Relatório EP DSID

Marcos Paulo Tomás Ferreira – NUSP 13747950
Rafael Moura de Almeida – NUSP 11225505

1 Detalhes da Implementação

Para a implementação, utilizamos a linguagem GO, que é concorrente, imperativa, estruturada, e de tipagem forte e estática. A escolha da linguagem se deu pela facilidade de trabalhar com paralelismo (threads), que em GO chamamos de *goroutines*. No código, escolhemos implementar as *goroutines* para deixar o servidor rodando em paralelo. Durante o processo de desenvolvimento, resolvemos optar por implementar operações não-bloqueantes, para não impedir o código de responder a um pedido de busca, caso este fosse feito paralelamente em um outro nó da rede, e assim o usuário poderia escolher alguma ação sem problemas.

Também fizemos um bash script que permite inicializar vários nós de uma vez em diferentes terminais, localizado em “/scripts/script.sh”. O script abre um menu com uma lista de topologias que o usuário pode escolher, cada uma localizada no caminho /txts/topologia_[$topologia]. A depender da escolha, o script abre um arquivo “len.txt” dentro de cada pasta representando uma topologia, que contém uma linha apenas, com o número de nós. Depois fica em um laço de 1 até n, abrindo os arquivos que contém os vizinhos e os pares chave-valor de cada nó, e executa o comando:

```
gnome-terminal -- bash -c "../src/UP2P '$HOST:$PORT' '../txts/$topologia/$i.txt' '../txts/$topologia/pares_$i.txt'" bash
```

para iniciar cada nó. Também criamos listas personalizadas de pares chave-valor na topologia de três estrelas com o auxílio do chatgpt, que gerou uma 100 pares chave-valor no formato NOME_SOBRENOME 1, e dividimos esses pares entre os diferentes nós da topologia de três estrelas.

Detalhe para a variável $HOST, que não tem um valor estático como “localhost” ou “127.0.0.1”, mas sim o endereço IP local da máquina, por exemplo, algo como 192.168.0.6. Para o script funcionar corretamente, nos arquivos de vizinhos deve ser passado o endereço IP da sua máquina na rede WAN para cada vizinho, ao invés de “localhost” ou “127.0.0.1”. Fizemos essa adaptação para permitir testar com duas máquinas diferentes, já que nesse caso o que identifica cada nó não é apenas a porta, mas o host e a porta em conjunto. O endereço IP local da máquina é obtido pelo script na linha:

```
HOST=$(ip address | grep -oE "\b192.168.[0-9]{1,3}.[0-9]{1,3}\b" | head -n 1)
```

Também foi acrescentada uma outra opção secreta a ser inserida no menu, que não é printada no terminal, mas que é a opção 7, que chama uma função que roda uma série de testes de forma automática.

2 Testes de Funcionamento

- **Demonstrar o funcionamento das operações HELLO e BYE**
  No anexos [1] e [2] é possível verificar que as duas operações funcionam corretamente; a topologia usada no teste foi a ciclo_3. Para testar a operação BYE, usamos a topologia linha 3; os anexos [2], [23], [24] e [25] demonstram o passo-a-passo do algoritmo.
- **Demonstrar que a lógica do TTL está sendo aplicada**

Pelos anexos [3], [4] e [5] é possível visualizar, utilizando a topologia linha, que o TTL é decrementado a cada enlace percorrido.

- **Demonstrar que a busca por flooding é capaz de encontrar uma chave que existe na rede**
  Os anexos [26] e [27] mostram o funcionamento da rede flooding em dois nós específicos: o que inicia a busca e o que encontra a chave na tabela local. A topologia usada, para este caso, foi a três triângulos.
- **Demonstrar que as mensagens de busca por flooding não são re-enviadas para o nó remetente**
  Em [28], [29] e [30], dentro da topologia ciclo 3, vemos que a mensagem não é enviada para o nó remetente.
- **Demonstrar que as mensagens de busca por flooding repetidas são descartadas**
  Usando a topologia de três triângulos, será demonstrado com um exemplo no qual o nó 10 inicia uma busca pela chave Joaquin_Phoenix, que está no nó 1.

- **Demonstrar que a busca por random walk é capaz de encontrar uma chave existente tanto numa rede sem ciclos quanto numa rede com ciclos**
  Aproveitando os anexos [3], [4] e [5], que utilizam a topologia linha, é possível ver que a chave é encontrada numa rede sem ciclos, via random walk. Para testar numa rede com ciclos, vamos utilizar a topologia grid 3x3. Os anexos [6], [7], [8], [9], [10], [11], [12], [13] e [14] mostram o funcionamento.
- **Demonstrar qua a busca em profundizdade é capaz de encontrar uma chave existente tanto numa rede sem ciclos quanto numa rede com ciclos**
  Para o teste de uma rede sem ciclos, utilizamos a topologia de árvore binária; os anexos [15], [16], [17], [18] e [19] mostram o funcionamento da rede via busca em profundidade para esta topologia. Para a rede com ciclos, utilizamos a topologia três triângulos é usada; os anexos [20], [21], [22] ilustram o funcionamento da rede.
- **Demonstrar a coleta correta de estatísticas**
  Para essa demonstração, utilizamos a topologia de três triângulos, com os vizinhos as chaves cada nó definidos em "/txts/topologia_tres_triangulos/[n].txt" e "/txts/topologia_tres_triangulos/pares_[n].txt", respectivamente. Rodamos o script definido na seção de implementação para abrir 10 terminais de forma automatizada, e utilizamos os pares chave-valor gerados pelo ChatGPT para cada nó.
  Foi criada uma função "func testesAutomatizados(no *node.No)", que é ativada secretamente através da tecla 7, e que roda uma série de testes automáticos. No total são realizadas 14 buscas para cada tipo de busca, contabilizando 42 buscas no total através da chamada dessa função. As chaves usadas para as buscas estão em uma lista de strings declarada no começo do arquivo main.go, sendo:

"Chris_Evans" → NÓ 7
"Kate_McKinnon" → NÓ 7
"Idris_Elba" → NÓ 6
"Julianne_Moore" → NÓ 6
"Ryan_Gosling" → NÓ 5
"Tilda_Swinton" → NÓ 5
"Emma_Watson" → NÓ 4
"Samuel_Jackson" → NÓ 4
"Joaquin_Phoenix" → NÓ 3
"Julia_Roberts" → NÓ 3
"Sandra_Bullock" → NÓ 2
"Brad_Pitt" → NÓ 2
"Emma_Stone" → NÓ 1

"Robert_Downey" → NÓ 1

Repare que as chaves foram escolhidas de forma que todas estivessem em nós que não fizessem parte do triângulo de nós do nó 10, sendo necessário a busca passar pelo nó 1 que liga os três triângulos. Isso foi feito para aumentar as chances de acontecerem ciclos no Random Walk e no Search in Depth. Após chamar uma vez a função testesAutomatizados(), as seguintes estatísticas foram obtidas com o uso da opção 5, na primeira tentativa com um valor de ttl de 100:

```
Estatisticas
        Total de mensagens de flooding vistas: 14
        Total de mensagens de random walk vistas: 14
        Total de mensagens de busca em profundidade vistas: 14
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.571429 0.937614
        Media e desvio padrao de saltos ate encontrar destino por random walk: 19.928571 21.556596
        Media e desvio padrao de saltos ate encontrar destino por busca em profuncidade: 5.571429 2.027286
```

Depois foi realizado um segundo teste, com valor de ttl de 100:

```
Estatisticas
        Total de mensagens de flooding vistas: 14
        Total de mensagens de random walk vistas: 14
        Total de mensagens de busca em profundidade vistas: 14
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.642857 1.008208
        Media e desvio padrao de saltos ate encontrar destino por random walk: 17.071429 12.542017
        Media e desvio padrao de saltos ate encontrar destino por busca em profuncidade: 5.357143 1.780542
```

E um terceiro, ainda com valor de ttl de 100:

```
Estatisticas
        Total de mensagens de flooding vistas: 14
        Total de mensagens de random walk vistas: 14
        Total de mensagens de busca em profundidade vistas: 14
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.428571 0.755929
        Media e desvio padrao de saltos ate encontrar destino por random walk: 13.071429 11.076408
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 6.000000 1.839732
```

Podemos ver, baseado nesses resultados, que a média da quantidade de saltos para encontrar a mensagem no random walk foi a mais alta, o que faz sentido, já que o caminho que a mensagem faz na topologia é totalmente aleatório, ainda mais usando uma topologia como a de três triângulos. Outro fator a ser observado é o desvio padrão do random walk, que é muito mais alto que o desvio padrão dos outros métodos de busca. Isso também é totalmente esperado, já que em algumas buscas a mensagem pode chegar rapidamente ao nó que tem a chave, enquanto que em outras ela pode ficar vagando pela topologia e levar vários saltos até chegar no nó com a chave. Por outro lado, o desvio padrão do flooding é bem baixo, em ambos os casos, perto de 1. Isso porque, o flooding tem um comportamento menos aleatório, já que a mensagem, depois de passar por um triângulo, não volta mais.

A busca em profundidade, por outro lado, não teve um desvio padrão tão alto, mas foi bem mais alto que o do flooding. Foi observado, em outros testes, analisando as saídas dos terminais, que ao se realizar a busca em profundidade começando um nó que está em uma extremidade da topologia até uma chave que está em um nó da outra extremidade, por vezes a mensagem fez vários ciclos ao passar pelos triângulos, passando por todos

nós ou quase todos antes de chegar no nó com a chave, mas em outras vezes ela fez um caminho mais direto, o que explica o desvio padrão mais acentuado.

Porém, realizando apenas 14 buscas para cada tipo de busca, pode-se observar que os valores do random walk variaram bastante entre os testes, por isso foi feito outro teste, ainda com ttl valendo 100, mas dessa vez a função testesAutomatizados() foi chamada dez vezes, e como à cada chamada ela realiza 14 buscas por tipo de busca, foram realizadas 140 buscas para cada tipo de busca.

```
Estatisticas
        Total de mensagens de flooding vistas: 140
        Total de mensagens de random walk vistas: 140
        Total de mensagens de busca em profundidade vistas: 140
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.608392 0.927172
        Media e desvio padrao de saltos ate encontrar destino por random walk: 13.535714 11.218183
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 5.357143 1.734423
```

Com esse teste, que conta com muito mais buscas, pode-se observar que o comportamento do flooding, random walk, e da busca em profundidade se mantém de acordo com o observado nos outros testes, mas a média do random walk ficou mais próxima do valor 13.5.

Outro teste foi feito, com 140 buscas semelhante ao teste de cima, mas dessa vez com ttl de 20:

```
Estatisticas
        Total de mensagens de flooding vistas: 140
        Total de mensagens de random walk vistas: 140
        Total de mensagens de busca em profundidade vistas: 140
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.554054 0.843412
        Media e desvio padrao de saltos ate encontrar destino por random walk: 7.684685 4.540689
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 5.464286 1.672506
```

Pode-se observar que o comportamento do flooding e da busca em profundidade se manteve o mesmo, mas a média do random walk foi bem menor. Isso porque, o ttl menor limitou o random walk quando a mensagem ficou muito tempo vagando pela topologia.

Mais um teste foi feito, com 140 buscas para cada tipo de busca, mas agora com ttl de 50:

```
Estatisticas
        Total de mensagens de flooding vistas: 140
        Total de mensagens de random walk vistas: 140
        Total de mensagens de busca em profundidade vistas: 140
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.538462 0.853899
        Media e desvio padrao de saltos ate encontrar destino por random walk: 11.116788 8.244486
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 5.264286 1.548314
```

Como esperado, o flooding e a busca em profundidade mantiveram o mesmo comportamento, mas a média do random walk ainda continuou sendo freiada pelo ttl.

Subindo o ttl para 70:

```
Estatisticas
        Total de mensagens de flooding vistas: 140
        Total de mensagens de random walk vistas: 140
        Total de mensagens de busca em profundidade vistas: 140
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.578231 0.875270
        Media e desvio padrao de saltos ate encontrar destino por random walk: 13.654676 11.924711
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 5.442857 1.485040
```

Pode-se observar que a partir de 70, ou um valor um pouco mais baixo, o ttl para de frear a média do random walk, ou seja, dificilmente a partir desse valor a mensagem vai ficar vagando pela topologia sem encontrar a mensagem até o ttl chegar a zero.

Agora subindo o ttl para 150:

```
Estatisticas
        Total de mensagens de flooding vistas: 140
        Total de mensagens de random walk vistas: 140
        Total de mensagens de busca em profundidade vistas: 140
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.510638 0.833384
        Media e desvio padrao de saltos ate encontrar destino por random walk: 12.621429 11.356529
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 5.235714 1.603103
```

Como esperado, o comportamento das 3 buscas se manteve parecido aos obtidos com ttl valendo 70 e 100. Ou seja, existe um limiar a partir do qual o ttl para de influenciar nas estatísticas obtidas.

Essa observação nos levou a testar o que acontece com ttl valendo 10:

```
Estatisticas
        Total de mensagens de flooding vistas: 140
        Total de mensagens de random walk vistas: 140
        Total de mensagens de busca em profundidade vistas: 140
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.551724 0.857411
        Media e desvio padrao de saltos ate encontrar destino por random walk: 5.835294 2.458450
        Media e desvio padrao de saltos ate encontrar destino por busca em profundidade: 5.000000 1.443137
```

Que nos mostra que com ttl muito baixo as estatísticas para as 3 buscas começam a ficar cada vez mais parecidas entre si.

3 Análise de Dados

Para esta parte, utilizamos uma topologia circular, como um chord, onde em cada computador estavam cinco nós. Um dos computadores executava Windows 11 e o outro um Manjaro Linux. O anexo [34] ilustra como está desenhada a topologia. Fizemos as buscas com testes automatizados, os mesmos usados na última demonstração da parte 2. Ao todo, fizemos duas rodadas de teste, cada rodada com três testes, cada teste executava 14 buscas de cada tipo, isto é, cada teste roda 42 buscas. Os resultados obtidos estão apresentados nos anexos [35], [36], [37] e [38].

Pelos valores apresentados, é possível notar uma consistência na quantidade de mensagens vista entre os nós para cada rodada executada. A média e o desvio-padrão se mantém estáveis entre nas duas rodadas e apresentam valores semelhantes ao esperado para a topologia utilizada. A menor quantidade de saltos necessários, sem passar duas ou mais vezes pelo mesmo nó, para entre 5 e 9 é quatro e a máxima, utilizando o mesmo critério, é seis.

4 Operando com Outra Dupla – opcional

5 Anexos

[1] Demonstração da operação HELLO.

```
Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
1

Escolha o vizinho:
Há 2 vizinhos na tabela
        [0] localhost 5002
        [1] localhost 5003
0
Encaminhando mensagem "localhost:5001 100 1 HELLO" para localhost:5002
        Envio feito com sucesso: localhost:5001 3 1

Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
1

Escolha o vizinho:
Há 2 vizinhos na tabela
        [0] localhost 5002
        [1] localhost 5003
1
Encaminhando mensagem "localhost:5001 100 1 HELLO" para localhost:5003
        Envio feito com sucesso: localhost:5001 4 1

Escolha o comando
```

[2]: Imagem dos nós na tabela de vizinhos do nó 1 antes do envio da mensagem de busca e o envio da mensagem de saída para o nó 2.

```
0
Há 1 vizinhos na tabela:
        [0] localhost 5002

Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
9
Encaminhando mensagem "localhost:500121BYE" para localhost:5002
        Envio feito com sucesso: "localhost:500121BYE"
```

[3]: Primeiro, selecionamos a opção 3 do menu, depois digitamos a chave a ser buscada e a mensagem é enviada. Ocorre um erro, devido à condição de corrida, em que a mensagem de chave encontrada aparece antes da mensagem de envio feito com sucesso. Esta operação foi realizada no nó 1.

```
3
Digite a chave a ser buscada
zenith_peak
Encaminhando mensagem "localhost:5001 2 100 SEARCH RW 1" para localhost:5002
        Valor encontrado!
                Chave: zenith_peak valor: 27
        Envio feito com sucesso: localhost:5001 2 1
```

[4]: Print do nó 2, após receber o pedido de busca do nó 1.

```
Mensagem recebida localhost:5001 2 100 search RW 5001 zenith_peak  1
Encaminhando mensagem para localhost:5003
```

[5]: Print do nó 3, após receber o pedido de busca do nó 2. A mensagem recebida pelo nó 3 mostra que no segundo salto o TTL da mensagem foi decrementado em uma unidade.

```
Mensagem recebida localhost:5001 2 99 search RW 5002 zenith_peak  2
        Chave encontrada!
```

[6]: Print do nó 1 inciando a busca pela chave “pine_cone”

```
Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
3
Digite a chave a ser buscada
pine_cone
Encaminhando mensagem "localhost:5001 3 100 SEARCH RW 1" para localhost:5004
```

[7]: Nó 4 recebe a mensagem do 1 e devolve ela

```
Mensagem recebida localhost:5001 3 100 search RW 5001 pine_cone  1
Encaminhando mensagem para localhost:5001
```

[8]: Nó 1 recebe a mensagem do 4 e também a devolve para o nó 4

```
Mensagem recebida localhost:5001 3 99 search RW 5004 pine_cone  2
Encaminhando mensagem para localhost:5004
```

[9]: Nó 4 recebe, pela segunda vez, a mensagem do nó 1 e, agora, sorteia o nó 7 para encaminhar o pedido de busca da chave “pine_cone”

```
Mensagem recebida localhost:5001 3 98 search RW 5001 pine_cone  3
Encaminhando mensagem para localhost:5007
```

[10]: O nó 7 recebe a mensagem do nó 4 e encaminha para o nó 8

```
Mensagem recebida localhost:5001 3 97 search RW 5004 pine_cone  4
Encaminhando mensagem para localhost:5008
```

[11]: O nó 8 recebe a mensagem do nó 7 e encaminha para o nó 9

```
Mensagem recebida localhost:5001 3 96 search RW 5007 pine_cone  5
Encaminhando mensagem para localhost:5009
```

[12]: O nó 9 recebe a mensagem do nó 8 e encaminha para o nó 6

```
Mensagem recebida localhost:5001 3 95 search RW 5008 pine_cone  6
Encaminhando mensagem para localhost:5006
```

[13]: O nó 6 recebe a mensagem do nó 9 e encontra a chave na tabela local e, então, responde para o nó 1

```
Mensagem recebida localhost:5001 3 94 search RW 5009 pine_cone  7
        Chave encontrada!
```

[14]: O nó 1 imprime que a chave “pine_cone” foi encontrada e que seu valor é 8.

```
        Valor encontrado!
                Chave: pine_cone valor: 8
```

[15]: O nó 5 incia pela chave “hidden_valley” e encaminha, aleatoriamente, para o nó 2

```
4
Digite a chave a ser buscada
hidden_valley
Encaminhando mensagem "localhost:5005 2 100 SEARCH BP 1" para localhost:5002
```

[16]: O nó 2 recebe a mensagem de 5 e encaminha para o nó 1

```
Mensagem recebida localhost:5005 2 100 search BP 5005 hidden_valley  1
Mensagem nova!
DFS Message com localhost:5005 2 99 search BP 5002 hidden_valley  2 adicionada
Encaminhando mensagem "localhost:5005 4 100 search BP 2" para localhost:5001
        Envio feito com sucesso: localhost:5005 4 1
```

[17]: O nó 1 recebe a mensagem de 2 e encaminha para o nó 3

```
Mensagem recebida localhost:5005 2 99 search BP 5002 hidden_valley  2
Mensagem nova!
DFS Message com localhost:5005 2 98 search BP 5001 hidden_valley  3 adicionada
Encaminhando mensagem "localhost:5005 3 100 search BP 3" para localhost:5003
        Envio feito com sucesso: localhost:5005 3 1
```

[18]: O nó 3 recebe a mensagem de 1 verifica que a chave buscada está em sua tabela local, logo, ele imprime que a chave foi encontrada e comunica ao nó 5.

```
Mensagem recebida localhost:5005 2 98 search BP 5001 hidden_valley  3
        Chave encontrada!
```

[19]: O nó 5 imprime na tela dizendo que a chave foi encontrada e que seu valor é 31.

```
        Valor encontrado!
                Chave: hidden_valley valor: 31
```

[20]: O nó 1 inicia a busca e envia, aleatoriamente, para o nó 3.

```
4
Digite a chave a ser buscada
lavender_field
Encaminhando mensagem "localhost:5001 3 100 SEARCH BP 1" para localhost:5003
        Valor encontrado!
                Chave: lavender_field valor: 45
        Envio feito com sucesso: localhost:5001 3 1
```

[21]: O nó 3 recebe a mensagem do nó 1 e encontra a chave buscada na sua tabela local.

```
Mensagem recebida localhost:5001 3 100 search BP 5001 lavender_field  1
        Chave encontrada!
```

[22]: O nó 1 recebe a mensagem de 3 informando que a chave foi encontrada, e que seu valor é 45

```
        Valor encontrado!
                Chave: lavender_field valor: 45
```

[23]: Os vizinhos da tabela local do nó 2, antes do recebimento da mensagem de saída do nó 1.

```
0
Há 2 vizinhos na tabela:
        [0] localhost 5001
        [1] localhost 5003
```

[24]: Mensagem de saída do nó 1 recebida no nó 2.

```
Mensagem recebida: localhost 5001 2 1 BYE
```

[25]: A tabela local de vizinhos do nó 2, após o recebimento da mensagem de saída do nó 1.

```
0
Há 1 vizinhos na tabela:
        [0] localhost 5003
```

[26]: Imagem do início da busca, usando flooding e a partir do nó 9, pela chave “Rami_Malek”, localizada no nó 7. A imagem também ilustra a comunicação do nó 7 ao nó 9 de que a chave foi encontrada e seu valor é 3.

```
2
Digite a chave a ser buscada
Rami_Malek

Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
Encaminhando mensagem "localhost:5009 4 100 SEARCH FL 1" para localhost:5008
Encaminhando mensagem "localhost:5009 4 100 SEARCH FL 1" para localhost:5010
        Envio feito com sucesso: "localhost:5009 4 100 SEARCH FL Rami_Malek 1"
        Envio feito com sucesso: "localhost:5009 4 100 SEARCH FL Rami_Malek 1"
Mensagem recebida: "localhost:5009 3 99 search FL 5008 Rami_Malek 2"
Mensagem recebida: "localhost:5009 3 99 search FL 5010 Rami_Malek 2"
Mensagem recebida: "localhost:5009 3 99 search FL 5010 Rami_Malek 2"
Mensagem recebida: "localhost:5009 3 99 search FL 5008 Rami_Malek 2"
        Valor encontrado!
                Chave: Rami_Malek valor: 3
```

[27]: Quando a mensagem de busca é recebida pelo nó 7, ele encontra a chave na sua tabela local e imprime na tela a seguinte mensagem.

```
Mensagem recebida: "localhost:5009 3 97 search FL 5005 Rami_Malek 4"
        Chave encontrada!
```

[28]: Início da busca pela chava “zenith_peak” no nó 1, utilizando a topologia linha 3. A mensagem é encaminhada para o nó 2 e nenhuma mensagem é recebida do nó 2. Nesta topologia, o único vizinho é de 1 é 2. Na imagem ainda aparece a resposta do nó 3 para 1, com o valor da chave buscada.

```
2
Digite a chave a ser buscada
zenith_peak

Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
Encaminhando mensagem "127.0.0.1:5001 2 100 SEARCH FL 1" para 127.0.0.1:5002
        Envio feito com sucesso: "127.0.0.1:5001 2 100 SEARCH FL zenith_peak 1"
        Valor encontrado!
                Chave: zenith_peak valor: 27
```

[29]: Mensagem de busca recebida no nó 2 enviada por 1. A mensagem é repassada ao outro vizinho de 2, o nó 3.

```
Mensagem recebida: "127.0.0.1:5001 1 100 search FL 5001 zenith_peak 1"
Encaminhando mensagem "127.0.0.1:5001 3 100 search FL 2" para 127.0.0.1:5003
        Envio feito com sucesso: "127.0.0.1:5001 3 99 search FL zenith_peak 2"
```

[30]: Mensagem recebida por 3 do nó 2. O nó verifica em sua tabela local que a chave buscada está lá e comunica ao nó original o valor da chave.

```
Mensagem recebida: "127.0.0.1:5001 1 99 search FL 5002 zenith_peak 2"
        Chave encontrada!
```

[31]: Imagem do nó 9, que inicia a busca. A chave procurada está no nó 7, que não faz parte do ciclo.

```
2
Digite a chave a ser buscada
Rami_Malek

Escolha o comando
[0] Listar vizinhos
[1] Hello
[2] SEARCH (flooding)
[3] SEARCH (random walk)
[4] SEARCH (busca em profundidade)
[5] Estatísticas
[6] Alterar valor padrão de TTL
[9] Sair
Encaminhando mensagem "127.0.0.1:5009 2 100 SEARCH FL 1" para 127.0.0.1:5010
Encaminhando mensagem "127.0.0.1:5009 2 100 SEARCH FL 1" para 127.0.0.1:5008
        Envio feito com sucesso: "127.0.0.1:5009 2 100 SEARCH FL Rami_Malek 1"
        Envio feito com sucesso: "127.0.0.1:5009 2 100 SEARCH FL Rami_Malek 1"
        Valor encontrado!
                Chave: Rami_Malek valor: 3
```

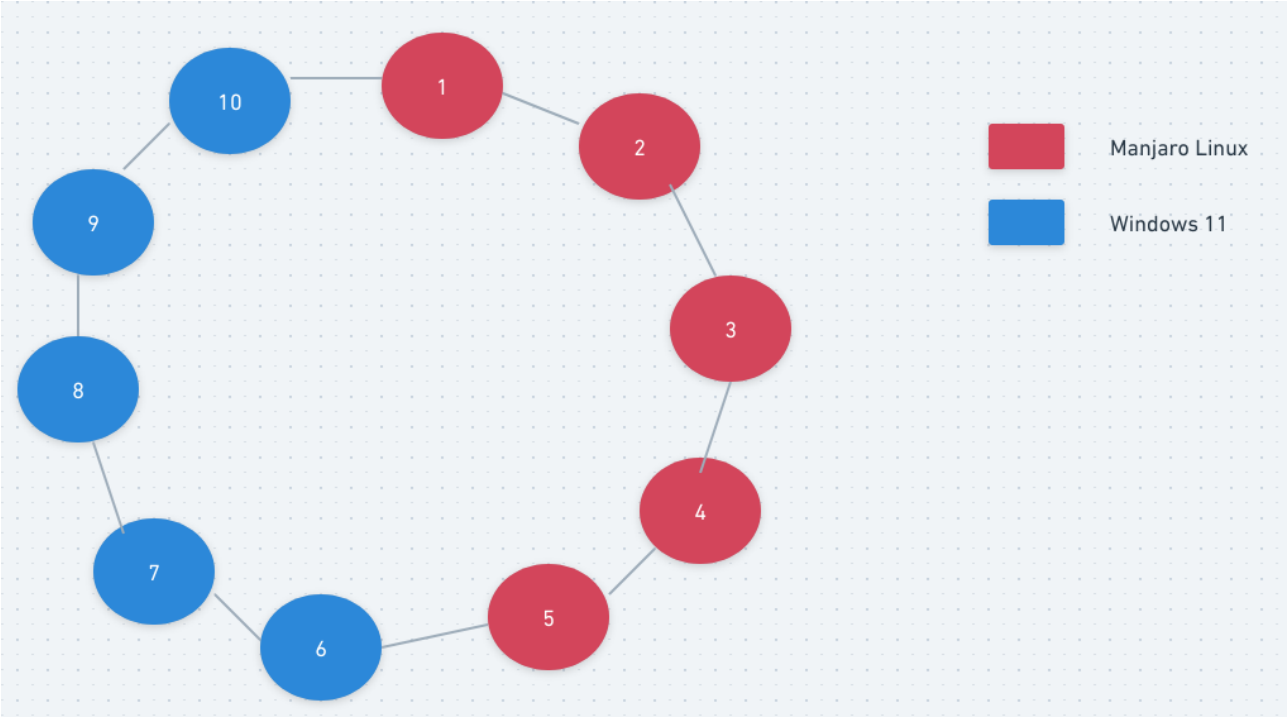
[32]: Imagem do nó 8, que recebe a mensagem diretamente dó nó de origem e do nó 10. É possível ver que a mesma mensagem duas vezes e descarta da segunda vez

```
Mensagem recebida: "127.0.0.1:5009 1 100 search FL 5009 Rami_Malek 1"
Encaminhando mensagem "127.0.0.1:5009 3 100 search FL 2" para 127.0.0.1:5001
Encaminhando mensagem "127.0.0.1:5009 3 100 search FL 2" para 127.0.0.1:5010
Mensagem recebida: "127.0.0.1:5009 1 99 search FL 5010 Rami_Malek 2"
Flooding: Mensagem repetida!
        Envio feito com sucesso: "127.0.0.1:5009 3 99 search FL Rami_Malek 2"
        Envio feito com sucesso: "127.0.0.1:5009 3 99 search FL Rami_Malek 2"
```

[33]:Imagem do nó 10, que recebe a mensagem diretamente dó nó de origem e do nó 8. É possível ver que a mesma mensagem duas vezes e descarta da segunda vez.

```
Mensagem recebida: "127.0.0.1:5009 1 100 search FL 5009 Rami_Malek 1"
Encaminhando mensagem "127.0.0.1:5009 2 100 search FL 2" para 127.0.0.1:5008
Mensagem recebida: "127.0.0.1:5009 1 99 search FL 5008 Rami_Malek 2"
Flooding: Mensagem repetida!
        Envio feito com sucesso: "127.0.0.1:5009 2 99 search FL Rami_Malek 2"
```

[34]: Arquitetura do cíclica, utilizada para análise de dados.



[35]: Windows 11. Rodada 1. Nó 9.

```
5
Estatisticas
        Total de mensagens de flooding vistas: 64
        Total de mensagens de random walk vistas: 48
        Total de mensagens de busca em profundidade vistas: 48
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.900000 1.372665
        Media e desvio padrao de saltos ate encontrar destino por random walk: 4.600000 1.729009
        Media e desvio padrao de saltos ate encontrar destino por busca em profuncidade: 4.700000 1.750188
```

[36]: Manjaro Linux. Rodada 1. Nó 5.

```
5
Estatisticas
        Total de mensagens de flooding vistas: 64
        Total de mensagens de random walk vistas: 51
        Total de mensagens de busca em profundidade vistas: 49
        Media e desvio padrao de saltos ate encontrar destino por flooding: 2.500000 1.142080
        Media e desvio padrao de saltos ate encontrar destino por random walk: 5.583333 2.733316
        Media e desvio padrao de saltos ate encontrar destino por busca em profuncidade: 5.500000 2.750494
```

[37]: Windows 11. Rodada 2. Nó 9.

```
5
Estatisticas
        Total de mensagens de flooding vistas: 77
        Total de mensagens de random walk vistas: 61
        Total de mensagens de busca em profundidade vistas: 61
        Media e desvio padrao de saltos ate encontrar destino por flooding: 3.866667 1.332183
        Media e desvio padrao de saltos ate encontrar destino por random walk: 5.066667 1.760355
        Media e desvio padrao de saltos ate encontrar destino por busca em profuncidade: 5.133333 1.756433
```

[38]: Manjaro Linux. Rodada 2. Nó 5.

```
5
Estatisticas
        Total de mensagens de flooding vistas: 78
        Total de mensagens de random walk vistas: 61
        Total de mensagens de busca em profundidade vistas: 59
        Media e desvio padrao de saltos ate encontrar destino por flooding: 2.500000 1.142080
        Media e desvio padrao de saltos ate encontrar destino por random walk: 4.500000 2.750494
        Media e desvio padrao de saltos ate encontrar destino por busca em profuncidade: 5.250000 2.785834
```