گزارش پروژه تمرین سری دوم بخش اول

کد نوشته شده سه بخش دارد که هر بخش را به طور کامل شرح خواهیم داد.

بخش اول: کلاس گراف:

- زنجیره ارتباطاتی که در txt به ما میدهند با کمک کلاس گراف به صورت یک لیست مجاورت نگهداری می‌کنیم
- در این بخش یک کلاس طراحی کردیم که بتوانیم اطلاعاتمون رو به صورت گراف دخیره کنیم. برای این ذخیره سازی از لیست مجاورت استفاده کردیم.
- این کلاس مجموعه راس‌های گراف را در V ذخیره میکند و خود گراف را درgraph نگهداری می‌کند که یک دیکشنری از لیست است.

```
class Graph:
    def __init__(self,v):
        self.v = v
        self.graph = defaultdict(list)
```

- متد addEdge() برای این است که به مجموعه رئوسمون یال اضافه کنیم. متد printGraph() برای این است که تابع گرافمان را چاپ کنیم. متد isReachable(a,b) برای این است چک می‌کند که ایا از a به b مسیر است یا نه؟ (آیا a شکارچی b است). این متد با کمک BFS پیامش میکند و مسیر را پیدا می‌کند. متدهای DFS و DFSUtil نیز متدهای مربوط به DFS گراف است که البته در الگوریتم استفاده نشده است.

```
class Graph:
    def __init__(self,v):
        self.v = v
        self.graph = defaultdict(list)

    def addEdge(self, src, dest):
        self.graph[src].append(dest)

    def printGraph(self):
        for i in range(self.v):
            print("src=",i)
            for j in self.graph[i]:
                print(j)

    def isReachable(self,src,dest):
        visited =[False]*(self.v+1)

        queue=[]
        queue.append(src)
```

```
        visited[src] = True
        while queue:
            n = queue.pop(0)
            if n == dest:
                    return True
            for i in self.graph[n]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True
        return False

    def DFSUtil(self, v, visited):
        visited.add(v)
        print(v, end=' ')

        for j in self.graph[v]:
            if j not in visited:
                self.DFSUtil(j, visited)

    def DFS(self, v):
        visited = set()
        self.DFSUtil(v, visited)
```

بخش دوم: خواندن ورودی از فایل و تبدیل به گراف

- در این بخش فایل ورودی را میخوانیم.
- File_address همان آدرس فایل متنی ما است
- Lines خطوط فایل را در آن‌ها قرار می‌دهیم. که خط اول فایل همان تعداد راس‌های گرافمان است. و سایر خطوط را نیز با addEdge به گراف اضافه میکنیم.

```
# GET INPUT FUNCTION and COVERT TO GRAPH
file_address = "C:/Users/Ashkan/Desktop/Term 8/ هوش/تمرین ها/تمرین 2 - بخش
اول/SA/input.txt"

lines = []
with open(file_address) as f:
    lines = f.readlines() # har khato mirize to y khune array

V = int(lines[0])
graph = Graph(V+1)
count = 0
for line in lines:
    if(count!=0):
```

```
        edge = line.split(' ')
        graph.addEdge(int(edge[0]),int(edge[1]))
    count += 1
```

بخش سوم: اجرای الگوریتم SA: که شامل تابع هدف، مقداردهی اولیه و تکرار الگوریتم

- تابع هدف مسئله به این صورت است که تعداد ترتیب‌های نادرست را می‌شماریم. روش این کار نیز به این صورت است که لیستی از ترتیب هر گره داریم و چک میکنیم ایا مسیر وجود دارد یا نه؟ اگر مسیر وجود داشت و ولی جاشون برعکس بود یک واحد به q اضافه میکند و هدف ما کاهش q است.
- البته توجه کنید با توجه به این که بدترین حالت عدد خیلی بزرگی مثل ۴۰۰۰ در مسئله ما تا ۲۰ کاهش میابد. (و لزوما صفر نمیرسد ولی بسیار به جواب نزدیک می‌وشود.)

```
# OBJECTIVE FUNCTION
def objective(sol):
    q = 0
    for i in range(n): # 3 4 12 5
        for j in range(i,n):
            if(graph.isReachable(sol[j],sol[i])):
                q = q + 1
    return q
```

- مقداردهی اول برای الگوریتم که n همان تعداد راس‌های گراف است. T که همان دما است و عدد بزرگ می‌دهیم و t_change را نیز برای ضریب کاهش استفاده کردیم.
- همچنین به صورت تصادفی مقدار اولیه را برای جواب انتخاب میکنیم. به این صورت که یک دنباله درست میکنیم و سپس جایگاهشان را تصادفی میکنیم (توجه کنید که این کار یک انتخاب بدون تکرار است). مقدار fitness را با کمک تابع هدف نیز برای مقدار اولیه محاسبه میکنیم.

```
# INITIAL DATA & PARAMETERS
n = V
T = 100000
t_change = 0.99

# INITIAL SOLUTION
sequence = [i+1 for i in range(n)]
solution = random.sample(sequence, n)
print(solution)
fitness = objective(solution)
print(fitness)
```

- اما حلقه اصلی برنامه به صورتی است که می‌آید آن جواب اولیه را می‌گیرد و همسایه‌های آن را می‌سازد. اما ساخت همسایه در این روش (SA) خیلی حساس است. دو مقدار تصادفی تولید میکنیم و چک میکنیم که ایا از اولی به دومی مسیر است و جایگاه اولی و دومی در لیست به صورتی است که اولی قبل دومی باشد؟ اگر اینطوری بود جاشون درست است و نیازی به جابجایی نداریم و دو عدد تصادفی جدید انتخاب میکنیم. و سپس جای دو عدد را عوض میکنیم.
- حال برای همسایه جدید مقدار fitness را حساب کرده و مانند الگوریتم SA برای آن عمل میکنیم. به این صورت که delta را حساب کرده و اگر دلتا مثبت بود که یعنی جواب ما بهتر شده و نگه میداریمش و اگه منفی بود با شروط و فرمول‌های SA و دما تعیین میکنیم که انتخاب کنیم یا نه.

```
while T > 0:
    neighbour = solution.copy()
    temp = np.random.randint(n)
    temp2 = np.random.randint(n)
    if((graph.isReachable(neighbour[temp],neighbour[temp2]) and temp<temp2)):
        continue


    neighbour[temp] , neighbour[temp2] = neighbour[temp2] , neighbour[temp]

    fit = objective(neighbour)

    delta = fitness - fit
    if delta >= 0:
        solution = neighbour
        fitness = fit
    else:
        pr = math.exp(delta / T)
        if pr >= .999:
            solution = neighbour
            fitness = fit

    #print(fitness)
    T = int(T * t_change)
```

دونمونه نمونه جواب (دو خط اول وضعیت اولیه است و دو خط دوم وضعیت نهایی)

```
PS C:\Users\Ashkan> & C:/Users/Ashkan/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Ashkan/Desktop/Term 8/???/????? ??/????? 2 - ??? ???/SA
/index.py"
[13, 17, 6, 5, 15, 12, 10, 19, 9, 8, 7, 14, 4, 3, 18, 1, 2, 11, 16, 20]
125
[4, 20, 7, 1, 3, 2, 10, 19, 18, 17, 16, 15, 6, 11, 12, 13, 8, 9, 14, 5]
21
PS C:\Users\Ashkan> & C:/Users/Ashkan/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Ashkan/Desktop/Term 8/???/????? ??/????? 2 - ??? ???/SA
/index.py"
[18, 12, 2, 5, 9, 13, 17, 4, 6, 20, 8, 1, 16, 14, 10, 15, 3, 11, 7, 19]
103
[20, 4, 3, 2, 1, 10, 19, 18, 17, 16, 15, 11, 6, 12, 7, 8, 13, 9, 14, 5]
20
```