Warehouse Location Problem

📌 Hazırlayan: Kağan Can Şit - 202802007

Ödevin çözümü için genel yaklaşım olarak Greedy yaklaşım ve cezalandırma kullandım. Mantıksal olarakher kişi tarafından farklı algortima yaklaşımı ile geliştirilebileceği için kurduğum algoritmanın mantıksal olarak temelini ödev tanımında bulunan soru üzerinden gerçekleştirdiğim yaklaşımlar üzerinden anlatacağım.

Ödev tanımında verilen örnek değerleri hatırlayalım.

Input	**Output**
100 100.123 100 100.456 100 100.789 50 100.1 200.2 2000.3 50 100.4 200.5 2000.6 75 200.7 100.8 2000.9 75 200.1 200.11 100.12	1002.888 1 1 0 2

Nasıl Yaklaşmalıyız?

Şimdi elimizdeki örneği ele almadan önce minimum maliyet için neler gerekli? Sorusunu sormalı ve bu sorunun cevapları ışığında yol almalıyız.

- Yüksek depolama talebini düşük fiyata vermek avantajlıdır. Bunu şu şekilde düşünebilirsiniz. 50 ve 100 iki yer talebiniz var. İkisini de 100 birime vereceksiniz. 100/50 - 100/100 şeklinde bakarsanız. Birinci durum için 1 deponun maliyeti iki birim, ikinci durum ise bir birimdir. Bu sebepten dolayı algoritmamızı kurarken büyük talebe sahip müşteriler, düşük talebe sahip müşterilerden önceliklidir.
- Yeni deponun inşa maliyeti ve yeni depoya ulaşım maliyetinin toplamı, eski depoya ulaşım maliyetinden yüksekse(daha önce o depoyu kurduğumuz için kurulum maliyeti yok) ve eski depoda bu ürünler için yer varsa ürünün eski depoya gitmesi daha avantajlıdır.
- Müşterileri önceliklendirdik, peki depoları nasıl önceliklendirmeliyiz? En düşük kurulum maliyeti olan ve en düşük kurulum maliyetli depoya gidiş maliyetinin toplamı yeni depo kurulum maliyeti ve yeni depoya gidiş maliyeti toplamından küçükse en düşük maliyetli depoya git. Neden daha iyi fiyatlı yol varken diğerini tercih edeyim ki?
- Bunun yanı sıra algoritmamızın yeterli kapasiteye sahip olmayan yerler için bir cezalandırma sistemine ihtiyacı var. Çünkü eğer yer yoksa tekrar tekrar aynı alanın kontrol edilmesine gerek yok. Bunun için seçilen depoda müşterinin talebini karşılayacak kadar alan yoksa bu durum zatn kullanışsızdır ve tekrar sorgulanmamalıdır. Bunun için o depoya gidiş maliyetini arttırıyoruz. (x3) Bu sayede bu durum algoritmanın sorgu aralığının dışına çıkmış oluyor.

Bu yaklaşımlar ışığında örneğimize dönecek olursak en yüksek talebe sahip müşteri ve onun yol maliyetini alıyoruz.

75 Talep

Depoya gidiş maliyeti→ 100.8

Deponun Kurulum Maliyeti → 100.456 (2.)

→ Maliyet = 201.256

Depoda yer var.

Bu değerleri en düşük maliyetli depo kurulum maliyeti ve en düşük maliyetli depaya gidiş yolu maliyeti ile kıyaslayalım.

En düşük maliyetli depo → 100.789

En düşük maliyetli depoya o müşterinin gidiş maliyeti → 2000.9

→ Maliyet = 2,101.689

Bu durumda yeni depoyu tercih et. (2)

Daha sonra tekrar müşterinin en çok talebi olana gidiyoruz.

75 Talep

Depoya gidiş maliyeti→ 100.12

Deponun Kurulum Maliyeti → 100.789 (3.)

→ Maliyet = 200.9

En düşük maliyetli depo → 100.789

En düşük maliyetli depoya o müşterinin gidiş maliyeti → 100.12

→ Maliyet = 200.9

Eski depoya gitme maliyeti → 200.11

Burada en düşük maliyetli depoya gönder. (3) Eski deponun maliyeti bu depodan yüksek.

Daha sonra tekrar müşterinin en çok talebi olana gidiyoruz.

50 Talep

Depoya gidiş maliyeti→ 100.1

Deponun Kurulum Maliyeti → 100.123 (1.)

→ Maliyet = 200.2

En düşük maliyetli depo → 100.789

En düşük maliyetli depoya o müşterinin gidiş maliyeti → 2000.3

→ Maliyet = 2101.0

Eski depoya gitme maliyeti → 2000.3

Burada en düşük maliyetli depo ve en düşük maliyetli depo aynı fakat maliyetleri yeni depodan fazla olduğu için yeni depoya gönder. (1.)

Daha sonra tekrar müşterinin en çok talebi olana gidiyoruz.

50 Talep

Depoya gidiş maliyeti→ 100.4

Deponun Kurulum Maliyeti → 100.123 (1.)

→ Maliyet = 200.5

En düşük maliyetli depo → 100.789

En düşük maliyetli depoya o müşterinin gidiş maliyeti → 2000.6

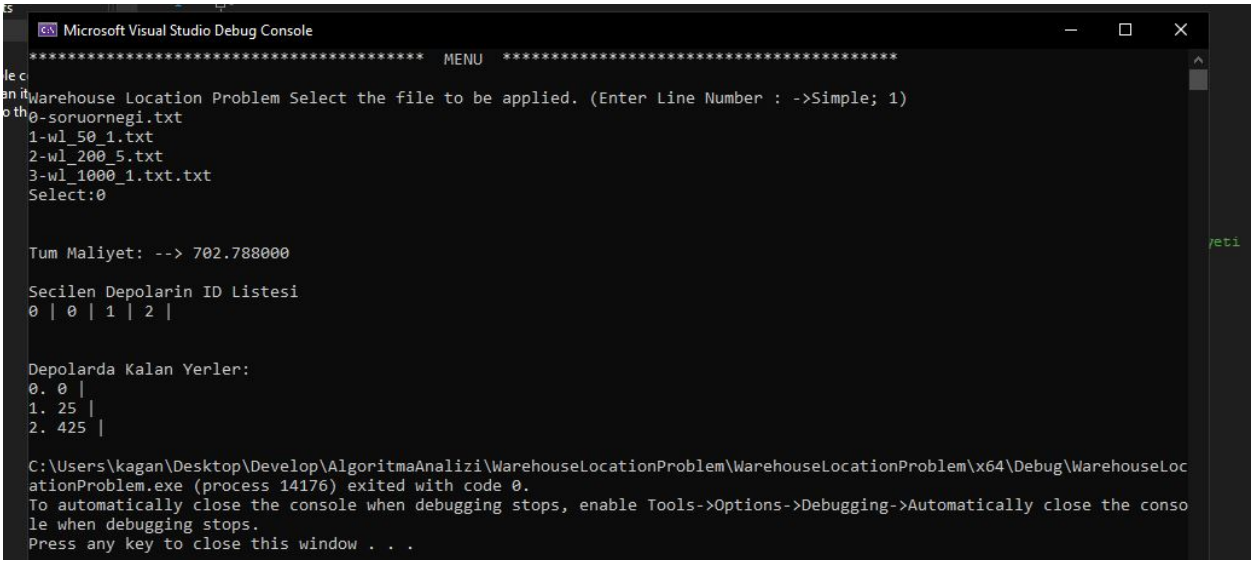
→ Maliyet = 2001.3

Eski depoya gitme maliyeti → 100.4

Burada eski depo seçeneğini farkı açık ara gözüküyor. Burada eski depo üzerinden depoya gönder. (1.) şeklinde algoritma çalışır.

Not: Her aşamada kapasite kontrolü yapılmaktadır.

Elde edilen sonuç şu şekildedir;



🎯Sonuç;

Burada koşulan algoritma her ne kadar belirli şartları kontrol ediyor olsa da tüm olasılıkları gözden geçirmez. Bu sebeple optimal bir yaklaşım değildir. Farklı dosya değerleri için farklı verimler sergileyebilir. Unutulmaması gereken bu algortimanın bir GREEDY yaklaşım algoritması olduğudur.

Bizim burada yaptığımız fedakarlık optimala olabildiğince yakın fakat zaman olarak tüm olasılıklarını denemekten daha kısa sürede dönüt verebilecek bir algoritma ortaya koyamaktır.

İyileştirmek için çeşitli işlemler gerçekleştirilebilir. Şu an için yazılın algoritmanın kilit noktası “void findCustomerMinCost(int customer)” fonksiyonu içerisinde yer alan aşağıda verilen kısımdır.

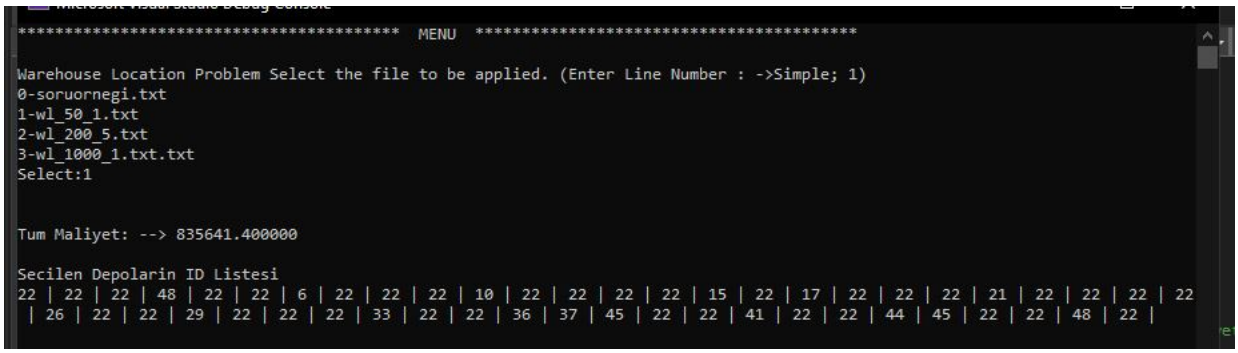
```
//Depoda yer yoksa onlem!
  if (capasityWH.data[temp] < customerCapasity.data[customer])
  {
    roadCost[customer].data[temp] *= 3; //O depoda yer yoksa kullanilmasi zaten mumkun degil. Maliyetini arttiriyoruz.
  }
  else
  {
    //Mantikin Ozu -> Yeni Deponun Kurulum Maliyeti + Yeni Depoya Ulasim Maliyet > Eski Depoya Yolculuk Maliyetinden fazlaysa ve eski depod
    if (customerWHCounter != 0 && (buildCostHW.data[temp] + roadCost[customer].data[temp]) > roadCost[customer].data[lastWHChoise] && capas
    {
      SalesOperation(lastWHChoise, customer);
    }
    else
    {
      customerWHCounter++;

      //Eger en dusuk maliyetli depo + en dusuk maliyetli depoya yolculuk maliyeti < Yeni Depo Kurulum Maliyeti + Yeni Depoya Yolculuk Mali
      if (buildCostHW.data[minBCostIndis] + roadCost[customer].data[minBCostIndis] < buildCostHW.data[temp] + roadCost[customer].data[temp]
      {
        SalesOperation(minBCostIndis, customer);
        lastWHChoise = minBCostIndis; //En son secilen depo. -> Kurulum maliyeti farki icin gerekli.
      }
      else
      {
        SalesOperation(temp,customer);
        lastWHChoise = temp;  //En son secilen depo. -> Kurulum maliyeti farki icin gerekli.
      }
    }
  }
```

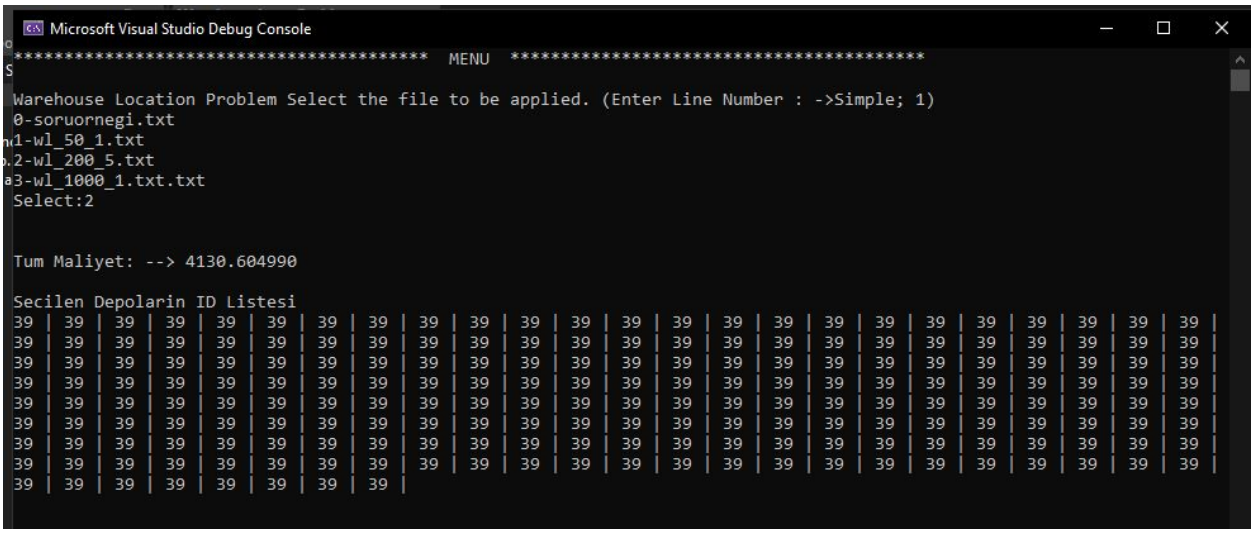
EK;

Elimizdeki artan verilere sahip dosyalar için sonuçlar ise şu şekildedir.

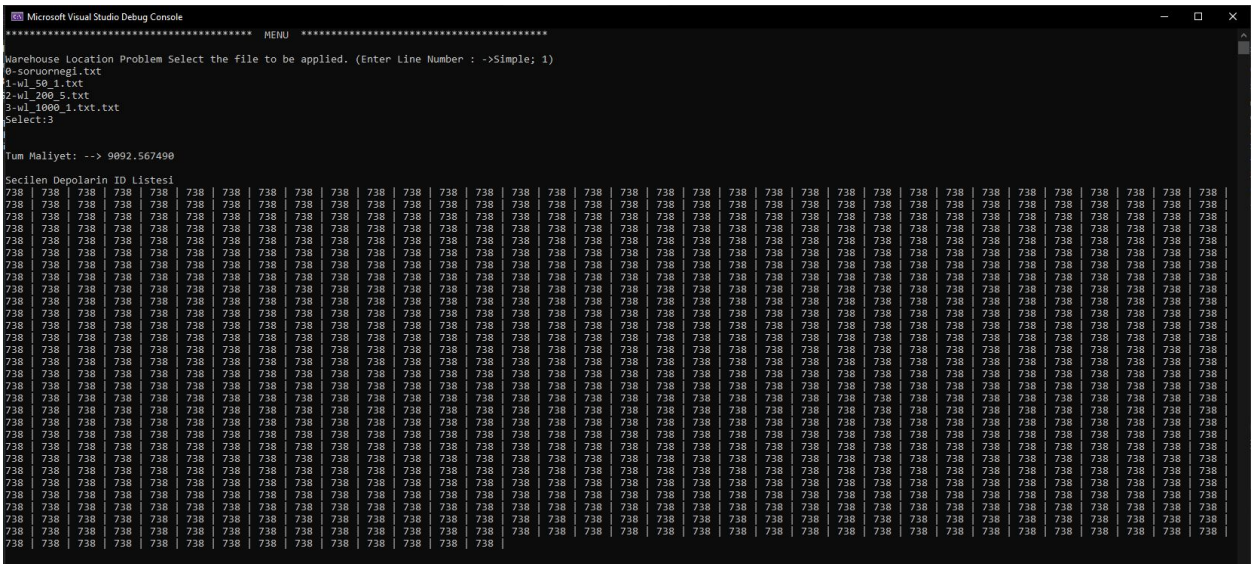
50 Değerlik Liste;



200 Değerlik Liste;



1000 Değerlik Liste;



Bu algoritmaya geliştirmek için herhangi bir önerin varsa lütfen bir issue oluştur veya bana mail yaz. (kagancansit@hotmail.com)

Umarım ilgini çekmiş ve yeni bir şeyler keşfetmişsindir. Görüşmek üzere...