

Exercício 1.

Considere o problema de um mouse preso que tenta encontrar o caminho para uma saída em um labirinto (Figura 1). O rato espera escapar do labirinto tentando sistematicamente todas as rotas. Se chegar a um beco sem saída, ele refaz seus passos até a última posição e começa pelo menos mais um caminho não experimentado. Para cada posição, o mouse pode ir em uma das quatro direções: direita, esquerda, baixo, cima. Independentemente de quão próximo esteja da saída, ele sempre tenta os caminhos abertos nessa ordem, o que pode levar a alguns desvios desnecessários. Ao reter informações que permitem retomar a busca depois que um beco sem saída é alcançado, o mouse usa um método chamado *backtracking*.


	0	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	0	1	0	0	0	0	0	1
2	1	0	1	0	1	1	1	0	1	0	1
3	2	0	1	0	0	0	0	0	1	0	1
4	1	0	1	1	1	1	1	0	1	0	1
5	1	0	1	0	1	0	0		1	0	1
6	1	0	0	0	1	0	1	0	1	0	1
7	1	1	1	1	1	0	1	0	0	0	1
8	1	0	1	0	1	0	1	0	1	1	1
9	1	0	0	0	0	0	1	0	0	0	1
10	1	1	1	1	1	1	1	1	1	1	1

Figura 1: (a) Um rato num labirinto; (b) matriz de caracteres bidimensional representando a situação.

O labirinto é implementado como uma matriz bidimensional de números inteiros em que as passagens são marcadas com 0s, paredes com 1s, a posição de saída com 2, e a posição inicial do rato pelo número 3.

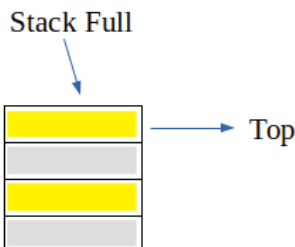
Implemente o jogo descrito acima e utilize uma pilha de posições como memória, para que o rato guarde as informações necessárias à escapada.

- Inicie o rato em uma posição aleatória válida.
- O crie uma função para indicar a próxima direção que o rato vai seguir.
- Indique se ele escapou ou não.
- Crie um loop de jogo para que o rato continue procurando até encontrar uma saída.
- Toda vez que uma posição válida é encontrada o rato empilha a posição atual e muda de posição, marque esta posição como *visitada*, se o caminho resultar em um beco sem saída você deve desempilhar posições da memória e ir testando até encontrar uma posição que possua uma vizinha válida ainda não visitada.

Exercício 2.

Todos nós sabemos sobre as pilhas também conhecidas como estruturas *Last-In-First-Out* (LIFO). Stack tem principalmente duas operações principais, a saber, push e pop, em que push insere um elemento no topo e pop remove um elemento do topo da pilha.

Agora, sempre que uma implementação de pilha é considerada, seu tamanho é pré-determinado ou fixo. Mesmo que seja alocado dinamicamente, mesmo assim uma vez feito seu tamanho não pode ser alterado. E, portanto, surge uma condição chamada “pilha cheia”.

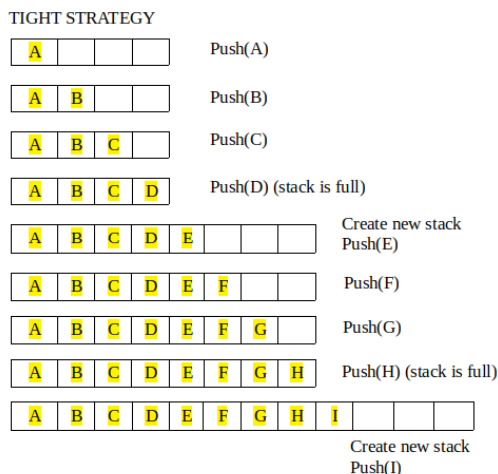


Mas e se uma pilha puder crescer conforme mais elementos são inseridos ou mais elementos serão inseridos no futuro. Lembre-se, estamos falando sobre Stack baseado em array. Pilha expansível é o conceito de alocar mais memória de forma que a condição de “pilha cheia” não surja facilmente.

Uma pilha baseada em array expansível pode ser implementada alocando uma nova memória maior do que a memória da pilha anterior e copiando elementos da pilha antiga para a nova. E então, finalmente, mude o nome da nova pilha para o nome que foi dado à pilha antiga.

Existem duas estratégias para a pilha crescente:

1. **Estratégia de crescimento fixo:** adicione uma quantidade constante à pilha antiga ($N + c$)
2. **Estratégia de crescimento variável:** dobre o tamanho da pilha antiga ($2N$)



Existem duas operações nesta implementação de pilha:

1. **operação Push Regular:** adicione um elemento no topo da pilha,
2. **operação Push Especial:** crie uma nova pilha de tamanho maior do que a pilha antiga (de acordo com uma das estratégias acima) e copie todos os elementos da pilha antiga e, em seguida, empurre o novo elemento para a nova pilha.

Exercício 3.

Dado um arquivo de extensão **.c**, escreva um programa para descobrir se há parênteses, colchetes ou chaves desequilibrados.

Uma abordagem para verificar parênteses equilibrados é usar pilha. Cada vez que um parêntese aberto for encontrado, coloque-o na pilha, e quando um parêntese de fechamento for encontrado, combine-o com o topo da pilha e remova-o. Se a pilha estiver vazia no final, não há desbalanceamento, caso contrário, há desbalanceamento.