Exercício Programa 2 - Caminho em um labirinto
MAC0122 Princípios de Desenvolvimento de Algoritmos - POLI

Prof. Ronaldo Fumio Hashimoto

1 Descrição do Exercício Programa

1.1 Visão Geral

Um dado labirinto é composto por várias salas. As salas são interligadas por corredores e há apenas uma entrada e uma saída. O seu objetivo neste exercício programa (EP) é desenvolver um algoritmo que encontra um caminho que leva da entrada à saída do labirinto ou determina que tal caminho não existe.

1.2 Passos a serem implementados

A seguir são listados os passos a serem seguidos para a implementação deste EP. Cada um destes passos é descrito com mais detalhes nas seções subsequentes.

1. Ler o labirinto de um arquivo externo.
2. Armazenar as salas do labirinto usando uma `struct` contendo uma lista ligada das salas adjacentes.
3. Percorrer o labirinto, encontrando a saída usando busca em profundidade implementada com pilha.

2 Leitura do labirinto

2.1 Descrição da representação

Um labirinto será representado por n salas e m corredores[1]. Cada sala receberá um identificador único (id) com valor entre 0 e $n-1$. Cada corredor conectará duas salas e será identificado pelo par das salas que ele conecta (por exemplo, o corredor que conecta a sala 1 à sala 2 será identificado pelo par $\{1, 2\}$). Assume-se que, dadas duas salas distintas, não há mais que um corredor que as conecta. Não há corredores conectando uma sala a ela mesma. Além disso, há apenas uma entrada e uma saída. A entrada será dada na sala com id 0, enquanto que a saída será a sala com id $n-1$.

Um exemplo de labirinto é mostrado na Fig. 1:

[1] Em termos mais formais, o labirinto será representado como um grafo, onde as salas representam o conjunto de vértices e os corredores representam o conjunto de arestas. Os

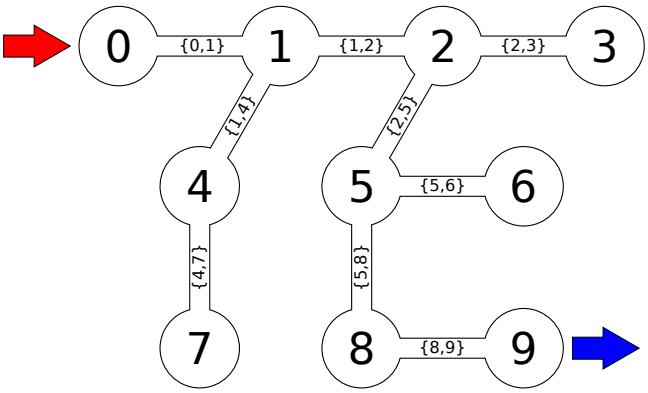

Figura 1: Exemplo de labirinto com $n = 10$ salas e $m = 9$ corredores. A seta vermelha indica a entrada do labirinto e a seta azul indica a saída do labirinto.

2.2 Formato de arquivo do labirinto

O labirinto deverá ser lido de um arquivo externo, usando os comandos de leitura da biblioteca `stdio.h`. O arquivo de labirinto terá a seguinte estrutura:

```
n
m
corredor_1
corredor_2
...
corredor_m
```

Mais especificamente, o arquivo de entrada conterá $m + 2$ linhas, onde a primeira linha consiste do valor n, a segunda linha consiste do valor m e as m linhas seguintes consistem dos corredores do labirinto, representados como o par de salas que o corredor conecta. Por exemplo, o arquivo abaixo consiste do labirinto da Fig. 1:

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

algoritmos e estruturas de dados usados neste EP podem ser usados em quaisquer outros problemas que podem ser modelados usando grafos.

3 Armazenamento do labirinto

3.1 Representação das salas

Dado um labirinto, diremos que duas salas são adjacentes se existe um corredor que as conecta (por exemplo, na Fig. 1, a sala 1 é adjacente às salas 0, 2 e 4). Uma forma de se armazenar o labirinto consiste em armazenar, para cada sala, o conjunto de salas adjacentes a ela. Por exemplo, o labirinto da Fig. 1 pode ser armazenado como descrito na Tabela 1.

id da sala	Salas adjacentes
0	1
1	0, 2, 4
2	1, 3, 5
3	2
4	1, 7
5	2, 6, 8
6	5
7	4
8	5, 9
9	8

Tabela 1: Tabela de salas adjacentes.

Assim sendo, podemos descrever uma sala usando uma `struct` que contém o conjunto de salas adjacentes a ela. Para este EP, definiremos uma sala como uma `struct room` que contém uma lista ligada `adj` com as salas adjacentes, tal como mostrado abaixo.

```
typedef struct item Item;
struct item{
  int id;
  Item* next;
};
typedef struct room Room;
struct room{
  Item* adj;
};
```

4 Encontrando a saída do labirinto

4.1 Estratégia para percorrer o labirinto

Uma conhecida estratégia para encontrar a saída de um labirinto consiste em, a partir do ponto de vista do caminhante, virar sempre para a mesma direção quando uma encruzilhada é encontrada. Esta estratégia é equivalente a colocar

uma das mãos em uma das paredes do labirinto e caminhar sem nunca tirar a mão da parede.

Por exemplo, suponha que o caminhante se encontre na entrada do labirinto da Fig. 1 e decida sempre tomar o caminho mais à direita (equivalente a andar sempre com a mão direita encostada na parede). Então, o caminhante percorre o corredor que leva da sala 0 até a sala 1, onde ele se depara com uma encruzilhada (pegar o corredor para a sala 2 ou para a sala 4). Seguindo a estratégia de sempre tomar o caminho à direita, o caminhante seguiria para a sala 4, para a sala 7 e assim por diante. No final, o caminhante encontra a saída (sala 9) após passar pelas seguintes salas: $0, 1, 4, 7, 4, 1, 2, 5, 8, 9$.

Se não existem ciclos no labirinto (dois caminhos diferentes que levam de uma sala a outra), então esta estratégia consiste em fazer uma **busca em profundidade**.

4.2 Busca em profundidade

Em termos genéricos, assumindo que existe caminho que leva da entrada à saída do labirinto, a estratégia de encontrar a saída do labirinto usando busca em profundidade consiste do seguinte pseudocódigo:

1. Coloque o caminhante na sala inicial.
2. Enquanto o caminhante não encontrar a sala com a saída:
   (a) Se há pelo menos uma sala adjacente na qual o caminhante nunca esteve, mova-o para uma destas salas.
   (b) Senão, retorne para a sala da qual o caminhante veio.

Na estratégia descrita, a instrução "mova-o para uma destas salas" consiste em movê-lo para a sala mais à direita. Note que esta estratégia também automaticamente retorna para a sala anterior quando não há mais caminho porque o caminhante dá uma volta pela sala até retornar ao corredor pelo outro lado da parede.

Neste EP, como não é possível definir o que é "mais à direita", você pode escolher qualquer corredor que leva a uma sala não-visitada. Para implementar a parte que retorna para a sala anterior, você deverá fazer uso de uma **pilha**. Note que, se você empilhar o id das salas percorridas pelo caminhante, então retornar para a sala anterior consiste simplesmente em retornar para a sala indicada no topo da pilha.

5 Implementação

Para a implementação deste EP, você pode considerar que o grafo do labirinto não possui ciclos e é conexo[2], ou seja, dadas duas salas quaisquer do labirinto,

[2]Usando termos de grafos, dizemos que o grafo do labirinto é uma árvore. Toda árvore com n vértices tem necessariamente $m = n - 1$ arestas.

existe exatamente um caminho (sem passar pelo mesmo corredor duas vezes) que liga essas duas salas.

5.1 Código

1. Você pode criar quantos arquivos achar necessário para implementar seu EP. Lembre-se, porém, de não violar o padrão interface/implementação/cliente.
2. Você pode adicionar mais variáveis na `struct room` se você assim achar necessário.
3. A pilha usada na busca em profundidade pode ser implementada tanto como lista ligada quanto como um vetor.
4. Um dos objetivos deste EP é treinar o uso de pilhas. Não escreva um algoritmo recursivo.
5. Seu EP deve compilar com os parâmetros `-Wall -ansi -pedantic -O2`. Warnings ou erros de compilação acarretarão em desconto de nota.

5.2 Saída esperada

A saída do seu EP deve ser a sequência das salas percorridas pelo caminhante até chegar na saída do labirinto. Por exemplo, usando o labirinto da Fig. 1, uma possível saída é:

```
0 1 4 7 4 1 2 5 8 9
```

Sua saída pode ser diferente dependendo da forma com que a instrução "mova-o para uma destas salas" do pseudocódigo for implementada.

6 Entrega do EP

Você deve enviar todos os códigos-fonte (`.c` e `.h`) usados para a implementação do seu EP. Além disso, você deve enviar um arquivo `readme.txt` contendo as seguintes informações:

1. Sistema operacional utilizado para compilar o programa.
2. Programa usado para compilar o programa (terminal / Codeblocks / XCode etc.)
3. Instruções de compilação e de uso do programa.

Os arquivos enviados devem ser enviados em uma única pasta compactada como um arquivo `.zip`.