Problema do Ciclo Hamiltoniano

Método “Composição Latina”

https://malbarbo.pro.br/arquivos/2013/5189/ademir-pcv.pdf

Algoritmo

Trata-se de um método algébrico para enumeração de caminhos hamiltonianos. Este método gera todos os caminhos simples por multiplicação de matrizes.

O resultado é uma matriz P_i, onde cada elemento representa um caminho entre os vértices **s** e **t**, primeiro e último vértices do ciclo.

Passos do algoritmo:

1. Construir a matriz B e a matriz de adjacência A do grafo dado
2. Fazer $P_1 \leftarrow A$
3. Para i = 1,2,...., n-2 fazer $P_{i+1} \leftarrow B * P_i$
4. Passo final (Eliminar caminhos irregulares)

B é uma matriz n por n, onde n é o número de vetores, e $B_{ij} = v_j$ (se existir uma aresta (v_i, v_j), caso contrário é 0)

Código: principais funções

```python
''' iterações do algoritmo '''


def aplicacao(A, B, n):
    itr = 1
    Result = solucao(A, B, n, itr)
    itr += 1
    for i in range(n-3):
        Result = solucao(Result, B, n, itr)
        itr += 1

    return Result
```

```python
''' elimina caminhos que não fazem parte '''


def verificar(strng, vet, vt, n):
    if(vet == vt):
        return '0'
    aux = strng.split('+')
    novo = ''
    for i in range(len(aux)):
        if(not(vet in aux[i] or vt in aux[i] or len(aux[i]) != n)):
            novo += aux[i]
            if(i < len(aux)-1):
                novo += '+'
    if(novo == ''):
        return '0'
    return novo
```

```python
''' função que realiza uma multiplicação de matrizes '''


def solucao(A, B, n, itr):
    i, j, k = [0, 0, 0]
    vet = []
    Linhas = []
    for x in range(n):  # conta o número de vértices e enumera alfabeticamente
        vet.append(chr(x + 97))
    while(i < n):
        Colunas = []
        while(j < n):
            somatorio = '0'
            while(k < n):
                # os elementos das matrizes recebidas são multiplicados
                p = produto(B[i][k], A[k][j])
                # e somados, resultando no elemento da matriz saída
                somatorio = soma(somatorio, p)
                k += 1
            k = 0
            somatorio = verificar(somatorio, vet[i], vet[j], itr)
            Colunas.append(somatorio)
            j += 1
        j = 0
        Linhas.append(Colunas)
        i += 1

    return Linhas
```

Exemplo de saída

```
['0', 'b', '0', 'd', '0']
['0', '0', '0', 'd', 'e']
['0', 'b', '0', '0', 'e']
['0', '0', 'c', '0', '0']
['a', '0', 'c', '0', '0']

['0', '1', '0', '1', '0']
['0', '0', '0', '1', '1']
['0', '1', '0', '0', '1']
['0', '0', '1', '0', '0']
['1', '0', '1', '0', '0']

resultado:
['0', '0', '0', '0', 'bdc+dcb']
['dce', '0', 'ead', '0', '0']
['0', '0', '0', 'bea+eab', '0']
['cbe', 'cea', '0', '0', '0']
['0', 'adc', 'abd', '0', '0']
```

Complexidade

- $\approx n^5 + n^2 - 3n$
- $O(n^5)$