Introdução

Neste contexto, temos dois problemas distintos a serem abordados. O primeiro problema trata da multiplicação de duas matrizes quadradas de ordem n × n, utilizando o paradigma de divisão e conquista para reduzir a complexidade computacional para O(n^3). O segundo problema envolve a implementação do algoritmo de Dijkstra, utilizado para encontrar o caminho mais curto entre dois vértices em um grafo ponderado. Para ambos os problemas, é necessário criar estruturas de dados adequadas para representar as matrizes e o grafo, respectivamente, além de implementar as operações necessárias para manipular essas estruturas.

Para o problema de multiplicação de matrizes será implementado um tipo abstrato de dados (TAD) para representar uma matriz. Serão criadas funções para criar e destruir a matriz, bem como funções para leitura, impressão e multiplicação de matrizes. Para a multiplicação de matrizes, será utilizado o paradigma de divisão e conquista, que permite reduzir a complexidade para O(n^N) ao dividir as percorrer as matrizes e fazer o produto entre a linha da primeira matriz com a coluna da segunda matriz.

Já o segundo problema serão implementados tipos abstratos de dados (TAD) para representar o grafo, as arestas e os vértices. Além disso, serão criadas funções para criar, destruir o grafo, para leitura e impressão. A entrada será fornecida por meio de arquivos contendo as instâncias do grafo. A saída corresponderá ao caminho mais curto encontrado entre dois vértices especificados. Por fim, dentro do arquivo main será implementado o algoritmo de Dijkstra para encontrar o caminho mais curto entre dois vértices.

Implementação

Sobre a implementação do algoritmo responsável pela multiplicação de matrizes podemos elencar algumas decisões tomadas, dentre elas a utilização de alocação dinâmica de memória para permitir a manipulação de matrizes de tamanho variável definido pelo usuário. Outro ponto também é a utilização de ponteiros de ponteiros para representar as matrizes, facilitando a passagem de argumentos para as funções, tendo em vista que o valor N é passado também como parâmetro. O último ponto importante é que não há tratamento de erros para entradas inválidas, isso quer dizer que se o usuário inserir um valor negativo para N ocorrerá uma falha na alocação de memória.

Sabendo desses três pontos importantes, agora será explicado o funcionamento de cada função e sua utilização dentro de todo o processamento do algoritmo. A primeira delas é a multiplyMatrices, essa função é responsável por multiplicar duas matrizes mat1 e mat2 e armazenar o resultado em result. Ela recebe as três matrizes como parâmetros, além do tamanho N das matrizes quadradas. Para fazer isso, utiliza três loops aninhados para iterar sobre os elementos das matrizes e realizar a multiplicação. O loop mais interno realiza a multiplicação dos elementos correspondentes das linhas de mat1 pelos elementos correspondentes das colunas de mat2, acumulando o resultado em result[i][j]. O desempenho dessa função é O(N^3), pois há três loops aninhados com complexidade O(N) cada.

Existe uma função auxiliar chamada printMatrix responsável por imprimir uma matriz de tamanho N x N. Utiliza dois loops para iterar sobre as linhas e colunas da matriz, imprimindo cada elemento com um espaço entre eles e uma quebra de linha ao final de cada linha. Por fim a classe main é a classe principal onde é solicitado ao usuário inserir o tamanho N das matrizes quadradas. São alocadas dinamicamente memória para as três matrizes: mat1, mat2 e result, todas de tamanho N x N. Em seguida, são lidos os elementos de mat1 e mat2 a partir da entrada padrão. Chama a função multiplyMatrices() passando as matrizes e o tamanho N como argumentos. Por fim, chama printMatrix() para imprimir a matriz resultado.

Após o uso, libera a memória alocada dinamicamente para evitar vazamentos de memória.

Para o segundo algoritmo temos também algumas decisões tomadas, a primeira delas é que foi usado listas de adjacência para representar o grafo, o que é eficiente para grafos esparsos. A segunda decisão é a utilização de alocação dinâmica de memória para permitir a manipulação de grafos com conjuntos de vértices e arestas muito grandes. Foi feito também tratamento de erros para caso haja um erro relacionado a alocação de memória. Para manter um bom desempenho do código, ao final do programa é feita a desalocação de todas as estruturas usadas liberando a memória.

Explicando agora como foi feito o algoritmo, ele se divide em basicamente 6 partes. A primeira delas, é a função imprimir que dado um grafo G imprime o grafo, exibindo os vértices e suas respectivas listas de adjacência. Outra função é a adicionaAresta() que adiciona uma nova aresta com peso entre dois vértices no grafo e verifica se os vértices de origem e destino são válidos e alocam memória para a nova aresta. A função destruirGrafo() libera toda a memória alocada dinamicamente para o grafo, incluindo os vértices e as arestas.A função criarGrafo() aloca memória para um novo grafo com um número especificado de vértices, em seguida, inicializa a estrutura do grafo, alocando memória para os vértices e inicializando a lista de adjacência de cada vértice como nula e por fim retorna um ponteiro para o grafo recém-criado.

Agora, explicaremos um pouco sobre a função responsável por implementar o algoritmo de dijkstra e suas funções auxiliares. A primeira dela é chamada de encontrarMenorDistancia() que é utilizada pelo algoritmo de Dijkstra para encontrar o vértice não visitado com a menor distância em relação à origem. No final, retorna o índice do vértice encontrado ou -1 se todos os vértices já foram visitados. Finalmente, a função dijkstra() implementa o algoritmo de Dijkstra para encontrar o caminho mais curto de um vértice de origem para um vértice de destino em um grafo ponderado. É inicializado um vetor de distâncias e um vetor de visitados, ambos com tamanho igual ao número de vértices do grafo. Além disso, é criado um vetor anterior para armazenar os predecessores de cada vértice no caminho mais curto. O algoritmo itera sobre todos os vértices, atualizando as distâncias mínimas a partir da origem e marcando os vértices visitados. Ao final, o caminho mais curto e a distância total são impressos na saída padrão.

Estudo de complexidade

Sobre o algoritmo de multiplicação de matrizes o loop externo do algoritmo de multiplicação de matrizes é executado N vezes, onde N é a ordem das matrizes. Dentro desse loop externo, há dois loops aninhados, ambos executados N vezes. Dentro desses loops aninhados, há uma operação de multiplicação de dois elementos da matriz e uma operação de soma, ambas com complexidade O(1). Portanto, a complexidade temporal total do algoritmo de multiplicação de matrizes é O(N^3). Quando falamos de complexidade espacial devemos levar em conta que a alocação de memória para as três matrizes tem complexidade O(N^2), pois cada matriz tem N^2 elementos. Logo, a complexidade espacial total é O(N^2).

Agora, sobre o algoritmo de dijkstra podemos levantar alguns pontos. O primeiro é que a inicialização dos vetores distancia, visitado e anterior tem complexidade O(V), onde V é o número de vértices. Segundo o loop principal do algoritmo de Dijkstra é executado V vezes, uma vez para cada vértice. Dentro do loop principal, a função encontrarMenorDistancia() é chamada, o que tem complexidade O(V). Em seguida, o loop interno itera sobre todas as arestas, o que pode ser, no pior caso, O(E), onde E é o número de arestas. A complexidade espacial tem complexidade O(V). Em conclusão, a complexidade temporal total do algoritmo de Dijkstra é O(V^2 + E) e a complexidade espacial total é O(V).

Quando fazemos uma comparação entre a complexidade dos dois algoritmos, temos que o algoritmo de Dijkstra tem uma complexidade temporal maior que o algoritmo de multiplicação de matrizes, principalmente porque depende do número de arestas no grafo.

Ambos os algoritmos têm uma complexidade espacial que cresce quadráticamente em relação ao tamanho dos dados de entrada, mas o algoritmo de multiplicação de matrizes tem uma constante oculta maior devido à alocação de memória para três matrizes.

Testes

Para o teste da multiplicação de matrizes foi feito a seguinte entrada

2

1 1

1 1

2 2

2 2  
  
O resultado apresentado foi

4 4

4 4  
  
Para o segundo algoritmo desenvolvido a entrada foi:  
1 2

Como saída recebemos o seguinte  
Grafo:

Caminho mais curto de 1 para 2: 2 1

Distância total: 803

Conclusão

Implementar o algoritmo de Dijkstra e o algoritmo de multiplicação de matrizes são tarefas desafiadoras que demandam um entendimento profundo das estruturas de dados e dos algoritmos envolvidos. Cada um desses algoritmos apresenta características únicas que influenciam na sua implementação e complexidade.

Ambos os algoritmos demandam um cuidadoso planejamento e uma abordagem sistemática para garantir sua corretude e eficiência. A implementação bem-sucedida desses algoritmos não apenas requer habilidades técnicas, mas também criatividade, raciocínio lógico e capacidade de análise crítica. Durante a implementação enfrentei alguns desafios mas que consegui superar me apoiando em clássicos da literatura e técnicas de execução de mesa.

Bibliografia

JURKIEWICZ, Samuel. Grafos–uma introdução. **São Paulo: OBMEP**, 2009.

JAVAID, Adeel. Understanding Dijkstra's algorithm. **Available at SSRN 2340905**, 2013.

TORRUBIA, G.; TERRAZAS, V. Algoritmo de Dijkstra. Un tutorial interactivo. **VII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2001)**, 2012.

SOFFNER, Renato. **Algoritmos e programação em linguagem C**. Saraiva Educação SA, 2017.

PEREIRA, SILVIO DO LAGO. **Algoritmos e Lógica de Programação em C–Uma Abordagem Didática**. Saraiva Educação SA, 2018.