In [1]:

```
import random
import operator
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%run TSP.ipynb
%matplotlib inline
```

In [2]:

```
class Ponto:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "({},{})".format(str(self.x), str(self.y))

    def distancia(self, ponto):
        xd = abs(self.x - ponto.x)
        yd = abs(self.y - ponto.y)
        return np.sqrt((xd ** 2) + (yd ** 2)).astype(int)
```

In [3]:

```
class Fitness:
    def __init__(self, rota):
        self.rota = rota
        self.fitness = 0
        self.distancia = 0

    def percorrerRota(self):
        if self.distancia == 0:
            distanciaPercorrida = 0
            for i in range(0, len(self.rota)):
                pontoA = self.rota[i]
                pontoB = None
                if i + 1 < len(self.rota):
                    pontoB = self.rota[i + 1]
                else:
                    pontoB = self.rota[0]
                distanciaPercorrida += pontoA.distancia(pontoB)
            self.distancia = distanciaPercorrida
        return self.distancia

    def getFitnessRota(self):
        if self.fitness == 0:
            self.fitness = 1 / float(self.percorrerRota())
        return self.fitness
```

In [4]:

```
# Retorna novo individuo
def criarRota(pontos):
    rota = random.sample(pontos, len(pontos))
    return rota
```

In [5]:

```
# Retorna lista de individuos
def inicializaPopulacao(tamanhoPopulacao, pontos):
    populacao = []
    for i in range(0, tamanhoPopulacao):
        populacao.append(criarRota(pontos))
    return populacao
```

In [6]:

```
# Retorna Dicionário ordenado por fitness
def rankInidividuos(populacao):
    fitness = {}
    for i in range(0,len(populacao)):
        fitness[i] = Fitness(populacao[i]).getFitnessRota()
    return sorted(fitness.items(), key = operator.itemgetter(1), reverse = True)
```

In [7]:

```
# Seleção Proporcional(Roleta)
# Retorna lista de indices dos individuos selecionados
def selecao(rankPopulacao, numEleitos):
    selecionados = []

    # Cria DataFrame
    df = pd.DataFrame(np.array(rankPopulacao), columns=["Indice","Fitness"])

    # Realiza Soma Acumulativa
    df['Soma_Acumulativa'] = df.Fitness.cumsum()

    # Calcula Percentual de aptidão
    df['Percentual'] = 100 * df.Soma_Acumulativa / df.Fitness.sum()

    # Seleciona eleitos
    for i in range(0, numEleitos):
        selecionados.append(rankPopulacao[i][0])

    # Seleção Proporcional
    for i in range(0, len(rankPopulacao) - numEleitos):

        # Roda roleta
        pontera = 100 * random.random()

        # Seleciona primeiro individuo apontado
        for i in range(0, len(rankPopulacao)):
            if pontera <= df.iat[i,3]:
                selecionados.append(rankPopulacao[i][0])
                break

    return selecionados
```

In [8]:

```
# Retorna lista de individuos
def getProcriadores(populacao, selecionados):
    procriadores = []
    for i in range(0, len(selecionados)):
        indice = selecionados[i]
        procriadores.append(populacao[indice])
    return procriadores
```

In [9]:

```
# +/- order crossover operator (OX1 ou OX)
def procriar(pai, mae):
    filho_DNA = []
    p1_DNA = []
    p2_DNA = []

    # Determina genes do pai que serão passados para o filho
    geneA = int(random.random() * len(pai))
    geneB = int(random.random() * len(pai))

    inicioGene = min(geneA, geneB)
    fimGene = max(geneA, geneB)

    # Adicio pedaço do DNA do Pai para o filho
    for i in range(inicioGene, fimGene):
        p1_DNA.append(pai[i])

    # Completa restantando do DNA com o da Mãe
    p2_DNA = [gene for gene in mae if gene not in p1_DNA]

    filho_DNA = p1_DNA + p2_DNA
    return filho_DNA
```

In [10]:

```
def procriaPopulacao(procriadores, numEleitos, taxaCruzamento):
    filhos = []
    naoEleitos = len(procriadores) - numEleitos
    numProcriadores = int(naoEleitos * taxaCruzamento)
    naoProcriadores = naoEleitos - numProcriadores
    pais = random.sample(procriadores, numProcriadores)

    # Passa eleitos para próxima geração
    for i in range(0, numEleitos):
        filhos.append(procriadores[i])

    # Procria nova geração conforme taxa de cruzamento
    for i in range(0, numProcriadores):
        filho = procriar(pais[i], pais[numProcriadores-i-1])
        filhos.append(filho)

    # Preenche restante da população procriadores
    naoPais = random.sample(procriadores, naoProcriadores)
    for i in range(0, naoProcriadores):
        filhos.append(naoPais[i])

    return filhos
```

In [11]:

```
# Mutação por troca
def mutar(dna, taxaMutacao):
    for i in range(len(dna)):
        if(random.random() < taxaMutacao):
            j = int(random.random() * len(dna))

            geneA = dna[i]
            geneB = dna[j]

            dna[i] = geneB
            dna[j] = geneA
    return dna
```

In [12]:

```
# Retorna população mutada
def mutarPopulacao(populacao, taxaMutacao):
    populacaoMutada = []

    for ind in range(0, len(populacao)):
        individuoMutado = mutar(populacao[ind], taxaMutacao)
        populacaoMutada.append(individuoMutado)
    return populacaoMutada
```

In [13]:

```
def proximaGeracao(populacao, numEleitos, taxaCruzamento, taxaMutacao):
    # Ordena população conforme fitness
    rankPopulacao = rankInidividuos(populacao)

    # Realiza seleção dos melhor individuos
    selecionados = selecao(rankPopulacao, numEleitos)

    # Recupera individuos conforme indices selecionados
    procriadores = getProcriadores(populacao, selecionados)

    # Cria filhos a partir dos procriadores
    filhos = procriaPopulacao(procriadores, numEleitos, taxaCruzamento)

    # Realiza mutação nos filhos
    proximaGeracao = mutarPopulacao(filhos, taxaMutacao)
    return proximaGeracao
```

In [14]:

```
def geneticAlgorithm(pontos, tamanhoPopulacao, numEleitos, taxaCruzamento, taxaMutacao,
geracoes):
    populacao = inicializaPopulacao(tamanhoPopulacao, pontos)
    progresso = []

    print("Distancia Inicial: " + str(1 / rankInidividuos(populacao)[0][1]))

    melhorFitness = 999999
    geracoesSemMelhora = 0
    i = 0

    while geracoesSemMelhora < geracoes:

        fitnessAtual = 1 / rankInidividuos(populacao)[0][1]

        if fitnessAtual < melhorFitness:
            melhorFitness = fitnessAtual
            geracoesSemMelhora = 0
        else:
            geracoesSemMelhora += 1


        progresso.append({
            "geracao": i,
            "fitness": fitnessAtual,
            "solucao": populacao[rankInidividuos(populacao)[0][0]]
        })
        populacao = proximaGeracao(populacao, numEleitos, taxaCruzamento, taxaMutacao)
        i += 1

    print("Geração Final:" + str(i))
    print("Distancia Final: " + str(1 / rankInidividuos(populacao)[0][1]))

    indiceMelhorRota = rankInidividuos(populacao)[0][0]
    melhorRota = populacao[indiceMelhorRota]
    return {"melhorRota": melhorRota, "progresso": progresso}
```

In [15]:

```
def geneticAlgorithmPlot(pontos, tamanhoPopulacao, numEleitos, taxaCruzamento, taxaMuta
cao, geracoes):
    populacao = inicializaPopulacao(tamanhoPopulacao, pontos)
    progresso = []
    progresso.append(1 / rankInidividuos(populacao)[0][1])

    print("Distancia Inicial: " + str(1 / rankInidividuos(populacao)[0][1]))

    melhorFitness = 999999
    geracoesSemMelhora = 0
    i = 0
    while geracoesSemMelhora < geracoes:

        fitnessAtual = 1 / rankInidividuos(populacao)[0][1]

        if fitnessAtual < melhorFitness:
            melhorFitness = fitnessAtual
            geracoesSemMelhora = 0
        else:
            geracoesSemMelhora += 1


        populacao = proximaGeracao(populacao, numEleitos, taxaCruzamento, taxaMutacao)
        progresso.append(1 / rankInidividuos(populacao)[0][1])
        i += 1

    print("Distancia Final: " + str(1 / rankInidividuos(populacao)[0][1]))

    indiceMelhorRota = rankInidividuos(populacao)[0][0]
    melhorRota = populacao[indiceMelhorRota]

    plt.plot(progresso)
    plt.ylabel('Distancia')
    plt.xlabel('Geração')
    plt.show()
    plt.close()

    melhorRota.append(melhorRota[0])

    x = []
    y = []

    for p in melhorRota:
        x.append(p.x)
        y.append(p.y)

    plt.title("Melhor Rota !   Fitness: {}".format(1 / rankInidividuos(populacao)[0][1
]))
    plt.plot(x, y)
    plt.plot(x, y, 'ro')
    plt.show()
```