

ONUR
SEZER
121044074

CSE 321 - HW05

1) a) Greedy algoritmasıyla mümkün olan ve sonuca en yakın olan seçim yapılır. Yani basitçe bir seçim yapılması gerektiğinde sonuca en çok yaklaştıracak olan seçim yapılmasını önerir. Bazı sorunları tahminen veya sezgisel olarak gözlemimiz gerekmektedir, işte bunu sağlayan Greedy Algoritmasıdır. Greedy yönlü ve maliyetli graflarda kullanılır.

b) Greedy algoritması her zaman en iyi seçimi vermez. Örneğin para üzünü verilmesi için greedy algoritmasını kullanalım. Satıcı kendisinden alışıveriş yapan kişiye 24 lira para vermesi gereksin. Para birimlerini 20, 19, 5 ve 1 olsun. Greedy algoritmasına göre 24'e tamamlamak için elindeki para birimlerinden sonuca en çok yaklaştıran 20 lirayı seçecektir. Daha sonra geriye kalan boşluğu ($24 - 20 = 4$) doldurmak için elindeki tek imkanı olan 4 tane 1'lik ile dolduracak ve toplamda 5 adet bozuk parayı müşteriye geri verecektir. Oysaki aynı problem bir 19'lık bir 5'lik bozuk paralar ile çözümlenerek 2 bozuk para vermek mümkün olabilirdi.

2) a) Doğru .

b) Yanlış .

c) Doğru .

d) Yanlış .

3)

// Python code

```
##Prints a maximum set of activities that can be done by a
single person, one at a time

# n --> Total number of activities
# s[]--> An array that contains start time of all activities
# f[] --> An array that contains finish time of all activities

def printMaxActivities(s, f):
    n = len(f)
    print ("The following activities are selected")

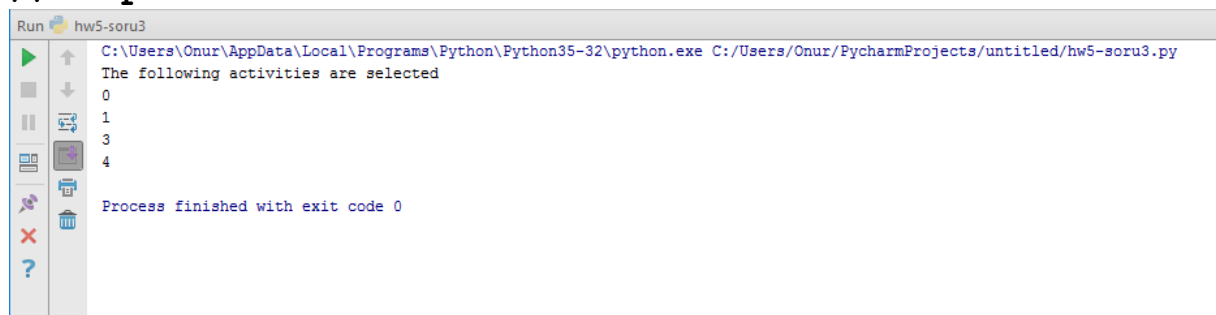
    # The first activity is always selected
    i = 0
    print (i)

    # Consider rest of the activities
    for j in range(n):

        # If this activity has start time greater than
        # or equal to the finish time of previously
        # selected activity, then select it
        if s[j] >= f[i]:
            print (j)
            i = j

# Driver program to test above function
s = [1, 3, 0, 5, 8, 5]
f = [2, 4, 6, 7, 9, 9]
printMaxActivities(s, f)
```

//Output



```
Run hw5-soru3
C:\Users\Onur\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/Onur/PycharmProjects/untitled/hw5-soru3.py
The following activities are selected
0
1
3
4
Process finished with exit code 0
```

- 1) Bitirme sürelerine göre sırala etkinlikleri sırala
- 2) Sıralanmış diziden ilk etkinliği seç ve yazdır
- 3) Sıralama dizisindeki kalan etkinlikler için aşağıdakileri yapın.
 - a) Etkinliğin başlangıç saati daha önce seçilen etkinliğin bitiş süresinden büyükse, bu etkinliği seç ve yazdır.

Complexities -> $\Theta(n)$ 'dir. (Bir tane for loopuna sahip olduğu için)

4)

// Python code

```
import numpy as np

def nearestNeighbor(A, start):
    path = [start]
    cost = 0
    N = A.shape[0]
    mask = np.ones(N, dtype=bool) # boolean values indicating which
                                   # locations have not been visited
    mask[start] = False

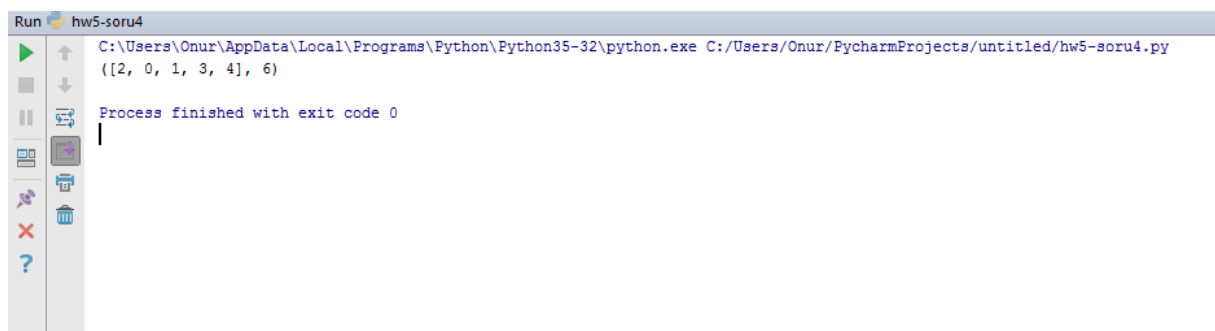
    for i in range(N-1):
        last = path[-1]
        # find minimum of remaining locations
        next_ind = np.argmin(A[last][mask])
        # convert to original location
        next_loc = np.arange(N)[mask][next_ind]
        path.append(next_loc)
        mask[next_loc] = False
        cost += A[last, next_loc]

    return path, cost

A = np.array([
    [0, 2, 1, 2, 2],
    [2, 0, 2, 1, 1],
    [1, 2, 0, 1, 2],
    [2, 1, 1, 0, 2],
    [2, 1, 2, 2, 0]])

print (nearestNeighbor(A,2))
```

// Output



```
Run hw5-soru4
C:\Users\Onur\AppData\Local\Programs\Python\Python35-32\python.exe C:/Users/Onur/PycharmProjects/untitled/hw5-soru4.py
([2, 0, 1, 3, 4], 6)
Process finished with exit code 0
```

Travelling Salesman için greedy algoritmasını kullanmak için Nearest-neighbor ve Multifragment-heuristic algoritmaları vardır. Yukarda implement ettiğim Nearest-neighbor algoritmasıdır. nearestNeighbor methodu parametre olarak konumlar arasındaki mesafeyi belirten bir NxN dizisi ve başlangıç vertexini alır, pathi ve costu return eder.

Complexities -> $\theta(n^2)$ 'dir.

5)

// Python code

```
colors = ['Red', 'Blue', 'Green', 'Yellow', 'Black']

states = ['Ankara', 'Konya', 'Afyon', 'Eskişehir']

neighbors = {}
neighbors['Ankara'] = ['Konya', 'Afyon']
neighbors['Konya'] = ['Afyon', 'Ankara', 'Eskişehir']
neighbors['Afyon'] = ['Ankara', 'Konya', 'Eskişehir']
neighbors['Eskişehir'] = ['Konya', 'Afyon']

colorsOfStates = {}

def checkState(state, color):
    for neighbor in neighbors.get(state):
        colorOfNeighbor = colors_of_states.get(neighbor)
        if colorOfNeighbor r == color:
            return False
    return True

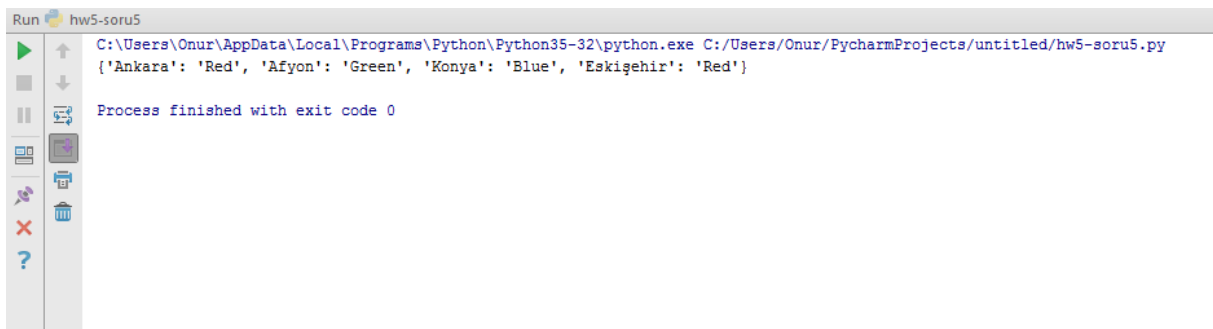
def getColorForState(state):
    for color in colors:
        if promising(state, color):
            return color

def main():
    for state in states:
        colorsOfStates [state] = getColorForState(state)

    print (colorsOfStates)

main()
```

// Output



Complexty => checkState methodundaki condition sağlanırsa
Best Case : $O(n^2)$, diğer durumda **Worst Case** : $O(n^3)$ olur

6) Evet, greedy algoritması daima optimal bir çözüm getirir. Aşağıdaki formül i_1, i_2, \dots, i_n işlerinin sıralanması için sistemdeki toplam süre formülünü verir

$$\left[t_{i_1} + (t_{i_1} + t_{i_2}) + \dots + (t_{i_1} + t_{i_2} + \dots + t_{i_n}) = n \cdot t_{i_1} + (n-1) t_{i_2} + \dots + t_{i_n} \right]$$

Böylece $n, n-1, \dots, 1$ sayısına "ağırlık" t_1, t_2, \dots, t_n numaraları sırasıyla atandı. Böyle bir toplamı en aza indirmek için, daha büyük sayılara daha büyük t atamalıyız. Başka bir deyişle, işler ıera sürelerinin azalan sırasına göre yerine getirilmelidir.

İşlerin bazı k için $t_{i_k} > t_{i_{k+1}}$ 'in olduğu bazı sırasıyla i_1, i_2, \dots, i_n 'de görüldürse, böyle bir sıra için toplam sistemin süresi azaltılabilir k ve $k+1$ işlerini değiştirerek elde edilen diğer iş emridir. Sistemdeki zaman bu iki iş haricinde hep aynı kalacaktır. Bu nedenle yeni sipariş için sistemdeki toplam süre ile swap öncesindeki toplam süre arasındaki fark

$$\left[\left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_{k+1}} \right) + \left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_{k+1}} + t_{i_k} \right) \right] -$$

$$\left[\left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_k} \right) + \left(\sum_{j=1}^{k-1} t_{i_j} + t_{i_k} + t_{i_{k+1}} \right) \right]$$

$$= t_{i_{k+1}} - t_{i_k} < 0$$

7) Tüm matrix tipleri aşğıdaki işlemi n defa tekrarlar.
işaretlenmemiş satırdaki ve maliyet matrixinin sütun-
larındaki en küçük ögeyi seç ve sonra satırını ve
sütununu işaretle.

İlk satırdan başlayıp maliyet matrixinin son satırı ile
biten satırda önceden işaretli bir sütunda bulunmayan
en küçük ögeyi seçilir. Böyle bir öge seçildikten sonra
aynı sütundan başka bir öge seçmekten kaçınmak için
sütun işaretlenir.

Hiç bir versiyonda daima optimal bir çözüm getirmez.
Örnek olarak aşğıdaki matrix verilebilir

$$C = \begin{bmatrix} 1 & 2 \\ 2 & 100 \end{bmatrix}$$

8) Algorithm Change ($n, D[1..m]$)

```
for  $i \leftarrow 1$  to  $m$  do  
     $C[i] \leftarrow n / D[i]$   
     $n \leftarrow n \bmod D[i]$   
if  $n = 0$   
    return  $C$   
else  
    return "cevap yok"
```

// Kalan sıfır geldiğinde durabileceği için time efficiency $O(m)$ 'dir.

9)

a) En basit ve en akıllı çözüm, tüm kenar ağırlıklarını 1'e atamaktır.

b) Deep-first search tree elde etmek için deep-first search aramayı geçici uygulamak kararsız olarak daha basittir ve onların bitişik listeleriyle daha basit ve gliklilik listeleri tarafından temsil edilen seyrek grafikler için daha hızlı listeler.

10)

Travelling Salesman

Travelling Salesman 2012 entelektüel gerilim çözüme dört matematikçiler hakkında bir film P versus NP karşı sorunu , tarihinin en zorlu matematik problemlerinden birini. Başlık atıfta Seyahat satıcı problemi , zor olduğu düşünülen diğer matematiksel problemleri çözüme için bir anahtar gibi davranan bir optimizasyon problemi; Hızlı bir seyirci satıcısı algoritmasının, varsa, çok sayıda faktoring gibi birçok zorlu görev için hızlı algoritmalara dönüştürülebileceği kanıtlanmıştır. Birçok veri şifreleme düzenleri verilerini korumak için tamsayılar faktoring zorluk güveniyor, bu kişisel yazışmalar, banka hesapları ve muhtemelen devlet sırları gibi özel veri erişimi sağlayacak.Hikaye yazdığı ve yönettiği tarafından bir Timothy Lonzon ve Cumartesi, 16 Haziran'da Philadelphia Uluslararası evinde galası Pennsylvania Üniversitesi ve Cambridge Üniversitesi'nde gösterimleri de dahil olmak üzere 4 kıtada yayılan 8 ülkede gösterimleri, sonra filmi, 10 Eylül 2013 tarihinde dünya çapında piyasaya sürüldü. Dört matematikçiler toplandı ve bir üst yetkilisi ile karşılayan Amerika Birleşik Devletleri Savunma Bölümü . Bazı görüşmelerden sonra grup, çözümüne kimin kime güvenileceğine ve kontrol edeceğine dikkat etmesi gerektiğini kabul eder. Resmi makamlar, algoritmalarının bir kısmıyla karşılıklı olarak 10 milyon dolar ödülünü sunuyor ve kaygılarını dile getirerek onları sallıyorlar. Dört kişiden yalnızca bir tanesi satışa karşı çıkıyor ve bunu yaparken çözümün kendi kısmı hakkında karanlık bir gerçeği ortaya çıkarmak zorunda kalıyor. Bununla birlikte, hükümet lisansı imzalamadan önce, keşiflerinin ahlaki sonuçlarıyla uğraşıyorlar.