Matemática discreta

Matriz de adjacência - a que me diz onde então as relações - linha (origem) coluna (destino)

No começo do código, eu incluo as bibliotecas padrão do C (stdlib.h, stdio.h, string.h) que são necessárias para alocação de memória e leitura de arquivos. Também define uma constante chamada max_elementos com valor 100, que serve apenas como uma referência de limite superior, embora o código utilize alocação dinâmica para se ajustar exatamente ao tamanho informado no arquivo de entrada.

Em typedef struct, eu crio a estrutura de dados principal chamada relação, que vai organizar todas as informações do grafo na memória. Dentro dela, defino int **matriz, que é um ponteiro para ponteiro usado para criar a matriz de adjacência dinamicamente; defino o inteiro n, que armazena a quantidade de elementos (nós) do conjunto; e um vetor de strings elementos opcional para nomes, garantindo que todos os dados da relação fiquem agrupados em um único objeto lógico.

No requisito 1, implementei a função ler_entrada, que é responsável por abrir e interpretar o arquivo de texto. O algoritmo lê o arquivo caractere por caractere: quando encontra a letra "n", ele lê o tamanho do conjunto e aloca a memória necessária para a matriz; quando encontra a letra "r", ele lê o par de números (x, y) e marca a posição correspondente na matriz com o valor 1; e quando encontra a letra "f", ele entende que o arquivo terminou e encerra a leitura.

No requisito 2, a estrutura de dados escolhida foi uma matriz de adjacência, inicializada com zeros. A lógica é que as linhas representam a origem da aresta e as colunas representam o destino. Se existe uma relação de x para y, a posição matriz[x-1][y-1] recebe o valor 1 (verdadeiro); caso contrário, permanece 0 (falso).

No requisito 3, para verificar a propriedade reflexiva, o algoritmo percorre apenas a diagonal principal da matriz, onde a linha é igual à coluna (posições [i][i]). Para que a relação seja reflexiva, todo elemento deve se relacionar consigo mesmo. Se o algoritmo encontrar qualquer zero na diagonal principal, ele retorna imediatamente que a relação não é reflexiva; se chegar ao fim sem encontrar zeros, confirma que é reflexiva.

No requisito 4, a verificação da propriedade simétrica é feita percorrendo a matriz e comparando cada posição [i][j] com sua posição inversa [j][i]. A lógica é que, se existe uma ida de A para B, deve existir uma volta de B para A. Se o algoritmo encontrar qualquer caso onde o valor em [i][j] é diferente do valor em [j][i], ele conclui que a relação não é simétrica.

No requisito 5, para checar a transitividade, utilizo três laços de repetição aninhados para testar todas as combinações de três elementos (i, j, k). O algoritmo verifica a condição: se existe uma aresta de i para j e uma aresta de j para k, então obrigatoriamente deve existir uma aresta direta de i para k. Se essa aresta direta faltar em qualquer caso, a função retorna que a relação não é transitiva.

No requisito 6, o cálculo do fecho reflexivo começa criando uma cópia exata da matriz original para não perder os dados iniciais. Em seguida, o algoritmo percorre a diagonal principal dessa nova matriz e força todas as posições [i][i] a receberem o valor 1. Isso garante que todos os laços de auto-relacionamento sejam adicionados, tornando a relação reflexiva com o mínimo de alterações necessárias.

No requisito 7, para calcular o fecho simétrico, eu também copio a matriz original. O algoritmo então varre toda a matriz procurando arestas existentes (valor 1). Sempre que encontra uma aresta na posição [i][j], ele automaticamente define a posição inversa [j][i] como 1. Dessa forma, garantimos que todas as arestas tenham par de ida e volta, satisfazendo a simetria.

No requisito 8, o fecho transitivo é calculado usando um método iterativo. O algoritmo copia a matriz e entra em um ciclo de repetição que busca por pontes entre elementos: se existe caminho de i para j e de j para k, ele cria o atalho direto de i para k. Esse processo se repete continuamente até que uma passagem completa pela matriz não resulte em nenhuma nova alteração, garantindo que todas as transitividades possíveis foram encontradas.

No requisito 9, a função de saída gera um arquivo no formato DOT, compatível com o GraphViz. Primeiro, ela escreve o cabeçalho do grafo e lista todos os nós. Depois, ela percorre a matriz original para escrever as arestas pretas. Por fim, ela compara a matriz original com a matriz do fecho calculado: qualquer aresta que exista no fecho mas não exista no original é escrita no arquivo com o atributo [color=red], destacando visualmente o que foi adicionado.

Na função main, é feita toda a execução do programa. Primeiro leio os argumentos da linha de comando para pegar os nomes dos arquivos. Depois, chamo a função de leitura e verifico cada propriedade (reflexiva, simétrica, transitiva) sequencialmente. O ponto chave é o uso de condicionais: o cálculo do fecho e a geração do arquivo de saída só são acionados se a verificação da propriedade retorna falso, garantindo que o programa só trabalhe no que é necessário.