Mini-Manual do Código

Antonio Monteiro Guedes Forte

- **Linhas**

  1-5 (Bibliotecas)
  7-53 (Estruturas)
  54-130 (Fila de Prioridades – Funções)
  131-284 (Grafo – Funções)
  285-482 (Funções para os Eventos)
  483-488 (Função Main)

- # Explicação de cada Parte do Código

Linha 14 – Estrutura Evento:

Estrutura que representa o evento da simulação, onde tem como variaveis o nome do evento, o seu tempo de atraso que ocorrerá a cada tempo do evento, a sua probabilidade, se o evento irá se repetir e os estados que os vértices serão atualizados.

Linha 26 – Estrutura Pessoa:

Estrutura que representa o estado de cada vértice do grafo, podendo ser ‘S’ ‘I’ ou ‘R’

Linha 34 – Estrutura Grafo:

Estrutura que representa o grafo do código, onde tem como variáveis uma estrutura Vértice que é basicamente o estado de cada vértice do grafo, a quantidade de vértices e uma matriz adjacente que representa as ligações do grafo sendo 1 tem ligação 0 não tem ligação

Linha 44 – Estrutura Fila de Prioridades:

Estrutura que representa a fila de prioridades do código, tendo um vetor de eventos que que são os eventos que serão adicionados na fila, um n para representar em qual o evento adicionado atual está e um tam que dirá o tamanho da fila.

Linha 61 – Cria_Filap(int tam)

Função que aloca um espaço na memória para criação fila de prioridades.
Recebe como parâmetro o tamanho total da fila.
Retorna uma fila de prioridades.

Linha 73 – inserction_sort(Fila_Prioridades *fp)

Função que ordena os eventos da fila de prioridades do menor para o maior em relação tempo de atraso do evento. É utilizado o método de inserção para ordenar os eventos.
Recebe uma Fila de prioridades como parâmetro

Linha 89 – insere_fp(Fila_Prioridades *fp, Evento X)

Função que insere um evento no final da fila de prioridades e em seguida chama a função inserção para ordenar a fila.
Recebe como parâmetro uma fila de prioridades e um evento.

Linha 99 – remove_fp(Fila_Prioridades *fp)

Função que remove o primeiro evento de uma lista de prioridades, e em seguida atualiza como o novo primeiro evento da lista, o evento seguinte do anterior.
Recebe como parâmetro uma fila de prioridades.

Linha 110 – reinsere_fp(Fila_Prioridades *fp)

Função que, caso um evento precisar se repetir, salvará o evento em um evento auxiliar atualizara o seu novo tempo de atraso, remover o evento antigo(o primeiro evento da fila) e irá inserir esse evento auxiliar na fila de acordo com seu novo tempo de atraso.
Recebe como parâmetro uma fila de prioridades.

Linha 122 – liberafila(Fila_Prioridades *fp)

Função que liberará o espaço na memória alocado pelos eventos da fila e a própria fila de prioridades.
Recebe como parâmetro uma fila de prioridades.

Linha 138 – criagrafo(int qv)

Função que aloca espaço na memória para criação do grafo , criação dos vértices, matriz.
Recebe como parâmetro a quantidade de vértices que se deseja ter no grafo.
Retorna um grafo.

Linha 154 – criaAresta(Grafo *g, int u, int v)

Função que cria as ligações de um vértice para outro de um determinado grafo, onde é determinado que esse grafo tem ligação caso a matriz adjacente de suas posições seja 1.
Recebe como parâmetro um grafo, a posição da origem da ligação e o destino da ligação

Linha 165 – create_grade_graph(Grafo *g)

Função que cria um grafo grade perfeito.
I)Primeiro é testado se esse grafo é par ou ímpar, caso seja ímpar, sera adicionado 1 para o aux.

II) Para a quantidade total de vértices é testado se eles são diviseis por algum dos numeros começando de 5(Escolhi 5 pois como a quantidade de vertices pedido no trabalho é de 15 a 30, 5 é a melhor opção para divisiveis) e é salvo o divisor da quantidade de vértices em aux2.

III) Temos um for que criará as arestas que estão do lado desse vértice(i+1) e abaixo do vértice(i+aux2) com duas restrições caso o vértice não tenha mais vizinhos do lado que é o caso de termos(i+1 % aux2 == 0) já que temos aux2 vértices por linha e temos o caso de o vértice não ter vizinhos abaixo que é o caso de termos(i+aux2 <=g→qv) onde g→qv é o total de vértices.

Recebe como parâmetro um grafo.

Linha 209 – create_smallworld_graph(Grafo *g, int gps)

Função que cria um grafo do tipo de grupos.
Eu não consegui achar um modo de implementar esse tipo de grafo, onde os grupos teriam numeros aleatórios de vértices, então eu fiz de modo que cada grupo tenha o mesmo número de vértices.
Primeiro é calculado a quantidade maxima de vértices que pode-se ter por grupo

Em seguida é feito um laço contendo dois laços dentro cada, um que cria as ligações dos vértices dentro de um grupo, e outro que ligará o vértice do grupo aos demais.

I) É testado se o vértice atual de i é um multiplo do numero de valores por grupo, pois caso seja, esse vértice será o que se ligará aos vértices de outros grupos, ou seja é criado outro laço que comecara do primeiro multiplo de max_p_gp e acrescentara max_p_gpx para ligar todos os vertices de outros grupos ou seja imagine-se que o grafo tem 4 grupos os vértices a serem ligados serão 0,4,8,12 que serão os “representantes” do grupo, onde o grupo do 0, é composto por, 0,1,2,3; etc…

II) Este outro for é o que faz as ligações de um grupo. Ou seja caso o i atual não ser divisivel pelo max_p_gp, quer dizer que este numero está em um dos grupos dos “representantes” por isso é salvo uma variavel chamada “seed” que é o representante atual do i atual, então esse for ligará esse i atual com todos os vértices do grupo desse representate tendo que esse for vai até que seja menor que seed+max_per_gp que é o próximo representante de outro grupo.

Função Recebe um Grafo e a quantidade de grupos como parâmetro.

Linha 233 – inicializaMatrizAdj(Grafo *g)

Função que irá inicializar o grafo, ou seja , colocar como 0 o valor da matriz adjacente ou seja , isso representa um grafo sem nenhuma ligação.
E também irá colocar todos os vértices como estado suscetível.
Recebe um grafo como parâmetro.

Linha 247 – liberaGrafo(Grafo *g)

Função que liberará o espaço alocado na memória de todos vértices e também do próprio grafo.
Recebe um grafo como parâmetro.

Linha 258 – imprimeMatrizAdj(Grafo *g)

Função que apenas imprime a matriz adjacente, só coloquei ela alí para caso se deseje ver a matriz adjacente do gráfo.
Recebe um grafo como parâmetro.

Linha 271 – imprimeligacoes(Grafo *g)

Função que apenas imprime a ligação de cada vértice do grafo, também , só deixei ela alí para caso de teste, caso precise ver as ligações de cada vértice do grafo.
Recebe um grafo como parâmetro.

Linha 292 – cria_evento(char *n,int td,float pb,bool repeat)

Função utilizada para criar um evento.
Tem como parâmetro uma string para nome do evento, um int para o tempo de atraso desse evento, um float para a probabiliade desse evento acontecer, e um bool para saber se esse evento irá se repetir ou não.
Retorna um evento.

Linha 305 – infectar(Grafo *g, int id)

Função simples, é somente para escolher um ou mais vértices para começarem infectados ou seja, receberem ‘I’ em seu estado.
Tem como parâmetro um grafo , e um int que é a posição do vértice a ser infectado.

Linha 313 – recupera50prct(Grafo *g)

Função que selecione vertices aleatorios e coloca como estado ‘R’ 50% do total

Tem como parâmetro um grafo

Linha 331 – Event_ProbT(float prob)

Função que retornará se o evento irá ou não acontecer dado uma probabilidade.
Funciona da seguinte maneira: Dado uma probabilidade em float, essa probabiliade é multiplicada por 100, em seguida é gerado um numero aleatorio de 1 a 100, caso esse numero seja menor que a probabilidade multilpicada por 100 quer dizer que o evento acontecera.
Recebe um float como parâmetro que é a probabilidade do evento acontecer
Retorna true se o evento irá acontecer, false se não acontecerá.

Linha 345 – percorretest(Grafo *g, Fila_Prioridades *fp)

Função que irá percorrer o grafo vértice por vértice executando os eventos. Funciona da seguinte maneira:
Primeiro é salvo o estado atual de todos os vertices na variavel dos novos estados
Depois é testado se o primeiro evento da fila de prioridades é de infecção ou de recuperação , não consegui fazer essa comparação com a função strncmp, de algum jeito a função em vez de comparar o primeiro evento na fila, ela comparava o segundo, mesmo eu colocando fp→X[0].Nome, então resolvi comparar apenas a primeira letra do evento.
Depois de definir, é feito um laço procurando cada vértice que esteja em estado ‘I’, caso o evento seja de infecção é procurado em todas as ligações com o vértice ‘I’ achado, os vértices que estejam em estado ‘S’ e para cada estado desse, é feito o teste de probabilidade de infecção para que esse ‘S’ seja ou não infectado e caso seja o estado novo é salvo na variavel newstate.Caso o evento seja de Recuperação a função simplesmente roda o evento de probabilidade para esse vértice ‘I’, E no final é atualizado os estados pela variavel newstate.
Recebe um grafo e uma fila de prioridades como parâmetro

Linha 390 – calc_PorcIR(Grafo *g)

Função que calcula a porcentagem do numero de pessoas infectadas, recuperadas e suscetiveis, no final da execucao do programa.
Recebe um grafo como parâmetro.

Linha 413 – printa_resultados(Grafo *g)

Função que printa os estados de cada vértice do gráfo
Recebe um grafo como parâmetro

Linha 423 – run()

Essa é a função principal onde todo código será rodado, onde é declarado as variaveis e funções, nessa função caso ,queira fazer simulações, deve-se escrever as modificações

Linha 485 – main()

A função main, onde apenas tem-se um srand(time(null)) usado para ativar a semente na geração de um valor aleatorio na função de probabilidade, e a função run() para rodar os acontecimentos do evento.

- **Como rodar e fazer simulações no código**

AS MODIFICAÇÕES SÃO SEMPRE NA FUNÇÃO RUN()

I)
Dê um valor para tfinal referente a quantidade de dias das simulaçoes

II)
Crie a quantidade de eventos desejado, no formato:
Evento “X” = cria_evento(“N”,”F”,”B”)
onde:
X é um nome para a variavel evento
N é o nome do evento
F é a probabilidade desse evento acontecer(float; 0<=F<=1)
B se o evento irá ou não se repetir(bool; true=repetira|false=nao repetira)

III)
Coloque o numero de eventos total criado dentro de cria_Filap()

IV)
Diga a quantidade total de vértices que se deseja ter na simulação

V)
Escolha com qual tipo de grafo irá se fazer as simulações colocando ‘//’ em um,caso seja um small world, digitar a quantidade de grupos dentro de create_smallworld_graph(grafo,’ ‘)

VI)
Escolher quais vértices começarão sendo infectados e se começaram com 50% recuperados(utilize a função recupera50prxt(grafo)).
Basta utilizar a função infectar(grafo, ‘Posição do vértice onde deseja começar infectado’).