Samai Soares Pereira **TP01**

2020054811

1. Introdução

O problema proposto foi implementar um algoritmo onde em um arquivo de entrada, são dadas uma matriz com caracteres representando uma planta de uma casa onde os "." são locais disponíveis para a alocação da mesa e os "#" construções onde não há como alocar a mesa. Também nesse arquivo é disponibilizado uma lista de mesas para verificar qual a maior mesa possível para alocar na planta da casa. Para solucionar esse trabalho de forma ótima, foi utilizado um algoritmo de programação dinâmica para a análise de cada retângulo possível da matriz binária. Também nesse algoritmo que foi utilizada a função std::sort que utiliza da combinação de dois algoritmos sendo que um foi visto durante a aula que foram Heap sort e insertion sort (mencionado no qual foi aprendido na aula). Dessa forma para cada retângulo encontrado é analisado sua área e suas dimensões e analisa se é possível

2. Implementação

O programa foi implementado utilizando apenas 4 arquivos fontes main.cpp, manipulamatriz.cpp, encontraretangulos.cpp e struct.hpp no qual há duas funções principais para execução do código que são:

```
void processamatriz(int **matriz, int comprimento_matriz, int largura_matriz, dimensoes *mesa, int tamanhopesquisa)
```

Essa função tem como principal objetivo incrementar a matriz binária criada referente à matriz disponibilizada pelo arquivo de entrada de maneira que, caso o valor de

matriz[i][j] <> 0, ele soma o valor ao seu anterior e dessa forma teremos uma matriz cujo os valores de cada elemento da matriz seja a soma de todos os anteriores possíveis. Dessa forma caso tenhamos um vetor de 10 posições onde todas as posições são possíveis alocar uma mesa na posição 1 teremos um valor 1 para o primeiro elemento e na posição 10 teremos o valor 10. Caso Na posição 4 tenha um obstáculo na posição 10 terá um valor igual a 10.

Além disso essa função chama a função processalinha e envia 3 ponteiros nos quais serão os responsáveis pelo resultado final da maior mesa.

```
void processalinha(int *linha, int tamanho_linha, dimensoes *mesa, int *tamanho_pesquisa, int *comprimento, int *largura)
```

Essa função tem como principal objetivo encontrar a maior área de retângulo na matriz disponibilizada e analisar se cada cadeira cabe em cada retângulo encontrado. Nessa função primeiramente cada linha passada da matriz passa pela iteração onde é adicionado um valor à pilha de elementos dessa forma ele calcula o valor da área máxima entre o valor I * posição da linha e analisa qual a maior área se é a areamaxima anterior ou area atual, caso seja a área anterior, mantem o valor da areamaxima que está, se for atual atualiza o valor areamaxima. Esse procedimento é baseado no algoritmo de Bellman uma vez que ele sempre procura o valor ótimo para cada elemento da matriz, no caso, maior área.

Logo após se a área atual, for maior que o comprimento*largura armazenado, realizar um loop percorrendo cada valor da mesa, caso ache uma mesa em que o as dimensões sejam menores que a das dimensões encontradas na função, atualize o ponteiro largura, o ponteiro comprimento, e o valor da posição do vetor, para que nas próximas vezes ir diminuindo a profundidade da pesquisa no vetor mesas.

3. Complexidade

Para calcular a complexidade do algoritmo vamos analisar cada função separadamente.

Observaremos cada loop e observaremos a que tem maior complexidade pois ela sobressairá sobre as demais.

Consideremos para fazer o cálculo de complexidade a variável M, sendo as linhas, N colunas e P sendo a pesquisa.

main.cpp:

No documento main.cpp temos na função **main** um loop no qual tem complexidade **O(M*N),** pois cria uma matriz dinâmica com as dimensões da matriz inicial passada no documento de entrada.

manipulamatriz.cpp:

No arquivo manipulamatiz.cpp temos duas funções com complexidades semelhantes que são **transforma_matriz e preenchematriz** ambas funções com complexidade similares, uma vez que estas percorrem todos os elementos da matriz ou escreve a matriz por completa. Logo a complexidade é igual a **O(M*N).**

Também nesse arquivo há uma função chamada **mesas** no qual preenche o vetor mesas que tem complexidade igual a **O(P)**, que é igual à quantidade de pesquisas referente ao arquivo de entrada.

encontraretangulos.cpp:

Nesse arquivo há duas funções principais para a funcionalidade do código que são as:

processamatriz que tem complexidade igual a **O(M*N*P)**, mas para chegar nessa complexidade, vamos derivar o código.

A própria processa matriz passa por todos elementos da matriz, logo ela tem complexidade

O(M*N). Entretanto dentro do loop de M é chamada a função **processalinha** que vamos derivar agora.

processalinha: Essa função ela executa para cada elemento de cada linha **O(N)** e para cada elemento do vetor decrementado de um para cada iteração. Dessa forma no pior caso primeira iteração percorrerá o vetor por completo, na segunda percorrerá P-1 vez, até 1, e isso tem complexidade (1+P)/2, que no final resultará O(P). Finalmente terá complexidade **O(P*N).**

Veremos que no final, a complexidade será **O(M*N*P).**

4. Pseudocódigo

Processalinha(linha, tamanho_linha, mesa, tamanho_pesquisa, comprimento, largura)

```
Cria pilha valores;
for(i cada elemento da linha)
{
//se a pilha de valores indices estiver vazia ou linha na posição do topo da pilha for maior que linha[i];
        //atualiza o valor no topo da pilha;

if (a pilha estiver vazia ou linha[posição topo da pilha]<= linha[i])
{
Adiciona valor de i na pilha;
}
Else
{
Valormaximo = linha[posição topo pilha]
Area = valormaximo * i;
Desempilha uma posição da pilha;
If(pilha <> vazia)
{
// função acha a área atual baseado na posição matriz[i][j] * indice;
        Area = valormaximo*(posição topo da pilha - i)
        Areamaxima = max(area, areaMaxima)
        }
        If(comprimento * largura < area atual)
        {
        For(j para cada elemento do tamanho_pesquisa)
```

```
        {
        If (mesa[j].dimensao=<dimensoes atual )
        {
        Atualiza valor comprimento
        Atualiza valor largura
        Taamanho_pesquisa = j
        }

}
}
```

processamatriz(matriz,comprimento_matriz, largura_matriz, mesa,tamanhopesquisa)

```
{
For(i para cada linha){
For (j para cada coluna)
{
Se (valor matriz[i][j]<>0)// soma valor da matriz aos anteriores caso seja diferente de um
        {
        Matriz[i][j]=matriz[i][j]+ matriz[i][j-1]
        }

        }

}
Processalinha()//para achar o maior comprimento e largura dinamicamente
If(coluna e linha for igual ao primeiro elemento do vetor )
{
break}

}
```

Imprime linha e coluna;

}

5. Instruções para Compilação

Para compilar o código necessita apenas executar o código na pasta do arquivo:

g++ main.cpp -o main

E para executar o programa, após compilado na linha de comendo escrever

main.exe < arquivoentrada.txt