Projeto de algoritmos: uma visão prática

Aula 07

Rafael Melo
Instituto de Computação
Universidade Federal da Bahia

RELEMBRANDO: Passos do *Backtracking*

- *Defina a representação da solução (normalmente um array).*
- *Estabeleça os conjuntos A_1 , . . . , A_n e a ordem em que seus elementos são processados.*
- *Estabeleça as condições que tornam uma solução parcial válida.*
- *Estabeleça um critério para identificar a solução final.*

RELEMBRANDO: Passos do *Backtracking*

- O algoritmo induz uma árvore do espaço de estados.
- Cada nó representa enuplas parciais com os primeiros *i* componentes da solução parcial.
- Para cada $(x_1 , x_2 , \dots , x_i)$ válida, o algoritmo acha o próximo elemento em A_{i+1} que é consistente com
  - os valores de $(x_1 , x_2 , \dots , x_i)$;
  - as restrições do problema;
- e o adiciona à $(x_1 , x_2 , \dots , x_i)$ como seu *(i + 1)-ésimo* componente.

Problema do labirinto em um tabuleiro (*maze*)

Problema do labirinto em um tabuleiro

- Dado um labirinto representado por uma matriz de tamanho $m \times n$ (m linhas e n colunas), uma posição inicial $p_i = (x_i, y_i)$ e uma posição final $p_f = (x_f, y_f)$, tal que $p_i \neq p_f$, determinar se existe um caminho entre p_i e p_f.
  - Considere células livres representadas por 1's e células bloqueadas por 0's

p_i

1	0	0	1	1	1	0
1	0	0	1	0	1	0
1	1	1	1	0	1	0
0	1	0	0	0	1	0
0	1	0	1	1	1	0
0	1	0	1	0	0	0
1	1	0	1	1	1	1

p_f

Preliminares para o algoritmo

- Representação da solução: uma matriz indicando as células visitadas
- Solução parcial é promissora se satisfizer:
  - A célula está dentro dos limites da matriz labirinto
  - A célula está livre
  - A célula ainda não foi visitada (evitar busca cíclica)
- Critério para definição de solução final: *célula é* $p_f = (x_f, y_f)$.

Algoritmo

```python
def VALID(x, y, m, n, visitado):
    return 0 <= x < m and 0 <= y < n and labirinto[x][y] == 1 and visitado[x][y] == False

def BACKTRACKING_LABIRINTO(labirinto, x, y, x_final, y_final, caminho, visitado):
    visitado[x][y] = True

    if x == x_final and y == y_final:
        return caminho + [(x, y)]

    direcoes = [(1, 0), (0, 1), (-1, 0), (0, -1)]
    for dx, dy in direcoes:
        x_vizinho, y_vizinho = x + dx, y + dy
        if VALID(x_vizinho, y_vizinho, len(labirinto), len(labirinto[0]),visitado) :
            novo_caminho = BACKTRACKING_LABIRINTO(labirinto, x_vizinho, y_vizinho, x_final, y_final, caminho + [(x, y)], visitado)
            if novo_caminho:
                return novo_caminho

    return None

def CHAMADA_INICIAL(labirinto, posicao_inicial, posicao_final):
    #labirinto: lista de lista (matriz); posicao_inicial e posicao_final: tupla (x, y)
    x_inicial, y_inicial = posicao_inicial
    x_final, y_final = posicao_final
    visitado = [[False for _ in range(len(labirinto[0]))] for _ in range(len(labirinto))]
    return BACKTRACKING_LABIRINTO(labirinto, x_inicial, y_inicial, x_final, y_final, [], visitado)
```

Tempo de execução

- *O(m x n)*
- Por que?

Atividades práticas

Contato:
rafael.melo@ufba.br
rafaelmelo.ufba@gmail.com