




# Restaurante - P2P

## Guião de Avaliação 1

João Santos - 73761  
Diego Trovisco - 80228



# Criação do Anel - NODE\_JOIN()

A partir do momento que o cliente tem um ponto fixo de entrada no anel tornou a implementação do mesmo bastante mais parecida com o DHT, sendo que a maior diferença diz respeito à não existência de relação com antecessor.

1. O Restaurante, sendo a unidade central do anel, forma um anel com ele mesmo e fica á espera de pedidos de entrada no anel. Atributo *self.inside\_ring* = *True*.
2. Todas as outras entidades sabem qual é o ponto de entrada no anel, assim enviam pedidos de NODE\_JOIN para o restaurante e aguardam que lhe seja atribuído o seu sucessor com base nos ID's.
3. Se não existir nenhum sucessor, ou seja, não existe nenhum nó com ID superior ao dele mesmo é atribuído como seu sucessor o ID = 0, que corresponde ao nó de entrada e consequentemente forma um anel unidirecional. Atributo *self.inside\_ring* = *True*.
4. Quando não existem mais entidades que estejam a tentar juntar-se ao anel é assumido que o anel se encontra formado, estável e ordenado. Por isso podemos passar para a fase de naming pelos diferentes nós.  
*NODE\_DISCOVERY()*



# Descoberta no Anel - NODE\_DISCOVERY()

O processo de naming deveria ser totalmente dinâmico, ou seja, cada nó só precisa de resolver o endereço do qual precisa e quando precisa. Na nossa implementação os nós procedem à descoberta de todas as entidades do anel imediatamente depois da formação total do anel.

1. Todas as entidades no anel possuem um dicionário com o seu id e os id's das diferentes entidades a None.

```
self.ring_ids_dict = {'RESTAURANT': self.id, 'WAITER': None, 'CHEF': None, 'CLERK': None}
```

2. Sendo o restaurante a entidade “central” do anel é por ele que começa o envio de mensagem de Node\_Discovery() que contém o dicionário.
3. O seu sucessor vai comparar o dicionário recebido com o seu. Assim está em condições de fazer um “Join” dos dois dicionários e enviar para o seu sucessor.
4. Quando o restaurante recebe de volta o dicionário este vem completo. De seguida atualiza a variável `self.ring_completed = True` e envia novamente o dicionário para o sucessor para todos os nós atualizarem o seu dicionário e assim conhecerem todos os Nós/Id's no anel. Neste momento o restaurante está “Aberto” e pronto a receber clientes.



# Codificação de Mensagens

A codificação de mensagens é feita em pickle utilizando por base um dicionário do tipo: {"method":XXXX, "args": XXXXX}.

Métodos usados:

- "ORDER" - {"method": "ORDER", "args": PEDIDO}      \* PEDIDO vem do cliente
- "TICKET" - {"method": "TICKET", "args": {"number": self.count, "args": "o["args"]"}}
- "START" - {"method": "ORDER", "args": PEDIDO}
- "COOK" - {"method": "ORDER", "args": PEDIDO}
- "COOKED" - {"method": "ORDER", "args": PEDIDO}
- "DONE" - {"method": "ORDER", "args": PEDIDO}
- "PICKUP" - {"method": "ORDER", "args": PEDIDO}
- "DELIVERY" - {"method": "ORDER", "args": PEDIDO}



# Trabalho das Entidades

O método “ORDER” é enviado pelo cliente para o Restaurante e este envia para a Waiter. A Waiter recebe e envia um método “TICKET” dando-lhe um número de pedido que vai até ao Restaurante.

O Restaurante recebe o “TICKET” e envia o número do pedido ao cliente, e passa a mensagem de “START” para o seu sucessor. Quem aceita este método é o Chef que envia outro com o nome “COOK” até ao Restaurante. O Restaurante cozinha tanto no Grelhador no caso de o pedido ser “hamburger”, na Fritadeira no caso de o pedido ser “fries” e Bebidas no caso do pedido ser “drinks”.

Quando termina envia um método “COOKED” até ao Chef que recebe e envia um método “DONE”. Este método chega ao Restaurante e a variável `self.done` passa True que significa que passa aceitar o método “PICKUP” do cliente.

O método “PICKUP” vai até ao Clerk este envia um método “Delivery” e ao chegar novamente ao Restaurante este envia o pedido para o cliente.