Dinamik Programlama

Dinamik Programlama, ya da kısaca "DP", karşımıza çıkacak problemlerin çok büyük bir bölümünde işimize yarayacak, soruları parça parça olarak alt problemlere indirgememizi sağlayacak, kodlarımızın çalışmasını büyük ölçüde hızlandıracak bir metottur. Konu anlatır gibi yazılıp okunması bu beceriyi kazanmanızı sağlamaz hiçbir zaman. DP için soru çözmek gerekir, ne kadar çok DP sorusu çözerseniz o kadar gelişirsiniz DP üzerinde. Aşağıda örnekler üzerinden DP'nin bazı temellerine değineceğiz. Sıkıldığınız yerde sonrakine geçin.

Giriş - Başlangıç düzeyi

DP kullanabilmek için öncelikle, o anki durumumuzu tam olarak ifade eden en masrafsız Durum'u bulmalıyız. Durum, az önce dediğim gibi, o anki şartlarımızı belirten ifadedir. Problemi çözmemiz için bize verilen verileri buna dahil edebiliriz, veya çözümü bulmamızı kolaylaştıracak kendi işimize yarayan verilerimizi tutabiliriz. Ayrıca Durum'umuzun parçalanabiliyor olması gerekiyor. Yani bir Durum'un sonucunu hesaplayabilmek için, onun altındakileri kullanıyor olmamız gerekiyor, DP'nin özelliği de budur zaten. Örnek bir problemle açıklayalım;

Her biri farklı değerde olan N çeşit maden paramız var. En az kaç adet maden para kullanarak S miktarını elde edebiliriz? Bir çeşit maden paradan istediğimiz kadar mevcut.

Öncelikle, **Durum ne işe yarar?** Bu örnek için düşünecek olursak, Durum'umuz tek başına i olacak. Ve her bir Durum için i'den küçük olanların sonuçlarından faydalanarak (i) Durum'unun kendi sonucunu hesaplayacağız. (i) Durum'u dediğimiz şey, yukarıdaki örneğin S için değil de herhangi bir i sayısı için olan cevabı oluyor. 1, 2, 3, ... ve i-1 Durumları için bu problemin cevabını biliyorsak, bu sonuçlardan faydalanarak (i) Durum'unun çözümünü de kolayca hesaplayabiliriz. Şöyle ki;

Bütün paralarımıza teker teker bakalım. j numaralı paranın değerine V_j diyelim. i-V_j Durumu için bu problemin cevabına da m diyelim (daha önceden hesaplamıştık m'i). (i) Durum'umuzun cevabı, (bulabildiğimiz m'lerin en küçüğü)+1 dir. Bütün paralarımızı teker teker denesek, bütün j, V_j ve m değerlerini kullanarak, en küçük sonucu (i)'nin çözümü olarak kaydedebiliriz. Daha sonra, bulup kaydettiğimiz bu sonucu (i+1) Durumu'nu hesaplamak için kullanabiliriz. **i+1 cevabını hesaplarken de i'nin cevabını 'm' olarak kullanacağız.** Bunu anladıysanız temel mantığı kaptınız demektir. Aşağıdaki örnek koda bakarak pekiştirebilirsiniz. Anlamadıysanız koda bakıp tekrar anlatımı okuyun, üzerine düşünün.

```cpp
const int MAXN = 10010 ;
const int inf = 0x7fffffff ;

int N;
int ar[MAXN];

int main(){

    cin >> N >> S ;              //>para sayisi ve cevabini aradigimiz S toplami

    for(int i=1;i<=N;i++)
        scanf(" %d", ar+i);      //>paralarin degerlerini okuduk

    for(int i=1;i<=S;i++)        //>henuz ulasamadigimiz toplamlar icin
        DP[i]= inf;              //sonuclarina buyuk bir deger atiyoruz
                                 //sonsuz kabul edebilmek icin

    DP[i] = 0;                   // >gerekli

    for(int i=1;i<=S;i++)        //>her bir toplamin sonucunu tek tek hesapliyoruz
                                 //kucukten buyuge gidiyoruz cunku buyugun sonucunu
                                 //hesaplarken kucuklerin sonucunu biliyo olmamiz gerekiyor.
        for(int j=1;j<=N;j++)    //>butun paralarimizi teker teker deniyoruz
            if(i-ar[j]>=0 && DP[i-ar[j]]!=inf) //>eger orasi sorunsuz hesaplanmissa
                DP[i]=min(DP[i],DP[i-ar[j]]+1) //>yeni degerimizle guncelliyoruz

    if(DP[S]==inf) printf("Bu paralarla bu toplama ulasilmiyor\n");

    cout << DP[S] << endl ;

    return 0;
}
```

En küçük sonuçları hesaplama işini tersi sırayla yapsak olur muydu? Önce madeni paraları alsak elimize, sonra her bir madeni para için o parayı elde ettiğimiz toplamlara ekleyip yeni toplamların sonuçlarını güncellesek? Aynen çalışır mıydı? Çalışmazsa ne gibi problemler çıkmış olabilirdi karşımıza?

Veya f(S) şeklinde çağırılan bu fonksiyon da işimizi görür:

```cpp
int f(int a){

    if(DP[a]!=inf) return a;

    for(int i=1;i<=N;i++)
        if(a-ar[i]>=0)
            DP[a]=min(DP[a],DP[a-ar[i]]+1);

    return DP[a];
}
```

Orta öncesi seviyede DP

N uzunluğunda bir dizimiz var ve bu dizideki en uzun **azalmayan** alt diziyi bulmak istiyoruz. Önce şunu düşünelim, ortadan bir sayı seçsek, buraya kadar olan en uzun alt diziyi nasıl bulabiliriz? Eğer o sayının solundaki bütün sayılar için bu soru cevaplanmışsa, soldaki bizden küçük-eşit olan sayılar arasından çözümü en büyük olanı seçebiliriz. Önce soldakileri cevaplamaktan başlayalım o halde, onları hesaplarken de onların solundakileri biliyor olmamız gerekecek. En baştan gelelim o zaman.

- Birinci eleman için cevap 1'dir. Tek elemanı kendisi olan bir dizi.
- İkinci eleman için cevap; eğer ilk eleman kendisinden küçükse 2, değilse 1'dir.
- Üçüncü eleman için cevap; 1, ilk elemanın çözümü+1(ilk eleman kendisinden küçükse), ikinci elemanın çözümü+1(ikinci eleman kendisinden küçükse). Bunlardan en büyük olanıdır.
- **Her bir eleman için cevap; kendisinden solda ve kendisinden küçük olan sayılardan çözümü en büyük olanın çözümü, +1'dir.**

```
cin >> N ;

for(int i=1;i<=N;i++)
    scanf(" %d",ar+i);

for(int i=1;i<=N;i++) // bütün sayılar için minimum çözüm 1 dir.
    DP[i]=1;          // sayının kendisi, 1 uzunluğunda bir dizi.

for(int i=1;i<=N;i++) // i numaralı elemanımıza kadarki max sonucu hesaplayacağız
    for(int j=1;j<i;j++) // kendisinden soldaki elemanlara bakıyoruz
        if(ar[j]<=ar[i]) // dizinin azalmayan olması için
            DP[i]=max(DP[i],DP[j]+1); // sonucumuzu öncekilere göre güncelliyoruz

int res=0;
for(int i=1;i<=N;i++) // en büyük sonucu arıyoruz.
    res=max(res,DP[i]);

cout << res << endl ;
```

Ödev: N tane node'u olan bir graphta 1 ve N numaralı node'lar arasındaki en kısa yolu bulalım.

Bu sorularda bir cevabı hesaplarken önceki verilerden yararlanmayı öğreniyoruz. Durum'larımız(state) henüz tek boyutlu. Buradan bol bol örnek çözdükten sonra diğer seviyeye geçebilirsiniz.

Örnek sorular:

http://community.topcoder.com/tc?module=ProblemDetail&rd=4493&pm=1259

http://community.topcoder.com/tc?module=ProblemDetail&rd=5009&pm=2402

http://community.topcoder.com/tc?module=ProblemDetail&rd=5006&pm=1918

Orta seviye DP

NxM büyüklüğünde bir matrisimiz var. Her hanesinde belli bir miktar elma var. Sol üst kareden başlıyoruz. Yalnızca bir birim sağa veya bir birim aşağı hareket ederek sol alt köşeye ulaşıyoruz. Öyle bir yol izleyelim ki topladığımız elma sayısı max olsun. En fazla kaç elma toplarız?

Önce Durum'umuzu belirleyelim. Sol üstten başlayıp herhangi bir kareye ulaşırken en fazla kaç toplayabileceğimizi nasıl bulabiliriz? O kareye nereden gelmiş olabiliriz? Ya solundaki kareden, ya da üzerindeki kareden. Çünkü sadece sağa ve aşağıya hareket edebiliyorduk. O halde **soldaki ve üstteki karelerden çözümü büyük olan hangisiyse oradan gelmiş olmak işimize gelir.** Her kare için soldaki ve üsttekinin çözümünden büyük olanı alsak bu şekilde doldurabiliriz çözüm matrisimizi. Sol üst kareden başlayalım o zaman. Burayı anladıysanız artık çift boyutlu Durum'lar yazabilirsiniz demektir.

```
cin >> N >> M ;

for(int i=1;i<=N;i++)
    for(int j=1;j<=M;j++)
        scanf(" %d",&ar[i][j]);

for(int i=1;i<=N;i++)
    for(int j=1;j<=M;j++)
        DP[i][j]=max(DP[i-1][j],DP[i][j-1])+ar[i][j];

cout << DP[N][M] << endl ;
```

```
For i = 0 to N - 1
   For j = 0 to M - 1
   S[i][j] = A[i][j] +
      max(S[i][j-1], if j>0 ; S[i-1][j], if i>0 ; 0)

Output S[n-1][m-1]
```

Örnek sorular:

http://community.topcoder.com/tc?module=ProblemDetail&rd=4709&pm=1889

http://community.topcoder.com/tc?module=ProblemDetail&rd=4482&pm=1592

Orta ilerisi seviye DP

Burada artık bir şeylerin sonucunu hesaplarken bazı kısıtlamalara riayet etmeye başlayacağız. Şöyle bir soru güzel bir örnek olur:

Elimizde M paramız var. N node'u olan bir graphımız var. Bu nodelar arasındaki çift yönlü yolların hem uzaklığı hem de ücreti var. 1 numaralı nodedan N numaralı node'a giden en kısa yolu bulacağız. Ancak ödeyeceğimiz ücret M'i geçmemeli. Eğer birden fazla en kısa yol varsa, en ucuzunu bulmalıyız.

Öncelikle burada Durum'umuzu (State) çift boyutlu yapmak zorundayız. Çünkü hem hangi şehirde olduğumuz önemli, hem de elimizde kaç para kaldığı önemli. Dinamik değerlerimizi tutacağımız dizimiz, DP[N][M] şeklinde bir matris olmalı. Bildiğimiz Djikistra'nın Shortest Path algoritmasından farklı olarak burada elimizde kalan parayı da tutmamız gerekiyor. Yine aynı şekilde her adımda daha önce kullanılmamış bir [i][j] Durumu alacağız elimize. Onu işaretleyeceğiz(used). Sonra komşularına bakıp bulduğumuz minimum değerleri daha da iyileştirmeye çalışacağız. Aşağıda örnek bir Pseudocode görebilirsiniz:

```
Set states(i,j) as unvisited for all (i,j)
Set Min[i][j] to Infinity for all (i,j)

Min[0][M]=0

While(TRUE)

Among all unvisited states(i,j) find the one for which Min[i][j]
is the smallest. Let this state found be (k,l).

If there wasn't found any state (k,l) for which Min[k][l] is
less than Infinity - exit While loop.

Mark state(k,l) as visited

For All Neighbors p of Vertex k.
   If (l-S[p]>=0 AND
    Min[p][l-S[p]]>Min[k][l]+Dist[k][p])
      Then Min[p][l-S[p]]=Min[k][l]+Dist[k][p]
   i.e.
If for state(i,j) there are enough money left for
going to vertex p (l-S[p] represents the money that
will remain after passing to vertex p), and the
shortest path found for state(p,l-S[p]) is bigger
than [the shortest path found for
state(k,l)] + [distance from vertex k to vertex p)],
then set the shortest path for state(i,j) to be equal
to this sum.
End For

End While

Find the smallest number among Min[N-1][j] (for all j, 0<=j<=M);
if there are more than one such states, then take the one with greater
j. If there are no states(N-1,j) with value less than Infinity - then
such a path doesn't exist.
```

Olayın sadece sonuç hesaplamak olmayıp bazı farklı kısıtlamaları da dikkate alarak soruyu çözebilmek olduğunu gördüyseniz ve Durum'larınızı buna göre belirleyip problemi alt parçalara bölebiliyorsanız bu aşama da tamam demektir.

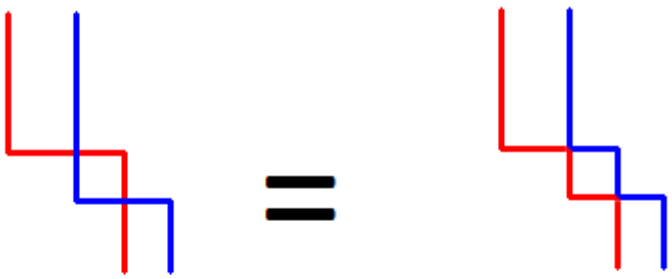
Örnek birkaç soru çözüp sonraki aşamaya geçebilirsiniz.

http://community.topcoder.com/tc?module=ProblemDetail&rd=4705&pm=1166
http://community.topcoder.com/tc?module=ProblemDetail&rd=4555&pm=1215
http://community.topcoder.com/tc?module=ProblemDetail&rd=5072&pm=2829
http://community.topcoder.com/tc?module=ProblemDetail&rd=4630&pm=1861

İleri düzey DP

Verilen NxM lik bir matriste sol üst köşeden başlayıp sağ alt köşeye ulaşıp, sonra tekrar sol-üst, sonra tekrar sağ alt köşeye gideceğiz. Daha önce basit versiyonunu çözmüştük, oradaki gibi hareket edeceğiz. Geçtiğimiz karelerdeki sayıları toplayacağız. En fazla kaç toplayabiliriz? (N,M<=50)

Öncelikle git-gel olayları aynı matris üzerinde gerçekleştiği için sol-üst köşeden sağ-alt köşeye 3 defa gidilecek gibi düşünebiliriz. Yani 3 farklı yol belirleyeceğiz. Önceden çözmüş olduğumuz kolay sorunun üçlü versiyonu gibi. Burada gözlememiz gereken şey, bu yolların kesişmemesi gerektiğidir. Aşağıdaki iki durum arasında hiçbir fark yoktur, sonuç olarak.



Ayrıca yollarımızın kesişmeleri bize hiçbir fayda sağlamaz. Çünkü bir kareden ne kadar geçersek geçelim oradaki sayıyı sadece bir kez alabiliyoruz. Seçeceğimiz yollar birbirinden ayrı olmalı yani.

Soruyu daha rahat çözmek için matrise satır ve sütunlar olarak değil diyagonaller olarak yaklaşmamız daha iyi olur. Çözümü hesaplarken, Durum'umuz 4 boyutlu ve [2*N][N][N][N] şeklinde olur. İlk kısımda kaçıncı diyagonalde olduğumuzu, diğerlerinde ise satır numaralarını tutarız. Diyagonal ve satır numarası bilindiği zaman sütun numarası da bulunabilir zaten.

Her bir durumun sonucunu hesaplarken oradan gidilebilecek durumlardan maksimum olanının cevabını bulup ona göre kendi sonucumuzu güncelleriz. Bulduğumuz bu sonucu bir önceki diyagonaldeki cevapları hesaplarken kullanırız.

Ancak çoğu zaman bu büyüklükte bir dizi almanıza izin verilmez. Bu yüzden [N][N][N] şeklinde iki dizi alıp bunların üzerinden ilerleyebiliriz. Çünkü bir diyagonaldeki cevapları hesaplarken bize sadece 2 dizi lazım olacak, biri o an hesapladığımız, biri de ondan bir sonraki diyagonalin cevapları. Bu ikisini kullanarak o anki diyagonalin cevabını hesaplarız, daha sonra bulduğumuz sonuçları diğer diziye aktarıp, sonuç hesapladığımız diziyi bir üstteki diyagonalin sonuçlarını hesaplamak için kullanırız. Bu şekilde en son bulduğumuz çözümler toplana toplana [1][1][1][1] Durum'umuzun çözümüne ulaşırız, yani sorumuzun cevabına.

Örnek Soru : http://community.topcoder.com/tc?module=ProblemDetail&rd=4710&pm=1996

Not: Bir soruyu okuduktan sonra sınırlarına bakın, eğer polinom zamanlı bir algoritma geliştirilebiliyorsa sorunun dinamik bir çözümü olması muhtemeldir. Eğer öyleyse problemi parçalamanızı sağlayacak Durum'lar bulmaya çalışın ve bu Durum'lar arasındaki geçişi nasıl yapabileceğinizi bulun. Eğer bir soru DP gibi duruyorsa ama Durum'u belirleyemiyorsanız, problemi başka problemlere benzetmeye-indirgemeye çalışın. Az önceki sorudaki yaptığımız gibi.

DP soru çözmekle geliştirilir.