Projeto Computacional - Labirinto

1. Ajude um rato a encontrar um pedaço de queijo num labirinto como o do desenho abaixo (`R` denota a posição do rato e `Q` a do queijo):

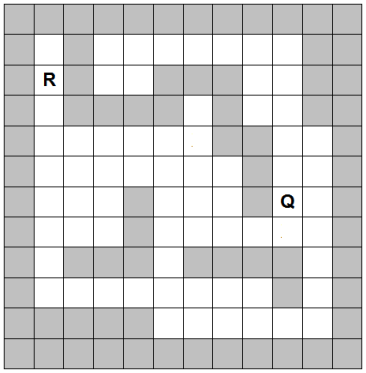

Figura 1: Labirinto

Uma possível representação computacional se dá por uma matriz quadrada `L`, cujo elemento L_{ij} vale `-1`, se corresponder a uma passagem livre, ou `-2`, caso contrário.

Neste contexto, diferentes abordagens poderiam ser exploradas para fazer com que o rato encontre o caminho até o queijo. Contudo, algumas só são eficientes em cenários específicos. Outras demandam o teste de muitas condições em cada passo, deixando a implementação pouco eficiente.

Uma abordagem para resolver esse problema consiste em marcar com o número `k`, `k= 1,2,...`, todas as casas livres que estejam a exatamente `k` passos de distância do queijo (pelo caminho mais curto possível). Suponha que, a cada passo, o rato possa se deslocar apenas uma casa na vertical ou na horizontal. Então, rotula-se inicialmente a posição do queijo com `k=0` e, para cada `k` subsequente, examinam-se todas as casas livres do labirinto, marcando-se com `k` aquelas ainda não marcadas E que sejam adjacentes a alguma casa marcada com `k-1`. Para o exemplo acima, teríamos configuração ilustrada na Figura 2(a). Neste contexto, o caminho delineado na Figura 2(b) poderia ser escolhido.

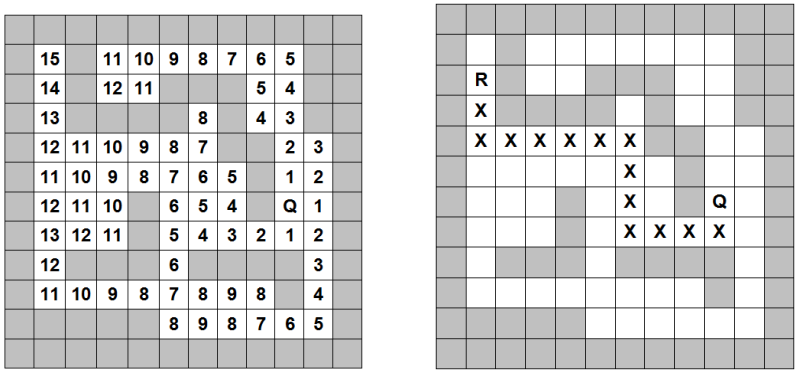

Figura 2: Exemplos de configuração e caminho.

Você está recebendo um programa com as seguintes funções:

(a) `int ** alocaMatriz(int tam)`: função que aloca dinamicamente uma matriz de tamanho `tam` × `tam`;

(b) `void inicializaLabirinto(int **m, int tam)`: inicializa a matriz `m`, de dimensão `tam`×`tam`, com `-1` nas posições livres e `-2` nas demais (correspondentes às paredes). OBS.: você pode alterar a inicialização, lendo do teclado os índices correspondentes às paredes, iniciando com uma configuração padrão, etc (este último caso é o que está implementado sendo o mais adequado para esta fase inicial de testes).

(c) `void criaMatrizCusto(int **m, int tam, int xQueijo, int yQueijo)`: cria a matriz de custo, segundo o processo descrito acima (`xQueijo` e `yQueijo` denotam as coordenadas do queijo).

TAREFAS

Nesta fase inicial considere os valores de coordenadas (queijo e rato), tamanho e padrão de inicialização do labirinto conforme indicado no código disponibilizado e implemente as seguintes funções:

1. `Coordenada* calculaCaminho (int **m, int tam, int xRato, int yRato, int *nroPassos)`: esta função recebe a posição do rato e a matriz de custo já preenchida (denotada por `m` e de tamanho `tam` × `tam`) e preenche vetor `caminho` com os índices de linha e coluna do caminho a ser percorrido entre o rato e o queijo (note que esse vetor será alocado dentro da função). Além disso, retorna na variável `nroPassos` (passada por referência) o número de passos do caminho. O tipo Coordenada é dado por:

```
typedef struct coordenada
{
    int x;
    int y;
} Coordenada;
```

2. Uma função `void imprimeCriativ(int **m, int tam)` que mostre na tela o caminho da forma mais criativa que você conseguir.

3. Usando as funções anteriores, o programa deverá calcular e imprimir na tela o menor caminho percorrido entre o rato e o queijo.

Lembre-se de desalocar ao final da função principal todos os vetores e matrizes alocados!!!!

DESAFIOS

Implemente novas funções ou modifique as funções já implementadas conforme descrito abaixo:

1. `main()`: modifique a função principal, de forma a ler do teclado a posição dos alvos (ie., do rato e do queijo) e o tamanho do labirinto (`tam`). Cuidado para que não haja sobreposição das paredes e dos alvos.

   (a) Para isso, implemente uma função `int testaCoordenadas(Coordenada coord, int tam, char alvo[])` que recebe como parâmetros uma variável `coord` tipo Coordeanada, o tamanho do labirinto `tam` e o alvo (string contendo "Queijo"ou "Rato"), e retorne 1 se as coordenadas do alvo estiverem dentro dos limites permitidos (observe que a implementação disponível não permite que o rato ou o queijo estejam posicionados nas bordas do labirinto) ou -1 caso as coordenadas estejam nas bordas ou fora dos limites (a mensagem do tipo de erro e do alvo específico deve ser impressa na função `testaCoordenadas(..)`).

(b) teste agora uma nova função `int testaCoordenadas2(Coordenada *coord, int tam, char alvo[])` que recebe como parâmetros uma variável `coord` tipo ponteiro para Coordeanada, o tamanho do labirinto `tam` e o alvo (string contendo "Queijo"ou "Rato") mas que faça o mesmo que a função `int testaCoordenadas(Coordenada coord, int tam, char alvo[])`. É usual passar ponteiros para struct ao invés da própria struct (evita cópia de structs).

2. Modifique a função `void inicializaLabirinto(int **m, int tam)` para que, além das coordenadas do rato e queijo lidas na função principal, o padrão do labirinto também possa ser fornecido pelo usuário (via leitura das posições das paredes). Cuidado para que as paredes não sobreponham a posição do rato ou queijo.

3. Modifique o programa para que o rato ou o queijo ou ambos possam ser posicionados nas bordas.

4. Modifique o programa para que a função `void criaMatrizCusto(int **m, int tam, int xQueijo, int yQueijo)` possa realizar o preenchimento da matriz de forma mais eficiente.