



PUC
CAMPINAS
PONTIFÍCIA UNIVERSIDADE CATÓLICA

Centro de Ciências Exatas, Ambientais e de Tecnologias
Faculdade de Análise de Sistemas
Curso Sistemas de Informação
Projeto Integrado C
1º Trabalho do 1º Semestre de 2019

O propósito deste trabalho é, usando pilhas, construir em Java um programa apropriadamente orientado a objetos que encontre o caminho até a saída de um labirinto. Para tanto, considere que existam arquivos texto que descrevam labirintos, como, por exemplo, um arquivo chamado **teste1.txt** com o seguinte conteúdo:

```
5
#####
E      #
###   ###
###   ###
###S###
```

Na primeira linha dele, **5** indica que o labirinto será descrito por **5** linhas texto. Veja que as as linhas de texto que descrevem o labirinto propriamente dito.

O número de caracteres de cada linha deverá ser identificado por você quando feita a leitura da primeira linha, lembrando que todas as linhas deverão conter a mesma quantidade de caracteres para a formação do labirinto. Caso haja diferença na quantidade de caracteres de cada linha, você não deverá prosseguir com a verificação do caminho e deverá notificar o usuário que há inconsistência no arquivo.

Nas linhas de texto, o caractere **#** representa parede, o **espaço em branco** representa passagem, o caractere **E** representa a entrada do labirinto e o caractere **S** representa a saída do labirinto.

Para resolver o problema, seu programa deve realizar uma sequência de passos, a saber:

1. Solicitar a digitação do nome do arquivo texto que contém a estrutura do labirinto; suponha que tenha sido digitado **teste1.txt**;
2. Carregar os caracteres do arquivo texto em uma matriz chamada **labirinto**. no caso deste exemplo, esta matriz terá **5** linhas e **8** colunas. A quantidade de linhas você deverá ler do arquivo. A quantidade de colunas você irá descobrir quando ler a segunda linha do arquivo, e verificar a quantidade de caracteres desta linha. Esta quantidade de caracteres da segunda linha deverá ser assumida como a quantidade de colunas a matriz. Desta forma, garanta que ao ler as demais linhas do arquivo, estas estarão consistentes com o tamanho da segunda linha, e caso não esteja, deverá notificar o usuário que o arquivo está inconsistente. Caso seja possível carregar os dados do arquivo, o **labirinto** fica assim:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E							#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

3. Instanciar um objeto do tipo **Pilha<Coordenada>** chamada **caminho** com capacidade para armazenar **5x8**, ou seja 40, coordenadas;
4. Instanciar um objeto do tipo **Pilha<Pilha<Coordenada>>** chamado **possibilidades** com capacidade para armazenar **5x8**, ou seja 40, pilha de coordenada;
5. Procurar a entrada do labirinto (o caractere E) nas bordas do **labirinto**, acusando erro, caso não encontre ou instanciando um objeto chamado **atual** da classe Coordenada para representar a posição onde o caractere E foi encontrado; no caso, (1,0).
6. Instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a pilha gerada conteria:

(1,1)

7. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (1,1) e **pilha de adjacentes** passa a conter:

8. Colocar no **labirinto** na posição indicada por **atual** um caractere *****, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*						#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

9. Empilha atual em caminho; assim, caminho fica:

(1,1)

10. Empilha a pilha de adjacentes em possibilidades; assim, possibilidades fica:

11. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

(1,2)

12. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (1,2) e **pilha de adjacentes** passa a conter:

13. Colocar no **labirinto** na posição indicada por **atual** um caractere *, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*					#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

14. Empilha **atual** em **caminho**; assim, **caminho** fica:

$(1,2)$
$(1,1)$

15. Empilha a **pilha de adjacentes** em **possibilidades**; assim, **possibilidades** fica:

16. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

(1,3)

17. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (1,3) e **pilha de adjacentes** passa a conter:

18. Colocar no **labirinto** na posição indicada por **atual** um caractere *, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*				#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

19. Empilha **atual** em **caminho**; assim, **caminho** fica:

(1,3)
(1,2)
(1,1)

20. Empilha **pilha de adjacentes** em **possibilidades**; assim, **possibilidades** fica:

[illegible]

21. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

(1,4)
(2,3)

22. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (1,4) e **pilha de adjacentes** passa a conter:

(2,3)

23. Colocar no **labirinto** na posição indicada por **atual** um caractere *****, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*	*			#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

24. Empilha **atual** em **caminho**; assim, **caminho** fica:

(1,4)
(1,3)
(1,2)
(1,1)

25. Empilha a **pilha de adjacentes** em **possibilidades**; assim, **possibilidades** fica:

	(2,3)		

26. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

(1,5)

27. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (1,5) e **pilha de adjacentes** passa a conter:

28. Colocar no **labirinto** na posição indicada por **atual** um caractere *****, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*	*	*		#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

29. Empilha **atual** em **caminho**; assim, **caminho** fica:

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

30. Empilha a **pilha de adjacentes** em **possibilidades**; assim, **possibilidades** fica:

	(2,3)	

31. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

(1,6)

32. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (1,6) e **pilha de adjacentes** passa a conter:

33. Colocar no **labirinto** na posição indicada por **atual** um caractere *****, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*	*	*	*	#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

34. Empilha **atual** em **caminho**; assim, **caminho** fica:

(1,6)
(1,5)
(1,4)
(1,3)
(1,2)
(1,1)

35. Empilha **a pilha de adjacentes** em **possibilidades**; assim, **possibilidades** fica:

(2,3)	

36. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

37. Agora, o que deveria ser feito seria retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; ocorre que isso será impossível de ser feito, um vez que **pilha de adjacentes** se encontra vazia; até agora estivemos em Modo Progressivo e progredíamos na construção de um caminho em **labirinto** marcado por *; por conta da situação, isso vai mudar e o programa entra em um modo de funcionamento que podemos chamar de Modo Regressivo que funciona da seguinte forma:

38.Desempilha em **atual** uma coordenada do **caminho**; **atual** fica (1,6) e **caminho** passa a conter:

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

39. Retirar do **labirinto**, da posição indicada por **atual** o caractere *****, anteriormente ali colocado, substituindo-o por um **espaço em branco**, indicando que foi dado um passo para trás; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*	*	*		#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

40. Desempilha em possibilidades uma pilha de adjacentes; pilha de adjacentes passa a conter:

e **possibilidades** passa a conter:

	(2,3)	

41. Para que o programa retornasse ao Modo Progressivo, agora, o que deveria ser feito seria retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; ocorre que isso

será impossível de ser feito, uma vez que **pilha de adjacentes** se encontra vazia; assim sendo, o programa se mantém em Modo Regressivo e continuaremos a partir do passo 38, ou seja:

42. Desempilha em **atual** uma coordenada do **caminho**; **atual** fica (1,5) e **caminho** passa a conter:

(1,4)
(1,3)
(1,2)
(1,1)

43. Retirar do **labirinto**, da posição indicada por **atual** o caractere *, anteriormente ali colocado, substituindo-o por um **espaço em branco**, indicando que foi dado um passo para trás; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*	*			#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

44. Desempilha em **possibilidades** uma **pilha de adjacentes**; **pilha de adjacentes** passa a conter:

e **possibilidades** passa a conter:

(2,3)

45. Para que o programa retornasse ao Modo Progressivo, agora, o que deveria ser feito seria retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; ocorre que isso será impossível de ser feito, um vez que **pilha de adjacentes** se encontra vazia; assim sendo, o programa se mantém em Modo Regressivo e continuarmos a partir do passo 38, ou seja:

46. Desempilha em **atual** uma coordenada do **caminho**; **atual** fica (1,4) e **caminho** passa a conter:

(1,3)
(1,2)
(1,1)

47. Retirar do **labirinto**, da posição indicada por **atual** o caractere *, anteriormente ali colocado, substituindo-o por um **espaço em branco**, indicando que foi dado um passo para trás; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*				#
2	#	#	#		#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

48. Desempilha em **possibilidades** uma **pilha de adjacentes**; **pilha de adjacentes** passa a conter:

(2,3)

e **possibilidades** passa a conter:

49. Para que o programa retornasse ao Modo Progressivo, agora, o que deveria ser feito seria retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; felizmente, agora isso volta a ser possível de ser feito, um vez que **pilha de adjacentes** não se encontra vazia; assim sendo, o programa sai do modo Modo Regressivo, volta ao Modo Progressivo e continuarmos a partir do passo 7, ou seja:

50. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (2,3) e **pilha de adjacentes** passa a conter:

51. Colocar no **labirinto** na posição indicada por **atual** um caractere *, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

	0	1	2	3	4	5	6	7
0	#	#	#	#	#	#	#	#
1	E	*	*	*				#
2	#	#	#	*	#	#	#	#
3	#	#	#		#	#	#	#
4	#	#	#	S	#	#	#	#

52. Empilha **atual** em **caminho**; assim, **caminho** fica:

(2,3)
(1,3)
(1,2)
(1,1)

53. Empilha **pilha de adjacentes** em **possibilidades**; assim, **possibilidades** fica:

54. Continuar a partir do passo 6, ou seja, instanciar um objeto do tipo **Pilha<Coordenada>** chamado **pilha de adjacentes** com capacidade de armazenar 3 coordenadas e empilhar ali as possíveis posições adjacentes a **atual** (acima, abaixo, à esquerda ou à direita) válidas e contendo um **espaço em branco** ou o caractere **S**, indicando, respectivamente, passagem livre ou saída; no caso a **pilha de adjacentes** gerada conteria:

(3,3)

55. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (3,3) e **pilha de adjacentes** passa a conter:

56. Colocar no **labirinto** na posição indicada por **atual** um caractere *****, indicando que foi dado um passo naquela direção; assim, **labirinto** fica:

60. Retirar da **pilha de adjacentes** uma coordenada, armazenando-a em **atual**; **atual** fica (4,3) e **pilha de adjacentes** passa a conter:

61. Como na posição indicada por **atual** há um caractere **S**, indicando que a saída do labirinto foi encontrada; desta forma, dizer que o caminho foi encontrado e mostrar na tela todo o conteúdo de **labirinto**.

62. Instanciar então um objeto do tipo **Pilha<Coordenada>** chamado **inverso** e, um a um, desempilhar todo o conteúdo de **caminho** para empilhar em **inverso**; no fim dessa operação, a pilha **caminho** fica vazia e **inverso** passa a conter:

(1,1)
(1,2)
(1,3)
(2,3)
(3,3)

63. Informar agora que o caminho da entrada até a saída será mostrado; desempilhar então, um a um, todas coordenadas armazenadas em inverso e mostrar na tela; será mostrado na tela:

(1,1) (1,2) (1,3) (2,3) (3,3)

Observações Finais:

- Caso, o programa entre em Modo Regressivo e não consiga mais voltar ao Modo Progressivo, esgotando todo o conteúdo de **possibilidades** e **caminho**, informar que não existe caminho que leva da entrada até a saída.
- É imprescindível: (a) que o programa seja adequadamente dividido em classes; (b) que, na medida do possível, as classes sejam genéricas; (c) que todas as validações cabíveis sejam feitas por todos os métodos e que incorretudes sejam sinalizadas através de exceções que, posteriormente, sejam apropriadamente tratadas; (d) que as classes possuam os métodos obrigatórios cabíveis; e (e) que as classes sejam totalmente documentadas com Javadoc.
- O presente trabalho deve ser feito em grupos de até 3 alunos e deverá ser entregue, impreterivelmente no **dia 04 de abril** (turma da Profª Daniele) e no dia **05 de abril** (turma da Profª Eliane).

Bom trabalho!

**Profs André Carvalho, Daniele Frosoni,
Eliane Azevedo e Patrícia Nogueira**

Campinas, 28/Fev/2019