Matemática Discreta
Trabalho Pratico – Relações Binárias
Professor: Antonio Alfredo Ferreira Loureiro

Aluno: Vinicius Julião Ramos - Matricula: 2018054630

Documentação do código

Introdução

O objetivo final do exercício é listar as propriedades de uma determinada relação binária que será informada através de um arquivo de texto que o programa lerá. A resolução de tal problema foi implementada juntamente com a validação do arquivo de relações, verificando se ele está corretamente digitado ou não, além de um pequeno controle de memória e de erros para a otimização do programa.

Todos os dados foram armazenados em vetores de uma ou duas dimensões, de acordo com a necessidade. O conjunto, da primeira linha do arquivo da relação foi locado em um vetor de uma dimensão. As tuplas das demais linhas do arquivo foram armazenadas em matrizes de duas dimensões na forma **m[n][2]**, em que **n** é o numero de tuplas e **2** representa os elementos **(a, b)** da tupla. Por ultimo, utilizou-se a criação de uma matriz de duas dimensões para a modelagem do problema, em que o grafo dirigido seria mapeado.

O desenvolvimento do problema central do trabalho está contido no item **3.3** das funcionalidades listada abaixo. Tal desenvolvimento foi proposto através da criação de 3 laços FOR aninhados para percorrer a matriz do grafo dirigido destacando os índices com os quais se percorre a matriz:

O primeiro laço realiza a captura dos itens das propriedades reflexiva e irreflexiva, visto que estes se encontram na diagonal principal da matriz

No segundo laço, realiza-se a captura das propriedades simétrica, assimétrica e antissimétrica. Pois é necessário comparar dois elementos que se encontram em índices diferentes.

Para obter o resultado da propriedade transitiva, não basta o terceiro laço aninhado. Após a execução deste, deve-se verificar, recursivamente, se com a adição dos novos elementos encontrados para o fecho transitivo, é necessário adicionar mais algum elemento, o que foi executado pelas funções descritas nos itens **3.4** e **3.5** abaixo.

Funcionalidades

1 – Leitura de arquivos / Validação do arquivo de relações

Para tal tarefa algumas as seguintes funções foram desenvolvidas:

1.1 int getNumber(FILE *filePointer)

Recebe:

O ponteiro do arquivo para o caracter que será lido.

Retorna:

O valor de apenas 1 numero lido.

Um erro caso seja identificado algum caracter diferente de ‘0’ até ‘9’, ‘-’, ‘ ‘.

1.2 int readNumber(FILE *filePointer, int initialNumber)

Recebe:

O ponteiro do arquivo para ler algum caracter

Um numero inicial que será multiplicado por 10 e somado ao próximo algarismo lido para cada recorrência diferente de ‘ ‘ ou ‘\n’.

Retorna:

Um erro caso seja identificado algum caracter diferente de ‘0’ até ‘9’, ‘-’, ‘ ‘.

Executa o processo de montagem do numero a partir dos caracteres digitados no arquivo de relações.

1.3 char getLastReadChar(FILE *filePointer)

Recebe:

O ponteiro do arquivo após a leitura de uma determinada linha

Retorna:

EOF ou ‘\n’

Um erro caso não encontre EOF ou ‘\n’

Valida se o ultimo carater lido, é algum dos dois acima. Caso não seja, é por que o arquivo está incorretamente montado, com algum elemento a mais no final de determinada linha.

1.4 void getElements(FILE *filePointer, int *elements, int size, int canEqualsElements)

Recebe:

O ponteiro de arquivo no início da linha

O vetor aonde os elementos lidos serão alocados e o tamanho desse vetor.

Uma flag que valida se os elementos naquela linha podem repetir ou não.

Executa a chamada da função **getNumber()** e verifica a flag **canEqualsElements**, caso não seja permitido a repetição de elementos dentro do vetor, a função não insere o mesmo.

1.5 int **new2ulpa(int **oldMatrix, int *length)

Recebe:

A matriz de tuplas quem será realocada para a inserção de um novo elemento

O ponteiro com da variável que indexa a matriz

Retorna:

Uma nova matriz com uma tupla a mais disponível para ser preenchida.

Realiza um controle de memória realocando 1 espaço a mais para uma nova tupla e copiando o**ldMatriz** para essa nova matriz retornada.

1.6 int **read2uplas(FILE *filePointer, int *length)

Recebe:

O ponteiro do arquivo de relações apontando para o inicio da linha.

A referencia de uma variável que armazenará o tamanho da matriz de tuplas que será retornada.

Retorna:

Uma matriz de tuplas com os elementos lidos do arquivo.

Executa o processo de alocação dinâmica, reservando apenas a memória necessária para tal matriz. Preenche a matriz com a tuplas lidas do arquivo.

1.7 int **addNewElementOn2upla(int **oldMatrix, int *length, int elementX, int elementY)

Recebe:

Uma matriz que receberá uma nova tupla.

A referência da variável que contem o tamanho da matriz.

os elementos da nova tupla.

Retorna:

A nova matriz contendo os novos elementos informados nos parâmetros.

2 – Controle de memória/Controle de erros

2.1 void freeMatrix(int **matrix, int linesNumber)

Recebe:

A matriz que será liberada

O numero de linhas da matriz para eliminar cada ponteiro das linhas

2.2 int main(int argc, char *argv[])

Usa as funções **setjmp()** e **longjmp()** para realizar o controle de erros nas chamadas de cada função secundária e caso encontre um erro retorna um mensagem ao usuário e realiza a limpeza de memória.

2.3 biblioteca setjmp.h

A biblioteca executa um tipo de GoTo, de acordo com o local marcado pelas variváeis do tipo **jmp_buf**, os locais são marcados pela função **setjmp()** que recebe um **jmp_buf,** caso deseja-se voltar ao local indicado basta chamar **longjmp()** passando o mesmo **jmp_buf** de onde de deseja retornar.

2.4 void printError(char *errorMessage, int errorCode, int typeJumper)

Recebe:

Uma string com a mensagem de erro.

Um ineiro com o código do erro observado.

O código do local onde o programa deve seguir seu fluxo.

Usa os métodos **setjmp()** e **longjmp()** para manipular o fluxo do código e retornar até o ponto de liberação da memória.

3 – Resolução do Trabalho

3.1 int getIndexInArray(int searchedElement, int *elements, int elementsNumber)

Recebe:

O elemento procurado dentro vetor **elements**.

Vetor de elementos lido do arquivo e o numero de elementos do vetor

Retorna:

A posição que **searchedElement** se encontra

Erro caso não exista o elemento procurado dentro do vetor

3.2 int **mountGraphMatix(int *elements, int elementsNumber, int **relation, int relationLength)

Recebe:

Vetor de elementos lido do arquivo e o número de elementos.

Matriz contendo as tuplas lidas do arquivo e o numero de tuplas lidas.

Retorna:

Uma matriz que mapeia o grafo dirigido montado a partir das tuplas lidas.

3.3 void verifyRelations(int *elements, int **graphMatrix, int elementsNumber, int **relations, int relationsLength)

Recebe:

Elementos lidos do arquivo e o tamanho do vetor de elementos

Matriz do grafo dirigido

Tuplas lidas do arquivo e a quantidade de tuplas

Procedimento central do trabalho. Após a correta leitura do arquivo e o mapeamento da relação, tal função executa 3 laços para que seja possível varrer a matriz **graphMatrix** e obter o resultado do trabalho. Além disso, este método realiza a montagem dos subfechos da relação, marcando os elementos faltantes para satisfazer determinadas propriedades e com tais subfechos o procedimento imprime a saída esperada do exercício.

Este método também realiza a impressão dos fechos simétrico, reflexivo e transitivo, caso estes necessitem de serem completados pelo subfechos encontrados anteriormente. Se não for o caso, os fechos não serão impressos, pois o fecho será a própria relação descrita no arquivo lido.

3.4 int **getTransitiveSubset(int **graphMatrix, int elementsNumber, int **initialSubset, int *lengthOfInitialSubset)

Recebe:

Matriz do grafo dirigido e o tamanho do vetor de elementos.

A matriz de tuplas do subfecho transitivo após realizar os 3 laços em **verifyRelations()** e a quantidade de tuplas.

Retorna:

Retorna uma nova matriz de tuplas do subfecho transitivo, contendo os elementos faltantes.

3.4 int **confirmTransitivity(int **graphMatrix, int elementsNumber, int **initialSubset, int *lengthOfInitialSubset)

Recebe:

Matriz do grafo dirigido e o tamanho do vetor de elementos.

A matriz de tuplas do subfecho transitivo após realizar os 3 laços em **verifyRelations()** e a quantidade de tuplas.

Retorna:

Retorna uma nova matriz de tuplas do subfecho transitivo, contendo os elementos faltantes.

Tal procedimento executa a recursão para validar se após a primeira validação dos elementos transitivos faltantes no fecho, o fecho ainda carece de alguma tupla que complete a transitividade do mesmo.

3.5 void print2uplas(char *propertyName, int *elements, int **subset, int subsetLength)

Recebe:

O nome da propriedade.

O vetor de elemento lidos do arquivo e a quantidade de elementos.

O subfecho que completa a relação o tamanho de tuplas no subfecho.

Executa a impressão na tela na forma “(x,y); ”, em que **x** e **y** são elementos do subfecho que identificam o motivo pelo qual a relação possui ou não possui a propriedade descrita por **propertyName**.

3.6 void printAntissymmetry(char *propertyName, int *elements, int **subset, int subsetLength)

Recebe:

O nome da propriedade.

O vetor de elemento lidos do arquivo e a quantidade de elementos.

O subfecho que completa a relação o tamanho de tuplas no subfecho.

Realiza a impressão dos pares antissimétricos, identificando o motivo pelo qual a relação pode não pode ter tal propriedade.

3.7 char getOrdemEquivalencia(int transitiveSubsetLength, int simetricSubsetLength, int reflectiveSubsetLength)

Recebe:

Tamanho dos subfechos reflexivo, simetrico e transitivo

Retorna:

‘V’ caso a relação seja de ordem de equivalencia

‘F’ caso a relação não seja de ordem de equivalencia

3.8 char getOrdemParcial(int transitiveSubsetLength, int antissimetricSubsetLength, int reflectiveSubsetLength)

Recebe:

Tamanho dos subfechos reflexivo, antissimétrico e transitivo

Retorna:

‘V’ caso a relação seja de ordem parcial

‘F’ caso a relação não seja de ordem parcial

3.9 void printSubset(int **relations, int relationsLength, int *elements, int **subset, int subsetLength)

Recebe:

A matriz contendo as tuplas lidas do arquivo e o tamanho da mesma.

O vetor de elemento lidos do arquivo.

O subfecho que completa a relação o tamanho de tuplas no subfecho.

Faz a impressão de algum determinado fecho (transitivo, simétrico ou reflexivo).

ESTE DOCUMENTO SEGUE OS PADRÕES DESCRITOS EM:
https://www.ime.usp.br/~pf/algoritmos/aulas/docu.html