Relações binárias

Nome: Gabriel Victor Carvalho Rocha **Matricula:** 2018054907

1. Introdução

O objetivo do programa é verificar as propriedades de uma relação binária, tendo como resultado final a validade dessas propriedades e seus possíveis fechos. Após a leitura do arquivo, foi colocado os valores obtidos dentro de variáveis (números de elementos), vetores (elementos da relação) e matrizes (pares de tuplas) para realizar a criação de uma matriz que tivesse zeros (não existe a relação) e uns (existe a relação) e assim determinar cada propriedade a partir das funções que serão listas abaixo.

2. Funções

2.1. int main(int argc, char *argv[ ]):

Na função ***2.1***, após declarar as variáveis necessárias, foi iniciado a leitura do arquivo, verificando sua validade, pegando os números de elementos, colocando os elementos em um vetor elementos[50] e alocando as relações em uma matriz de tuplas [2500][2]. O *primeiro ninho de for (linha 433)* dessa função zera todos os elementos da matriz[50][50], que posteriormente, no *terceiro ninho de for (linha 461)* irá achar a posição das tuplas de acordo com o vetor elementos, e colocará “1” nessa posição encontrada da matriz[50][50].

Após isso, irá fazer as chamadas das funções que serão listadas abaixo.

2.2. void testaRefleIrrefle(int matriz[ ][50], int num_elementos, int elementos[ ], int *reflexiva):

Essa função verifica as propriedades Reflexiva e Irreflexiva, irá receber a matriz de “zeros e uns”, os números de elementos, o vetor com os elementos da relação e um ponteiro “reflexiva” que irá ser usado posteriormente nas funções ***2.5*** e ***2.6,*** após isso, verificará quando os índices da matriz de “zeros e uns” forem iguais, caso tenha “1”, entrará no primeiro *if* desta função e colocará esse índice em uma matriz de tuplas irreflexivas, que são as relações que fazem ela não ser irreflexiva. Caso entre no *else*, irá fazer o mesmo, porém colocará os índices na matriz de tuplas reflexivas, que são as relações que

fazem ela não ser reflexiva e então irá validar pelos dois *if's* se cada uma das duas propriedades é V ou F, imprimindo isso quando for chamada na função ***2.1***.

2.3. void testaSimetriAntisimetriAssi(int matriz[ ][50], int num_elementos, int elementos[ ], int *simetrica, int *antisimetrica):

Essa função verifica as propriedades Simétrica, Anti-simétrica e Assimétrica, irá receber a matriz de "zeros e uns", os números de elementos, o vetor com os elementos da relação, um ponteiro simétrico que será usado posteriormente na função ***2.5*** e um ponteiro antissimétrico que será usado posteriormente na função ***2.6*****,** então irá verificar caso uma matriz[i][j] for igual a uma matriz[j][i] **e** as duas matrizes forem iguais a "1", colocará esses índices em uma matriz de tuplas anti-simétrica, que são as relações que fazem ela não ser anti-simétrica e, caso ela entre nesse *if*, automaticamente ela não será assimétrica, aumentando em "1" seus elementos q não a fazem ser assimétrica. E quando entrar no *else if* com a condição de matriz[i][j] diferente de matriz[j][i] e **uma** delas terem "0" como valor, colocará em uma matriz de tuplas simétricas, que são as relações que fazem ela não ser simétrica e então irá validar pelos três *if's* se cada uma das três propriedades é V ou F, imprimindo isso quando for chamada na função ***2.1***.

2.4. void testaTransitiva(int matriz[ ][50], int num_elementos, int elementos[ ], int *transitiva):

Essa função verifica a Propriedade Transitiva, irá receber a matriz de "zeros e uns", os números de elementos, o vetor com os elementos da relação e um ponteiro "transitiva" que será usado nas funções ***2.5*** e ***2.6***, então irá verificar caso exista uma matriz[i][j] e matriz[j][k] iguais a "1", e caso exista uma matriz[i][k] igual a "0", colocará esses índices em uma matriz de tuplas transitivas, que são as relações que fazem ela não ser transitiva, porém transitiva existe uma peculiaridade, pois após essa verificação, a matriz de zeros e uns será modifica podendo ocasionar mais casos que façam ela não ser transitiva. Então é feito uma copia da matriz de "zeros e uns" que entrará em um *while* que irá testar até que seu retorno seja "0" e então irá validar pelo

if se a propriedade é V ou F, imprimindo isso quando for chamada na função ***2.1***.

2.5. void testaEquivalencia(int *reflexiva, int *simetrica, int *transitiva):

Essa função verifica se a Relação é de Equivalência, irá receber o ponteiro “reflexiva”, o ponteiro “simétrica” e o ponteiro “transitiva” que foram testadas nas funções ***2.2***, ***2.3*** *e* ***2.4*** respectivamente. As funções citadas retornaram 0 ou 1 para seus respectivos ponteiros, sendo 0 para falso e 1 para verdadeiro. De acordo com esses resultados, essa função irá analisar pelos *if's* quando todos os valores forem 1, o que retorna verdadeiro para Relação de Equivalência e caso um dos ponteiros seja 0, retorna falso para Relação de Equivalência, imprimindo isso quando for chamada na função ***2.1***.

2.6. void testaParcial(int *reflexiva, int *antisimetrica, int *transitiva):

Essa função verifica se a Relação é de Ordem Parcial, irá receber o ponteiro “reflexiva”, o ponteiro “antisimetrica”, o ponteiro “transitiva” que foram testadas nas funções ***2.2***, ***2.3*** e ***2.4*** respectivamente. As funções citadas retornaram 0 ou 1 para seus respectivos ponteiros, sendo 0 para falso e 1 para verdadeiro. De acordo com esses resultados, semelhante a função ***2.5***, essa função irá analisar pelos *if's* quando todos os valores forem 1, o que retorna verdadeiro para Relação de Ordem Parcial e caso um dos ponteiros seja 0, retorna falso para Relação de Ordem parcial, imprimindo isso quando for chamada na função ***2.1***.

2.7. void fechoReflexivo(int matriz[ ][50], int num_elementos, int tuplas[ ][2], int elementos[ ], int num_tuplas):

Essa função imprime o fecho reflexivo da relação, irá receber a matriz de “zeros e uns”, os números de elementos, a matriz com as relações do arquivo, o vetor de elementos e o numero de relações do arquivo. Essa função basicamente faz a verificação da função ***2.2***, porém restritamente para reflexiva, pegando todas as relações que fazem ela não ser reflexiva. Então imprime todas as relações do arquivo e logo após imprime as relações que não fazem ela ser reflexiva que foram adquiridas na verificação acima. No final, a

união das impressões quando chamada na função ***2.1*** resulta no fecho reflexivo.

2.8. void fechoSimetrico(int matriz[ ][50], int num_elementos, int tuplas[ ][2], int elementos[ ], int num_tuplas):

Essa função imprime o fecho simétrico da relação, irá receber a matriz de “zeros e uns”, os números de elementos, a matriz com as relações do arquivo, o vetor de elementos e o numero de relações do arquivo. Essa função irá fazer a mesma verificação da função ***2.3***, porém restritamente para simétrica, pegando todas as relações que fazem ela não ser simétrica. Então imprime todas as relações do arquivo e logo após imprime as relações que não fazem ela ser simétrica que foram adquiridas na verificação acima. No final, a união das impressões quando chamada na função ***2.1*** resulta no fecho simétrico.

2.9. void fechoTransitivo(int matriz[ ][50], int num_elementos, int tuplas[ ][2], int elementos[ ], int num_tuplas):

Essa função imprime o fecho transitivo da relação, irá receber a matriz de zeros e uns, os números de elementos, a matriz com as relações do arquivo, o vetor de elementos e o numero de relações do arquivo. Essa função irá fazer exatamente a mesma verificação da função ***2.4***, pegando todas relações que fazem ela não ser transitiva. Então imprime todas as relações do arquivo e logo após imprime as relações que não fazem ela ser transitiva que foram adquiridas na verificação acima. No final, a união das impressões quando chamada na função ***2.1*** resulta no fecho transitivo.