1. Bir markette aşağıdaki 5 ürün vardır. Her ürünün fiyatı ve protein değeri verilmiştir.  
Toplamda **en fazla 100 TL** harcayarak **maksimum protein değeri** elde etmeye çalışın.

| **Ürün** | **Fiyat (TL)** | **Protein (g)** |
| --- | --- | --- |
| A | 20 | 15 |
| B | 50 | 40 |
| C | 30 | 25 |
| D | 70 | 50 |
| E | 60 | 45 |

1. Greedy yaklaşımla çözüm yapınız (protein/fiyat oranı ile).
2. Brute-force ile tüm kombinasyonları deneyerek en yüksek protein değerini ve hangi ürünlerin alındığını bulunuz.
3. İki yöntemi karşılaştırınız.

public static void GreedyCozum()

{

string[] urunler = { "A", "B", "C", "D", "E" };

int[] fiyat = { 20, 50, 30, 70, 60 };

int[] protein = { 15, 40, 25, 50, 45 };

double[] oran = new double[5];

int i, j;

for (i = 0; i < 5; i++)

oran[i] = (double)protein[i] / fiyat[i];

// Bubble sort ile oranlara göre azalan sırala

for (i = 0; i < 4; i++)

{

for (j = 0; j < 4 - i; j++)

{

if (oran[j] < oran[j + 1])

{

double tempOran = oran[j];

oran[j] = oran[j + 1];

oran[j + 1] = tempOran;

int tempFiyat = fiyat[j];

fiyat[j] = fiyat[j + 1];

fiyat[j + 1] = tempFiyat;

int tempProtein = protein[j];

protein[j] = protein[j + 1];

protein[j + 1] = tempProtein;

string tempUrun = urunler[j];

urunler[j] = urunler[j + 1];

urunler[j + 1] = tempUrun;

}

}

}

int toplamFiyat = 0;

int toplamProtein = 0;

Console.WriteLine("Greedy ile Seçilen Ürünler:");

for (i = 0; i < 5; i++)

{

if (toplamFiyat + fiyat[i] <= 100)

{

toplamFiyat += fiyat[i];

toplamProtein += protein[i];

Console.WriteLine ("{0} (fiyat: {1}, protein: {2})", urunler[i], fiyat[i], protein[i]);

}

}

Console.WriteLine ("Toplam Protein: {0}, Toplam Harcama: {1}", toplamProtein, toplamFiyat);

}

static void BruteForceCozum()

{

string[] urunler = { "A", "B", "C", "D", "E" };

int[] fiyat = { 20, 50, 30, 70, 60 };

int[] protein = { 15, 40, 25, 50, 45 };

int maxProtein = 0;

int enIyiToplamFiyat = 0;

int[] secilen = new int[5];

int[] enIyiSecim = new int[5];

// 2^5 = 32 kombinasyon

for (int i = 0; i < 32; i++)

{

int toplamFiyat = 0;

int toplamProtein = 0;

for (int j = 0; j < 5; j++)

{

//int sec, o anda ürünün seçilip seçilmediğini temsil eder. Bu ürün (A, B, C, D, E)

//bu kombinasyonda seçilmiş mi, seçilmemiş mi?

int sec = (i >> j) & 1;

secilen[j] = sec;

if (sec == 1)

{

toplamFiyat += fiyat[j];

toplamProtein += protein[j];

}

}

if (toplamFiyat <= 100 && toplamProtein > maxProtein)

{

maxProtein = toplamProtein;

enIyiToplamFiyat = toplamFiyat;

for (int k = 0; k < 5; k++)

enIyiSecim[k] = secilen[k];

}

}

Console.WriteLine("Brute-force ile En İyi Seçim:");

for (int i = 0; i < 5; i++)

{

if (enIyiSecim[i] == 1)

Console.WriteLine ("{0} (fiyat: {1}, protein: {2})", urunler[i], fiyat[i], protein[i]);

}

Console.WriteLine(“Toplam Protein: {0}, Toplam Harcama: {1}", maxProtein, enIyiToplamFiyat);

}

Greedy yöntemde C, A, B seçilir (toplam protein = 80)

Brute-force yöntemde A + B + C de seçilir, ancak bazı kombinasyonlarda E + C + A gibi varyantlar 85 protein verebilir.

Greedy her zaman en iyiyi vermez çünkü yerel optimuma göre karar verir.

| **i (decimal)** | **i (binary 5 bit)** | **Seçilen Ürünler (A~E)** |
| --- | --- | --- |
| 0 | 00000 | hiçbiri seçilmedi |
| 1 | 00001 | sadece A seçildi |
| 3 | 00011 | A ve B seçildi |
| 19 | 10011 | A, B ve E seçildi |
| 31 | 11111 | hepsi seçildi |

### i >> j

i sayısını **j bit sağa kaydırır.**

* Örnek: i = 19 (ikilik: 10011)
  + - j = 0 ⇒ i >> 0 = 10011
    - j = 1 ⇒ i >> 1 = 1001
    - j = 2 ⇒ i >> 2 = 100
    - j = 3 ⇒ i >> 3 = 10
    - j = 4 ⇒ i >> 4 = 1

Sağa kaydırmak demek, en sağdaki j kadar basamağı **atmak** demek.

2. Aşağıdaki iki string veriliyor:

* birinci = "AXYT"
* ikinci = "AYZX"

1. Bu iki string için **en uzun ortak subsequence** (LCS) uzunluğunu **recursive yöntemle** hesaplayın.
2. Aynı işlemi **dynamic programming tablosu** ile yapın.
3. Hem LCS uzunluğunu, hem de ortak subsequence’in kendisini yazdırın.

Subsequence: Karakterler sıralı kalmalı ama ardışık olmak zorunda değil.

AXYT ve AYZX için en uzun ortak subsequence: "AY" ya da “AX”

|  | **0** | **A** | **Y** | **Z** | **X** |
| --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 |
| A | 0 | 1 | 1 | 1 | 1 |
| X | 0 | 1 | 1 | 1 | 2 |
| Y | 0 | 1 | 2 | 2 | 2 |
| T | 0 | 1 | 2 | 2 | 2 |

Ortak harfler neler?

A → var

X → var

Y → var

T → sadece birinci dizide

Hangi sıralı yollar mümkün?

"A" → "Y"

"A" → "X"

"X" → "Y" (ama sıralama bozuluyor çünkü A'dan sonra X olmaz bu dizilerde)

static int LCSRecursive(string birinci, string ikinci, int i, int j)

{

if (i == 0 || j == 0)

return 0;

//AXYT //AYZX

if (birinci[i - 1] == ikinci[j - 1])

return 1 + LCSRecursive(birinci, ikinci, i - 1, j - 1);

else

{

int a = LCSRecursive(birinci, ikinci, i - 1, j);

int b = LCSRecursive(birinci, ikinci, i, j - 1);

return (a > b) ? a : b;

}}

**Adım 1:** **(i=4, j=4)**

birinci[3] = 'T' vs ikinci[3] = 'X' eşleşmedi

İki yol denenecek:

LCS("AXY", "AYZX") → yani i-1, j

LCS("AXYT", "AYZ") → yani i, j-1

**Adım 2:** LCS("AXY", "AYZX") (i=3, j=4)

birinci[2] = 'Y' vs ikinci[3] = 'X' eşleşmedi

LCS("AX", "AYZX")

LCS("AXY", "AYZ")

**Adım 3:** LCS("AX", "AYZX") (i=2, j=4)

birinci[1] = 'X' vs ikinci[3] = 'X' eşleşti

* 1 puan kazan
* İkisini de küçült: LCS("A", "AYZ")

**Adım 4:** LCS("A", "AYZ") (i=1, j=3)

birinci[0] = 'A' vs ikinci[2] = 'Z' eşleşmedi

2 yol denenir:

1. LCS("", "AYZ") → i=0 return 0
2. LCS("A", "AY")

**Adım 5:** LCS("A", "AY") (i=1, j=2)

birinci[0] = 'A' vs ikinci[1] = 'Y' eslesmedi

Yine 2 yol:

1. LCS("", "AY") → i=0 return 0
2. LCS("A", "A")

**Adım 6:** LCS("A", "A") (i=1, j=1)

birinci[0] = 'A' vs ikinci[0] = 'A' eşleşti

* 1 puan kazan
* Kalan: LCS("", "") → i=0, j=0 return **0**

Toplam: 1

Adım 6 → 1 puan

Adım 5 → max(0, 1) = 1

Adım 4 → max(0, 1) = 1

Adım 3 → eşleştiği için 1 + 1 = 2

Şimdi geri gidip diğer yolu deneyelim:

**Adım 2'nin diğer yolu**: LCS("AXY", "AYZ")

'Y' vs 'Z' eslesmedi

2 yol:

* LCS("AX", "AYZ")
* LCS("AXY", "AY")

Bu dallanma da 1 puan verir çünkü "A" ve "A" eşleşir ama daha uzun olan "AX" yolu 2 puan verecektir.

Bütün olasılıklar denendiğinde:

* "AX" yolu → 2
* "AY" yolu → 2

LCS algoritması ilk hangisine ulaştıysa onu döner (sıralı kıyaslamaya göre "AX" önce bulunmuş olabilir)

En uzun ortak sıralı dizi: "AX" veya "AY"

Uzunluk: 2

**.Bu kodu dp ile çözüp, izlediği yolu backtracking ile yazdırınız.**

3. Bilgisayar mühendisliği öğrencisi Elif, kampüste ders çalışmak için kütüphaneye gitmek istiyor.  
Kampüs içinde 6 önemli nokta var (noktalar numaralandırılmıştır):

* 0: Yurt
* 1: Kantin
* 2: Sosyal Tesis
* 3: Amfi
* 4: Kütüphane
* 5: Mühendislik Fakültesi

Elif, yurdun önünden çıkıyor (başlangıç: 0) ve kütüphaneye (hedef: 4) ulaşmak istiyor.

Kampüs içindeki yollar yönlüdür ve her birinin geçiş süresi farklıdır (dakika cinsinden).  
Aşağıdaki tablo kampüs içi yolları ve geçiş sürelerini göstermektedir:

| **Başlangıç Noktası** | **Varış Noktası** | **Geçiş Süresi (dk)** |
| --- | --- | --- |
| 0 (Yurt) | 1 (Kantin) | 2 |
| 0 (Yurt) | 2 (Tesis) | 4 |
| 1 (Kantin) | 2 (Tesis) | 1 |
| 1 (Kantin) | 3 (Amfi) | 7 |
| 2 (Tesis) | 4 (Kütüphane) | 3 |
| 3 (Amfi) | 4 (Kütüphane) | 1 |
| 2 (Tesis) | 3 (Amfi) | 2 |
| 4 (Kütüphane) | 5 (Müh. Fak.) | 5 |

Elif’in, Yurt (0) noktasından çıkıp Kütüphane (4) noktasına en kısa sürede ulaşması için:

1. Hangi güzergâhtan gitmesi gerektiğini adım adım belirtiniz.
2. Bu yolun toplam süresini bulunuz.

using System;

class Program

{

static void Main()

{

int dugumSayisi = 6;

int sonsuz = 9999;

// Komşuluk matrisi (başlangıçta tüm yollar kapalı)

int[,] gecisSuresi = new int[dugumSayisi, dugumSayisi];

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

{

for (int j = 0; j < dugumSayisi; j++)

{

gecisSuresi[i, j] = sonsuz; //henüz gerçek yolları yazmadık

}

}

// Yolları tanımlayalım

gecisSuresi[0, 1] = 2; // Yurt → Kantin

gecisSuresi[0, 2] = 4; // Yurt → Tesis

gecisSuresi[1, 2] = 1; // Kantin → Tesis

gecisSuresi[1, 3] = 7; // Kantin → Amfi

gecisSuresi[2, 4] = 3; // Tesis → Kütüphane

gecisSuresi[2, 3] = 2; // Tesis → Amfi

gecisSuresi[3, 4] = 1; // Amfi → Kütüphane

gecisSuresi[4, 5] = 5; // Kütüphane → Müh. Fak. (kullanılmayacak)

int[] enKisaSure = new int[dugumSayisi]; //Her düğüme gitmenin en kısa süresini burada tutacağız

bool[] ziyaretEdildi = new bool[dugumSayisi]; // Bu düğüm daha önce ziyaret edildi mi? (true/false)

int[] oncekiDugum = new int[dugumSayisi]; //Bu düğüme gelirken hangi düğümden geldik?

// Başlangıç noktası: Yurt (0) (baslangic 4 olsaydı??)

int baslangic = 0;

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

{

enKisaSure[i] = sonsuz; //Tüm düğümlerin en kısa süresini sonsuz yapıyoruz başta hiçbir yere yol //bilinmiyor

ziyaretEdildi[i] = false; //Hepsi ziyaret edilmemiş gibi işaretleniyor

oncekiDugum[i] = -1; //henüz hiçbiri bir başkasından gelmiyor -1 ile başlatıyoruz

}

enKisaSure[baslangic] = 0; //Başlangıç noktası zaten orada olduğumuz için süresi 0 dakika

//Bu döngüyle her seferinde en kısa süresi olan yeni bir düğüm bulup ona geçiyoruz

//Graftaki tüm düğümleri sırayla merkez nokta olarak kabul ederız

for (int tur = 0; tur < dugumSayisi; tur++)

{

//Henüz ziyaret edilmemiş düğümler arasında en kısa sürede ulaşılabilen düğümü bulmak için

//Başta en kısa süremiz yine "sonsuz"

int enKisa = sonsuz;

int secilenDugum = -1;

// Henüz ziyaret edilmemiş en kısa mesafeli düğümü bul

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

{

if (!ziyaretEdildi[i] && enKisaSure[i] < enKisa)

{

enKisa = enKisaSure[i];

secilenDugum = i;

}

}

//Eğer artık seçilebilecek düğüm kalmadıysa algoritmayı bitiriyoruz

if (secilenDugum == -1)

break;

ziyaretEdildi[secilenDugum] = true; //Bu düğüm artık ziyaret edildi bir daha gitmeyeceğiz

// Komşuları kontrol et

for (int j = 0; j < dugumSayisi; j++) //Bu düğümden gidilebilecek komşular kontrol ediliyor

{

if (gecisSuresi[secilenDugum, j] != sonsuz && !ziyaretEdildi[j])

{

int yeniSure = enKisaSure[secilenDugum] + gecisSuresi[secilenDugum, j];

if (yeniSure < enKisaSure[j]) // Yeni bir yol bulunduysa ve önceki süreden daha kısa ise

{

enKisaSure[j] = yeniSure;

oncekiDugum[j] = secilenDugum; //Bu düğüme gelirken hangi düğümden geldik kaydediliyor

}

}

}

}

// Kütüphane (4) noktasına giden yol

//hedef düğüme hangi yollardan geçerek ulaşıldığını geriye doğru bulmak için

int hedef = 4;

int[] yol = new int[dugumSayisi];

int sayac = 0; //indisleri takip etsin

int dugum = hedef;

//oncekiDugum[] dizisini kullanarak yolu geriye doğru topluyoruz

//Her adımda bir önceki noktaya gidiyoruz ta ki -1 gelene kadar

// Ben kütüphaneye geldim ama nasıl geldim önce tesisten mi?

while (dugum != -1) // dizide bir düğümün değeri -1 ise, o başlangıç noktasıdır Yurt

{

yol[sayac] = dugum; // Şu an bulunduğumuz düğümü yol[] dizisine yaz

sayac++;

dugum = oncekiDugum[dugum];

}

// Yolu tersten yazdır çünkü baştan sona kadar olan yol elimizde ters sırada tutuldu

Console.Write("En kısa yol: ");

for (int i = sayac - 1; i >= 0; i--)

{

Console.Write(yol[i]);

if (i > 0) Console.Write(" --->");

}

Console.WriteLine();

Console.WriteLine("Toplam süre: " + enKisaSure[hedef] + " dakika");

}

}

enKisaSure = [0, 2, 4, 9999, 9999, 9999];

ziyaretEdildi = [true, false, false, false, false, false];

* i = 0 zaten ziyaret edilmiş geç
* i = 1 ziyaret edilmemiş && süresi 2 şu an en kısa bu
* i = 2 süresi 4 ama 2 daha kısa olduğu için değiştirmiyoruz
* i = 3-4-5 hepsi 9999 ulaşılamaz

Sonuç: secilenDugum = 1

Diyelim secilenDugum = 1 (Kantin)

* j = 0 ➝ Kantin → Yurt yönü yok (veya ziyaret edilmiş) → geç
* j = 1 ➝ Aynı düğüm → geç
* j = 2 ➝ Kantin → Tesis: geçiş süresi = 1
  + Şu anki yol: enKisaSure[1] = 2
  + yeniSure = 2 + 1 = 3
  + enKisaSure[2] zaten 4 ise → güncellenir: enKisaSure[2] = 3
* j = 3 ➝ Kantin → Amfi: geçiş süresi = 7
  + yeniSure = 2 + 7 = 9
  + enKisaSure[3] hâlâ sonsuzsa → güncellenir: enKisaSure[3] = 9

Bu döngü, her turda yeni noktaları keşfeder ve önceki sürelerden daha kısa yollar buldukça günceller.

* Başlangıç noktasından diğer düğümlere olan mesafeler sonsuz olarak başlatılır. Sadece başlangıç düğümünün mesafesi sıfırdır.
* Tüm düğümler “ziyaret edilmedi” olarak işaretlenir.
* Henüz ziyaret edilmemiş düğümler arasında, başlangıç noktasından en kısa mesafede olan düğüm seçilir.
* Seçilen düğüm ziyaret edilmiş olarak işaretlenir.
* Bu düğümden komşu düğümlere gidilip gidilemeyeceği kontrol edilir.
* Gidilebiliyorsa, komşu düğüme ulaşmak için gereken toplam süre hesaplanır.
* Bu yeni süre, o düğüme daha önce bilinen süreden kısa ise güncellenir.
* Aynı zamanda o düğüme hangi düğümden gelindiği kaydedilir.
* Tüm düğümler ziyaret edilene kadar 3. adıma geri dönülür.
* Hedef düğüme ulaşıldığında, geldiği yerler oncekiDugum[] dizisinden geriye doğru takip edilerek yol oluşturulur.
* Yol ters çevrilerek başlangıçtan hedefe gidiş sıralı olarak elde edilir.

//sıralama algoritmaları

//bitwise

//dynamic programming (knapsack, nakliye kamyonu)

//backtracking

//integer programming

//recursion (brute force birlikte)

//greedy

//lcs (subsequence, substring)

//graflar

//sayı manipülasyonu (ya dizi içinde ya da integer programming içinde)

//otomata

//test & klasik olabilir.

//Zaman karmaşıklığı

//huffman