Integer Programming (Tamsayılı Programlama), Linear Programming (Doğrusal Programlama) problemlerinin bir çeşididir. En temel farkı şu:

* Lineer programlamada değişkenler reel sayılar olabilir (örneğin 3.2 sandalye üret diyebilir).
* Tamsayılı programlamada değişkenlerin tamsayı olması zorunludur (örneğin 3 veya 4 sandalye üret diyebilirsin ama 3.2 üretemezsin).

**Nerelerde Kullanılır?**

* Üretim planlaması
* Kaynak dağılımı
* Fabrika optimizasyonları
* Personel vardiya çizelgelemesi
* Taşıma problemleri
* Knapsack (sırt çantası) problemleri
* Seçim problemleri (örneğin: 5 makineden 3'ünü seç gibi)

BRUTE FORCE 4 ÖRNEK:

SORU1:

Diyelim ki 24 ahşap, 16 çivin var.  
Her sandalye: 4 ahşap, 2 çivi  
Her masa: 6 ahşap, 4 çivi  
Sandalye = 20 TL, Masa = 30 TL kar  
Maksimum kar için kaç sandalye ve masa üretmelisin?

**Amaç:**  
20\*S + 30\*M → **maksimize et**

**Kısıtlar:**  
4\*S + 6\*M ≤ 24  
2\*S + 4\*M ≤ 16  
S, M ∈ N (doğal sayılar)

Tüm olasılıkları deneyip en uygun sonucu seçerek yapılır (brute-force)

Ya da dinamik programlama, backtracking, branch & bound gibi tekniklerle optimize edilir.

SORU1: Bir fabrika gözlük ve saat üretiyor.

* Gözlük: 2 cam, 1 metal → 15 TL kazandırıyor
* Saat: 3 cam, 2 metal → 25 TL kazandırıyor
* Toplam: 18 cam ve 12 metal var

**AMAÇ:** 15 \* G + 25 \* S → maksimize et

**DENKLEMLER:**

2G + 3S ≤ 18 (cam)

1G + 2S ≤ 12 (metal)

G, S ≥ 0 ve tamsayı

**Gözlük:**

* cam: 18 / 2 = 9
* metal: 12 / 1 = 12  
  → **Maksimum gözlük = min(9, 12) = 9**

**Saat:**

* cam: 18 / 3 = 6
* metal: 12 / 2 = 6  
  → **Maksimum saat = min(6, 6) = 6**

ÇÖZÜM:

using System;

class Program

{

static void Main()

{

int maxKar = 0;

int enIyiGozluk = 0, enIyiSaat = 0;

// Gözlük için en fazla 18/2 = 9 üretilebilir

for (int g = 0; g <= 9; g++)

{

// Saat için en fazla 18/3 = 6 üretilebilir

for (int s = 0; s <= 6; s++)

{

int cam = g \* 2 + s \* 3;

int metal = g \* 1 + s \* 2;

// Kaynaklar yeterliyse karı hesapla

if (cam <= 18 && metal <= 12)

{

int kar = g \* 15 + s \* 25;

// Daha iyi bir kar varsa onu kaydet

if (kar > maxKar)

{

maxKar = kar;

enIyiGozluk = g;

enIyiSaat = s;

}

}

}

}

// Sonuçları yazdır

Console.WriteLine("En iyi üretim planı:");

Console.WriteLine("Gözlük adedi: " + enIyiGozluk);

Console.WriteLine("Saat adedi: " + enIyiSaat);

Console.WriteLine("Toplam Kar: " + maxKar + " TL");

}

}

Gözlük ve saat üretimi için her ihtimali deniyoruz. Her durumda cam ve metal yeterli mi diye bakıyoruz. Yeterliyse karı hesaplıyoruz. En karlı durumu aklımızda tutuyoruz. Döngü burada tüm kombinasyonları geziyor.

SORU 2: Hediye Kutusu Seçimi : Bir mağazada 3 farklı hediye kutusu bulunmaktadır. Her kutudan en fazla 1 tane alınabilir. Elimizde 3 kutu var: A, B, C. A: 3 kg, 9 TL

B: 4 kg, 13 TL C: 2 kg, 6 TL

Toplam taşıma kapasitesi 9 kg . Değer toplamını maksimize et. (Knapsack tarzı). Toplam ağırlık ≤ 9 kg

Her kutu ya alınır ya alınmaz (0 veya 1)

a, b, c → 0 ya da 1 değer alır (seçildi mi, seçilmedi mi?)

Bu sayede tüm kombinasyonlar denenir:  
000, 001, 010, ..., 111

8 kombinasyon var (2³ = 8) Her kombinasyon için: Ağırlık ve değer hesaplanır Eğer ağırlık ≤ 9 ise o kombinasyon değerlidir. En yüksek değeri tutan kombinasyon akılda tutulur

ÇÖZÜM:

using System;

class Program

{

static void Main()

{

int maxDeger = 0;

int secilenA = 0, secilenB = 0, secilenC = 0;

for (int a = 0; a <=1 ; a++)

{

for (int b = 0; b <=1; b++)

{

for (int c = 0; c <=1 ; c++)

{

int agirlik = a \* 5 + b \* 6 + c \* 3;

int deger = a \* 9 + b \* 13 + c \* 2;

if (agirlik<=9)

{

maxDeger = deger;

secilenA = a;

secilenB = b;

secilenC = c;

}

}

}

}

Console.WriteLine("En iyi kutu kombinasyonu:");

if (secilenA == 1) Console.WriteLine("Kutu A");

if (secilenB == 1) Console.WriteLine("Kutu B");

if (secilenC == 1) Console.WriteLine("Kutu C");

Console.WriteLine("Toplam Değer: " + maxDeger + " TL");

}

}

3 yerine 30 kutu olsaydı? Çalışır mıydı, neden?

**Bitmask çözümü: (MAX 20=1 M )**

using System;

class Program

{

static void Main()

{

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

int[] degerler = {9,13,6 };

string[] kutuIsimleri = { "A","B","C"};

int maxAgirlik = 9;

int maxDeger = 0;

string secilenKutular = "";

//3 kutu var 8 kombınasyon eder

for (int mask = 0; mask < 8; mask++)

{

int toplamAgirlik = 0;

int toplamDeger = 0;

string secili = "";

for (int i = 0; i < 3; i++)

{

//maskenin 1. biti 1 mi?

if ((mask&(1<<i))!=0) //Sadece i. biti 1 olan sayı

{

toplamAgirlik += agirliklar[i];

toplamDeger+=degerler[i];

secili += kutuIsimleri[i] + " ";

}

}

if (toplamAgirlik<=maxAgirlik && toplamDeger>maxDeger)

{

maxDeger = toplamDeger;

secilenKutular = secili;

}

}

Console.WriteLine("En iyi kutu kombinasyonu: ");

for (int i = 0; i < secilenKutular.Length; i++)

{

Console.WriteLine(secilenKutular[i] + " ");

}

Console.WriteLine("Toplam deger: " + maxDeger + "TL");

}

}

for (int i = 0; i < secilenKutular.Length; i++)

{

Console.WriteLine(secilenKutular[i] + " ");

}

Console.WriteLine("Toplam deger: " + maxDeger + "TL");

}

}

mask (int) binary Anlamı

0 000 Hiçbirini alma

1 001 Sadece C'yi al

2 010 Sadece B'yi al

3 011 B ve C'yi al

4 100 Sadece A'yı al

5 101 A ve C'yi al

6 110 A ve B'yi al

7 111 Hepsini al

Mask=5 binary=0101 yani 0. Ve 2. Kutu seçili

SORU3:

Bir serada iki çiçek türü üretilmektedir: lale, gül

Lale🡪 2 su 3 gübre kar 12tl

Gül🡪 3 su 2 gübre kar 16 tl

Toplam kaynak:

* 18 litre su
* 18 birim gübre

Her çiçekten tamsayı kadar üretilebilir, yarım çiçek olmaz.

ÇÖZÜM:

using System;

class Program

{

static void Main()

{

int maxKar = 0, enIyiLale = 0, enIyiGul = 0;

int maxLale = Math.Min(18/2,18/3);

int maxGul = Math.Min(18/3,18/2);

for (int l = 0; l <=maxLale; l++)

{

for (int g = 0; g <=maxGul; g++)

{

int kar = 1 \* 12 + g \* 16;

if (kar>maxKar)

{

maxKar = kar;

enIyiLale = l;

enIyiGul = g;

}

}

}

Console.WriteLine("En karlı üretim: ");

Console.WriteLine("Lale: "+enIyiLale);

Console.WriteLine("Gul: "+enIyiGul);

Console.WriteLine("Toplam kar: "+maxGul+" TL");

}

}

int maxLale = Math.Min(18/2,18/3);

int maxGul = Math.Min(18/3,18/2); neden?

Sınırları gereksiz büyük yazmamak için.

Mesela 18 su / 2 su → max 9 lale olabilir.

Ama gübre daha önce biterse o sınıra göre azaltılır.

Bu hem verimli hem güvenli tarama yapmamızı sağlar.

Tam sayı çıkmasaydı?

Hiç özel bir kontrol yazmasak bile, int değişkenlerle çalıştığımız sürece yarım ürün üretmek mümkün değildir.

SORU4:

**Kargo Alanı Optimizasyonu (Greedy Tabanlı Integer Programming) Problem Tanımı:**

Bir kargo şirketi, taşıma alanını maksimum kâr elde edecek şekilde doldurmak istiyor.  
Kullanılabilecek 3 tür koli vardır:

| **Koli Türü** | **Hacim (Desi)** | **Kâr (TL)** | **Stok Durumu** | **Kar / Desi Oranı** |
| --- | --- | --- | --- | --- |
| Küçük | 3 | 6 | Sınırsız | 2.00 |
| Orta | 5 | 10 | Sınırsız | 2.00 |
| Büyük | 7 | 13 | Sınırsız | 1.857 |

Toplam taşıma kapasitesi: 50 desi

Amaç:

* Bu alanı tamsayı sayıda koliyle doldur
* Toplam kârı maksimize et
* Yarım koli kullanılamaz

Kısıtlar:

* Toplam kullanılan hacim ≤ 50
* Sadece tamsayı adet koli yerleştirilebilir
* Koli adedinde üst sınır yok (stok sınırsız)
* Her koliden istenildiği kadar seçilebilir, ama yer kadar

Bu soruda şunları bul:

1. Hangi kolilerden kaçar adet kullanılmalı?
2. Toplam kaç koli yerleştirilmiştir?
3. Toplam kâr nedir?

using System;

class Program

{

static void Main()

{

//Değişkenleri tanımla

int toplamAlan = 50;

int kalanAlan = toplamAlan;

int kucukHacim = 3, kucukKar = 6;

int ortaHacim = 5, ortaKar = 10;

int buyukHacim = 7, buyukKar = 13;

int adetKucuk = 0, adetOrta = 0, adetBuyuk = 0;

//Greedy Alg: en verimli kolilerden başla, en çok kar getiren kolileri önce seç

while (kalanAlan>=kucukHacim)

{

adetKucuk++;

kalanAlan -= kucukHacim;

}

while (kalanAlan>=ortaHacim)

{

adetOrta++;

kalanAlan -= ortaHacim;

}

while (kalanAlan>=buyukHacim)

{

adetBuyuk++;

kalanAlan -= buyukHacim;

}

//Kar hesabı ve ekrana basma

int toplamKar = adetKucuk \* kucukKar + adetOrta \* ortaKar + adetBuyuk \* buyukKar;

Console.WriteLine("Koli Dağılımı:");

Console.WriteLine("Küçük: " + adetKucuk + " adet → " + (adetKucuk \* kucukKar) + " TL");

Console.WriteLine("Orta : " + adetOrta + " adet → " + (adetOrta \* ortaKar) + " TL");

Console.WriteLine("Büyük: " + adetBuyuk + " adet → " + (adetBuyuk \* buyukKar) + " TL");

Console.WriteLine("Toplam Kar: " + toplamKar + " TL");

Console.WriteLine("Kalan boşluk: " + kalanAlan + " desi");

Greedy, şu anda en iyi görünen seçeneği seçer ama bunun globalde en iyi olduğunu garanti etmez.

Eğer problem yapısı uygunsa (örneğin kar/desi oranı monoton azalıyorsa) greedy çok hızlı ve doğru çözüm verir.

//Kutu sayısına sınır getirirsek?

5 tane küçük 4 tane orta 2 tane büyük=?

Dynamic Programming?

Stok sınırı geldiğinde bazı durumlarda greedy artık optimum sonucu veremeyebilir.

Mesela küçük koliler sınırlıysa ama greedy onları tükettiyse, orta koliye yer kalmayabilir, oysa daha karlı olurdu.

Bu yüzden bu tip sorularda:

Greedy = hızlı ama riskli

DP / brute-force = garantili ama daha yavaş

using System;

class Program

{

static void Main()

{

int kapasite = 50;

int[] hacim = {3,5,7};

int[] kar = {6,10,13};

int[] stok = {5,4,2};

int[] dizi=new int[kapasite+1];

for (int i = 0; i < 3; i++) //Tüm kutular icin

{

//Her kutuyu stok kadar tekrar kullanabiliriz

for (int s = 0; s < stok[i]; s++)

{

//diziyi tersten guncelle

for (int j = kapasite; j >=hacim[i] ; j--)

{

dizi[j] = Math.Max(dizi[j],dizi[j-hacim[i]]+kar[i]); //Al ya da alma, hangisi daha karlı

}

}

}

Console.WriteLine("Maksimum kar: "+dizi[kapasite]+ " TL");

}

}

Neden tek boyut? Zaten sadece oncekı satıra ıhtıyacımız var, aldığı kutuların hangısı olduğunu bılemeyız, bılmek ıcın 2 boyutlu olmalı!

Bu j hacimlik çanta zaten bir miktar kar getiriyordu (dp[j])

Ama şimdi elimde bir kutu var. Bu kutuyu da bu çantaya sığdırabilir miyim?

Sığarsa, eskiden kalan boşluğa (j - hacim[i]) ne koymuştum?

Onun üstüne bu kutunun karını ekleyince daha karlı olur mu?

Eğer ben bu kutuyu 10’luk çantaya koyarsam geriye 5 desilik yer kalır.

Acaba bu 5 desilik yerde daha önceden kaç lira kazanmıştım?

Diyelim orada zaten 20 TL vardı.

Bu yeni kutu 10 TL getiriyor.

O zaman toplam 30 TL olur.

Bu, eskiden 10’luk çantadan kazandığım paradan fazlaysa,

eskiyi çöpe atarım, yeni karı yazarım

Aynı kutuyu tekrar kullanmamak için ne yapıyoruz?

Bu satırı geriden ileriye doğru çalıştırıyoruz (j = kapasite → hacim[i]),

çünkü aynı kutuyu aynı aşamada iki kez kullanmak istemiyoruz.

Bir çantada iki yol var:

Bu kutuyu almadan devam et (dp[j])

Bu kutuyu alarak devam et (dp[j - hacim[i]] + kar[i])

Hangisi daha çok kar getiriyorsa onu çantaya yaz.

Senin çantan bir değer saklıyor.

Yeni kutuyla eskiden kalan boşluğu birleştirip daha karlı bir yol varsa,

çantadaki değeri güncelliyoruz.

Bu satır o karşılaştırmayı yapıyor

Eğer ileri gidersen (j = 0 → kapasite),

daha önce güncellediğin dp[j - hacim[i]] değerini tekrar kullanırsın bu da aynı kutuyu 2 kere koymuş gibi olur

Ama geriden gelirsen (j = kapasite → hacim[i]),

her dp[j - hacim[i]] hala eski halini korur yani kutuyu sadece 1 kez kullanmış olursun