

Documentação

Trabalho Prático - Matemática Discreta

Nome: Augusto Carvalho Porto Pereira

Matricula: 2019054358

O projeto realizado pela disciplina de Matemática Discreta na UFMG consiste na análise de nós e suas relações. Ele é executado a partir da leitura de um arquivo de texto com valores de entrada, informando quais são os nós e quais as relações, e um arquivo de texto gerado pelo programa com as informações de saída, como a simetria e a transitividade.

Estrutura de Dados:

A estrutura de dados do projeto é regida principalmente pela variável inteira “numeroNos”, o vetor “listaElementos” e a matriz “matrizRelacoes”, definidas como globais no sistema. Além delas, existem as variáveis de ponteiro para tipo arquivo denominadas “arquivoEntrada” e “arquivoSaida”, responsáveis pelos arquivos de tipo texto que o programa rege.

```
FILE* arquivoEntrada;  
FILE* arquivoSaida;  
int **matrizRelacoes;  
int *listaElementos;  
int numeroNos;
```

Por se tratar de um único arquivo C, o arquivo tem como função principal a função main, que orquestra as chamadas para outras funções que inicializam as variáveis e realizam as validações importantes para o sistema.

Funções:

Para separar cada etapa em blocos diferentes do código, de modo a leitura e compreensão, as operações foram agrupadas em métodos separados da rotina main. Eles serão separados aqui, para fins de

agrupamento, entre funções de inserção de dados e verificação de relações. É importante notar que a verificação de relação de equivalência e de ordem parcial são realizadas dentro da rotina main, utilizando os retornos das funções de validação como parâmetros.

Inserção de dados:

```
> void inicializaMatrizRelacoes(){ ...
```

-

A função “inicializaMatrizRelacoes” lê o primeiro número do arquivo de entrada e o insere na variável “numeroNos”. Em seguida, são alocados dinamicamente um vetor para as variáveis “listaElementos” e “matrizRelacoes”, com o tamanho n igual ao número de nós.

Em seguida, é criado um laço for que será executado n vezes e, para cada execução, é lido um valor da primeira linha de entrada que é inserido na lista “listaElementos”. Além disso, também para cada loop do laço é criado um novo vetor na variável “matrizRelacoes”.

Por fim, são criados dois laços for aninhados, executados n vezes cada para setar os valores iniciais de cada campo da “matrizRelacoes” com o valor 0. A **complexidade assintótica de tempo** para essa função é $O(n^2)$. A **complexidade assintótica de espaço** é também $O(n^2)$ devido ao espaço reservado para a matriz principal.

```
> void preencheMatrizRelacoes(){ ...
```

-

Para cada linha do arquivo de entrada além da primeira, essa função insere no nodo correspondente da “matrizRelacoes” o valor 1, para indicar que existe uma relação do elemento i ao elemento j .

A **complexidade assintótica de tempo** dessa função é $O(n)$, sendo n igual à quantidade de linhas do arquivo de entrada, com exceção da primeira. A **complexidade assintótica de espaço** é de $O(1)$, pois a quantidade de variáveis auxiliares é sempre a mesma.

Validações de relações:

```
> int verificaReflexiva(){ ...
```

-

Para verificar a reflexividade das relações, a função verifica a diagonal da matriz e verifica se algum dos nodos possui um valor diferente de 1, indicando que a relação não é reflexiva. Retorna 1 caso seja reflexiva.

A **complexidade assintótica de tempo** dessa função é **$O(n)$** , pois o laço verifica somente a diagonal e a **complexidade assintótica de espaço** é de **$O(1)$** , pois a quantidade de variáveis auxiliares é sempre a mesma.

```
> int verificaIrreflexiva(){ ...
```

-

Para verificar se a relação é irreflexiva, o caminho realizado é similar ao da função anterior, porém são procurados campos com valor igual a 1. Retorna 1 caso seja irreflexiva.

A complexidade assintótica é igual à da função anterior, ou seja: a **complexidade assintótica de tempo** é **$O(n)$** e a **complexidade assintótica de espaço** é de **$O(1)$** .

```
> int verificaSimetrica(){ ...
```

-

Essa função verifica a simetria das relações através de dois loops aninhados, procurando alguma relação presente em um campo da matriz apontado por *matrizRelacoes[i][j]* que não esteja presente em *matrizRelacoes[j][i]*. Caso não encontre, a relação é simétrica. Retorna 1 caso seja simétrica.

A **complexidade assintótica de tempo** dessa função é **$O(n^2)$** e a **complexidade assintótica de espaço** é de **$O(1)$** , pois a quantidade de variáveis auxiliares é sempre a mesma.

```
> int verificaAntiSimetrica(){ ...
```

- A função verifica a anti-simetria da mesma maneira que a função anterior verifica a simetria, porém buscando por relações simétricas existentes. Retorna 1 caso seja anti-simétrica.

A **complexidade assintótica de tempo e de espaço** dessa função são **$O(n^2)$** e **$O(1)$** , respectivamente.

```
> int verificaAssimetrica(){ ...
```

- A assimetria é verificada comparando *matrizRelacoes[i][j]* e *matrizRelacoes[j][i]*. Caso os valores sejam iguais, a função não é assimétrica. A função retorna 1 caso seja assimétrica.

A **complexidade assintótica de tempo e de espaço** dessa função são **$O(n^2)$** e **$O(1)$** , respectivamente.

```
> int verificaTransitiva(){ ...
```

- Por fim, a transitividade é verificada por meio de 3 loops. Uma relação é transitiva se, quando *matrizRelacoes[i][j]* e *matrizRelacoes[j][z]* possuem valor 1, *matrizRelacoes[i][z]* também deve possuir valor 1. Retorna 1 caso seja transitiva.

A **complexidade assintótica de tempo** dessa função é **$O(n^3)$** devido à quantidade de laços for e a **complexidade assintótica de espaço** é de **$O(1)$** , pois a quantidade de variáveis auxiliares é sempre a mesma.

Conclusões:

A **complexidade assintótica geral de tempo do programa** é **$O(n^3)$** , devido à função “verificaTransitiva”, que possui a maior complexidade do código. A **complexidade assintótica geral de tempo do programa** é **$O(n^2)$** , devido ao espaço armazenado para a matriz de relações.

