Redes de computadores EP3

Nome: Ramon Neres Teixeira Jardim

RA: 21202410009

Link do vídeo: https://youtu.be/ddnsSVc4h-U

Link para o repositório:
https://github.com/RamonJardim/Redes-de-computadores-pos/tree/main/EP3

Instruções para execução estão no README.md.

Explicação das funcionalidades e threads

A classe principal (*Main.java*) exibe o menu para interação com o usuário e executa os roteadores na linha 32 com a função *run* que recebe uma matriz de adjacência e faz os tratamentos necessários para execução. Caso seja a opção do usuário, uma matriz de adjacência aleatória será gerada na linha 99 com a função *generateAdjacencyTable*.

Na linha 59, dentro da função *run*, os roteadores (classe Router que será descrita a seguir) são instanciados e iniciador e em seguida na linha 64 é feito um *join* para aguardar o fim da execução de todos os roteadores. Na sequência é exibida a matriz de adjacência inicial e a matriz de adjacência final (no caso, os vetores de distâncias calculados por cada roteador). O total de pacotes enviados e de iterações realizadas também são obtidos de cada roteador e exibidos na tela.

A classe *Router* (*Router.java*), é uma *thread* que representa um roteador. As propriedades da classe são definidas entre as linhas 7 e 27 e são: *DELAY* que é o tempo que o roteador aguarda entre envios de pacotes, *TIMEOUT* que é o tempo máximo de espera por um pacote (finaliza execução após este tempo sem receber pacotes), *distanceVector* que é o vetor de distâncias, *routingTable* que é a tabela de roteamento, *myId*, que é o identificador do roteador, *neighbors*, que é a lista de vizinhos daquele roteador, *sender* que é uma *thread* para enviar informações, *receiver* que é uma *thread* para receber informações, *spy* que é uma thread para derrubar um roteador ou alterar um link, *channel* é o canal de comunicação, *finished* flag que indica se o roteador foi derrubado ou se terminou execução, *vectorChanged* flag que indica se o vetor de distâncias foi alterado, *sentPacketsCount* é o contador de pacotes enviados, *updateCount* é contador de atualizações no vetor de distâncias (iterações), *routerPadding* e *edgePadding* são os tamanhos do padding para o id do roteador e peso da conexão (para alinhamento na impressão da matriz de adjacência),

linkToChange é o link a ser alterado caso seja optado pelo usuário, *newWeight* é o novo peso do link (caso algum seja alterado) e routerToDrop é o roteador a ser derrubado (caso seja optado).

A tabela de roteamento é inicialmente populada pela funcção *populateInitialRoutingTable* na linha 67 e a lista de vizinhos pela função *getNeighbors* da linha 92. Ambas são chamadas dentro do construtor (linha 45) que também instancia *Sender*, *Receiver* e *Spy*. A *thread* Sender na linha 165 envia o vetor de distâncias para os vizinhos caso a flag *vectorChanged* esteja setada como *true*. O envio é feito utilizando a função *sendVectorToNeighbors* (linha 182). Nesta função, é criada uma instância da classe *DatagramInfo* (linha 15 do arquivo Channel.java) que contém *id* do roteador atual e o vetor de distâncias e ela é enviada pelo canal. Como a flag *vectorChanged* inicia em *true*, o primeiro vetor de distância já é enviado logo de início. Sempre que há um envio a flag é marcada como *false*.

O *Receiver* (linha 195) fica no aguardo por novas mensagens e assim que recebe uma mensagem utiliza a função *calculateNewDistanceVector* (linha 102) para calcular o novo vetor de distâncias. Caso o novo vetor calculado seja diferente do atual, a flag *vectorChanged* é marcada como *true* e a *thread Sender* fará o envio deste novo vetor. Caso haja um *timeout* no recebimento de um novo pacote, o programa é finalizado utilizando a flag *finished*, que sinaliza para todas as thread que o programa terminou. Na linha 199, há um aguardo de 2 segundos no receiver para corrigir um bug que especificamente no linux fazia com que o primeiro roteador instanciado não recebesse as mensagens.

A classe *Spy* (linha 217) aguarda metade do tempo de *timeout* e executa as ações de derrubar roteador ou mudar um link, dependendo da escolha do usuário no início do programa. Caso a opção seja derrubar um roteador e o *id* do roteador seja o mesmo a ser derrubado, a flag *finished* é marcada para *true* e o vetor de distância é zerado (indicando que nenhum roteador é acessível) e com isso o roteador morre sem replicar seu novo vetor de distâncias zerado. Caso o roteador a ser derrubado não seja o mesmo, a função *routerDown* (linha 129) é executada, zerando também a posição do roteador que caiu no vetor de distâncias do roteador atual (apenas se forem vizinhos).

Caso a opção do usuário seja alterar um link, a função *modifyLink* (linha 137) é executada (caso o atual roteador faça parte do link alterado) alterando o valor do link no vetor de distâncias e sentando a flag *vectorChanged*.

Por fim, a classe *Channel* (Channel.java) foi baseada no componente canal do EP1 e foi simplificada para incluir apenas as funcionalidades necessárias para o EP3, removendo o envio e recebimento de ACKs, por exemplo. A constante *eliminateProbability* na linha 44 representa a probabilidade de uma mensagem enviada por este canal ser eliminada, a propriedade foi removida do arquivo *config.json* para simplificar o código já que apenas eliminações são relevantes agora.

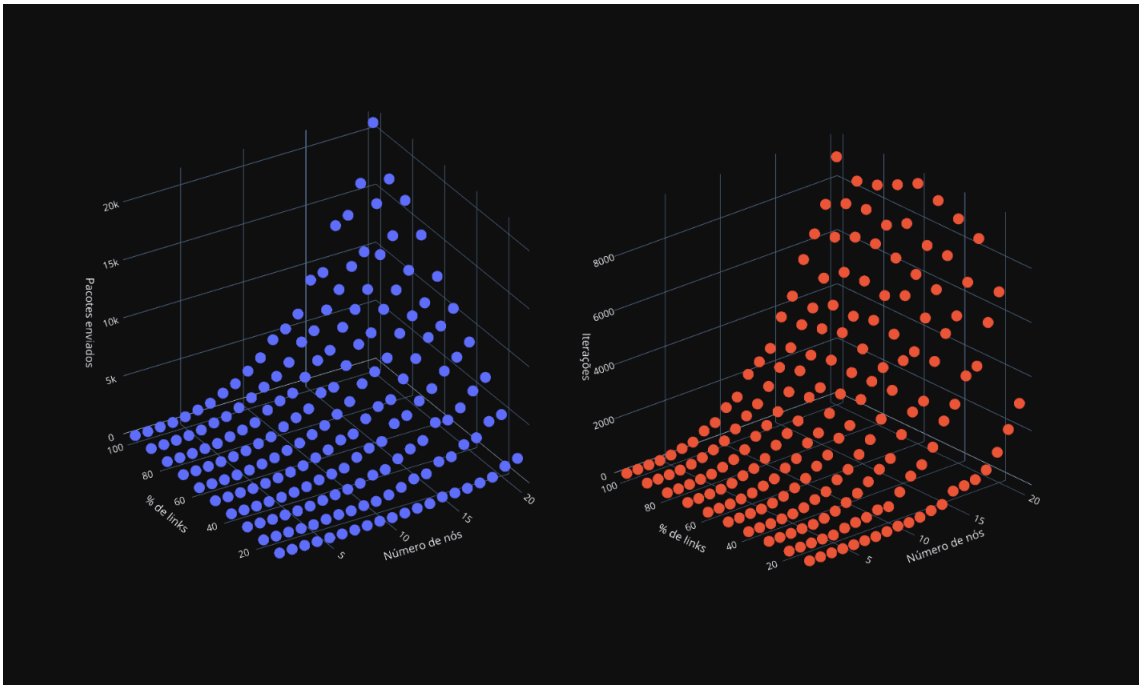
Esta classe oferece as funções *send* (linha 62) e *receive* (linha 75) para os roteadores. A função *send* recebe um *DatagramInfo* e o envia para o roteador com id recebido (somando 10000 ao *id* para usar como porta). O envio é feito serializando o objeto *DatagramInfo* em um array de bytes. Similarmente, a função *send* recebe um array de bytes que é desserializado em um objeto da classe *DatagramInfo* e o retorna para o roteador.

Os logs da classe *Channel* foram suprimidos e foi disponibilizada a função *sendCount* (linha 156) para que seja feita a consolidação do número de mensagens trocadas.

Gráficos

Para fazer as médias do número de mensagens trocadas e número de iterações, foi feita uma alteração no código para que de forma autônoma fossem feitas execuções variando entre 1 e 20 roteadores e variando a porcentagem aproximada de conexões entre 100% e 10%. Para cada combinação foram feitas 10 execuções e as médias foram escritas em um arquivo csv. Para evitar a criação de um número muito grande de gráficos a partir dos dados, dois gráficos tridimensionais foram criados para exibir todos os dados de uma vez.

Os dois gráficos são representados na figura a seguir:



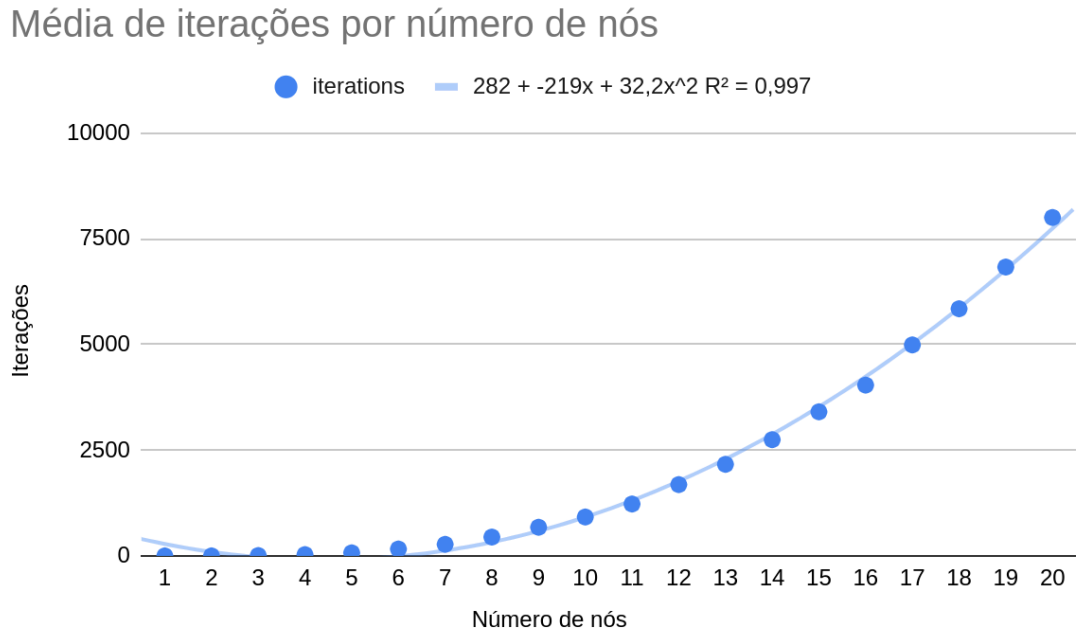
O gráfico azul representa o número de pacotes enviados por % de links por número de nós, já o gráfico laranja representa o número de iterações por % de links por número de nós.

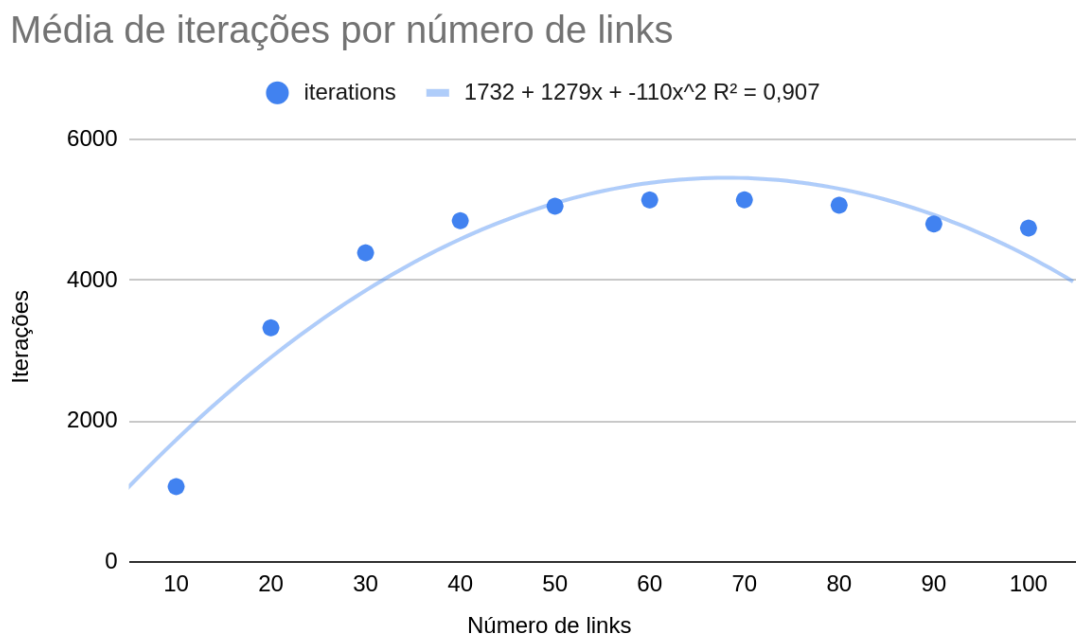
É possível notar um aumento quadrático no número de transmissões com o aumento do número de nós, o que é compatível com a complexidade do algoritmo de Bellman-Ford que é O(V*E). Nos dois gráficos a seguir vemos uma representação agregada dos pacotes enviados por número de nós e links respectivamente.

Em ambos observa-se um crescimento quadrático do número de pacotes enviados, como já foi possível perceber pelo gráfico tridimensional. O aumento quadrático com o aumento de nós se deve ao fato de que mais iterações serão necessárias para finalizar o algoritmo (como veremos a seguir), já o aumento

quadrático com o aumento de links se deve ao fato de que cada roteador tem um número maior de vizinhos, o que causa um maior número de envios.

Nos dois gráficos a seguir vemos a média de iterações por número de nós e por número de links respectivamente.





Também é possível notar um crescimento quadrático de iterações com o aumento do número de nós, pois existem mais combinações de caminhos, portanto mais atualizações acontecerão para cada roteador.

O comportamento da curva da média de iterações por número de links foge do padrão visto anteriormente, há um aumento entre 0% e 50% de links que se estabiliza entre 50% e 100%. Este resultado também é esperado, uma vez que muitos nós já estão conectados há uma maior chance dos caminhos ideais serem recebidos nas primeira iterações.