

UNIVERSIDADE FEDERAL DE VIÇOSA  
CAMPUS FLORESTAL

# Trabalho Prático I

## Projeto e Análise de Algoritmos

Luciano Belo - 3897  
Mariana Souza - 3898  
Guilherme Correa - 3509

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

Florestal  
Fevereiro de 2022

# CCF 330 - Projeto e Análise de Algoritmos

## TP 01 - Alienígena Giygas

Luciano Belo - 3897  
Mariana Souza - 3898  
Guilherme Correa - 3509

20 de Fevereiro de 2022

### Contents

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Desenvolvimento</b>	<b>2</b>
2.1	Arquivo de entrada . . . . .	3
2.2	TADs . . . . .	4
2.2.1	Map . . . . .	4
2.2.2	Enemies . . . . .	4
2.2.3	Moviments . . . . .	5
2.2.4	Ness . . . . .	5
2.3	Backtracking . . . . .	5
2.4	Modo Análise . . . . .	6
<b>3</b>	<b>Execução</b>	<b>7</b>
<b>4</b>	<b>Conslusão</b>	<b>9</b>
<b>5</b>	<b>Referências</b>	<b>10</b>

# 1 Introdução

O presente trabalho prático da disciplina de Projeto e Análise de Algoritmos tem como objetivo principal a derrota do alienígena Giygas pelo jovem Ness, dessa forma é traçado um caminho até o seu arqui-inimigo, assim contém um mapa que apresenta as possíveis possibilidades de caminhos até o Giygas. Onde os caminhos são marcados por "@", "-", "|" e "+", são os possíveis caminhos para o Ness, já se o caminho possuir "." ele não poderá passar pelo mesmo.

Além do Giygas (G) o Ness também possui outros inimigos, sendo eles Li'l UFO representado pela letra (U), Territorial Oak (T), Starman Junior (S), Master Belch (B), cada um contendo diferentes tipos de força. Pode - se encontra um exemplo de como é o mapa do caminho a ser percorrido por Ness na Figura 1 abaixo. Para a colaboração dos integrantes do grupo foi usado o GitHub para realizar o versionamento de código e a IDE Visual Studio Code para a implementação do código.

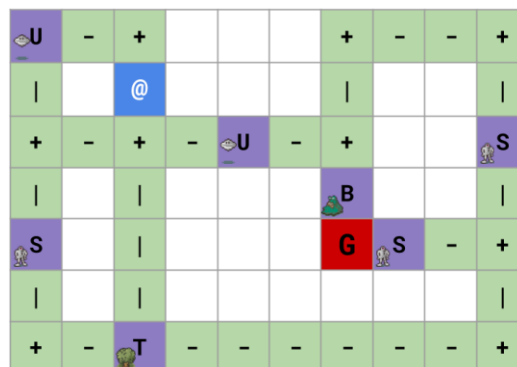


Figure 1: Mapa do caminho a ser percorrido por Ness.

## 2 Desenvolvimento

No desenvolvimento do trabalho a implementação calcula um percurso possível para Ness chegar até Giygas representado pela letra G e assim encerrar o programa, para o caminho até G ocorrer algumas restrições precisam ser atendidas como, Ness só poderá avançar no mapa se tiver os caracteres "-", "|" e "+", além disso, ele parte de uma localização "@" que é onde está o campus da UFV Florestal. Cada inimigo marcado com uma letra, tem um nível de força P', assim para derrotar o inimigo o Ness precisa ter pelo menos P' pontos de força, onde cada vez que ele vencer um abatalha a força aumenta em X' de acordo com o inimigo que for derrotado. Ele também conta com técnica PK Flash que derrota um inimigo na hora sem depender do seu nível de força.

O código foi desenvolvido na linguagem C e está dividido em pastas, onde "data" são encontrados os arquivos .txt para testes, "dist" o arquivo executável, "docs" a documentação, "headers" os arquivos .h, "src" é os arquivos .c, sendo que foram implementados quatro TADs, que serão elucidados posteriormente. Sendo assim, o trabalho segue a seguinte estrutura de pastas:

```

TP1-PAA
├── data
│   └── test.txt
├── dist
│   └── main
├── docs
├── headers
│   ├── enemies.h
│   ├── map.h
│   ├── menu.h
│   ├── moviments.h
│   └── ness.h
├── src
│   ├── enemies.c
│   ├── main.c
│   ├── map.c
│   ├── menu.c
│   ├── moviments.c
│   └── ness.c
└── Makefile

```

## 2.1 Arquivo de entrada

O arquivo de entrada tem um formato padronizado a primeira linha contém as variáveis P e K já nas próximas cinco linhas estão as variáveis P' e X' para cada um dos inimigos, na ordem U,T,S,B e G. Nas linhas seguintes são apresentadas as dimensões do mapa. Como apresentado na Figura 2 baixo. Dessa forma, foram criados quatro arquivos para teste.

```

1  20 2
2  10 5
3  25 10
4  30 15
5  40 20
6  65 0
7  5 7
8  +-T---+
9  |.|.B.|
10 S.|.GS+
11 |.|...|
12 +-U---+

```

Figure 2: Arquivo de entrada.

## 2.2 TADs

Utilizamos Tipos Abstratos de Dados para os arquivos de *map*, *ness*, *moviments* e *enemies*.

### 2.2.1 Map

Esse TAD contém a altura e a largura do mapa e um ponteiro para ponteiro que é o mapa. Na chamada para a criação do mapa verificamos todas as posições de forma se pode seguir para ela ou se é um inimigo comum ou gygas, além disso temos a função *isIntersection()* que verifica se o inimigo já foi derrotado ou não. Temos também uma função que verifica se o ponto que Ness está atualmente é ou não um cruzamento. Tem a função *move()* que realiza as movimentações no mapa, a *goMove()* que verifica se o movimento que está sendo feito é válido ou não. Já a função *battle()* realiza a luta de ness com os inimigos que não são o gygas. Por fim temos a função de printar o mapa e a que libera a mapa para a próxima execução do programa.

```
typedef struct Map
{
    int width, height;
    char **map;
} Map;
```

### 2.2.2 Enemies

No TAD com os atributos dos inimigos de ness e o id de cada um, além de uma lista contendo todos os inimigos. No arquivo *enemies.c* primeiramente criamos uma lista vazia de inimigos, depois começamos a criar os inimigos cada um com seus atributos de força e quantos pontos ness ganha ao derrotar o mesmo, logo em seguida inserimos os mesmos na lista. Temos uma função que remove o inimigo da lista e outra que remove todos, além de uma função de printar os inimigos e suas estatísticas. A função *enemieId()* atribui o id de cada inimigo, a função *findEnemie()* retorna os atributos do inimigo dado o id do mesmo, e a função *gygasPower()* retorna o poder de gygas.

```
typedef struct Atributes
{
    int p, k;
    char *id;
} Atributes;

typedef struct Enemie
{
    Atributes attributes;
} Enemie;

typedef struct Node *Pointer;

typedef struct Node
{
    Enemie enemie;
    Pointer next;
} Node;

typedef struct
```

```
{
    Pointer first, last;
} Enemies;
```

### 2.2.3 Moviments

TAD que possui as coordenadas do movimento, juntamente com sua força e quantidade de PKFlash, além da direção que ele está seguindo no mapa, também tem um caracter que informa se aquele ponto do mapa é um cruzamento. Além de uma lista de movimentos que está sendo feito e que serão feitos a seguir. A função `makeEmptyQueue()` faz uma lista vazia, a `toQueue()` inicializa a lista de movimentos, `dequeue()` desinfiltra a lista de movimentos, já a `printQueue()` printa a lista. Temos a função `newItem()` que adiciona um novo movimento na lista, a `isInQueue()` que armazena o que já foi inserido na lista, a `isInQueueWhateverDirection()` verifica se um inimigo já foi enfrentado, já a `backToLastNode()` retorna a uma posição anterior se não tiver como avançar da posição atual, por fim temos a função que libera a lista para próxima execução do programa.

```
typedef struct
{
    int x, y, nessP, nessK, direction;
    char intersection;
} item;

typedef struct TypeNode *TypePointer;
typedef struct TypeNode
{
    item queueItem;
    TypePointer next;
} TypeNode;

typedef struct
{
    TypePointer front, back;
    int size;
} Queue;
```

### 2.2.4 Ness

TAD com as estatísticas do movimento, força, quantidade de PKFlash e sua coordenada inicial linha e coluna. No arquivo `Ness.c` temos as funções que inicializa nosso personagem movimento, as coordenadas iniciais dele. Uma função que printa as estatísticas e outra que libera estas para a próxima execução do programa.

```
typedef struct Ness
{
    int p, k, line, column;
} Ness;
```

## 2.3 Backtracking

A backtracking no algoritmo do grupo encontra-se no arquivo `map.c`, sendo a função de nome `move`.

A função **move**, verifica primeiramente se Ness tem poder para vencer Giygas, caso tenha a função finaliza já que é o nosso objetivo principal. Em segunda instância, verifica se a posição atual é um cruzamento. Caso seja, é verificado qual movimentação será possível ( *UP*, *DOWN*, *LEFT* ou *RIGHT* ), sendo que para isso é necessário ponderar se a posição é válida, ou seja, respeita os limites do mapa e não é um ponto. Sendo uma posição válida a função é chamada de forma recursiva.

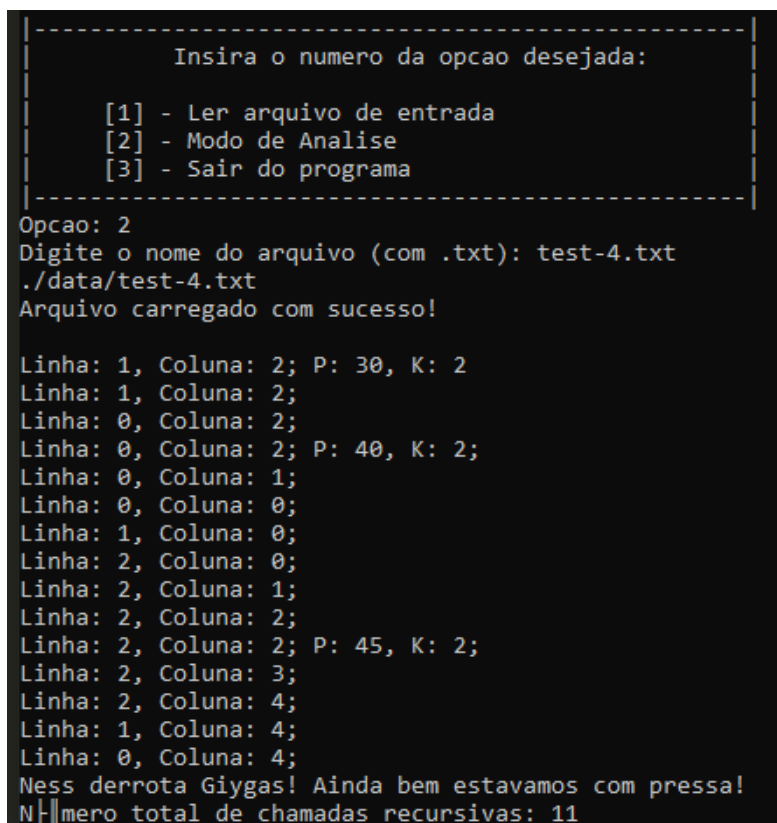
Ao "caminhar" pelo mapa, utilizamos também uma fila de movimentos que salva sempre um ponto de cruzamento, de tal forma que, caso não tenha mais um caminho, volte para o último cruzamento para que um novo caminho possa ser tentado. Caso mais nenhum destes movimentos seja possível, a função será encerrada.

Por fim, seguindo a ideia de direção, temos uma última verificação para que caso a direção atual seja possível ser seguida continuaremos com ela, até que não seja mais possível.

## 2.4 Modo Análise

Para o Modo de Análise foi criado um Tipo Abstrato de Dados Data, que está sendo apresentado no trecho de código abaixo, contendo o número de recursões, assim foi criado uma função para inicializar o mesmo, onde ele é incrementado a na função de movimentar contabilizando quantas chamadas recursivas foram realizadas. A Análise foi realizada em tempo de execução, onde foi acrescentado no modo uma opção 2 que possibilita relizar o modo. Na Figura abaixo é apresentado como é realizada a compilação do modo pela interface do algoritmo.

```
typedef struct Data
{
    int number_of_recursions;
} Data;
```



```
-----
                        Insira o numero da opcao desejada:
-----
[1] - Ler arquivo de entrada
[2] - Modo de Analise
[3] - Sair do programa
-----
Opcao: 2
Digite o nome do arquivo (com .txt): test-4.txt
./data/test-4.txt
Arquivo carregado com sucesso!

Linha: 1, Coluna: 2; P: 30, K: 2
Linha: 1, Coluna: 2;
Linha: 0, Coluna: 2;
Linha: 0, Coluna: 2; P: 40, K: 2;
Linha: 0, Coluna: 1;
Linha: 0, Coluna: 0;
Linha: 1, Coluna: 0;
Linha: 2, Coluna: 0;
Linha: 2, Coluna: 1;
Linha: 2, Coluna: 2;
Linha: 2, Coluna: 2; P: 45, K: 2;
Linha: 2, Coluna: 3;
Linha: 2, Coluna: 4;
Linha: 1, Coluna: 4;
Linha: 0, Coluna: 4;
Ness derrota Giygass! Ainda bem estavamos com pressa!
Número total de chamadas recursivas: 11
```

Figure 3: Execução Modo de Análise.

### 3 Execução

O trabalho possui um arquivo makefile contendo um conjunto de diretivas usadas para automação de compilação, execução e remoção de arquivos binários e serão apresentados em seguida.

```
all:
    gcc src/main.c src/menu.c src/moviments.c src/map.c src/enemies.c src/ness.c
    -o dist/main
run:
    dist/main
clean:
    rm dist/main
clean_exec:
    rm dist/main.exe
```

No menu apresentado na Figura 4 abaixo foram implementadas opções para ler um arquivo com as entradas padronizadas, o modo de análise apresentando o numero de recursão de um arquivo lido e a opção para encerrar e sair da execução do programa.

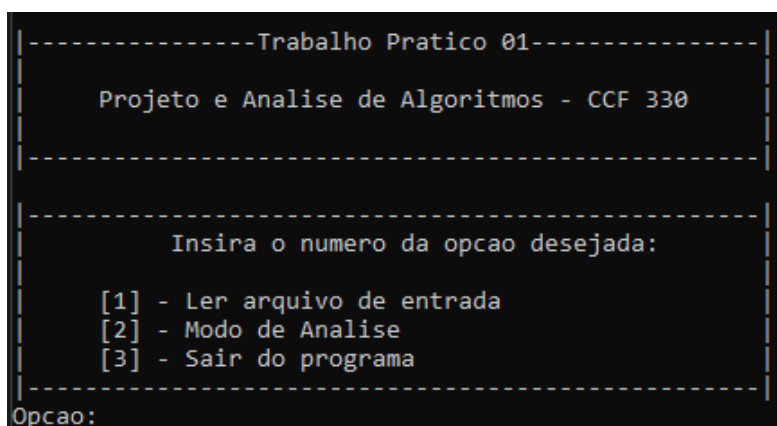


Figure 4: Menu com opções para execução.

Na Figura 5 é apresentado o resultado da saída após a leitura do arquivo de teste, contendo as restrições necessárias para realização do caminho por Ness até chegar em Giyga, apresentando os números das linhas e colunas para cada movimentação e também o P e K após a derrota dos inimigos, para exemplificar a execução temos a Figura 6, onde cada seta representa a movimentação realizada por Ness utilizando o algoritmo de Backtracking partindo de @ até chegar em G e finalizar.



```

dist/main
Digite o nome do arquivo (com .txt): test-4.txt
./data/test-4.txt
Linha: 1, Coluna: 2; P: 30, K: 2
Linha: 1, Coluna: 2;
Linha: 0, Coluna: 2;
Linha: 0, Coluna: 2; P: 40, K: 2;
Linha: 0, Coluna: 1;
Linha: 0, Coluna: 0;
Linha: 1, Coluna: 0;
Linha: 2, Coluna: 0;
Linha: 2, Coluna: 1;
Linha: 2, Coluna: 2;
Linha: 2, Coluna: 2; P: 45, K: 2;
Linha: 2, Coluna: 3;
Linha: 2, Coluna: 4;
Linha: 1, Coluna: 4;
Linha: 0, Coluna: 4;
Ness derrota Giygas! Ainda bem estávamos com pressa!

```

Figure 5: Saída para teste 4.

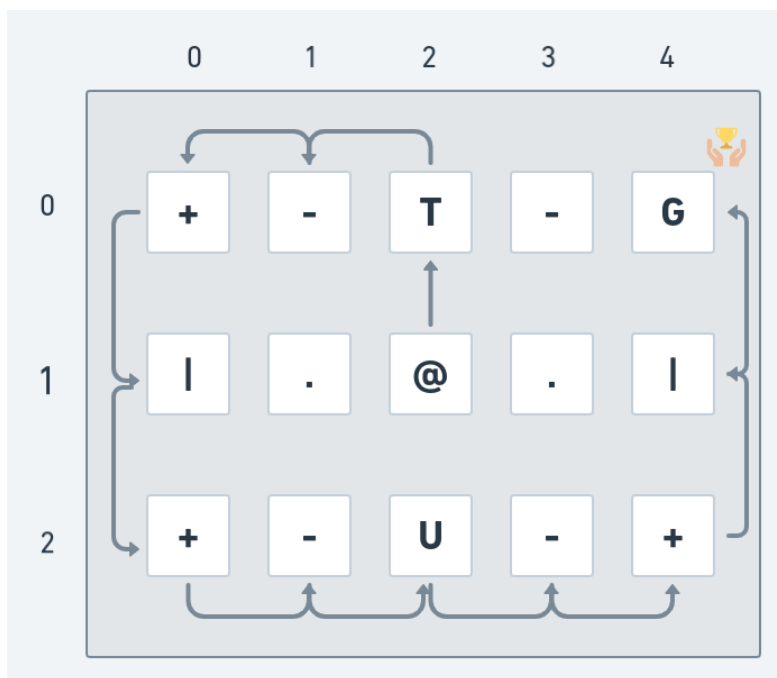


Figure 6: Visualização do resultado obtido para o caso de teste.

## 4 Conclusão

Desta forma, podemos concluir que o trabalho em questão foi desenvolvido conforme o esperado, atingindo todas as especificações requeridas na descrição do mesmo, já que o intuito principal do trabalho foi atingido, a criação e utilização de um algoritmo **backtracking**.

Tivemos algumas dificuldades relacionadas a implementação do algoritmo backtracking para o caso proposto, porém posteriormente conseguimos compreender melhor a lógica e funcionamento do algoritmo e assim construí-lo para sanar o problema.

Posto isso, é válido dizer que apesar das dificuldades na implementação do código, o grupo foi capaz de superar e corrigir quaisquer erros no desenvolvimento dos algoritmos. Haja vista que o trabalho foi executado conforme o planejado, sendo tratado tudo que foi pedido pelo professor. Por fim, verificou-se a assertiva para o objetivo do projeto em implementar um programa utilizando backtracking.

## 5 Referências

- [1] <http://www.bosontreinamentos.com.br/hardware/o-que-e-um-sistema-de-arquivos-file-system/>, Acesso em 19 de fevereiro de 2022;
- [2] <https://www.ic.unicamp.br/~zanoni/teaching/mc102/2013-1s/aulas/aula22.pdf>, Acesso em 19 de fevereiro de 2022
- [3] <https://github.com/fernandafs/8-rainhas/blob/main/rainhas.c>, Acesso em 19 de fevereiro de 2022
- [4] <https://github.com/isabellazramos/Projeto-e-Analise-de-Algoritmos>, Acesso em 19 de fevereiro de 2022
- [5] <https://pt.stackoverflow.com/questions/103184/o-que-%C3%A9-um-algoritmo-backtracking>, Acesso em 19 de fevereiro de 2022