Caixas dentro de caixas

Gabriel D. Rebello*
Faculdade Politécnica — PUCRS

30 de junho de 2024

Resumo

Este artigo mostra o passo a passo para a resolução do segundo problema proposto na disciplina de Algoritmos e Estruturas de Dados II, no terceiro semestre do curso de Engenharia de Computação, que trata de analisar as dimensões de caixas advindas de um arquivo de texto, comparando-as em suas dimensões a fim de descobrir a maior sequência possível de caixas dentro de outras caixas.

Para isso, foi desenvolvida uma proposta de resolução trabalhando com ordenação, bem como as dimensões das caixas são ordenadas, a fim de analisa-las mais facilmente e conseguir então achar o maior caminho por meio de um algoritmo de grafos.

Introdução

Dentro do escopo da disciplina de Algoritmos e Estruturas de Dados II, o problema a ser solucionado tem a seguinte descrição: você decidiu dar um presente criativo à sua sogra e, para isso, teve a ideia de comprar várias caixas e colocá-las uma dentro da outra, para que ela abra uma por uma e ache o presente dentro da última delas, então, para realizar esta ideia, pesquisou na internet o catálogo de uma fabricante de caixas de papelão mas, antes de compra-las, decidiu achar a maior sequência de caixas que poderão ser colocadas uma dentro da outra.

Porém, ao se deparar com o catálogo, percebeu que o mesmo é composto apenas por uma lista de caixas e seus respectivos valores de altura, largura e comprimento, mas sem fornecer qual dos dados seria cada uma das dimensões. Como exemplo para elucidar, temos a seguinte sequência de caixas:

991 443 126

733 281 710

910 720 218

Com isso a tarefa é simples, analisar os valores fornecidos e identificar a maior sequência dentre eles, sem que nenhuma dimensão da caixa posterior seja superior a mesma dimensão da caixa anterior.

* g.rebello@edu.pucrs.br, turma 11

Para resolver o problema proposto, analisaremos uma possível alternativa de solução, bem como suas particularidades, demonstrando sua eficiência e abordagem central, com uma metodologia relativamente simples e intuitiva. Juntamente a isso, serão apresentados os resultados finais obtidos com base nos casos de teste previamente fornecidos, bem como o tempo de execução em cada um dos casos e a complexidade do algoritmo.

Solução

Depois de considerar o problema, foi elaborado um código capaz de ler um arquivo de texto recebido que conterá as caixas, bem como transportar estas caixas para um vetor que as armazena, usando uma struct capaz de armazenar as dimensões da caixa, que são inseridas previamente ordenadas por suas dimensões, para então serem analisadas pelo programa principal. Para tal abordagem, o código foi feito de maneira modular, usando uma função capaz de ler o arquivo e transportá-lo para um vetor, bem como uma função auxiliar para comparar as dimensões entre caixas e uma função implementando o algoritmo DFS para grafos, que tem a funcionalidade de achar a maior sequência de caixas. Sendo assim, o código será demonstrado em partes, mostrando primeiramente suas funções e, após isso, a parte principal. Começaremos com a função de ler o arquivo:

```
funcao LerArquivo():
   arquivo = recebe(“arquivo.txt”)
   se arquivo.abriu() é FALSE então
       exibe “Erro ao ler arquivo”
       sai do programa
   fim
   vetor[Caixa] caixas
   string linha
   enquanto arquivo tem linha
       linha.recebe(dimensao)
       Caixa caixa
       se linha tem dimensao
            vetor[int] dimensoes = caixa.dimensao
            ordena(dimensoes)
            caixas.recebe(caixa)
       fim
   fim
   retorna mapa
fim
```

A função “LerArquivo” é simples, criando uma variável que receberá o endereço do arquivo e colocando os dados que esse arquivo contém em um vetor, fazendo verificações básicas para garantir que o programa terá feito a abertura e o fechamento do arquivo de maneira correta, a fim de não comprometer o andamento do programa, bem como a organização das dimensões das caixas, da menor até a maior, para auxiliar posteriormente na comparação entre caixas feita pela função seguinte. A seguir temos a função que compara as caixas:

```
funcao comparaCaixas(caixa_a, caixa_b):
   se caixa_a.dimensoes < caixa_b.dimensoes então
      retorna verdadeiro
   se não
      retorna falso
fim
```

O princípio da função é bem simples, ela recebe duas caixas com suas respectivas dimensões e as compara, a fim de descobrir se a caixa_b é maior que a caixa_a e, caso positivo, retorna verdadeiro, ou caso contrário, retorna falso.

A simplicidade desta função se dá pelas caixas terem suas dimensões previamente ordenadas na função que lê o arquivo, sendo necessário aqui apenas compará-las, funcionalidade necessária para a implementação da função principal, mas antes de chegarmos na função principal, veremos a implementação de outra função usada nesta, sendo ela a implementação do algoritmo DFS (Depth First Search), um algoritmo que busca em profundidade para grafos, que auxiliará a descobrir a maior sequência possível entre caixas:

```
funcao DFS(nodo, adjacentes, comparadas, depth):
   comparadas[nodo] = verdadeiro
   para cada vizinho em adjacentes[nodo]
        se vizinho nao foi visitado entao
             chama DFS(vizinho, adjacentes, comparadas, depth)
        depth[nodo] = máximo entre depth[nodo] e depth[vizinho] + 1
   fim
fim
```

A função “DFS” tem uma certa complexidade, nela é feita iteração sobre os nodos vizinhos do nodo recebido previamente, marcando-os quando são visitados e então fazendo uma chamada recursiva para cada nodo vizinho ainda não visitado, assim então calculando o máximo entre o valor atual de profundidade e novo valor de profundidade buscado ao analisar o vizinho, para conseguir a maior sequência até o momento. Esse algoritmo se torna útil dentro da função principal, que o usará no vetor que contém as caixas com seus tamanhos comparados.

A seguir temos a função principal:

```
funcao main():
   caixas = lerArquivo()
   int n = caixas.tamanho
   vetor[int] adjacentes(n)
   de i igual a 0 até i igual a n faça
       de j igual a 0 até j igual a n faça
           se comparaCaixas(caixa[i], caixa[j]) for verdadeiro então
               adjacentes[i].recebe(j)
   fim
   vetor[booleano] comparadas(n, falso)
   vetor[int] depth(n, 1)
```

```
        de i igual a 0 até i gual a n faça
            se comparadas[i] for falso então
                DFS(i, adjacentes, comparadas, depth)
        fim
        int maior_sequencia = máximo elemento de depth
    fim
```

A função “main” apenas cria alguns vetores para serem trabalhados nas funções auxiliares, alinhando as caixas em suas adjacências para que então possam ser comparadas e trabalhadas na função “DFS” a fim de encontrar a maior sequência possível.

Resultados

Depois de implementar o algoritmo acima em linguagem C++ e executar o mesmo nos casos de teste fornecidos, executando-os dez vezes e fazendo uma média de tempo de execução, chegamos aos seguintes resultados:

Caso de teste	Maior sequência	Tempo de execução (ms)
caso00010	3	0.0094
caso00020	5	0.0432
caso00050	8	0.1436
caso00100	12	0.4683
caso00200	17	1.6324
caso00300	20	3.2691
caso00500	24	8.1763
caso01000	32	32.681
caso02000	42	128.284
caso05000	59	749.489
caso10000	76	2846.51

Figura 1 - Casos de teste

Após a obtenção destes valores, foi montado um gráfico com a relação dos casos de teste e o tempo de execução dos mesmos, para conseguirmos fazer uma melhor análise da relação do tamanho do vetor com o tempo de execução dos testes:

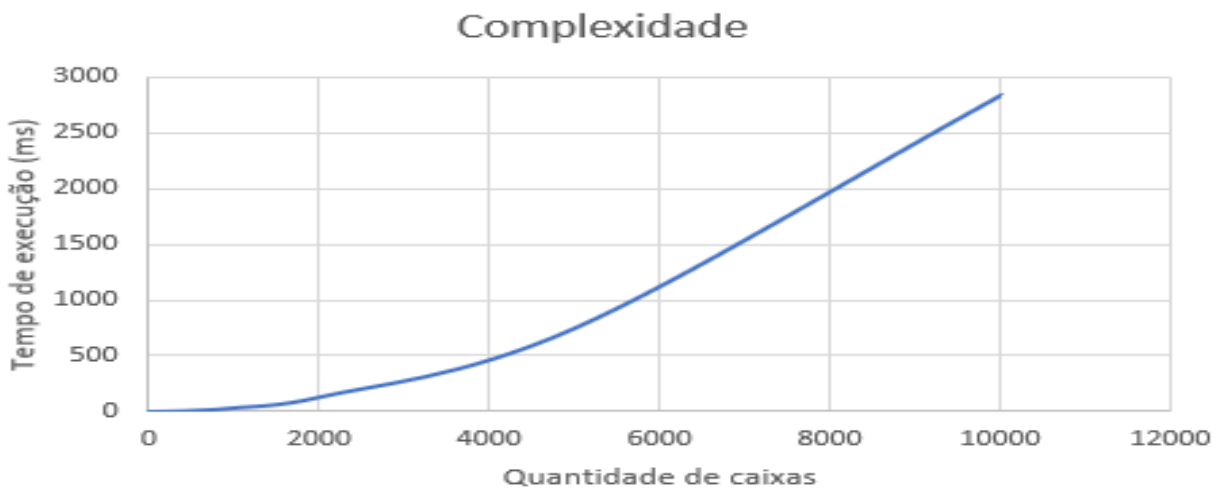

Figura 2 - Complexidade do algoritmo

Visualizando o gráfico em que o eixo y é composto pelo tempo de execução e o eixo x composto pela quantidade de caixas em cada catálogo, é possível perceber que o algoritmo implementado é de complexidade $O(n^2)$, sustentado pelo fato de o algoritmo necessitar aplicar a função DFS para cada caso em que o vizinho do nodo atual ainda não foi visitado.

Conclusões

A abordagem escolhida para resolver o problema da leitura de um mapa foi o uso de um vetor de structs capaz de armazenar as dimensões das caixas, que são inseridas no vetor com suas dimensões ordenadas, a fim de simplificar o trabalho das outras funções e, juntamente com isso, a aplicação de um algoritmo de DFS para conseguir descobrir de maneira simples a maior sequência possível de caixas, fornecendo um algoritmo de complexidade $O(n^2)$, portanto sendo um algoritmo eficiente e capaz de lidar com a problemática apresentada, fornecendo os dados corretos com base nos casos de testes e valores finais fornecidos.

Assim, foi possível implementar uma das soluções possíveis representando a metodologia de análise de profundidade com grafos, de maneira eficiente e correta.