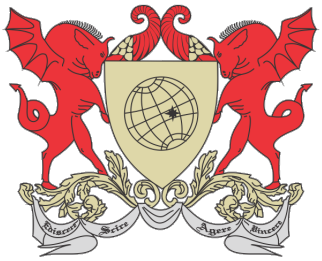

UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho Prático 2
Projeto e Análise de Algoritmos

Artur Papa - 3886
Guilherme Fernandes - 3398

Trabalho Prático apresentado à disciplina de CCF 330 - Projeto e Análise de Algoritmos do curso de Ciência da Computação da Universidade Federal de Viçosa.

Florestal
Novembro de 2022

CCF 330 - Projeto e Análise de Algoritmos

Trabalho Prático 2

Artur Papa - 3886
Guilherme Fernandes - 3398

3 de Novembro de 2022

Contents

1 Introdução

Primeiramente, sabe-se que muitos algoritmos eficientes são baseados no método de *programação dinâmica*. Esse método, ou estratégia de projeto de algoritmos é uma espécie de tradução iterativa inteligente da recursão e pode ser definido vagamente como "recursão com o apoio de uma tabela". [1]

Dessa maneira, algo que é muito comum em programação dinâmica é a questão de dividir um problema em subproblemas. Dizemos que um problema possui subproblemas sobrepostos se o problema puder ser dividido em subproblemas e cada subproblema for repetido várias vezes, ou um algoritmo recursivo para o problema resolver o mesmo subproblema repetidamente, em vez de sempre gerar novos subproblemas.

Outrossim, sabe-se que há duas abordagens para resolver os subproblemas, uma abordagem *top-down* e outra *bottom-up*. Vale ressaltar que a que foi usada nesse trabalho foi a abordagem *bottom-up*, haja visto que foi foi resolvido os subproblemas primeiro e depois usada as soluções para construir e chegar à soluções para subproblemas maiores. [2]

2 Desenvolvimento

2.1 Lógica do programa

Primeiramente, para o cálculo do caminho mínimo foi usado uma matriz de pesos para salvar o valor mínimo da distância. Primeiro inicializamos a primeira posição com o valor do primeiro peso do nosso arquivo, depois inicializamos a primeira linha e primeira coluna da nossa matriz de peso com os valores já colocados anteriormente mais o peso do nosso mapa.

Após ter feito isso, preenchemos a nossa tabela com o valor mínimo se movimentando para a direita e para baixo e sempre somando o respectivo peso do nosso arquivo. Dessa forma, a última posição da nossa tabela vai ser a soma mínima que podemos alcançar saindo da posição (0, 0) e indo para a posição (N, M).

```
int gridMap(map *map, int **memo)
{
    memo[0][0] = map->map[0][0];
    // Init the first row of memo
    for (int j = 1; j < map->width; j++){
        memo[0][j] = memo[0][j-1] + map->map[0][j];
    }

    for (int i = 1; i < map->height; i++){
        memo[i][0] = memo[i-1][0] + map->map[i][0];
    }

    for (int i = 1; i < map->height; i++)
    {
        for (int j = 1; j < map->width; j++)
        {
            memo[i][j] = min(memo[i-1][j], memo[i][j-1]) + map->map[i][j];
        }
    }

    return memo[map->height-1][map->width-1];
}
```

Figura 1: Cálculo para a soma mínima

Para calcularmos a quantidade de caminhos mínimos possíveis, foi usada a seguinte lógica. Seja T uma posição arbitrária, e P a posição de destino. Sejam v1, v2, v3, ..., vK vizinhos de T. Se T == P, então existe 1 único caminho de T para P, que é justamente ficar aonde está. Senão, a quantidade de caminhos de T para P é igual ao somatório da quantidade de caminhos de v1-¿P + v2-¿P + ... + vK-¿P. Para limitar aos caminhos mínimos, bastou limitar os v's para serem somados apenas os v's mínimos.

```
int numberWays(map *map, int **memo, int x, int y, int count, int before)
{
    if (x == (map->height - 1) && y == (map->width - 1))
    {
        return ++count;
    }
    else if (x == (map->height - 1) && y != (map->width - 1) && memo[x][y] >= before && memo[x][y] < memo[map->height-1][map->width-1])
    {
        count = numberWays(map, memo, x, y+1, count, memo[x][y]) + count;
    }
    else if (x != (map->height - 1) && y == (map->width - 1) && memo[x][y] >= before && memo[x][y] < memo[map->height-1][map->width-1])
    {
        count = numberWays(map, memo, x+1, y, count, memo[x][y]) + count;
    }
    else if (memo[x][y] >= before && memo[x][y] < memo[map->height-1][map->width-1])
    {
        count = numberWays(map, memo, x+1, y, count, memo[x][y]) + numberWays(map, memo, x, y+1, count, memo[x][y]) + count;
    }
    else
    {return 0;}
}
```

Figura 2: Quantidade de caminhos mínimos possíveis

2.2 Estrutura do projeto

Para a estrutura do projeto, foram criadas sete pastas, assim temos a seguinte arquitetura:

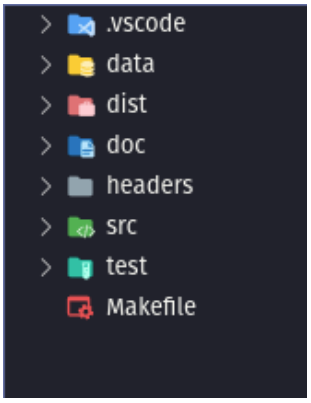

Figura 3: Estrutura do projeto

Dessa maneira, vale citar que a pasta **data** possui os arquivos de texto que serão utilizados no programa principal, caso queira adicionar mais um teste é necessário colocar nesta pasta, na pasta **dist** é onde se encontra as saídas dos programas principais, como por exemplo a *main*, após o programa ser compilado, na pasta **doc** encontra-se os documentos do projeto, no diretório **headers**, para a pasta **test** temos arquivos que foram usados apenas para testar determinadas funções, na pasta **src** é aonde se localiza os códigos fonte do projeto e os demais arquivos com suas funções.

2.3 Execução

Para a execução do projeto foi feita a escolha de adicionar um arquivo **Makefile**. Assim, para compilar e executar o programa basta realizar no terminal o comando **make**, para compilar os arquivos e *make run* para rodar o programa. Caso seja executado o programa várias vezes, é recomendável que seja executado o comando **make clean**, para limpar as compilações, e depois

executar os comandos para compilar e rodar o programa.

2.4 Casos de teste e resultado

Para os casos de teste, foram utilizados os três casos da especificação do TP.

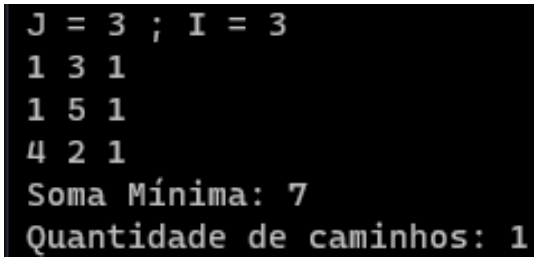

Figura 4: Resultado primeiro teste

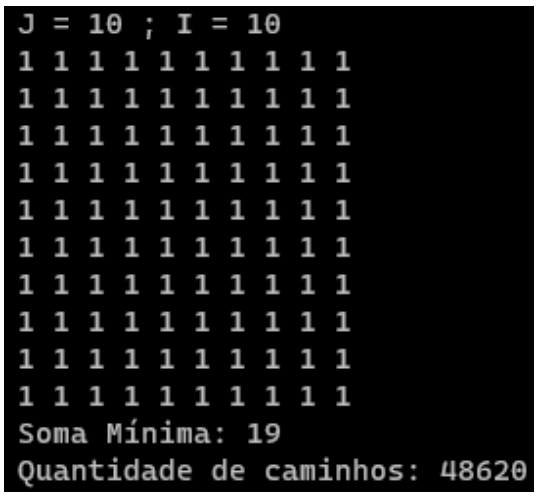

Figura 5: Resultado segundo teste

3 Conclusão

Posto isso, conclui-se que o trabalho em questão foi desenvolvido conforme o esperado, atingindo as especificações requeridas na descrição do mesmo em implementar um programa que utiliza ***programação dinâmica*** para determinar a quantidade de caminhos mínimos.

Por conseguinte pode-se ressaltar que o material apresentado na disciplina foi de suma importância para o desenvolvimento do projeto, haja vista que as explicações dadas pelo professor ajudou bastante a entender as etapas a serem executadas e na construção dos códigos, assim como as monitorias dadas pelo monitor da disciplina.

Em adição, é válido dizer que apesar das dificuldades na implementação do código, a dupla em questão foi capaz de superar e corrigir quaisquer erros no desenvolvimento do algoritmo. Por fim, verificou-se a assertiva para o objetivo do projeto implementar um programa que utiliza *programação dinâmica* para determinar a quantidade de caminhos mínimos.

References

[1] Programação dinâmica. https://www.ime.usp.br/~pf/analise$_$de$_$algoritmos/aulas/dynamic-programming.html. (Accessed on 27/11/2022).

[2] Programação dinâmica 2. https://www.techiedelight.com/pt/introduction-dynamic-programming/. (Accessed on 27/11/2022).