**KNAPSACK PROBLEMİ**

Elimizde ağırlık ve değer bilgileri verilen n adet eşya var. Ayrıca bir sırt çantamız var ve bu çantanın kapasitesi W kadar. Her eşyayı ya çantaya tamamen koyabiliriz ya da hiç koyamayız. Maksimum hangi değerle çantayı doldurabiliriz?

* n = 4 (eşya sayısı)
* Ağırlıklar = [2, 3, 4, 5]
* Değerler = [3, 4, 5, 6]
* Kapasite = 5

Maksimum değer kaç olacaktır?

| **Yöntem** | **Açıklama** |
| --- | --- |
| Recursion | Her eşya için "al" veya "alma" kararı vererek çözülür. |
| Dynamic Programming | Tablo kullanarak çözülür. Zaman ve alan karmaşıklığı O(n\*W). En verimli ve sık kullanılan yöntemdir. |

Memoization recursıve metod kullanılacak fakat her yol 1 kez kaydolacak

**Rec Dez:** Hem alma durumunda hem de almama durumunda aynı parametreler için yeniden çalışıyor, O(2^n). Her bir adımda, her bir yol 2ye dallanıyor. 2,4,8,… Her eşya için 2 seçenek.

**Çözüm:** Bu tür tekrarların önüne geçmek için, Daha önce hesaplanan alt durumları bir yerde saklarız (dizi, tablo vs.). Memoization.

**Memo Avantaj:** Memoization’da memo[indeks, kapasite] ile önceden hesaplananları saklıyoruz. Bu sayede her kombinasyon sadece bir kez hesaplanıyor. Zaman karmaşıklığı: O(n \* kapasite) → çok daha hızlı. N eşya sayisi, w kapasite. İndis, w sadece 1 kez hesaplanır. 2 ye dallansa bıle her hücreye yalnızca 1 kere hesaplama yaptık.

Recursionda 2^n cunku her bırı ıcın al, alma 2 kere dallandı. Bu nedenle memo>rec

**DYNAMIC PROGRAMMING(BOTTOM UP)**

* Tüm alt problemleri küçükten büyüğe çöz.
* Bir tablo (dp[n+1][W+1]) oluştur.
* dp[i][w] → ilk i eşyayla, kapasitesi w olan çantada elde edilebilecek maksimum değer.
* n = 4 (eşya sayısı)
* Ağırlıklar = [2, 3, 4, 5]
* Değerler = [3, 4, 5, 6]
* Kapasite = 5

i/w 0 1 2 3 4 5

0 0 0 0 0 0 0

1 0 0 3 3 3 3

2 0 0 3 4 4 7

3 0 0 3 4 5 7

4 0 0 3 4 5 7

0,0 satırı? Base case durma şartımız. Return 0 oldugundan o satır ve sutun 0.

İ=1 1 tane eşya kullansaydık

W=3 çanta kapasitesi 3 olsaydı en fazla ne kadar değerle doldururdum?

Alt problemlere ayırıp çözüyoruz, tam kapasite dolduğunda ya da tüm nesneler kullanıldığında best case elimde kayıtlı. Gerçek çanta kapasitemiz 5.

Ama dynamic programming yaklaşımıyla, 1’den 5’e kadar olan tüm alt kapasite senaryolarını da çözüyoruz. Çünkü bazı eşyaları alırsak, çantada geriye kalan kapasiteye bakmak zorundayız. Tüm olasılıkları tabloya önceden dolduruyoruz.

* Tabloyu doldurduktan sonra
* dp[n,W] değerine ulaşan yolu bulmak istiyoruz
* Hangi seçimler bizi bu değere getirdi?
* Yani: "Yukarıdan mı geldim, yoksa çaprazdan mı?" gibi düşünüyoruz.

Alınan kararların izini sürüyoruz.

Bu eşyayı alırsam, çantanın içinde yer kalıyor.  
Kalan yerle neler yapabilirim?  
O yerle önceki eşyaları kullanarak ne kadar daha değer kazanabilirim? Bu yüzden geri izleme yapıyoruz. Önceki eşyayı almışsam onu da durumlara eklemek zorundayım. Belki önceki 2 eşyadan biri bu kalan yere sığıyor?

**Recursive** :Ne yaptık? Her eşya için iki seçenek düşündük. Al ya da alma.

Bu iki seçenek üzerinden tüm olasılıkları denedik.Amaç: En iyi sonucu bulmak için tüm yolları denemek.Sorun: Aynı alt problemleri tekrar tekrar çözdü. Karmaşıklık → O(2^n)

**Memoization:** Recursive çözümdeki alt problemleri hafızaya aldık (tablo memo[i,w]). Aynı alt problem tekrar geldiğinde hesaplamadan direkt hafızadan aldık. Zaman O(n \* W) oldu. Ama hala recursive yapıdayız.

**Dynamic**: Alt problemleri küçükten büyüğe doğru bir tabloyla çözdük (dina[i,w]). Tüm tabloyu doldurup, en sağ alt köşede sonucu bulduk. Aynı zaman karmaşıklığı: O(n \* W). Ama daha sistematik, stack taşma riski yok. Geriye doğru izleyip hangi eşyaların seçildiğini çıkarabildik.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Algoritmalar\_2025\_Knapsack

{

class Program

{

public static void HangisiSecildi(int[,] dp, int[] agirliklar, int[] degerler, int n, int kapasite)

{

int i = n;

int w = kapasite;

Console.WriteLine("Seçilen eşyalar (indeksler 0'dan başlar):");

while (i>0 && w>0)

{

//Esya alınmadıysa önceki değerini korur

if (dp[i,w]!=dp[i-1,w])

{

Console.WriteLine("Eşya {0} → Ağırlık: {1}, Değer: {2}", i - 1, agirliklar[i - 1], degerler[i - 1]);

w -= agirliklar[i - 1];

}

i--;

}

}

public static int KnapsackRec(int[] agirliklar, int[] degerler, int kapasite, int indis)

{

if (indis==0 || kapasite==0)

{

return 0;

}

if (agirliklar[indis-1]>kapasite)

{

return KnapsackRec(agirliklar,degerler,kapasite, indis-1);

}

else

{

int alma=KnapsackRec(agirliklar,degerler,kapasite, indis-1);

int al = degerler[indis - 1] + KnapsackRec(agirliklar, degerler, kapasite - agirliklar[indis - 1], indis - 1);

return Math.Max(al, alma);

}

}

public static int[,] memo;

public static int KnapsackMemo(int[] agirliklar, int[] degerler, int kapasite, int indis)

{

if (indis==0 || kapasite==0)

{

return 0;

}

if (memo[indis,kapasite]!=-1) //Problem çözülmüşse tekrar hesaplama

{

return memo[indis,kapasite];

}

if (agirliklar[indis-1]>kapasite)

{

memo[indis, kapasite] = KnapsackMemo(agirliklar, degerler, kapasite, indis - 1);

}

else

{

int alma = KnapsackMemo(agirliklar, degerler, kapasite, indis - 1);

int al = degerler[indis - 1] + KnapsackMemo(agirliklar, degerler, kapasite - agirliklar[indis - 1], indis - 1);

memo[indis, kapasite] = Math.Max(alma, al);

}

return memo[indis, kapasite];

}

public static int KnapsackDina(int[] agirliklar, int[] degerler, int kapasite, int indis)

{

int [,] dina=new int[indis+1,kapasite+1]; //hepsi 0

for (int i = 1; i <= indis; i++) //i=indis

{

for (int w = 1; w <= kapasite; w++) //w=kapasite

{

if (agirliklar[i-1]<=w)

{

int al = degerler[i - 1] + dina[i - 1, w - agirliklar[i - 1]];

int alma = dina[i - 1, kapasite]; //Üst satırdaki aynı kapasiteyle ne almısım

dina[i, w] = Math.Max(al, alma);

}

else

{

// Eşyayı alamıyoruz

dina[i, w] = dina[i - 1, w];

}

}

}

HangisiSecildi(dina, agirliklar, degerler, indis, kapasite);

//Tablonun en sağ alt köşesi: tüm eşyalarla maksimum değer

return dina[indis, kapasite];

}

static void Main(string[] args)

{

int[] degerler = { 3, 4, 5, 6 };

int[] agirliklar = { 2, 3, 4, 5 };

int kapasite = 5;

int n = degerler.Length;

int indis = agirliklar.Length;

memo = new int[n + 1, kapasite + 1];

for (int i = 0; i <= n; i++)

{

for (int w = 0; w <= kapasite; w++)

{

memo[i, w] = -1;

}

}

//int sonuc = KnapsackRec(agirliklar, degerler, kapasite, n);

//int sonuc = KnapsackMemo(agirliklar, degerler, kapasite, n);

int sonuc = KnapsackDina(agirliklar, degerler, kapasite, indis);

Console.WriteLine("Maksimum değer: " + sonuc);

}

}

}

//integer programming