OBJETIVO: Mostrar a execução e funcionamento do algoritmo de rede "*Flooding*" através da implementação de um Simulador por Eventos Discretos.

QUESTÃO ÚNICA

A Figura 1 representa uma rede sem fio formada por 10 nós com raio de comunicação de, por exemplo, 38m. O algoritmo *Flooding* é um algoritmo distribuído usado para propagar uma informação de um nó para toda a rede. Neste algoritmo um nó inicia um *flooding* (nó 0 da Figura 2) mandando um pacote via *broadcast*. Um pacote *broadcast* é recebido por todos os nós dentro do raio de comunicação do nó que o enviou. No caso da Figura 2, o pacote *broadcast* enviado pelo nó 0 será recebido pelos nós 1 e 2.

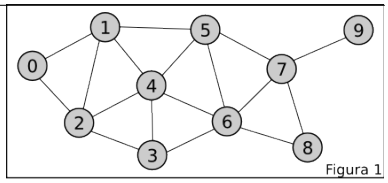

Figura 1

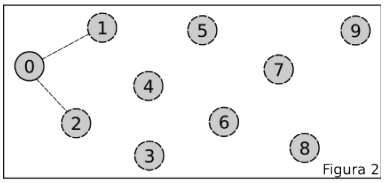

Figura 2

Um nó, ao receber um pacote, irá verificar se o mesmo já repassou este pacote antes. Caso não, este irá repassar o pacote também via *broadcast* (que será recebido por todos os seus vizinhos). No caso da Figura 3, o nó 1, que recebeu pacote vindo do nó zero, irá repassar o pacote via *broadcast*, que será recebido pelos seus vizinhos (nós 0, 2, 4 e 5). O nó 2 (Figura 4) também repassará o pacote recebido de zero a seus vizinhos (nós 0, 1, 4 e 3).

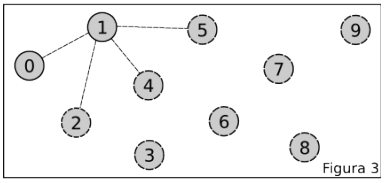

Figura 3

Caso um nó receba um pacote que o mesmo já havia repassado antes, ele simplesmente não fará nada (irá descartar o pacote recebido). Isso acontecerá, por exemplo, com o nó zero ao receber (de volta) os pacotes dos nós 1 e 2.

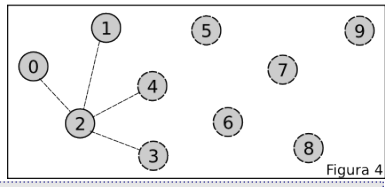

Figura 4

Este processo irá se repetindo, até que todos os nós da rede já tenham repassado o pacote uma vez. Ao final do algoritmo, a "informação" enviada pelo nó 0 terá sido recebido por todos os nós da rede.

Só por curiosidade, em termos de "Algoritmos Distribuídos", o algoritmo acima é representado como:

```
Variables:
    reached_i = false.          (Variável presente em todos os nós.)

Input:
    msg_i = nil.
Action:
    if n_i == N_0 then          (Nó zero envia primeiro pacote.)
            begin
                    reached_i := true;
                    Send inf to all n_j ∈ Neig_i
            end.

Input:
    msg_i = inf.                (Pacote recebido por um nó.)
Action:
    if not reached_i then
            begin
                    reached_i := true;
                    Send inf to all n_j ∈ Neig_i     (Nó envia pacote adiante.)
            end.
```

Este mesmo algoritmo será executado por todos os nós da rede. Referência: *An Introduction to Distributed Algorithms. Barbosa C. Valmir. 1996.*

Neste trabalho, assim como na MATRIX, você implementará um Simulador por Eventos Discretos. Entretanto, seu simulador será usado para simular o comportamento do algoritmo *Flooding*. Um simulador por eventos discretos é bem simples. Basicamente, os eventos do sistema são modelados através de uma lista ordenada de eventos (que vocês implementaram no TP1). Um evento, ao ser executado, poderá gerar outros eventos futuros, que serão inseridos ordenadamente na lista de eventos de acordo com o seu tempo de ocorrência. Para iniciar o simulador, você precisa adicionar um primeiro evento na lista de eventos. A partir daí, um *loop* se inicia em que o simulador (1) pega o primeiro elemento na lista de eventos, (2) remove-o da lista e (3) executa o evento (que poderá inserir outros eventos na lista de eventos). Este *loop* de três passos será executado até que a lista de eventos seja nula (ou seja, não existe mais eventos no sistema).

Assim como no TP3 (grafos), as informações da rede (número de nós e raio de comunicação) e dos nós serão todos passados por um arquivo de entrada (argv[1]). A primeira linha deste arquivo terá o seguinte formato:

`<Número de Nós>\t<Raio de Comunicação>\n`

onde `<Número de Nós>` é a quantidade de nós na rede e `<Raio de Comunicação>` é o raio de comunicação entre os nós, ou seja, caso a distância física entre dois nós seja menor que este raio de comunicação, eles podem se comunicar e, portanto, existirá uma aresta entre eles.

As linhas seguintes do arquivo de entrada, terão o seguinte formato:

`<Id>\t<Posição X>\t<Posição Y>\n`

onde `<Id>` é o `id` do nó (número da linha) e `<Posição X>` e `<Posição Y>` a posição física do nó. Para cada linha, adicione o nó atual em um grafo que representará a rede (como no TP3). Depois de inserir todos os nós, execute uma função para popular

(atualizar) a lista de vizinhos de todos os nós. Esta função, para cada nó (nó "*i*"), irá compará-lo com todos os nós da rede (nó "*j*") e, caso a distância entre *i* e *j* seja menor que o raio de comunicação, adicione *i* à lista de vizinhos de *j* (como no TP3).

Após ler todo o arquivo de entrada e atualizar a lista de vizinhos do grafo, crie um novo evento em que nó "alvo" será o nó zero, o "tipo" do evento será 1 (pacote recebido) e o "tempo" do evento será 1.0. Adicione este evento na lista de eventos. Inicie a simulação (i.e., chame a função para executar a simulação).

A função para executar a simulação (`void simulacao_iniciar()`) terá um *while*. Este *while* será executado enquanto a lista de eventos for diferente de nulo. Dentro dele, você executará os três passos mencionados anteriormente. (1) Salve um ponteiro para o primeiro evento da lista de eventos. (2) Faça a lista de eventos apontar para o próximo evento (isso irá "remover" o evento atual a lista). Imprima a mensagem: "[%3.6f] Nó %d recebeu pacote.\n", onde o primeiro valor é o tempo do evento salvo e o segundo é o "alvo" do evento. (3) Por fim, execute o evento, que funcionará conforme descrito a seguir.

Para executar o evento, acesse as informações do nó "alvo" do evento atual (usando o grafo implementado no TP3). A diferença é que neste caso, cada nó (`struct no_t`) terá um campo extra "`bool pacote_enviado`", indicando se o nó já enviou o pacote do *flooding* ou não (inicialize todos os nós com `pacote_enviado` como sendo `false`). Em seguida, verifique se o nó atual já enviou o pacote (`pacote_enviado`). Caso negativo, ele o fará agora.

Para simular o envio do pacote via *broadcast*, você irá simplesmente adicionar os eventos de recebimento do pacote atual para cada vizinho do nó atual. Desta forma, para cada vizinho do nó atual, imprima a mensagem `"\t--> Repassando pacote para o nó %d ...\n"`, onde `%d` é o `id` do vizinho atual. Em seguida, crie um novo evento onde o "alvo" será o `id` do vizinho, o "tipo" do evento será também 1 (recebimento de pacote), e o "tempo" do novo evento será o tempo do evento atual mais `(0.1 + (vizinho->id * 0.01))`, onde este cálculo simula o tempo de processamento + propagação do pacote que, em geral é um número mais aleatório, mas que neste trabalho deverá ser determinístico para fins de correção automática. Adicione este evento de forma ordenada na lista de eventos (usando a função `lista_eventos_adicionar_ordenado` implementada no TP1). Como mencionado, estes passos serão executados para cada vizinho do nó atual, o que gerará possivelmente diversos outros eventos. Por fim, sete o `pacote_enviado` do nó atual para `true`, para indicar que este nó já encaminhou o pacote e que o mesmo não fará mais isso futuramente. Isso termina esta iteração do *while* e a próxima iteração será executada, em que o próximo evento do sistema será executado.

Exemplo de Entrada:

```
10      38.0
0       6.0     20.5
1       33.4    6.2
2       23.7    41.6
3       51.0    54.8
4       50.9    27.8
5       71.1    7.2
6       79.1    41.9
7       99.6    21.6
8       109.3   51.6
9       128.7   7.3
```

Exemplo de Saída:

```
[1.000000] Nó 0 recebeu pacote
    --> Repassando pacote para o nó 2 ...
    --> Repassando pacote para o nó 1 ...
[1.110000] Nó 1 recebeu pacote
    --> Repassando pacote para o nó 5 ...
    --> Repassando pacote para o nó 4 ...
    --> Repassando pacote para o nó 2 ...
    --> Repassando pacote para o nó 0 ...
[1.120000] Nó 2 recebeu pacote
    --> Repassando pacote para o nó 4 ...
    --> Repassando pacote para o nó 3 ...
    --> Repassando pacote para o nó 1 ...
    --> Repassando pacote para o nó 0 ...
[1.210000] Nó 0 recebeu pacote
[1.220000] Nó 0 recebeu pacote
[1.230000] Nó 1 recebeu pacote
[1.230000] Nó 2 recebeu pacote
[1.250000] Nó 3 recebeu pacote
    --> Repassando pacote para o nó 6 ...
    --> Repassando pacote para o nó 4 ...
    --> Repassando pacote para o nó 2 ...
[1.250000] Nó 4 recebeu pacote
    --> Repassando pacote para o nó 6 ...
    --> Repassando pacote para o nó 5 ...
    --> Repassando pacote para o nó 3 ...
    --> Repassando pacote para o nó 2 ...
    --> Repassando pacote para o nó 1 ...
[1.260000] Nó 4 recebeu pacote
[1.260000] Nó 5 recebeu pacote
    --> Repassando pacote para o nó 7 ...
    --> Repassando pacote para o nó 6 ...
    --> Repassando pacote para o nó 4 ...
    --> Repassando pacote para o nó 1 ...
[1.360000] Nó 1 recebeu pacote
[1.370000] Nó 1 recebeu pacote
[1.370000] Nó 2 recebeu pacote
[1.370000] Nó 2 recebeu pacote
[1.380000] Nó 3 recebeu pacote
[1.390000] Nó 4 recebeu pacote
[1.400000] Nó 4 recebeu pacote
[1.400000] Nó 5 recebeu pacote
[1.410000] Nó 6 recebeu pacote
    --> Repassando pacote para o nó 8 ...
    --> Repassando pacote para o nó 7 ...
    --> Repassando pacote para o nó 5 ...
    --> Repassando pacote para o nó 4 ...
    --> Repassando pacote para o nó 3 ...
[1.410000] Nó 6 recebeu pacote
[1.420000] Nó 6 recebeu pacote
[1.430000] Nó 7 recebeu pacote
    --> Repassando pacote para o nó 9 ...
    --> Repassando pacote para o nó 8 ...
    --> Repassando pacote para o nó 6 ...
    --> Repassando pacote para o nó 5 ...
[1.540000] Nó 3 recebeu pacote
[1.550000] Nó 4 recebeu pacote
[1.560000] Nó 5 recebeu pacote
[1.580000] Nó 5 recebeu pacote
[1.580000] Nó 7 recebeu pacote
[1.590000] Nó 6 recebeu pacote
[1.590000] Nó 8 recebeu pacote
    --> Repassando pacote para o nó 7 ...
    --> Repassando pacote para o nó 6 ...
[1.610000] Nó 8 recebeu pacote
[1.620000] Nó 9 recebeu pacote
    --> Repassando pacote para o nó 7 ...
[1.750000] Nó 6 recebeu pacote
[1.760000] Nó 7 recebeu pacote
[1.790000] Nó 7 recebeu pacote
```

Dicas:

Sugestão de funções a serem implementadas:

`bool lista_eventos_adicionar_ordenado(evento_t *evento, lista_eventos_t **lista);` → **Como no TP1**

`void bool lista_vizinhos_adicionar(int vizinho, lista_vizinhos_t **lista);` → **Como no TP3**

`grafo_t grafo_criar(int tam);` → **Como no TP3**

`void grafo_atualizar_vizinhos(int tam, double raio_comunicacao, grafo_t grafo)` → **Como no TP3**

`void simulacao_iniciar();`

→ Conforme explicado no texto acima.
→ Enquanto lista de eventos não for nulo
→ Pegue o primeiro evento e remova-o da lista de eventos
→ Se o nó do evento atual ainda não mandou o pacote
→ Para cada vizinho do nó atual
→ Crie um evento de recebimento do pacote para o vizinho atual
→ Adicione o evento à lista de eventos
→ Sete pacote_enviado do nó atual para true.

`int main(int argc, char **argv);`

→ Conforme explicado no texto acima.
→ Lê arquivo de entrada e gera o grafo que representará a rede
→ Atualiza lista de vizinhos dos nós do grafo
→ Cria o evento inicial (nó zero "recebendo" um pacote) e adicione-o na lista de eventos
→ Inicia a simulação

ENTREGA DO LABORATÓRIO

Envie, até 12/10/20 às 23:59, o código-fonte para horacio@icomp.ufam.edu.br com o assunto “Entrega do 4o Laboratório de LPA”.