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

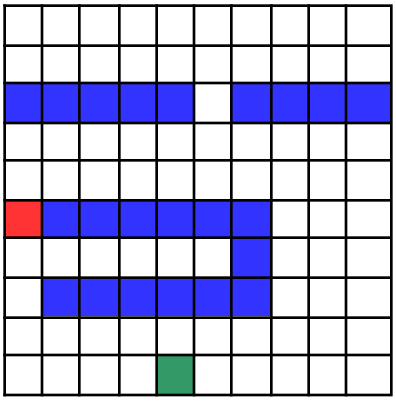
Trabalho Prático 1

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.

Vários estudantes de ciência da computação foram presos em um país cujo governo os considerava uma ameaça à sua manutenção no poder! O governo decidiu ainda fazer uns experimentos com objetivos desconhecidos, onde cada um dos prisioneiros era então retirado de sua cela e colocado em um labirinto, com uma ou mais chaves. O prisioneiro deve então tentar escapar deste labirinto. Obviamente ele não consegue atravessar as paredes, a não ser que seja uma porta e que ele tenha chaves sobrando em seu bolso. **O objetivo** é escapar do labirinto, mas não se sabe se importa o número de tentativas erradas e nem o número de chaves usadas. Se você voltar atrás em uma porta que você usou uma chave, pode pegar de volta a chave, já que o caminho não levou à saída do labirinto, e então poderá usá-la em outra porta de outro caminho possível do labirinto. Por sorte, o estudante de ciência da computação, prevenido, estava com seu notebook e com um drone. Com isso usou o drone para fotografar o labirinto por cima, e então bastava escrever um algoritmo para encontrar a saída mais rapidamente, tendo como entrada a foto tirada convertida para um arquivo texto.

Você deve portanto imaginar que você é o tal estudante e projetar um algoritmo com **backtracking** para encontrar uma saída do labirinto neste contexto, implementá-lo em C e documentá-lo, de acordo com as especificações abaixo.

Observe a figura:



Esta tabela representa o espaço geográfico do labirinto, onde cada célula representa um quadrado de 1 metro quadrado. Temos neste espaço o estudante de ciência da computação, cuja posição inicial está representada por um quadrado verde (ou seja, o estudante está inicialmente nesta posição). O estudante deverá caminhar pelos vários quadrados (ou seja, várias posições) até chegar em uma das células da primeira linha da tabela (linha de cima).
Os quadrados azuis representam paredes, pelas quais o estudante obviamente não pode passar. O estudante só pode movimentar para cima, para baixo e para os lados. Ou seja, **não pode se movimentar nas diagonais.**

Você deverá escrever um programa na linguagem C que utilize um algoritmo projetado por você, que leia um arquivo com as informações do labirinto onde o estudante está, bem como sua exata posição inicial. Seu programa deverá então encontrar um caminho que leve o estudante de sua posição inicial até uma das células da primeira linha da tabela, mostrando na tela cada tentativa de movimento feita e qual o caminho encontrado.

Seu programa deverá obrigatoriamente usar **backtracking**. Uma função recursiva chamada **movimenta_estudante** deverá ser criada. Isso significa que primeiramente você deverá encontrar a posição inicial do estudante. Quando encontrar, deverá chamar esta função uma única vez, e a partir daí ela chamará ela mesma, até que o estudante chegue na primeira linha (linha zero).

Importante:
- você deverá definir as estruturas de dados necessárias ao algoritmo;
- na **documentação** você deverá explicar seu algoritmo com base nos conceitos de backtracking, e como ele foi implementado;

Formato de entrada de dados:

O espaço geográfico do labirinto que será a entrada para seu programa, será fornecido a partir de um arquivo texto, previamente obtido a partir da foto tirada pelo drone. O arquivo terá um formato padronizado, sendo que na primeira linha do arquivo temos o número de linhas, um espaço, um número de colunas, um espaço e o número de chaves que o estudante já tem no seu bolso.
Nas linhas seguintes deverão ser informadas as cores de cada uma das células, de cada linha, sendo que as células de cada linha **não** são separadas por espaço. Cada cor será representada por um número de 0 a 3, sendo:

0 - **verde** (célula onde o estudante está inicialmente)
1 - branco (célula vazia, por onde o estudante pode passar)
2 - **azul** (célula ocupada por parede)
3 - **vermelho** (célula acessível apenas por uma chave)

Veja o exemplo abaixo, que seria o conteúdo do arquivo de entrada referente ao primeiro labirinto mostrado anteriormente:

```
10 10 1
1111111111
1111111111
2222212222
1111111111
1111111111
3222222111
1111112111
1222222111
1111111111
1111011111
```

Este exemplo possui 10 linhas com 10 colunas cada e considera que o estudante está com uma chave no bolso. Mas estes valores podem ser valores quaisquer, e o espaço geográfico do labirinto pode ser retangular (número de linhas diferente do número de colunas).

Formato de saída de dados:

O programa deverá imprimir a resposta na tela. **Cada posição** ocupada pelo estudante deverá ser impressa em uma linha da saída. Imaginando que o estudante tenha começado na linha 9 e coluna 4, e depois tenha se movimentado para a linha 8 e coluna 4, as duas primeiras linhas do resultado seriam, portanto:

```
Linha: 9 Coluna: 4
Linha: 8 Coluna: 4
```

Ao final da execução e da impressão de todas as células pelas quais o estudante passou, deverá ser impressa a quantidade total de movimentos feitos e em qual coluna da primeira linha o estudante chegou. Exemplo:

```
O estudante se movimentou 25 vezes e chegou na coluna 7 da primeira linha
```

Observe que muitas vezes o estudante pode seguir por um caminho sem saída, e aí deverá voltar e achar outro caminho. Todas essas movimentações deverão ser contabilizadas no total de movimentações.

Importante: além disso, o labirinto pode não ter saída! Nesse caso o estudante para na célula em que estiver quando perceber que não há saída e será impresso pelo programa (por exemplo):

```
O estudante se movimentou 25 vezes e percebeu que o labirinto nao tem saida.
```

4) Aparência geral do programa:

Portanto, o programa deverá mostrar as seguintes opções para o usuário:

```
PROGRAMA Labirinto: Opcoes do programa:
1) Carregar novo arquivo de dados.
2) Processar e exibir resposta.
3 ou qualquer outro caracter) Sair do programa.
Digite um numero:
```

Obs.: Todos esses exemplos de interface são sugestões. Desde que as funcionalidades estejam presentes, o grupo pode personalizar a interface, justificando as alterações, mas de forma que melhore o trabalho. Isso pode ser levado em consideração na correção.

Veja abaixo exemplos de execução de cada opção (cada opção limpa a tela inicialmente):

Escolhendo a opção 1:

```
Por favor digite o nome do arquivo:
```

Escolhendo a opção 2, o arquivo deverá ser processado e a saída exibida conforme explicado na seção anterior.

Caso a opção 2 seja escolhida antes de carregar qualquer arquivo, deverá ser impresso:

```
Por favor, carregue antes um arquivo de dados!
Pressione qualquer tecla para continuar...
```

Escolhendo a opção 3, o programa é encerrado.

Lembrando que enquanto o programa não for encerrado, o usuário poderá carregar novos arquivos e processá-los.

Você não poderá escrever nos arquivos de entrada e nem em outros arquivos, ou seja, só poderá abri-los no modo "r" (somente leitura). Para armazenar os dados do arquivo no programa, para poderem ser processados, você deverá **alocar memória dinamicamente**, visto que não se sabe a princípio o tamanho do arquivo (sabe-se apenas quando for lida sua primeira linha).

Considere que os arquivos de entrada seguirão fielmente o formato que foi definido.

Além disso, seu programa deverá ter um **#define** para configurar se o programa estará no **modo análise** ou não. Se não estiver no modo análise, a execução será como descrito anteriormente. Se estiver no modo análise, deverá fazer tudo, mas também contabilizar o número total de chamadas recursivas que foram feitas e o nível máximo de recursividade alcançado durante toda a solução. O programa irá contabilizar isso e imprimir na tela somente se o modo análise estiver ligado. Deverá ser estudada melhor forma de se fazer

isso com #define e explicada no relatório como utilizar (compilar o programa em modo análise ou não).

Tarefas que podem ser feitas além do básico (que podem servir inclusive como diferencial no ranqueamento das notas dos trabalhos):

1) A interface e como exibir o resultado também ficará a critério do grupo. Poderá ser oferecida mais de uma forma de se exibir os resultados, podendo o usuário escolher qual formato deseja.

2) Vocês podem ainda criar uma opção (ou até mesmo um outro programa) para a geração de labirintos de teste, considerando todos os dados envolvidos e o formato, como descrito acima. Labirintos não necessariamente possuem saída. Seu programa de geração de arquivos de teste deverá ter alguns parâmetros de configuração, como largura e altura do labirinto, quantidade de portas, quantidade de chaves iniciais, e "dificuldade" do labirinto, entre outros, que vocês poderão colocar.

3) Permitir que no labirinto possam existir chaves no chão. **Seriam células amarelas, com o número 4**. No entanto, o estudante só poderia usar esta chave em caminhos que passam por esta célula. Ou seja, no caso de voltar por ela para tentar um novo caminho depois, teria que deixar a chave no chão novamente.

4) Criar uma outra opção inventada pelo grupo, com a devida especificação, implementação e resultados mostrados na documentação (também resolvendo com **backtracking**).

5) Plotar um gráfico de **complexidade x tempo**. Basicamente, vocês devem criar a curva de complexidade do algoritmo criado, testando o mesmo para diferentes tamanhos de labirinto. **Considerem o eixo x como o tamanho da entrada e o eixo y como o tempo total gasto para cada entrada.** Vocês podem usar qualquer software para a plotagem, mas devem colocá-lo nas referências do relatório.

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 de um IDE 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", também disponível no PVANet Moodle.

Importante: no PVANet Moodle você fará a entrega de dois arquivos: um para a documentação e um para o zip com o projeto todo. 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 em sua capa ou cabeçalho os nomes e números de matrícula de todos os alunos do grupo que efetivamente participaram do trabalho. **Assim como no trabalho prático 0, vocês devem utilizar a linguagem de programação C.**

Bom trabalho!