Relatório

Trabalho prático de

Programação

Simulação da Propagação de Vírus

Trabalho realizado por:

Gonçalo Ramalho – 2019106561

Índice

Programação 2019/2020
Trabalho prático - Simulação da Propagação de Vírus

Introdução

Este relatório tem como base o trabalho prático da cadeira de Programação leccionada em 2019/2020, do primeiro ano de Licenciatura em Engenharia Informática.

O trabalho em si é criar um programa em linguagem C sobre um modelo de propagação de um vírus, onde são introduzidas salas de propagação e pacientes com determinados estados. Estes dois são carregados a partir de ficheiros binários e de texto, respectivamente. Depois de carregar estes dados, o programa está pronto para avançar/recuar iterações, adicionar pessoas, mover pessoas, apresentar estatisticas e no fim da utilização guardar essa mesma estatistica, bem como a lista de pacientes da última iteração.

A configuração do modelo de propagação do vírus foi criada para seguir os parâmetros dados pela ficha. Como tal, esses parâmetros só poderão ser alterados através de alteração directa no código.

É construido inteiramente na norma C99, usando funções e código que temos aprendendo ao longo do semestre. O IDE usado neste trabalho foi o Apache NetBeans 11.2.

Principais Estruturas de Dados

- Estrutura das Salas

```
typedef struct sala local, *plocal;
struct sala{ //SALAS
    int id; // id numérico do local
    int capacidade; // capacidade máxima
    int liga[3]; // id das ligações (-1 nos casos não usados)
};
```

- Estrutura dos Pacientes

```
typedef struct lista pessoa, *ppessoa;
struct lista{ //PESSOAS
    int id_sl; //id numérico do local que foi atribuido à pessoa
    char nome[100]; //nome da pessoa
    int idade; //idade da pessoa
    char estado; //estado em que a pessoa se encontra
    int dias_infectado; // há quantos dias está infectado
    int cura; //dias para a infecção passar
    ppessoa prox;
};
```

- Estrutura das listas de listas de pacientes

```
typedef struct nr_itera itr, *pitr;
struct nr_itera{ //LISTA DE LISTA DE PESSOAS
    int r; //id numérico da iteracção introduzida
    ppessoa lista_clone;
    pitr prox;
};
```

Principais Estruturas Dinâmicas

- Estrutura Dinâmica das Salas – Array de Estruturas Dinâmico

  (retirado do ficheiro prepara_ficheiro.c – muito código foi omitido)

```
local *prepara_ficheiro(char *nomeficheiro, local *dados, int *contador){

    local p;
    FILE *f = fopen(nomeficheiro, "rb");
    if (f==NULL){ // NÃO ESQUECER!!!
        return NULL;
    }

    while (fread(&p, sizeof(local),1,f) == 1)//Enquanto tiver dados para ler
    {
        local *verifica_espaco; //Variável auxiliar para testar realloc
        verifica_espaco = realloc (dados, (*contador+1) * sizeof(local));
        if(verifica_espaco == NULL)
            return dados;
        dados = verifica_espaco;
        dados[*contador] = p; //Atribuição da estrutura
        (*contador)++;
    }
    fclose(f);
    return dados;
}
```

- Estrutura dos Pacientes – Listas Ligadas

  (retirado do ficheiro prepara_lista.c – muito código foi omitido)

```
ppessoa prepara_lista(ppessoa lista, char *ficho, local dados[], int cont){
    ppessoa listanova = NULL, novo, aux;
    pessoa p;
    int i, k, j, controlador, falta, random;
    char sai = 'N';

    FILE *f = fopen(ficho,"rt"); //vai abrir o ficheiro de texto para leitura

    p.prox = NULL;

    while (fscanf(f," %s %d %c", p.nome, &p.idade, &p.estado) == 3 && sai != 'S') { //leitor do ficheiro de texto
       if (p.estado == 'S' || p.estado == 'I' || p.estado == 'D') { //verificador de estado e atribuidor de quantos dias para cura se doente
            if (p.estado=='D') {
                p.cura=(5+(p.idade/10));
                fscanf(f,"%d", &p.dias_infectado);
            }
        }

    novo = malloc(sizeof(pessoa)); //alocar espaço para a estrutura

    while (!controlador) //ciclo infinito para atribuição de salas
        {
        random = intUniformRnd(y, x);
        falta = 0;
            for (j=0; j<cont; j++)
            {
```

```
                if (random == dados[j].id  && dados[j].capacidade > 0) {
//encontra uma sala disponovel, atribuí e sai
                    p.id_sl = random;
                    (dados[j].capacidade)--;
                    *novo = p;
                    controlador = 1;
                }
                if (random == dados[j].id  && dados[j].capacidade == 0){ //
quando encontra uma sala vazia, sai e não atribuí
                    for (k=0; k<cont; k++){
                        falta = dados[k].capacidade + falta;
                    }
                    if (falta == 0){
                        controlador = 1;
                        sai= 'S';
                    }
                }
            }
        }

        novo->prox = listanova;
        listanova = novo;

    }

    fclose(f); //fechar o ficheiro de texto
    return listanova;
}
```

- Estrutura das listas de listas de pacientes – Lista de Listas

  (retirado do ficheiro volta_atras.c – muito código foi omitido)

```
pitr cria_listas(pitr lista_guarda, ppessoa lista, int itera){
    pitr novo = NULL;
    ppessoa copia = NULL;

    novo = malloc(sizeof(itr));
    if(novo == NULL)
        return lista_guarda;
    novo->r = itera;
    novo->lista_clone=NULL;
    while(lista!=NULL){
        copia = (ppessoa)malloc(sizeof(pessoa));
        if(copia == NULL){
            free(novo);
            return lista_guarda;
        }
        *copia = *lista;

        copia->prox = novo->lista_clone; //introduzir à cabeça
        novo->lista_clone = copia;//ligar à lista principal
        lista=lista->prox;//avançar na lista a copiar
    }


    novo->prox = lista_guarda; //ligar ao nó principal
    lista_guarda = novo;
    return lista_guarda;
}
```

Opções tomadas em termos de implementação

Em termos de implementação, foi requerido pelos docentes que o programa deveria carregar as informações dos espaços num vector dinâmico de estruturas, e as informações dos pacientes numa estrutura dinâmica de listas ligadas. Tudo o resto foi deixado ao critério do aluno.

Como poderão constatar no código do programa, recorri muito a ciclos em especial para atribuir salas, quem fica curado, quem fica infectado e quem fica imune entre outros pequenos eventos. Graças às funções int **intUniformRnd**(int a, int b) e int **probEvento**(float prob) dadas pelos docentes. A primeira gera um número entre a e b, a última gera a probabilidade e um determinado evento com x probabilidade acontecer. Vamos ver dois exemplos como estas funções ajudaram em muito randomizar certos eventos:

- Atruibuir o estado Saudável ou Imune a determinada pessoa, quando já passou o tempo de infecção máximo desta (ppessoa **duracao_maxima**(ppessoa lista))

```
p=probEvento(0.2); //p recebe 1, se acontecer a probabilidade de "sair" 0.2 em 1.0
    if (aux->cura == 0 && aux->estado == 'D')
        if (p == 1) //se a probabilidade acontecer, a pessoa fica Imune
            aux->estado = 'I';
        else //senão, fica Saudável
            aux->estado = 'S';
```

- Atruibuir o estado Doente a uma pessoa aleatoriamente devido à presença de outra pessoa Doente na sala, ao avançar uma iteração (ppessoa **disseminacao**(ppessoa lista, int id))

```
while (cont != nr_infectos){ //ciclo só pára quando o número de pessoas infectadas for igual ao numero de pessoas (definido na taxa de disseminação) que cada doente obrigatoriamente infecta
     while (aux!=NULL){ //ciclo para correr a lista de pessoas
        random = intUniformRnd(0, 1); //gera um número entre 0 e 1
        if(id == aux->id_sl){ //atribuir aleatoriamente novas pessoas doentes
```

```
            if (random == 1){ //se randomizou 1, quer dizer que essa pessoa vai ser infectada (ou não, se esta for imune), se randomizar 0, essa pessoa não fica infectada
                if(aux->estado == 'S'){
                   aux->dias_infectado = 0;
                   aux->cura=(5+(aux->idade/10));
                }
                if(aux->estado != 'I') //só infecta se não for imune, mas fica incluido na contagem de pessoas que foram infectadas
                   aux->estado = 'D';
                cont++; //contador de controlo, que incrementa cada vez que uma pessoa foi infectada
                }
            }
            if(cont == nr_infectos) //se o número de infectados for igual ao numero requerido da taxa de disseminação, sai do ciclo de pacientes, e consequentemente sai do ciclo principal
                break;
            aux = aux->prox;
        }
        aux = lista; //se saiu da lista de pessoas, e o numero de pessoas que foram infectadas ainda não for igual ao numero requerido da taxa de disseminação, voltamos ao inicio e o ciclo recomeça. Pode por exemplo a função gerar muitos 0's e poucos 1's, e por isso temos de voltar a randomizar mais.
    }
```

Programação 2019/2020
Trabalho prático - Simulação da Propagação de Vírus

Manual de utilização

Este programa tem como objectivo simular a propagação de um agente viral num espaço fechado com pessoas. Consegue prever o comportamento do virus usando vários modelos matemáticos. Este programa está preparado para as seguintes funções:

- Avançar 1 Iteracao na Simulacao: são efectuados os cálculos e a recolha de novos dados.
- Apresentar Estatística: apresenta dados estatísticos entre os quais incluí taxa de pessoas doentes, taxa de pessoas saudáveis e taxa de pessoas imunes.
- Adicionar Doente: adiciona uma pessoa num espaço à escolha.
- Transferir Pacientes: transfere pacientes para um espaço à escolha.
- Retroceder X Iterações: permite retroceder até 3 iterações.

Ao iniciar o programa, o utilizador deve introduzir o nome do ficheiro de espaços (obrigatório ser em formato .bin), e o nome do ficheiro de pessoas (obrigatório ser em formato .txt). Qualquer erro na formatação destes dois ficheiros o programa encerra imediatamente.

O ficheiro binário das salas deve conter o seu ID, a sua capacidade e as suas ligações para outras salas (no máximo 3 ligações). Estas ligações servem para mover pessoas para outras salas, e em nada interfere na disseminação, não existe propagação do agente viral inter-salas. Para uma sala para ligar à outra tem de obrigatóriamente existir ligações entre as duas, isto é, para que uma sala que ligue a outra, esta deve ter ligação para a última e vice-versa.

O ficheiro de texto com a lista de pessoas deve conter uma código alfanumérico (único de cada pessoa), a sua idade, e o seu estado (S-saudável, D-doente e I-imune). No caso de ser doente, deve também ser introduzido a contagem do número de dias que tem estado doente. Deve ser introduzido esta informação numa linha por paciente, e os dados de cada um separados por pelo menos um espaço.

No final do programa também é guardado automáticamente o relatório estatístico final e a lista de pessoas final, cujo nome do ficheiro é escolhido pelo utilizador.

Para qualquer alteração dos modelos matemáticos e dúvidas de erros apresentados durante a execução que não sejam possíveis de resolver mesmo verificando o troubleshooting, contactar o programador via email (a2019106561@isec.pt).

Troubleshooting:

Código de erro	Solução
[0]	Memória do computador insuficiente para alocar espaço – Instalar numa máquina com mais memória
[1]	Formato do ficheiro inválido (deve ser .bin) ou não existente
[2]	Formato do ficheiro inválido (deve ser .txt) ou não existente
[3]	Espaços com ID's iguais – requer alteração directa no ficheiro
[4]	Espaços com ID's negativos – requer alteração directa no ficheiro
[5]	Espaços com ligações inválidas – requer alteração directa no ficheiro
[6]	Lista de pessoas contém um código alfanumérico duplicado – requer alteração directa no ficheiro
[7]	Lista de pessoas contém uma idade inválida (negativa) – requer alteração directa no ficheiro
[8]	Lista de pessoas contém um estado inválido (só pode ser S, D ou I) – requer alteração directa no ficheiro
[9]	Lista de pessoas contém um registo dos dias infectado inválido (negativo) – requer alteração directa no ficheiro
[10]	Erro a criar ficheiro de texto – Reiniciar o programa, caso o problema persista considerar o erro [0]

Conclusão

A aprendizagem de novas funções, estratégias de implementação e lógica em linguagem C na cadeira de Programação, foram muito úteis para implementar as funcionalidades do programa. Com elas aprendi a construir um programa que fosse realmente "útil", pois este já consegue alocar memória e criar/ler ficheiros para quando o programa terminar, os dados importantes para nós não serem perdidos.

Este trabalho prático contribuiu em muito no aumento da minha capacidade de lógica e resolução de problemas que é, de facto, um pilar essencial para qualquer pessoa que pretenda licenciar-se em engenharia informática.