Algoritmos e Estruturas de Dados BCC 2018/1

Exercício Programa 3: Joãozinho e Maria

Conteúdo: Backtracking; algoritmo KMP

Rodrigo de Souza

Entrega: 15/07/2018

1 Enunciado

Nosso objetivo neste Exercício-Programa é implementar um método para ajudar Joãozinho e Maria a sair de um labirinto.

A entrada do problema é uma matriz quadrada $n \times n$ representando o labirinto; um caminho nesse labirinto é uma sequência de posições duas a duas contíguas da matriz. *Contígua* aqui significa: *à direita*, *à esquerda*, *acima* ou *abaixo* (não é possível andar para uma nova posição na diagonal). Além da matriz, faz parte da entrada dois pares de inteiros representando posições: o primeiro par (x, y) indica a posição inicial de Joãozinho e Maria (linha x e coluna y) e o segundo par, (x', y'), indica a posição da única porta de saída.[1]

Cada posição da matriz pode ser uma parede, o que se representa pelo símbolo 0, ou, caso seja uma casa livre, é pintada com uma de quatro cores possíveis: *amarelo*, *vermelho*, *roxo* e *laranja*, o que se representa por letras do alfabeto $\{a, v, r, l\}$. Uma parede é uma posição não pode ser visitada por Joãozinho e Maria, ou seja, não pode fazer parte de um caminho que leva à saída. As demais posições coloridas são livres e podem ser usadas.

Para piorar a situação de Joãozinho e Maria, determinados caminhos, mesmo que levem à porta de saída, fazem aparecer um dragão mortífero logo após a porta. Os caminhos que evitam o aparecimento do dragão são aqueles que contém, na sequência de cores que contitui o caminho, pelo menos uma ocorrência de uma cadeia s sobre o alfabeto $\{a, v, r, l\}$. Assim, a entrada do problema inclui também uma cadeia s sobre o alfabeto $\{a, v, r, l\}$, representando a sequência de cores a ser respeitada para evitar o dragão.[2] Veja na próxima seção uma descrição detalhada e um exemplo de arquivo de entrada.

Formalmente, um caminho válido é uma sequência de posições

$$(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$$

[1] Você pode escolher se os índices começam em 0 ou 1, mas isso deve ficar bem claro no seu programa.

[2] Você pode supor, por simplicidade, que o padrão tem comprimento no máximo P, onde P é uma constante que você declara no início do seu programa; pode colocar P como um número pequeno, 7 está bom.

sem passar por nenhuma parede, onde:

- (x_1, y_1) é a posição inicial lida do arquivo de entrada;
- (x_k, y_k) é a posição de saída lida do arquivo de entrada;
- conforme ja dissemos, duas posições contíguas (x_j, y_j), (x_{j+1}, y_{j+1}) precisam ser adjacentes na matriz: (x_{j+1}, y_{j+1}) deve estar à direita, à esquerda, acima ou abaixo de (x_j, y_j) – não é permitido andar na diagonal.

O objetivo do seu programa é encontrar um caminho válido, e depois dizer se ele termina no dragão ou não.

Sua solução consiste, de forma intuitiva, em sair andando no labirinto, a partir da posição de início, construindo o caminho incrementalmente, selecionando a cada passo uma nova posição viável – livre de parede, e *ainda não visitada*. *Selecionar* significa marcar essa posição, para que você saiba que já passou por ela (como Joãozinho e Maria fizeram jogando migalhas de pão no caminho). Possivelmente, nesse processo eles vão chegar em um beco sem saída, ou seja, onde todas as posições adjacentes são ou paredes ou posições já visitadas. Neste caso, sua solução executa o que se chama de *backtracking*: ela volta um passo no caminho que construiu até agora, e tenta outra direção, a partir do passo anterior; caso não haja novamente outra direção, faz um novo *backtrack*, etc.

Para viabilizar o *backtracking*, Joãozinho e Maria devem conhecer a posição anterior que tinham escolhido no caminho que estão tentando construir. Você pode fazer isso armazenando a última posição visitada, na casa que está visitando atualmente. Também deve armazenar, em cada casa, a última direção tentada como próximo passo, a partir daquela casa. Para ser feito com critério, vamos supor que seu algoritmo considera novas tentativas de direções sempre no sentido horário: *acima*, *direita*, *abaixo*, *esquerda*. Seu algoritmo procura assim a próxima posição viável (não foi ainda visitada, nem é uma parede) no sentido horário; caso encontre, anda nessa direção. Ao se voltar para uma casa, em um *backtracking*, lê, nessa casa, a última direção tentada, e tenta a próxima; caso não haja mais direção possível, precisa executar um novo *backtrack*.

Uma vez encontrado um caminho de saída, seu programa deve imprimir a sequência de posições que o compõe (você pode fazer isso seguindo os *links* construídos durante a execução do algoritmo – mas tome cuidado para não imprimir de trás para frente), ou então imprimir uma mensagem dizendo que não há caminho de saída. Finalmente, seu algoritmo testa se a sequência de cores desse caminho evita o dragão, ou seja, se tem uma ocorrência do padrão lido do arquivo de entrada. Para isso, seu algoritmo deve recuperar a senha de cores do caminho encontrado, e executar o algoritmo KMP de busca de padrões.

2 Exemplo

Seu programa deve ler o labirinto de um arquivo `labirinto.dat` cujo formato descrevo a seguir.

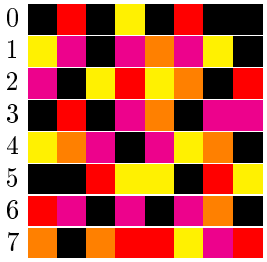
A primeira linha do arquivo de entrada é a dimensão n do labirinto (que é uma matriz quadrada). A segunda linha tem dois inteiros, indicando a posição de início, e a terceira, dois inteiros, indicando a posição de saída (no exemplo abaixo, estou assumindo que a numeração começa em 0, de cima para baixo, e da esquerda para a direita). Esses inteiros são separados por um espaço

em granco. Em seguida, vem n linhas representando o labirinto: cada uma tem uma sequência de n símbolos sobre o alfabeto $\{0, a, v, r, l\}$, separados por espaço, indicando o conteúdo de cada posição (uma parede, ou uma cor). A última linha é o padrão s de cores (uma cadeia sobre o alfabeto $\{a, v, r, l\}$).

Segue um exemplo de arquivo de entrada:

```
8
1 3
7 5
0 v 0 a 0 v 0 0
a r 0 r l r a 0
r 0 a v a l 0 v
0 v 0 r l 0 r r
a l l 0 r a l 0
0 0 v a a 0 v a
v r 0 r 0 r l 0
l 0 l v v a r v
lral
```

Eis uma representação pictórica desse labirinto:



Execute mentalmente ou desenhe sobre esse labirinto algumas iterações do algoritmo.

3 Instruções gerais

- Seu programa deve ser feito em C.
- Não tente embelezar a saída do seu programa com mensagens, formatações, etc. Seu programa será julgado segundo a adequação de sua saída à descrição do exercício.
- Seu programa deve consistir de um único arquivo, e deve ser razoavelmente modular, ou seja: deve consistir de diversas funções que, juntas, realizam a tarefa descrita.
- Documente cada função dizendo o quê ela faz.
- Escreva no início do código um cabeçalho com comentários, indicando nome, número do EP, data, nome da disciplina.
- A entrega será eletrônica no ambiente Moodle da disciplina (não receberei exercícios impressos ou via email).

4 Correção

- Trabalhos copiados (relatório ou programa) são anulados.
- Um programa que não compila vale no máximo 3. Neste caso, leio o código para tentar julgar as tentativas do aluno.
- Um programa mal escrito, desorganizado, recebe 2 pontos de desconto, *mesmo que funcione*. Ler o código leva muito tempo, mas tento fazer o melhor, lendo tudo o que vocês me entregam. Se o programa está organizado, documentado, a leitura é rápida. Se está confuso, às vezes é muito difícil compilar na cabeça o que o aluno faz, e isso leva muito tempo, então, *não vou compilar na cabeça códigos difíceis de ler*. É por isso que esses códigos recebem desconto. Daí a importância de caprichar no código também.

5 Plágio

O problema do plágio de código é endêmico em disciplinas que envolvem programação. Essa prática será fortemente combatida. Você está se enganando se copiar o programa de seu colega. O maior prejudicado é você mesmo.

Neste semestre, vou usar uma ferramenta automática para detecção de plágios em código. Essa ferramenta faz uma análise de um conjunto de arquivos fonte (os EP's) e emite um relatório, indicando a semelhança entre pares de arquivos. Em particular, essa ferramenta (que já tem anos de estrada, e uma *expertise* subjacente) detecta tentativas de burlar o plágio como mudança de nomes de variáveis e alterações na indentação. Ou seja, é muito difícil copiar e não ser descoberto. E infelizmente preciso ser bastante rigoroso caso a ferramenta detecte cópias (veja critérios de correção acima).

Vocês podem e devem pensar em soluções em conjunto. Mas uma vez encontrada a solução, cada um faz seu programa. O máximo que podem fazer é, eventualmente, trocar pequenos trechos sem importância para a solução, por exemplo, código para entrada e saída. E caso o façam, devem indicar com comentários ("esse trecho foi desenvolvido em conjunto com fulano de tal", etc.).

Ao copiar, você também prejudica seu colega que fez o código, porque ambos os trabalhos serão anulados. Melhor evitar essa prática. Se não fez o exercício, melhor não entregar e se concentrar no próximo.

6 Bônus

Resolver esse EP vale de 0 a 10. Vou conceder um bônus de 2 pontos caso, além de resolver o problema original, você implemente uma modificação que sempre encontre *um caminho livre de dragão, se o mesmo existir* (e mostra uma mensagem caso não exista um tal caminho). Você deve explicar cuidadosamente sua solução no seu código.