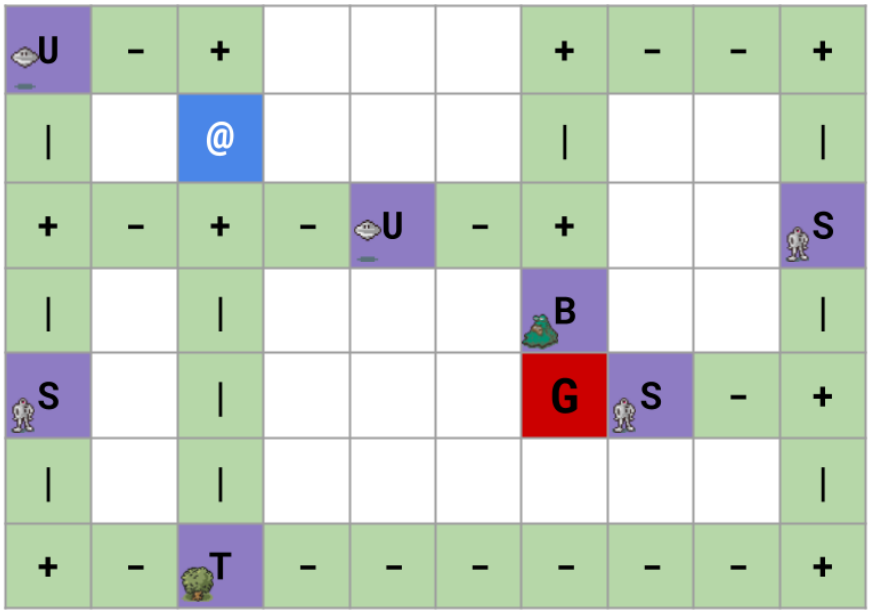
Universidade Federal de Viçosa – Campus UFV-Florestal
Ciência da Computação – Projeto e Análise de Algoritmos
Professor: Daniel Mendes Barbosa

Trabalho Prático 1

Este trabalho é **obrigatoriamente em grupo**. Os grupos já foram definidos nesta planilha e este trabalho deverá ser entregue no PVANet Moodle de acordo com as instruções presentes no final da especificação.

O terrível alienígena Giygas ataca a Terra novamente, e apenas o jovem Ness e seus amigos podem derrotá-lo. No entanto, seus amigos foram raptados pelos lacaios de Giygas, e agora Ness buscou a ajuda de brilhantes estudantes de Ciência da Computação em Florestal para traçar um caminho vitorioso até seu arqui-inimigo. Sua amiga Paula tem habilidades telepáticas impressionantes e descobriu a localização de cada inimigo e seus poderes, e contatou Ness. Assim, ele trouxe para vocês um mapa planificado dos possíveis caminhos que ele pode tomar, marcado com as informações que recebeu de Paula. O **objetivo** de Ness é chegar até Giygas e derrotá-lo, não importa quantos caminhos ele tenha que passar, no entanto, ele - e o mundo todo! - tem pressa, e não quer passar por um mesmo caminho mais de uma vez. Observe um exemplo do mapa:



Sabemos que Ness inicia sua jornada com um certo nível de força *P*, a partir da localização do campus da UFV Florestal, marcada com um `@`. Os caminhos que ele pode tomar são indicados com os caracteres `-`, `|` e `+`, sendo um caminho horizontal, vertical e um cruzamento, respectivamente. A posição de cada inimigo é marcada com uma letra maiúscula `U`, `T`, `S`, `B` ou `G`, sendo `G` a posição do arqui-inimigo Giygas.

Cada inimigo tem também um nível de força P'. Para derrotar um inimigo, Ness precisa ter ao menos P' pontos de força ($P \geq P'$), e a cada vez que vence uma batalha, seu nível de força aumenta em X', de acordo com o inimigo derrotado. Além disso, Ness é um garoto proficiente em habilidade psíquicas e sabe a técnica especial PK Flash Ω, que derrota um inimigo instantaneamente independente de seu nível de força (exceto Giygas), porém essa técnica só pode ser usada K vezes, e Ness recorre a ela apenas se sua força não for capaz de derrotar o inimigo.

Cada caminho só pode ser percorrido uma única vez. Os cruzamentos, no entanto, podem ser visitados mais de uma vez, desde que o caminho tomado em seguida seja diferente. Os espaços onde se encontram inimigos ou o espaço inicial de Ness podem ser considerados cruzamentos.

Você deve portanto projetar um algoritmo com **backtracking** para encontrar um caminho possível neste contexto, implementá-lo em C e documentá-lo, de acordo com as especificações dadas.

Entrada

O mapa do mundo, junto com o valor das variáveis P, K, P' e X', serão a entrada para seu programa a partir de um arquivo texto, obtido a partir da comunicação telepática de Paula. O arquivo terá um formato padronizado, sendo que na primeira linha estão as variáveis P e K, nas próximas cinco linhas estão as variáveis P' e X' para cada um dos inimigos, na ordem `U`, `T`, `S`, `B` e `G`. Em seguida, são informadas as dimensões do mapa, em altura e largura, respectivamente. Por fim, o mapa em si é dado, de acordo com a codificação em caracteres previamente apresentada. Os espaços onde Ness não pode passar são demarcados com um caractere `.`. Segue um exemplo de entrada, com um mapa correspondente à figura inicial:

```
20 2
10 5
25 10
30 15
40 20
80 0
7 10
U-+...+--+
|.@...|..|
+-+-U-+..S
|.|...B..|
S.|...GS-+
|.|......|
+-T------+
```

U - Li'l UFO

T - Territorial Oak

S - Starman Junior

B - Master Belch

Os tipos de monstros enviados por Giygas.

Nesse exemplo, nosso herói começa com 20 de força e 2 usos do PK Flash. Quanto a seus inimigos, Paula descobriu que o Li'l UFO possui 10 de força e fornece 5 pontos de recompensa, o Territorial Oak possui 25 e fornece 10, o Starman Junior possui 30 e fornece 15, o Master Belch possui 40 e fornece 20, e por fim Giygas possui 80 pontos de força. Não importa quantos pontos Giygas forneceria, pois ao derrotá-lo a missão está cumprida, mas a entrada terá um valor nessa posição apenas por fins de padronização.

Saída

O programa deverá calcular um percurso possível e imprimir a resposta final na tela. Cada posição ocupada por Ness nesse percurso deverá ser impressa em uma linha da saída. Além disso, se houver alguma batalha numa posição, os novos valores de *P* e *K* após a batalha devem ser impressos. Imaginando que um percurso possível envolva Ness começando descendo completamente a coluna onde inicia a jornada, e batalhando contra o Territorial Oak usando PK Flash, as primeiras linhas da saída seriam:

```
Linha: 1, Coluna: 2; P: 20, K: 2
Linha: 2, Coluna: 2;
Linha: 3, Coluna: 2;
Linha: 4, Coluna: 2;
Linha: 5, Coluna: 2;
Linha: 6, Coluna: 2; P: 30, K: 1
```

Pode haver casos em que não existe um percurso em que Ness consiga derrotar Giygas, e o programa deve relatar esse resultado:

```
Apesar de todas as tentativas, Ness falha em derrotar Giygas!
```

Obs.: o grupo é livre para definir outras formas mais interessantes de exibir essa saída, desde que essas informações mínimas estejam presentes.

Backtracking

Seu programa deverá obrigatoriamente usar **backtracking**. Uma função recursiva chamada `movimentar` (ou de nome similar) deverá ser criada. Isso significa que primeiramente você deverá partir da posição inicial, marcada com `@` e chamar essa função

uma única vez, e a partir daí ela chamará ela mesma, até que Ness encontre e derrote Giygas, ou descubra que não há solução para o problema.

Na documentação, o grupo deverá explicar seu algoritmo com base nos conceitos de backtracking, e como ele foi implementado, explicando as estruturas de dados necessárias. É importante ressaltar que só se sabe o tamanho das estruturas depois de ler os arquivos de entrada, portanto deverá ser usada a **alocação dinâmica de memória**.

O grupo deve definir a possibilidade do programa ser executado em um "modo de análise", para verificar o funcionamento do algoritmo. Nesse modo, seu programa contabiliza quantas chamadas recursivas foram feitas, e qual foi o nível máximo de recursão alcançado. O modo de análise pode ser feito em tempo de compilação (por exemplo, através do uso de `#define`), ou em tempo de execução (alguma opção dentro da interface do programa que permita ligar e desligar esse modo). O importante é que o grupo escolha e documente como isso foi feito.

Quando o modo de análise estiver ligado, essas informações sobre a recursão devem constar na saída final do programa, independente se houve uma solução ou não.

Interface

O grupo é livre para definir os detalhes da interface do programa, desde que ela atenda aos seguintes requisitos:

- O programa deve aceitar arquivos de texto como entrada, segundo a formatação especificada.
- O programa deve ser capaz de aceitar novos arquivos de entrada depois de cada execução, encerrando apenas quando o usuário desejar.

Tarefas extras

1. A interface e como exibir o resultado fica a critério do grupo. Portanto, poderá ser oferecida mais de uma forma de se exibir os resultados, podendo o usuário escolher qual formato deseja.
2. Criar uma opção (ou até mesmo um outro programa) para a geração de arquivos de teste, considerando todos os dados envolvidos e o formato, como descrito acima. Os mapas não necessariamente possuem solução. Seu programa de geração de arquivos de teste deverá ter alguns parâmetros de configuração, como largura e altura do mapa, quantidade de inimigos, limites das variáveis, e "dificuldade" do mapa, entre outros, que vocês poderão colocar e especificar na documentação.

Faça exatamente o que está sendo pedido neste trabalho, ou seja, mesmo que você tenha uma ideia mais interessante para o programa, você deverá implementar exatamente o que está definido aqui no que diz respeito ao problema em si e ao paradigma backtracking. No entanto, você pode implementar algo além disso, desde que não atrapalhe a obtenção dos resultados necessários a esta especificação.

Formato e data de entrega

Os arquivos com o código-fonte (projeto inteiro do Codeblocks ou arquivos .c, .h e makefile), juntamente com um arquivo PDF (**testado, para ver se não está corrompido**) contendo a **documentação**. A documentação deverá conter:

- explicação do algoritmo projetado;
- implementação do algoritmo projetado (estruturas de dados criadas, etc);
- resultados de execução, mostrando entrada e saída;
- arquivos de entrada usados nos testes.
- explicação de como compilar o programa em modo normal e modo análise.
- nos resultados deverá constar a quantidade total de chamadas recursivas e o nível máximo de recursão obtido para cada teste através da execução no modo análise.

Mais direcionamentos sobre o formato da documentação podem ser vistos no documento "Diretrizes para relatórios de documentação".

Importante: Entregar no formato **ZIP**. As datas de entrega estarão configuradas no PVANet Moodle. É necessário que apenas um aluno do grupo faça a entrega, mas o PDF da documentação deve conter os nomes e números de matrícula de todos os alunos em sua capa ou cabeçalho.

Bom trabalho!