Trabalho Prático – Redes I – Rebeca Neto – 617314

1 – Batalha Naval

- ## Sumário do problema a ser tratado.

O problema a ser tratado consiste em desenvolver o jogo batalha naval para o paradigma cliente-servidor, usando o protocolo TCP. Assim como colocado nas especificações do trabalho, o tabuleiro da batalha naval tem tamanho 10x10, no qual cada quadrado é identificado por uma letra na horizontal e por um número na vertical. Os navios disponíveis para alocação são porta-aviões (cinco quadrados), navios-tanque (quatro quadrados), contratorpedeiros (três quadrados) e submarinos (dois quadrados). O jogo consiste em escolher um espaço do tabuleiro oponente e tentar acertar um dos navios da frota. O navio afunda quando todos quadrados forem adivinhados pelo oponente, quem perder todos os navios primeiro perde o jogo.

Estabelecida a conexão entre o cliente e o servidor, a frota do cliente é posicionada de acordo com o arquivo de texto "frotaC.txt", o servidor posiciona a sua aleatoriamente. Após a criação dos tabuleiros o jogo se inicia, o cliente via teclado irá escolher onde será feito seu ataque e a resposta é enviada para o servidor que responde indicando se acertou ou não um navio e o seu ataque.

- ## Algoritmos, tipos abstratos de dados, principais funções e procedimentos, decisões de implementação.

Para funcionamento e organização do problema foram criadas as seguintes classes:

1. Class Mapa (arquivo mapa.py): Define o tamanho do tabuleiro (atributos: eixoX e eixoY) e possui duas funções para verificar se as coordenadas recebidas como parâmetros estão dentro do limite do tabuleiro.

2. Class Navio(arquvio navio.py):

    2.1. – Estruturas de dados/Atributos da classe:

    self.coordIX , self.coordIY, self.coordFX, self.coordFY => armazena coordenadas iniciais e finais da posição do navio

    self.vivo => controle se foi afundado ou não

    self.danos => coordenadas no navio que já foram atingidas

    self.lista_posicoes => todas as coordenadas que o navio ocupa

    2.2. – def verifica_dano(posX, posY) => baseadas nas coordenadas recebidas, verifica se atingiu o navio e atualizar a lista_danos e a variável vivo

    2.3. – def verificar_existencia(posX, posY) => verificar se as coordenadas recebidas referenciam alguma posição do navio

    2.4. – def listar_posicoes() => settar a lista de posições

3. Class Servidor_TCP (arquivo servidor_tcp.py): Classe que faz o gerenciamento do jogo, abre a conexão do servidor e envia/recebe os ataques

    3.1. – def iniciaConexao(): coloca o servidor no modo de escuta, faz a conexão com o cliente, chama a função para iniciar o jogo e fecha a conexão

    3.2. – def jogar(): Inicializa os dois tabuleiros(Class Cliente e Class Servidor), faz o gerenciamento dos turnos de cada jogador e computa os ataques deles, retorna quem venceu o jogo ou se o cliente encerrou o jogo antecipadamente

    Para facilitar o gerenciamento do jogo, optei por criar uma cópia do tabuleiro do cliente no servidor, caso não houvesse isso cada parte teria que verificar seu próprio tabuleiro ao receber o ataque e enviar o "status" dele para a outra parte. Preferi centralizar essa vigilância na parte do servidor

4. Class Cliente_TCP (arquivo cliente_tcp.py): Faz a conexão com o servidor, cria o tabuleiro do cliente para controle próprio, lê as entradas do teclado, envia o ataque para o servidor e receba as respostas

    4.1. – Estruturas de dados/Atributos da classe:

    self.lista_de_ataques => lista de coordenadas dos ataques realizados no mapa inimigo

    self.mapa = mapa => tipo Mapa

    self.lista_de_navios => lista de Navios da frota do Cliente

    self.lista_ataque_recebido => lista de coordenadas ataques recebidos do servidor

    self.lista_acertos => lista de coordenadas de ataques que acertaram um navio

    self.lista_afundados => lista de coordenadas de navios que já foram afundados no mapa inimigo

    self.mortosAtuais => string com a lista dos números dos navios inimigos já afundados

    4.2. – def jogar(msg): enviar os ataques para o servidor, recebe as respostas do servidor, controle dos acertos do cliente e ataques recebidos, finaliza o jogo e finaliza a conexão

    A resposta do servidor é um caracter 'A' ou 'E' (A – acerto, E – erro em relação ao último ataque do cliente), seguida de uma letra e um número (as coordenas de ataque do servidor) e uma sequência de números caso haja navios afundados no mapa do servidor. Ex: AE313 - último ataque acertou um navio, o servidor atacou a posição E3 e os seus navios já afundados é o navio de número 1 e 3.

    4.3. – def turno_cliente(): faz a leitura do teclado: ataque, imprimir ou sair

    4.4. – def print_mapaCliente(): de acordo com o posicionamento da sua frota e os ataques recebidos, armazena as informações em uma matriz e imprimi o seu mapa ('X' = erro, 'a' = navio atingido, '@' = navio afundado, 'o' = navio não atingido)

4.5. – def print_mapaServidor(): de acordo com os ataques feitos e os acertos nos navios inimigos, armazena as informações em uma matriz e imprimi o mapa conhecido do servidor (‘X’ = erro, ‘o’ = navio atingido, ‘@’ = navio afundado)

4.6. – def inicializar_cliente(): faz a leitura do arquivo, posiciona a frota, imprimi quais navios foram criados e retorna se o posicionamento é válido ou não

4.7. - def verificar_posicao(): recebidas duas coordendas, verifica se já existe algum navio no local

4.8. – def atacado(coordX, coordY): atualiza a lista de ataques recebidos e a chama a função que da classe Navio que atualizar a variável “vivo”

4.9. – def criar_navio(): recebes as coordenadas, verifica se é possível criar o navio e o adiciona a lista de navios da classe

5. Class Cliente_Jogador(arquivo cliente.py): Semelhante a classe Cliente_TCP, possui as funções criar_navio, verificar_posicao, inicializar_cliente e os atributos lista_de_ataques, self.mapa, self.lista_de_navios

5.1. – def atacar(coordX, coordY, jogador): verifica se as coordenadas informadas acertam um navio do jogador inimigo

5.2. – def verificar_status(): verifica se existe pelo menos 1 navio vivo no tabuleiro do cliente

5.3. – def verificar_vivos(): retorna a quantidade de navios do cliente que ainda estão vivos

5.4. – def verificar_mortos(): retorna a quantidade de navios do cliente que foram afundados

5.5. – def getMortos(): retorna todas as posições dos navios já afundados

6. Class Servidor_Jogador(arquvio servidor.py): Semelhante a classe Cliente, possui as funções criar_navio, atacar, verificar_posicao, verificar_status, verficar_vivos, verificar_mortos e os atributos lista_de_ataques, self.mapa, self.lista_de_navios.

6.1. – Estruturas de dados/Atributos da classe:

self.lista_acertos => lista das últimas coordenadas de ataques que acertaram um navio, sempre que um navio é afundado a lista é zerada

self.orientacao => controle para determina qual o próximo vizinho a ser escolhido

self.qntH, self.qntV => controle para determina qual o próximo vizinho a ser escolhido

self.origA => controle para saber qual é a orientação do navio que está tentando afundar

6.2. – def atacarNovo(coordX, coordY): verificar se as coordenadas já foram usadas para ataque

6.3. – def inicializar_servidor(): posiciona a frota escolhendo pontos aleatórios

6.4. – def turno_serv(): gera as coordenadas do próximo ataque do servidor.

Explicação turno_serv: Caso não tenha acertado nenhum navio, gera uma coordenada aleatoriamente. Caso contrário ele tenta acerta um vizinho seguindo a seguinte lógica: a cada acerto, as coordenadas são armazenadas na lista_acerto, no próximo turno tenta-se acerta a posição do vizinho em baixo, na próxima à cima, depois na direita e depois à esquerda da primeira coordenada que não se visitou todos os vizinhos. Se uma delas acertar é definida a posição do navio alvo, se for horizontal nas próximas vezes tentará acerta somente direita/esquerda, se for vertical para cima/baixo.

Conexão TCP

```
host = sys.argv[1] #ip/nome
port = int(sys.argv[2]) #porta

# Cria um socket TCP/IP
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

s.connect((host, port))
```

Na classe Cliente_TCP a criação e conexão com o servidor é feita da forma mostrada acima. O parâmetro “socket.SOCK_STREAM” caracteriza que a conexão é usando o TCP.

```
# Cria um socket TCP/IP
self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
#garante que o socket será destruído (pode ser reusado) após uma interrupção da execução
self.s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# associa o socket a porta
self.s.bind((self.host, self.port))

print('Server %s on port %s' % (socket.gethostname(), self.port))
self.s.listen(1)
```

O trecho de código acima, na classe Servidor_TCP, o socket é criado e o servidor é posto no modo escuta e no código abaixo ele aceita a conexão feita pela porta indicada:

```
# Espera por novas conexões
print('waiting for a connection')
# Aceita conexões
self.conn, self.address = self.s.accept()
```

O recebimento de mensagem é através do “.recv(1024)” onde a resposta é recebidas em bytes e o “.sendall()” envia a mensagem.

```
s.sendall(bytes(msg.encode()))

dataR = s.recv(1024)
respS = str(dataR.decode())
```

Para ambos o envio e recebimento de resposta usa o mesmo método e o termino da conexão é feita com o “.close()”

- **Testes e Print screens**

```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\batalhaNaval> python servidor_tcp.py 1234
Server DESKTOP-Rebeca on port 1234
waiting for a connection
```

Inicialização do servidor

```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\batalhaNaval> python cliente_tcp.py 127.0.0.1 1234
Navio  1  criado
Navio  2  criado
Navio  3  criado
Navio  4  criado
Digite S para começar a batalha naval
```

Inicialização do cliente

```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\batalhaNaval> python servidor_tcp.py 1234
Server DESKTOP-Rebeca on port 1234
waiting for a connection
Connected to DESKTOP-Rebeca on port 1234
```

Após iniciar o cliente, o servidor já aponta que foi feita a conexão.

```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\batalhaNaval> python cliente_tcp.py 127.0.0.1 1234
Navio  1  criado
Navio  2  criado
Navio  3  criado
Navio  4  criado
Digite S para começar a batalha naval
s
Servidor responde: Tabuleiros Criados
Vertical(letra):
```

Depois de digitar o ‘s’, o jogo se inicia.

Obs: O código foi manipulado para que o primeiro ataque do servidor fosse no F3 para mostrar a decisão da ordem: vizinho de baixo, de cima, a direita e depois a esquerda (quando não se tem a orientação do navio ainda).

```
Servidor responde: Tabuleiros Criados
Vertical(letra):a
Horizontal(número):8
Servidor responde: Acertou um navio. Ataque: F 3
Vertical(letra):a
Horizontal(número):7
Servidor responde: Tiro no mar. Ataque: G 3
Vertical(letra):p
Mapa Cliente
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |o |o |o |  |
B |  |  |o |  |  |  |  |  |o |  |
C |  |  |o |  |  |  |  |  |o |  |
D |  |  |o |  |  |  |  |  |  |  |
E |  |  |o |  |  |  |  |  |  |  |
F |  |  |  |a |o |o |o |o |  |  |
G |  |  |  |X |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |  |
J |  |  |  |  |  |  |  |  |  |  |
Mapa Servidor
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |  |X |O |  |
B |  |  |  |  |  |  |  |  |  |  |
C |  |  |  |  |  |  |  |  |  |  |
D |  |  |  |  |  |  |  |  |  |  |
E |  |  |  |  |  |  |  |  |  |  |
F |  |  |  |  |  |  |  |  |  |  |
G |  |  |  |  |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |  |
J |  |  |  |  |  |  |  |  |  |  |
Vertical(letra):
```

O cliente fez os ataques a8, a7, assim como ilustrado na “impressão”, e o servidor respondeu em cada um “Tiro no mar” ou “Acertou um navio” seguido do seu ataque. O servidor fez o primeiro ataque no F3, depois tentou acertar o vizinho de baixo. No mapa: X – tiro no mar, a – acerto, o – onde tem navio ainda não atingido

```
Vertical(letra):a
Horizontal(número):9
Servidor responde: Acertou um navio. Ataque: E 3
Navios inimigos já afundados: 4
Vertical(letra):p
Mapa Cliente
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |o |o |o |  |
B |  |  |o |  |  |  |  |  |o |  |
C |  |  |o |  |  |  |  |  |o |  |
D |  |  |o |  |  |  |  |  |  |  |
E |  |  |o |X |  |  |  |  |  |  |
F |  |  |  |a |o |o |o |o |  |  |
G |  |  |  |X |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |  |
J |  |  |  |  |  |  |  |  |  |  |
Mapa Servidor
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |  |X |O |O |
B |  |  |  |  |  |  |  |  |  |  |
C |  |  |  |  |  |  |  |  |  |  |
D |  |  |  |  |  |  |  |  |  |  |
E |  |  |  |  |  |  |  |  |  |  |
F |  |  |  |  |  |  |  |  |  |  |
G |  |  |  |  |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |  |
J |  |  |  |  |  |  |  |  |  |  |
Vertical(letra):
```

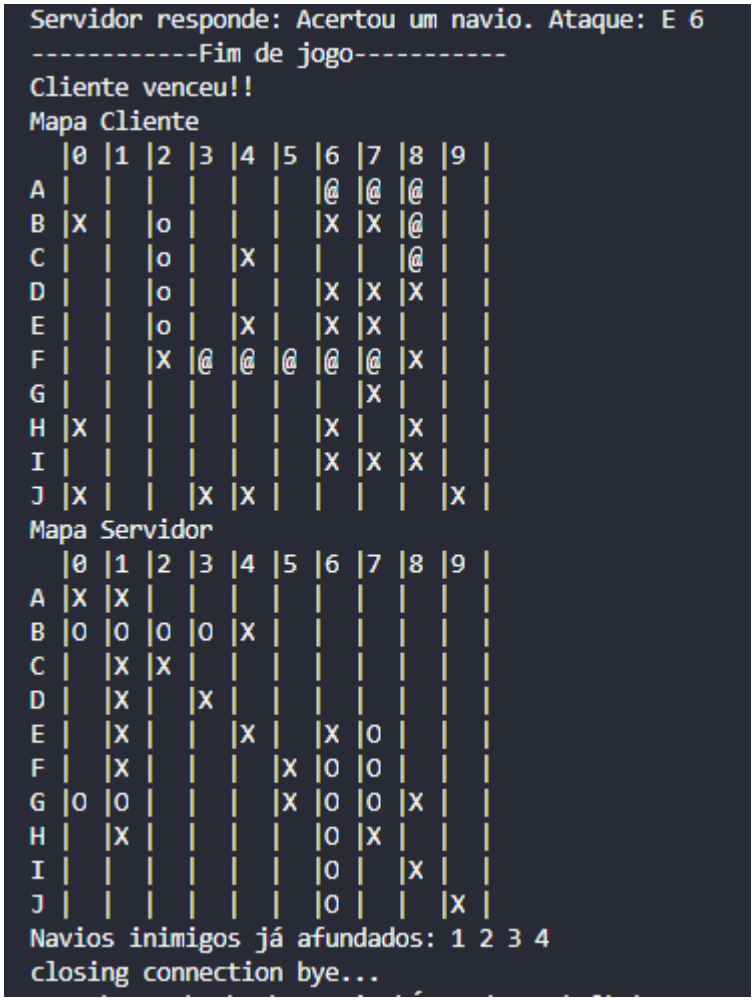
No próximo ataque do cliente, o navio 4 é afundado e servidor continua sua sequência de tentativas para cima.

```
Vertical(letra):d
Horizontal(número):5
Servidor responde: Tiro no mar. Ataque: F 4
Vertical(letra):i
Horizontal(número):9
Servidor responde: Tiro no mar. Ataque: F 2
Vertical(letra):p
Mapa Cliente
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |o |o |o |  |
B |  |  |o |  |  |  |  |  |o |  |
C |  |  |o |  |  |  |  |  |o |  |
D |  |  |o |  |  |  |  |  |  |  |
E |  |  |o |X |  |  |  |  |  |  |
F |  |  |X |a |a |o |o |o |  |  |
G |  |  |  |X |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |  |
J |  |  |  |  |  |  |  |  |  |  |
Mapa Servidor
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |  |X |O |O |
B |  |  |  |  |  |  |  |  |  |  |
C |  |  |  |  |  |  |  |  |  |  |
D |  |  |  |  |  |X |  |  |  |  |
E |  |  |  |  |  |  |  |  |  |  |
F |  |  |  |  |  |  |  |  |  |  |
G |  |  |  |  |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |X |
J |  |  |  |  |  |  |  |  |  |  |
Vertical(letra):
```

No ataque seguindo o servidor acerta o vizinho à direita e determina que a posição do navio é horizontal, por isso tenta acertar o vizinho à esquerda em seguida.

```
Vertical(letra):h
Horizontal(número):2
Servidor responde: Tiro no mar. Ataque: F 5
Vertical(letra):b
Horizontal(número):7
Servidor responde: Tiro no mar. Ataque: F 6
Vertical(letra):e
Horizontal(número):4
Servidor responde: Tiro no mar. Ataque: F 7
Vertical(letra):p
Mapa Cliente
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |o |o |o |  |
B |  |  |o |  |  |  |  |  |o |  |
C |  |  |o |  |  |  |  |  |o |  |
D |  |  |o |  |  |  |  |  |  |  |
E |  |  |o |X |  |  |  |  |  |  |
F |  |  |X |@ |@ |@ |@ |@ |  |  |
G |  |  |  |X |  |  |  |  |  |  |
H |  |  |  |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |  |
J |  |  |  |  |  |  |  |  |  |  |
Mapa Servidor
  |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |
A |  |  |  |  |  |  |  |X |O |O |
B |  |  |  |  |  |  |  |X |  |  |
C |  |  |  |  |  |  |  |  |  |  |
D |  |  |  |  |  |X |  |  |  |  |
E |  |  |  |  |X |  |  |  |  |  |
F |  |  |  |  |  |  |  |  |  |  |
G |  |  |  |  |  |  |  |  |  |  |
H |  |  |X |  |  |  |  |  |  |  |
I |  |  |  |  |  |  |  |  |  |X |
J |  |  |  |  |  |  |  |  |  |  |
Vertical(letra):
```

Depois de mais alguns ataques ele afunda o navio (marcado com @ na imagem)



Ao fim do jogo os dois tabuleiros são mostrados e a conexão é finalizada. O servidor continua no modo escuta e aguarda outra conexão.

2 – Sistema de preços

- ## Sumário do problema a ser tratado.

O problema consiste em desenvolver um sistema para enviar o preço de vários postos de combustíveis e requisitar o preço mais barato em uma dada região, utilizando o protocolo UDP. Nesse programa o cliente poderá enviar dois tipos de mensagens ao servidor: dados (D) e pesquisa (P). As mensagens de dados que começam com a letra D são seguidas de um inteiro identificador da mensagem, um inteiro que indica o tipo de combustível (0 - diesel, 1 - álcool, 2- gasolina), um inteiro com o preço x 1000 (ex: R$3,299 fica 3299) e as coordenadas do posto de combustível (latitude e longitude). O servidor confirma a recepção da mensagem e adiciona a informação no arquivo de texto “postos.txt”. As mensagens de pesquisa que começam com a letra P são seguida de um inteiro identificador da mensagem, um inteiro que indica o tipo de combustível (0 - diesel, 1 - álcool, 2- gasolina), um inteiro com o raio de busca e as coordenadas do centro de busca (latitude e longitude). O servidor responde com o menor preço para aquele combustível para postos de combustível que dentro da área formada pelo centro de busca e o raio de busca.

- ## Algoritmos, tipos abstratos de dados, principais funções e procedimentos, decisões de implementação.

Para funcionamento e organização do problema foram criadas as seguintes classes:

1. Class Servidor_UDP:
    1.1. def comunicao(): Faz a leitura dos dados recebidos, imprime na tela a mensagem recebida, o endereço de origem e a porta em que está sendo feita a comunicação e envia a mensagem para o cliente
    1.2. def adcionar(adc): Adiciona o tipo de combustível, preço e localização do posto no arquivo (longitude e latitude respectivamente) . Ex: 1 3999 2 5
    1.3. def pesquisar(pesq): Lê todas as linhas do arquivo e verifica uma a uma para saber se o tipo de combustível é igual ao procurado, caso seja, verifica se está dentro do raio e compara com o menor preço já encontrado. Ela retorna o menor preço do arquivo.
2. Class Cliente_UDP:
    2.1. def comunicacao(): Eniva os dados digitados para o servidor, imprime na tela o identificador da mensagem, faz leitura dos dados recebidos e os imprime na tela, fecha a conexão do socket
    2.2. def leitura(): Lê os dados do teclado digitados pelo usuário e retorna a mensagem a ser enviada para o servidor
    2.3. def reenviaErro(): Quando há um erro ao receber a mensagem, tenta reenviar novamente com um intervalo de 5 segundos entre cada tentativa e em cada uma delas é verificado se o identificador da mensagem recebida é igual a envida, o número máximo de tentativas é 6, ou seja, 30 segundos

Optei por deixar somente 30 segundos pois meus testes foram em uma só máquina onde o servidor e cliente estavam juntos, por isso o tempo de espera para o recebimento de mensagens não deveria oscilar muito. Apesar disso, não foi possível testar a falha de envio da mensagem.

Conexão UDP

```
self.host = host
self.porta = port
#cria um UDP/IP socket
self.s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

Na classe Cliente_UDP a criação do socket é feita da forma mostrada acima e não é necessária fazer uma conexão com o servidor igual no TCP. O parâmetro “socket.SOCK_DGRAM” caracteriza que a conexão é usando o TCP

```
self.host = ''
self.porta = port

#cria um UDP/IP socket
self.s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
#garante que o socket será destruído (pode ser reusado) após uma interrupção da execução
self.s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
#associa o socket à uma porta
self.s.bind((self.host, self.porta))
```

O trecho de código acima, na classe Servidor_UDP, o socket é criado e é associado a porta. No código abaixo ele fica esperando uma nova mensagem ser recebida. No recebimento (“recvfrom”), a variável “data” é a mensagem em bytes e o “address” é composto pelo endereço IP da origem e a porta pela qual os dois irão se comunicar:

```
#Espera por novas mensagens
print('waiting to receive message')
#Recebe a mensagem
data, address = self.s.recvfrom(1024)
```

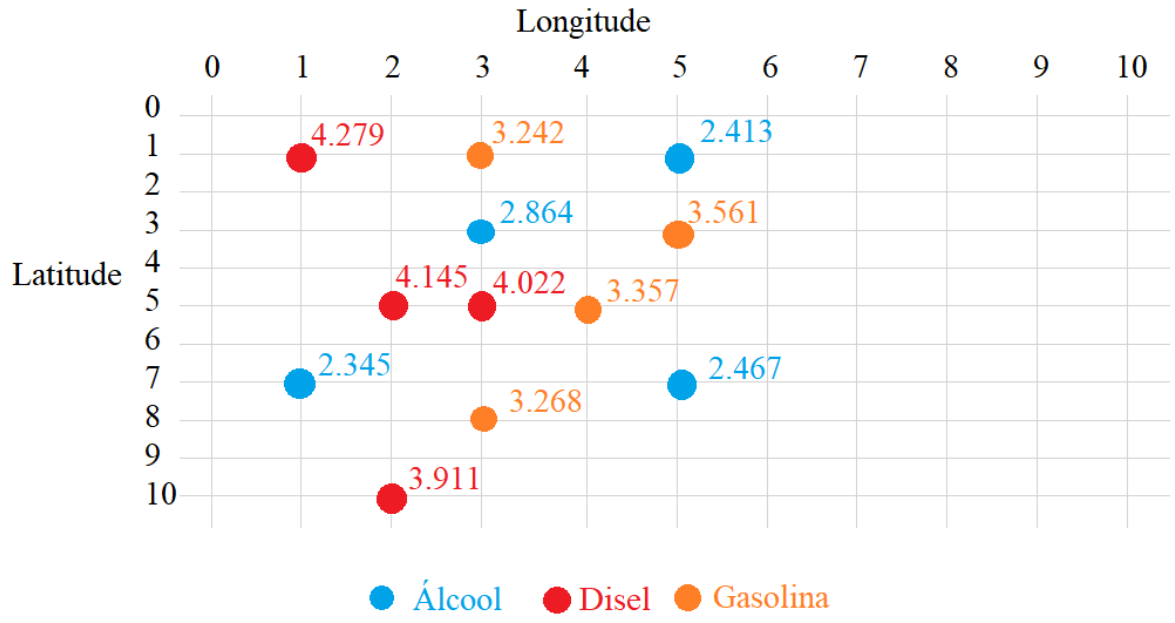
O envio de mensagem usa “.sendto()” que envia a mensagem em bytes e envia para o mesmo IP e pela mesma porta recebidos.

```
data = "Mensagem " + msg[1]  + " recebida\n" + resp
self.s.sendto(bytes(data.encode()), (address[0], address[1]))
```

Para ambos o envio e recebimento de resposta usa o mesmo método e o termino da conexão é feita com o “.close()”

- **Testes e Print screens**

Para os testes, o arquivo de dados inicial estava organizado da seguinte forma:




```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\sistemaPreços> python servidor.py 1234
waiting to receive message
```

Inicialização do servidor

```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\sistemaPreços> python cliente.py 127.0.0.1 1234
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q):
```

Inicialização do cliente

```
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q): p
Escolha o tipo de combustivel (0 - diesel, 1 - álcool, 2- gasolina): 1
Digite o raio de pesquisa: 5
Digite a longitude: 0
Digite a latitude: 0
Mensagem número  1  enviada.
Servidor responde:  Mensagem 1 recebida
O menor preço da região é 2413
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q):
```

No terminal do cliente: para a pesquisa P 1 1 5 0 0, teve a confirmação e a resposta do menor preço.

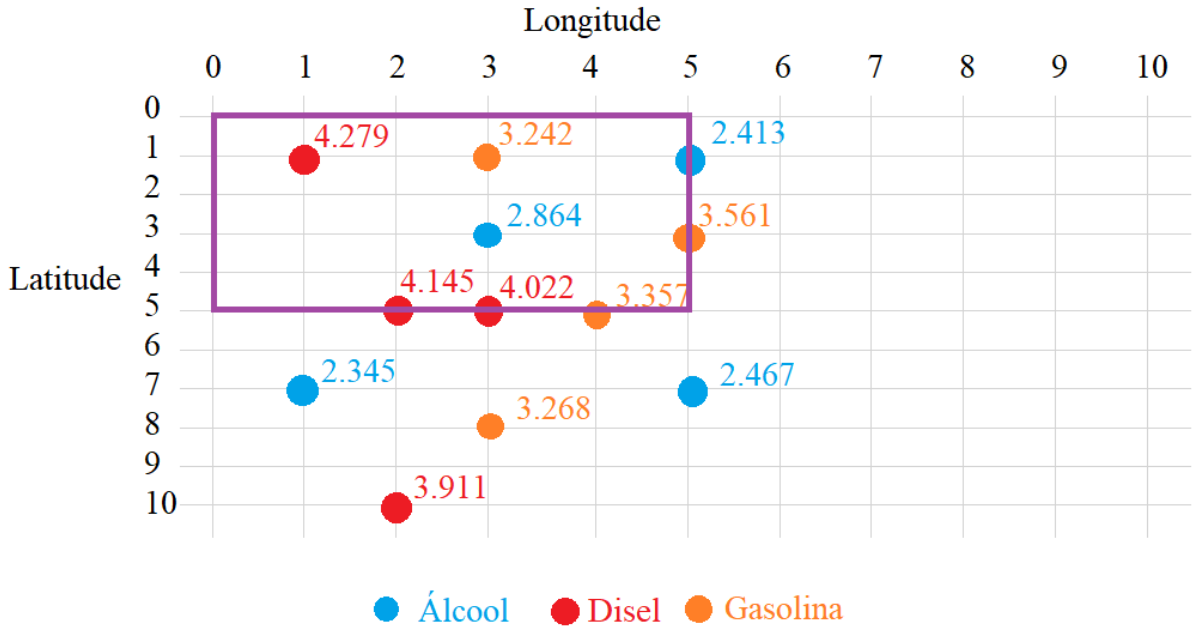


Ilustração da pesquisa feita

```
waiting to receive message
Msg recebida: P 1 1 5 0 0
De: 127.0.0.1
Escutando na porta: 62110
waiting to receive message
```

No terminal do servidor: Tem a exibição da mensagem recebida, do ip origem e da porta que foi feita a comunicação.

```
O menor preço da região é 2413
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q): p
Escolha o tipo de combustivel (0 - diesel, 1 - álcool, 2- gasolina): 1
Digite o raio de pesquisa: 7
Digite a longitude: 0
Digite a latitude: 0
Mensagem número  2  enviada.
Servidor responde:  Mensagem 2 recebida
O menor preço da região é 2345
```

Expandindo o raio de busca, a resposta muda e o identificador da mensagem têm o acréscimo de uma unidade.

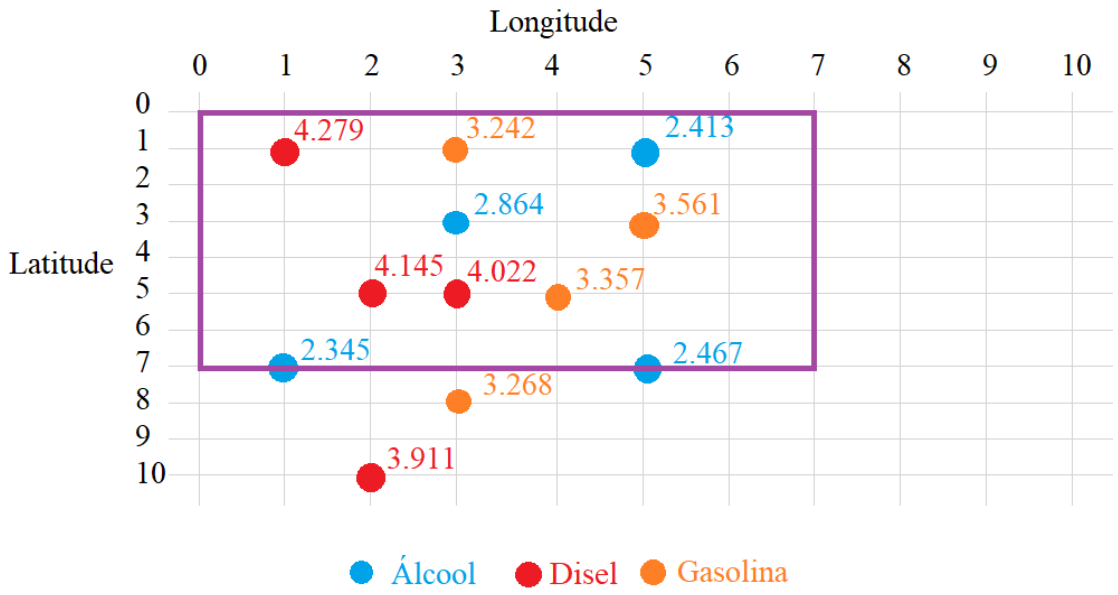


Ilustração para a nova pesquisa

```
PS C:\Users\rebec\OneDrive\Área de Trabalho\TP - Redes\sistemaPreços> python cliente.
py 127.0.0.1 1234
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q): d
Escolha o tipo de combustivel (0 - diesel, 1 - álcool, 2- gasolina): 1
Digite o preço do combustivel(Tam min 4): 2344
Digite a longitude: 7
Digite a latitude: 7
Mensagem número  1  enviada.
Servidor responde:  Mensagem 1 recebida
O novo posto da coordenada 7:7 foi adiconado.
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q):
```

Iniciando um novo cliente (cliente 2), foi adicionado um novo posto de álcool na coordenada 7,7 com um preço menor que o da pesquisa anterior. Para esse cliente o número identificador começa do 1 também e há a confirmação da mensagem.

```
waiting to receive message
Msg recebida: P 2 1 7 0 0
De: 127.0.0.1
Escutando na porta: 62110
waiting to receive message
Msg recebida: D 1 1 2344 7 7
De: 127.0.0.1
Escutando na porta: 61661
waiting to receive message
```

No terminal do servidor, tem o histórico da última pesquisa e a nova mensagem. Nota-se que esse cliente usa uma nova porta, diferente da usada pelo cliente 1, o IP origem é igual pois os dois estão na mesma máquina.

```
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q): p
Escolha o tipo de combustivel (0 - diesel, 1 - álcool, 2- gasolina): 1
Digite o raio de pesquisa: 7
Digite a longitude: 0
Digite a latitude: 0
Mensagem número  5  enviada.
Servidor responde:  Mensagem 5 recebida
O menor preço da região é 2344
Deseja enviar dados(D) ou pesquisar(P) ou sair(Q):
```

```
waiting to receive message
Msg recebida: P 5 1 7 0 0
De: 127.0.0.1
Escutando na porta: 62110
waiting to receive message
```

O cliente 1 faz uma pesquisa igual à anterior e o servidor retorna o novo menor valor.

3 – Conclusão e referências bibliográficas

- ## Conclusão

O uso da biblioteca de sockets facilita muito a implementação da comunicação entre um servidor e os clientes

- ## Referências bibliográficas

http://www2.ic.uff.br/~debora/praticas/aplicacao/

https://riadnassiffe.github.io/disciplinas/lp1/codigos-de-exemplo/

https://www.w3schools.com/python/