Trabalho Prático 3 em Algoritmos 1

Gustavo Freitas Cunha

Matrícula: 2020054498

Departamento de Ciência da Computação - Instituto de Ciência Exatas -
Universidade Federal de Minas Gerais
Belo Horizonte - MG - Brasil

gustavocunha@dcc.ufmg.br

1. Modelagem

Para resolução do problema proposto, a casa da avó, representada pela planta arquitetônica, é aqui modelada como uma matriz de caracteres, denominada planta, de dimensões N por M. Ainda, uma mesa foi aqui abstraída como uma struct, chamada Retangulo[1]. As mesas pelas quais a avó gostou são armazenadas em um deque, chamado mesas.

O problema proposto consiste em encontrar, num primeiro momento, todas as áreas disponíveis - acumulado de caracteres '.', conforme define o enunciado - na planta da casa (a matriz planta). Essas áreas são armazenadas em um deque chamado retangulosLivres. Em seguida, é necessário verificar, para cada mesa, a maior área na qual ela se encaixa. Para tanto, o deque mesas e de retângulos são ordenados de forma decrescente por suas áreas, sendo o critério de desempate a maior largura, conforme solicitado na proposição do problema. A função selecionaMaiorRetangulo, após encontrar todas as áreas disponíveis em planta, faz a busca, para cada mesa, da maior área livre disponível que a caiba.

Para as áreas livres na planta, converte-se a matriz planta em uma matriz binária, utilizando o seguinte critério: os caracteres '#' são substituídos por 0 e os caracteres '.', substituídos por 1. Em seguida, utiliza-se um algoritmo que encontra a maior área disponível de forma gradual, isto é, percorrendo as linhas da matriz, armazenando os resultados parciais a cada passo: claramente um algoritmo de programação dinâmica. Mais detalhes sobre esse algoritmo na seção 3.

2. Estruturas de Dados

A representação computacional das mesas e áreas livres, conforme foi dito na seção 1, é feita através de uma struct, denominada Retangulo. Essa struct apresenta dois inteiros: um denominado comprimento que representa, obviamente, o comprimento da mesa, e outro chamado largura, cujo nome é, também autoexplicativo. Há, ainda uma função area, que retorna a multiplicação do comprimento pela largura do Retangulo.

[1] Mais detalhes sobre estruturas de dados na seção 2.

Outra estrutura de dados utilizada foi o deque, biblioteca padrão de C++, é semelhante à uma lista encadeada e permite inserções e remoções no início e no fim. Tal estrutura armazena as mesas que são dadas como entrada do problema e também as áreas livres encontradas em planta.

Por fim, foi utilizada também uma pilha de inteiros - stack<int>, da biblioteca padrão de C++ - no algoritmo que encontra as áreas disponíveis na planta. Essa pilha tem como finalidade definir os limites dos retângulos livres na matriz binária que representa a casa da avó, armazenando os índices matriciais (ou melhor, vetoriais) desses limites. Mais detalhes na próxima seção.

3. Algoritmos

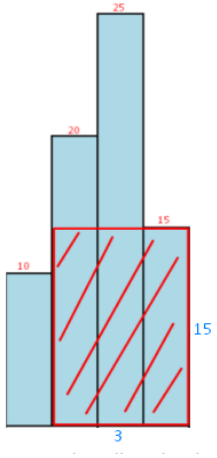
- Encontrando as áreas livres na planta da casa da avó

Após o processamento da planta da casa e das mesas de interesse da avó, é chamado o método selecionaMaiorRetangulo que, por sua vez, chama encontraAreasDisponiveis que, conforme o próprio nome indica, é o método responsável por encontrar as áreas livres na planta da casa. Esse método itera sobre as linhas da matriz binária da planta da casa (chamada matrizBinaria), acumulando, ao longo das posições matrizBinaria[i][j], cujo valor seja 1, a soma matrizBinaria[i-1][j] + matrizBinaria[i][j]. Daí, são encontradas todas as áreas de retângulos disponíveis analisando esta linha, mas somente após seus valores já serem os acumulados das linhas acima (onde o valor era, inicialmente, 1). As áreas disponíveis são encontradas da seguinte forma: cada número representa o comprimento de um retângulo cuja largura é sempre 1. Esse processo é feito na chamada da função encontraAreasParciais. Veja um exemplo:

Suponha que, em uma determinada linha L de matrizBinaria, com tamanho M = 4 e um N qualquer, que já passou pelo processo de acumular os pesos retroativos, tenhamos:

10 20 25 15

Teremos o seguinte retângulo:



Em fonte vermelha, o comprimento de cada retângulo, equivalente a cada elemento de L. Cada retângulo possui largura 1.

Em fonte azul, as dimensões do retângulo hachurado, de maior área.

Note que o maior retângulo possível (hachurado na figura) tem área 45: comprimento 15 e largura 3.

Além do maior retângulo, todos os outros possíveis retângulos com área disponíveis são explorados e armazenados no deque retangulosLivres.

O pseudocódigo do algoritmo para encontrar esses retângulos segue abaixo:

```
encontraAreasParciais(linha[], M, retangulosLivres[]){
   //linha é uma das linhas da matriz planta e M é o tamanho de linha
   P := pilha vazia
   indice := 0
   while(i<M){
          //pilha recebe os limites dos retângulos
          if(P não está vazia || linha[topo de P] <= linha[i]){
                 adiciona i à P
                 incrementa i de 1
          }
          //calculo da area local
          else{
                 if(P não está vazia){
                        areaLocal := linha[topo de P]*i
                 }
                 else{
                        areaLocal := linha[topo de P]*[i-(topo de P)-1]
                 }
                 if(areaLocal != 0){
                        adiciona areaLocal à retangulosLivres
                 }
          }
   }
   //calcula da área local para os componentes restantes na pilha
   while(P não está vazia){
          if(P não está vazia){
                 areaLocal := linha[topo de P]*i
          }
          else{
                 areaLocal := linha[topo de P]*[i-(topo de P)-1]
          }
          if(areaLocal != 0){
                 adiciona areaLocal à retangulosLivres
          }

   }
}
```

4. Análise de Complexidade

A solução implementada possui complexidade assintótica $O(max(K * log(K), NM * log(NM), NMK))$, sendo N o número de linhas da matriz que representa a planta da casa da avó, M o número de colunas desta matriz e K o número de mesas que agradaram a avó.

A função encontraAreasParciais, que é a implementação do algoritmo descrito na seção anterior, possui complexidade $O(M)$, já que cada elemento do vetor que é recebido como parâmetro, que **possui sempre tamanho M**, é adicionado e removido da pilha **apenas uma vez** e as operações feitas com ele são constantes.

Na função encontraAreasDisponiveis temos $O(NM)$ para gerar matrizBinaria e, em seguida, dois loops em N e M aninhados para fazer a soma cumulativa das linhas da matriz binária, mas note que encontraAreasParciais é chamado a cada iteração do loop externo. Sendo assim, temos, para esta função:

$$O(NM) + O(N * (O(M) + O(M))) = O(NM).$$

Atente para o fato de que o número de áreas disponíveis em um retângulo M por N pode ser da ordem de até NM.

A função selecionaMaiorRetangulo possui uma chamada para encontraAreasDisponiveis - $O(NM)$ -, em seguida, uma ordenação do vetor de mesas - $O(K * log(K))$ -, uma ordenação sobre o vetor de áreas - $O(NM * log(NM))$ - e dois loops aninhados, que fazem as combinações entre mesas e áreas disponíveis -$O(NMK)$:

$$O(NM) + O(K * log(K)) + O(NM * log(NM)) + O(NMK) = O(max(K * log(K), NM * log(NM), NMK)).$$

Note que como, por definição do enunciado, $1 \leq N, M \leq 1000$ e $1 \leq K \leq 10^6$, não podemos assumir que NM > K.

Finalmente, na função main, teremos o processamento da planta da casa da vó - $O(NM)$ -, o processamento das mesas - $O(K)$ - e uma chamada para selecionaMaiorRetangulo - $O(max(K * log(K), NM * log(NM), NMK))$. Assim, a complexidade final deste programa fica:

$$O(NM) + O(K) + O(max(K * log(K), NM * log(NM), NMK)) = O(max(K * log(K), NM * log(NM), NMK)).$$

5. Instruções para Compilação e Execução

O arquivo único, main.cpp, de código desta solução está no diretório raiz TP. Para compilação e execução, siga os passos a seguir:

Primeiro, **com o terminal no diretório TP**, execute o seguinte comando, para compilação do código fonte e geração do arquivo em código de máquina, de nome tp03:

```
g++ -std=c++17 -lstdc++fs -Wall main.cpp -o tp03
```

Na sequência, com o terminal no mesmo diretório, entre com o comando a seguir, para execução:

```
./tp03 < <nome do arquivo de entrada>.txt
```

É necessário que o arquivo de entrada também esteja no diretório TP, caso contrário, é preciso indicar seu caminho no terminal.

Conforme solicitado, os resultados serão impressos na saída padrão (stdout).

Por motivos de possível incompatibilidade na arquitetura de computadores, o arquivo executável gerado em meu computador não foi entregue, sendo gerado por você ao compilar o código.