Bernardo Morais Alfredo – 565524

Sumário

2.1 PROBLEMA 2 – SISTEMA DE PREÇOS – O PROBLEMA

O problema do sistema de preços consiste em desenvolver um sistema que recebe preços enviados por algum cliente, salva-os em arquivo e reporta ao cliente que o dado foi salvo, também pode receber mensagens pedindo para pesquisar o menor valor em dado raio de busca e deve retornar o menor valor encontrado na área para o cliente que requisitou a pesquisa. Essa comunicação deve ser feita com o protocolo UDP.

O arquivo ***postos.txt***, ou melhor dizendo, banco de dados de postos, armazena os valores recebidos, sendo definido linha a linha, como:
(Dado) (ID Mensagem de cadastro) (Tipo) (Valor x 1000) (Latitude) (Longitude)

O tipo do combustível recebido para salvar no arquivo é dado por:

0- Diesel
1- Álcool
2- Gasolina

O problema foi solucionado em python com auxilio das bibliotecas socket, re, ipaddress, _thread, sys, random, time e numpy.

2.2 IMPLEMENTAÇÃO DO CLIENTE

O programa cliente, ao ser inicializado, pedirá para que seja digitado *cliente <ip> <porta>*, enquanto não detectar essa entrada ele reportará *Dados inválidos*. Caso os dados digitados sejam válidos, porém o programa não estabeleça uma conexão com o servidor, ele irá encerrar. Toda a comunicação é feita por UDP.

O cliente apenas envia e recebe uma string, toda a validação de dados e manipulação de objetos fica para o servidor. Caso ele não receba nenhuma resposta do servidor dentro de 5 segundos, ocorrerá um timeout. O cliente aceitará enviar uma string para inserir dados ou pesquisar dados até que detecte um *KeyboardInterrupt* (CTRL-C), encerrando a execução do programa. Caso os dados enviados sejam do tipo D e estejam corretos, o servidor reportará que os dados enviados foram cadastrados com sucesso, caso seja do tipo P, o servidor reportará se o posto com menor valor no raio desejado e do tipo de combustível desejado, se não existir posto que satisfaça essas condições ele reportará que não foi localizado nenhum posto no raio de busca. O cliente sempre imprime a resposta do servidor.

2.3 IMPLEMENTAÇÃO DO SERVIDOR

O programa servidor, ao ser inicializado, pedirá para que seja digitado *servidor <porta>*, enquanto não detectar essa entrada ele reportará *Dados inválidos*. Caso os dados digitados sejam válidos ele irá prosseguir. Após o socket ser instanciado o programa servidor verificará se existe o arquivo postos.txt para carregar em memória todos os postos do arquivo, onde devem ser realizadas as buscas, caso algum cliente requisite um preço.

Aqui é onde se encontra a lógica do sistema, existe a classe **Posto()** que define o objeto Posto e será explanada mais à frente.

Após validar os dados, será criado um novo socket com a porta especificada que irá escutar por pacotes UDP enviados por um cliente, quando detectar que recebeu um pacote contendo uma mensagem ele verificará se a mensagem pede para inserir ou buscar dados. Caso a mensagem comece com D e possua os 6 atributos necessários (Dado, ID, Tipo, Valor, Latitude, Longitude) será criada um novo objeto Posto que será adicionado à lista de postos carregadas

na memória e será adicionado no final do arquivo ***postos.txt*** o servidor então irá reportar que os dados foram recebidos com sucesso. Se a mensagem começar com P, o servidor chamará o método ***search(tipo, raio, latitude, longitude)*** utilizando como parâmetros os dados recebidos na mensagem (tipo de combustível, raio de busca, latitude do centro, longitude do centro), se for encontrado algum posto o servidor reporta ao cliente o valor encontrado, caso contrário reporta que não foi encontrado nenhum dado correspondente. Se for recebida uma mensagem com dados inválidos o servidor reporta para o cliente que enviou a mensagem que nenhum dado foi encontrado.

A classe Posto está documentada abaixo:

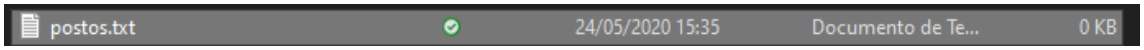
Class Posto:

msg_type -- propriedade (char D) que define um dado cadastrado
msg_id -- propriedade (int) que define o id da mensagem de cadastro
fuel_type -- propriedade (int) que define o tipo de combustível
fuel_price -- propriedade (int) que define o preço x 1000 do combustível
latitude -- propriedade (int) que define a latitude do posto
longitude -- propriedade (int) que define a longitude do posto

def __init__ (type, id, fuel_type, price, lat, long)
-- Método que inicializa o objeto, recebe os dados da mensagem como parâmetro
é responsável por transformar o valor do preço de char para int e multiplicar por 1000
def getString
-- Retorna os atributos do posto como uma única string
def saveToFile
-- Salva o objeto posto na ultima linha do arquivo postos.txt

2.4 TESTES

- Servidor inicializado na porta 8080 e arquivo postos.txt vazio:




```
Para começar digite:
servidor <porta>

servidor 8080
```

- Iniciar o cliente:

```
Para sair use CTRL+C

Para começar digite:
cliente <ip> <porta>

cliente 192.168.100.29 8080
Digite D para cadastrar posto ou P para pesquisar posto
```

- Se eu digitar uma mensagem invalida no cliente:

```
Digite D para cadastrar posto ou P para pesquisar posto
teste
Resposta do servidor: dados invalidos
```

- Tela do servidor:

```
servidor 8080
('192.168.100.29', 52845) teste
Dados invalidos
```

- Pesquisar posto sem nenhum posto cadastrado:

```
teste
Resposta do servidor: dados invalidos
P 23 0 5 33.55 37.89
Resposta do servidor: dados nao encontrados
```

- Tela do Cliente:

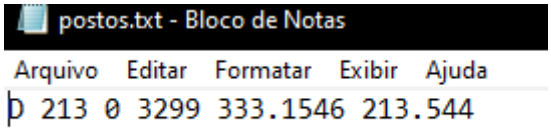
```
servidor 8080
('192.168.100.29', 52845) teste
Dados invalidos
('192.168.100.29', 52845) P 23 0 5 33.55 37.89
Buscando dados...
Nenhum posto localizado...
```

- Enviar D 213 0 R$3.299 333.1546 213.544 (cliente):

```
D 213 0 R$3,299 333.1546 213.544
Resposta do servidor: dados recebidos
```

- Servidor:

```
('192.168.100.29', 52845) D 213 0 R$3,299 333.1546 213.544
Posto inserido no banco de dados!
```



- Enviar vários Postos para cadastro:

```
D 213 0 R$3,299 333.1546 213.544
Resposta do servidor: dados recebidos
D 5 0 R$3 31.5 70.89
Resposta do servidor: dados recebidos
D 2 1 R$3,2 55.5 105.5
Resposta do servidor: dados recebidos
D 33 2 R$3,31 768.543 1000.568
Resposta do servidor: dados recebidos
```

- Servidor:

```
Posto inserido no banco de dados!
('192.168.100.29', 52845) D 5 0 R$3 31.5 70.89
Posto inserido no banco de dados!
('192.168.100.29', 52845) D 2 1 R$3,2 55.5 105.5
Posto inserido no banco de dados!
('192.168.100.29', 52845) D 33 2 R$3,31 768.543 1000.568
Posto inserido no banco de dados!
```

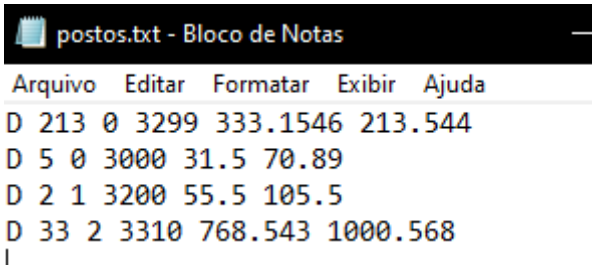
- Buscar menor valor na latitude 50 e longitude 100 com raio 6 do tipo 1:

```
P 1 1 6 50 100
Resposta do servidor: dados encontrados / R$ 3.2
```

- Servidor:

```
('192.168.100.29', 52845) P 1 1 6 50 100
Buscando dados...
Posto localizado...
```

- Arquivo postos.txt:



2.5 CONCLUSÃO

Essa tarefa foi mais fácil do que a primeira, foi apenas necessário compreender o problema e adaptá-lo para o UDP de forma que o cliente saiba se a transmissão de dados foi efetuada com sucesso, o servidor está sempre ouvindo novas requisições e respondendo conforme necessário. Ambos, servidor e cliente, mostram na tela as mensagens que recebem.

2.6 REFERÊNCIAS

https://kharisecario.wordpress.com/2017/03/25/create-nxn-matrix-in-pythonnumpy/

https://realpython.com/python-sockets/

https://wiki.python.org.br/SocketBasico

https://stackoverflow.com/questions/58424446/how-can-i-ensure-that-a-client-received-my-message-using-udp-sockets-in-python