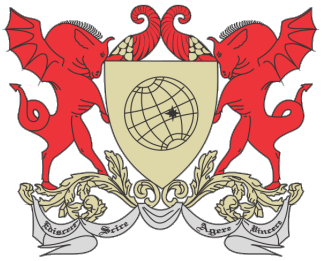

UNIVERSIDADE FEDERAL DE VIÇOSA
CAMPUS FLORESTAL

Trabalho Prático 1
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 1

Artur Papa - 3886
Guilherme Fernandes - 3398

3 de Novembro de 2022

Contents

1 Introdução

A força bruta pode ser eficiente na resolução de problemas simples na área da computação devido ao alto poder computacional, que permite a execução de bilhões de operações por segundo. Entretanto, existem casos em que a complexidade da situação exige técnicas sofisticadas, sendo uma delas o ***backtracking***.

Assim, sabe-se que o ***backtracking*** é um método para percorrer todas as possíveis configurações em um espaço qualquer. É um algoritmo geral que poderá ser personalizado para cada tipo de aplicação. De modo geral, a solução será um vetor *a = (a1, a2, ..., a-n)*, sendo cada elemento *a-i* selecionado de um conjunto *Si*. [1]

Desta forma, temos que o ***backtracking*** assegura o acerto por enumerar todas as possibilidades sem nunca visitar o mesmo estado, sendo também eficiente. Outrossim, tem-se como exemplos aplicavéis do ***backtracking*** problemas como o das **Oito rainhas** e o **Passeio do cavalo**.

2 Desenvolvimento

Dessa maneira, temos que foi proposto um problema para aplicação do ***backtracking***, neste problema, teríamos que completar um caminho ótimo em uma fazenda seguindo uma sequência de sequências de **Fibonacci** e, após ter sido feito esse percurso, imprimir as posições pelas quais o fazendeiro passou.

2.1 Lógica do programa

A principal função do programa é a ***movement***, que pode ser vista na Figura 1. Inicialmente a função recebe a matriz da fazenda, uma matriz do caminho correto que está sendo percorrido, a posição atual que está sendo avaliada na matriz da fazenda, um índice que irá mostrar em qual valor da sequência de sequências de fibonnaci que estamos e por fim a sequência de sequências de fibonacci.

A condicional principal da função irá verificar se o valor da posição atual presente na matriz da fazenda é igual o valor da sequência de sequências de fibonnaci apontado pelo índice. Se sim, ele irá adicionar a posição atual ao caminho correto e irá fazer o índice apontar para a próxima posição da sequência desejada, chamando a função ***movement*** agora para a posição acima na matriz, caso dê errado, a função é chamada para a posição abaixo, à esquerda ou então à direita.

Antes de chamar a função ***movement***, ele verifica se a posição já está presente no caminho, caso esteja ele irá pular esta chamada. Caso ele chegue na última linha, irá exibir o caminho correto e retornar *TRUE* encerrando as chamadas recursivas.

Caso nenhuma chamada da função ***movement***, em todos os valores da primeira linha retorne *TRUE*, irá ser mostrado na tela que não há nenhum caminho possível.

```c
int movement(Map *map, Data *data, int **track, int *actualPosition, int* index, int *sequence) {
    int result;
    int nextPosition[2];
    data->number_of_recursions++;

    if (map->map[actualPosition[0]][actualPosition[1]] == sequence[(*index)]) {
        track[(*index)][0] = actualPosition[0];
        track[(*index)][1] = actualPosition[1];

        if (actualPosition[0] == ((map->height)-1)) {
            printf("\n\nPosicoes: \n");
            printTrack(track, index);
            return TRUE;
        }

        (*index) = (*index) + 1;
        if (actualPosition[0] > 0) //movement para cima
        {
            nextPosition[0] = actualPosition[0]-1;
            nextPosition[1] = actualPosition[1];
            if (!inTracking(track, nextPosition, index)) {
                result = movement(map, data, track, nextPosition, index, sequence);
                if (result) return TRUE;
            }
        }

        if (actualPosition[0] < map->width-1) //movement para baixo
        {
            nextPosition[0] = actualPosition[0]+1;
            nextPosition[1] = actualPosition[1];
            if (!inTracking(track, nextPosition, index)) {
                result = movement(map, data, track, nextPosition, index, sequence);
                if (result) return TRUE;
            }
        }

        if (actualPosition[1] > 0) //movement para esquerda
        {
            nextPosition[0] = actualPosition[0];
            nextPosition[1] = actualPosition[1]-1;
            if (!inTracking(track, nextPosition, index)) {
                result = movement(map, data, track, nextPosition, index, sequence);
                if (result) return TRUE;
            }
        }

        if (actualPosition[0] < map->height-1) //movement para direita
        {
            nextPosition[0] = actualPosition[0];
            nextPosition[1] = actualPosition[1]+1;
            if (!inTracking(track, nextPosition, index)) {
                result = movement(map, data, track, nextPosition, index, sequence);
                if (result) return TRUE;
            }
        }
        (*index) = (*index) - 1;
    }
    return FALSE;
}
```

Figura 1: Função *movement*

2.2 Estrutura do projeto

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

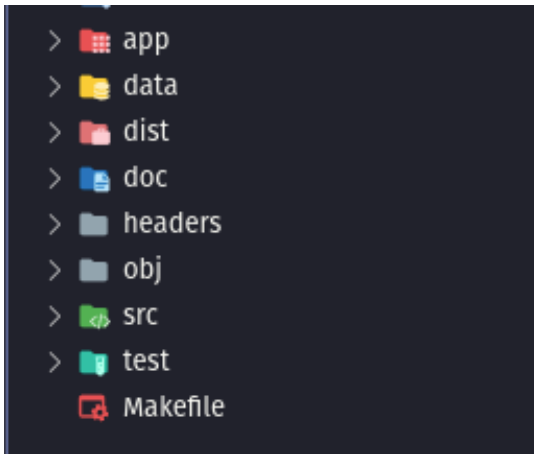

Figura 2: Arquitetura das pastas

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** e **obj** ficam localizados os cabeçalhos e os objetos gerados pelos arquivos .c dos *headers* respectivamente, 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 e, por fim, na pasta **app** fica localizado os programas principais do projeto.

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**. Caso seja executado o programa várias vezes, é recomendável que seja executado o comando **make clean**, para limpar as compilações, e depois executar **make** novamente.

2.4 Casos de teste e resultados

Para os casos de teste, foram utilizados quatro casos, contando com o caminho principal a ser resolvido. Vale ressaltar que temos dois casos com um resultado válido e dois casos em que não é possível encontrar uma solução ótima.

```
----------->2
Digite o nome do arquivo: field2.txt
./data/field2.txt
Arquivo carregado com sucesso!

Matriz: 5 x 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 1 2 3 4
1 1 3 5 4



Posicoes:
(1, 1)
(2, 1)
(3, 1)
(4, 1)
(5, 1)

Total de chamadas recursivas: 4
```

Figura 3: Resultado modo análise com caminho existente

```
 _____________________________________________________________________________________________________________
|                                                                                                             |
|                                                MENU                                                         |
|_____________________________________________________________________________________________________________|
|                                                                                                             |
|                                        Selecione o modo de execução:                                        |
|                              (1)Normal            (2)Análise            (3)Sair                             |
|_____________________________________________________________________________________________________________|
----------->2
Digite o nome do arquivo: field3.txt
./data/field3.txt
Arquivo carregado com sucesso!

Matriz: 5 x 5
1 2 3 4 5
1 2 3 4 5
1 2 3 4 5
1 1 2 3 4
2 1 3 5 4

IMPOSSÍVEL!
Total de chamadas recursivas: 20
```

Figura 4: Resultado modo análise sem caminho existente

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 ***backtracking*** para resolucionar um problema de caminho.

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 *backtracking* para resolucionar um problema de caminho.

References

[1] Backtracking. https://sites.google.com/site/tecprojalgoritmos/tecnicas-de-projetos/backtracking. (Accessed on 06/11/2022).