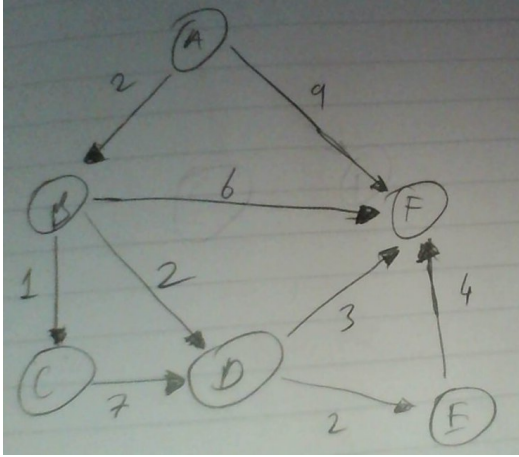


Dinamik Programlama

Dinamik programlamanın (DP) temelinde ardi ardina verilen kararlarin hesaplanmasi fikri yatar; her anda, her verilen karar baska karar seceneklerini ortaya cikarabilir, ve seceneklerin arasindan bir tanesi secilmelidir. Amac en iyi karar zincirini bulmaktır. Bu amac ve metot kismen “acgozlu algoritmalar (greedy algorithms)” olarak bilinen algoritmalarin yaptigina benzer fakat acgozlu algoritmalar, mesela en kısa yolu bulmaya ugrasirken, gezilen dugumlerde sadece “o an icin” en iyi secimi yapar, fakat nihai sonuc goze alindigi zaman bu anlik secimler en iyi sonucu vermeyebilir; Mesela alttaki grafice bakarsak,



diyelim ki a noktasından f noktasına en kısa yoldan ulasmaya calisiyoruz - acgozlu algoritma a,b,c,d uzerinden gidis yapardi cunku her an, sadece o an icin en iyi olani secerdi. Fakat nihai sonuca bakarsak secilen yolun en kısa yol olmadigini goruruz. En iyi yol a,b,d uzerinden giden yoldur.

Not: Ustteki cizit / ag yapisi (graph), yonlu ve çevrimsiz (directed, acyclic graph -DAG-) diye bilinen bir yapidir. Tipik kısa yol problemleri bu yapılar uzerinde calisirlar.

DP problemleri ozellikle bir problemi alt problemlere bolebildigimiz zaman faydalidirlar, ve bu alt problemler tekrar tekrar hesaplaniyorlarsa da bu daha iyidir, cunku DP o alt problemleri onbellekleyerek tekrar hesaplanmadan geri getirilmelerini saglayabilir.

Mesela ustteki en kısa yol problemini DP ile cozelim.

Once bazi teorik, mantiksal konular: tumevarimsal olarak dusunelim. Diyelim ki ustteki DAG’de a, f arasindaki en kısa yolu kesinlikle “biliyoruz”. Ve yine diyelim ki bu yol uzerinde / bir ara nokta x noktasi var. O zaman, a, x, ve x, f arasindaki yollar da tanim itibariyle en kisadir. Ispatlayalim: eger mesela x, f arasindaki en kısa yol bildigimizden baska olsaydi, o zaman eldekini atip o yolu kullaniyor olurduk (en kısa oldugunu kesin biliyoruz ya), ve bu sefer o alternatif en kısa olurdu. Fakat ilk basta en kısa yolu bildigimiz faraziyesi ile basladik. Bir celiski elde ettik, demek ki ara noktanin kisaligi dogrudur \square

Bu ispattan hareketle kısa yolu tek numerik bir deger olarak hesaplamaya ugrasabiliriz.

Oyle bir fonksiyon $d(v)$ olsun ki herhangi bir v nodu icin o nod'dan bitis noktasina olan en kısa uzakligi kesin biliyor olsun (dikkat, bu hesabin nasil olacagini dusunmuyoruz simdilik, sadece olabilecegini, olmus oldugunu farz ediyoruz). Cogtu tumevarimsal tasarimda oldugu gibi hesabin kendisinin ozyinelilik (recursive) cagri zincirinin mekanigi icinde halolmasini amacliyoruz. Dogru olan bir ifadeyi dusunuyoruz oncelikle, ve hesabin kendisini surekli bir sonraki noktaya erteliyoruz.

Devam edelim: u, v arasindaki parca mesafeler $w(u, v)$ 'dir. Simdi, eger bir ara nokta u 'ya gelmissek -yine tumevarimsal olarak dusunmeye devam ediyoruz- bu noktanin her komsusu w icin $d(w)$ 'yi "bildigimize" gore, en kısa yol icin tek yapmamiz gereken her secim aninda en minimal $w(u, v) + d(v)$ 'yi u 'nun uzakligi olarak almaktir.

Veri yapisi olarak DAG'i alttaki gibi gosterelim,

```
DAG = {
    'a': {'b':2, 'f': 9},
    'b': {'d':2, 'c':1, 'f': 6},
    'c': {'d':7},
    'd': {'e':2, 'f': 3},
    'e': {'f':4},
    'f': {}
}
```

Boylece $w(u, v)$ basit bir Python sozluk (dictionary) erisimi haline geliyor, mesela a, b arasi parca mesafe icin

```
print DAG['a']['b']
```

2

En kısa yolu bulacak program

```
from functools import wraps

def memo(func):
    cache = {}
    @wraps(func)
    def wrap(*args):
        if args not in cache:
            print 'onbellekte yok -', args[0]
            cache[args] = func(*args)
        else: print 'onbellekte var -', args[0]
        return cache[args]
    return wrap
```

```
def rec_dag_sp(W, s, t):
    @memo
    def d(u):
        print 'Dugum:' + u[0]
        if u == t: print 'Son nokta t, geri donus'; return 0
        min_dist = min(W[u][v]+d(v) for v in W[u])
        print 'Geri donus,',u,'uzerindeyiz, mesafe=',min_dist
        return min_dist
    return d(s)

dist = rec_dag_sp(DAG, 'a', 'f')
print 'toplaml mesafe=', dist
```

```
onbellekte yok - a
Dugum:a
onbellekte yok - b
Dugum:b
onbellekte yok - c
Dugum:c
onbellekte yok - d
Dugum:d
onbellekte yok - e
Dugum:e
onbellekte yok - f
Dugum:f
Son nokta t, geri donus
Geri donus, e uzerindeyiz, mesafe= 4
onbellekte var - f
Geri donus, d uzerindeyiz, mesafe= 3
Geri donus, c uzerindeyiz, mesafe= 10
onbellekte var - d
onbellekte var - f
Geri donus, b uzerindeyiz, mesafe= 5
onbellekte var - f
Geri donus, a uzerindeyiz, mesafe= 7
toplaml mesafe= 7
```

Simdi cagri mekaniginin hakikaten nasil isledigini gorelim. Not: Onbellek kodlamasi dekorator kullaniyor, bu kullanim icin [2]'ye bakabilirsiniz.

Baslangic u , oradan, minimum secerken, surekli $d()$ cagrisi yapiyoruz, yani $d()$ kendini cagiriyor. Cagrinin geri donmesinin tek yolu son noktaya erismek. Bu ne demektir? Programimiz daha hesap yapmadan “derinligine bir dalis” yapiyor. Son noktalara gelene kadar ozyneli cagrilari ardi ardina uyguluyor, esas hesaplar geri donus sirasinda yapiyor. Bu nasil ise yariyor? Ayrica onbelleklemenin hakikaten isleyip islemedigini nasil bilecegiz? Ya da onbellekteki bir degerin hep en iyisi oldugunu nereden bilecegim?

Not: Bu yaklasimda, hemen belirtelim, onbellek degerini, bir kez set edildi mi, hic degistirmeye gerek yok.

Nokta d 'ye bakalim. Bu noktanin mesafesi (yani son nokta f 'ye uzakligi) kararlaştirilirken algoritma d 'nin her komsusuna bakacaktır, bunu `for v in W[u]` ile yapacaktır. Her komşu için f 'ye gelene kadar o yol derinligine takip edilecektir. Mesela üstteki ciktida goruyoruz ki d sonrasi iki komşu e, f için önce $d-f$ ve $d-e-f$ gidisi yapılmıştır (amac hep o son noktaya ulasmak, unutmayalım). 'Komsulara bakma ve aralarından en azi secme" mantigi tum bu yollar denenene kadar bekleyecektir, ancak hepsi bittikten sonra iclerinden bir minimum sececektir.

Iste simdi niye her dugumdeki minimum hesabinin en iyisi oldugunu anliyoruz, cunku o noktadan nihai noktaya varis için tum alternatifler deneniyor. O derine dalisin sonuclari arasindan bir tanesi seciliyor. Onbellekteki deger bu sebeple bir kez set ediliyor, ve hic degismiyor. Tabii ki onbellekteki deger tekrar tekrar kullanilabiliyor, mesela c için bir d uzakligi gerektiginde bu onbellekten servis edilecektir.

Ve her dugumdaki minimum hesabi en iyiye, bu hesapları kullanan baslangica yakin noktaların hesabi da dogal olarak en iyisi (kisasi) olacaktır. Basta tumevarimsal olarak belirttigimizin tekrar ifade edilmesidir bu.

Kisa Yol Tarifini Bulmak

Mesafe hesabi iste boyle yapiliyor... Peki en kısa yolun kendisini nasıl biliriz? Yani önce suraya, sonra suraya git turunden yol tarifi bilgisi nasıl hesaplanır? Aslında komsular arasındaki en kısa mesafeyi secme problemi, o komsular icinden hangisinin o en mesafeyi sagladigini hatirlama problemine oldukca benziyor. Yani, her dugum uzerindeyken ve komsular arasindan en kısa mesafeyi secerken, o mesafenin "hangi komsudan" geldigini hatirlamak ve bunu bir yerlere kaydetmek yeterli. Her dugum için, son noktaya olan en kısa mesafe degismedigine gore, "o mesafe bilgisinin geldigi komsunun hangisi oldugu" bilgisi de degismeyecektir. Ve her nokta için o "ebeveyn komşu" bilindigi zaman hersey isleyip bittikten sonra en kısa yol tarifi için eldeki kayda bakariz, ve baslangic noktası a 'dan baslayarak ziplaya ziplaya o ebeveyn zinciri ile sona kadar geliriz. Bu degisiklikleri ekleyince kod su hale gelir,

```
parent = {}
```

```
def rec_dag_sp2(W, s, t):
    @memo
    def d(u):
        if u == t: return 0
        distances = [W[u][v]+d(v) for v in W[u]]
        min_dist = min(distances)
        parent[u] = list(W[u])[np.argmin(distances)]
        print 'Geri donus,',u,'uzerindeyiz, mesafe=',min_dist
        return min_dist
    return d(s)
```

```

print rec_dag_sp2(DAG, 'a', 'f')

print 'ebeveynler', parent

onbellekte yok - a
onbellekte yok - b
onbellekte yok - c
onbellekte yok - d
onbellekte yok - e
onbellekte yok - f
Geri donus, e uzerindeyiz, mesafe= 4
onbellekte var - f
Geri donus, d uzerindeyiz, mesafe= 3
Geri donus, c uzerindeyiz, mesafe= 10
onbellekte var - d
onbellekte var - f
Geri donus, b uzerindeyiz, mesafe= 5
onbellekte var - f
Geri donus, a uzerindeyiz, mesafe= 7
7
ebeveynler {'a': 'b', 'c': 'd', 'b': 'd', 'e': 'f', 'd': 'f'}

```

Iste sonuc. Baslangic a, onun ebeveyni b. b'ye bakiyoruz, onunki d. Oradan f'ye atliyoruz, ve sonuca erismis oluyoruz, en kısa yol a-b-d-f.

Analiz

Acgozlu yaklasimdan bu yaklasimin farkini simdi daha iyi gorebiliriz, acgozlu teknik her dugumde en azi bizzat takip eder, ve kisayol hesabi, mesafe hesabi hep bu takip eylemi sirasin o anda yapilir, elde bir toplam vardir ve ona eklenir, vs. Bu yaklasim daha hangi yolu sectigi, sonradan, birkac adim sonrasinda hicbir secimle ilgilenmez. Dinamik Programlama ise takip etme eylemi ile hesap eylemini birbirinden ayirir, ve tumevarimsal bir tanimdan yola cikarak, hep en kısa, en optimali bulmayi basarir.

Kaynaklar

[1] Python Algorithms, 8. Bolum

[2] <http://sayilarvekuramlar.blogspot.de/2013/07/onbelleklemeyi-dekorator-ile-yapmak.html>