TP 03 - Trabalho Prático 3
Algoritmos I

Igor Lacerda Faria da Silva

igorlfs@ufmg.br

1 Introdução

O TP foi desenvolvido em C++, em ambiente Linux. Em termos de boas práticas foram as usadas as ferramentas `clang-format` e `clang-tidy` para garantir um código limpo e consistente.

2 Modelagem Computational do Problema

O programa lê um mapa de uma planta de uma residência e um conjunto de tamanhos (comprimento, largura) de mesas, e busca encaixar a maior mesa (em termos de área, com desempate de largura) possível dadas as restrições de espaço da casa (alguns locais podem estar ocupados). Após a leitura da entrada, as mesas são ordenadas pelos critérios especificados e é realizada uma busca sequencial, verificando se na casa existe alguma região capaz de comportar cada mesa. Quando é encontrada uma mesa que cabe, o programa retorna suas dimensões.

A busca percorre toda a planta (excluindo as bordas e possíveis locais que "estouram" a casa), considerando cada elemento da matriz como um candidato (correspondendo à posição de canto superior esquerdo). Se no espaço que começa nessa posição existe um "bloco" ocupado, o candidato é descartado e é checado o bloco seguinte, e assim por diante. Em sucessão é testada a mesma mesa, mas com as dimensões invertidas. É "necessário"[1] fazer essa busca exaustiva porque dependendo do arranjo da casa, existem diversas maneiras de se expandir um retângulo que conteria a mesa. Em particular, isso se deve ao fato de que as *expansões máximas* dos retângulos podem se sobrepor.

Embora essa abordagem não seja eficiente, ela está correta, e não é difícil ver isso. Suponha que o algoritmo descrito acima dê a resposta errada para alguma entrada, ou seja, que existe uma mesa *maior* que a encontrada pelo algoritmo. Se ela é de fato maior, então foi checada antes da encontrada, pois as mesas são ordenadas. E como o algoritmo não parou nela, isso significa que ela não cabe (toda a matriz foi checada), o que é absurdo.

[1] É claro que existe de se resolver o problema que não faz uso dessa *gambiarra*.

3 Estruturas de Dados e Algoritmos

Como mencionado anteriormente, a planta é representada por uma matriz (um vetor de vetores de `char`'s), que na implementação é a classe `Map`, que é responsável por fazer a leitura da planta e por buscar. O outro algoritmo importante é a ordenação, com uma função de comparação que atende à especificação.

Uma descrição em alto nível do programa seria: ordene as mesas, para cada mesa, veja se ela cabe no retângulo que tem como elemento superior esquerdo (0,0). Se não for, procure na "expansão" (0,1) e continue assim, até esgotar o comprimento da casa. Prossiga para a linha seguinte, até esgotar todas as linhas. Se ainda não der certo, inverta as dimensões da mesa. Vá para a mesa seguinte se a mesa atual não couber[2].

4 Análise de Complexidade de Tempo

A leitura da entrada tem complexidade $\Theta(mn + k)$ em que m, n são as dimensões da casa e k é o número de mesas candidatas. Depois, a ordenação das mesas tem complexidade $k \log k$. A busca de uma mesa para uma posição especifica tem complexidade $\Theta(ij)$ em que i, j são as dimensões da mesa. Tomando $i_0 = \max i$ e $j_0 = \max j$ como as dimensões máximas, podemos avaliar a complexidade da busca como um todo: para uma cada das k mesas, são checadas mn posições em que se confere no máximo $i_0 j_0$ caracteres, no pior caso. Concluímos então que a complexidade da busca é $\Theta(kmni_0 j_0)$ e do programa como um todo é $\Theta(k \log k + kmni_0 j_0)$.

5 Compilação

```
g++ src/map.cpp -c -Wall -std=c++17 -Ilib -o obj/map.o
g++ src/main.cpp -c -Wall -std=c++17 -Ilib -o obj/main.o
g++ obj/map.o obj/main.o -o tp3
```

[2] Creio eu que, com uma ideia tão simples, não é necessário fazer um pseudocódigo.