

Universidade Federal De Viçosa (UFV) - Campus Florestal
Disciplina: Projeto e Análise de Algoritmos
Professor(a): Daniel Mendes Barbosa (CCF330)

Trabalho prático 2

Programação Dinâmica

Autores:

Miguel Antônio Ribeiro e Silva - 4680

Mateus Henrique Vieira Figueiredo - 4707

Alan Gabriel Martins Silva - 4663

Sumário

1 - Introdução	3
2 - Metodologia	4
3 - Desenvolvimento	5
3.1 - travel.c e travel.h	5
3.2 - file.c e file.h	6
3.3 - matrix.c e matrix.h	9
3.4 - main.c	16
4 - Resultados	18
5 - Conclusão	21
6 - Referências	21

1 - Introdução

O trabalho prático tem como principal problema a elaboração de um programa que calcule a quantidade de caminhos de custo mínimo, usando programação dinâmica.

O jovem E.T, Lancelot Alfredo II, precisa calcular a quantidade de caminhos mínimos para fazer uma viagem intergalática, com o objetivo de entregar uma mensagem criptografada, a pedido de seu pai. Sabe-se que a viagem foi mapeada para uma matriz de N linhas e M colunas e que a rota que ele deve percorrer obedece a seguinte proposta:

Determine a quantidade de caminhos mínimos possíveis começando do campo $(1, 1)$ e terminando no campo (N, M) , com movimentos apenas para direita e para baixo.

É sabido que exista pelo menos um caminho possível.

2 - Metodologia

Após a formação do grupo, destacamos as principais tarefas a serem cumpridas para a realização do trabalho prático. Como:

- Implementação da entrada por arquivos de texto.
- Desenvolvimento de funções simples.
- Pesquisas e estudos sobre as características de um algoritmo com programação dinâmica.
- Descriptografar a mensagem.
- Criação das funções necessárias para o funcionamento da programação dinâmica, de forma iterativa.
- Interatividade com o usuário.
- Funções extras, testes e gráficos.
- Makefile e documentação.

O código fonte foi desenvolvido em **C**[1] e versionado no **GitHub**[2], visando que seria a melhor forma de compartilhamento do mesmo entre os integrantes do grupo. Para uma melhor organização e visualização do projeto, este foi dividido em subpastas.

-/src - implementação dos arquivos **.c** e **.h**.

-/testes - arquivos **.txt** usados para testes.

Para compilar e executar o projeto, é necessário ter [gcc](#) e [make](#) instalado em sua máquina.

Comandos:

-make

Caso não funcione, tente:

**-gcc -o main src/file.c src/matrix.c src/travel.c src/main.c
./main**

3 - Desenvolvimento

Diversas funções foram criadas, organizadas na pasta /src dentre diversos arquivos fontes e cabeçalho, todas estão devidamente comentadas e referenciadas.

3.1 - travel.c e travel.h

Nesses arquivos, as funções (**figura 01**) necessárias para a elaboração da viagem intergaláctica, explicitada anteriormente, foram implementadas.

```
void minSum(long long int rows, long long int cols, long long int **m, long long int **p);  
  
long long int numPaths(long long int rows, long long int cols, long long int **p, long long int **m, long long int **numPaths);  
  
void printCoordinates(long long int rows, long long int cols, long long int **p);  
  
void colorPath(long long int rows, long long int cols, long long int **p, long long int **m, long long int **colorMatrix);  
  
long long int maxSum( long long int rows, long long int cols, long long int **m, long long int **p);  
  
long long int minSumDiagonal ( long long int rows, long long int cols, long long int **m, long long int **minPD);  
  
long long int numPathsSumWithDiagonal( long long int rows, long long int cols, long long int **m, long long int **minPD, long lon
```

Figura 01 - travel.h.

- ***void minSum(long long int rows, long long int cols, long long int **m, long long int **p);***
 - ***função:*** calcular a soma mínima dos custos dos campos, de (0,0) até (n,m)
 - ***parâmetros:***
 - ***rows:*** número total de linhas nas matrizes.
 - ***cols:*** número total de colunas das matrizes
 - ***m:*** matriz com o custo de viagem para cada campo, (matriz de campo)
 - ***p:*** matriz com a soma mínima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
 - ***retorno:*** não possui.
 - ***detalhamento:*** essa é a função (**figura 02**) para calcular o custo mínimo de cada **campo ij da matriz até o campo nm**.

Funciona da seguinte maneira, percorrendo a matriz de cima para baixo: se a posição da vez, na matriz *m*, é a última (*n*,*m*), o custo do último campo, na matriz *p*, é o mesmo. Caso a condição seja falsa, há três verificações: Se o campo atual estiver na última coluna, a soma mínima será a soma do campo atual e do campo abaixo dele. Se o campo atual estiver na última linha, a soma mínima será a soma do campo atual e do campo à direita dele. Se não estiver na última linha nem na última coluna, o valor mínimo é o valor do campo atual mais o menor valor entre o campo abaixo ou o campo à direita. A partir destas verificações, ele preenche os caminhos de custo mínimo, na matriz *p*. Funciona iterativamente, através de um comando **while**.

```
void minSum(long long int rows, long long int cols, long long int **m, long long int **p)
{
    long long int xCount = rows - 1;
    long long int yCount = cols - 1;
    long long int xPos = xCount;
    long long int yPos = yCount;
    while (yCount != -1)
    {
        if (xPos == rows - 1 && yPos == cols - 1)
        {
            p[rows - 1][cols - 1] = m[rows - 1][cols - 1];
        }
        else
        {
            if (xPos == rows - 1)
            {
                p[xPos][yPos] = m[xPos][yPos] + p[xPos][yPos + 1];
            }

            else if (yPos == cols - 1)
            {
                p[xPos][yPos] = m[xPos][yPos] + p[xPos + 1][yPos];
            }

            else
            {
                if (p[xPos][yPos + 1] <= p[xPos + 1][yPos])
                {
                    p[xPos][yPos] = m[xPos][yPos] + p[xPos][yPos + 1];
                }
            }
        }
        yPos--;
    }
}
```

Figura 02 - Parte da função minSum.

- ***long long int numPaths(long long int rows, long long int cols, long long int **p, long long int **m, long long int **numpaths***
 - ***função:*** descobrir o número de caminhos mínimos.
 - ***parâmetros:***
 - ***rows:*** número total de linhas nas matrizes.
 - ***cols:*** número total de colunas das matrizes.
 - ***m:*** matriz com o custo de viagem para cada campo, (matriz de campo)
 - ***p:*** matriz com a soma mínima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
 - ***numpaths:*** matriz com o número de caminhos mínimos possíveis de cada campo até o campo (n,m), utilizada para a programação dinâmica.
 - ***retorno:*** (long long int) número de caminhos possíveis de (0,0) até (n,m).
 - ***detalhamento: (figura 03)*** Primeiro, define o último campo da matriz **numpaths** com o valor 1, pois é o único caminho até o momento. Após isso, percorre a matriz de baixo para cima e da direita para esquerda, usando **dois comandos for**. Dentro dos comandos, ele faz 4 verificações, se a posição atual é a última, o algoritmo não faz nada. Se estiver na última linha da **matriz p**, pega o valor da direita e soma com o valor atual da **matriz m**, caso a soma for igual ao elemento **p[i][j]**, soma-se 1 ao número de caminhos naquela posição. Se estiver na última coluna da **matriz p**, pega o valor de baixo e soma com o valor atual da **matriz m**, caso a soma for igual ao elemento **p[i][j]**, soma-se 1 ao número de caminhos naquela posição. Se estiver em qualquer outra posição da **matriz p**, pega o valor da **direita ou de baixo (duas verificações)** e soma com o valor da **matriz m**. Se a soma for igual ao elemento **p[i][j]**, soma-se 1 ao número de caminhos naquela posição. A partir destas verificações, ele preenche a quantidade de caminhos de custo mínimo, na matriz numpaths. Funciona iterativamente, através de um comando **for**.

```

if (i == rows - 1 && j == cols - 1)
{
    // If we are on the last field, we don't need to do anything.
    continue;
}

else if (i == rows - 1)
{
    /*
     * Se estiver na última linha da matriz p, pega o valor da direita e soma com o valor da matriz m.
     * Se a soma for igual ao elemento p[i][j], soma 1 ao número de caminhos naquela posição.
     */
    if (p[i][j] == p[i][j + 1] + m[i][j])
    {
        numPaths[i][j] = numPaths[i][j + 1];
    }
}

else if (j == cols - 1)
{
    /*
     * Se estiver na última coluna da matriz p, pega o valor de baixo e soma com o valor da matriz m.
     * Se a soma for igual ao elemento p[i][j], soma 1 ao número de caminhos naquela posição.
     */
    if (p[i][j] == p[i + 1][j] + m[i][j])
    {
        numPaths[i][j] = numPaths[i + 1][j];
    }
}

```

Figura 03 - Parte da função numPaths.

- ***long long int maxSum(long long int rows, long long int cols, long long int **m, long long int **p)***
 - ***função: (FUNÇÃO EXTRA)*** calcular a soma máxima dos custos dos campos, de (0,0) até (n,m).
 - ***parâmetros:***
 - ***rows:*** número total de linhas nas matrizes.
 - ***cols:*** número total de colunas das matrizes.
 - ***m:*** matriz com o custo de viagem para cada campo, (matriz de campo)
 - ***p:*** matriz com a soma máxima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
 - ***retorno:*** (long long int) soma máxima.
 - ***detalhamento:*** funciona da mesma forma que a **minSum**, só que agora, computa na matriz **p**, a soma máxima.

- ***long long int minSumDiagonal (long long int rows, long long int cols, long long int **m, long long int **minPD)***

- **função: (FUNÇÃO EXTRA)** calcular a soma mínima dos custos dos campos, de (0,0) até (n,m), só que agora, usando a diagonal direita como possibilidade de viagem.
- **parâmetros:**
 - **rows:** número total de linhas nas matrizes.
 - **cols:** número total de colunas das matrizes.
 - **m:** matriz com o custo de viagem para cada campo, (matriz de campo)
 - **minPD:** matriz com a soma máxima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
- **retorno:** (long long int) soma mínima.
- **detalhamento:** funciona da mesma forma que a **minSum**, só que agora, computa na matriz **p**, a soma mínima, considerando também a diagonal baixa para direita.

- ***long long int numPathsSumWithDiagonal(long long int rows, long long int cols, long long int **m, long long int **minPD, long long int **numPD)***

- **função: (FUNÇÃO EXTRA)** descobrir o número de caminhos mínimos, considerando também, a diagonal.
- **parâmetros:**
 - **rows:** número total de linhas nas matrizes.
 - **cols:** número total de colunas das matrizes.
 - **m:** matriz com o custo de viagem para cada campo, (matriz de campo)
 - **minPD:** matriz com a soma mínima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
 - **numPD:** matriz com o número de caminhos mínimos possíveis de cada campo até o campo (n,m), utilizada para a programação dinâmica.
- **retorno:** (long long int) número de caminhos possíveis de (0,0) até (n,m).
- **detalhamento:** funciona da mesma forma que a função **numPaths**, só que agora, verifica a diagonal baixa para direita.

- ***void printCoordinates(long long int rows, long long int cols, long long int **p)***
 - **função: (FUNÇÃO EXTRA)** imprimir no terminal (**figura 04**) uma das coordenadas do caminho mínimo.
 - **parâmetros:**
 - **rows:** número total de linhas nas matrizes.
 - **cols:** número total de colunas das matrizes.
 - **p:** matriz com a soma mínima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
 - **retorno:** não possui.
 - **detalhamento:** utiliza a matriz **p** de caminhos mínimos para calcular as coordenadas de um dos caminhos mínimos gerados, imprimindo no terminal.

```
Coordinates:
(0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (0, 4) -> (0, 5) -> (0, 6) -> (0, 7) -> (0, 8) -> (0, 9) -> (1, 9) -> (2, 9) -> (3, 9)
-> (4, 9) -> (5, 9) -> (6, 9) -> (7, 9) -> (8, 9) -> (9, 9) -> END
```

Figura 04 - coordenadas da matriz 10 x 10 da especificação.

- ***void colorPath(long long int rows, long long int cols, long long int **p, long long int **m, long long **colorMatrix)***
 - **função: (FUNÇÃO EXTRA)** imprimir no terminal (**figura 05**) uma matriz. Em vermelho, o caminho mínimo impresso pelas coordenadas anteriormente.
 - **parâmetros:**
 - **rows:** número total de linhas nas matrizes.
 - **cols:** número total de colunas das matrizes.
 - **m:** matriz com o custo de viagem para cada campo, (matriz de campo)
 - **p:** matriz com a soma mínima dos custos dos campos de cada campo ao campo(n, m), utilizada para a programação dinâmica.
 - **colorPath:** matriz com 0s e 1s, em 1 está o caminho mínimo.
 - **retorno:** não possui.
 - **detalhamento:** utiliza a matriz **p** de caminhos mínimos para calcular as coordenadas de um dos caminhos mínimos gerados,

verifica a coordenada do caminho, e insere na mesma coordenada da matriz **colorPath**, o valor 1. A partir da matriz **colorPath** gerada, ela auxilia a matriz **m** a imprimir os valores coloridos do caminho mínimo.

```
Color matrix:
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
```

Figura 05 - Color matrix da matriz especificação 10 x 10.

3.2 - file.c e file.h

Funções (**figura 06**) necessárias para a leitura e criação de arquivos de texto, usados para testes.

```
long long int **readFileIntoMatrix(char *filepath, long long int *rows, long long int *cols);
char *generateRandomFile(long long int *rows, long long int *cols);
char *generateRandomFileWithInput(long long int *rows, long long int *cols, long long int *range);
```

Figura 06 - file.h.

- ***long long int **readFileIntoMatrix(char *filename, long long int *rows, long long int *cols):***

- **função:** ler um arquivo de texto, formatado de acordo com a especificação e inserir seus dados em uma matriz de inteiros.
- **parâmetros:**
 - **filename:** nome do arquivo.
 - **rows:** ponteiro para um inteiro que armazenará o número de linhas da matriz.
 - **cols:** ponteiro para um inteiro que armazenará o número de colunas da matriz.
- **retorno:** (long long int) matriz de inteiros.
- **detalhamento:** a função lê a primeira linha do arquivo passado como parâmetro e inicializa uma matriz rows x cols, alocada dinamicamente (stdlib.h). Após isso, a preenche com os dados restantes do arquivo.

- ***char *generateRandomFile(long long int *rows, long long int *cols):***

- **função: (FUNÇÃO EXTRA)** gerar um arquivo de texto para testes, com dados aleatórios.
- **parâmetros:**
 - **rows:** ponteiro para um inteiro que armazenará o número de linhas da matriz.
 - **cols:** ponteiro para um inteiro que armazenará o número de colunas da matriz.
- **retorno:** (char) caminho do arquivo.
- **detalhamento:** a função (**figura 07**) primeiramente define o número de linhas e colunas usando **rand** e **srand** (time.h); por padrão, a matriz contém no máximo **100x100** inteiros. Após isso, é gerado inteiros aleatórios, **entre 0 e 100 (figura 08)** e os insere no arquivo.

- ***char *generateRandomFileWithInput(long long int *rows, long long int *cols, long long int *range):***
 - ***função: (FUNÇÃO EXTRA)*** gerar um arquivo de texto para testes, com número de linhas e colunas e o range máximo de geração de números aleatórios definidos pelo usuário.
 - ***parâmetros:***
 - ***rows:*** ponteiro para um inteiro que armazenará o número de linhas da matriz.
 - ***cols:*** ponteiro para um inteiro que armazenará o número de colunas da matriz.
 - ***range:*** ponteiro para um inteiro que armazenará o range de geração de números.
 - ***retorno:*** (char) caminho do arquivo.
 - ***detalhamento:*** Primeiramente o usuário escolhe o número de linhas e colunas. Após isso, ele escolhe o range de geração de números, que são gerados aleatoriamente e os insere no arquivo.

```
fprintf(file, "%lld %lld \n", *rows, *cols);

for (long long int i = 0; i < *rows; i++)
{
    for (long long int j = 0; j < *cols; j++)
    {
        fprintf(file, "%d ", rand() % 100);
    }

    fprintf(file, "\n");
}

fclose(file);
file = NULL;

return filepath;
```

Figura 07 - Números aleatórios.

```

20 27
5 27 28 12 94 99 47 47 2 46 86 13 70 97 35 26 92 83 3 64 23 25 63 64 19 70 76
37 16 96 15 21 23 44 85 17 95 32 16 98 78 2 63 1 51 50 27 43 85 82 59 9 7 23
73 26 45 2 16 61 98 83 34 21 27 20 90 23 52 6 73 83 9 36 84 60 38 11 56 24 45
15 33 52 90 58 31 88 12 99 1 10 82 36 83 62 8 73 37 12 32 10 95 41 98 79 53 36
42 61 60 39 29 45 92 19 56 75 59 68 74 61 31 8 97 14 70 57 88 59 69 72 69 65 13
19 96 18 8 91 80 20 30 9 18 74 80 74 49 40 94 75 1 25 36 50 92 6 7 32 66 28
4 87 93 17 59 42 87 19 33 67 39 15 28 9 90 9 35 91 1 30 19 2 7 55 4 99 13
63 31 31 91 35 71 37 4 82 79 92 1 64 11 92 79 40 2 21 1 37 65 54 19 84 8 79
91 12 78 4 75 62 88 18 97 59 55 54 41 34 46 94 50 9 86 30 1 40 3 54 78 68 8
49 52 16 28 43 28 59 0 55 21 88 26 70 47 81 24 40 68 22 34 18 32 72 0 85 13 4
40 43 72 48 92 77 17 73 72 45 84 72 53 5 60 79 27 59 12 4 51 80 26 37 51 10 10
51 48 23 55 88 66 80 88 10 9 5 35 81 3 19 54 56 24 66 87 4 78 99 8 81 32 86
19 83 97 29 34 45 4 42 85 22 74 73 32 83 79 68 64 34 87 70 42 64 37 29 20 67 80
28 48 12 14 19 47 63 0 82 8 4 76 45 26 50 19 11 85 50 31 49 84 18 20 26 34 9
7 54 28 87 34 76 52 49 48 99 12 48 33 73 5 61 18 83 63 37 94 48 39 25 98 23 96
70 1 82 31 8 37 59 48 71 35 52 72 35 3 85 84 89 10 89 50 28 72 14 18 19 14 57
44 64 33 92 34 34 75 65 95 64 24 95 87 12 99 60 47 2 97 83 91 7 72 94 87 97 60
5 16 74 63 12 39 48 5 73 82 80 91 29 96 67 76 83 31 75 95 31 78 44 14 21 3 39
15 91 36 75 96 4 2 11 16 41 59 21 66 94 53 9 75 49 77 52 85 60 79 32 91 9 29
6 31 32 97 98 75 85 26 24 89 28 35 5 21 47 27 87 93 32 97 68 34 26 72 71 86 52

```

Figura 08 - Exemplo de um arquivo gerado randomicamente.

3.3 - matrix.c e matrix.h

Função usada para criação de matriz (**figura 09**).

```

long long int **initializeMatrix(long long int rows, long long int cols);

```

Figura 09 - matrix.h.

- ***long long int **initializeMatrix(long long int rows, long long int cols):***
 - ***função:*** inicializar uma matriz de inteiros com um determinado número de linhas e colunas.
 - ***parâmetros:***
 - ***rows:*** número de linhas da matriz.
 - ***cols:*** número de colunas da matriz.
 - ***retorno:*** (int) matriz de inteiros
 - ***detalhamento:*** aloca uma matriz dinamicamente.

3.4 - main.c

Programa principal do projeto.

Ao executar o programa, um menu (**figura 10**) será impresso na tela e o usuário deverá escolher uma das opções.

```
Welcome!

1 - Enter file path
2 - Generate random matrix
3 - Define parameters before generating random matrix
4 - Extra options
5 - Exit

Enter option: █
```

Figura 10 - Menu do programa

1 -Enter file path.

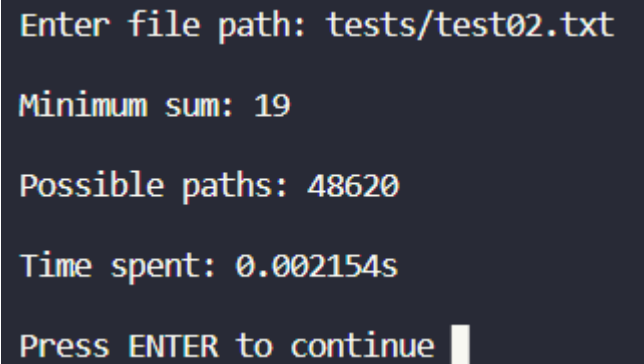
Selecionando essa opção, o usuário deverá digitar o diretório do arquivo que deseja usar. Após a escolha, o arquivo será aberto, e as funções necessárias para para o cálculo da soma mínima e quantidade de caminhos de custo mínimo são chamadas. Ao fim, o resultado (**figura 11**) será impresso no terminal, juntamente com o tempo de execução. Após isso, o usuário decide se deseja ver as coordenadas de um dos caminhos mínimos (**figura 04**).

2 - Generate random matrix.

Selecionando essa opção, um arquivo aleatório é gerado, salvo na pasta **/tests** como explicado anteriormente e executado em sequência. As funções para o cálculo da soma mínima e quantidade de caminhos de custo mínimo são chamadas. É impresso no terminal as propriedades do arquivo gerado e o resultado, juntamente com a medição do tempo. Após isso, o usuário decide se deseja ver as coordenadas de um dos caminhos mínimos (figura 04).

3 - Define parameters before generating a random matrix.

Esta opção é semelhante a anterior, só que o usuário poderá definir o número de linhas e colunas da matriz e o alcance dos números, gerados aleatoriamente. É impresso (**figura 12**) no terminal as propriedades do arquivo gerado e o resultado, juntamente com a medição do tempo. Após isso, o usuário decide se deseja ver as coordenadas de um dos caminhos mínimos (figura 04).

A terminal window with a dark background and light-colored text. The text is as follows:

```
Enter file path: tests/test02.txt
Minimum sum: 19
Possible paths: 48620
Time spent: 0.002154s
Press ENTER to continue
```

Figura 11 - Teste da matriz especificação do trabalho.


```
Enter number of rows: 500
Enter number of columns: 1000
Enter range of values: 10
Generating random matrix, please wait

=====
| Number of rows: 500 |
| Number of columns: 1000 |
| Range of numbers: 0 - 10 |
| Filepath: tests/random-500-1000.txt |
=====
File generated.

Minimum sum: 3662

Possible paths: 1408964021452800

Time spent: 0.063048s

Press ENTER to continue █
```

Figura 12 - Usuário definiu os parâmetros.

4 - Extra options

Lancelot Alfredo II é um E.T muito enrolado e decidiu realizar a viagem, a pedido de seu pai, **da forma mais longa possível**, para evitar outras tarefas. Por isso pediu para sua equipe de programadores que projetassem também, os caminhos cujo custo de viagem seja máximo.

Assim o fez, e essa opção **EXTRA** mostra para o usuário a soma máxima de custo e a quantidade de caminhos possíveis (**figura 13**).

Seu pai, sabendo do ocorrido, imediatamente chamou a atenção de seu filho e lhe deu um castigo. Agora, o pequeno E.T, para as próximas viagens, deverá realizá-las de forma muito mais rápida que o comum. Para que isso fosse feito, foi projetada uma nave que consiga **mover diagonalmente entre as galáxias**, reduzindo ainda mais, o custo mínimo da viagem e pediu para sua equipe de programadores, que calculassem esse caminho.

Assim o fez, e essa opção **EXTRA** mostra para o usuário a soma mínima de custo, considerando também as diagonais para direita.

```
Enter filepath: tests/test02.txt
Maximum sum: 19
Possible paths: 48620
Minimum sum with diagonal: 10
Possible paths: 1
Press ENTER to continue []
```

Figura 13: opções extras da matriz 10x10 da especificação

5 - Exit.

Encerra o programa.

- **void clearConsole():**

- **função:** limpa o terminal baseado no sistema operacional do usuário.
- **parâmetros:** não possui.
- **retorno:** não possui.
- **detalhamento:** se o usuário estiver usando o sistema operacional Windows, system("cls") é ativo, caso contrário, system("clear") é a opção.
-

- **void flush_in():**

- **função:** limpeza do buffer de entrada do teclado.
- **parâmetros:** não possui.
- **retorno:** não possui.
- **detalhamento:** ver [3]

5 - Resultados

A seguir, alguns testes foram feitos e registrados em tabelas:

test01.txt

- **matriz:** 3x3
- **soma mínima:** 4
- **número de caminhos mínimos possíveis:** 1
- **tempo de execução:** 0.002415s

Matriz teste 1, da especificação.

test02.txt

- **matriz:** 10x10
- **soma mínima:** 19
- **número de caminhos mínimos possíveis:** 48620
- **tempo de execução:** 0.002959s

Matriz teste 2, da especificação.

test03.txt

- **matriz:** 2x3
- **soma mínima:** 12
- **número de caminhos mínimos possíveis:** 1
- **tempo de execução:** 0.002712s

Matriz teste 3, da especificação.

random-83-79.txt

- **matriz:** 83 x 79
- **range:** 0 - 100
- **soma mínima:** 3807
- **número de caminhos mínimos possíveis:** 2
- **tempo de execução:** 0.002629s

Arquivo gerado pela opção 2 do menu.

random-69-44.txt

- **matriz:** 69 x 44
- **range:** 0 - 100
- **soma mínima:** 2964
- **número de caminhos mínimos possíveis:** 1
- **tempo de execução:** 0.002539s

Arquivo gerado pela opção 2 do menu.

random-500-500.txt

- **matriz:** 500 x 500
- **range:** 0 - 50
- **soma mínima:** 12659
- **número de caminhos mínimos possíveis:** 128
- **tempo de execução:** 0.027340s

Arquivo gerado pela opção 3 do menu.

random-1000-1000.txt

- **matriz:** 1000 x 1000
- **range:** 0 - 30
- **soma mínima:** 14464
- **número de caminhos mínimos possíveis:** 4096
- **tempo de execução:** 0.103843s

Arquivo gerado pela opção 3 do menu.

random-5000-5000.txt

- **matriz:** 5000 x 5000
- **range:** 0 - 10
- **soma mínima:** 22787
- **número de caminhos mínimos possíveis:** Perto de infinito.
- **tempo de execução:** 3.600270s

Arquivo gerado pela opção 3 do menu, quanto maior a matriz, a quantidade de caminhos possíveis aumenta, ainda mais com um range tão baixo.

random-5000-5000(01).txt
<ul style="list-style-type: none">• matriz: 5000 x 5000• range: 0 - 1000• soma mínima: 2523683• número de caminhos mínimos possíveis: 2• tempo de execução: 3.204651s

Segundo teste com 5000x5000, com um range maior, a quantidade de caminho caiu mas a soma mínima aumentou, obviamente.

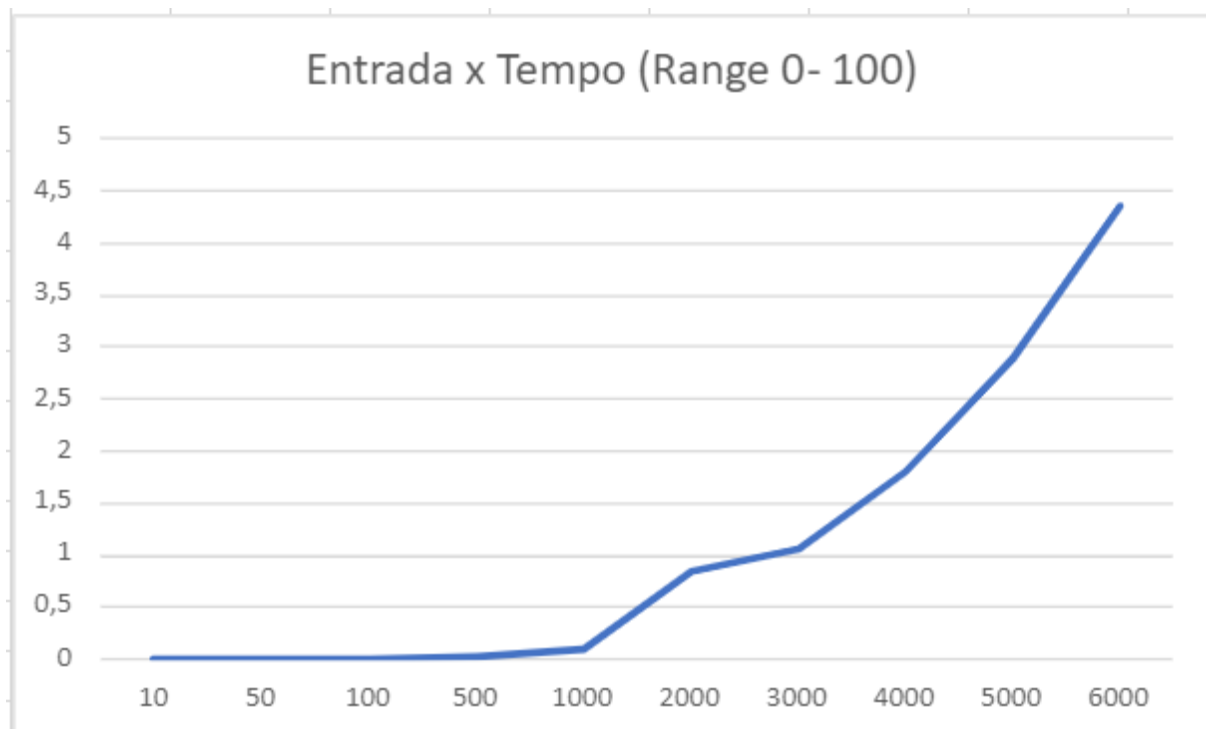
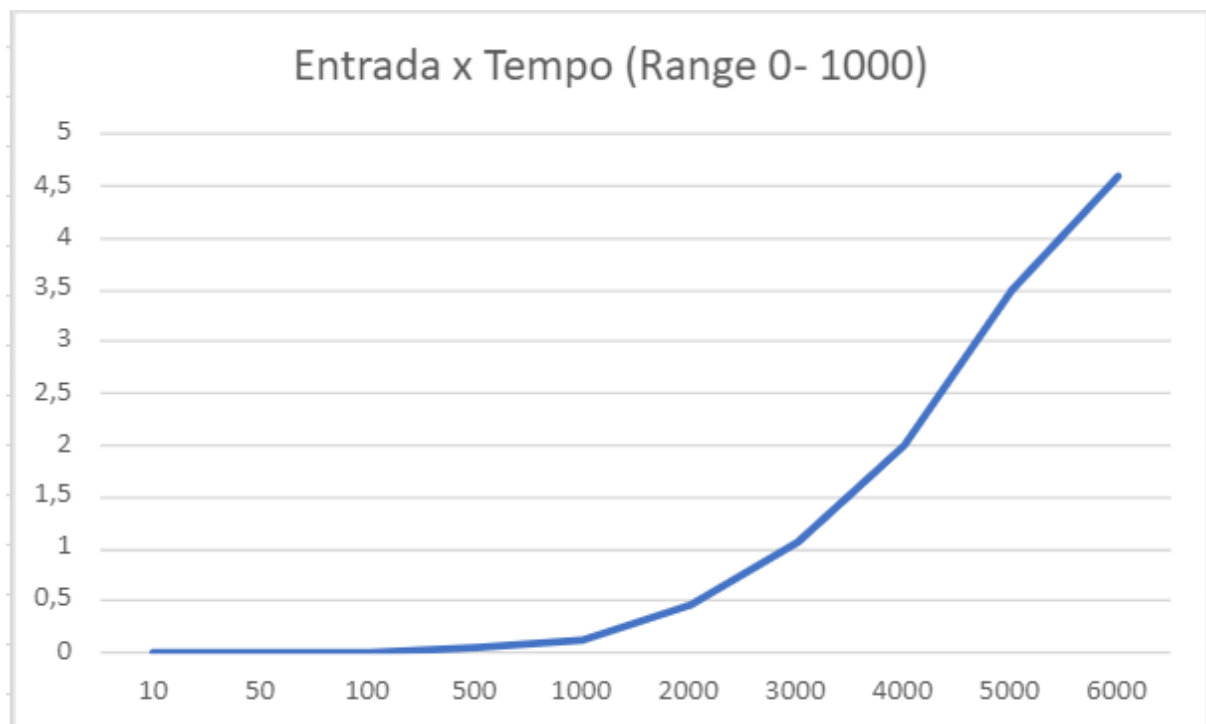
random-6000-6000.txt
<ul style="list-style-type: none">• matriz: 6000 x 6000• range: 0 - 1000• soma mínima: 3022623• número de caminhos mínimos possíveis: 1024• tempo de execução: 4.917236s

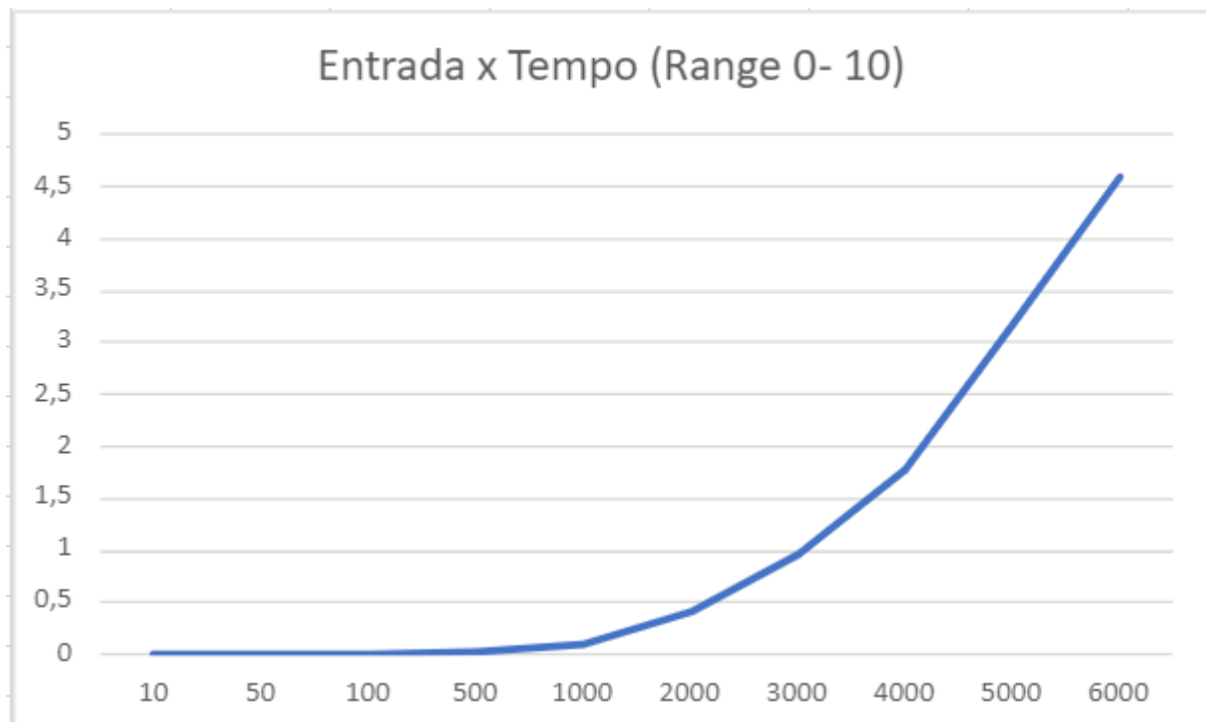
Teste semelhante ao anterior, com 6000 x 6000.

Observações:

- Foram testados centenas de arquivos, alguns deles estão na pasta **/tests** e podem ser executados.
- Os números da matriz randomizada possuem limite de 6000 para linhas e colunas e 1000 para range de aleatoriedade.
- O tempo medido é apenas na execução das funções **minSum** e **numPaths**, a função de gerar arquivo é a que demanda mais processamento.
- Foram realizados testes com as funções extras, mas não foram computados.

Outros testes foram realizados e gráficos foram gerados pelo **Excel**:





Observações:

- Verifica-se que os gráficos seguem um padrão, independentemente do range de aleatoriedade.
- Para N,M muito grande, acima de 6000 x 6000 a quantidade de caminhos possíveis era tão grande, que por vezes era impossível de calcular, dificultando a medição do tempo.
- Legenda: **x - N M (10 lê se 10 x 10).**
y - Tempo, em segundos.
- Foram realizados testes com as funções extras, mas não foram computados.

5 - Conclusão

Após o fim do trabalho prático podemos apontar certas dificuldades e facilidades encaradas pelo grupo durante a implementação desse projeto.

Notamos uma certa facilidade na hora de entender como o problema deveria ser resolvido.

Tivemos dificuldades de implementar a programação dinâmica e principalmente a soma dos caminhos, mas com algumas pesquisas e estudos obtivemos sucesso [4].

Os resultados obtidos, foram úteis na compreensão de como um algoritmo com programação dinâmica funciona, explicitando suas peculiaridades e possibilidades.

Por fim é visível o proveito e as lições aprendidas durante esse período de desenvolvimento do trabalho prático.

6 - Referências

[1] [C \(programming language\) - Wikipedia](#)

[2]

<https://github.com/Mateus-Henr/DynamicProgrammingOnMessageDelivering>

[3] [Limpeza do buffer do teclado após scanf - Stack Overflow em Português](#)

[4] Ziviani N. , Projeto de Algoritmos com implementações em Pascal e C