//Daiana Kathrin Santana Santos 120.357

//Isadora Rosa de Freitas Muniz 120.431

#define \_GNU\_SOURCE

#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>

#include <pthread.h>//Biblioteca para pthread\_self()

#include <dlfcn.h>

#include <sys/types.h>

#include <signal.h>

#define MaxVert 100

#define MAX 100

#define true 1

#define false 0

typedef int TVertice;

typedef int TAresta;

typedef struct *Cel* TApontadorLista;

typedef struct *Cel*{

int vertice;

TApontadorLista \*proximo;

}*TCelula*;

typedef struct{

int tamanho;

TApontadorLista \*primeiro;

TApontadorLista \*ultimo;

}*TLista*;

typedef struct{

TVertice vertice;

TAresta aresta;

}*TAdjacencia*;

typedef struct{

TLista \*Adjancentes[100];

int NVertices;

int NArestas;

}*TGrafo*;

typedef struct *Celula* TApontadorPilha;

typedef struct *Celula*{

int item;

TApontadorPilha \*proximo;

}*TCelulaPilha*;

typedef struct *Pilha*{

TApontadorPilha \*topo;

}*TPilha*;

//VARIAVEIS GLOBAIS

long vertices[MaxVert];

int NumVert = 0;

TGrafo grafo;

int IniciouGrafo = 0;

//ESTRUTURA DE DADOS LISTA UTILIZADA NA ESTRUTURA DE GRAFOS

*TLista* \*cria\_lista(){// FUNCAO DE CRIACAO DA LISTA QUE CONTERAO OS VERTICES DO GRAFO

TLista \*pLista = (TLista\*)malloc(sizeof(TLista));

pLista->primeiro=NULL;

pLista->ultimo=NULL;

pLista->tamanho=0;

return pLista;

}

void inicia\_lista(*TGrafo* \**pgrafo*, int *nvertices*){//INICIALIZACAO DA LISTA DE CONTEUDO DOS VERTICES DO GRAFO

int i;

for(i=0;i<nvertices;i++){

pgrafo->Adjancentes[i]=cria\_lista();

}

}

void insere\_vertice(*TGrafo* \**pgrafo*, int *posicao*, int *novo\_vertice*){// FUNCAO PARA INSERCAO DE UM NOVO VERTICE NO GRAFO

TCelula \*nova = (TCelula\*)malloc(sizeof(TCelula));

nova->vertice=novo\_vertice;

nova->proximo=NULL;

if(pgrafo->Adjancentes[posicao-1]->tamanho == 0)

{

pgrafo->Adjancentes[posicao-1]->primeiro=nova;

pgrafo->Adjancentes[posicao-1]->ultimo=nova;

}

else

{

pgrafo->Adjancentes[posicao-1]->ultimo->proximo=nova;

pgrafo->Adjancentes[posicao-1]->ultimo=nova;

}

pgrafo->Adjancentes[posicao-1]->tamanho++;

}

//ESTRUTURA DE DADOS GRAFOS UTILIZADO PARA ARMAZENAR AS THREADS

void tgrafo\_inicia(*TGrafo* \**pgrafo*, int *NVertices*){ //INICIA GRAFO

int i;

pgrafo->NVertices=NVertices;

pgrafo->NArestas=0;

for(i=0;i<pgrafo->NVertices;i++){

pgrafo->Adjancentes[i]=cria\_lista(); //INICIA LISTA DE ARESTAS COM NULL

}

}

int tgrafo\_inserearesta(*TGrafo* \**pgrafo*, int *posicao*, int *valor*){

insere\_vertice(pgrafo, posicao, valor); //INSERE ARESTA NO GRAFO

pgrafo->NArestas++;

return 0;

}

//ESTRUTURA DE DADOS PILHA USADA PARA ACHAR CICLOS

*TPilha*\* criaPilha(void){ //CRIA PILHA

TPilha \*pilha = (TPilha\*)malloc(sizeof(TPilha));

pilha->topo=NULL; //APONTA TOPO PARA NULL

return pilha;

}

int PilhaEhVazia(*TPilha*\* *pilha*){ //VERIFICA SE A PILHA ESTA VAZIA OU SEJA, TOPO APONTA PARA NULL

if(pilha->topo==NULL)

return 1;

else

return 0;

}

void Empilha(*TPilha*\* *pilha*, int *valor*){

TCelulaPilha \*novo = (TCelulaPilha\*)malloc(sizeof(TCelulaPilha)); //CRIA ESPACO PARA NOVO ITEM

novo->item=valor; //ADICIONA CONTEUDO DO NOVO ITEM

novo->proximo=pilha->topo;

pilha->topo=novo;//ULTIMO ITEM INSERIDO AGORA EH O TOPO DA PILHA

}

float Desimpilha(*TPilha*\* *pilha*){ //DESALOCA O ESPACO DA PILHA

float valor;

TCelulaPilha\* item;

if(PilhaEhVazia(pilha)==1){

exit(1);

}else{

item=pilha->topo;

valor=item->item;

pilha->topo=item->proximo;

free(item);

return valor;

}

}

int busca\_ciclo(int *vert*)

{

TPilha \*pilha = criaPilha();

int visitados[grafo.NVertices], pilha\_visitados[grafo.NVertices];

// INICIALIZA visitados e pilha\_visitados EM 0

for(int i = 0; i < grafo.NVertices; i++)

visitados[i] = pilha\_visitados[i] = 0;

// FAZ UMA BUSCA EM PROFUNDIDADE PARA ENCONTRAR CICLOS

while(true)

{

int achou\_vizinho = 0;// FLAG QUE SINALIZA A EXISTENCIA DE VIZINHOS

TCelula \*aux;

if(!visitados[vert]) {// SE VERTICE AINDA NAO FOI VISITADO, SEU VALOR EH ATUALIZADO NA PILHA DE VISITADOS

Empilha(pilha, vert);

visitados[vert] = pilha\_visitados[vert] = 1;

}

aux = grafo.Adjancentes[vert]->primeiro;

while(aux != grafo.Adjancentes[vert]->ultimo)

{

// SE VERTICE VIZINHO JA ESTA NA PILHA, HA UM CICLO, ENCONTRANDO UM DEADLOCK

if(pilha\_visitados[aux->vertice])

return 1;

else if(!visitados[aux->vertice])

{

// SE VERTICE NAO FOI VISITADO, EH APENAS UM VIZINHO

achou\_vizinho = 1;

break;

}

aux = aux->proximo;

}

if(!achou\_vizinho)// SE NAO HA VIZINHO

{

pilha\_visitados[pilha->topo->item] = 0; // SAI DA PILHA

Desimpilha(pilha); // REMOVE DA PILHA

if(PilhaEhVazia(pilha))

break;

vert = pilha->topo->item;

}

else

vert = grafo.Adjancentes[vert]->ultimo->vertice;

}

return 0;

}

int existe\_ciclo()// FUNCAO QUE EXECUTA A BUSCA EM PROFUNDIDADE PARA A PROCURA DE CICLOS

{

for(int i = 0; i < grafo.NVertices; i++)

{

if(busca\_ciclo(i))

return 1;

}

return 0;

}

int converteVertice(long *Valor*){ //PASSA A POSICAO QUE ESTA O VALOR NOS VÉRTICES

int i;

for(i = 0; i < NumVert; i++){

if(vertices[i] == Valor){

NumVert++;

return i;

}

}

NumVert++; //COMO NUMVERT EH UMA VARIAVEL GLOBAL, ELA EH INCREMENTADA PARA PROXIMA VEZ QUE FOR UTILIZADA

vertices[i] = Valor; // GUARDA OS VALORES NO VETOR DE VERTICES

return i;

}

int (\*\_sem\_wait)(*sem\_t* \*) = NULL; //APONTA PARA FUNCAO ORIGINAL, PARA PODE REESCREVELA

int sem\_wait(*sem\_t* \**sem*) {// REESCREVENDO A FUNCAO SEM\_WAIT PARA DETECTAR DEADLOCKS E NAO BLOQUEAR NO SEMAFARO

int r; //RECURSO R

if (IniciouGrafo == 0){ //SE NUNCA INICIOU O GRAFO PARA GUARDAR AS THREADS

IniciouGrafo = 1;

tgrafo\_inicia(&grafo, MaxVert);

}

if (!\_sem\_wait) {

\_sem\_wait = dlsym(RTLD\_NEXT, "sem\_wait"); //IRA APONTAR PARA O SEM\_WAIT ORIGINAL

}

else if (existe\_ciclo() == false) { //NAO TEM CICLO, ENTAO NAO TEM DEADLOCK

tgrafo\_inserearesta(&grafo, converteVertice((size\_t)sem), converteVertice(pthread\_self())); //INSERE NO GRAFO A ARESTA COM O ID DA THREAD

printf("\t Semaforo %ld na thread %d\n", (size\_t)sem, pthread\_self()); //TAMANHO EM BYTES DO SEMAFORO E ID DA THREAD

r = \_sem\_wait(sem);

return(r); //RETORNA RECURSO

}else{

printf("\t Encontrou deadlock\n"); //AVISA QUANDO ENCONTRA DEADLOCK

exit(127);

return 0; //RETORNA 0 CASO TENHA ENCONTRADO DEADLOCK

}

return 0;

}