Trabalho Prático

Gabriel Teixeira Carvalho

1. Introdução

Esta documentação lida com o problema de descobrir a maior mesa retangular que pode ser colocada em uma casa, dada uma planta quadriculada do espaço. Para resolver o problema citado, foi utilizado um algoritmo simples em cima da matriz que representa o espaço da casa onde a mesa escolhida será colocada.

A seção 2 desta documentação trata sobre a modelagem computacional do problema, enquanto na seção 3 são apresentadas as estruturas de dados e algoritmos utilizados e o pseudocódigo do sistema. Por fim, na seção 4 são apresentadas análises de complexidade assintótica de tempo, seguida por um manual de compilação e execução do programa.

2. Modelagem computacional do problema

A modelagem computacional desse problema parte da conversão da planta dada para uma matriz/vetor de vetores. Então, partindo da primeira linha e atualizando os valores a cada linha da matriz, as alturas máximas das colunas acima das linhas são salvas em um vetor. Caso o elemento i,j da matriz seja um '.', o elemento j do vetor recebe o valor de vetor[j] + 1. Caso contrário, ele recebe 0. Dessa forma, é possível saber o tamanho das colunas que estão sobre as linhas e achar os maiores retângulos para cada linha de forma eficiente. No exemplo abaixo, estão os retângulos encontrados na 3ª linha da matriz.

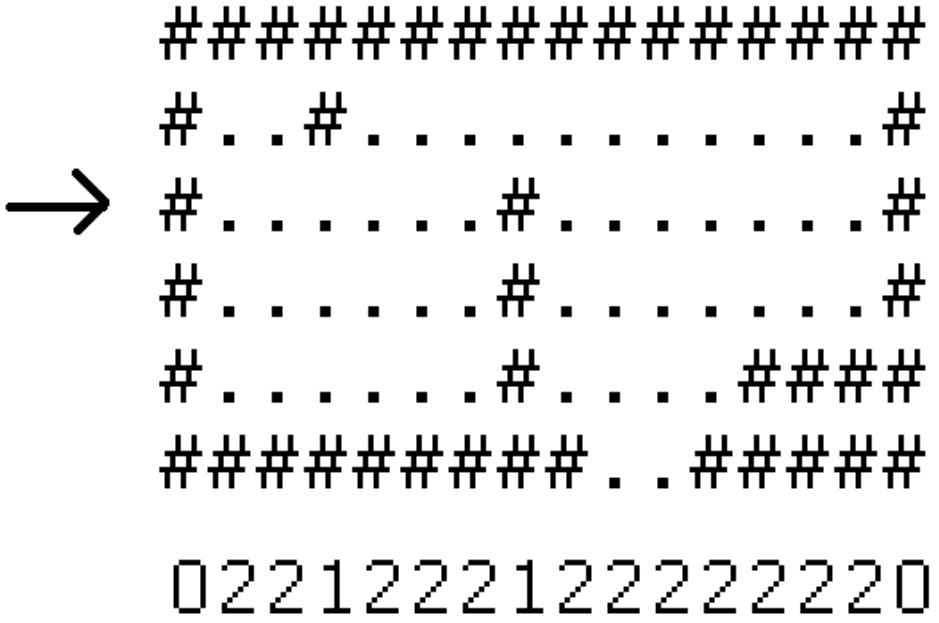

Figura 1: Exemplo de execução para a 3ª linha

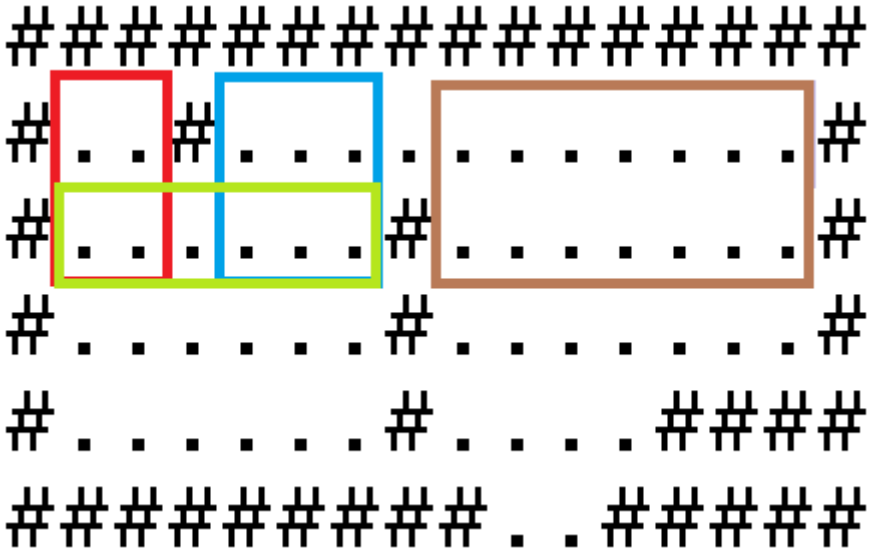

Figura 2: Retângulos encontrados na 2ª linha

3. Estruturas de dados e algoritmos utilizados

Neste trabalho prático utilizei das seguintes estruturas de dados e algoritmos:

- **Estruturas de dados:**
  - Struct: Utilizada para armazenar as mesas e retângulos encontrados, contendo o comprimento, a largura e a área do retângulo.
  - Vector (STL): Utilizado bastante ao longo do código para representar a planta da casa, a matriz dos retângulos já usados, o vetor com todos os retângulos maximais, o vetor com as mesas, etc. Muito útil pelo seu acesso rápido aos elementos e fácil utilização.
  - Stack (STL): Utilizada pelo algoritmo de calcular os maiores retângulos até uma linha, por conta da sua natureza LIFO e por sua facilidade de uso e acesso.

- **Algoritmos:**

O algoritmo implementado ordena as mesas por área, calcula, para cada linha, os retângulos maximais que se encontram acima dela e os salvam em um vetor, removendo retângulos duplicados, ordena esse vetor de retângulos pela área e, então, acha a primeira mesa que se encaixa em um desses retângulos. Como o algoritmo de achar retângulos considera cada coluna de cada linha e as formas como elas podem formar áreas maiores com colunas anteriores, após passar pela última linha da matriz, todos os retângulos maximais são encontrados, baseado no número de espaços vazios acima das colunas. Por isso, como ambos os vetores de mesas e de retângulos estão ordenados, basta iterar em ordem decrescente sobre os dois vetores e pegar a primeira mesa que se encaixa em algum retângulo, pois esta será a maior mesa possível.

3.1. Pseudocódigo

```
Programa () {
   - Recebe o número de linhas e colunas
   - Recebe a matriz a ser considerada
   - Recebe as mesas possíveis
   - Ordena as mesas pela área e então pela largura

   - Para cada linha:
        - Para cada coluna:
             - Acha o comprimento máximo de cada coluna
             acima da linha sendo considerada
        - Acha todos os retângulos maximais até a linha
        atual, com base nos comprimentos encontrados,
        chamando a função findRectangles
   - Ordena os retângulos pela área e então pela largura
   - Salva a maior área de retângulo encontrada
   - Para cada mesa:
        - Se a área da mesa for maior que a maior área de
        retângulo encontrada, pula para a próxima
        iteração
        - Para cada retângulo:

             - Se a mesa tem dimensões menores que a do
             retângulo, salva a mesa, que é o resultado
             final e sai do laço
   - Imprime a mesa que pode ser colocada na matriz
}

findRectangles {
   - Cria uma pilha
   - Para cada coluna:
        - Se a coluna é maior do que a coluna no topo da
        pilha:
             - Adiciona a coluna na pilha
        - Senão:
             - Calcula a área da coluna atual até a
             coluna no topo da pilha
   - Acha os retângulos da mesma forma para o restante da
   pilha
}
```

4. Análise de complexidade de tempo

Dado que o número de linhas seja M, o número de colunas seja N e o número de mesas seja T, a complexidade de tempo dos algoritmos usados no código são:

findRectangles: Esta função coloca (O(1)) e tira (O(1)) cada coluna do vetor uma única vez, portanto, tem complexidade O(2 * N) = O(N). Além disso, essa função adiciona O(N) retângulos em um vetor passado como parâmetro no pior caso.

Programa: Receber as entradas da planta (O(M * N)) e das mesas (O(T)) possui complexidade O(M * N + T). Então, ordena-se o vetor de mesas, total: (O(T * log(T))); para cada linha, itera-se sobre as colunas salvando os comprimentos máximos (O(N)) e chama-se a função findRectangles (O(N)), total: (O(M * N)); ordena-se o vetor de retângulos achados, de tamanho O(M*N), total: O(M * N * log(M * N)); para cada mesa, procura no vetor de retângulos um que a acomode (O(M*N)), total: O(T * M * N). Portanto, a complexidade total do programa é O(M * N + T + T * log(T) + M * N + M * N * log(M * N) + T * M * N) = O(T * log(T) + M * N * log(M * N) + T * M * N)

Instruções para compilação e Execução

1 – Extraia o arquivo `.zip` na pasta desejada.

2 – Execute o comando `‘g++ tp03.cpp -o tp03’` no terminal.

3 – Execute o programa `tp03` passando um arquivo de texto com a entrada para o programa pela linha de comando.