UFMG

Algoritmos 1 – Trabalho Prático 3

Percurso Máximo da Colheita

Nome: Arthur Pontes Nader

Matrícula: 2019022294

1. Apresentação

Para otimizar o processo de colheita de maçãs, um produtor de Santa Catarina necessita de uma estratégia que garanta que o máximo de frutas possíveis sejam colhidas em um percurso. Inicialmente, sabe-se que a máquina que realiza a colheita possui determinadas limitações de movimento, o que impede que todas as árvores tenham seus frutos colhidos. Assim, o trabalho a ser realizado busca definir o maior número de maçãs que possam ser colhidas em um percurso único que respeite as restrições de movimento, bem como gerar o caminho que indique esse percurso.

Dessa maneira, esse trabalho teve como objetivo a realização da modelagem e implementação computacional de um programa que seja capaz de resolver de forma eficiente o problema exposto acima. Para isso, utilizou-se o método de programação dinâmica.

2. Modelagem computacional

Para solução do problema proposto, utilizou-se matrizes dinâmicas como estrutura de dados, pois o uso delas é um modo eficiente de obter a resposta, além de simplificar as etapas de desenvolvimento do código. A seguir, há uma breve descrição dessas matrizes, algumas observações sobre cada uma das funções implementadas e informações sobre a “main” do programa. O algoritmo de programação dinâmica utilizado será abordado no próximo tópico.

- **Matrizes:**

  Faz-se necessário o uso de duas matrizes para atingir o objetivo proposto. A primeira matriz utilizada no programa irá guardar os dados lidos na entrada. Uma coluna nessa matriz representa uma fileira de macieiras, enquanto o total de linhas indica a quantidade de árvores que

haverá em cada fileira. Além disso, cada elemento da matriz representará uma macieira, sendo que o inteiro guardado indica quantas maças estão disponíveis naquele local.

Já a segunda matriz guardará os resultados intermediários para se obter a solução final. Ela possui o mesmo formato da matriz anterior. Entretanto, cada elemento da matriz agora tem um significado diferente, pois indica a quantidade máxima de maçãs que se pode obter partindo da primeira linha até aquele local.

- **Funções:**

void calcularCaminhoMaximo(): essa função utiliza as duas matrizes criadas para realizar os cálculos que levarão à solução do problema.

std::vector<long int> recuperarCaminho(): a função tem como principal objetivo recuperar o caminho que levou ao valor máximo, retornando assim um vetor que contém os índices que representam o percurso a ser realizado.

void liberarMemoria(): já essa função recebe uma matriz e realiza a desalocação da memória.

- **Main:**

A leitura dos dados que identificarão o formato e os valores dos elementos da matriz ocorrerá na “main” do programa, com a utilização de dois laços de repetição “for”. Após isso, a função que gera a matriz de resultados é chamada. Em seguida, obtém-se o valor máximo da última linha e o seu índice, que serão utilizados para se chamar a função responsável por recuperar o caminho que levou a esse valor. Por fim, o resultado obtido é mostrado na tela, a memória utilizada é desalocada e o programa é encerrado.

3. Descrição da solução

Como expresso anteriormente, a solução do programa foi desenvolvida por meio do uso do método de programação dinâmica. Escolheu-se esse método pois ele possui a grande vantagem de não ser necessário, a cada nova linha analisada, calcular o valor de todos os caminhos possíveis até um determinado elemento. É necessário analisar somente os dois (em caso de bordas) ou três elementos adjacentes imediatamente superiores na matriz de resultados.

Optou-se por implementar as funções de maneira iterativa ao invés de recursiva, pois isso evita que muitas chamadas recursivas sobrecarreguem a pilha de execução. A seguir, encontra-se o pseudocódigo resumido e uma descrição das principais funções utilizadas para resolução do problema.

calcularCaminhoMaximo(dados, resultados)

copie a primeira linha da matriz de dados para a matriz de resultados

para cada linha da matriz de resultados

para cada elemento na linha da matriz de resultados

encontre o elemento máximo na linha superior que seja adjacente ao elemento analisado

adicione o elemento equivalente na matriz de dados ao elemento máximo encontrado

atribua o resultado ao elemento na matriz de resultados

Essa função representa a parte da solução responsável pelo método de programação dinâmica usado. A seguinte equação representa os cálculos realizados nessa etapa do processo:

$$\mathbf{resultados}[\mathbf{i}][\mathbf{j}] = \mathbf{dados}[\mathbf{i}][\mathbf{j}] + \mathbf{max}\begin{cases}\mathbf{resultados}[\mathbf{i}-\mathbf{1}][\mathbf{j}-\mathbf{1}]\\ \mathbf{resultados}[\mathbf{i}-\mathbf{1}][\mathbf{j}]\\ \mathbf{resultados}[\mathbf{i}-\mathbf{1}][\mathbf{j}+\mathbf{1}]\end{cases}$$

recuperarCaminho (dados, resultados, indiceAtual, valorAtual)

adicione o indiceAtual ao vetor de caminhos

para cada linha da matriz de dados começando da última

calcule valorAtual - elemento correspondente na matriz de dados

avalie qual dos elemento adjacentes superiores corresponde ao valor encontrado

atualize o indiceAtual apropriadamente

adicione o indiceAtual no início do vetor de caminhos

retorne o vetor de caminhos

A função acima recebe como entrada o índice e o valor máximo encontrados na última linha da matriz de resultados. Por meio de subtrações e comparações sucessivas entre as matrizes, é possível encontrar o índice na linha acima que levou ao resultado na linha atual. Faz-se isso até que chegue à primeira linha.

Dessa forma, como o índice sempre é inserido no início do vetor de caminhos, ao final do processo tem-se a sequência de índices partindo da primeira linha que levam ao valor máximo possível.

4. Análise da complexidade de tempo e espaço

Para realização da análise assintótica de cada método, define-se duas variáveis: “n” será o número de linhas da matriz e “m” o número de colunas. As funções utilizadas durante a execução são analisadas a seguir. Assim, é possível chegar a uma conclusão sobre a complexidade de tempo e espaço do programa como um todo.

- **Função - calcularCaminhoMaximo:**

  Essa função calcula um determinado valor para cada elemento da matriz de resultados. Assim, como há mn elementos, sua complexidade de tempo será O(mn).

  Nenhuma memória extra é necessária nessa parte, sendo que por isso a complexidade de espaço é O(1).

- **Função - recuperarCaminho:**

  Já essa função itera sobre as linhas da matriz de resultados. Dessa forma, sua complexidade de tempo é O(n).

  Mais uma vez, como não é necessário memória extra nessa etapa, a complexidade de espaço é O(1).

- **Função - liberarMemoria:**

  A função percorre cada uma das linhas da matriz desalocando-as. Portanto, a sua complexidade de tempo é O(n).

  A complexidade de espaço é O(1), pois nenhuma memória extra é necessária para execução.

- **Main:**

A complexidade de tempo é determinada pela maior complexidade dentre as funções chamadas durante a execução. Dessa forma, é O(mn).

Na main do programa ocorre a alocação dinâmica das duas matrizes utilizadas. Assim, cada matriz possui quantidade de elementos igual a mn, o que leva a uma utilização de espaço total igual a 2mn. Portanto, a complexidade de espaço é O(mn).

- **Conclusão**

Dessa maneira, tanto a complexidade de tempo como a de espaço do programa são O(mn).