

**UNIVERSIDADE FEDERAL DE VIÇOSA**  
**CAMPUS FLORESTAL**

PABLO FERREIRA - 3480

SAMUEL SENA - 3494

**TRABALHO PRÁTICO II**  
**RELATÓRIO TAREFA A**

FLORESTAL

2019

# Sumário

Introdução	3
Comparação de tempos de execução	4
Exemplos de Execução	7
Modo DEBUG	9
Desenvolvimento	11
Conclusão	14

## 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:

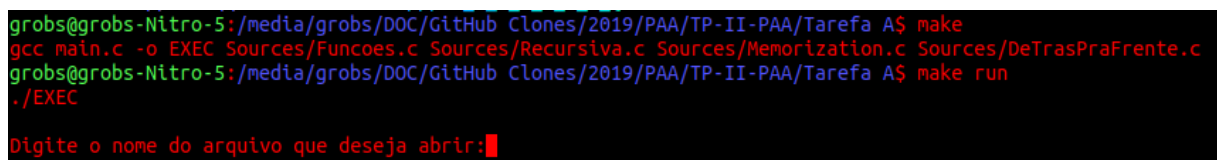
\$ make

E para executar:

\$ make run

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

Figura 1



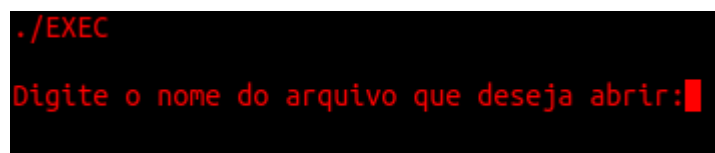
```
grobs@grobs-Nitro-5:/media/grobs/DOC/GitHub Clones/2019/PAA/TP-II-PAA/Tarefa A$ make
gcc main.c -o EXEC Sources/Funcoes.c Sources/Recursiva.c Sources/Memorization.c Sources/DeTrasPraFrente.c
grobs@grobs-Nitro-5:/media/grobs/DOC/GitHub Clones/2019/PAA/TP-II-PAA/Tarefa A$ make run
./EXEC
Digite o nome do arquivo que deseja abrir:█
```

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



```
./EXEC
Digite o nome do arquivo que deseja abrir:█
```

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

```
Digite o nome do arquivo que deseja abrir:Entrada.txt

Arquivo aberto com sucesso!

Quantidade de linhas: 5

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

Piramide Carregada!

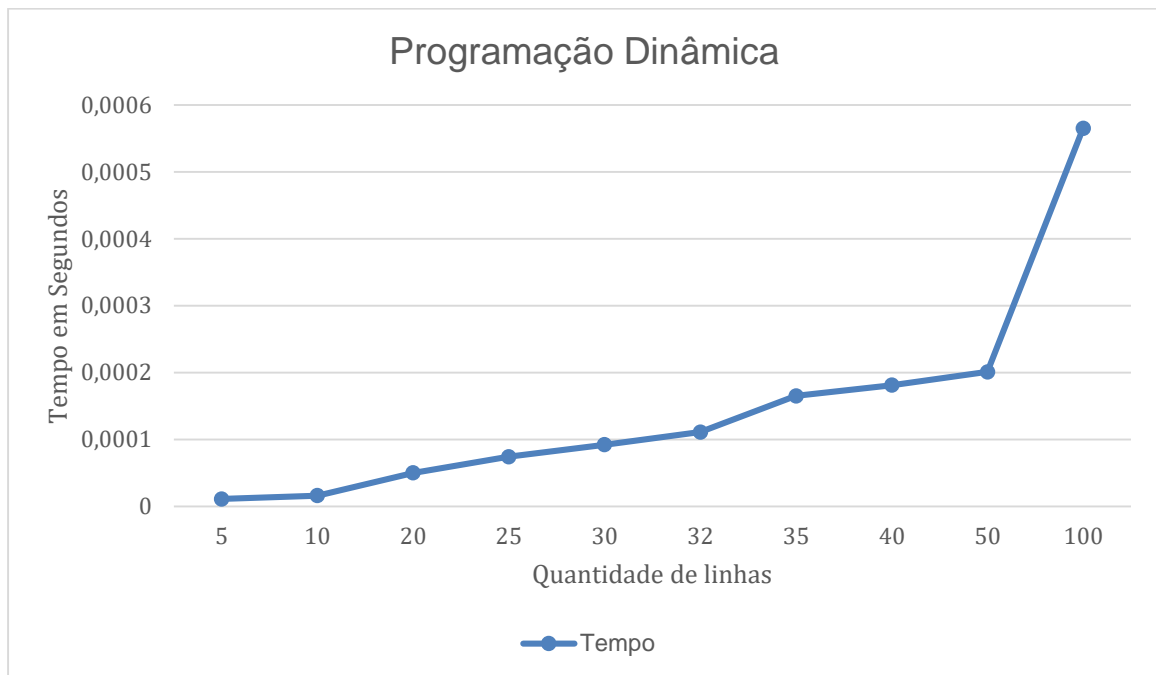
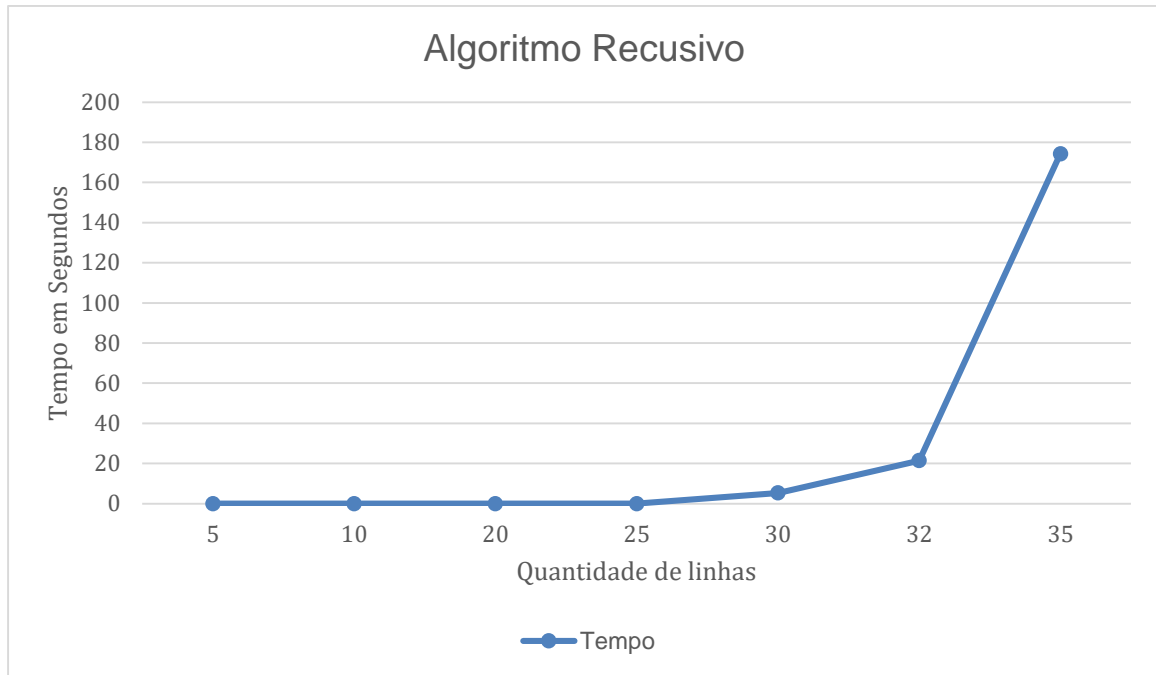
-----Menu-----
1 -> Metodo Recursivo:
2 -> Metodo Programacao Dinamica:
3 -> De Tras para Frente:
4 -> Imprimir Rota utilizando Programacao Dinamica:
5 -> Imprimir Rota utilizando De Tras para Frente:
Entre com a opcao: █
```

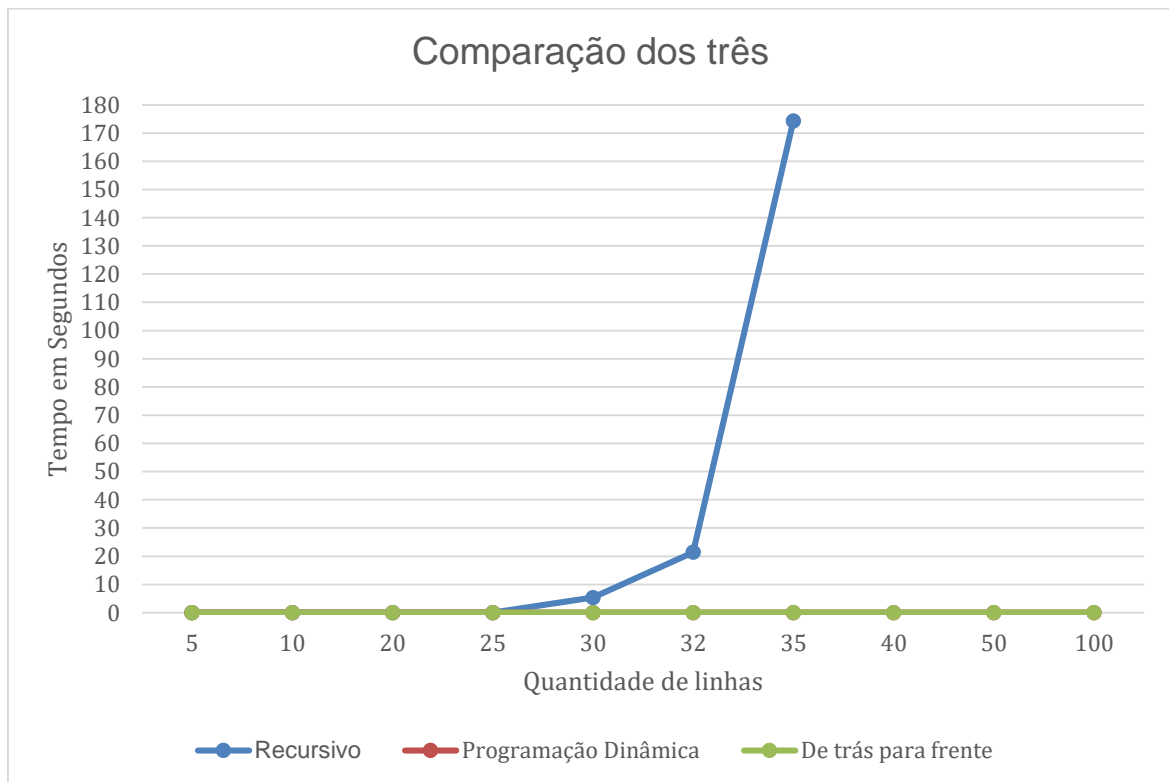
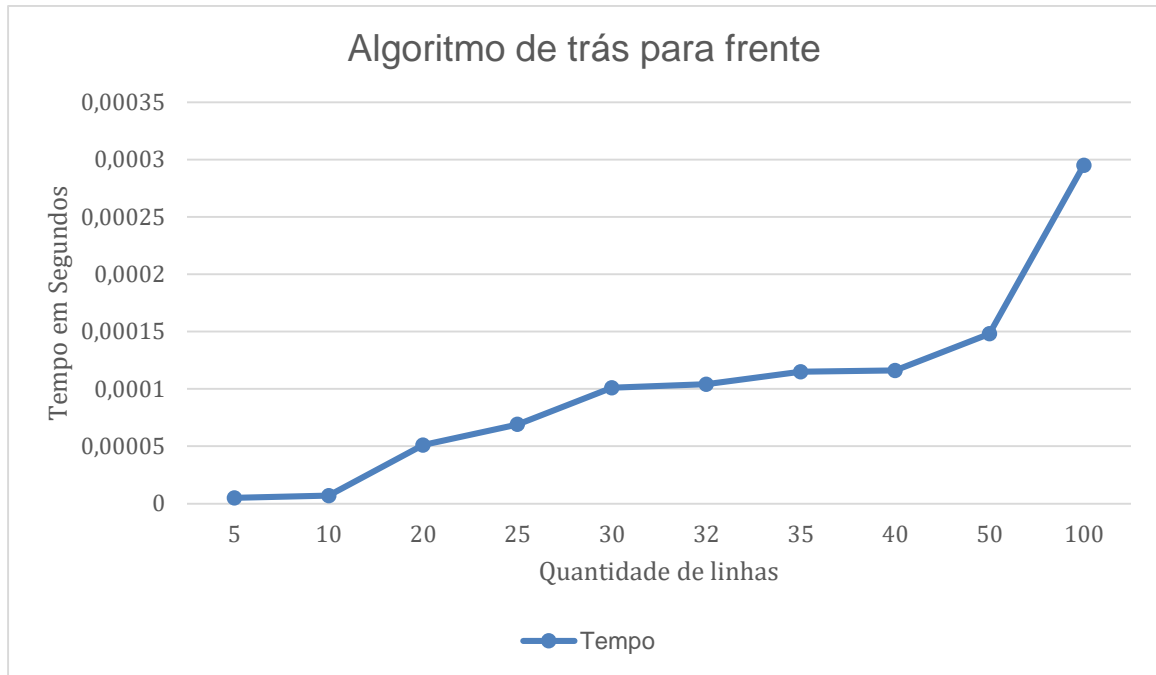
Fonte: Terminal Linux

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

```
Digite o nome do arquivo que deseja abrir:Entrada20.txt

Arquivo aberto com sucesso!

Quantidade de linhas: 20

29
42 67
50 45 5
15 16 29 8
37 64 8 65 65
42 1 2 75 48 25
66 31 76 29 40 43 31
8 34 44 37 77 11 40 74
68 7 43 49 16 80 13 24 97
30 66 98 84 93 99 10 11 82 86
41 74 81 24 82 16 68 72 45 31 12
19 99 19 14 48 87 95 13 64 44 43 82
95 79 28 94 89 39 28 28 32 2 61 8 85
29 76 9 74 59 73 46 58 44 60 58 32 7 71
48 52 66 82 47 46 10 93 87 2 21 67 86 23 29
95 60 58 23 21 85 83 94 31 93 39 43 52 23 51 23
23 55 42 5 54 88 16 99 27 70 20 95 56 95 24 3 8
34 27 29 19 62 76 2 55 15 46 7 90 49 83 13 4 77 18
10 65 86 9 44 56 29 39 65 76 15 68 84 50 95 66 21 57 42
24 65 57 70 72 99 19 7 64 75 84 34 85 1 21 94 46 29 75 37

Piramide Carregada!

-----Menu-----

1 -> Metodo Recursivo:
2 -> Metodo Programacao Dinamica:
3 -> De Tras para Frente:
4 -> Imprimir Rota utilizando Programacao Dinamica:
5 -> Imprimir Rota utilizando De Tras para Frente:
Entre com a opcao: 1

-> O caminho de maior custo na piramide utilizando recursividade tem valor total de: 1336
-> O tempo de execucao foi de 0.009156 segundos

OBRIGADO POR UTILIZAR O PROGRAMA!
```

Fonte: Terminal Linux

Figura 5

```
Digite o nome do arquivo que deseja abrir:Entrada20.txt

Arquivo aberto com sucesso!

Quantidade de linhas: 20

29
42 67
50 45 5
15 16 29 8
37 64 8 65 65
42 1 2 75 48 25
66 31 76 29 40 43 31
8 34 44 37 77 11 40 74
68 7 43 49 16 80 13 24 97
30 66 98 84 93 99 10 11 82 86
41 74 81 24 82 16 68 72 45 31 12
19 99 19 14 48 87 95 13 64 44 43 82
95 79 28 94 89 39 28 28 32 2 61 8 85
29 76 9 74 59 73 46 58 44 60 58 32 7 71
48 52 66 82 47 46 10 93 87 2 21 67 86 23 29
95 60 58 23 21 85 83 94 31 93 39 43 52 23 51 23
23 55 42 5 54 88 16 99 27 70 20 95 56 95 24 3 8
34 27 29 19 62 76 2 55 15 46 7 90 49 83 13 4 77 18
10 65 86 9 44 56 29 39 65 76 15 68 84 50 95 66 21 57 42
24 65 57 70 72 99 19 7 64 75 84 34 85 1 21 94 46 29 75 37

Piramide Carregada!

-----Menu-----

1 -> Metodo Recursivo:
2 -> Metodo Programacao Dinamica:
3 -> De Tras para Frente:
4 -> Imprimir Rota utilizando Programacao Dinamica:
5 -> Imprimir Rota utilizando De Tras para Frente:
Entre com a opcao: 2

-> O caminho de maior custo na piramide utilizando programacao dinamica tem valor total de: 1336
-> O tempo de execucao foi de 0.000016 segundos

OBRIGADO POR UTILIZAR O PROGRAMA!
```

Fonte: Terminal Linux



Figura 6

```
Digite o nome do arquivo que deseja abrir:Entrada20.txt

Arquivo aberto com sucesso!

Quantidade de linhas: 20

29
42 67
50 45 5
15 16 29 8
37 64 8 65 65
42 1 2 75 48 25
66 31 76 29 40 43 31
8 34 44 37 77 11 40 74
68 7 43 49 16 80 13 24 97
30 66 98 84 93 99 10 11 82 86
41 74 81 24 82 16 68 72 45 31 12
19 99 19 14 48 87 95 13 64 44 43 82
95 79 28 94 89 39 28 28 32 2 61 8 85
29 76 9 74 59 73 46 58 44 60 58 32 7 71
48 52 66 82 47 46 10 93 87 2 21 67 86 23 29
95 60 58 23 21 85 83 94 31 93 39 43 52 23 51 23
23 55 42 5 54 88 16 99 27 70 20 95 56 95 24 3 8
34 27 29 19 62 76 2 55 15 46 7 90 49 83 13 4 77 18
10 65 86 9 44 56 29 39 65 76 15 68 84 50 95 66 21 57 42
24 65 57 70 72 99 19 7 64 75 84 34 85 1 21 94 46 29 75 37

Piramide Carregada!

-----Menu-----
1 -> Metodo Recursivo:
2 -> Metodo Programacao Dinamica:
3 -> De Tras para Frente:
4 -> Imprimir Rota utilizando Programacao Dinamica:
5 -> Imprimir Rota utilizando De Tras para Frente:
Entre com a opcao: 3

-> O caminho de maior custo na piramide utilizando de tras pra frente tem valor total de: 1336
-> O tempo de execucao foi de 0.000028 segundos

OBRIGADO POR UTILIZAR O PROGRAMA!
```

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

```
#include "Headers/DeTrasPraFrente.h"
#define DEBUG 1
```

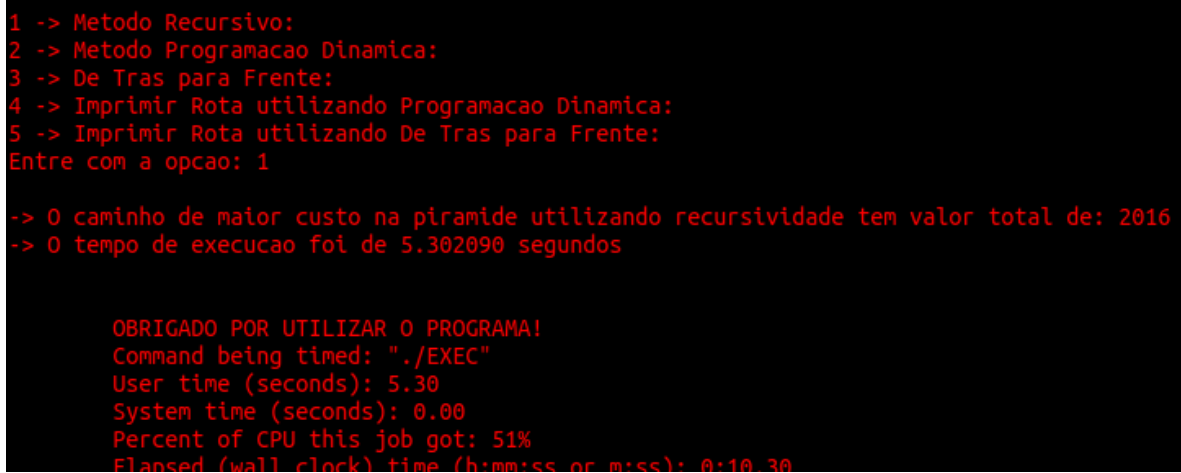
Fonte: main.c

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



```
1 -> Metodo Recursivo:
2 -> Metodo Programacao Dinamica:
3 -> De Tras para Frente:
4 -> Imprimir Rota utilizando Programacao Dinamica:
5 -> Imprimir Rota utilizando De Tras para Frente:
Entre com a opcao: 1

-> O caminho de maior custo na piramide utilizando recursividade tem valor total de: 2016
-> O tempo de execucao foi de 5.302090 segundos

OBRIGADO POR UTILIZAR O PROGRAMA!
Command being timed: "./EXEC"
User time (seconds): 5.30
System time (seconds): 0.00
Percent of CPU this job got: 51%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:10:30
```

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

```
-----Menu-----
1 -> Metodo Recursivo:
2 -> Metodo Programacao Dinamica:
3 -> De Tras para Frente:
4 -> Imprimir Rota utilizando Programacao Dinamica:
5 -> Imprimir Rota utilizando De Tras para Frente:
Entre com a opcao: 1

-> O caminho de maior custo na piramide utilizando recursividade tem valor total de: 1336
-> O tempo de execucao foi de 0.215195 segundos

      OBRIGADO POR UTILIZAR O PROGRAMA!
==6017==
==6017== HEAP SUMMARY:
==6017==    in use at exit: 0 bytes in 0 blocks
==6017==   total heap usage: 4 allocs, 4 frees, 6,696 bytes allocated
==6017==
==6017== All heap blocks were freed -- no leaks are possible
==6017==
==6017== For counts of detected and suppressed errors, rerun with: -v
==6017== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

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

```
int Recursiva(int Matriz[][100],int QuantLinhas,int x,int y){
    if(QuantLinhas-1 == x){
        return Matriz[x][y];
    }
    else{
        return Matriz[x][y] + Maior(Recursiva(Matriz,QuantLinhas,x+1,y+1),Recursiva(Matriz,QuantLinhas,x+1,y));
    }
}
```

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

```
int TopMemoization(int Matriz[][100],int QuantLinhas,int x,int y){
    int MatrizAux[QuantLinhas][100];
    for(int i=0;i<QuantLinhas;i++){
        for(int j=0;j<100;j++){
            MatrizAux[i][j] = -1; //Iniciando matriz para chamada de Memoization.
        }
    }
    return Memoization(Matriz,QuantLinhas,x,y,MatrizAux);
}

int Memoization(int Matriz[][100],int QuantLinhas,int x,int y,int MatrizAux[][100]){
    if(QuantLinhas-1 == x){
        MatrizAux[x][y] = Matriz[x][y]; //So preencho para poder usar matriz de pesos posteriormente
        return Matriz[x][y];
    }
    else{
        if(MatrizAux[x][y] == -1){
            MatrizAux[x][y] = Matriz[x][y] + Maior(Memoization(Matriz,QuantLinhas,x+1,y+1,MatrizAux),Memoization(Matriz,QuantLinhas,x+1,y,MatrizAux));
        }
        return MatrizAux[x][y];
    }
}
```

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

```
int DeTrasPraFrente(int Matriz[][100],int QuantLinhas){
    for(int i=QuantLinhas-2;i>=0;i--){ //QuantLinhas-2 devido a começar do 0;
        for(int j=0;j<i+1;j++){
            Matriz[i][j] += Maior(Matriz[i+1][j],Matriz[i+1][j+1]);
        }
    }
    return Matriz[0][0];
}

int TopDeTrasPraFrente(int Matriz[][100],int QuantLinhas){
    int MatrizAux[QuantLinhas][100];
    CopiaMatriz(Matriz,MatrizAux,QuantLinhas);
    return DeTrasPraFrente(MatrizAux,QuantLinhas);
}
```

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

```
void MelhorCaminho(int QuantLinhas,int x,int y,int MatrizAux[][100]){
    MatrizAux[x][y]=-2; //Valor de referencia.
    if(x<QuantLinhas-1){
        if(MatrizAux[x+1][y]>MatrizAux[x+1][y+1]){
            MelhorCaminho(QuantLinhas,x+1,y,MatrizAux);
        }
        else{
            MelhorCaminho(QuantLinhas,x+1,y+1,MatrizAux);
        }
    }
    return;
}
```

Fonte: Funcoes.c

Figura 14

```

Rota De Tras pra Frente:
(29)
42 (67)
50 (45) 5
15 16 (29) 8
37 64 8 (65) 65
42 1 2 (75) 48 25
66 31 76 29 (40) 43 31
8 34 44 37 (77) 11 40 74
68 7 43 49 16 (80) 13 24 97
30 66 98 84 93 (99) 10 11 82 86
41 74 81 24 82 16 (68) 72 45 31 12
19 99 19 14 48 87 (95) 13 64 44 43 82
95 79 28 94 89 39 28 (28) 32 2 61 8 85
29 76 9 74 59 73 46 (58) 44 60 58 32 7 71
48 52 66 82 47 46 10 (93) 87 2 21 67 86 23 29
95 60 58 23 21 85 83 (94) 31 93 39 43 52 23 51 23
23 55 42 5 54 88 16 (99) 27 70 20 95 56 95 24 3 8
34 27 29 19 62 76 2 (55) 15 46 7 90 49 83 13 4 77 18
10 65 86 9 44 56 29 39 (65) 76 15 68 84 50 95 66 21 57 42
24 65 57 70 72 99 19 7 64 (75) 84 34 85 1 21 94 46 29 75 37

OBRIGADO POR UTILIZAR O PROGRAMA!
```

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](#).