Caminhos em Labirinto - Heurística

Técnicas de Programação - 2025/02

Última atualização: 26/10/2025 19:57

O algoritmo da mão direita é muito complicado! Vamos aproveitar hoje e "reciclar" uma ideia que já vimos outras vezes: a enumeração exaustiva usando backtracking. Vocês já viram essa ideia em IA com outro nome: **busca em profundidade**. Em cada posição, a busca em profundidade tenta visitar todas as casas vizinhas que ainda não foram visitadas.

Definição

Sejam duas casas distintas `f`(fonte) e `d`(destino), existe um caminho que vai de `f` até `d` se , e somente se, pelo menos uma das alternativas abaixo é verdade

- existe um caminho entre a casa acima de `f` até `d`
- existe um caminho entre a casa abaixo de `f` até `d`
- existe um caminho entre a casa à esquerda de `f` até `d`
- existe um caminho entre a casa à direita de `f` até `d`

As 4 condições podem ser resumidas em uma só: **existe um caminho entre um dos vizinhos de `f` até `d`**. Repare que essa ideia não funciona somente no labirinto, mas em qualquer situação em que tenhamos a ideia de nós que estão conectados a outros. Estudaremos essas estruturas e seus algoritmos no próximo semestre.

O desenvolvimento de um algoritmo para a busca em profundidade vem direto da definição acima. Vamos tentar destrinchá-la abaixo. Nosso problema será: dado uma casa inicial `f`, existe caminho até uma casa destino `d`?

Um algoritmo para este problema claramente seria recursivo, já que a definição apresentada acima foi apresentada dessa forma. Vamos então definir os seguintes pontos antes de montar esse algoritmo:

1. estando em uma casa `C`, quais são as chamadas recursivas que posso fazer?
2. qual seria a base da recursão? Ou seja, em quais casos **não fazemos chamadas recursivas?**

Chamadas recursivas

Vamos começar pelo item 1. No labirinto abaixo, queremos sair da casa `C` e chegar na casa `D`. Toda casa marcada com `.` nunca foi visitada, enquanto toda casa marcada com `x` já foi visitada.

```
##########
#........#
#..D.....#
#........#
#........#
#........#
#........#
#......Cx#
#.....xxx#
##########
```

Exercício: Marque abaixo em quais vizinhos deveriam ser feitas chamadas recursivas?

- ☐ cima
- ☐ baixo
- ☐ esquerda
- ☐ direita

Vamos para mais um labirinto.

```
##########
#......Cx#
#..D...xx#
#......xx#
#......xx#
#.....xxx#
#.xxxxxxx#
#.xxxxxxx#
#.xxxxxxx#
##########
```

Exercício: Marque abaixo em quais vizinhos deveriam ser feitas chamadas recursivas?

- ☐ cima
- ☐ baixo
- ☐ esquerda
- ☐ direita

```
##########
#......xx#
#..D...xx#
#......xx#
#xx....xx#
#Cx...xxx#
#xxxxxxxx#
#xxxxxxxx#
#xxxxxxxx#
##########
```

Exercício: Marque abaixo em quais vizinhos deveriam ser feitas chamadas recursivas?

- ☐ cima
- ☐ baixo
- ☐ esquerda
- ☐ direita

Registre suas conclusões abaixo

Conclusões

Critérios de parada

Vamos agora definir o critério de parada de nosso algoritmo. Levando em conta que temos uma casa f fonte e uma casa d destino. . . .

Exercício: escreva a condição de parada para quando chegamos na casa d

Exercício: escreva a condição de parada para quando todos os vizinhos já foram vizitados

---

Montando o algoritmo final

Todos nossos algoritmos de enumeração exaustiva terão, essencialmente, três partes:

1. **BASE** - contém as condições de parada e os valores de retorno correspondentes. Aqui **não** são feitas chamadas recursivas
2. **RECURSÃO** - contém uma ou mais chamadas para a mesma função, mas com um tamanho de entrada menor.
3. **RETORNO** - junta os valores de retorno das chamadas do item anterior e define o valor de retorno da chamada atual.

Vamos agora tentar organizar nosso algoritmo. Usaremos as seguintes variáveis/funções auxiliares

- `L` - labirinto
- `F` - casa fonte
- `D` - casa destino
- `OCUPADA(C)` - devolve *true* se a casa já está ocupada
- `OCUPAR-CASA(C)`
- `DESOCUPAR-CASA(C)`

```
TEM-CAMINHO(L, F, D)

# BASE
    if f = D then
    | return true
    end


# RECURSÃO
    OCUPAR-CASA(f)
    for v in (f_C, f_D, f_B, f_E) do
    |   if !OCUPADA(v) e LIVRE(v) then
    |   |   if TEM-CAMINHO(L, v, d) then
    |   |   |   return true
    |   |   end
    |   end
    end
# RETORNO
    DESOCUPAR-CASA(f)
    return false
```