yaklaşım, yol uzunluklarını her zaman tam sayı yapmak ve en uzun yol ağırlığının $4mn$ (artı düğümlerdeki ∞) olmasıdır. Yani, öncelikli sırayı $O(mn)$ bağlantılı liste gibi hazırlayabiliriz. Burada her bir bağlantılı liste, her gidilebilir yol ağırlığına denk gelmektedir.

Aslında, eğer genişlik önce benzeri bir algoritma kullanacaksak, gerçekte iki bağlantılı listeye ihtiyacımız vardır. Düz yollar için kenar ağırlıkları 0, sağa dönüşler için 1 olacak. 0 ağırlıklı bir kenarda keşfettiğimiz yeni düğümler bir listeye eklenir. 1 ağırlıklı kenarda keşfettiğimiz yeni düğümler ise diğer listeye eklenir. Bu algoritmanın başlangıçtaki varsayımlarını ve doğruluk ispatını okuyucuya egzersiz olarak bırakıyoruz.

Düz yollar için beraberlik olduğunu varsaydığımızda, bu sorunu çözmek için $O(mn)$ bir algoritma olduğunu görürüz. Eğer "tam bir doğruluk ispatlı" bir $O(mn)$ çözümü bulursanız, bu konuda bizi de bilgilendirin.

(b) a bölümündeki algoritmanız ile üretilen yol haritası çıktıktan sonra, polisin sizi takip ettiği durumda garajınıza kadar götürebileceğinizi düşünün. Takipten kurtulmak için en iyi stratejinin hızlı bir şekilde 2 sağ dönüş yapmak olduğunu düşünün. (örnek, komşu karelerde, 90 derecelik dönmeler 1 birim düz gitme ile kesilir)

Yani 2 tane ardışık sağa dönüş içeren ve hala sol dönmediğiniz, mümkün olduğunca az sağ dönüşle sahip, eşit sağa dönüşlü yollarda da en kısa düz yolu tercih edeceğiniz bir yol bulmak istiyorsunuz. Böyle bir yolu bulmak veya yol olmadığını belirlemek için verimli bir algoritma yazın.

Çözüm: Bir önceki problemdeki G grafiğinin $G_0=(V_0,E_0)$ ve $G_1=(V_1, E_1)$ kopyalarını içeren yeni G' grafiğini yaratırız. Eğer u 'dan V' 'ye 2 sağa dönüş ile gidilebiliyorsa, grafiğe $u_0 \in V_0$ 'dan $v_1 \in V_1$ 'e giden bir kenar ekleriz. Bu kenar $w_1(u_0, v_1) = 2$ ve $w_2(u_0, v_1) = 0$ ağırlıklarına sahiptir. Yeni grafik G' grafiğinde, s_0 'den t_1 ' en kısa yolu bulalım.

Kaynak sadece G_0 'da, gider de sadece G_1 'de olduğundan, en azından bir kez iki ardışık sağa dönüş yapmanız garantilidir. G' grafiğinin orijinal G grafiğine göre (yani $8mn$) iki kat fazla düğümü vardır. Kenarların sayısını ikiye katlar ve sonra iki ardışık sağa dönüşü temsil eden en çok $8mn$ kenar ekleriz (orijinal grafikteki her düğümden iki ardışık sağa dönüş yaparak en çok bir başka düğüme ulaşabiliriz). Bu nedenle Dijkstra's algoritmasıyla G' araması yapmak asimptotik olarak yine $O(mn \lg mn)$ süresini alır.

Problem 8-2. Video Oyun Tasarımı

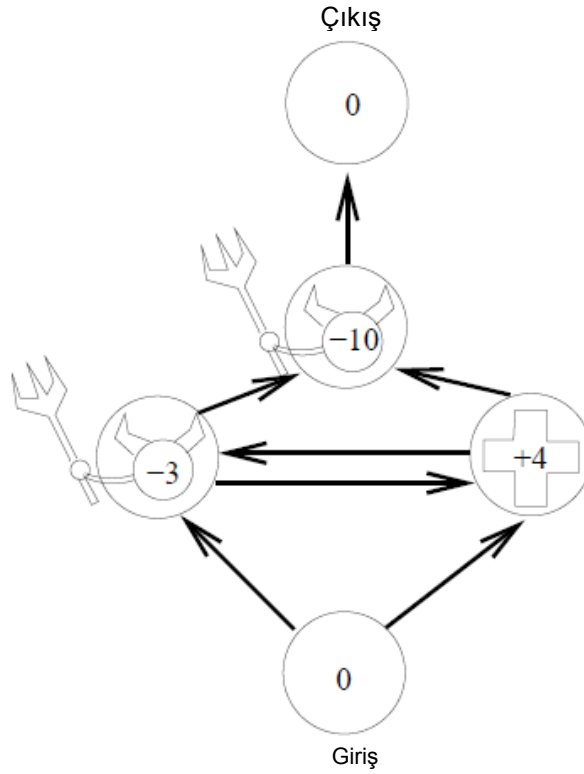
Profesör Cloud, bir senedir, uzun zamandır beklenen Take-home Fantasy XII isimli oyunun tasarımına danışmanlık yapıyor. Oyundaki seviyelerden birinde, oyuncular girişten çıkışa gitmek için bir labirentteki birçok odayı gezmek zorunda. Odalarda hayat iksiri veya canavar olabileceği gibi, odalar boş da olabilir. Oyuncu odalarda dolaşırken L yaşam puanları azalır veya artar. Hayat iksiri içmek L 'yi artırırken, canavarla karşılaşmak L 'yi azaltır. Eğer L , 0'ın altına düşerse, oyuncu ölür.

Figur 3' de gösterildiği gibi bu labirent, köşelerin odaları, tek yönlü kenarların da koridorları simgelediği bir $G=(V,E)$ yönlendirilmiş grafiği ile simgelenebilir. $f : V \rightarrow \mathbb{Z}$ köşe ağırlık fonksiyonu odanın içindekileri simgeler.

- Eğer $f(v) = 0$ ise oda boştur.
- Eğer $f(v) > 0$ ise odada hayat iksiri vardır. Oyuncu odaya her girdiğinde L yaşam puanı $f(v)$ kadar artar.
- Eğer $f(v) < 0$ ise odada canavar vardır. Oyuncu odaya her girdiğinde L yaşam puanı $|f(v)|$ kadar azalır. L eksi olursa oyuncu ölür.

Labirente giriş odası $s \in V$, çıkış odası da $t \in V$ ' dir. s ' den her v ' ye ve her v 'den t 'ye bir yol vardır. Oyuncu, s girişinden $L = L_0 > 0$ yaşam puanı ile girişten oyuna başlar.

Basit olsun diye girişin boş olduğunu düşünün: $f(s)=0$



Şekil 3 1-girilebilir labirent örneği.

Profesör Cloud canavarları ve hayat iksirlerini labirente rastgele yerleştirmek için bir program tasarlıyor, ama bazı labirentlerde girişten çıkışa gitmek $L_0 > 0$ olmak zorunda olduğundan mümkün olmayabiliyor. s' 'den t' 'ye giden yolda eğer oyuncu sağ kalabiliyorsa bu yol girilebilirdir.

Eğer $L_0=r$ ile başlayan bir labirentte girilebilir bir yol varsa, r -girilebilir labirenti tanımlayın.

Profesöre r 'nin en küçük değerini tanımlamak için verimli bir algoritma tasarlamasında yardımcı olun. Öyle ki, bu algoritma, verilen labirentin r -girilebilir olduğunu veya böyle bir r olmadığını belirlesin. (Gidiş yolundan puan almak için, bir labirentin verilen bir r için r -girilebilir olup olmadığı problemini çözün.)

(a) Şekil 3' teki labirent için güvenli bir yol bulun.

(b) Ağırlıkların kenarlarda olduğunu düşünerek soruyu denk bir probleme çevirin ve bu denkliği kanıtlayın.

Çözüm: Eğer giriş düğümü bir ağırlığa sahipse, 0 ağırlıklı yeni bir giriş yaratır ve bu girişten orijinal girişe bir kenar tanımlarız. Daha sonra, köşelerdeki ağırlıkları kenarlara taşırız. Eğer v düğümü $f(v)$ ağırlığına sahipse, bütün $(u,v) \in E'$ ler için $w(u, v) = f(v)$ işlemini yaparız. Denk olan problem kenar ağırlıklı bu grafikte, her alt yolun artı olduğu bir yol bulmadır.

(c) Bu problem için hiç bir hayat puanı arttıran devre olmadığını varsayın. Verilen bir r için, labirentin r -girilebilir olup olmadığını nasıl kontrol edersiniz?

Çözüm: Bunun için Bellman-Ford' un değiştirilmiş bir sürümünü kullanırız. Verilen bir r için, her bir u düğümü için, oyuncunun u ' ya erişmesi için sahip olacağı en fazla $q[u]$ puanı bulunur. Eğer $q[t]$ artı bir değer ise grafiğimiz r -güvenilirdir.

$u \in V$ olan her köşe için, $q[u]$ ' nun alt sınırı $p[u]$ hesaplanır. Giriş hariç bütün $p[u]$ ' ları başlangıçta $-\infty$ ' a atarız. Bellman-Ford algoritmasını koşturarak ve kenarları rahatlatarak $p[u]$ değeri $q[u]$ ' ya yakınsayana kadar artar. Burada, eksi değerli bir düğüme ulaşmanın, bu düğüme ulaşamamaktan iyi olması önemlidir. Yani, eğer $p[u]$ artı oluyorsa onu değiştiririz, aksi takdirde, $-\infty$ olarak saklarız. Rahatlama rutinini aşağıdaki gibi değiştiririz.

```
V-RELAX( $u, v$ )  (v-rahatla)
1  if ( $u, v$ )  $\in E$ 
2    then if ( $(p[v] < p[u] + f[v])$  and ( $p[u] + f[v] > 0$ ))
3      then  $p[v] \leftarrow p[u] + f[v]$ 
4   $\pi[v] \leftarrow u$ 
```

Bütün kenarlar V defa rahatlatıldıktan sonra, eğer eksi ağırlıklı hiçbir devre yoksa, bütün $p[u]$ ' lar karşılık gelen $q[u]$ ' ya yakınsar. (u köşesine erişirken sahip olunan en fazla puanların sayısı). Bu noktada eğer $q[t]$ artı değerli ise, oyuncu çıkışa artı hayat puanları ile ulaşabilir ve grafik r -güvenilir olur.

(d) Şimdi, hayat puanlarını arttıran bir devre olabileceğini varsayın. Verilen bir r için, labirentin r -girilebilir olup olmadığını nasıl kontrol edersiniz?

Çözüm: Eğer $p[t]$ artı değerli değil ise, bütün kenarları bir kere daha rahatlatırız. (Tıpkı Bellman-Ford' da olduğu gibi). Eğer herhangi bir düğümün $p[u]$ ' su değişirse, r puan ile s ' den ulaşılabilen artı ağırlıklı bir devre bulduk demektir. Yani, oyuncu devrede t ' ye ulaşmak için gerekli olan puana erişebilecek şekilde yeterli defa devrede dolaşır. Eğer artı ağırlıklı bir devre bulamazsak ve $p[t]$, $-\infty$ ise, grafiğimiz r -güvenilir değildir. Algoritmanın doğruluğu Bellman-Ford' un doğruluğu ile aynıdır ve koşma süresi $O(V E)$ ' dir.

(e) r -girilebilir bir labirentte, en küçük r değerini nasıl bulabilirsiniz.

Çözüm: Yukarıda verilen alt yöntemi kullanarak en küçük r ' yi buluruz. Önce grafiğin 1-güvenilir olup olmadığını kontrol ederiz. Eğer öyle ise 1' i cevap olarak döndürürüz. Eğer değilse, 2 ve 4' ü kontrol ederiz. i ' inci basamakta 2^{i-1} ' i kontrol ederiz. En sonunda 2^{k-1} ' in güvenilir olmadığı ama 2^k ' nın olduğu bir k buluruz. Yani, r ' nin en küçük değeri iki değer arasındadır. Bundan sonra $r=2^{k-1}$ ile $r=2^k$ arasında r ' nin tam değerini bulmak için ikili arama yaparız.

Analiz :

$k = \lceil \lg r \rceil$ iken, yapılan işlemlerin sayısı $k + O(\lg r) = O(\lg r)$ dir. Bellman-Ford' u $O(\lg r)$ sürede koşturmak zorunda olduğumuzdan toplam koşma süresi $O(V E \lg r)$ dir.