Trabalho Prático 3
Algoritmos 1

Vinicius Silva Gomes

Universidade Federal de Minas Gerais
vinicius.gomes@dcc.ufmg.br

11 de julho de 2022

Resumo

Este artigo descreve os passos para a solução de um problema de escolha da maior mesa que cabe em uma casa, uma vez que a planta dessa casa seja conhecida. O objetivo é desenvolver um programa que seja capaz de identificar os espaços disponíveis na planta da casa e escolher a maior mesa se adeque nesses espaços. Será documentado desde a análise do problema e suas entradas até a solução alcançada e suas propriedades.

1 Introdução

O enunciado contextualiza um problema relacionado a escolha de mesas para a casa da Vó, que sofreu uma reforma para que muitas reuniões de família possam acontecer após a pandemia. É solicitado o desenvolvimento de um software que, dado a planta da casa e um conjunto de mesas (descritas por suas dimensões) previamente escolhidas, encontre qual a maior mesa que pode ser colocada em algum cômodo da casa da Vó.

A planta da casa já foi transformada do 3D para uma representação no terminal (que facilita a manipulação via software) e no conjunto de dimensões das mesas, o espaço destinado para as cadeiras já foi considerado, o que simplifica o processamento necessário para identificar qual a maior mesa que cabe na casa. Dessa forma, resta ao software computar o maior espaço livre na casa e identificar quais mesas cabem nesse espaço e qual delas aproveita ele da melhor forma.

Baseado nisso, este artigo tem como objetivo documentar a solução para este problema, enfatizando como foi feita a Modelagem Computacional do cenário descrito, quais as características e algoritmos foram utilizados na Implementação e, por fim, qual a Complexidade Assintótica desses algoritmos e da solução como um todo.

2 Modelagem Computacional

Para modelar o problema, inicialmente, foi preciso analisar bem a entrada que ele fornece, que tipo de informações deverão ser computadas e qual o tipo de saída é esperada. A entrada é composta pela planta da casa da Vó, adaptada para ser representada num terminal de computador; e as dimensões das mesas escolhidas pela Tia. Dado isso, é preciso fazer um programa que compute e informe qual a maior mesa, presente nessa lista, que cabe na casa da Vó, após a reforma.

Uma forma prática de representar essa planta seria através de uma matriz. Isso facilitaria muito as manipulações necessárias, para que seja possível calcular as maiores áreas e identificar qual a melhor mesa a ser escolhida. No entanto, a forma como essas áreas serão computadas precisa ser bem pensada, evitando que o problema escale para uma solução com complexidade temporal/espacial ruim.

A escolha da matriz para representar a planta também favorece outro ponto, a existência de vários algoritmos para calcular áreas em matrizes. No entanto, só isso não será o bastante para que o algoritmo compute corretamente o que o problema pede. Dessa forma, a abordagem inicial será se inspirar em algum desses algoritmos e tentar modificá-lo para que ele realize a computação que precisamos.

Para aplicar o algoritmo, a matriz com a planta da casa foi transformada para inteiros, ao invés de caracteres, sendo '.' = 1 e '#' = 0. Assim, é possível transformar cada linha da matriz em um histograma cumulativo das linhas anteriores e, com isso, identificar qual a maior área na matriz se resume a identificar qual o retângulo de maior área no conjunto de histogramas. Essa estratégia, apesar de parecer pouco intuitiva

a princípio, faz bastante sentido quando pensamos que construir histogramas cumulativos com os valores das linhas anteriores ("gráfico de barras") é uma forma de medir a "altura" e contar a quantidade de barras seria uma forma de medir a "largura" de retângulos ao longo da matriz. Assim, diversos retângulos poderiam ser estipulados e comparados até que o maior seja obtido. Essa é uma abordagem característica de algoritmos de Programação Dinâmica e é uma das formas mais clássicas de resolver problemas de área em matrizes com inteiros (matriz binária, especificamente para esse caso).

No entanto, existem alguns casos onde a maior mesa não necessariamente cabe na maior área, por possuir alguma de suas dimensões maior que a do maior retângulo, mas cabe em alguma área menor que é calculada ao longo da execução do algoritmo. Para esses casos, considerar apenas a maior área seria um grande equívoco e faria o algoritmo falhar. Portanto, para corrigir esse problema, ao invés de procurar pelo maior retângulo nos histogramas, todas as áreas calculadas serão salvas. Desse modo, elas poderiam ser comparadas com as mesas e a verdadeira maior mesa que cabe na casa seria encontrada.

Dessa forma, após definir qual a melhor forma de representar a planta da casa e traçar qual estratégia será usada para computar o que o problema pede, podemos discutir sobre como o algoritmo soluciona o problema proposto e sobre os aspectos específicos da implementação do programa, na seção 3.

3 Implementação

O programa foi desenvolvido na linguagem **C++** e compilado pelo compilador **G++**, da **GNU Compiler Collection**. Algumas variáveis de entrada aparecerão com frequência nos pseudocódigos: N e M, as dimensões da planta da casa; e K o número de mesas escolhidas pela Tia. A seguir, os principais algoritmos do programa serão discutidos.

3.1 `main()`

A função `main` recebe uma entrada do problema, armazenando a planta da casa e as mesas escolhidas pela Tia em matrizes do tipo `vector<vector<int>>` e `vector<pair<int, int>>`, respectivamente. Após isso, ela chama a função `find_largest_table()`, que computa qual a maior mesa, dentre aquelas informadas pela Tia, que cabe na casa e exibe as dimensões (comprimento e largura) dessa mesa.

Para representação da planta da casa, foi utilizado uma matriz de 2 dimensões compostas pela estrutura `vector`. O vetor realiza o acesso a uma posição em $\mathcal{O}(1)$ e oferece alguns facilitadores, como os métodos de manipulação dos seus elementos, que já foram previamente implementados, etc.

Para representar as mesas escolhidas pela Tia, a estrutura `pair` também será interessante pois torna a representação do comprimento e da largura de uma mesa mais semântica e prática no código.

3.2 `interpolate_tables_and_areas()`

É uma das principais funções da solução. Ela realiza uma intercalação entre as mesas selecionadas e os retângulos obtidos através da planta da casa. Para tornar esse método mais rápido, tanto as mesas quanto as áreas já estão ordenadas pela maior área/maior largura. No pior caso, ainda serão necessárias $\mathcal{O}(N \cdot M \cdot K)$ iterações, mas esse número será bem menor para casos mais simples de instâncias do problema.

O funcionamento do método é simples, sendo baseado em uma série de comparações iterativas das mesas com as áreas obtidas. Caso a *i-ésima* mesa tenha área maior que a *j-ésima* área, a próxima mesa é escolhida e a iteração pelas áreas é reiniciada. Caso a *i-ésima* mesa tenha dimensões maior que a *j-ésima* área mas caiba nessa área *j*, a próxima mesa *i* é escolhida. Por fim, caso a *i-ésima* mesa caiba na *j-ésima* área, considerando tanto a área total quanto as dimensões, essa mesa será retornada pelo algoritmo e será a mesa procurada pela solução do problema. Ela é a maior mesa que cabe nos cômodos da casa.

O pseudocódigo 1 mostra esse algoritmo que compara as áreas e as mesas e escolhe a melhor mesa possível.

3.3 `find_largest_table()`

Esse é o principal algoritmo do programa, sendo ele o responsável por montar o conjunto de histogramas definido na modelagem e calcular as áreas dentro da planta da casa. Esses histogramas serão armazenados em uma matriz $N \times M$ do tipo `vector<vector<int>>`. Primeiramente, os histogramas são montados percorrendo

Algorithm 1 Computa a maior mesa que cabe no conjunto de áreas calculadas

Input: Tamanho K das mesas selecionadas, conjunto A de áreas ordenadas pela área e conjunto T de mesas ordenadas pela área e pela largura
Output: Par comprimento-largura da maior mesa que cabe na casa da Vó

```
i ← 0
j ← 0
while i < |A| e j < K do
    area ← i-ésima área calculada A[i]
    table ← área da j-ésima mesa T[j]
    if area ≥ table then
        if TABLE-FITS(AREA, TABLE) then
            return (width, length)
        else
            i ← i + 1
        end if
    else
        j ← j + 1
        i ← 0
    end if
end while
return (−1, −1)
```

toda a planta da casa e aplicando a seguinte regra para cada célula da matriz: se a célula for 1, somamos seu valor com a célula presente na mesma coluna mas na linha anterior do histograma, se 0 o valor no histograma também será 0. Podemos traduzir essa estratégia em uma equação que descreve o comportamento da criação desse histograma, a *equação de Bellman*. Chamando a matriz da planta da casa de A e a matriz que contém os histogramas por H, teremos:

$$H[i][j] = \begin{cases} A[i][j] + H[i-1][j], & \text{se } A[i][j] = 1 \\ 0, & \text{caso contrário} \end{cases}$$

Ao final das iterações, os histogramas terão sido corretamente calculados. Com isso em mãos, é possível percorrer cada histograma individualmente e calcular as áreas que podem ser obtidas a partir deles. O algoritmo para calcular a área, então, vai definir uma pilha que armazena as posições onde as barras do histograma se iniciam e vai iterar por todas as barras presente no histograma daquela linha. Caso a altura da barra da iteração atual seja maior que a da iteração anterior, essa nova barra será inserida na pilha para que um "novo retângulo" possa ser calculado (sucessivos possíveis retângulos serão calculados até que as iterações finalizem, no qual cada retângulo é iniciado por um dos elementos presentes na pilha).

Caso a barra atual seja menor que a anterior, um possível retângulo terá sua área calculada. O comprimento desse retângulo será definido pela iteração atual e pelo último item inserido na pilha (pilha armazena posições, logo a relação entre a iteração atual e os itens inseridos na pilha definem o deslocamento, que por sua vez determina o comprimento do retângulo). A largura, por outro lado, é definida pela menor barra presente dentro desse deslocamento. Assim, ao multiplicar esses valores, conseguimos obter uma área intermediária dentro da planta da casa. Esse processo é repetido até que todas as barras presentes no histograma daquela linha sejam analisados.

No entanto, pode ser que mesmo que as iterações pelo histograma tenham terminado, ainda existam elementos na pilha. Dessa forma, o passo de calcular as áreas é repetido enquanto a pilha tiver algum elemento.

Cada área calculada nessa parte do algoritmo será armazenadas num vetor, do tipo `vector<pair<int, int>>`, para que possam ser comparadas com as mesas disponíveis e a melhor mesa para as configurações de áreas obtidas possa ser selecionada.

Após obter as áreas, elas e as mesas selecionadas são ordenadas de maneira decrescente seguindo o critério da maior área (multiplicação das componentes do par) e da maior largura (segunda componente do par).

Após a ordenação, a função `interpolate_tables_and_areas()` é chamada, passando esses vetores ordenados como parâmetros e o resultado da função é retornado para a `main()`. Dessa forma, a melhor mesa, dado uma instanciação do problema, terá sido obtida.

O pseudocódigo 2 mostra esse algoritmo que computa os histogramas e as áreas, dado a planta da casa.

Algorithm 2 Computa os histogramas e as possíveis áreas na planta da casa

Input: Dimensões N e M da planta da casa, tamanho K das mesas selecionadas, matriz da planta da casa X e vetor com as mesas selecionadas B
Output: Par comprimento-largura da maior mesa que cabe na casa da Vó

```
H ← ∅
U ← ∅
for i ← 0 até N do
    for j ← 0 até M do
        if i = 0 then
            H[i][j] ← X[i][j]
        else
            if X[i][j] = 0 then
                H[i][j] ← 0
            else
                H[i][j] ← H[i − 1][j] + X[i][j]
            end if
        end if
    end for
    S ← ∅
    k ← 0
    while k < M do
        if S ≠ ∅ ou H[i][Top(S)] ≤ H[i][k] then
            S ← k
            k ← k + 1
        else
            area ← Calculate-Area(S, H)
            Pop(S)
            U ← area
        end if
    end while
    while S ≠ ∅ do
        area ← Calculate-Area(S, H)
        Pop(S)
        U ← area
    end while
end for
T ← Sort(B)
A ← Sort(U)
table ← Interpolate-Tables-And-Areas(K, A, T)
return table
```

Observação: A função Calculate-Area(S, H) existe somente no pseudocódigo, para facilitar a visualização do que está acontecendo no código e não poluir o que, na teoria, deve ser uma representação simplificada do que o código faz. No código original, o cálculo da área é feito de forma mais clara e explícita.

3.4 `Funções auxiliares`

Ao longo do código, algumas funções auxiliares foram utilizadas para compor funções maiores no programa. Esta subseção vai listar cada uma delas e resumir a sua utilização dentro do programa.

- `sort_by_area_and_width()`: sobrescreve a função de comparação da função `sort()`, da biblioteca STL, para que ela ordene os pares de maneira decrescente, baseando-se no cálculo da área (multiplicação das componentes do par) e na largura da mesa (segundo parâmetro do par);
- `table_fits()`: função que compara se uma mesa informada cabe em um retângulo também informado. A função realiza a comparação tanto da mesa em sua orientação original quanto da mesa rotacionada 90 graus e retorna `true`, caso a mesa caiba em algum cômodo da casa, e `false` caso contrário.

4 Análise de Complexidade

Para definir a complexidade do algoritmo, serão usados alguns padrões para se referir as variáveis do problema. Iremos denotar N e M como as dimensões da matriz que compõem a planta da casa; e denotaremos K como o número de mesas de interesse a serem comparadas e a maior delas que cabe em algum cômodo da casa seja encontrada.

Os algoritmos foram separados e suas complexidades serão analisadas individualmente, a princípio. Ao final, será dado a complexidade total do algoritmo implementado que resolve o problema proposto.

- Para **calcular o histograma** e **calcular as áreas presentes na planta da casa**, as tarefas mais custosas são percorrer a matriz da planta da casa, passando por cada linha no máximo 3 vezes. Isso acontece pois, para calcular o histograma, cada linha é percorrida uma vez e, para calcular as áreas, a pilha pode fazer com que o algoritmo passe por cada linha 2 vezes (uma vez empilhando todos os elementos e outra para desempilhá-los depois, considerando o pior caso). Dessa forma, a complexidade assintótica para executar essa atividade é $\mathcal{O}(N \cdot 3M) \in \mathcal{O}(N \cdot M)$.
- A entrada para a função `interpolate_tables_and_areas()` precisa ser tratada antes da chamada da função. Assim, é necessário **ordenar os vetores de mesas e de áreas calculadas**. A complexidade dessas ordenações, considerando o pior caso para cada uma delas é $\mathcal{O}(K \log K)$ e $\mathcal{O}((N \cdot M) \log (N \cdot M))$ (pois $N \cdot M$ é o número máximo de áreas que podem existir na planta), para ordenar as mesas e as áreas, respectivamente.
- A função que **intercala as áreas e as mesas** e obtém qual a melhor mesa para essa instância do problema executa, por mais que de maneira implícita, dois *for loop's*. Para o pior caso, um deles vai de 0 a K (percorre todas as mesas selecionadas pela Tia) e o outro vai de 0 até $N \cdot M$ (percorre todas as possíveis áreas na planta da casa). Dessa forma, a complexidade total para esse trecho, considerando o pior caso, é $\mathcal{O}(N \cdot M \cdot K)$.

Dessa forma, ao analisar o programa com todos esses algoritmos, é possível constatar que ele possui complexidade de $\mathcal{O}(N \cdot M + K \log K + (N \cdot M) \log (N \cdot M) + N \cdot M \cdot K) \in \mathcal{O}(N \cdot M \cdot K)$.

5 Compilando o Código

Para compilar o código, foi usado um `makefile` com o código:

```
all:
    g++ main.cpp -Wall -o tp03
```

Desse modo, caso o `makefile` não esteja disponível no ambiente de testes, o comando `g++ main.cpp -Wall -o tp03` deverá ser utilizado para compilar, e o comando `./tp03 < ./test_cases/1.txt` para executar o programa com alguma entrada de exemplo "1.txt".