Instituto de Computação - Unicamp

MC102 - Algoritmos e Programação de Computadores

Laboratório 17 - Banco de Dados Geográfico

Prazo de entrega: **15/06/2019 23:59:59**

Peso: **2**

Professor: Eduardo Candido Xavier
Professor: Luiz Fernando Bittencourt

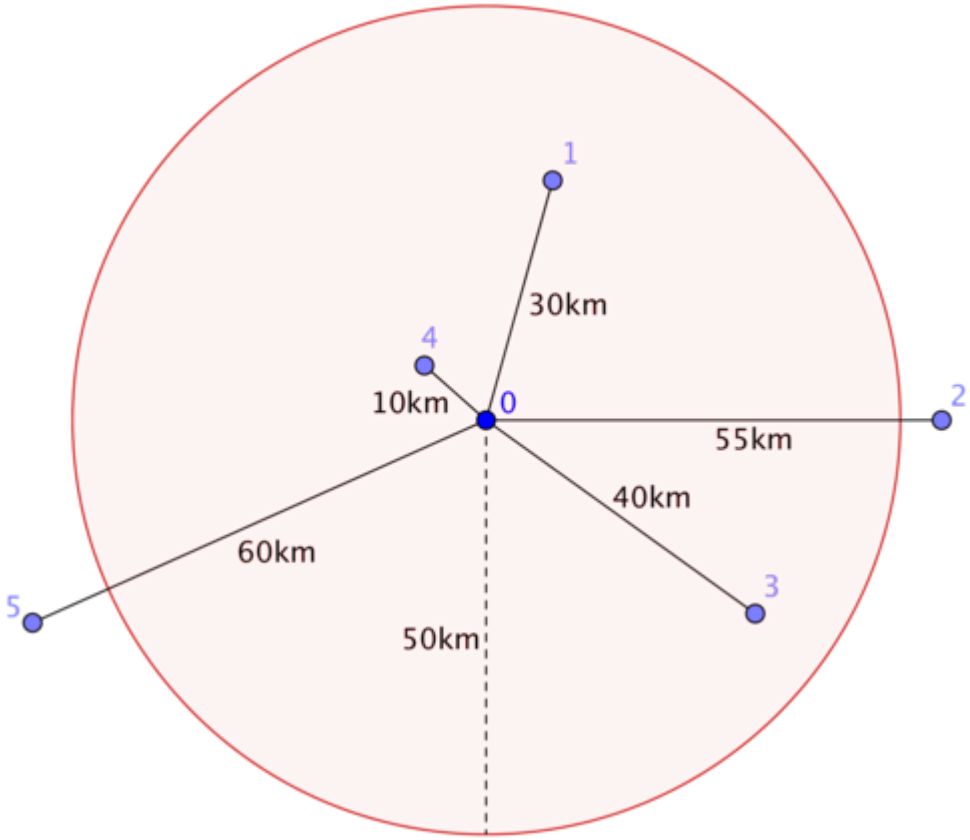
Descrição

Você já deve ter usado serviços como o Google Maps. Estes serviços possuem extensos bancos de dados com as mais diversas informações geográficas. Neste laboratório você irá implementar um versão simplificada de um banco de dados geográfico que conterá informações sobre cidades. Nesta simplificação, iremos considerar uma versão planificada da Terra, onde qualquer ponto pode ser representando por uma par de coordenadas `(x, y)` no plano.

No nosso banco de dados, para cada cidade iremos armazenar as seguintes informações:

- Coordenadas `(x, y)` de um ponto representando a localização da cidade;
- Faixa de CEPs atribuídas àquela cidade, e.g. 12300-000 - 12300-130 (o que inclui os CEPs 12300-001, 12300-002, ..., 12300-129);
- Número de habitantes (em milhares de pessoas).

Com as coordenadas definidas acima, podemos realizar consultas baseadas na distância entre cidades como no exemplo abaixo. Neste exemplo, gostaríamos de encontrar todas as cidades que estão num raio de até 50km da cidade 0, neste caso a própria cidade 0 e as cidades 1, 3 e 4.



O objetivo deste laboratório é criar uma biblioteca de funções em Python capaz de realizar consultas no banco de dados geográfico definido acima.

O registro `Ponto` representa uma coordenada no plano. Assumimos que os valores são reais e as medidas estão em Km. O registro `Cidade` representa os dados de uma cidade incluindo sua localização (`coordenadas`), a faixa de CEPs que ocorrem na cidade (de `inicioCep` até `fimCep`) e finalmente o número de habitantes na cidade (`numHabitantes`).

Em Python usaremos as seguintes classes para representar as informações de uma cidade:

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

class Cidade:
    def __init__(self, coordenadas, inicioCEP, fimCEP, numHabitantes):
        self.coordenadas = coordenadas
        self.inicioCEP = inicioCEP
        self.fimCEP = fimCEP
        self.numHabitantes = numHabitantes
```

A classe `Ponto` representa uma coordenada no plano. Assumimos que os valores são reais e as medidas estão em Km. A classe `Cidade` representa os dados de uma cidade incluindo sua localização (`coordenadas`), a faixa de CEPs que ocorrem na cidade (de `inicioCep` até `fimCep`) e finalmente o número de habitantes na cidade (`numHabitantes`).

Você deve implementar funções que realizam as seguintes operações:

- **Consulta cidade por CEP:** dado um CEP, deve-se procurar no banco de dados a cidade que contém este CEP.
- **Consulta cidades por raio:** dada uma cidade de referência e uma distância máxima, deve-se buscar todas as cidades que estão dentro do raio desejado assumindo-se a cidade de referência como centro.
- **Total de habitantes:** dada uma cidade de referência e uma distância máxima de raio, deve-se calcular qual é o total de habitantes de todas as cidades que estão dentro do raio definido, considerando-se a cidade de referência como centro.
- **Mediana da população:** dada uma cidade de referência e uma distância máxima, deve-se calcular qual é a mediana do número de habitantes das cidades dentro deste raio. Iremos considerar que a mediana de `n` elementos (ordenados) será o elemento intermediário se `n` for ímpar, e a média dos elementos intermediários caso contrário.

Obs.: Note que a função para calcular a distância euclidiana entre dois pontos já está implementada. Você *não* deve modificar esta função.

A descrição geral dos parâmetros de entrada e saída das funções estão descritos nos comentários dos protótipos das funções, que são fornecidos a seguir:

Linguagem Python 3:

```
#!/usr/bin/env python3

import math

# Laboratorio 17 - Banco de Dados Geografico
# Nome:
# RA:

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

```
        self.x = x
        self.y = y

class Cidade:
    def __init__(self, coordenadas, inicioCEP, fimCEP, numHabitantes):
        self.coordenadas = coordenadas
        self.inicioCEP = inicioCEP
        self.fimCEP = fimCEP
        self.numHabitantes = numHabitantes

#
# Funcao: distancia
#
# Parametros:
#   a, b: pontos
#
# Retorno:
#   A distancia euclidiana entre a e b.
#
def distancia(c1, c2):
    return int(100 * math.sqrt((c1.x - c2.x)**2 + (c1.y - c2.y)**2)) / 100.0

# Funcao: consulta_cidade_por_cep
#
# Parametros:
#   cidades: lista de cidades (base de dados)
#       cep: CEP desejado
#
# Retorno:
#   O indice da cidade que contem o CEP desejado ou None caso não haja tal c
#
def consulta_cidade_por_cep(cidades, cep):
    # Implementar a funcao e trocar o valor de retorno
    return None

# Funcao: consulta_cidades_por_raio
#
# Parametros:
#           cidades: lista de cidades (base de dados)
#   cidadeReferencia: indice da cidade referencia (centro da circunferencia)
#              raio: raio da busca
#
# Retorno:
#   Lista dos indices das cidades que estao contidas no raio de busca (inclu
#   a cidade referencia) *ordenados pelas respectivas distancias da cidade r
#
def consulta_cidades_por_raio(cidades, cidadeReferencia, raio):
    # Implementar a funcao e trocar o valor de retorno
    return None

# Funcao: populacao_total
```

```
#
# Parametros:
#            cidades: lista de cidades (base de dados)
#   cidadeReferencia: indice da cidade referencia (centro da circunferencia)
#               raio: raio da busca
#
# Retorno:
#   O numero de habitantes das cidades que estao contidas no raio de busca
#
def populacao_total(cidades, cidadeReferencia, raio):
    # Implementar a funcao e trocar o valor de retorno
    return None

# Funcao: mediana_da_populacao
#
# Parametros:
#            cidades: lista de cidades (base de dados)
#   cidadeReferencia: indice da cidade referencia (centro da circunferencia)
#               raio: raio da busca
#
# Retorno:
#   Mediana do numero de habitantes das cidades que estao contidas no raio d
#
def mediana_da_populacao(cidades, cidadeReferencia, raio):
    # Implementar a funcao e trocar o valor de retorno
    return None
```

Submissão

Neste laboratório você não precisará se preocupar em ler a entrada a partir da entrada padrão, nem em escrever a saída. Seu trabalho é apenas implementar as funções descritas. A função `main()` que é fornecida no arquivo `lab17_main.py` se encarrega dessa parte.

Você também **não deve** submeter o arquivo `lab17_main.py` para o *SuSy*, somente o arquivo `lab17.py`.

As seções abaixo, de Entrada e Saída, descrevem os formatos de entrada e saída, mas você não precisa se preocupar com eles.

A primeira linha da entrada contém um inteiro `n`, o número de cidades no banco de dados. A seguir temos `n` linhas descrevendo cada uma das cidades. Cada linha contém 5 números: `x`, `y`, `inicio CEP`, `fim CEP` e `numero de habitantes`. `x` e `y`

descrevem as coordenadas da cidade e `inicio CEP` e `fim CEP` indicam a faixa de CEPs atribuída à cidade. Após as `n` linhas, a entrada contém um inteiro `q`, o número de consultas que serão realizadas. As `q` linhas seguintes contém a descrição de cada consulta. O primeiro parâmetro é sempre o tipo de consulta que deverá ser executada, e segue a tabela abaixo:

- `0` : busca cidade por CEP
- `1` : busca cidades por raio
- `2` : população total
- `3` : mediana da população

Os parâmetros seguintes dependem da consulta a ser realizada. Para a consulta `0`, há apenas um parâmetro adicional, o CEP desejado. Para as demais consultas teremos dois parâmetros adicionais, a cidade referência e o raio da busca.

A saída conterá o resultado de cada uma das consultas requisitadas na entrada. A primeira linha da saída de cada consulta contém `Consulta X`, onde `X` é o número da consulta, começando em 0. A seguir temos a saída específica para cada consulta:

- `0` : `Cidade com CEP Y: Z` ou `Nao ha uma cidade com o CEP Y`, onde `Y` é o CEP recebido na entrada, e `Z` a cidade que contém tal CEP;
- `1` : primeiramente contém `Cidades no raio de R:`, onde `R` é o raio recebido na entrada. A seguir é listada, uma por linha, cada cidade contida no raio desejado, ordenadas por sua distância da cidade referência.
- `2` : `Populacao total: A`, onde `A` corresponde à população total calculada
- `3` : `Mediana da populacao: M`, onde `M` corresponde à mediana calculada

Exemplos

Teste 01

Entrada

```
36
456.85 716.02 10000001 10000119 880
5.05 407.26 10000120 10000196 319
939.89 846.79 10000197 10000511 736
307.75 523.13 10000512 10000574 430
801.17 566.67 10000575 10001398 996
```

```
944.24 760.03 10001399 10001441 409
145.26 41.35 10001442 10001542 606
972.14 574.72 10001543 10001839 559
508.96 833.83 10001840 10001924 960
629.1 664.73 10001925 10002645 878
386.13 672.02 10002646 10002986 58
214.21 665.16 10002987 10003119 455
552.73 748.62 10003120 10003400 701
469.68 439.76 10003401 10003465 812
798.8 905.24 10003466 10003642 120
903.26 707.57 10003643 10003838 93
190.94 357.68 10003839 10003988 824
147.93 6.54 10003989 10004085 501
162.76 795.82 10004086 10004591 376
570.51 474.65 10004592 10004705 220
171.25 399.71 10004706 10004868 932
544.2 417.19 10004869 10005218 106
649.77 634.01 10005219 10005230 629
842.02 261.47 10005231 10005615 626
673.16 37.05 10005616 10005741 926
412.65 861.23 10005742 10006335 333
883.28 998.47 10006336 10006356 536
353.88 202.23 10006357 10006814 916
793.37 820.77 10006815 10006874 461
448.34 707.08 10006875 10007532 538
674.19 274.34 10007533 10007774 150
360.29 273.78 10007775 10008501 841
137.23 604.46 10008502 10008637 809
968.96 219.83 10008638 10008722 291
844.91 774.43 10008723 10008763 768
592.0 225.64 10008764 10008937 345
4
3 7 734
2 15 488
3 33 863
2 24 459
```

Saída

```
Consulta 0:
Mediana da populacao: 548.5
Consulta 1:
Populacao total: 10216
Consulta 2:
Mediana da populacao: 582.5
Consulta 3:
Populacao total: 5233
```

Teste 08

Entrada

```
39
389.48 850.95 10000001 10000149 836
911.37 298.92 10000150 10000153 956
742.7 361.76 10000154 10000163 437
811.26 844.03 10000164 10000235 327
601.54 325.13 10000236 10000691 624
723.74 774.83 10000692 10000786 136
492.59 948.28 10000787 10001133 385
890.9 691.84 10001134 10001236 663
132.21 803.11 10001237 10001784 176
776.17 289.26 10001785 10001881 51
242.63 203.47 10001882 10002844 906
302.36 657.42 10002845 10003006 146
576.53 799.97 10003007 10003191 732
736.69 721.09 10003192 10003254 486
821.51 950.67 10003255 10003798 781
628.98 80.08 10003799 10003846 332
115.79 802.72 10003847 10003878 985
355.3 205.12 10003879 10004250 774
222.72 646.47 10004251 10004252 188
612.51 669.31 10004253 10004346 148
229.86 73.4 10004347 10004790 358
391.59 218.62 10004791 10005276 914
239.15 883.32 10005277 10005497 90
12.28 428.37 10005498 10005834 795
450.41 627.59 10005835 10005848 229
972.08 366.68 10005849 10006541 438
75.17 101.87 10006542 10006994 332
799.18 949.53 10006995 10007471 916
466.85 345.64 10007472 10008430 447
764.98 715.37 10008431 10009131 554
682.42 806.12 10009132 10009501 694
263.99 49.94 10009502 10010166 293
402.81 721.02 10010167 10010366 531
374.32 260.67 10010367 10010505 902
570.7 811.98 10010506 10010621 603
751.21 67.63 10010622 10010782 591
201.23 855.36 10010783 10011382 481
636.33 159.12 10011383 10011683 83
914.34 480.8 10011684 10012046 445
14
3 27 985
0 10002431
0 10010385
0 10008303
```

```
3 20 854
3 38 737
2 17 746
2 30 889
0 10013333
0 10002247
3 38 951
3 14 855
1 30 224
2 13 737
```

Saída

```
Consulta 0:
Mediana da populacao: 508.5
Consulta 1:
Cidade com CEP 10002431: 10
Consulta 2:
Cidade com CEP 10010385: 33
Consulta 3:
Cidade com CEP 10008303: 28
Consulta 4:
Mediana da populacao: 464.0
Consulta 5:
Mediana da populacao: 531.0
Consulta 6:
Populacao total: 17356
Consulta 7:
Populacao total: 19433
Consulta 8:
Nao ha uma cidade com o CEP 10013333
Consulta 9:
Cidade com CEP 10002247: 10
Consulta 10:
Mediana da populacao: 481.0
Consulta 11:
Mediana da populacao: 481.0
Consulta 12:
Cidades no raio de 224:
30
5
13
12
34
29
3
19
27
14
```

```
Consulta 13:
Populacao total: 17987
```

Para mais exemplos, consulte os testes abertos no Susy.

Observações

- Você **não deve** submeter o arquivo `lab17_main.py` para o *SuSy*, somente o arquivo `lab17.py` .
- O número máximo de submissões é **10**.
- Para a realização dos testes do SuSy, a execução do código em Python se dará da seguinte forma: (Linux e OSX) `python3 lab17_main.py` .
- Você deve incluir, no início do seu programa, uma breve descrição dos objetivos do programa, da entrada e da saída, além do seu nome e do seu RA.
- Indente corretamente o seu código e inclua comentários no decorrer do seu programa.