**UNIVERSIDADE FEDERAL DE VIÇOSA**

**CAMPUS FLORESTAL**

PABLO FERREIRA - 3480

SAMUEL SENA - 3494

**TRABALHO PRÁTICO II**

**RELATÓRIO TAREFA A**

FLORESTAL

2019

Sumário

[Introdução 3](#_Toc24317546)

[Comparação de tempos de execução 4](#_Toc24317547)

[Exemplos de Execução 7](#_Toc24317548)

[Modo DEBUG 9](#_Toc24317549)

[Desenvolvimento 11](#_Toc24317550)

[Conclusão 14](#_Toc24317551)

# Introdução

O trabalho apresentado a seguir entrega um algoritmos capaz de resolver o problema da pirâmide de números (Tarefa A).

Inicialmente para se executar o programa da Tarefa A, é necessário realizar a compilação do código fonte em C. Para isso, em algum terminal Linux execute o “*makefile”* da seguinte forma:

Para compilar:

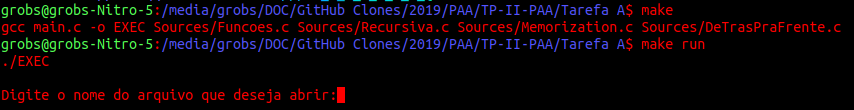
$ make

E para executar:

$ make run

A figura abaixo exemplifica o processo de compilação e execução pelo Linux:

Figura 1

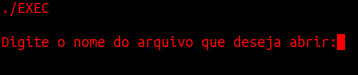


Fonte: Terminal Linux

O algoritmo foi testado apenas em sistema operacional baseado em Linux, a execução em Windows pode não ser satisfatória.

O programa inicialmente exibirá um menu com as seguintes opções:

Figura 2

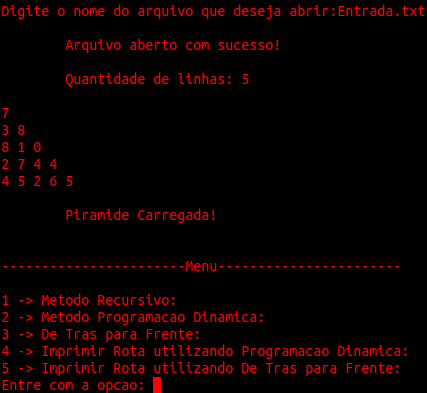


Fonte: Terminal Linux

Primeiramente, entre com o nome do arquivo de entrada desejado, em seguida, caso a abertura seja realizada com sucesso, a pirâmide lida é impressa na tela e um menu com 5 opções é exibido. Dentre as possibilidades de opções estão:

1 - Resultado da maior soma utilizando recursividade; 2 - Resultado da maior soma utilizando programação dinâmica; 3 - Resultado da maior soma utilizando algoritmo de trás para frente; 4 - Imprimir rota utilizando como base programação dinâmica; 5 - Imprimir rota utilizando como base algoritmo de trás para frente. A figura a seguir ilustra bem a situação descrita:

Figura 3



Fonte: Terminal Linux

Após a escolha desejada e execução da mesma, o programa ira sempre finalizar em seguida. Caso deseje executar mais de uma operação em um mesmo arquivo de entrada, será necessário reexecutar o programa.

# Comparação de tempos de execução

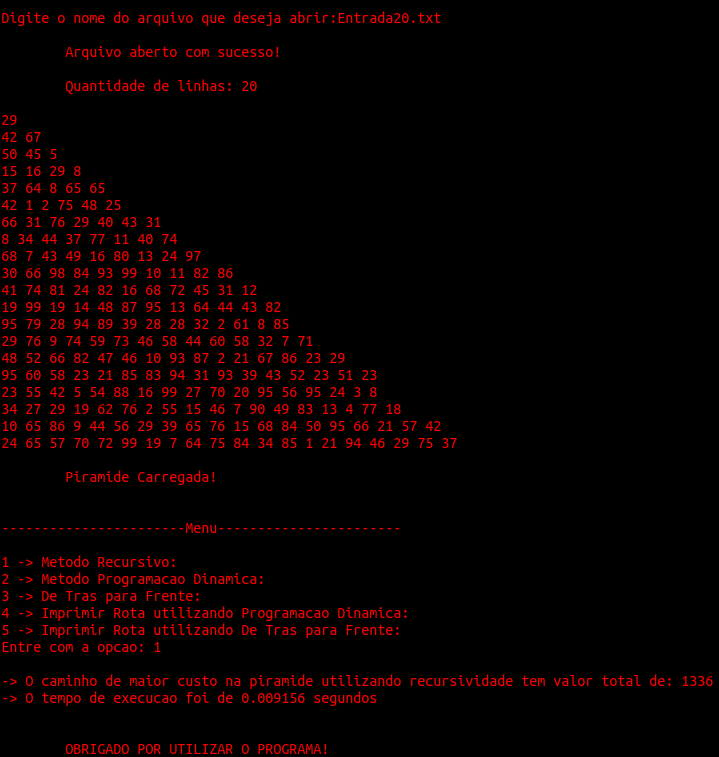
Os gráficos a seguir servem de comparação entre o desempenho dos algoritmos utilizados. Os mesmos relacionam tempo X tamanho da entrada (em linhas da pirâmide):

É notável a enorme diferença de desempenho entre o algoritmo recursivo com os demais com entradas maiores que 25 linhas. Para as pequenas entradas, nenhuma diferença impactante é sentida. Dentre os 3, o de melhor desempenho é o algoritmo de trás para frente.

# Exemplos de Execução

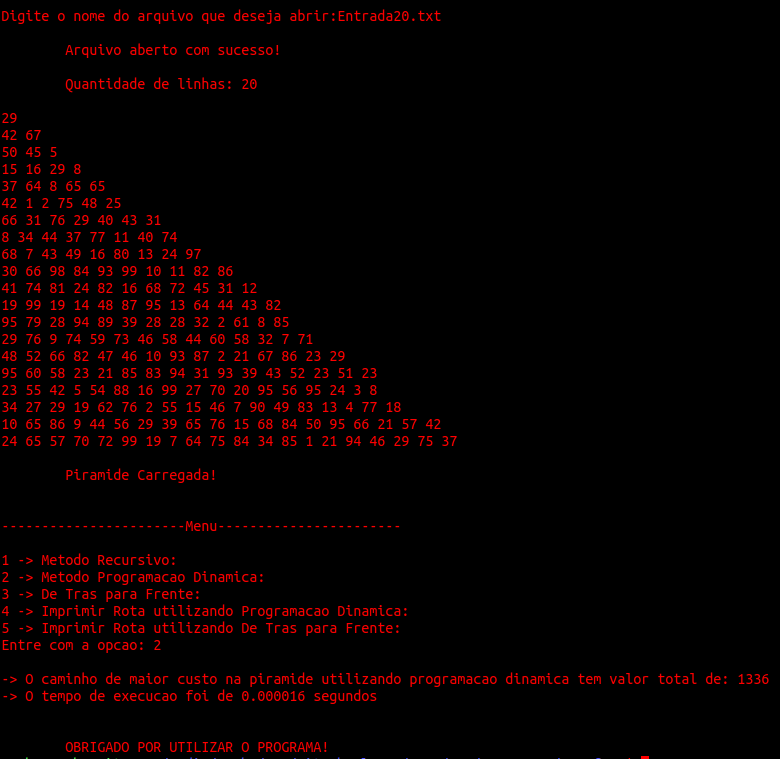
As figuras (4 a 6) a seguir ilustram a execução do programa com as 3 possíveis formas de resolução do problema da pirâmide para entrada igual a 20 linhas:

Figura 4



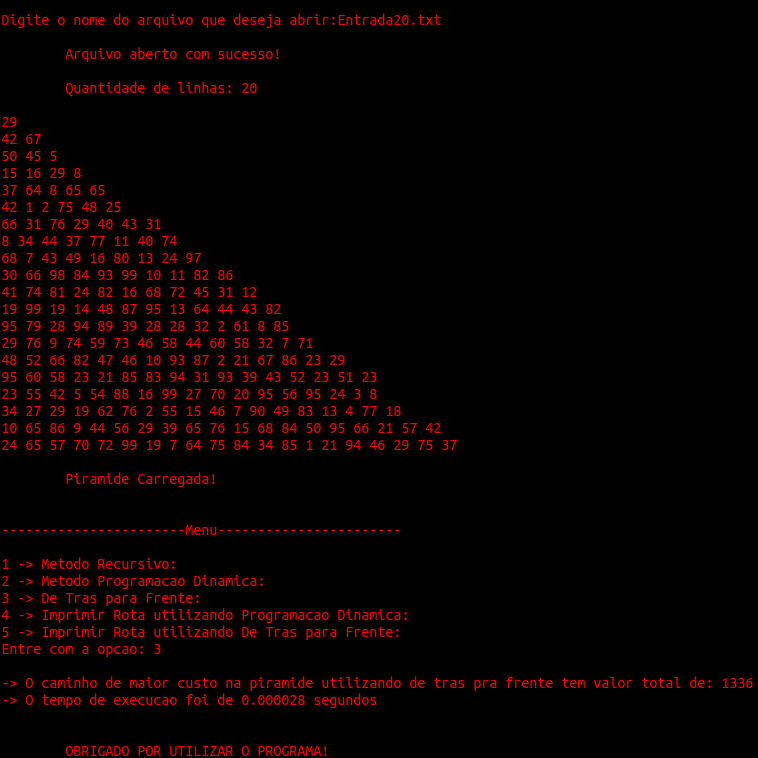
Fonte: Terminal Linux

Figura 5



Fonte: Terminal Linux

Figura 6



Fonte: Terminal Linux

# Modo DEBUG

O programa conta com a opção de utilização em modo “debug” que (quando ativado) exibe na tela o tempo de execução de cada algoritmo escolhido, para utiliza-lo, o valor de “DEBUG” no arquivo “main.c” deve ser definido para o 1, caso seja definido como 0, o programa ira rodar sem exibir o tempo de execução de cada algoritmo. A figura abaixo demonstra o local da definição do valor de “DEBUG”:

Figura 7



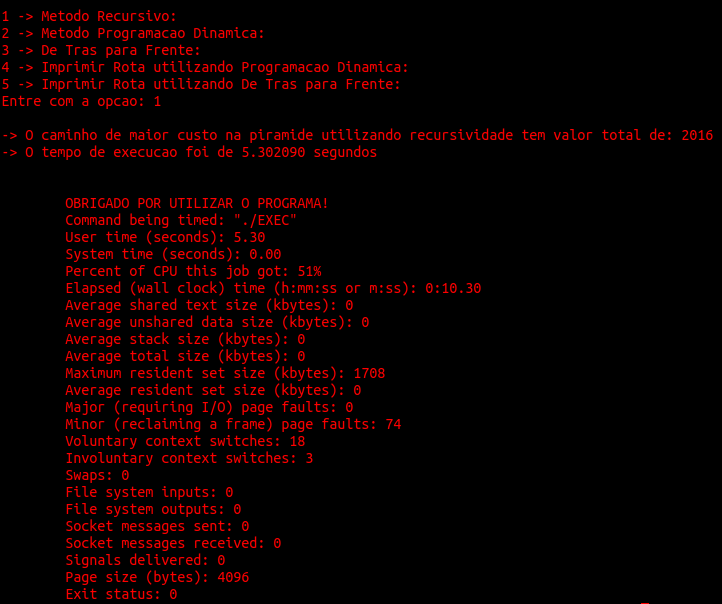
Fonte: main.c

A medição do uso de CPU de cada algoritmo é realizado através da chamada do seguinte comando no momento da execução:

$ command time -v ./EXEC

Assim sendo, é necessário ter o programa “time” devidamente instalado no sistema operacional Linux no qual a medição será realizada. O uso de CPU será impresso da seguinte forma:

Figura 8



Fonte: Terminal Linux

No exemplo acima, o uso de CPU foi de 51%.

A medição do uso de memória é obtido através do uso da ferramenta “Valgrind”, que assim como o programa “time” é chamado no momento da execução do programa. Para executar a ferramenta e obter o uso de memória execute:

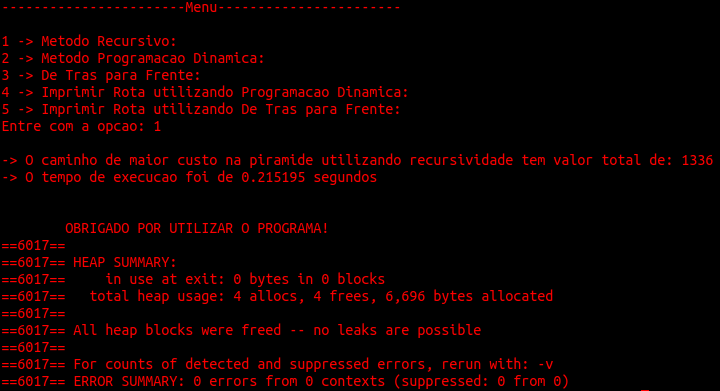
$ valgrind --tool=memcheck ./EXEC

Para facilitar o uso da ferramenta “Valgrind”, o comando para a respectiva chamada foi adicionado ao “makefile”. Para realizar a chamada da medição de memória utilizando o “makefile” digite o seguinte comando:

$ make Mem

Em ambos os comandos, o seguinte resultado será impresso na tela:

Figura 9



Fonte: Terminal Linux

No exemplo acima, o total de memória utilizada foi de cerca de 6,969 bytes.

# Desenvolvimento

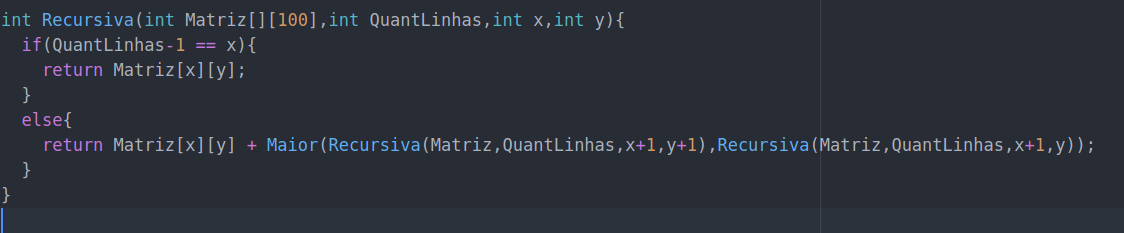
O funcionamento do algoritmo consistem basicamente na leitura de cada linha do arquivo e a gravação de todo conteúdo da pirâmide em uma matriz de números inteiros de 100 linhas por 100 colunas (alocado estaticamente). Dessa forma, apenas as posições inicializadas são utilizadas. Após a leitura do arquivo de entrada e inicialização da matriz, o usuário escolhe qual algoritmo é utilizado para encontrar o resultado da maior soma. Dentre os principais algoritmos e funções utilizadas, temos:

**Algoritmo Recursivo:**

O algoritmo recursivo ilustrado abaixo, foi implementado como um tipo de solução para o problema da pirâmide. O algoritmo utiliza uma função auxiliar denominada “Maior()”, a qual retorna o maior dos 2 parâmetros passados. O algoritmo basicamente funciona através da chamada recursiva exaustiva, onde apenas os maiores valores retornados são considerados.

Figura 10

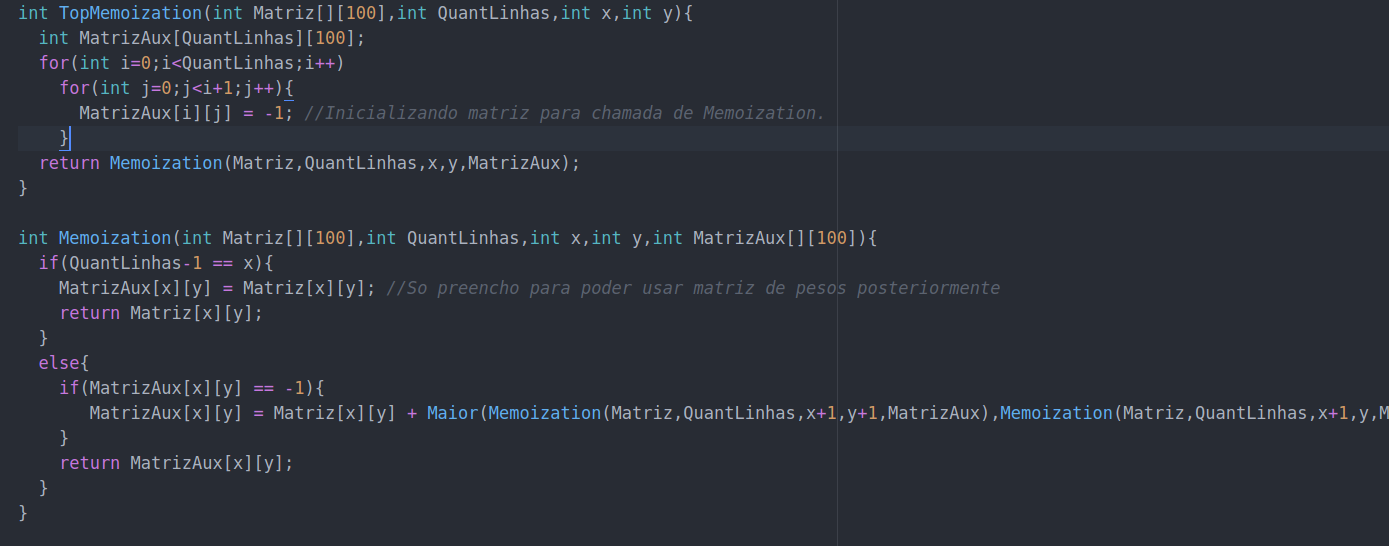
Fonte: Recursiva.c



**Programação Dinâmica:**

O algoritmo ilustrado na figura abaixo utiliza uma forma de programação dinâmica. Ele utiliza como base o algoritmo recursivo, no entanto possui uma matriz auxiliar (inicializada com o valor -1) para manter salvo os cálculos já realizados. Caso a posição acessada ainda contenha o valor -1 durante o processo de cálculo, ele é substituído pelo resultado encontrado na instância recursiva atual e caso a mesma posição seja novamente acessada em outra instância, o valor salvo na matriz auxiliar será automaticamente utilizado ao invés do retrabalho ser realizado.

Figura 11



Fonte: Memoization.c

**Algoritmo de trás para frente:**

O algoritmo apresentado na figura abaixo tem como critério principal uma solução de caráter polinomial para o problema da pirâmide. Ele utiliza uma matriz auxiliar, apenas para evitar a alteração da matriz original (além de possibilitar a impressão da rota escolhida, que será descrito mais a frente). O algoritmo também utiliza a função auxiliar “Maior()” para determinar qual dos elementos na linha abaixo é o maior e deverá ser somado ao valor do elemento atual.

Figura 12

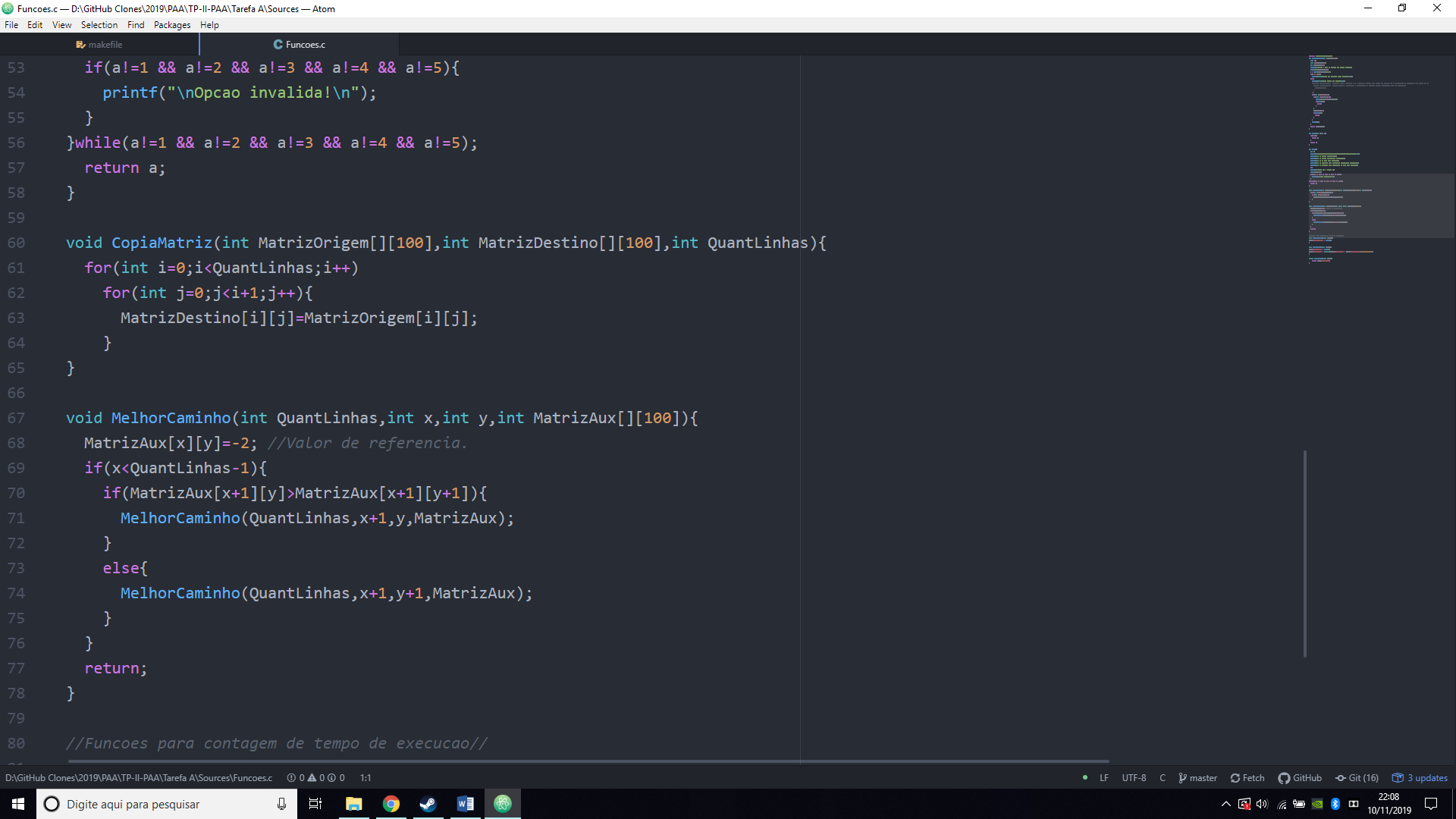


Fonte: DeTrasPraFrente.c

**Impressão de rotas:**

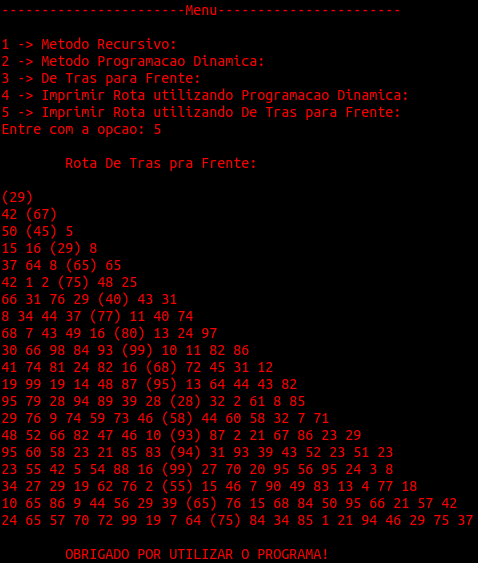
As figuras a seguir ilustram o processo de busca pela rota na qual a maior soma foi encontrada. A função “MelhorCaminho()” recebe como parâmetro uma matriz auxiliar de peso (seja ela a utilizada no algoritmo “Memoization” ou “DeTrasPraFrente”), em seguida as posições de maior peso acessíveis em cada linha recebem o valor -2 (são marcadas). Após isso, durante a chamada da impressão da rota, a matriz auxiliar de peso é percorrida e exatamente nas coordenadas das posições correspondentes aos valores -2, os valores presentes na matriz original são impressos entre parênteses, destacando assim a rota encontrada. As figuras a seguir demonstram a implementação das funções e a execução:

Figura 13



Fonte: Funcoes.c

Figura 14



Fonte: Terminal Linux

# Conclusão

Sem dúvidas, o desenvolvimento e execução dos algoritmos implementados possibilitou uma melhor visualização do impacto do tempo de execução com diferentes tamanhos de entradas. Dessa forma, é possível aprender de forma totalmente prática, quando um algoritmo recursivo é totalmente contraindicado.

Agradecimentos ao professor Daniel Mendes pela oportunidade de realização do trabalho e dúvidas sanadas.

Todo o desenvolvimento e distribuição do trabalho encontra-se hospedado na seguinte página do [GitHub](https://github.com/Globson/TP-II-PAA).