Trabalho 3

Flávio Marcílio de Oliveira

1. Modelagem Computacional do Problema

O problema, como detalhado nas especificações do trabalho, consiste em encontrar uma lista de todos os retângulos liberados e determinar a mesa de maior área, de uma lista específica, que pode ser inserida nesses retângulos.

Para resolver o problema foi utilizada a abordagem da programação dinâmica com armazenamento dos passos intermediários para encontrar a lista dos retângulos liberados. Uma vez determinada a lista de retângulos liberados, ela é ordenada em ordem decrescente de área, assim como a lista das mesas disponíveis. Então, para cada mesa é verificado se tem um retângulo liberado onde é possível inserir essa mesa. Se a mesa tem uma área maior que a da maior área disponível, ela é retirada da lista de mesas. Se a área da mesa é menor ou igual, é verificado se as dimensões da mesa são compatíveis com a do retângulo e, caso não seja, o retângulo é retirado da lista. Esse processo é repetido até que os valores sejam compatíveis e, assim, tem-se a mesa de maior área que cabe na casa.

A planta da casa foi tratada como sendo uma matriz N x M, com N sendo as linhas e M as colunas.

2. Descrição da Solução

a. Estruturas de Dados

pair<int, int> ii: par de inteiros para armazenar o índice da coluna e quantidade de linhas liberadas a partir daquela célula;

struct rectangle: struct para armazenar comprimento, largura e área dos retângulos determinados;

vector<rectangle> tables: vetor de retângulos para armazenar as mesas disponíveis;

vector<rectangle> max_rect: vetor de retângulos para armazenar os retângulos liberados na planta;

stack<ii> rectangles: uma pilha (LIFO - last in first out) para contabilizar o início e o fim de um retângulo. Sendo o início do retângulo determinado pela célula superior esquerda e o fim do retângulo como sendo a célula inferior direita;

vector<int> space: um vetor de inteiros para contabilizar o número de espaços disponíveis na mesma coluna;

b. Algoritmos

compare_table - função de comparação de mesas utilizada para ordenar as mesas pela área e pela largura como critério de desempate;

compare_space - função de comparação dos espaços liberados. O critério utilizado nessa função foi considerar primeiramente as dimensões e por fim a área de forma a melhorar o desempenho na busca da compatibilidade de mesas com os espaços liberados. Essa estratégia foi utilizada tendo em vista que é possível ter muitos retângulos com mesma área porém com dimensões diferentes;

main - Programa principal

No programa principal, inicialmente o vetor space é inicializado como zero e para cada coluna lida o valor é incrementado se for um espaço vazio (caractere '.') linha por linha. Para cada célula lida é calculado o maior retângulo que é possível obter a partir desta célula.

Assim, utilizando como exemplo a primeira entrada dada no arquivo de especificação, na primeira linha não é identificado nenhum retângulo, sendo o primeiro retângulo calculado na leitura da segunda linha, conforme exemplificado abaixo:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0

Nesse exemplo, o primeiro retângulo tem área igual a 2 e começa na primeira célula disponível (célula azul), o segundo retângulo tem área 1 e começa na segunda célula disponível. O terceiro retângulo tem área igual a 11 e começa na célula verde.
Na leitura da terceira linha, o vetor space será incrementado como segue e os próximos retângulos calculados são como apresentados em vermelho:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
0	2	2	1	2	2	2	0	2	2	2	2	2	2	2	0

No final, a matriz que representa nossa planta estará representada assim:

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0
0	2	2	1	2	2	2	0	2	2	2	2	2	2	2	0
0	3	3	2	3	3	3	0	3	3	3	3	3	3	3	0
0	4	4	3	4	4	4	0	4	4	4	4	0	0	0	0
0	0	0	0	0	0	0	0	0	5	5	0	0	0	0	0

Observe que, a contabilização dessa dimensão é avaliada em um vetor que é incrementado, de forma que não armazenamos a matriz inteira. Porém, a cada linha são computados os maiores retângulos vazios até aquela linha.
Os retângulos calculados dessa forma são colocados em um vetor e ordenados utilizando a função compare_space. Logo em seguida, é lido a quantidade de mesas disponíveis e todas são lidas e colocadas em um vetor que também é ordenado segundo a função compare_table.
A verificação da compatibilidade de mesas com os espaços vazios é feita da seguinte forma: 1) se a mesa tem área maior que a maior área disponível, descartamos a mesa e verificamos a próxima em ordem decrescente. 2) se a mesa tem área menor ou igual ao maior retângulo disponível verificamos a compatibilidade das dimensões da mesa com a do retângulo e, se a maior dimensão da mesa é maior que a maior dimensão do retângulo descartamos a mesa e verificamos a próxima e, caso contrário, verificamos se a menor dimensão da mesa é maior que a menor dimensão do retângulo e se for descartamos o retângulo e verificamos o próximo.
Segue o pseudo-código do programa desenvolvido:

```
main {
    inicializa space[colunas] = 0
    para cada linha faça {
        para cada coluna faça {
            se coluna for diferente de '#'
                space[coluna] += 1;
            senão
                space[coluna] = 0;
        }
        inicializa length = 0;
        para cada coluna faça {
            se space[coluna] > length {
                abre um novo retângulo;
                length = space[coluna];
            }
            senão se (space[coluna] < length) {
                fecha o retângulo aberto;
                calcula a largura;
                calcula o comprimento;
                calcula a área;
                insere no vetor de retângulos;
            }
        }
    }
    ordena o vetor de retângulos;

    lê as mesas;
    ordena o vetor de mesas;

    para cada mesa e cada retângulo faça {
        verifica a compatibilidade da mesa com o retângulo;
    }

    imprime o comprimento e a largura da mesa encontrada;
}
```

3. Análise de Complexidade de Tempo Assintótica

Considerando uma entrada consistindo de uma planta de N linhas e M colunas e uma lista de K mesas.

A inicialização do vetor *space* tem um custo linear em M, de acordo com a documentação, ou seja, O(M).

Para cada linha da planta, as colunas são percorridas calculando os possíveis retângulos com um custo máximo O(M), pois em uma linha não se pode ter mais de M/2 retângulos. Portanto, a etapa de determinação dos retângulos tem complexidade O(NM).

Em seguida, tem-se a etapa de ordenação dos retângulos que, pela documentação, é O(n log n) com n sendo o tamanho do vetor. O máximo de retângulos no vetor será NM/2, portanto, a ordenação dos retângulos terá uma complexidade O(NM log NM).

Na etapa seguinte, tem-se a leitura das mesas e a ordenação com complexidade O(K log K), não considerando o tempo de leitura.

Por fim, a determinação da mesa compatível com a casa tem uma complexidade de tempo no pior caso de O(max(K,NM/2)), que ocorre quando todas as mesas e todos os retângulos são comparados, observando que, uma vez incompatíveis, ou a mesa é retirada da lista ou o retângulo é retirado da lista evitando novas comparações.

Portanto, o algoritmo implementado tem uma complexidade global de O(M+NM+NM log NM+ K log K + max(K, NM/2)) = O(NM log NM + K log K + max(K, NM/2)).

4. Conclusão

O trabalho foi desenvolvido utilizando a programação dinâmica, permitindo encontrar todos os retângulos disponíveis na planta. Depois foi realizada a verificação da compatibilidade das mesas com os retângulos, permitindo encontrar a mesa de maior área que é possível colocar na casa.

Apêndice

O programa foi desenvolvido, compilado e executado em ambiente Linux: WSL: UBUNTU-20.04

A. Compilação

O programa é compilado utilizando o comando ***g++ main.cpp***

B. Execução

A execução é realizada utilizando o comando ***./a.out < input.txt***

onde input.txt é o arquivo com os dados do problema