Nama : Stephen Prasetya Chrismawan
NIM : H1D021025

UTS ALGEN (Knapsack)

1. Untuk membuat code knapsack kita perlu mengimpor library bantuan yang diperlukan

```
import random

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn.preprocessing import MinMaxScaler

np.set_printoptions(suppress=True)  # Untuk menghindari notasi

ilmiah
```

2. Membaca dataset dengan library dari pandas, kemudian hanya mengambil 3 kolom yaitu size, cound, juga freq.

```
#membaca dataset

df = pd.read_csv("tb_data_knapsack.csv")

df = df[["size", "count","freq"]]

print(df)

scaler = MinMaxScaler()

scaler.fit(df)

print(scaler.transform(df))


selected_columns = ["size", "count", "freq"]

df_selected = df[selected_columns]

data_array = df_selected.values
```

3. Membuat fungsi untuk mencari nilai fitness, dengan bobot yang sudah ditentukan. Alasan bobot seperti tersebut adalah tidak terlalu jauh bea antar 3 bobot dan jika dijumlahkan hasilnya 1 sehingga ideal.

```
w1 = 0

w2 = 0

w3 = 0



def fitness( fds, fdc, fdf):

   w1 = 0.3

   w2 = 0.4

   w3 = 0.3


   res = w1*(1-fdf)+ w2*(1-fdc) + w3*fds

   return res
```

4. Membuat kelas Gen. Dimaksudkan untuk menyimpan nilai size, count, freq dari setiap gen. Yang kemudian akan disimpan di array list gen sebagai kamus untuk diambil datanya saat dibutuhkan oleh hkromosom yang digenerate

```
class Gen :

   def __init__(self, size, count, freq, fds, fdc, fdf) :

       self.size = size

       self.count = count

       self.freq = freq

       self.fds = fds

       self.fdc = fdc

       self.fdf = fdf

       self.fitness = fitness(fds, fdc, fdf)


list_gen = []

combined_array = np.concatenate((data_array,

scaler.transform(df)), axis=1)
```

```
num_rows = combined_array.shape[0]

for i in combined_array :

   data_gen = Gen(i[0],i[1],i[2],i[3],i[4],i[5])

   list_gen.append(data_gen)
```

5. Mendefinisikan variabel awal

```
max_size = 950

populasi = []

jml_individu = 100

jml_gen = num_rows

p = 0.5

jumlah_generasi = 100

sumbuy=[]

sumbuy2 = []
```

6. Membuat Kelas Individu untuk menyimpan informasi dalam setiap individu, seperti total size, total count, total fre, nilai fitnessnya, yang datanya dihubungkan dengan array listgen yang telah dibuat berisikan informasi setiap Gen. Setiap Gen memiliki informasi 1 baris data dari dataset.

```
#Membuat class Individu untuk dapat mengenali bit dan fitness

class Individu :

   def __init__(self, bit, id) :

       self.id = id

       self.bit = bit

       self.jadiparent = 0

       self.probabilitas = 0

       self.probkumulatif = 0

       self.fitness = 0

       self.totalsize = 0

       self.totalcount = 0

       self.totalfreq = 0
```

```
        j = 0
        for i in bit :
            if i == 1 :
                size = list_gen[j].size
                count = list_gen[j].count
                freq = list_gen[j].freq
                fitness = list_gen[j].fitness
                self.fitness += fitness
                self.totalsize += size
                self.totalcount += count
                self.totalfreq += freq
                j+=1
            if i == 0 :
                j+=1
                continue
```

7. Melakukan pembuatan populasi awal. Yaitu dengan melakukan pembuatan kromosom sebanyak yang diinginkan, dengan setiap kromosom melakukan pembuatan gen sebanyak panjang baris dataset

```
iloop = 1
while iloop <= jml_individu :
   calon = []

   #Melakukan perulangan untuk membuat 1 per satu gen
   for j in range(jml_gen):
       #melakukan pemilihan pengacakan antara 0 sampai 1 dengan
digit maksimal 4
       r = round(random.uniform(0, 1), 4)
       if r<p :
           r = 0 #jika nilai random kurang dari 0,5 maka jadi 0
       else :
```

```
            r = 1 #jika nilai random lebih dari 0,5 maka jadi 1


        #Menyisipkan gen (nilai r) ke calon individu
        calon.append(r)



    #Memasukan setiap individu ke kelas individu
    individu = Individu(calon, iloop)
    #Menyisipkan individu ke populasi
    populasi.append(individu)
    iloop+=1
```

8. Selanjutnya membuat probabilitas kumulatif sebagai bahan tahap seleksi

```
#membuat persentase probabilitas
sum = 0
for i in populasi    :
    sum += i.fitness
    sumbuy.append(i.fitness)




avg = sum/jml_individu
sumbuy2.append(avg)


urut = 1
probkumulatif = 0
for i in populasi    :
    i.probabilitas = i.fitness /sum
    probkumulatif+=i.probabilitas
```

```
    i.probkumulatif = probkumulatif
    urut +=1
```

9. Selanjutnya Melakukan Seleksi Parent, Jumlah pasangan parent yang dipilih adalah ½ populasi, sehingga semua menjadi parent.

```
igen = 1
while igen <= jumlah_generasi:
    # Tahap Seleksi menggunakan Probabilitas Kumulatif
    #random angka dari 0 - 1
    semua_parent = []
    for _ in range(int(jml_individu/2)):
        parents_pair = []
        Parent1 =[]
        idp1 = 0
        dapet = 0
        while dapet  == 0:
            s = round(random.uniform(0,1), 4)
            for l in populasi :
                if l.probkumulatif < s or l.jadiparent == 1:
                    continue
                elif l.probkumulatif >= s:
                    l.jadiparent = 1
                    Parent1 = l.bit.copy()
                    idp1 = l.id
                    dapet =1
                    break
        Parent2 = []
        idp2 = 0
        dapet = 0
        while dapet  == 0:
```

```
            r = round(random.uniform(0,1), 4)
            for j in populasi :
                if j.probkumulatif < r or j.jadiparent == 1 :
                    continue
                elif j.probkumulatif >= r:

                    j.jadiparent = 1
                    Parent2 = j.bit.copy()
                    idp2 = j.id
                    dapet =1
                    break
        parents_pair.append(Parent1)
        parents_pair.append(Parent2)
        semua_parent.append(parents_pair)
```

10. Selanjutnya adalah crossover dan mutasi. Seperti pada algoritma genetika umumnya, crossover adalah untuk membuat child baru dari pasangan parent yang ada, mutasi adalah untuk menggubah nilai dari kromosom tersebut.

```
#Crossover
    #uniform crossover1

    # Membuat list masking dengan 10 elemen
    masking_co1 = []

    # Mengisi list masking dengan nilai acak antara 0 dan 1
    for _ in range(jml_gen):
        nilai_acak = random.randint(0, 1)
        masking_co1.append(nilai_acak)
    childtotal = []
    for pairs in semua_parent:
```

```
        ParentOne = []
        ParentTwo = []

        ParentOne = pairs[0]

        ParentTwo = pairs[1]
        child1 = []
        child2 = []
        i = 0
        while i < len(masking_co1) and i < len(ParentOne) and i <
len(ParentTwo):
            if masking_co1[i]==1 :
                child1.append(ParentTwo[i])
                child2.append(ParentOne[i])
            elif masking_co1[i]==0 :
                child1.append(ParentOne[i])
                child2.append(ParentTwo[i])
            i+=1

        #jadikan self.jadiparent = 0 lagi

        for j in populasi :
                j.jadiparent = 0

        #Mutasi
        #Probabilitas Mutasi
        prob_mutasi = 1/jml_gen

        #Mutasi Child 1
        child1termutasi = []
        randommutasi = random.uniform(0,1)
        for k in child1 :
```

```
        if randommutasi<prob_mutasi :
            if k==0:
                child1termutasi.append(1)
            elif k==1:
                child1termutasi.append(0)
        else :
            child1termutasi.append(k)
    #Mutasi Child 2
    child2termutasi = []
    randommutasi = random.uniform(0,1)
    for k in child1 :
        if randommutasi<prob_mutasi :
            if k==0:
                child2termutasi.append(1)
            elif k==1:
                child2termutasi.append(0)
        else :
            child2termutasi.append(k)


    #MUTASI KEDUA
    #menggunakan swap mutation
    #CHILD 1
    index1=0


    index2 =0
    while 1 :


        index1 = random.randint(0,jml_gen-1)
        index2 = random.randint(0,jml_gen-1)
        if index1!=index2:
            break
```

```
        child1termutasi[index1],  child1termutasi[index2] =
child1termutasi[index2],  child1termutasi[index1]


        #CHILD 2
        index1=0


        index2 =0
        while 1 :


            index1 = random.randint(0,jml_gen-1)
            index2 = random.randint(0,jml_gen-1)
            if index1!=index2:
                break


        child2termutasi[index1],  child2termutasi[index2] =
child2termutasi[index2],  child2termutasi[index1]


        childtotal.append(child1termutasi)
        childtotal.append(child2termutasi)
```

11. Tahap selanjutnya adalah untuk menentukan setiap nilai dari kromosom tersebut, jika total size melebihi maksimal size yaitu 950 maka child tidak digunakan dan lanjut ke pengecekan berikutnya. Tujuan dari tahapan ini adalah subtitusi nilai child jika lebih baik dari populasi awal.

```
#mencari dan menggantikan nilai fitness terendah


   for c in childtotal:
        fitnesschild = 0
        totalsize = 0
        totalcount = 0
        totalfreq = 0
        minimfitness=0
```

```
        minimbit=[]
        minimid=0
        oversize = 0
        overbit = []
        overid =0
        j = 0
        for i in c :
            if i == 1 :
                size = list_gen[j].size
                count = list_gen[j].count
                freq = list_gen[j].freq
                fitnessc = list_gen[j].fitness
                fitnesschild += fitnessc
                totalsize += size
                totalcount += count
                totalfreq += freq
                j+=1
            if i == 0 :
                j+=1
                continue
        if totalsize>max_size :
            continue
```

12. Tahap selanjutnya adalah mencari nilai total size dari populasi yang lebih dari max size yang akan digantikan oleh child baru. Jika ternyata nilainya sudah dibawah maxsize maka lanjut ke tahap berikutnya.

```
#Tahap pertama, membatasi supaya maksimal totalsize = 950
        penanda_lanjut_tahap_2 = 1
        for p in populasi:
            if p.totalsize > max_size :
                oversize = p.totalsize
                overbit = p.bit
```

```
                overid = p.id
                penanda_lanjut_tahap_2 = 0



        #Perhitungan dan Penggantian dari Child 1
        if penanda_lanjut_tahap_2 ==0 :
            for q in populasi :
                if q.id == overid:
                    q.bit =  c
                    q.fitness = fitnesschild
                    q.totalsize = totalsize
                    q.totalcount = totalcount
                    q.totalfreq = totalfreq


                    break
                else:
                    continue
```

13. Jika sudah saatnya lanjut tahap 2, maka dilakukanlah subtitusi didasarkan pada nilai fitness, jika nilai fitness child lebih besar dari nilai fitness populasi yang sedang di looping maka akan digantikan.

```
        if penanda_lanjut_tahap_2 == 1 :
            tanda_disubs = 0
        #Tahap Kedua, memperbaiki nilai fitness
            for p in populasi:


                if  p.fitness <  avg and p.fitness <  fitnesschild
:
                    minimfitness = p.fitness
                    minimbit = p.bit
                    minimid = p.id
```

```
                tanda_disubs = 1



        if tanda_disubs == 1 :
        #Perhitungan dan Penggantian dari Child 1
            for q in populasi :
                if q.id == minimid:
                    q.bit =  c
                    q.fitness = fitnesschild
                    q.totalsize = totalsize
                    q.totalcount = totalcount
                    q.totalfreq = totalfreq


                    break
                else:
                    continue
```

14. Selanjutnya adalah menghitung dan memperbaiki ulang nilai probabilitas kumulatif

```
sum = 0
for i in populasi   :
    sum += i.fitness


    sumbuy.append(i.fitness)


urut = 1
probkumulatif = 0
for i in populasi   :
    i.probabilitas = i.fitness /sum
    probkumulatif+=i.probabilitas
```

```
        i.probkumulatif = probkumulatif
        urut +=1
```

15. Selanjutnya adalah perhitungan rata rata fitness pada tiap generasi.

```
#Perhitungan rata rata nilai fitness generasi ke generasi


    avg = sum/jml_individu
    sumbuy2.append(avg)
    print(f"\nFitness average Generasi ke - {igen} = {avg}")
    igen+=1
urut = 1
```

16. Terakhir adalah membuat tulisan di terminal Kromosom seperti apa yang terpilih dan hasilnya item mana saja yang terpilih dan menghasilkan nilai fitness berapa. Serta memunculkan grafik dari generasi ke generasi

```
print(f"\nPopulasi Hasil Iterasi Terakhir :")
for manusia in populasi    :


    print(f"Bit Individu ke - {urut} = {manusia.bit}")
    print(f"Fitness Individu ke - {urut} = {manusia.fitness}")
    print(f"Size Individu ke - {urut} = {manusia.totalsize}")
    print(f"Probabilitas Individu ke - {urut} =
{manusia.probabilitas}")
    print(f"Prob Kumulatif Individu ke - {urut} =
{manusia.probkumulatif}")
    urut +=1
maxfit = 0
for p in populasi :
    if p.fitness > maxfit :
        maxfit = p.fitness
        maxbit = p.bit
optimal = []
```

```
index = 1
for i in maxbit :
   if i == 1 :
       optimal.append(index)
   index +=1
print(f"\n\n Baris Data Terpilih : {optimal}")
i=0
sumbux=[]
while i <= jumlah_generasi:
   j = 0
   while j < jml_individu:
       sumbux.append(i)
       j+=1
   i+=1
xpoints = sumbux
ypoints = sumbuy
i=0
sumbux2=[]
while i <= jumlah_generasi:
   sumbux2.append(i)
   i+=1
plt.plot(xpoints, ypoints, 'o', markersize=2, label='Semua
Fitness')
plt.plot(sumbux2, sumbuy2, 'o', markersize=3,color="red",
linestyle="-", label='Rerata Fitness')
plt.xlabel('Generasi')
plt.ylabel('Fitness')
plt.title('Perkembangan Fitness pada Setiap Generasi')
plt.legend()
plt.show()
```

Hasil dari perhitungan ini jika saya menggunakan jumlah individu 100 dengan perulangan 100 generasi serta dengan bobot yang ditentukan maka nilai fitness dan grafiknya serta item yang terpilih yaitu sebagai berikut.

Bit Individu ke - 100 = [1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0]
Fitness Individu ke - 100 = 6.770828344786489
Size Individu ke - 100 = 931.0
Probabilitas Individu ke - 100 = 0.00999999999999999
Prob Kumulatif Individu ke - 100 = 1.0000000000000002
Baris Data Terpilih : [1, 2, 3, 8, 9, 10, 11, 12, 14, 15, 17, 18, 19]

```
Bit Individu ke - 100 = [1, 1, 1, 0, 0, 0
, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1,
0]
Fitness Individu ke - 100 = 6.77082834478
6489
Size Individu ke - 100 = 931.0
Probabilitas Individu ke - 100 = 0.009999
99999999999
Prob Kumulatif Individu ke - 100 = 1.0000
000000000002


 Baris Data Terpilih : [1, 2, 3, 8, 9, 10
, 11, 12, 14, 15, 17, 18, 19]
```