**DERS 17**

**En kısayollar** hakkında konuşacağız ve önümüzdeki üç ders enkısa yollar olacak. Böylece bu bir üçleme oluyor. Bugünkü, ''Enkısa Yollar Bir''. Hafta sonu, Star Wars Yıldız Savaşları filmlerinin bir sürü versiyonunu seyrettim. Dün matinesinde müzikalini seyrettim. Bir MIT müzikali idi. Eğlenceli idi, üç film birden yaklaşık dört saat sürdü. Biraz uzundu ve sonra, cuma günü, tek kişilik gösteriyi seyrettim. Tek kişilik Yıldız Savaşları: Üç orijinal film, bir saat içinde. O da, çok uzunun tersi oldu. Her ikisi de eğlenceli idi. Böylece üçlememi saptıyorum. Tüm Kısımlar. Önce, Yeni Ümit ile başlayacağız, enkısa yollar problemini göreceğiz ve onun özel bir durumunu çözeceğiz; çok enteresan bir versiyonunu.

Ve ilerledikçe, giderek genelleşen versiyonları ele alacağız. Enkısa yollar, geçen hafta gördüğümüz dinamik programcılığın bir çeşit uygulamasıdır; ve geçen hafta hırslı algoritmaları görmüştük. Böylece, bunu geliştireceğiz ve Alderon'dan, ne bileyim, Cambridge'e mümkün olan enkısa ve çabuk yoldan varmak gibi önemli bir problemin çözümü için bazı çok enteresan algoritmalar elde edeceğiz; tabii tüm bunlar bir grafik dünyasında yaşadığınız zaman geçerli.

Geometrik enkısa yollar var ki bunlar biraz daha güçtür. Biz, burada sadece grafiklerdeki enkısa yollara bakacağız. Şimdi, umarım hepiniz grafikte bir yolun ne olduğunu biliyorsunuzdur. Ama, hemen çok kısa bir özetleme yapayım, çünkü, weighted graphs yani ağırlıklı grafiklere bakacağız. Yani, normal yapılanmış bir yönlendirilmiş G grafiğimiz olsun; belli köşeleri ve kenarları var. Kenar ağırlıklarımız var, bu durumu daha ilginç kılıyor. Bu, her kenar üzerindeki gerçek sayı. Böylece,kenar ağırlıkları, bir w fonksiyonuyla veriliyor. Her kenar için, bir gerçek sayı elde ediyorsunuz.

Ve sonra,grafikteki yollara bakarak, bazı basit adlandırmalar kullanacağız, bir yol olarak adlandırılan yollar, p.. Belli bir köşeden başlıyor, ve belli başka bir köşeye gidiyor ve böyle devam ediyor. Son köşeye $v_x$ diyelim ve bunlardan herbirinin, digraph' ta, yönlendirilmiş bir kenar olmaları gerekiyor.Yani bu, bir yönlendirilmiş-ağırlık'tır. Buradaki kenarlara uyumlu olması lazım. Öyle bir yol'un ağırlığının, yol boyunca yeralan tüm kenarların toplamı olduğunu söyleyeceğiz. Ve buna, w(p) diyeceğiz. Bu toplam, **i**, 1'den k-1'e kadar değişirken, w( $v_i$, $v_{i+1}$). Sadece vurgulamak ve bunun ne kadar genellenebileceğini ifade etmek için; belli bir yolumuz var, belli bir köşeden hareket ediyor, yol boyunca kenar ağırlıkları var.

Hipotetik bir Grafikte, kendi seçmiş olduğumuz rastgele bir yol var. Bunları tekrarlamamızın nedeni, kenar ağırlıklarından bazıları,negatif olabilirler. Bazıları sıfır olabilir. Buradaki toplam eksi iki'dir; yani bu yolun ağırlığı eksi 2..

Ve muhtemelen grafik bundan daha büyük. Bu, grafikteki sadece bir yol. Genelde

köşelerde tekrarlayamayan, basit yolları düşünürüz; ama bazen tekrarlamalarına izin veririz. Bundan sonra önem verdiğimiz şey, belli bir enkısa yol veya herhangi bir enkısa yol. Elimizdeki tek en kısa yol değil ama buna rağmen, buna ‘’enkısa-yol’’ diyeceğiz.

Demek ki, belli bir A dan, belli bir B ye olan enkısa yolu istiyoruz. Örneğin u ile v köşeleri arasındakini.. İlaveten, bunun u’dan başlayıp v’ye gitmek şartıyla olabilecek en az ağırlıktaki, bir enkısa yol olmasını istiyoruz. Pekala, böylece aradığımız bunlar. Genel olarak, size bir u köşesi veririz, bir de v köşesini; en hızlı şekilde “enkısa yolu bul”, deriz. Bunun için iyi bir algoritma nedir? Gelecek üç dersimizin konusu bu..

Nispeten daha basit bir problem çözmeye çalışacağız, o da,yolun ağırlığını hesaplamak ki, esas itibariyle A ile B arasındaki mesafeyi hesaplamak olacak. Buna da, u dan v ye kadar olan enkısa yolun ağırlığı diyeceğiz. Bu ağırlığı, delta ile göstereceğiz, yani delta (u,v). Böylece enkısa yolun, veya enkısa her yolun ağırlığı bu oluyor. Başka bir deyim ile, herbir enkısa yolun ağırlığı üzerindeki minimum. Burada p, İngilizce ‘’path’’ kelimesinin ilk harfi olarak, bir ‘’yol’’ oluyor. Şimdi ne kadar çok yol olabileceğini düşünün. Prensip olarak sonsuz olabilirler, bilhassa köşeleri tekrarlamanıza izin verilirse.. Tüm bu yollara tabii ki, hipotetik olarak bakıyorsunuz. Asgari ağırlıktakini alıyorsunuz. Soru mu? İyi, takip edecek sorum, enkısa yollar ne zaman mevcut olmazlar,idi? Siz, böylece bir versiyona isabet kaydettiniz : Negatif-kenar-ağırlıkları olduğu zaman.

Prensipte, negatif kenar ağırlıklarınız olduğu zaman, bazı enkısa yollar, enkısa yol olmadığından mevcut olamayabilirler. Enkısa yollar olmaz; u dan v ye, enkısa yol yok.. Özellikle, eğer iki köşem varsa, u ile v, bunların arasındaki enkısa yolu istiyorsam ve negatif kenar ağırlıkları varsa, bu gene de iyi’dir. Zira,negatif ağırlığı olan bir yolu hala hesaplayabilirim demek istiyorum. Öyle ise, u dan v ye, bir enkısa yol olmaması durumu ne zaman ortaya çıkar? Siz devam edin. --- İyi.

Demek ki, bu yol boyunca bir yerde, bu kenarların hepsinin toplam ağırlığının negatif olduğu bir döngü varsa, oraya gidip, etrafında istediğim kadar dönerim. Ağırlık negatif olduğu için, ağırlığı azaltmaya devam ederim. Sabit bir değere düşürünce, o noktadan, v’ye gidebilirim.Öyle ise, u dan v ye ulaşabilen negatif ağırlıklar, döngüler olduğu sürece, enkısa yol yoktur; çünkü eğer, herhangibir yolu alırsam, birkaç defa daha fazla dönerek, onu daha kısa yapabilirim.

Öyle ise, bir anlamda bu gerçekten minimum değildir. Bu tip şeylerde fantaziyi seven kimseler için, buna minimum değil, “infimum” denir. Bu durumda bizim söyleyeceğimiz : **n’nin** delta(u,v)’si eksi sonsuz’dur. u dan v ye bir negatif ağırlık döngüsü var. Öyle ise, bir anlamda endişe etmemiz gereken şey işte bu durum. Ama hiç negatif ağırlık döngüsü olmadığı sürece, (u,v) nin deltası, eksi sonsuzdan daha büyük bir sınırlı değer olacaktır.. Bazen negatif ağırlıklarınız olabilir ama negatif ağırlık döngüsü yoktur; grafiğinizde herhangi saykıl olmayabilir.

Bu hala enteresan bir durumdur. Ve not etmenizde fayda olan şey, A dan B’ye, eksi sonsuz zamanda ulaşabilirsiniz.. Bu zamanda gezinme gibi bir şey olur; eğer ağırlıklarınız zaman bağlamında kullanılmışsa..

Enkısa yollar başka ne zaman oluşmazlar?  Gördüğümüz, sadece bir durum idi, daha basit bir durum daha var. Arada bir bağlantı yoksa.. u ile v arasında herhangi bir yol olmayabilir. Bu küme boş olabilir. u dan v ye hiç yol olmayabilir. Ne olacağını burada tanımlamalıyız ve eğer u dan v ye bir yol yok ise, “bu sonsuzdur” diyeceğiz.

Demek ki, bu istisnai durumlar, artı-sonsuz ile eksi-sonsuz var, bunlar çok sezgi gerektiren şeyler, çünkü eğer yol yok ise, u dan v ye varmak gerçekten çok uzun zaman alır. Pekala, tanım bu oluyor. Elbette, zamanın en büyük kısmını bu durum üzerinde durarak sarf ederiz. Genellikle,bu  sınırlanmış bir kümedir. Pekala, iyi, böylece, tanım bu.

Şimdi enkısa yollarla ilgili bazı yapısal özellikler elde edeceğiz ve bunlar bu yolları bulacak iyi algoritmalar kuramlamamızı mümkün kılacak.

Ve özellikle, dinamik programlamadan alacağımız fikirleri kulllanmak istiyoruz. Öyle ise, enkısa yolları çözmek için dinamik programlama kullanmak istersem, neye ihtiyacım olur? İlk olarak neyi kontrol etmeliyim?  Şimdiye kadar hepiniz dinamik programlama yaptınız, bu nedenle hepinize bir anlam ifade etmesi gerekiyor; en azından birkaç hafta öncesine göre geçen hafta öğrendiklerimizin çok anlam ifade etmesi gerekiyor. Dinamik programlama içinizde olgunlaşan bir şeydir. Ben her yıl, bir önceki yıla nazaran daha iyi anladığımı görüyorum.  Ama dinamik programlamayı, özellikle bu sınıfta öğrenmiş iseniz, kontrol etmeniz gereken bir güzel anahtar özellik var.  Efendim?  En iyi alt-yapı: doğru. Bu terimi aklınızda tutmalısınız.Tek başına dinamik programlamanın etkin ve faydalı olması yeterli değildir, ama en azından size onu kullanmayı denemeye yöneltir.

Çok zayıf bir beyan oldu ama dinamik programlamanın bir anlam ifade etmesi için bunu kontrol etmeniz gerekir; bu gerek şarttır. Burada eniyi alt-yapı demek,  bir en kısa yola baktığımda, onun alt-yolunun da en kısa yol olmasıdır. Sonlandığı kendi uçları arasında değil, altyolunu iki tarafında sonlandıran uçlardan  her hangi birinden çıkan yollar da enkısa yollardır.

Ancak, eğer ayrı yerlerdeki iki uç arasında sona eren bir enkısa yolunuz var ise, bunun herhangi bir alt yolu da enkısa yoldur. Bu izah ettiğim, eniyi alt-yapının sadece bir versiyonudur. Bu versiyon elimizdeki bu düzenleme için de doğrudur.

En iyi alt-yapı özelliğini nasıl ispatlayabilirim? Kesip yapıştırarak. Evet, bu işlem burada da çalışır. Demek istediğim bu her zaman geçerli değildir. Ama burada iyi bir tekniktir. Bunu bir resimle açıklayacağım.. En kısa yolunuzun belli bir alt-yapısı olduğunu varsayalım. Alt-yolun, x ten y ye olduğunu söyleyelim. Ve enkısa yolun da u dan v’ ye gittiğini farzedelim. (u,v) nin enkısa yol olduğunu kabul edersek, (x,y) nin de en kısa yol olduğunu ispatlamak istiyoruz.

Biran için, (x,y) nin enkısa yol olmadığını düşünün. Öyle ise, x’den y’ ye giden daha kısa yollar vardır. Ama, x den y’ ye bundan daha kısa yolunuz var ise, o zaman u dan v’ye bu en kısa yolun, bu kısmını silmeli ve daha kısa olan ile değiştirmeliyim. Örneğin bu, sanal bir daha kısa yol olsun; bunun var olduğunu farzedin. Eğer varsa,  x’den y’ye eski yolu kesip, x‘den y’ye bu yeni yolu yapıştırmalıyım.

Mutlak olarak daha kısadır. Böylece, u dan v ye, mutlaka daha kısa bir yol elde etmiş oluyorum. Ancak ben, u – v'nin enkısa yol olduğunu varsaydım: Çelişki. Öyle ise, daha kısa yol yoktur. Bu da, bizim enkısa yolların alt-yollarının da enkısa yol olduğuna dair ön kuramımızı doğrular. Bu şimdi oldukça tanıdık bir ispat tekniğidir;  kes ve yapıştırın başka bir versiyonudur.

Pekiyi, öyle ise bu, enkısa yolları hesaplamak için, iyi bir işaret… Dinamik proqramlama bağlamında;, burada dinamik  programlamaya  bakmayacağız, çünkü, daha güçlü olan hırslı algoritmaları hedefliyoruz. Ancak gelecek hafta  bazı dinamik programlama yaklaşımlarını göreceğiz. Sezgisel yaklaşımla, burada normal bazı alt-problemler var. Yani, u dan v ye giderken, u dan v ye enkısa yolun ne olduğunu bulmak istersem, bu özel bir problem oluşturur. Belki, u'dan, aradaki belli bir x noktasına, ve sonra oradan da, x'den u'ya, enkısa yolları hesaplamayı içerir.

Bu iyi gibi görünüyor. Karesellikte olduğu gibi, pek çok alt problem, ve $V^2$ düzeyinde alt problemler, bizi  dinamik programlamaya götürecek gibi görünüyor. Bunu çalıştırabilirsiniz ancak biraz şaşırtıcıdır. Gelecek derste göreceğiz. Fakat, bu ara nokta hakkında düşünürsek, **üçgen eşitsizliği** denen bir şeye ulaşırsınız. Belki üçgen eşitsizliğinin bir çeşidini bir vesileyle görmüşsünüzdür.Tüm geometrik alanlar için geçerli olmakla birlikte, enkısa yollar için de geçerlidir. Bu sonuncu durum, sanırım eğiliminize göre bazan az belirgin bazen de çok belirgindir. Böylece, eğer herhangi bir üçlü köşeniz varsa, u dan v ye enkısa yol, en çok u'dan x'e en kısa yol, artı x'den v'ye enkısa yoldur.

Tabii bunlar burada , u'dan x'e kadar enkısa yolun ağırlığı ile, x'den v'ye kadar enkısa yolun ağırlıklarını gösteriyor.. Bunun tanım gereği doğal olması lazım, ama eğer resmini çizerseniz daha da doğallaşır. Böylece belli bir köşe u var… Zigzaglı çizgiler çiziyorum ki kenarlarla karıştırılmadan yolların uzunluğunu göstersin. Belli bir x ara noktası var ve hedef noktası v var, ve bu üç enkısa yolu değerlendiriyoruz.

Bu, u'dan v'ye enkısa yol,veya onun ağırlığı.Bu u' dan x'e en kısa yol ,. Ve bu da  x'den, v'ye enkısa yol; ve bu da onun ağırlığı.. Konu, bunun enkısa yol veya en kısa ''tek'' yol olması,  tabii u'dan v' ye. Ve özellikle, böyle bir yol önce u'dan x'e, sonra da, x'den v'ye gittiğiniz durum. Demek istediğim bu toplam, bu özel yolu tamamen ölçüyor. Buradaki enkısa yolu alın, şuradaki enkısa yolu alın. Bunun da tüm yollar arasındaki minimum olması lazım.Böylece, bu özel yol, bu iki değerin toplamıdır.Bu tamam mı?. Bu resim ile ispatı. Açık mı?

Pekala, bunlar kolaydı.  Daha enteresan algoritma kümeleri içine girelim ve özellikle daha fazla heyecan verenlerini görelim.

Bugün, enkısa yolların özel bir versiyonunu, ''tek kaynaklı enkısa yol'' denen, enkısa yol problemini göreceğiz.Tabiatıyla  A den B ye gitmekten biraz daha genel bir problem. Problem şu: bir kaynak tepe noktanız ya da köşeniz var, o kaynak köşeden heryere nasıl ulaşabileceğinizi bilmek istiyorsunuz. Böylece, bu kaynak köşesine **s** diyoruz. Ve o kaynaktan, s'den, her yere olan enkısa yolların ağırlıklarını bulmak istiyoruz. Özellikle,enkısa yolları da bilmek istiyoruz ama bu da fazla zor değil.

Böylece bu,  delta s virgul v;  tüm v köşeleri için... Pekala bu pratikte ilk başladığımız, Alderon' dan - Cambridge' e ulaşma probleminden birazcık daha güç. Şimdi  Alderon'dan bütün evrene gitmek istiyoruz. Şimdi ve bu garip bir gerçektir ama, bugünkü bilgi düzeyimiz çerçevesinde en kısa yollar için birazdan söyleyeceğim şey, her zaman geçerli kalacak gibi görünüyor; gene de bilmiyoruz.

A dan B ye problemin çözümünde, verilen s, verilen t,  s den t ye gitme, bu problemden daha kolay değil. A dan B ye gitme problemini çözmek  için bildiğimiz en iyi yol, A dan, başka heryere gitme problemini çözmektir. Öyle ise, çözüm için bildiğimiz bu yolları kullanmamazlık edemeyeceğimiz anlaşılıyor.

Problem şaşırtıcı olduğu için, bu problem üzerindeki bir kısıtlamaya biraz daha fazla odaklanacağız.

Ana problemi gelecek derste çözeceğiz. Bugün, negatif ağırlıkları yasaklayarak, negatif ağırlık döngüsü konusundan kurtulacağız. Öyle ise, bütün kenar ağırlıklarının negatif olmadıklarını var sayıyoruz, u ve v tanımlı bütün köşeler için.  Böylece enkısa yollar, yolların bulunmaları şartıyla mevcuttur.

Bu eksi-sonsuzlar için endişelenmemiz gerekmiyor. (u,v) nin deltası, eksi sonsuzdan daima daha büyüktür. Eğer bir yol yok ise hala artı-sonsuz olabilir, ama bu çözümü daha da kolaylaştırıcıdır. Ve bugün ele alacağımız algoritma bu olguları  gerektiriyor.  Yani algoritma bu özellikler olmazsa çalışmaz.

Gelecek ders, bunlardan daha fantazi ve daha yavaş bir algoritma ile kurtulacağız. Böylece,  bugün kullanacağımız algoritmanın hırslı olduğunu, yani genellikle dinamik programlamadan daha hızlı olduğunu ana fikir olarak ima etmiş oldum. İşin şaşırtıcı yanı, hırslı algoritmanın çalışacağını ispatlamak olacak. Böylece, etrafta dolaşmak için, sadece bir tabii yol var, etrafta hırslı şekilde dolaşmak için diyelim. Belki bu fazla belirgin değil. O nedenle, biraz yapısı hakkında bilgi vereyim. Muhafaza edeceğimiz değişmez, her seferinde, kaynaktan her bir köşeye,  uzaklıklar konusunda tahmin ler yapacağız.

Uzaklık derken, enkısa yol ağırlığını kastediyorum. Ağırlık ve uzaklığı, burada daha sezgisel olması için, dönüşümlü kullanacağım.  Ve bilhassa, tahminlerin  doğru çıktığı köşe kümesini muhafaza etmek istiyorum. Böylece: bu, küçük s. Bu, büyük S. Yani, cevabın bulunduğu tüm köşelerin kümesi. Küçük s den, büyük S içindeki o köşeye giden enkısa yol nedir?  Böylece, başlangıç noktaları olarak, hangi uzaklığı biliyorum? Afedersiniz?  Küçük  s.

s'den s'ye enkısa yol uzaklığını biliyorum, çünkü,  eğer tüm ağırlıklarımın negatif olmadıklarını varsayıyorsam, s'den s'ye, birşey yapmamaktan  daha hızlı ulaşamam. Pekala, eğer bir negatif ağırlık döngüm varsa, belki s'den s' ye olan uzaklık, eksi-sonsuz olabilir. Ama, negatif ağırlıklarım olamaz, öyle ise s'den s'ye sıfır zamandan daha hızlı ulaşamam.  Hala sıfır maliyeti olan daha uzun bir yol olabilir, ama o da sıfırdan iyi olamaz.

Böylece, özellikle bunu biliyorum. Başlangıçta, şüphesiz küçük s büyük S'nin içindedir. Ve temel fikir, bildiğimiz köşeleri giderek toplamaktır. Demek ki, belli bir noktada bazı köşelere

olan uzaklıkları biliyoruz.Bu grup S, bu grafik de, büyük G. Yani tüm köşelerin alt-kümesi. Ve bundan dışarı giden bazı kenarlar var.

Ve böylece, bu köşelere nasıl ulaşılacağı konusunda tahminlerimiz var. Bazılarını daha görmedik bile. Bu S bölgesine bağlantılı bile olmayabilirler. Demek istediğim : doğrudan değil ama daha uzun bir yol ile bağlantılı olabilirler.Tamamen farklı şekilde bağlantılı bir bileşenin içinde olabilirler. Bunları henüz bilmiyoruz. Bazıları hakkında tahminlerimiz var, çünkü S den, onlara ulaşma yolunu biliyoruz. Ve temel fikir, burada bir köşe olan küçük s den diğer köşelere giden yol tahminleri arasında en küçük olanı alacağız.

Bu hırslı bir tercihdir. Ve o köşeyi, büyük S ye ekleriz. Böylece S bir köşe büyür. Her adımda, büyük S'ye bir köşe ekleriz. Şüphesiz gene bu tek değildir, V-S kümesinin içinde bir küçük v köşesidir. Dolayısıyla bu henüz hesaplanmamıştır ama, büyük S den uzaklığı minimumdur. Başka bir deyimle, büyük S'ye henüz eklemediğimiz bütün köşelere bakarız. Sadece en kısa mesafede olduğunu tahmin ettiğimiz bir köşeyi alırız.

Sezgimiz, bize bunun iyi bir tercih olduğunu söylüyor. Böylece gördüklerimin içinden küçük s'ye en yakın olanı, yani gördüğüm yollar içinden küçük s'ye en yakın yolu seçiyor isem, en iyileri bir biçime belirliyor gibi olurum. Ama,demek istediğim, belki görmediğim başka bir yol da vardır. Belki, buraya kadar gidiyor ve sonra gördüğümüz bir köşeye, başka bir yoldan gidiyorsunuz. Pekala endişemiz, enkısa yol demesem daha iyi olur, çünkü belki, oraya ulaşmak için başka bir yol daha vardır. Tamam, büyük S'ye birşey ilave eder etmez, o köşe için problemi çözmüş olduğumu açıklarım. Ve bu yanıtı sonradan değiştiremem.

Tahminler, büyük S ye eklenebilecekleri ana kadar değişebilir. Bu sebepten, mesela, S'ye bunu ilave etmek istemiyorum, çünkü bu yolu düşünmedim. Öte yandan, bütün ağırlıklarım negatif değillerse ve buradaki köşeyi alırsam, ve bu s'den tahmini en kısa yolsa, ... yani bunun enkısa olduğunu bir an için düşünelim, bu durumda bu daha-kısa bir yol olamaz, çünkü, s'den o köşeye uzaklık tahmini, s' den bu köşeye olana göre daha büyüktür.

Bu yüzden, yolu hem uzatacak hem de uzaklığı azaltabilecek hiçbir olanak yoktur. Bu sezgidir. Pekala burası biraz belirgin değil, çünkü tasarlanmış bir tümevarım hipotezim yok, bunu ispatlamak bir hayli çalışma gerektirecek. Bunun yapılması gereken doğru şey olduğu sezgimin söylediği şey. Ancak uzaklık tahminlerini ispatlamanız gerekir. Ama sezgisel olarak kendimizi iyi hissediyoruz. Bu iyi bir başlangıç noktasıydı.

Ayrıca, muhtemelen, uzaklık tahminlerini korumamız gerekiyor. Böylece algoritmanın en önemli niteliği, uzaklık tahminlerini güncelleştirmesi. Yani, büyük S'ye eklenecek en iyi köşeyi seçmek; bu bir adım. Ondan sonra, uzaklık tahminlerini güncellemek, bir çeşit asıl çalışmanın olduğu yerdir demek istiyorum. Görünüşe göre, sadece bazı köşelerin uzaklık tahminlerini güncelleyeceğiz, v 'ye bitişik olanlarını... v, S'ye biraz evvel eklediğimiz köşe idi. Böylece, S'ye bir şey eklediğimizde, S'yi bir parçacık büyütmüş oluyoruz, ondan sonra, S'den o köşeden çıkan yeni kenarların tümüne bakarız ve birşeyi güncelleriz. Ana fikir bu.

Böylece, hırslı algoritmayı kullanma şeklimiz bu. Şimdi size algoritmayı vereceğim. Bunun adı, Dijkstra. Dijkstra, geçenlerde vefat eden, Hollandalı meşhur bilgisayar bilimcisi. Ve bu da, muhtemelen onun en ünlü algoritması.

Algoritmanın başlangıcında ,biraz ilklendirme var; bu kısmı çok heyecan verici değil. Ancak, bazı değişkenlerin ne anlama geldiklerini anlatayım.

Pekala, d, köşeler tarafından endekslenmiş belli bir dizilim, ve anlamı, x'in d si, x için uzaklık tahminidir, yani, s'den x 'e.. Böylece, özellikle gerçek enkısa yol ağırlığını, büyük harf S olan grubumuza x ilave ettiğimiz zaman eşitleyecektir. Pekala böylece bu algoritmaya bir çıkış olacaktır.

Bir sorunuz mu vardı, yoksa sadece geriniyor muydunuz? Pekala.

Öyle ise, işimiz bittiğnde, d[x] çıktı'dır. Her köşe için bize s'den o köşeye enkısa yol ağırlığını verecektir. Yol boyunca s'den o köşeye belli bir uzaklık tahmini olacaktır. Ve onu zamanla geliştireceğiz. Bu sonsuz olacak...

Başlangıçta uzaklığı biliyoruz, s'den s'ye uzaklığın sıfır olduğunu biliyoruz. Böylece onu tahminimiz olarak belirteceğiz. Bu doğru olacak. Diğer her şeyi sonsuz olarak kabul edeceğiz, çünkü arada bağlantı olmayabilir. Başlangıçta fazla birşey bilmiyoruz. Büyük S başlangıçta sonsuz olacak. Büyük S ye, küçük s'yi ilave edeceğiz. Sonra buradaki ilginç kısım, Q , grafikteki tüm köşeleri başlangıçta tamamiyle içermesi.

Bu bir öncelikler kuyruğu olacak. Özellikle, en küçük uzaklık tahmininin yapıldığı köşeyi içerecek. Demek ki, bu bir öncelik kuyruğu.

Bu aslında veri yapısını adlandırmanın kötü kullanımıdır. Pekala bu bir küme, yığın olabilir. Köşeler, d yordamıyla anahtarlanmış.. bizim mesafe tahminlerimizle... Ve özellikle S Min Heap yani yığını olacak, s'de, enaz değeri olan şey. Diğer herşey başlangıçta aynı anahtara sahip. Bu Q nun içinden minimum elemanı defalarca çekip başka şeyler yapacağız. Pekala, böylece bu ilklendirme. Buna ilklendirme diyeceğim. Çok basit bir şey. Sadece doğrusal zaman alıyor, karmaşık hiçbir şey yok.

Algoritmanın özü altı satır. Ve bu; aslında bir adım değil. Burada ilk adımda yapmamız gereken şey, tüm köşeler arasında uzaklık tahmini minimum olan köşeyi almak. S, şu anda boş. Her şey Q 'da. Genel olarak, Q, herşeye sahip olacak, tabii S hariç. Böylece, o öncelikler kuyruğu içinde minimum anahtarı olan u köşesini alacağız.

Yani Q dan Min' i çıkarıyoruz.

Pekala, S 'ye küçük u'yu ilave edeceğiz, İddia, tamamen burada söylediğimiz. S' ye ilave ettiğimiz köşe, minimum uzaklık tahmini olanı. Ve şimdi uzaklıkları güncellememiz gerekiyor. Böylece, bu işlem için u için bitişik listede yer alan her v için, her bir bitişik köşeye bakacağız. Bir takım uzaklıklara bakacağız. Aşağı yukarı, Algoritmanın hepsi bu kadar. Anahtar bu.

Burada ne olduğunu biraz tanımlamam lazım. Geçen sefer esas itibariyle, yönlendirilmemiş grafikten bahsettik. Burada yönlendirilmiş grafiklerle ilgileniyoruz.

Burada, u için komşuluk listesi sadece, "u'dan v'ye kenarı olan köşelerin hepsini bana ver" demek olacak. Böylece bu, gidenlerin komşuluk listesi, gelenlerin komşuluk listesi değil. Yönlendirilmemiş grafiklerde herşeyi listelersiniz. Biz burada sadece Yönlendirilmiş

grafiklerle meşgul olacağız. Bunun söylediği herbir kenar (u,v) için, v için şimdiki tahmini ve bu aday tahmini karşılaştırmamız gerektiği; ki bu sezgisel olarak, S'den u' ya gidersiniz demek olur. Bu d[u] , çünkü şimdi artık bunun doğru cevap olduğunu biliyoruz.

Algoritmanın doğru olduğunu varsayarak, bunun, s'den u'ya enkısa yol ağırlığı olduğunu ümit ediyoruz, çünkü u'yu S'ye demin ekledik. Ve ne zaman S'ye birşey eklersek, değeri doğru olmalı. Böylece, "s'den u'ya enkısa yolu alır, sonra da bu kenarı u'dan v 'ye takip edersiniz" diyebiliriz. Bunun ağırlığı w(u,v)'dir. Bu S'den v'ye bir olası yoldur. Ve bu, mevcut tahminimizden daha kısa bir yolsa, yani bu şundan daha kısa ise, bu durumda tahminimizi bu toplam olacak şekilde güncellememiz gerekir çünkü bu daha iyi bir yoldur ve bunu yollarımızın veritabanına ekleriz: Oldukça sezgisel bir işlem; açıkçası, kötü bir yanı olmamalı. Demek istediğim bunların anlamlı yollar olmaları lazım. Birazdan böyle olduklarını ispatlayacağız.

Bu doğruluğun birinci kısmı olur, hiçbir zaman yanıltmaz. Sonraki şaşırtıcı tarafı ilgilendiğimiz tüm yolları bulması gerektiğini gösterebilmektir. Bu kısmın adı relaxation yani "gevşetme" ya da "dengeye dönme" adımıdır. Gevşeme, MIT öğrencilerine öğretilmesi daima güç olan bir tekniktir. Bu pek olağan gelmez. Ama çok kolay bir işlemdir. Optimizasyon terminolojisinden, programlama terminolojisinden gelir diyebiliriz.

Bu eşitsizlik, özellikle bu şekilde yazmaya başladığınızda size hiç tanıdık geliyor mu? u'dan v'ye belli kenar üzerinde, S'den v' ye en kısa yol ve s'den u' ya enkısa yol dediğinizde, bu gördüğümüz herhangi birşeye benziyor mu? Aslında bu tahtaya yazılıydı ama biraz önce sildim. Evet, üçgen eşitsizliği, doğru. Yani bu, üçgen eşitsizliğini doğrulamaya çalışıyor. Elbette, S'den v ye enkısa yolun, küçük-eşit,olması gerek; daha büyük değil. s'den u'ya enkısa yol, artı u'dan v'ye herhangi bir yolun, en kısa yol olarak en çok bu olması lazım.

Böylece bu, bir dereceye kadar genel anlamda üçgen eşitsizliği olur. Ve biz bunun doğru olmasını istiyoruz. Yani, doğru değil ise, doğru hale getireceğiz. Daha büyük ise eşit kılacağız. Ama daha küçük yapmak istemiyoruz, çünkü bu her zaman doğru değildir. Ama kesinlikle küçük- eşit olmalı. Bu üçgen eşitsizliğini düzeltmektir. Sınırlamayı daha gerçek hale getirmeye çalışmaktır.

Optimizasyon yani en iyi kılma dilinde, buna sınırlamayı gevşetme denir. Yani burada üçgen eşitsizliğini bir çeşit gevşetiyoruz. Bu işlemin sonunda tüm en kısa yolları elde etmemiz gerekir. Bu bir iddiadır. Böylece: çok basit bir algoritma. Grafik üzerinde deneyelim, bu şekilde çalışırsak daha sezgisel olur ve böylece dersin kalan kısmında neden çalıştığını ispatlayabiliriz. Evet, burada yeterli yer var. Ama bir dakika, burada bir şeyi daha belirtmem lazım. Af edersiniz. d[v]'yi her değiştiğimizde, bu öncelik kuyruğunda, v'nin anahtarları da değişmektedir. Yani, buradaki atamada varsayılan şekilde oluşan, -- burası biraz karışık ama, değeri azalan bir anahtarlama işlemidir; buna geçen derste, minimum spanning trees yani en az yayılan yada kapsayan ağaçlar bağlamında kısaca değinmiştik ve orada da anahtarları küçültüyorduk.

Vurgulamak istediğim, bu öncelikler kuyruğundaki bir elemanın anahtarını bu gevşetme adımında değiştiriyoruz ki o bir minimum olsun ve buraya çıkarabilelim. Ve sadece

devamlı olarak anahtarları azaltıyoruz, çünkü daima daha büyük değerleri daha küçük değerler ile değiştiriyoruz. Aslında koşma süresini çözümlerken bu konuya tekrar döneceğiz. Ama burada bir tür veri yapısı çalısması sürüyor. Gene simgelemi biraz sulandırıyoruz.

Pekala, işte kenar ağırlıklı bir grafik. Ve öncelikler kuyruğumu burada istiyorum. Ve tahminlerimi de çizeceğim. Şimdi hile yapmak istemiyorum. Böylece algoritmayı bu grafik üzerinde çalıştıracağız. s büyük A olacak ve A dan her yere enkısa yolu bilmek istiyorum. Böylece kontrol edebilirsiniz; pekala, yollar mevcut. Öyleyse sonunda hepsinin kesinleşmiş bir değeri olmasını umuyoruz. Tüm ağırlıklar negatif olmadıklarından, bu algoritmanın çalışması lazım. Bu algoritmanın bağlantılılığa bile ihtiyacı yok ama bütün ağırlıkların negatif olmadığı anlamı var. Böylece algoritmayı işletmeye başlıyoruz. Başlangıçta, kaynağımızın uzaklık tahminini sıfır olarak saptadık, çünkü A'dan A'ya sadece bir yol var ve burada iş yapılmıyor; bu boş yol. Bu nedenle sıfırın anahtarını buraya koyuyorum. Ve diğerlerinin hepsi için sonsuz koyuyoruz, çünkü bu evrede daha iyisini bilmiyoruz.

Bu yüzden geri kalanlar için sonsuz anahtarları koyacağım. Böylece şimdi algoritmanın yaptığının bu kuyruktan minimumu çıkarmak olduğunu görebilirsiniz. Kurulumumuz gereği, kesinlikle s'yi, veya bu durumda A'yı seçeceğiz. Böylece bunun ağırlığı sıfır. Diğer her şeyin birazcık daha fazla ağırlıkları var. Yani, s'ye, bakıyoruz, veya burada A'ya..Yani, A'ya bakıyoruz. S ünitemize A ilave ediyoruz. Böylece bu kuyruktan ayrılmış oluyor. Kuyruğa bir daha hiç dönmeyecek çünkü kuyruğa asla birşey eklemiyoruz, tüm köşelerle başlıyor, içinden çıkarıyor ve anahtarları azaltıyoruz. Ama hiç araya yerleştirme yapmıyoruz. Böylece A gitti. Şimdi tüm diğer köşelerin anahtarlarını güncellemek istiyorum. Burada sav: Sadece, A'dan gelen kenarı olan köşelere bakmaya ihtiyacım var. Burada A'dan B'ye bir kenar var ve ağırlığı on. Ve bir karşılaştırma yapıyorum : A'dan A'ya gitmek, iyi bir fikir çünkü hiçbir şeye mal olmuyor ve sonra bu kenar,-- A-B kenarı boyunca gitmek ise on' a mal oluyor.

Bu oldukça güzel bir fikre benziyor, çünkü bunun toplam ağırlığı sıfır artı on, yani on eder, ki bu da sonsuzdan çok daha küçüktür.Öyle ise bu sonsuzu silip on yazıyorum ve bunu kuyrukta da yapıyorum. Bu azaltılmış anahtar operasyonudur. Böylece şimdi, A'dan B'ye bir yol biliyorum. Güzel. A' dan C'ye olan, diğer tek kenar. Sıfır artı üç, sonsuzdan küçüktür öyleyse bu da iyi. C için buraya üç koyacağım ve C de orada..

Pekala, diğer köşelere dokunmuyorum. Onları buraya yeniden yazacağım ama, algoritma onları kopyalamak zorunda değil. O anahtarlar önceden de oradaydılar. Sadece bu ikisine temas ediyor. Tamam şu ana kadar bayağı sıkıcı idi. Şimdi kuyruğumuza bakıyoruz ve minimum elemanı çıkarıyoruz. Böylece A artık orada değil ve buradaki minimum anahtarın değeri üç.. Burada iddiamız, bunun enkısa bir yol olduğu; A'dan C'ye; A'dan C'ye en kısa yol işte bu. Bundan daha kısa yol yok. Bunu kontrol edebilirsiniz ve birazdan bunu ispatlayacağız.

Çok güzel, böylece C' yi listeden çıkaracağız. Gitti. Sonra C'den çıkan bütün kenarlara bakıyoruz. Bir tanesi B'ye kadar gidiyor, ağırlığı dört ; dört artı üç, A'dan C'ye enkısa yolun

ağırlığı. Böylece, önce, A'dan C'ye, sonra da C'den B' ye gidiş, üç artı dört yani yedi eder ve bu on'dan küçüktür. Öyleyse, B'ye varmak için daha da iyi bir yol bulduk. Böyle gitmek, öyle gitmekten daha iyi. Bu nedenle, B'ye yedi yazarız ve C'den çıkıp D'ye giden bir kenar var ve ağırlığı sekiz.

Üç artı sekiz 11 dir. 11, son kontrol ettiğimde sonsuzdan daha küçüktü. Öyle ise, D için 11 yazarız. Sonra E'ye bakarız. Elimizde üç artı iki, yani beş var ve bu sonsuzdan küçük.. Bu nedenle E'nin yeni anahtarı olarak beş yazarız. Bu evrede heryere, sonsuz olmayan en kısa yollarımız var ama bunlar en iyileri olmayabilirler. Öyle ise, bakmaya devam etmeliyiz. Pekala, algoritmanın ikinci turunda bunların arasından minimum anahtarı çıkarırız.

Pekala bu B değil, daha önce gördüğümüz ve belki de cevabını bildiğimiz gibi.. Ama, bu E dir. Evet, E, en küçük anahtara sahip. Şimdi bunu enkısa bir yol olarak ilan ederiz. E'ye varış için şu yolu takip ettik. A'dan C'ye, C'den E'ye; bunun enkısa olduğunu söyleriz. Böylece E ile işimizin bittiğni farzederiz. Ama hala güncellemeyi yapmamız lazım. Ya E'den çıkan kenarlar ne olacak? Burada sadece bir tane var. Onun maliyeti beş artı dokuz, yani 14 eder ki, bu 11 den daha büyüktür.

Öyle ise yürünecek yol değil. İlginç bir yol değil. Maliyeti 11 olan bir önceki yolumuz şimdi üzerinde düşündüğümüz yoldan daha iyi. Ben yolun tamamını çiziyorum ama algoritma sadece bu iki rakamı toplar. Pekala, güzel..Böylece,hiç bir şeyi değiştirmiyorum. 7, 11 -- ve 5 çıkarıldı veya E çıkarıldı. Yeni anahtarlarımız 7 ve 11.Öyleyse anahtar yediyi, buraya alıyoruz; bu B elemanı, B köşesi için...

A'dan B'ye elimizdeki yolu, ki bu oluyor, en kısa yol ilan ederiz. Algoritma, aslında bunu söyleyemez ama buna rağmen biz bunu çiziyoruz. Bu A- C- B yolu, en kısa yol olmaya adaydır. İddiamız, bunun gerçekten de en kısa yol olduğudur. Şimdi çıkan tüm kenarlara bakıyoruz. Bir tane,C'ye geri giden var, yedi artı bir yani sekiz maliyetle; bu üçten büyük ve bu da iyi. C'nin bittiğini zaten söylemiştik. Fakat algoritma da bu yolu kontrol eder ve "Oo, bu daha iyi değil." der. Sonra B'den D 'ye giden diğer kenara bakarız. O, yedi artı iki yani dokuz maliyete sahip, ki bu da 11 den iyi.

Böylece aslında daha da kısa bir yol bulduk. Böylece, en kısa yolun ağırlığı şimdi D için dokuz, çünkü, A- C- B- D' ye, toplam maliyet olarak, üç artı dört artı iki yani dokuza giden bu yol var. Muhteşem, şimdi kuyrukta sadece bir eleman bulunuyor. Onu çıkarıyoruz. D'den çıkan kenarlara bakıyoruz. Buraya giden bir tane var, maliyeti dokuz artı yedi, yani16 ediyor ki beşten bayağı büyük. Böylece işimiz bitiyor. Başka birşey yapmıyoruz. Bu noktada kuyruk bomboş. Ve iddia, burada yazılmış olan sayıların hepsi, son değerleri, en kısa yol ağırlıklarıdır. Bu beşe çok benziyor ama aslında S. Ağırlığı da sıfır. Bütün en kısa yolları da buraya çizmiş oldum.

Ve bunu yapmak zor birşey değil. Bu sınıfta üzerinde fazla konuşmayacağız ama, ders kitabının sonunda bundan biraz daha detaylı bahsediliyor. Adına da en kısa yollar ağacı deniliyor. Enkısa yolları pratikte hesaplamak isterseniz, bilmenizde fayda olan bir konu. Bu sınıfta biz daha çok ağırlıklar üzerinde duruyoruz, çünkü bu aslında aynı problem. En kısa yollar ağacı, en kısa yolların birlikteliğidir.

Ve özellikle, eğer grafiğinizdeki her köşeye bakarsanız ve eğer u köşesi gibi tüm köşeler arasında gevşetilmiş olan bu köşeye gelen en son kenarı düşünürseniz, ( u,v) kenarlarına bakıp "gevşeyecek sonuncu kenar bu muydu?" diye sorarsınız. Öyleyse, burada son gevşettiğimiz kenarlara sadece bakın. Hepsini biraraya getirin: İşte bunun adı en kısa yol ağacıdır. Ve bunun özelliği, s'den her yere ağaç boyunca tek özgün yol olmasıdır..

Ve bu da en kısa yoldur. Bizim bulduğumuz en kısa yol. Pekala, öyleyse açıkca belirtilmiş olmamakla birlikte, bu algoritmadan pratikte en kısa yollar elde ediyorsunuz. Burada üzerinde konuştuğumuz ana konu da en kısa yolların ağırlıkları..Bu konuda algoritma açık mı? Kendiliğinden doğru şeyi yaptığını hissediyor. O sayıların hepsinin en iyi yollar olduklarını kontrol edebilirsiniz. Ve şimdi bunu ispatlayacağız. Yani: Doğruluğu.

İlk ispatlamam gereken şey gevşetmenin hiç hata yapmayacağı. Eğer, d[v]'yi bir değer olarak saptarsa, d[v]'nin, delta üzerinde daima bir üst sınır olduğunu ispatlamak istiyorum. Elimizde bu değişkenimiz var. Bu, Delta (s,v)'ye tüm v değerleri için büyük- eşit. Ve bu değişmez her zaman geçerli. Ama ilklendirme sonrasında-- öncesinde değil, çünkü o zaman d daha tanımlanmış değil.

Fakat, bu ilklendirme işleminde s'yi sıfır alır ve diğer herşeyi sonsuz yaparsınız --ve gevşetme adımları dizilerinden herhangi birini ele alırsınız, o zaman bu değişken, her gevşetme adımından sonra geçerliliğini koruyacaktır. Aslında bu çok genel bir önkuramdır. İspat edilmesi de oldukça kolaydır. Sadece Dijkstra algoritması için değil, göreceğimiz diğer pek çok algoritma için de geçerlidir. Göreceğimiz pekçok algoritma gevşetme işlevini içerecektir. Ve bunun anlamı hangi gevşetme işlemlerini yaparsanız yapın, sonucunda elinizdeki mantıklı kestirim gerçek en kısa yollar ağırlığına büyük - eşit olacaktır. Yani, bu yukardan yakınsamaktadır. Böylece, önkuram bu. Şimdi ispatlayalım. Bu önkuramı nasıl ispatlamamız gerektiği konusunda bir öneri var mı?

Hangi tekniği kullanabiliriz? Ne dediniz? Kes ve yapıştır mı? En iyi altyapı için iyi olabilir. Kes ve yapıştır burada olanlarla benzeşiyor ama tam olarak değil. Biraz daha genel bir şey. Burada sadece sezginizi kullanın; doğru cevap olması gerekmez. Gerçekten, pek çok yanıt doğrudur ve ciddi ispatları vardır. Tümevarım, evet. Böylece buraya tümevarım yazmayacağım ancak, burada tümevarım kullanıyoruz. Beklediğim cevap buydu. Demek ki burada zaman içinde devam eden bir tümevarım var. Başlangıçtan sonra geçerli olmalı diyoruz. Orası, bizim taban durumumuz. Ve sonra yapacağımız her gevşetmeden sonra da hala gerçerli olmalı. Yani tümevarımla, önceki bütün gevşetmelerin çalışmış olduğunu varsayacağız, sonra da son gevşetmenin, bu her neyse, çalışacağını isptlayacağız.

Önce taban durumunu yapalım Yani bu işi başlatmadan sonraki ilk durum, buna başlangıç diyelim. Demek ki başlangıçta, d[s] eşit sıfır. Ve, d[v] bütün diğer köşelerde sonsuza eşit; tüm v'lerin küçük s'ye eşit olmadığı köşeler için.. Şimdi bu eşitsizliğin korunduğunu kontrol etmeliyiz. Elimizde, Delta (s virgül s) var. Onun sıfır olduğunu zaten tartışmıştık. Yalnızca negatif olmayan kenar ağırlıkları bulunduğunda, negatif elde edemezsiniz. Bu en iyisidir. Bu nedenle, sıfır elbette sıfıra büyük - eşittir. Ve, diğer her çeşit şeyimiz var. Yani, Delta (s, **v)** elbette sonsuza küçük - eşit. Demek ki gevşetme

mevcut. Herşey, sonsuzdan daha küçük veya ona eşittir. Yani bu geçerli. Herşey sonsuza küçük – eşit. Böylece, taban durumu bitmiş olur. Öyleyse şimdi tümevarım yapalım.

Ve bunun kanıtlamasını çelişki yöntemini kullanarak yapacağım. Şimdi, bir an için bunun bir noktada başarısız olduğunu varsayalım. Yani, değişmezin ihlal edildiği çelişkisinin olduğunu farzedin. Bu durumda buna sebep olanı izlemek ve çelişkiyi bulmak istiyoruz. Yani değişmez ihlal edilecek. Öyleyse ilk ihlali, ilk kez ihlal edildiği anı ele alalım. Bu gene temelde tümevarımla kanıtlamadır. Diyelim ki bir ihlal var ve d[v], Delta (s, v)'den küçük. Bu en kısa yoldan bir şekilde daha küçük bir tahmin olur ki bu kötü bir şey. Tamam, bu durumda ilk ihlale bakmayı düşünürüm, çünkü diğer bütün değerlerin tümevarımdan doğru olduğunu biliyoruz.

Demek ki d[v], hata yaptığımız ilk şey oldu. Yani değişmez diğer her yerde geçerli kalıyor. Pekiyi bu ihlale ne sebep olmuş oluyor? d[v]'deki belli bir gevşetme. Yani elimizde belli bir d[v] vardı ve biz bunu başka bir d[u] artı u'dan v'ye olan kenarın ağırlığıyla değiştirdik. Ve bu işlem onu geçersiz kıldı. Yani, d[v] bir şekilde şundan daha küçük. Biraz önce d[v]'yi bu olarak belirledik.. Öyleyse bu, Delta (s, v)'den daha küçük olmalı. İddiaya gore, bu mümkün değil, çünkü -- buraya yeniden yazayım.

Elimizde, d[u] artı w(u,v) var. Ve başka bir köşenin u'su üzerinde geçerli tümevarım hipotezi var. d[u]' nun en az Delta (s, u) olduğunu biliyoruz. Öyleyse bunun en az Delta (s,u) artı w (u, v) olması lazım. Şimdi, w(u, v) nedir? Bu u'dan v'ye giden bir yol. Yani, enkısa yoldan daha uzun veya ona eşit olması lazım. Yani bu tabii ki Delta (u, v)'ye büyük veya eşit. Tamam, daha küçük toplam ağırlığı olan çok-kenarlı bir yol varsa daha büyük olabilir, ama Delta (u, v)'den kesinlikle daha küçük olamaz.

Ve bu iyi bir toplama gibi gözüküyor, Delta s'den u'ya ve u'dan v'ye bu --- bir üçgen eşitsizliği, evet. Ancak burada terse çevrilmiş. Ama üçgen, s'den u'ya ve u'dan v'ye ise, bu s'den v'ye olan mesafeden daha uzundur. Tamam, elimizde bu şey var; s'den v'ye en kısa yol ağırlığına büyük - eşit, ama aynı zamanda s'den v'ye en kısa yol ağırlığından da kesinlikle daha küçük. Bu bir çelişkidir. Belki de çelişki yöntemi takip edilmesi gereken en sezgisel yol değil. Buradaki sezgi, d[v]'ye hangi değeri yüklerseniz yükleyin, aklınızda bir yol var. s'den u' ya tümevarımsal bir yolunuz vardı. Sonra bu kenarı eklediniz. Böylece o gerçek bir yol idi. Başka bir yolun ağırlığının enkısa yoldan daha büyük - eşit olduğunu her zaman biliriz. Öyleyse bu doğru olmalı ve işte tümevarımsal kanıtı.

Pekala devam ediyoruz, bu kolay bir ısınma idi. Daha büyük veya eşit durumu var. Şimdi, algoritmanın sonunda daha küçük – eşit olduğunu ispatlamamız lazım. Bu her zaman doğrudur; daha küçük veya eşit, sadece en sonunda doğru olacak. Bu sebeple küçük veya eşit olmasını şimdilik kanıtlamayacağız. Başka bir önkuramı ispatlayacağız ve bu iki önkuram, diğer algoritmalar için de yararlılar. Yani burada daha sonra uygulayabileceğimiz bir tür enkısa yollar teorisi geliştiriyoruz. Bu sizde gevşetmenin neden sadece kötü olmadığı, tam tersine gerçekten iyi olduğu konusunda bir sezgi oluşturacak. Gevşetme, bir şeyi bozmamakla kalmaz, aynı zamanda birazdan göreceğimiz anlamda da onu geliştirir.

Şimdi, s' den herhangi bir köşeye enkısa yolu bidiğinizi varsayın. Pekala, öyleyse, s'den diğer köşelere gidiyorsunuz. Sonra u'ya gidiyorsunuz. Sonra v'ye gidiyorsunuz. Bunun

s'den v' ye enkısa yol olduğunu düşünün. Pekala ayrıca, d[u]'daki s'den u' ya enkısa yol ağırlığını da bildiğimizi farz edin. Yani böyle bir eşitliğimiz olduğunu düşünün. Şimdi, daima bir büyük - eşit durumu olduğunu biliyoruz. Yani enkısa yolun üzerinde, v'den hemen önceki u köşesi için bunların eşit olduğunu düşünün. Şimdi, o kenarı, (u,v)'yi gevşettiğimizi düşünün; aynı bu adımda olduğu gibi.

Bu, (u,v) kenarını gevşetmedir. Ama, burada buna sadece gevşetme diyeceğiz. O gevşetmeden sonra d[v], Delta (s, v)'ye eşit oluyor. Yani eğer u için doğru cevabı biliyorsak, (u,v) yi gevşettiğimizde, v için aldığımız yanıt da doğrudur.. Bu iyi haber. Bunun anlamı, eğer u için tümevarım yoluyla bir şekilde doğru yanıtı elde edersek, v için de nasıl doğru cevap elde edeceğimizi biliyoruz demektir. Algoritmada enkısa yolda v'den bir önceki köşeyi pratikte bilmiyoruz, ama çözümlemede onu bilebiliriz. Bu nedenle, bu önkuramı ispatlamamız lazım. Bu, bir öncekinden daha da kolay: bunda, tümevarıma bile ihtiyaç yok, çünkü sadece gevşetme üzerinde çalışırsınız ve sonuç doğrudur.

Başlıyoruz. Delta (s, v) ile ilgileniyoruz. Ve enkısa yolun ne olduğunu biliyoruz. Yani, enkısa yol ağırlığı bu yolun ağırlığıdır. Öyleyse, buraya bir eşitlik yazabiliriz. Bunu yaparken, yolun ilk kısmı ile son kısmını birbirinden ayıracağım. Böylece, yolun s'den u'ya kadar bu kısmının ağırlığı,-- artı bu u,v kenarının ağırlığı.

Unutmayın, bir yolun w' sini yazabilirdik ve w, tüm o kenarların toplam ağırlığıdır. Öyleyse bu nedir? Yani yolun s'den u'ya olan ağırlığı.. Yada, değerin ne olduğunu anlamak için hangi özelliği kullanmalıyım? Efendim? s'den v'ye gidenin enkısa yol olduğunu, değil mi? Öyleyse, eniyi altyapı ile, s'den u'ya giden de bir enkısa yoldur. Böylece bu, Delta(s,u)'dur. Güzel. Şimdilik duralım. Söyleyeceklerimizin hepsi bundan ibaret. Öte yandan, bu önkuramdan ne yaparsak yapalım d[v]' nin Delta (s,v)'ye büyük – eşit olduğunu biliyoruz.

Öyleyse bunu yazalım. Bir kaç durum var ve bu, o durumlardan bazılarını ortadan kaldıracak. Bu birinci önkuramın doğruluk ilkesi çerçevesinde, d [v]'nin, Delta (s, v)' ye büyük - eşit olduğını biliyoruz. Öyleyse her zaman ya eşit ya da büyük. Bu sebepten, bu (u,v) hakkında gevşetmeyi yapmadan önceki anı düşünüyorum. Yani o evrede, elbette bu doğru… Öyleyse, gevşemeden önce ya bunlar eşittir, veya d[v] daha büyüktür.

Eğer gevşetmeden önce eşitlerse, bundan memnun oluruz, çünkü gevşetme değerleri sadece birinci doğruluk önkuramı bağlamında bunu azaltır. Bundan daha küçük hale gelemez ve gevşetmeden sonra da eşit olacaktır. Öyle ise, bu durumda işi bitirmiş oluruz.Yani bu önemsiz olan durum. Şimdi, d[v]' nin gevşetmeden önce Delta (s, v)'den daha büyük olduğunu düşünelim. Bu durum da şüphesiz geçerlidir. Şimdi bunu halletmeyi umuyoruz. Pekala, burada bu Delta (s, v)'yi biliyoruz. Işte bu toplamdır... Ayrıca, bunu da biliyoruz. Yani, Delta (s, u)'nun,d[ u] olduğunu biliyoruz.

Ve bir de bu w (u, v) var.. Böylece, Delta (s, v), d[u] artı w (u,v) olur, çünkü s'den u'ya giderken bu enkısa yol yapısının olduğunu varsayıyoruz ve sonra (u,v) kenarını takip ediyoruz. Yani, bunu biliyoruz. Ayrıca, d[v]' nin, d[u] artı w(u, v)'den de daha büyük olduğunu biliyoruz. Allahtan gevşemedeki koşul bu. Böylece gevşetmenin burada bir şey yapıp yapmadığını kontrol ediyoruz. Şurada eğer yanlış mesafe tahmininiz varsa, bu eğer şartı karşılanıyor. Onun için bunu yapıyoruz.

Yani bu durumda gevşetiyoruz. Şimdi de ben gevşeyim. Sonra, d[v]'yi, d[u] artı w(u, v)'ye ayarlıyoruz, ki istediğimiz bu. Yani, d[v]'yi, d[u] artı w(u,v) 'ye ayarlıyoruz. Ve burada söylediğimiz gibi bu, Deltası (s, v)'ye eşittir. Bu da ispat etmek istediğimiz şeydir. Bitti. Bu ispatın can alıcı noktasına yaklaştıkça, gittikçe daha fazla heyecanlanıyorum. Buraya kadar herhangi bir soru var mı? Güzel. Şimdi güç kısmı başlıyor. Bunlar çok kolay önkuramlardı, öyle değil mi?

Evet.. Bu iki tahtayı kullanacağım. Bu ispatlara artık ihtiyacımız yok. Sadece bu ifadelere ihtiyacımız olacak; doğruluk bir, doğruluk önkuramlarına; büyük isimler. Şimdi, sonunda "**Doğruluk 2**'ye geliyoruz. Demek ki, elimizde, bir tam ve bir de yarım vardı. Yani sanırım, doğruluğun kendisi, bir mini-üçleme, bir mini-seri.. Pekala, doğruluk iki, algoritma bittiği zaman, doğru cevabı elde edeceğimizi söylüyor. Bu gerçek doğruluk. Ama "**Doğruluk 1**" ile "**Doğruluk önkuramının**" üzerine kurulacak.

Öyleyse tüm köşeler için, d[v]'nin algoritmanın sonunda Delta(s, v) eşit olmasını istiyoruz, tüm v köşeleri için.. Hedefimiz açık olarak bu. Şimdi bu teorem, bütün ağırlıkların negatif olmadığını varsayıyor; bunu hatırlatırım. Başka herhangi bir varsayımda bulunmuyor. Yani sonsuzları bir düzene koyacak. Ama eğer eksi sonsuzlar varsa, tüm bahisler geçersiz olacak. Öyleyse herhangi bir yerde negatif ağırlık kenarı varsa doğru şeyi yapmayabilecektir.

Ama, tüm ağırlıkların negatif olmadıklarını farzetmek, eğer zaman ölçüyorsa makuldur. Genelde, kenarlar boyunca yol almanın bir maliyeti vardır. Bunu yapmanız için kimse size para ödemez. Ama kim bilir? Bu konuda bir iki şey söyleyeceğim. Bir tanesini bir yerlerde söylemiştik: büyük S'ye bir köşe eklediğiniz zaman, ağırlığını hiç bir zaman değişmezsiniz. Evet, bu aslında ispat gerektirir. Bunu burada sadece ifade edeceğim. Görülmesi çok zor birşey değil. d[v] değişmiyor. Bu aslında v'nin, S'ye katıldığı andaki bir tümevarım. Ve bunu birazdan söyleyeceğimiz bir olay takip edecek. Bu nedenle, burada ilgilendiğim yegane şey, bir köşe S'ye eklendiğinde, elimizde doğru tahmin olsa iyi olur, çünkü sonradan değiştirmeyeceğiz.

Tamam, algoritmayı o şekilde tanımlayabilirdik. Tanımlamıyoruz ama, tanımlayabilirdik. Birazdan buna döneceğiz.Demek ki, ilgilendiğimiz tek şey, d[v]nin, Delta (s, v)'ye eşit olup olmadığı. İspat etmek istediğimiz şey bu. Sonunda doğru olması lazım. Ancak v, büyük S'ye eklendiği zaman mevcut olduğunu kanıtlamak yeterli. Bu aslında pratikte ilk söylemi ima etmek oluyor. Eğlenceli bir ima. Ama bunu, yani S' ye bir ekleme yaptığımızda d[v]' nin, Delta(s, v)'ye eşit olduğunu ispatlarsak.. ve gevşetmenin değerleri sadece küçülttüğünü biliyoruz, bu daha da küçülemez.Bu, Doğruluk 1'den..Doğruluk 1, Delta'dan daha küçük bir değer elde edemeyeceğimizi söylüyor. Öyle ise, o noktada bir nitelik elde edersek, bu kalıcı bir nitelik olacaktır. Bunun ima ettiği de, d[v]'nin o noktadan sonra hiçbir zaman değişmediğidir.

Pekala, öyleyse bunu ispatlayacağız. İyi. Doğru olmadığını düşünelim. Yani bu çelişki yöntemiyle ispatlama olacak. Çelişki olması için, bunun bir şekilde başarısız olduğunu düşünelim. Ve ilk başarısızlığa bakalım. u'nun ilk köşe olduğunu farzedelim – S'ye eklenmek üzere olsun. S'ye eklenmeden tam önceki anı düşünmek istiyorum Böyle bir

anda eklenecek bir şeyimiz yok ve bunlar eşit değiller. Yani, d[u], Delta (s, u)'ya eşit değil. Eğer eşit değiller ise, 'Doğruluk 1' den, d[v]'nin, Delta (s, u)'dan, yani u' nun d' sinden, kesinlikle daha büyük olduğunu biliriz. Böylece d[u], Delta (s, u)'dan, kesinlikle daha büyük olur.

Bu ispatın başlangıcı, henüz hiçbir şey fazla heyecan verici değil, sadece biraz ısınma. Fakat, bu Doğruluk 1'i zaten kullanmış oldu.Yanılmıyorsam bu ispat işleminde bunu ilk ve son kez kullanmış oluyoruz. Ne olup bittiğini bir çeşit resimle göstermek istiyorum. Ama bunu biraz tanımlamaya ihtiyacım var. Öyle ise enkısa yola bakalım. Her nasılsa, d[u], enkısa yoldan büyük. Öyleyse, ya tek enkısa yola, veya bir enkısa yola bakalım. Küçük p, bir enkısa yol olsun. Rastgele bir en kısa yol değil, s'den u'ya enkısa yol. Bunun anlamı, bu yolun ağırlığının enkısa yol ağırlığı olduğudur. Böylece, burada olup bitenle ilgili bazı eşitliklerimiz var. Ve biz Delta (s, u) ile ilgileniyoruz.

Burada, o ağırlıkta bir yol var. Bir tane olması lazım, çünkü burada enkısa yollar var; eğer artı sonsuz ise, bu az görünür bir durum ve bu hususta endişelenecek değilim. Öyleyse, bir yere bir resim çizeyim. Demek ki elimizde s var -- ve u var. İşte, s den u ya enkısa yol. O da, p. Neye benzediği hakkında şu ana kadar bir fikir yok. Şimdi, sahip olduğumuz bir diğer şey de büyük harf S nosyonu. Bu nedenle büyük S'yi çiziyorum. İşte, bu büyük S.

Küçük s nin büyük S içinde olduğunu biliyoruz. u'nun henüz büyük S içinde olmadığını biliyoruz. Henüz hiçbir hata yapmadım, değil mi? Bu yol s'de başlıyor ve onu bir noktada terk ediyor. Çünkü, S ye u'yu ekleme anına kadar, ki bu henüz olmadı, u, S'nin içinde değil. Güzel. Yapmak istediğim, p yolunun, burada S'yi terk ettiği ilk yere bakmak.. Öyleyse burada bir köşe olur; ona x diyelim. Burada da bir köşe var. Ona da y diyelim. Muhtemelen, x eşit s'dir. Muhtemelen, y eşit u'dur. Ama bir yerde dışarı çıkması lazım, çünkü içerde başlayıp dışarda bitiyor ve tanımlanabilir limitleri olan sonlu bir yol. Pekala, bunun ilk kez gerçekleştiği zamanı düşünün; ikinci defasını değil, ilk defasını. Yani ilk kenar olan (x,y)'nin, p' nin büyük S'yi terk ettiği yerde olduğunu düşünün.

s'den u ya enkısa yol, büyük S'den dışarı çıkıyor. Bunun bir yerlerde olması lazım. Güzel. Şimdi ne biliyoruz? Küçük x, S'nin içindedir. Öyleyse, doğru cevap onda, çünkü, u'yu S'ye eklemek üzereydik, ve bu S'nin içinde olan bir şeyin ilk yanlış d[x] tahmini idi; ilk ihlal idi. Öyle ise d[x], Delta (s, x)'e eşittir. Çünkü ilk ihlale bakıyoruz ve x de, önceden eklenmiş bir şey. Böylece, zaman bağlamında tümevarımla, ya da ilk ihlal açısından bakıldığında d[x], S'den x'e enkısa yol ağırlığına eşittir. Bu iyi haber. Şimdi, bu önkuramı uygulamaya çalışıyoruz. Geri kalan yapılacak tek şey. Bu önkuramı hiçbir şey için kullanmamıştık.

Şimdi bu kuruluma sahibiz. Eğer, d değerlerinden birinin doğru cevap olduğunu biliyorsak, ve ondan çıkan kenarı gevşettiysek, o takdirde bir diğer doğru yanıt elde ederiz.Tartışmak istediğim şey o. d[x]'in bu ağırlığa eşit olduğunu biliyoruz, çünkü en kısa yolların altyolları yine enkısa yollardır. Çünkü en iyi altyapıya sahibiz, öyleyse, s'den x'e bu, enkısa bir yoldur.Yegane değil, ama bir enkısa yoldur. Yani bunun uyduğunu biliyoruz. Şimdi, bu (x,y) kenarını gevşetme üzerinde düşünmek istiyorum.

x'in büyük S içinde olduğunu biliyoruz. Ve algoritma, "büyük grup S'ye, ne zaman bir u köşesi eklerseniz, oradan dışarı çıkan tüm kenarları gevşetmiş olursunuz." diyor. Öyle ise,

S’ ye x eklediğimizde, geleceğe bakarsanız başka köşelerler de ekleyeceğiz. S’ye x’i ekledikten hemen sonra, bu (x,y) kenarını gevşettik, çünkü, x’den çıkan tüm kenarları, ne idiyseler gevşettik. Bazıları S’nin içine gitti. Bazıları dışarı çıktı. İşte, onlardan bir tanesi burada. Böylece, S’ye x’i eklediğimizde, (x,y) kenarını gevşettik. Pekala, şimdi önkuramı kullanacağız.

Öyle ise, “Doğruluk önkuramı” ile --- ne elde ediyoruz? Bu doğru enkısa yol ağırlığını, şimdi x’e ekliyoruz. (x,y) kenarını gevşetiyoruz. Böylece şimdi, y için enkısa yol ağırlığına sahip olmamız lazım. D[y], eşittir Delta (s, y). Pekala, bu geçmişte bir zamandaydı. Özellikle şimdi, hala doğru olması lazım, çünkü bir kere doğru yanıtı elde edince onu hiç değiştirmezsiniz. Bitirmiş olmamız gerek. Niçin bitti? Pekala, burada başka ne biliyoruz? Çelişki olması için bir şeyler varsaydık, öyle ise ona karşı çıkmamız iyi olacak. d[u]’nun Delta (s,u) dan kesinlikle daha büyük olduğunu varsayıyoruz.

Demek ki, buradaki d[u], tüm bu yolun uzunluğundan kesinlikle daha büyük oluyor. Aslında, u’nun y’ye eşit olup olmadığını gerçekten bilmiyoruz. Olabilir de, olmayabilir de. Ya bu, s’den y’ye bu enkısa yol hakkında ne biliyoruz? Alt-yol olduğu için, s’den u’ ya olan yoldan sadece daha kısa olabilir. Ve enkısa yol’dur, çünkü, enkısa yolun altyoludur. s’den y’ye enkısa yolun, s’den u’ya enkısa yoldan daha küçük veya ona eşit olması lazım.

Yani, böylece s’den y’ye küçük - eşit -- Delta(s, u); sadece altyol olduğu için. Sonuca yaklaşıyorum. Şimdi, Delta (s, u)’yu elde ettim. Bir şekilde d[u]’yu da bu işe katmak istiyorum. Bu sebepten, d[y] ile d[u] arasında bir ilişki kurmak istiyorum. d[u] hakkında ne biliyorum? Evet? D[u] daha küçük, çünkü bir **Min**imum yığınımız var. İşte burada aşağıda. Daima u’yu seçeriz; bu algoritmanın ortasındadır. Bunu en küçük anahtar olarak saklamamın sebebi bu. Bu d üzerine anahtarlanmıştır. Böylece, u’yu S’ye eklemeye çalıştığımız tam bu anda bildiğimiz, y’nin S’ nin içinde olmadığı ve u’nun da S’nin içinde olmadığıdır..

Hatta aynı köşe bile olabilirler. Aynı veya farklı, bu iki köşe de S’nin dışındalar. u’ yu seçtik, çünkü d[u]’nun d tahmini içlerinde en küçüğüydü. Böylece, d[y], d[u]’dan daha büyük, veya ona eşit olmalıdır. Eğer aynı köşe iseler, eşittirler demektir, fakat daima büyük-eşit olarak. Böylece, d[y], burada d[u]’ya büyük - eşittir. Bu kullandığımız hırslı algoritma seçiminin sonucudur. Burası hırslı tercihi uyguladığımız tek yer. Bir yerlerde kullanmamız iyi olur, çünkü rastgele bir köşe seçip işin yapıldığını ilan edemezsiniz. Hırslı olanı almanız lazım. Pekala, şimdi elimizde d[u] var; Delta (s,u)‘ya küçük-eşit, ki bu da bununla çelişki oluşturuyor. Sanki herşey sihirli şekilde yerli yerine oturmuş gibi. Yani daha önceki kanıtlamalarda olduğu gibi, sadece ne olduğunu gözlemliyorsunuz ve gerisi çalışıyor. Tamam, buradaki genelleştirme yani yaklaşıklama bu.

Bu işlemdeki tek gerçek fikir, bu kenara bakmak. Aslında bu kenara da bakabilirsiniz. Ancak S’de olan ve S’den dışarı çıkan bir kenara bakalım ve x’in doğru olduğunu, x doğruysa, y’nin de doğru olması gerektiğini tartışalım ve sonra neden u’ ya baktığımızı soralım. Bakmanız gereken, y olmalıydı. Ve orada bir çelişkiyi elde ediyorsunuz; çünkü y doğru cevaba sahipti. Eğer u eşit y ise bu iyidir; ya da u ile y aynı derecede iyiyse ve bu ağırlıkların hepsi sıfır ise bu da iyidir.

Resme dökersek böyle görünecek. Ama o durumda d[u] doğru yanıttır. d[u] Delta(s,u) idi. Biz olmadığını varsaydık. Çelişkiyi böylece elde ettik. Oldukça açık mı? Bu kanıtlamanın üzerinden tekrar geçin, biraz karmaşıktır; doğal olarak.

İşlemek istediğim bir şeyler daha var, daha kolay şeyler. İlk şey, bu algoritmanın koşma süresi ne? Bunu çok çabuk geçeceğim, çünkü son dersten önce bunu bir çok kere gördük. Bazı ilklendirmeler vardı. Burada yok, çünkü bu doğrusal zamanlı. Önemli birşey değil. Min'i yani en küçüğü çekip çıkarıyorsunuz. Aslında bu bir veri yapısı. Yani elimizde V boyutlu bir şey var. Her köşede en küçüğü yani Min'i bir kez çıkarıyorsunuz, hepsi bu. Yani V'nin boyutu -- Min'leri çıkar.

Pekala bu oldukça basit. Sonra bu ana döngü vardı.Bu tamamen kavramsal bir işlem. Aslında algoritmalarda S kullanılmaz. Sadece düşünme amaçlı. Pekala bu sıfır zaman alıyor. Bunu sevmeniz lazım. Şimdi işin özü burada. Bu döngü kaç kez tekrarlanıyor? Bu u'nun derecesi. Öyleyse, gevşetme adımını toplam olarak kaç defa uyguladık? Sadece bunu yapıyoruz demek değil, fakat bu grubu işleme alıyoruz. Tüm algoritma üzerinde bunu kaç defa yapıyoruz? Her köşede çıkan tüm kenarlara bakıyoruz.

Böylece toplam ne olur? Kenarların sayısı, evet? Böylece bu kenar sayısı kadar tekrar. Aslında bu, geçen sefer gördüğümüz tokalaşma önkuramı ama yönlendirilmiş grafikler için. Ve sadece dışarı giden kenarlara bakıyoruz. Ama burada ikinin faktörü değil, çünkü sadece bir taraftan dışarı çıkıyorsunuz. Böylece, bazı tekrar döngüleri var. En kötü halde, hepsi için anahtar küçültme yani "decrease key" yapıyoruz. Böylece, en çok: E sayıda azaltılmış anahtar. Pekala, süre açısından --, elimizde Vsayıda "extract Min" yani en küçüğü çıkart var ki, bir extract Min yapacak zaman, her ne kadar ise.

Ve, E sayıda azaltılmış anahtar var, her ne kadarsa – ve bu, geçen sefer en az kapsayan ağaç yani **minimum spanning tree** bağlamında Prim'in Algoritmasında elde ettiğimiz koşma süresiyle bire bir aynı. Hangi koşma süresini elde edeceğiniz, hangi veri yapısını kullanacağınıza bağlıdır. Buradaki tüm tabloyu atlıyorum. Ama bir dizilim kullanırsanız, sonuçta koşma süresi $V^2$ düzeyinde olacaktır, çünkü V boyutlu "extract Min" düzeniniz var ve sabit zamanlı anahtar azaltma işleminiz var. Eğer bildiğimiz ve sevdiğimiz ikili yığını kullanırsanız, o zaman her işlem için log V düzeyini elde ederiz. Ve böylece bu, V artı E log V olur. Ve bu nasıl yapılacağını bildiğimiz şey..

Ve eğer, Fibonacci yığını denen bu fantazi veri yapısını kullanırsanız, sabit zamanlı amortize edilmiş azaltılmış anahtar elde edersiniz. Ve koşma süresinde en kötü durum sınırı olarak, E artı V log V elde edersiniz. Bu genelde, negatif kenar ağırlıklı olmayan, tek kaynaklı enkısa yolların, ekstra varsayımlar kullanmaksızın, bildiğimiz en iyi enkısa yollar çözümüdür. Bu, bazen bir onun kadar iyidir ve bu, bazen şundan daha iyidir. Bunlar ise, nasıl yapılacağını bilmeniz dışında bu konu ile ilgisiz çözümlerdir. Kitaptaki ilgili bölümü okumadıkça, Fibonacci yığınını çözmesini bilemezsiniz. Bu nedenle en iyi iki koşma süresini belirttim. Pekala şimdi, önceden görmüş olabileceğiniz daha kolay bir konu hakkında kısaca konuşmak istiyorum. Ve bunu bir grafik yardımıyla "breadth-first search" yani **enine arama** konusuyla ilişkilendirmek eğlenceli olacak...

Aslında bununla Dijkstra'nın sonuna geliyoruz. Ama şimdi ben özel bir durum, grafiğin ağırlıksız olduğu, yani tüm u ile v köşeleri için w (u,v) eşittir bir olduğu bir durum üzerinde düşünmek istiyorum. Böyle bir niteliğin olduğunu düşünün. Dijkstra'dan daha iyisini yapabilir miyiz? Bu koşma süresinden daha iyisini yapabilir miyiz? Muhtemelen, tüm kenarlara ve köşelere bakmamız lazım. Bu sebepten, burada sorguladığım tek şey, bu logV.

Onu ihmal edebilir miyim? Bunun yanıtını birazcık verdim. Yanıt, Breadth First Search yani enine arama veya kısaca **BFS**; belki de önceden görmüşsünüzdür. Depth First Search yani **derinliğine arama**nın yanında, bir grafiğe bakmanın standard yollarından biridir. Ama burada daha önce gördüklerinizden biraz daha fazla şeyler söyleyebiliriz. Enine Arama, aslında Dijkstra Algoritmasıdır; iyi tasarlanmış bir algoritmadır . İki değişiklik var. Birincisi, BFS, öncelik kuyruğunu kullanmaz. Onun yerine ne kullandığını size söyleyeceğim. Öncelik kuyruğu yerine, "**ilk giren ilk çıkar**" (FIFO) kuyruğunu kullanabilirsiniz.

Sonunda çalıştığı ortaya çıkar. Extract Min yapacağınız yerde, sadece kuyruktaki ilk şeyi kuyruktan alırsınız. Anahtar küçültme yapacağınız yerde, ama burada bir incelik vardır. Bu "eğer komutu" biraz değişir. İşte,gevşetme adımı burada. Böylece, gevşetme için çok daha basit birşey söylersiniz. Daha once v'yi ziyaret etmemişsek, v'nin enkısa yol ağırlığına sahip olduğunu belirtiriz, örneğin d[v] eşit d[u] artı bir deriz ve o da kenar (u,v) nin ağırlığıdır.

Ve kuyruğun sonuna v ilave ederiz. Böylece kuyruk boşken işe başlarız. Gerçekte sadece s köşesine sahip olacak, çünkü enkısa yolunun ne olduğunu bildiğimiz tek şey o. Böylece, kuyruk: "Sadece bu şey bu şeyin enkısa yolunu biliyorum. Bununla, sadece dışarı giden kenarlar ile uğraşabileceğim zaman ilgilenirim." der. Böylece başlangıçta, o sadece s. Dışarı giden kenarlar için s sıfır dersiniz. Oradan dışarı giden tüm kenarların ağırlığı bir olur. Kaynaktan enkısa yolun ağırlığı bir'dir. Eğer tüm ağırlıklar bire eşitse bundan daha iyisini yapamazsınız. Böylece tüm o köşeleri kuyruğun sonuna ekleriz. Sonra, bir düzen içinde tüm şeyleri işleme tabi tutarız ve arttırmaya devam eder, eğer değerleri, d[u] ise, ona bir ilave ederiz. Sonra kuyrukta oraya geldiğimizde v'yi S'ye ekleriz. Bu, enine aramadır. Çok kolay.

Bu algoritma ve bir örnek için kitaptaki metne bakabilirsiniz, çünkü onları da anlatacak zamanım yok. Ama önemli olan zamanın daha hızlı olduğudur. Zaman, V artı E düzeyindedir, çünkü önceki gibi tüm köşelerden dışarı giden her kenarlara sadece bir kez bakarız. d[v]'yi bir kez belirledikten sonra, artık o şekilde kalır.

Ona hiç dokunmayız; onu S'ye ekleyeceğiz. Bu sadece bir kere olur. Bu "eğer" komutu ve benzerleri ilk girişte E düzeyinde, daha doğrusu tam olarak E kere yapılır. Kuyruğa-giriş ve kuyruktan-çıkış, ki biz bunu burada Extract Min yerine kullanıyoruz, sabit zaman alır ve böylece toplam koşma süresi, köşelerin sayısı artı kenarların sayısıdır. Tamam, bunun çalışması o kadar belirgin olmamakla beraber, çalıştıklarını Dijkstra çözümlemesini kullanarak kanıtlayabilirsiniz. İspatlamanız gereken tek şey, FIFO kuyruğunun, öncelik kuyruğuyla aynı şeyi yaptığıdır. Bunu bildikten sonra, Dijkstra'nın doğruluğu ile enine

aramanın doğruluğuna ulaşırsınız. Yani enine arama yalnızca tüm köşeleri bulmakla kalmaz, ki belki de bunu sadece köşeleri bulmak için kullanıyorsunuzdur, ağırlıkların hepsi bire eşit olduğunda, s'den her köşeye giden enkısa yol ağırlıklarını da bulur.

İşte Enkısa Yollara Giriş..

Gelecek  sefere negatif ağırlıklarla uğraşacağız..