Tentantiva e Erro (Backtracking)

Prof. Luciano Antonio Digiampietri

Tentativa e Erro - **Encontrar uma solução**

```
tentar(estadoAtual) {
  se estadoAtual = estadoFinal retorne sucesso
  para cada possibilidade {
    se for plausível {
      executa ação – atualiza estado
      chama recursivo: tentar(estadoAtualizado)
      se chamada recursiva retornou sucesso
                                  retorne sucesso
      desfaz ação – retorna para estado anterior
    }
  }
  retorne insucesso
```

Tentativa e Erro - **Encontrar uma solução**

```
tentar(estadoAtual) {
  se estadoAtual = estadoFinal retorne sucesso
  para cada possibilidade {
    se for plausível {
      executa ação – atualiza estado
      chama recursivo: tentar(estadoAtualizado)
      se chamada recursiva retornou sucesso
                                  retorne sucesso
      desfaz ação – retorna para estado anterior
    }
  }
  retorne insucesso
```

Complexidade no pior caso (geral)

$T(n) = O(final())$ para n=0

(não há nada mais a ser feito)

$T(n) = \mathbf{pos} * T(n-1) + \mathbf{pos} * O(ação())$ para n>0

Sendo:

$O(final())$ – complexidade para verificar se o estado atual é igual ao final (muitas vezes é O(1)).

pos = número de possibilidades/alternativas

$O(ação())$ – complexidade de executar/desfazer a ação atual (muitas vezes é O(1)).

Se $O(final())$ e $O(ação())$ **$\in O(1)$:**

$T(n) = O(1)$ para n=0

$T(n) = \mathbf{pos} * T(n-1) + O(1)$ para n>0

Se resolvermos a equação: **$T(n) \in O(pos^n)$**

Tentativa e Erro - **Encontrar a melhor solução**

```
tentar(estadoAtual) {
  se estadoAtual = estadoFinal {
    se solucaoAtual > melhorSolucao
            melhorSolucao = solucaoAtual
        retorne
  }
  para cada possibilidade {
    se for plausível {
      executa ação – atualiza estado
      chama recursivo: tentar(estadoAtualizado)
      desfaz ação – retorna para estado anterior
    }
  }
  retorne
}
```

Tentativa e Erro - **Encontrar a melhor solução**

```
tentar(estadoAtual) {
  se estadoAtual = estadoFinal {
    se solucaoAtual > melhorSolucao
            melhorSolucao = solucaoAtual
        retorne
  }
  para cada possibilidade {
    se for plausível {
      executa ação – atualiza estado
      chama recursivo: tentar(estadoAtualizado)
      desfaz ação – retorna para estado anterior
    }
  }
  retorne
}
```

Complexidade no pior caso (geral)

$T(n) = O(final())$ para n=0

(não há nada mais a ser feito)

$T(n) = \mathbf{pos} * T(n-1) + \mathbf{pos} * O(ação())$ para n>0

Sendo:

$O(final())$ – complexidade para verificar se o estado atual é igual ao final e para atualizar melhor solução (muitas vezes é O(1)).

pos = número de possibilidades/alternativas

$O(ação())$ – complexidade de executar/desfazer a ação atual (muitas vezes é O(1)).

Se $O(final())$ e $O(ação())$ $\in$ **O(1):**

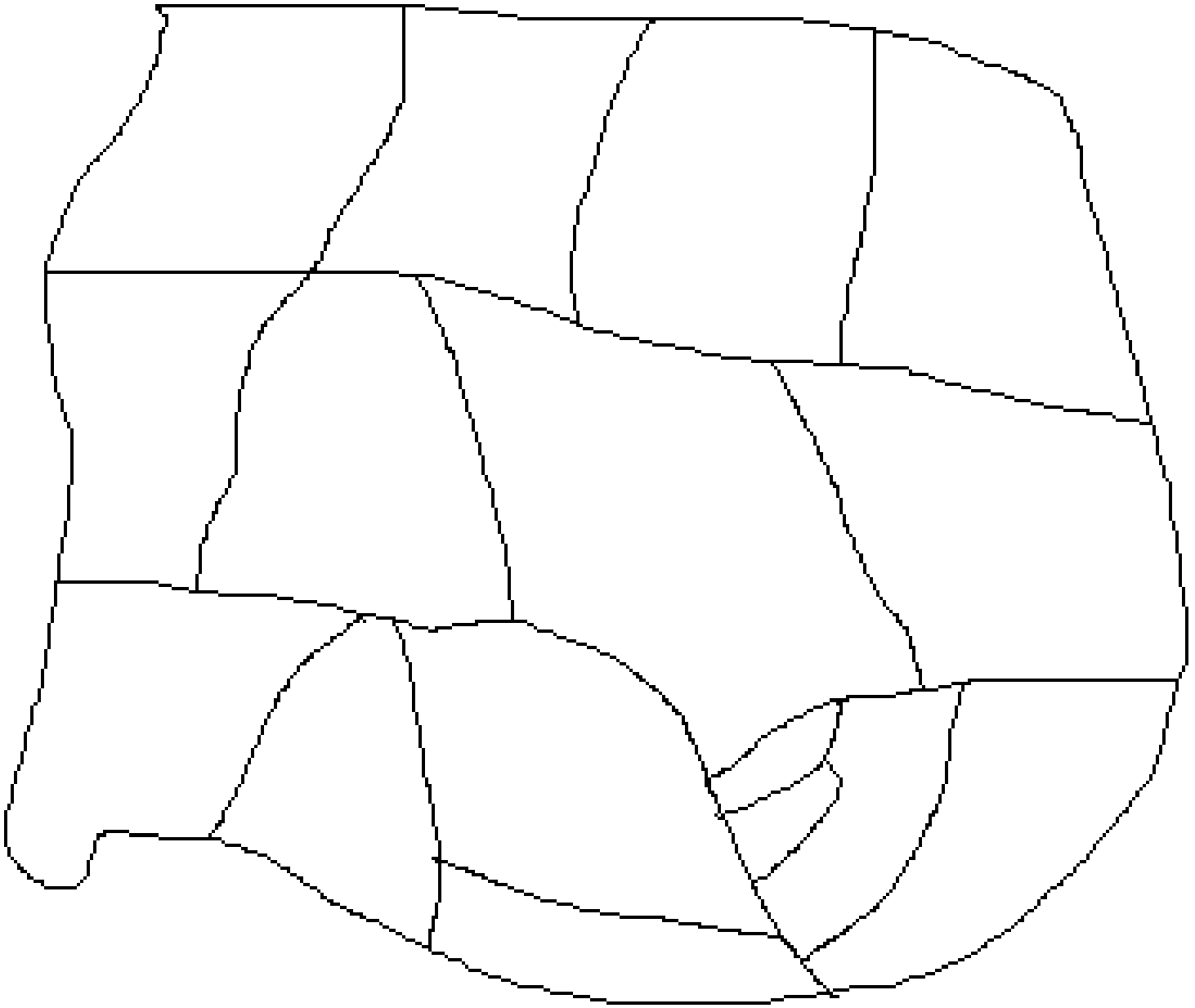
$T(n) = O(1)$ para n=0

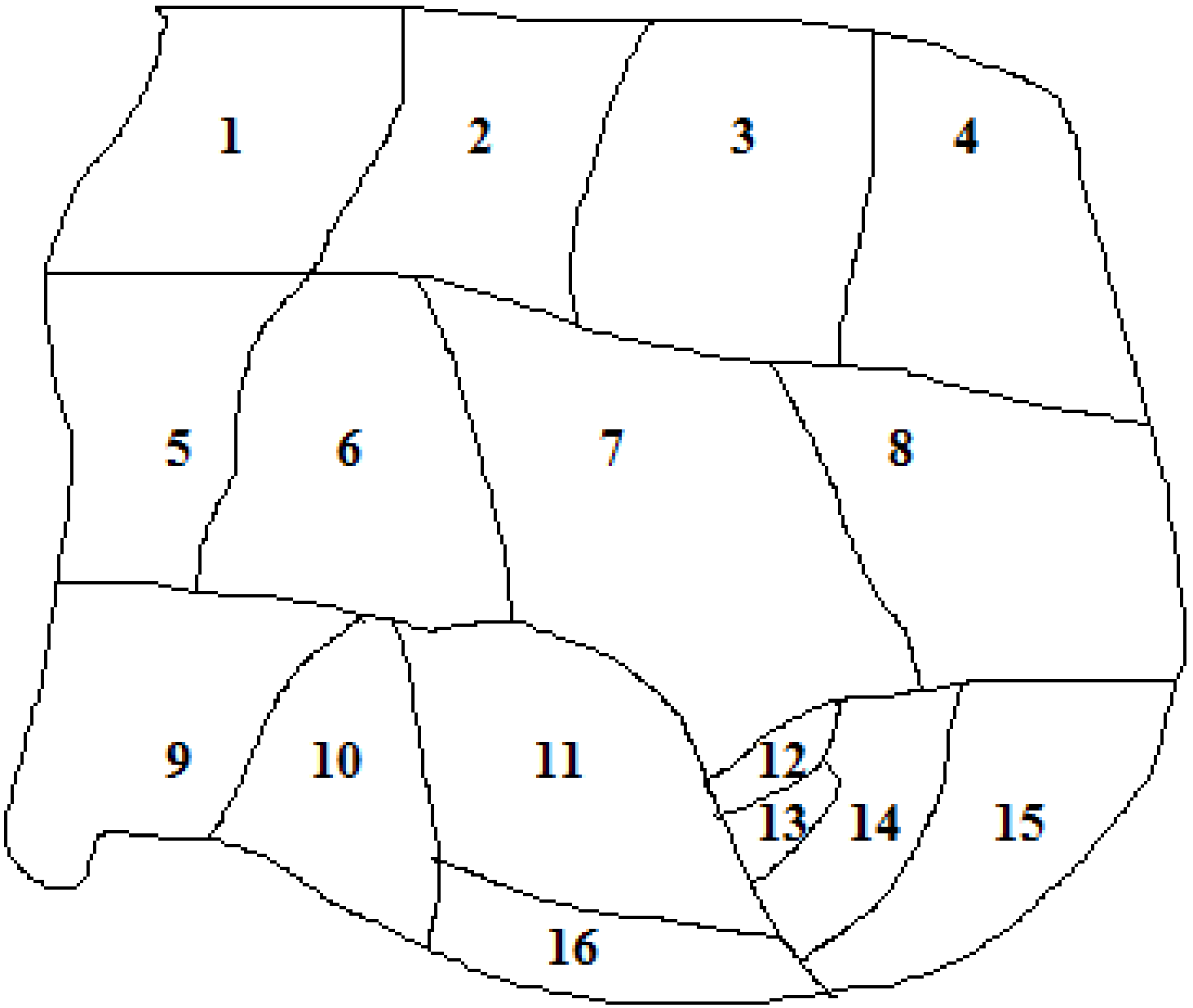
$T(n) = \mathbf{pos} * T(n-1) + O(1)$ para n>0

Se resolvermos a equação: $\mathbf{T(n) \in O(pos^n)}$

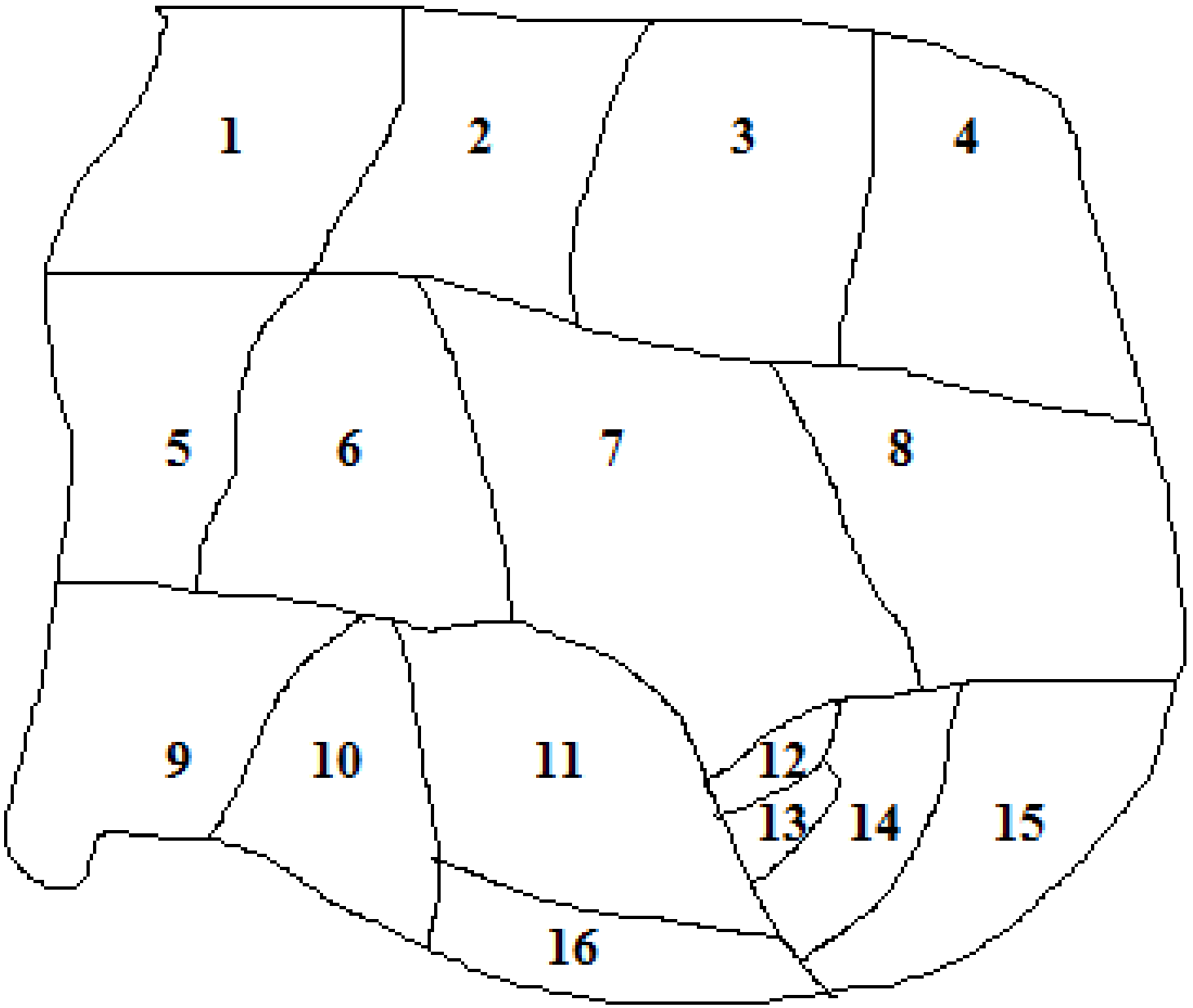
Coloração de Mapas

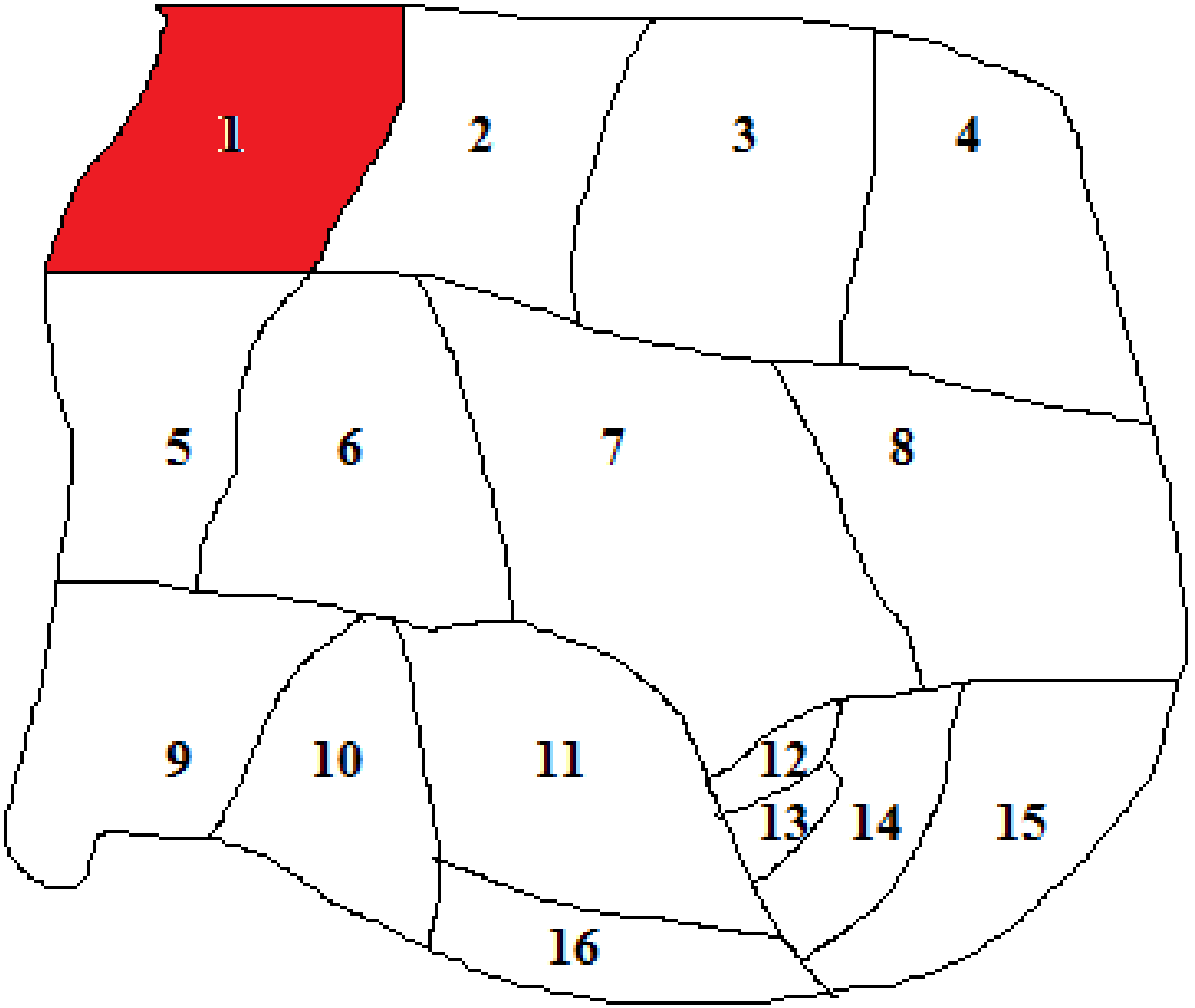
- Tentar colorir um mapa com quatro cores:
  1) Vermelho
  2) Amarelo
  3) Verde
  4) Azul



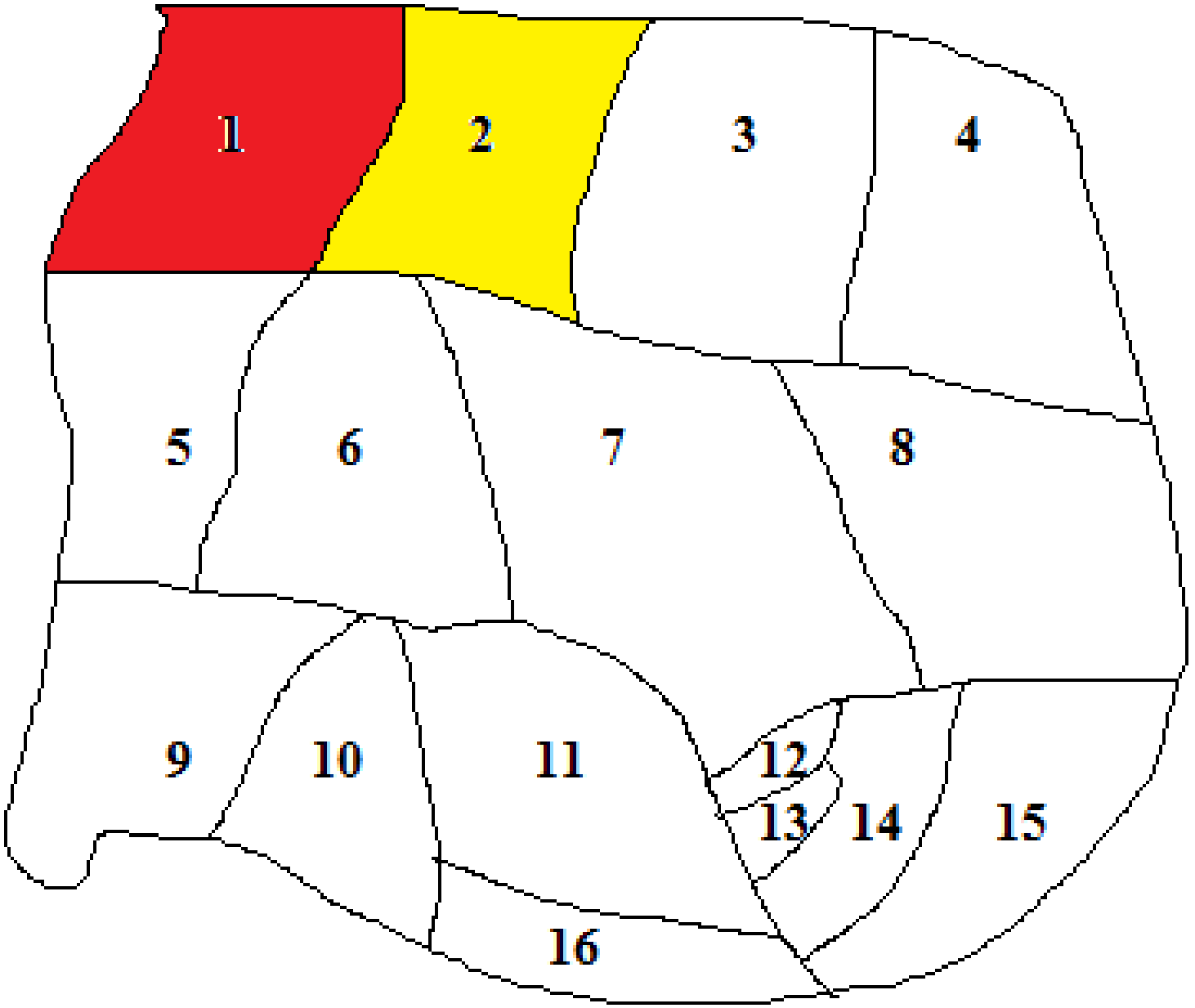

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

Recursão:

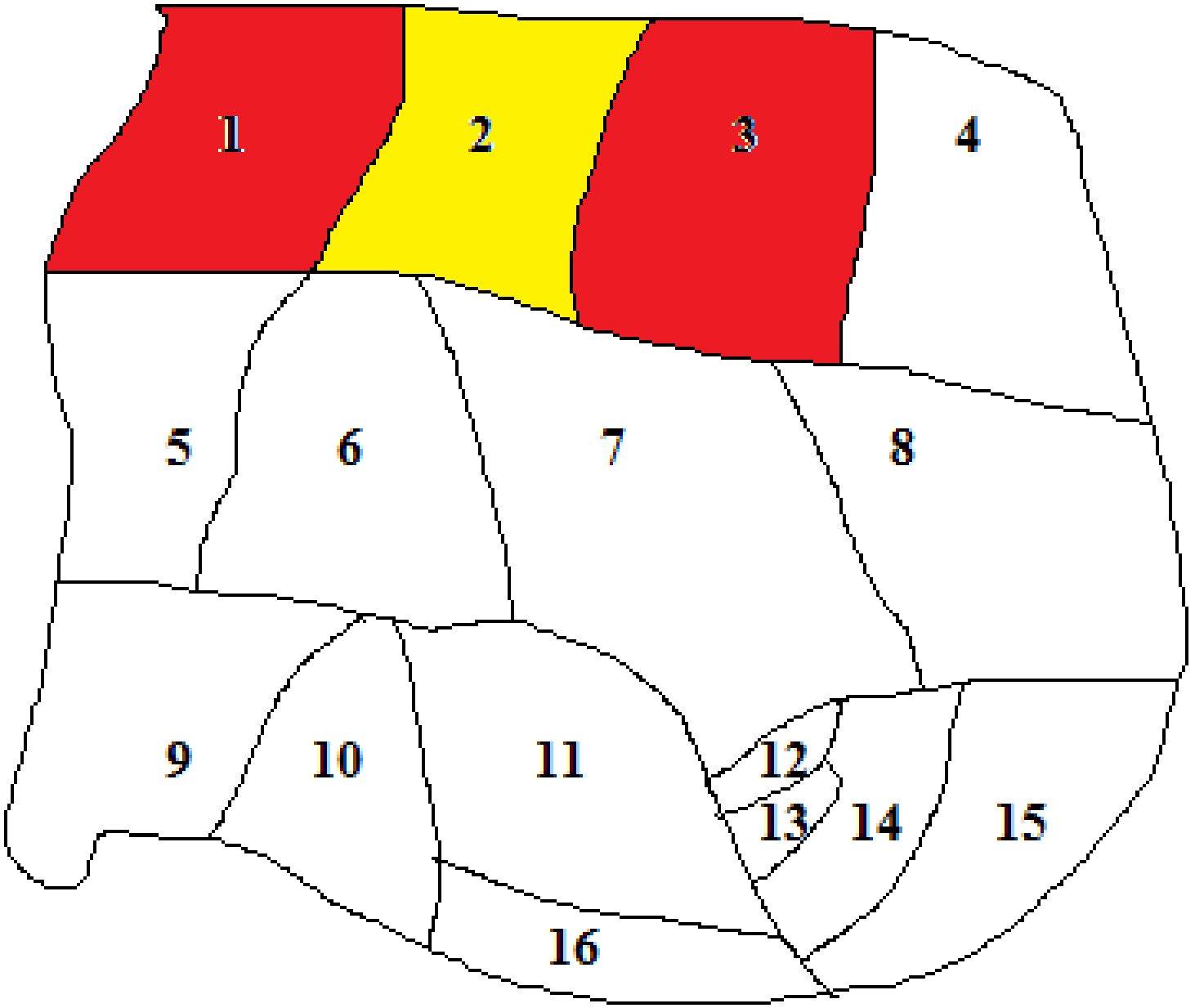




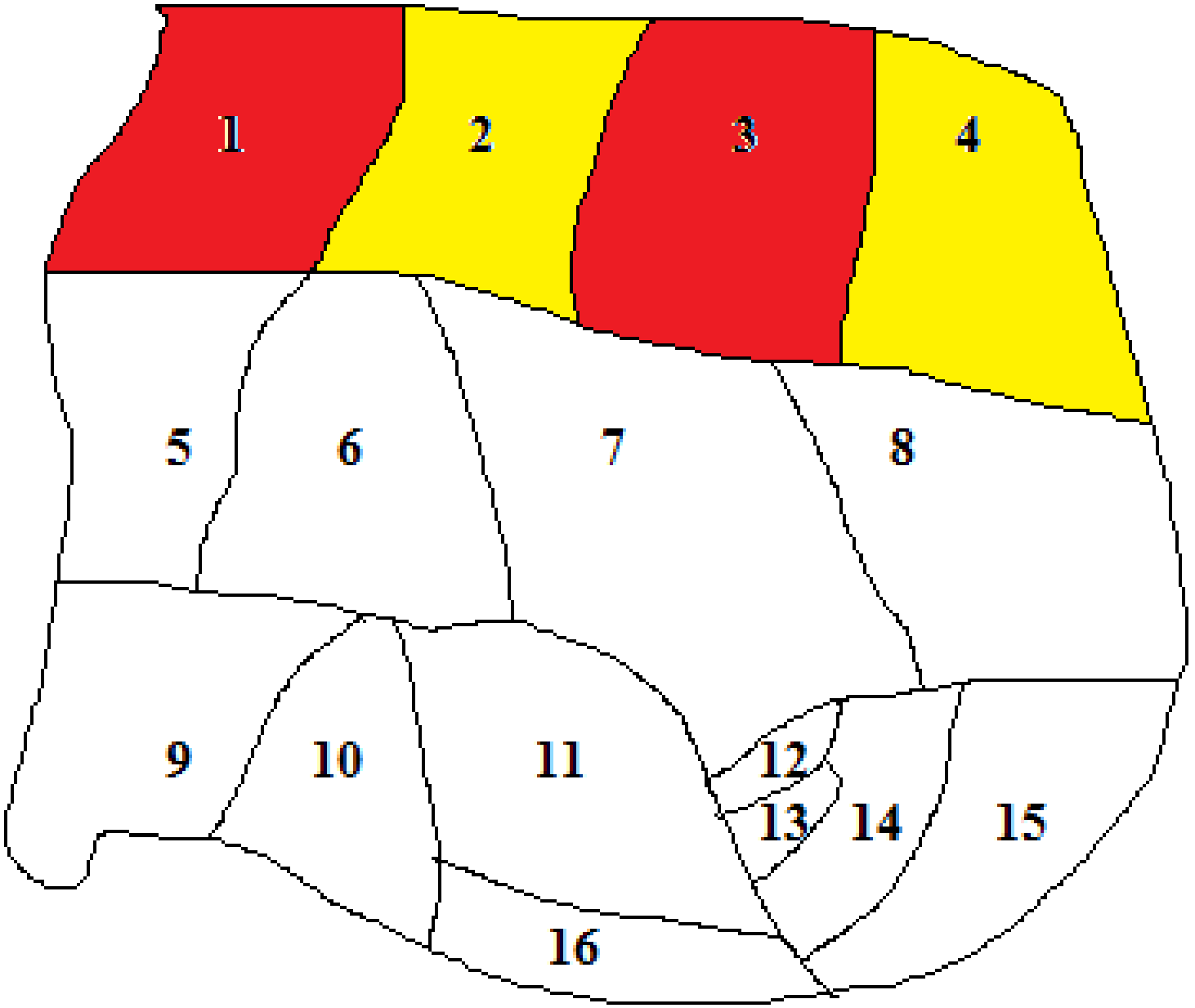
Recursão:
(1) p= 1 c=1



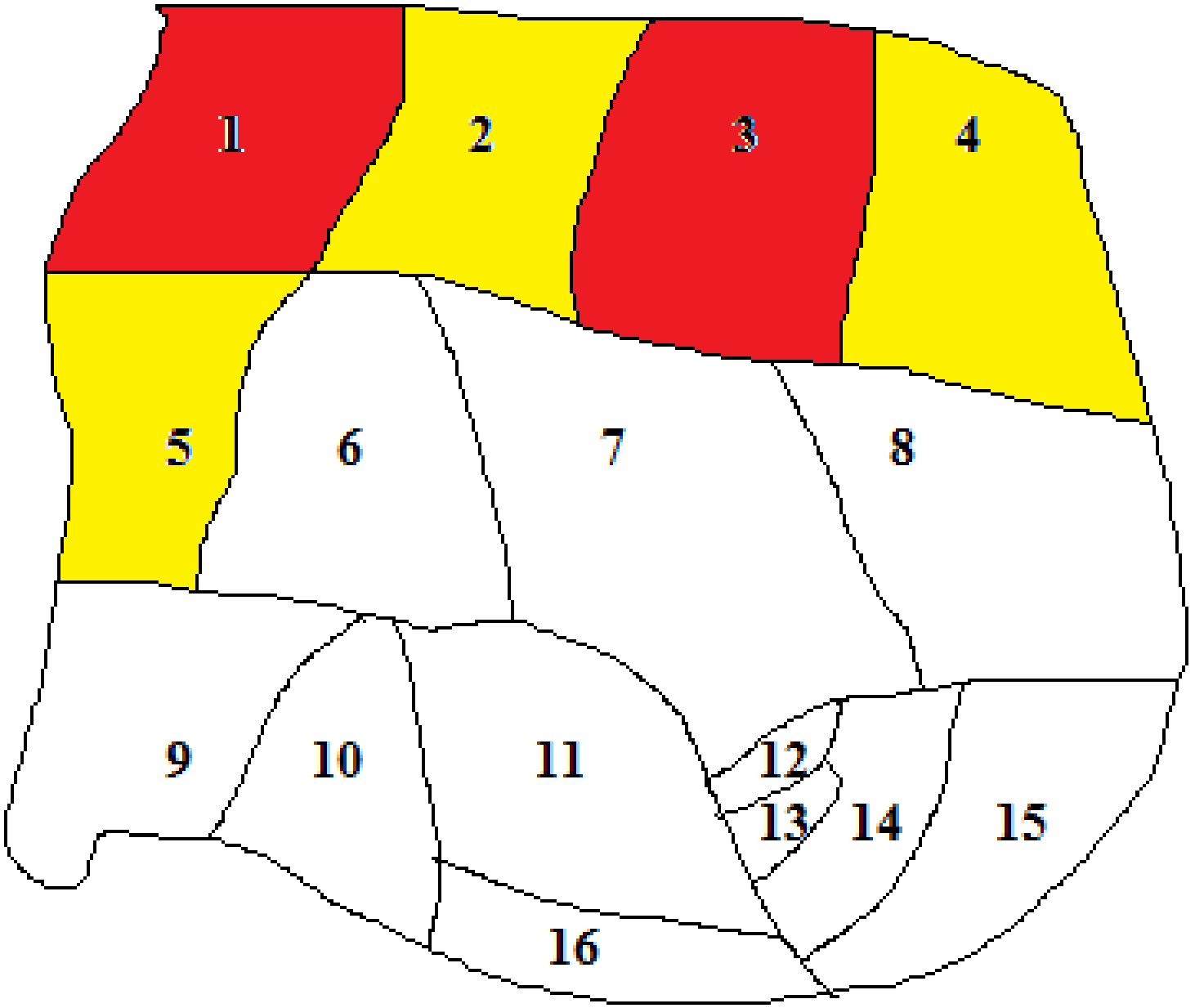
Recursão:
(1) p= 1 c=1
(2) p= 2 c=2



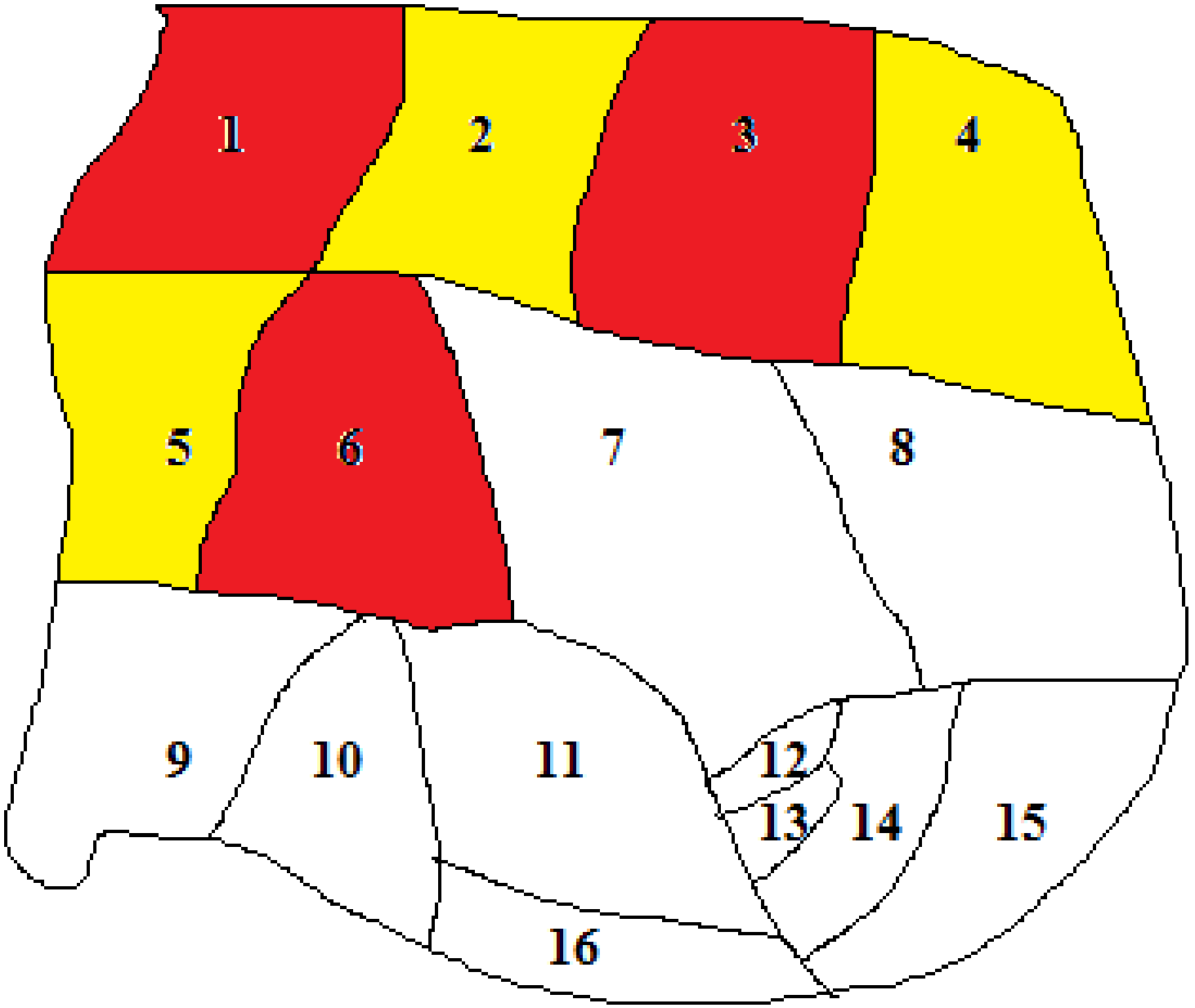
Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1



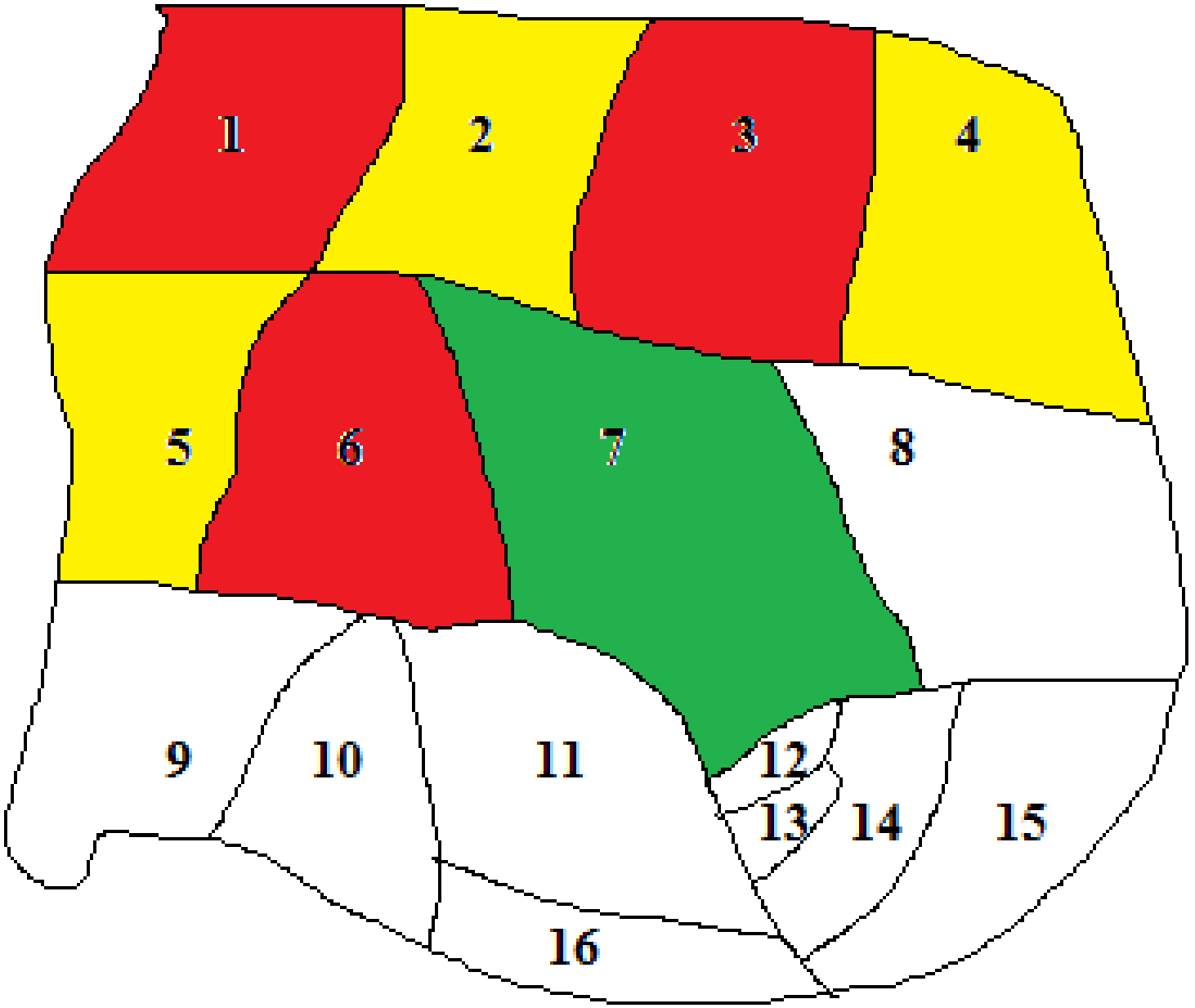
Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2



Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2

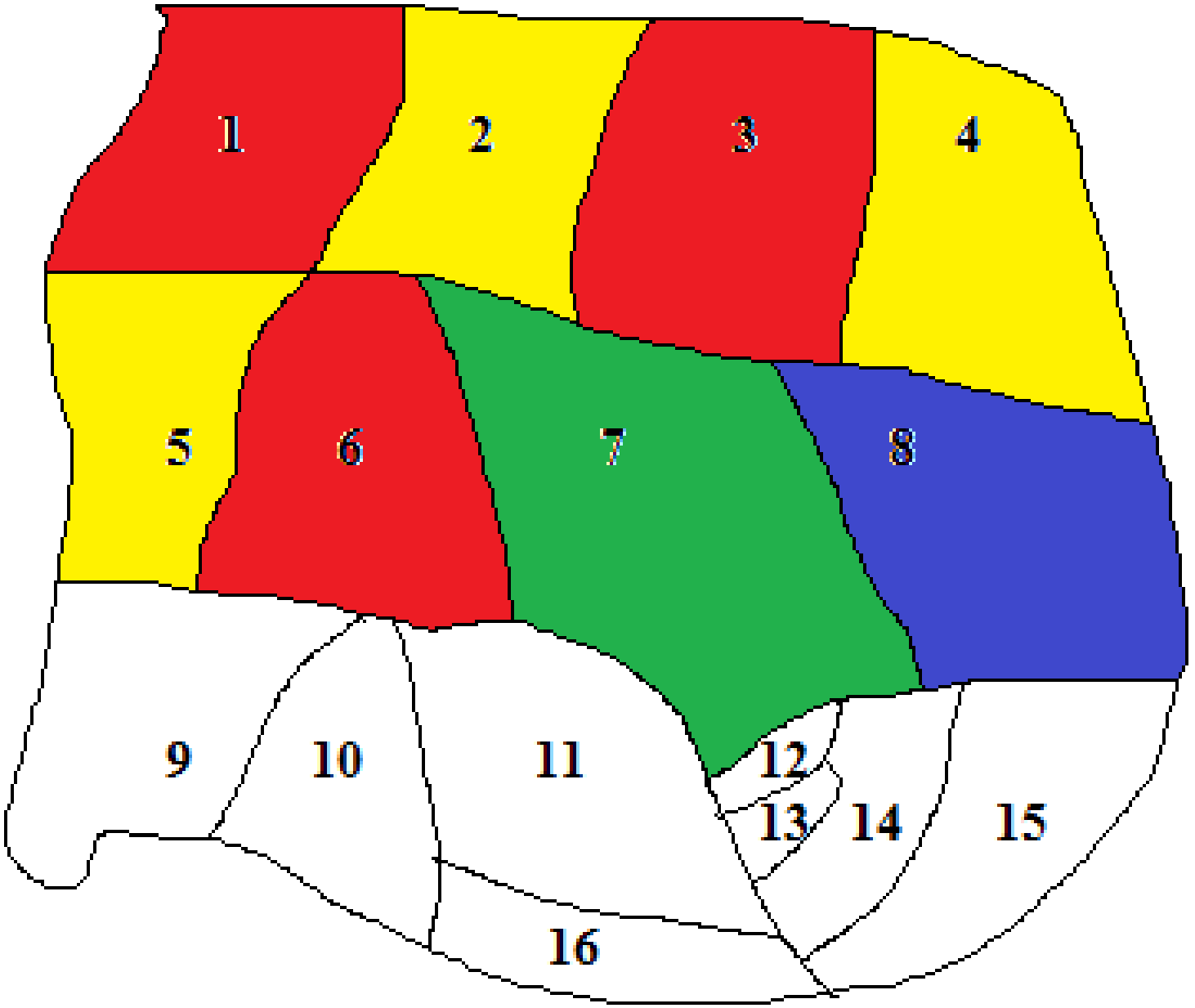


Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1

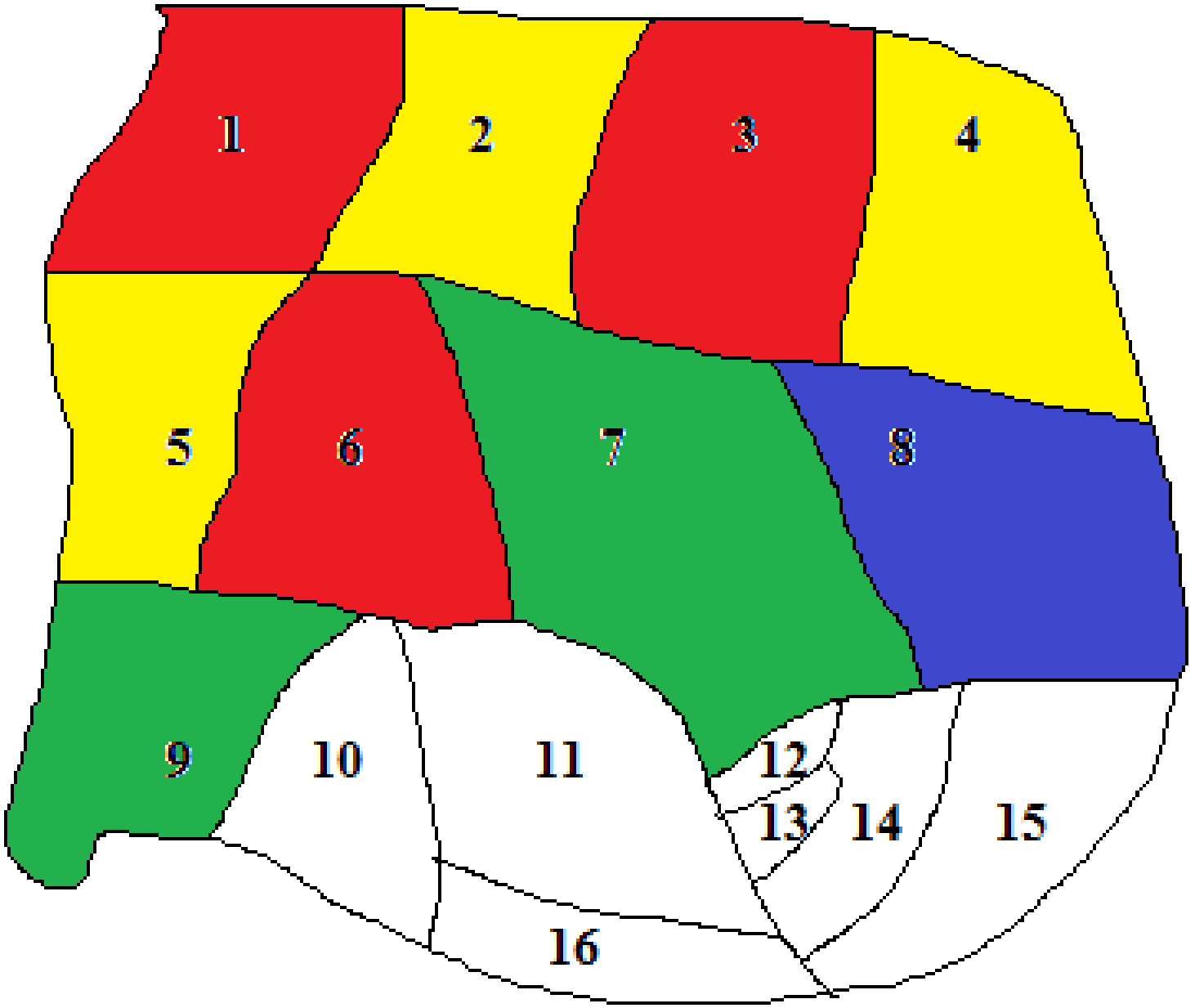


Recursão:

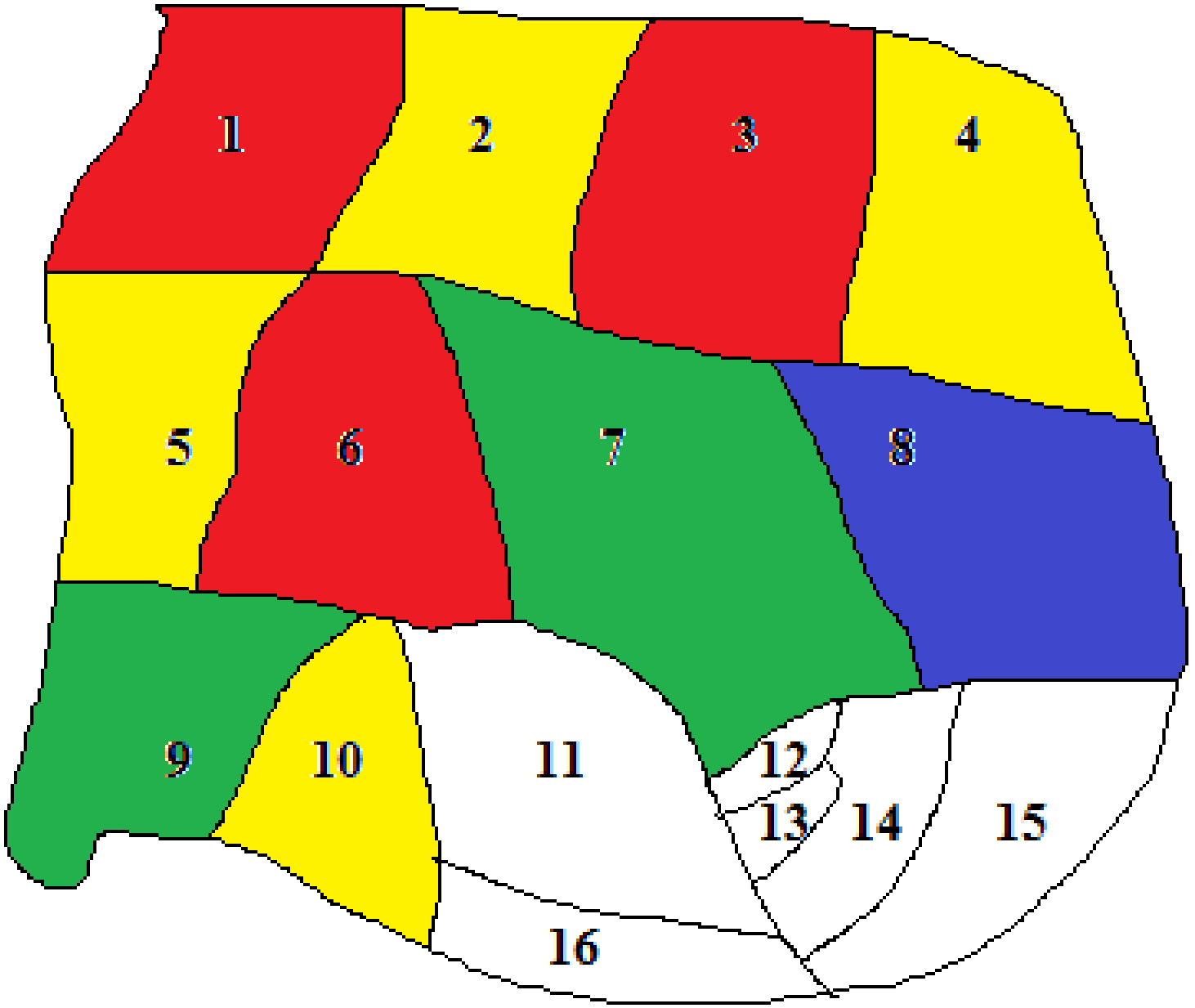
```
(  1) p=  1 c=1
(  2) p=  2 c=2
(  3) p=  3 c=1
(  4) p=  4 c=2
(  5) p=  5 c=2
(  6) p=  6 c=1
(  7) p=  7 c=3
```



Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4

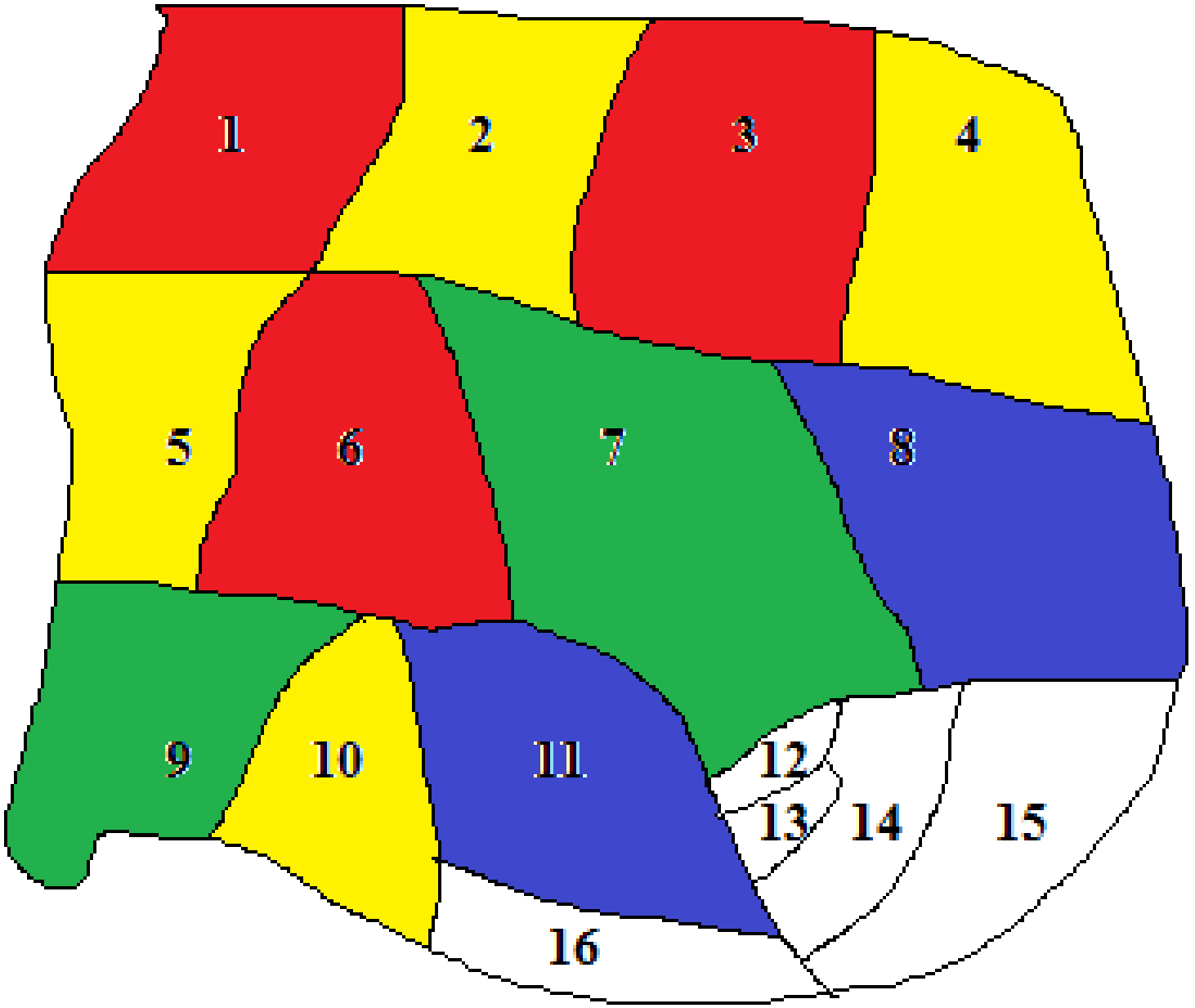


Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3

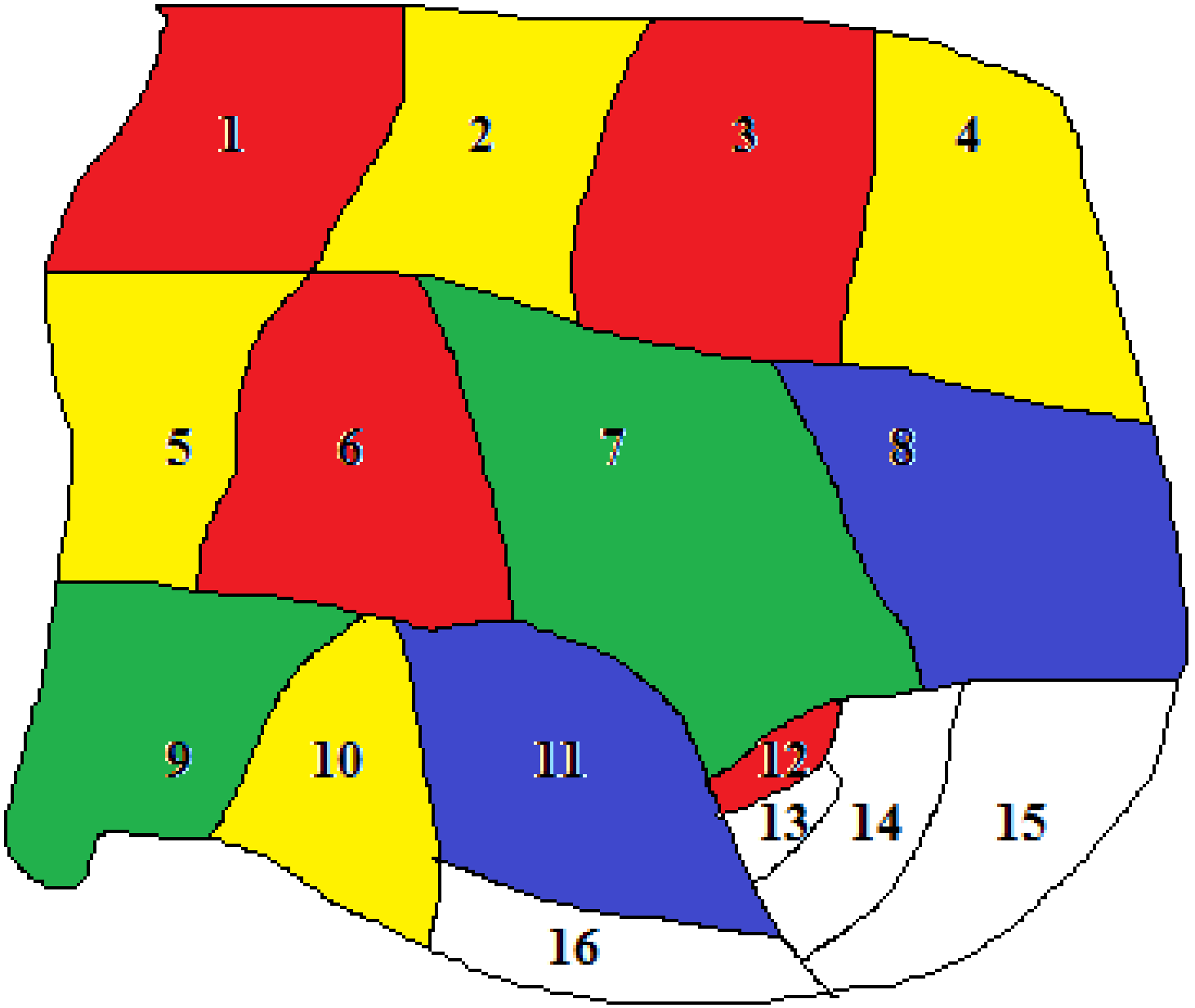


Recursão:

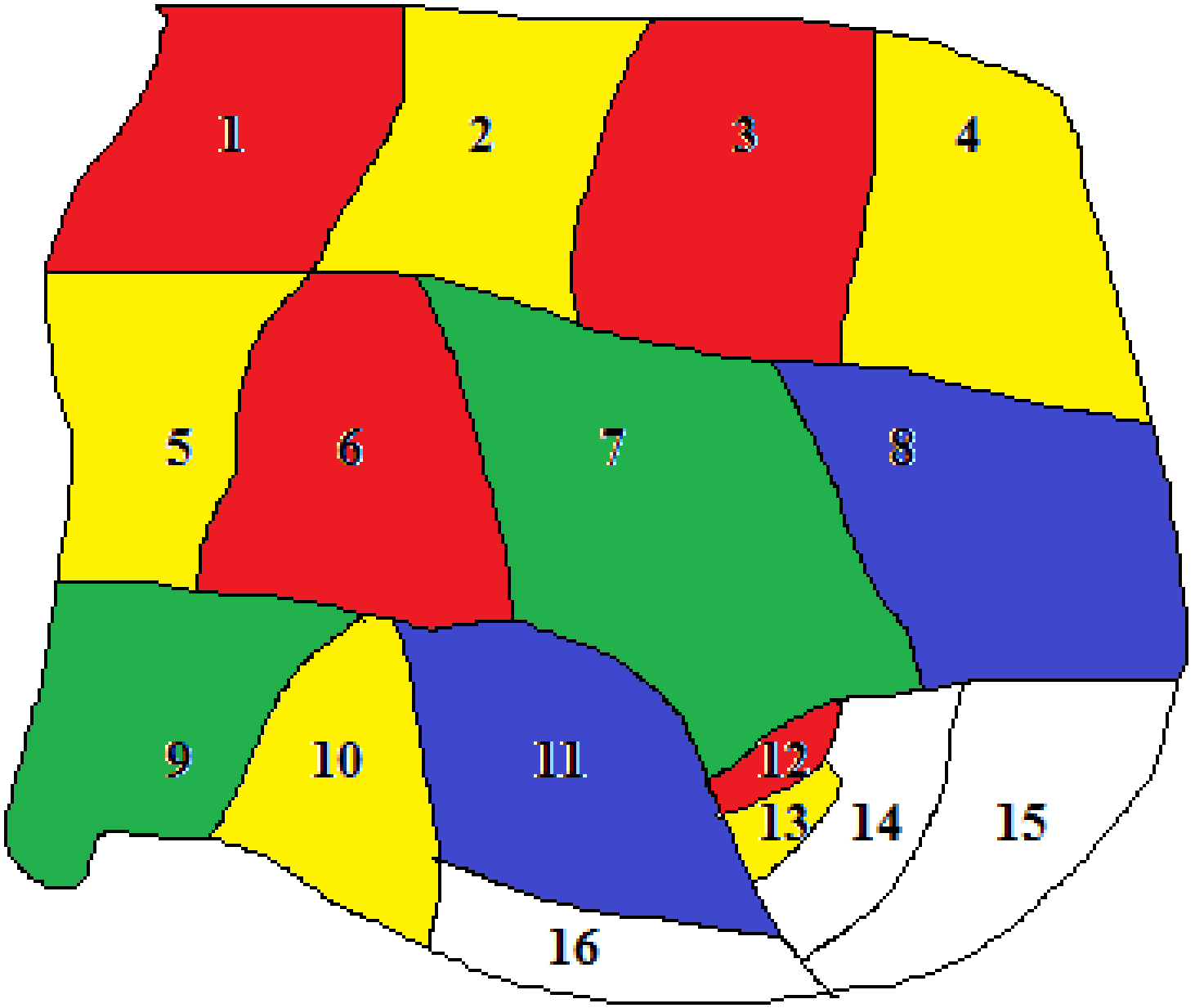
```
(  1) p=  1 c=1
(  2) p=  2 c=2
(  3) p=  3 c=1
(  4) p=  4 c=2
(  5) p=  5 c=2
(  6) p=  6 c=1
(  7) p=  7 c=3
(  8) p=  8 c=4
(  9) p=  9 c=3
( 10) p= 10 c=2
```



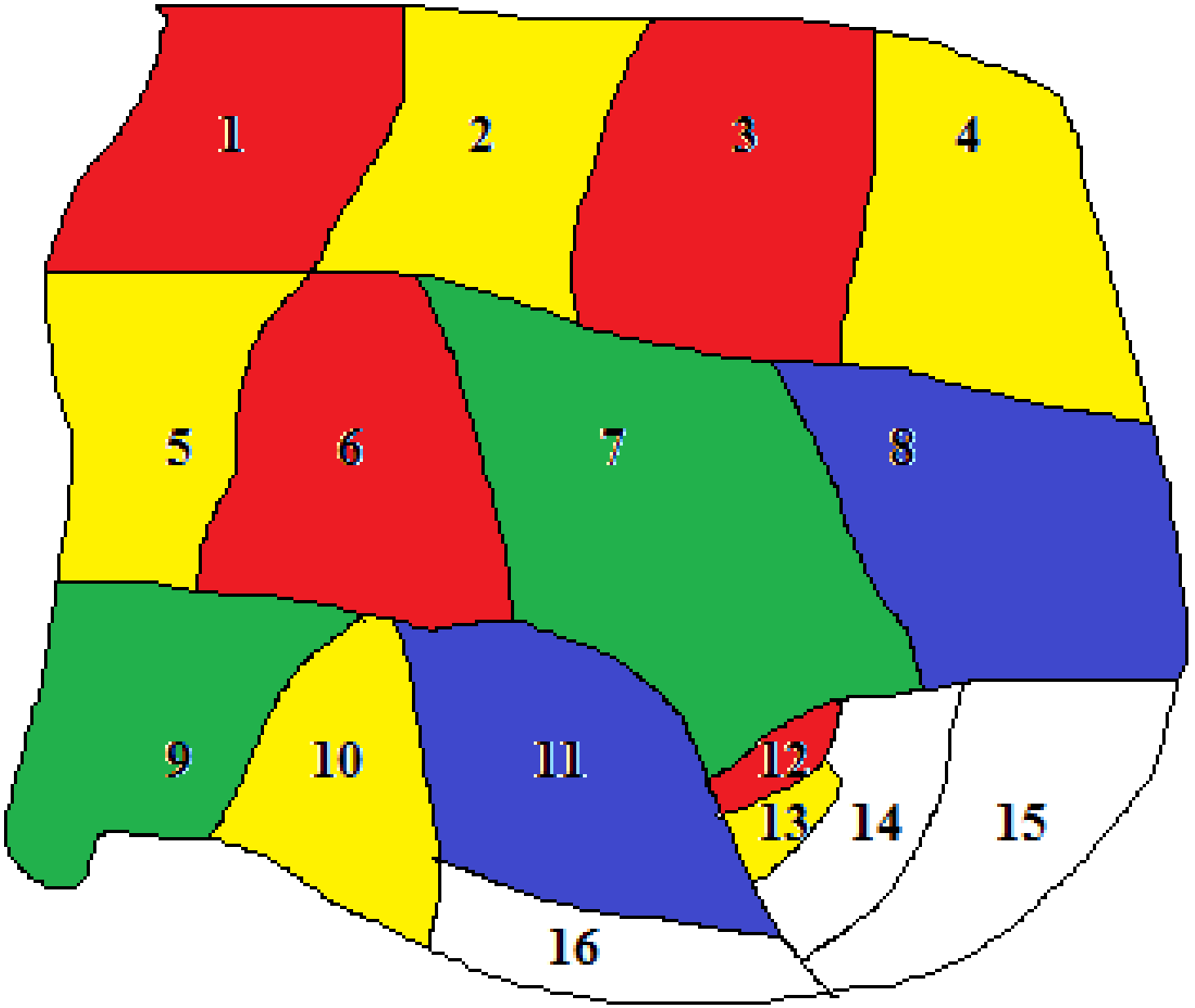
Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4



Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1

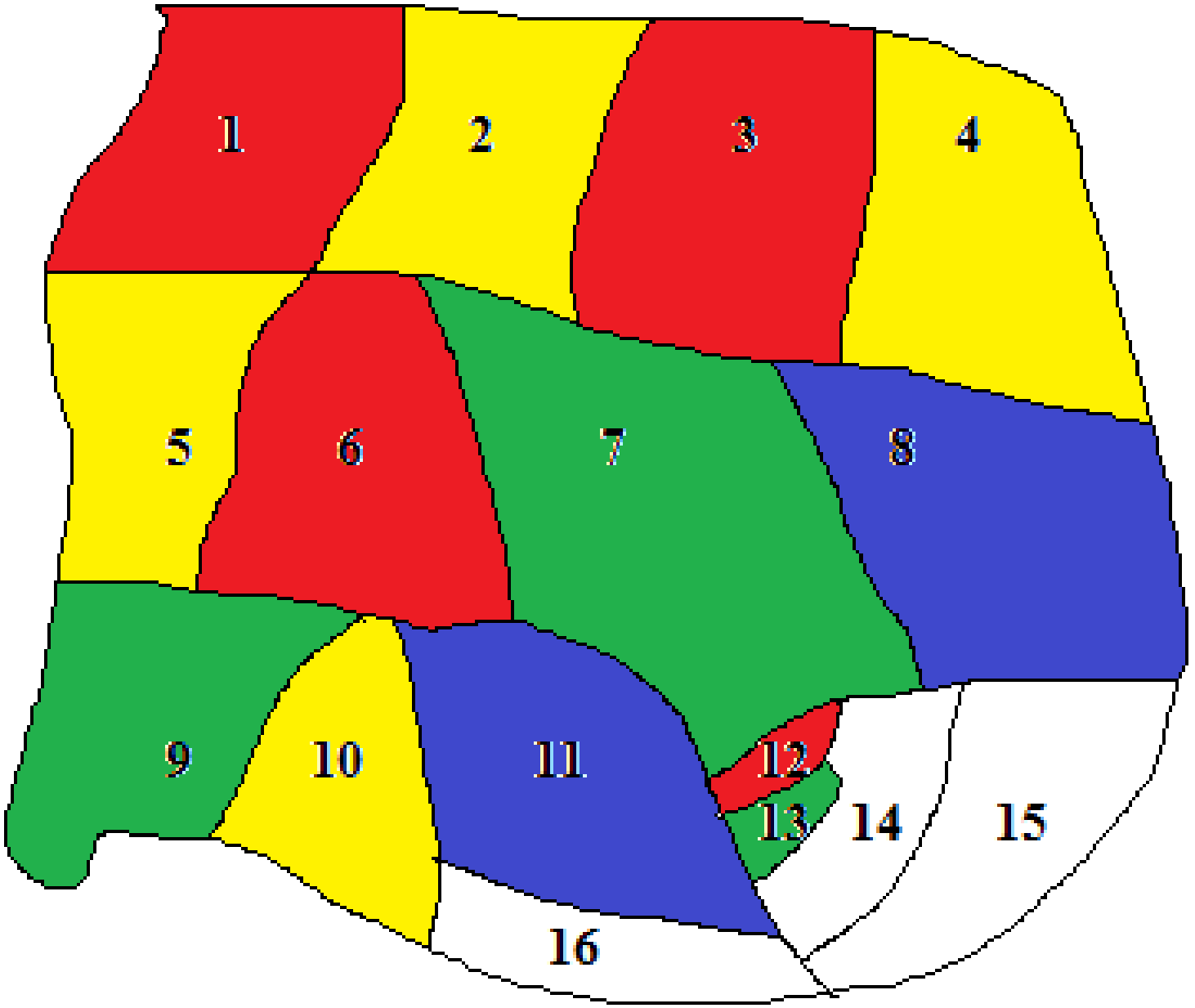


Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1
(13) p=13 c=2

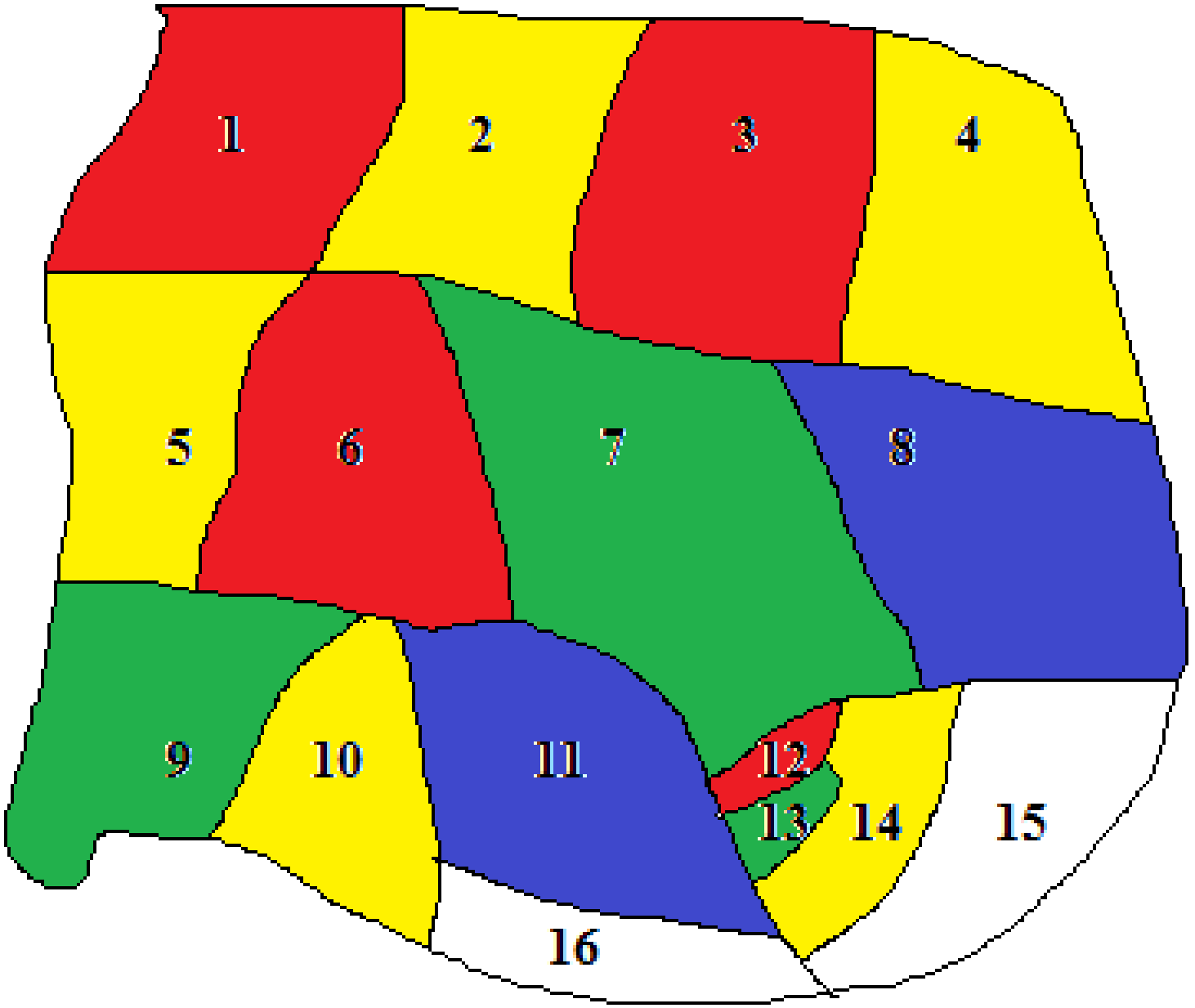


Recursão:

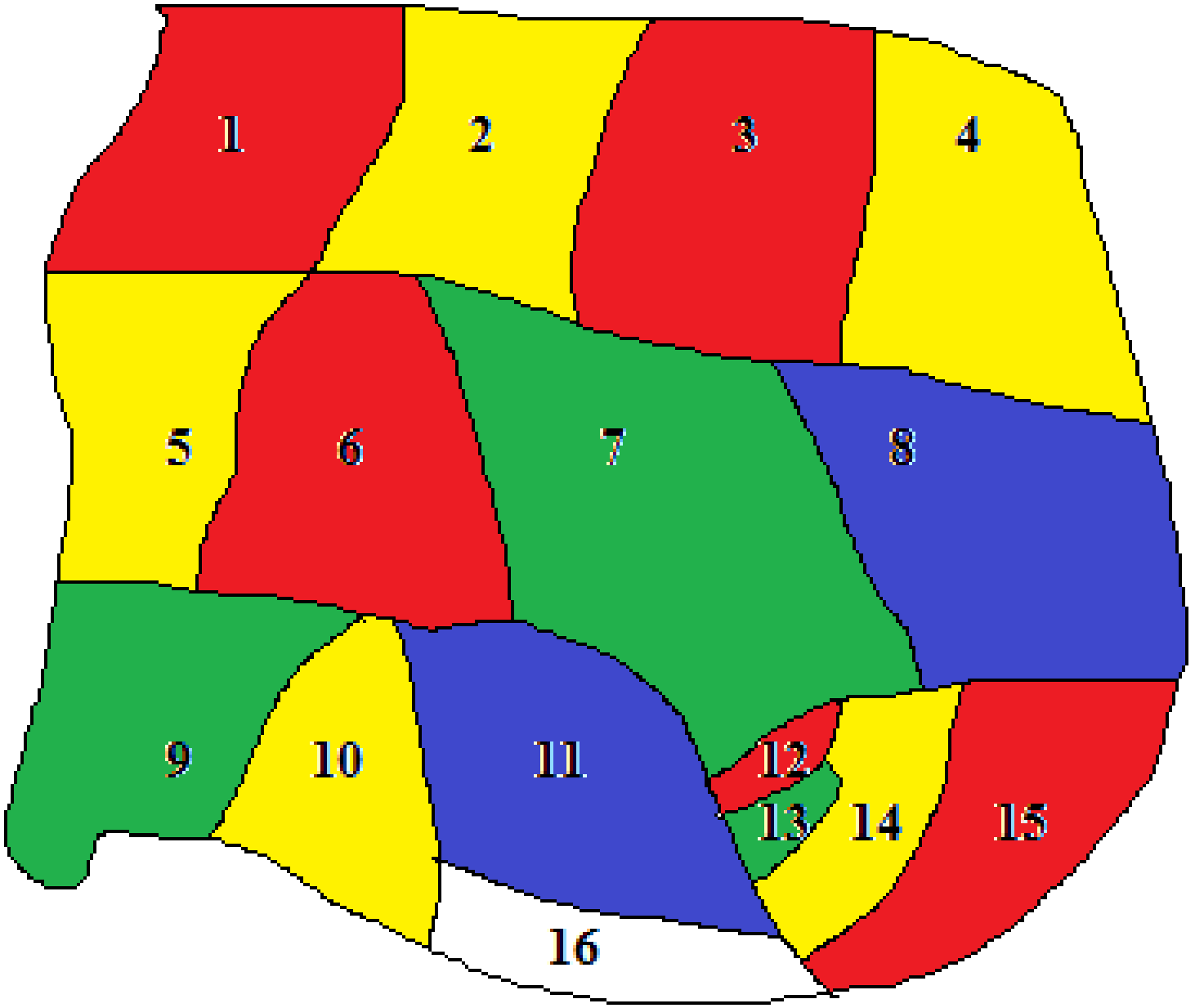
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1
(13) p=13 c=2
(14) p=14 c=?



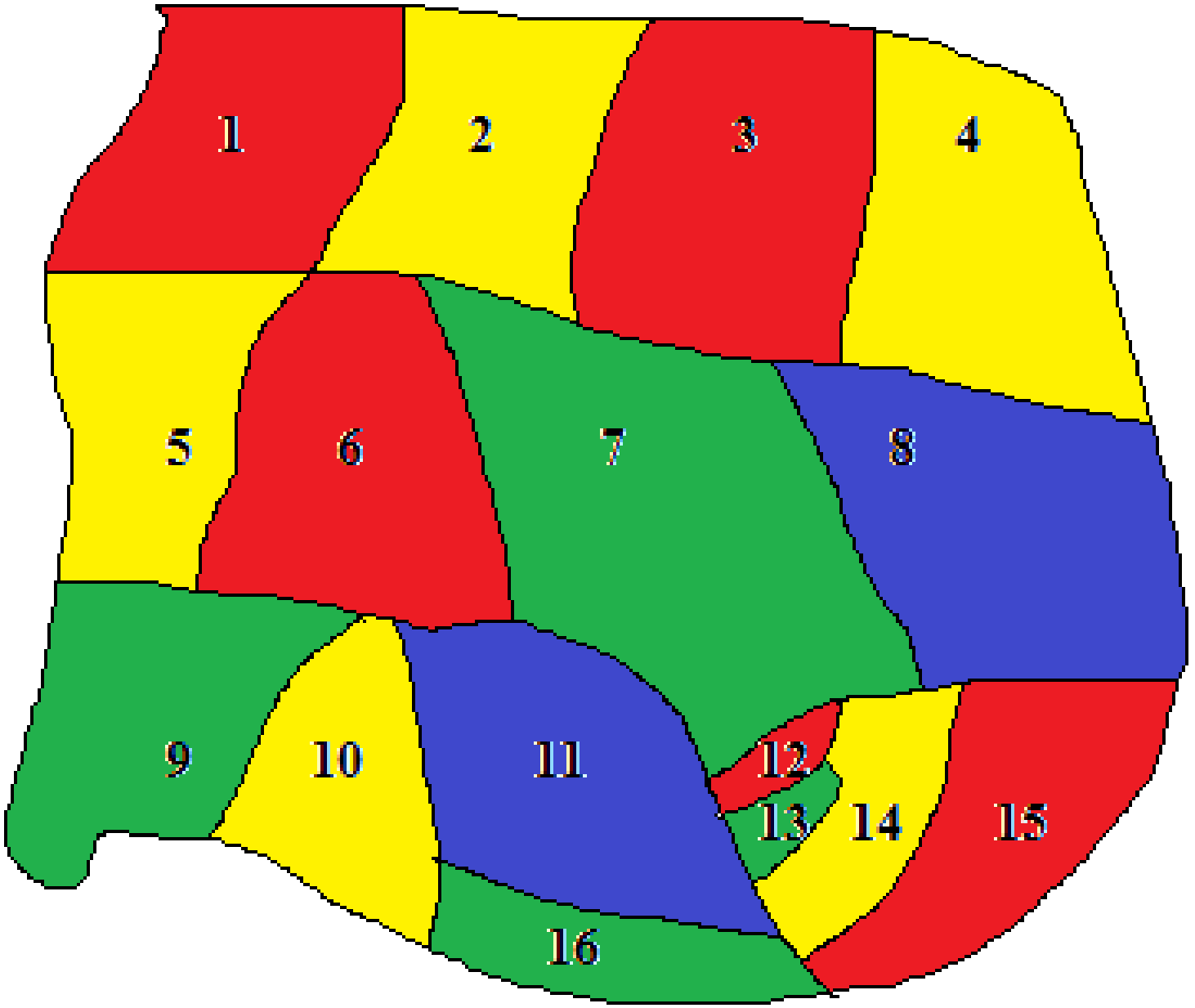
Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1
(13) p=13 **c=3**



Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1
(13) p=13 c=3
(15) p=14 c=2

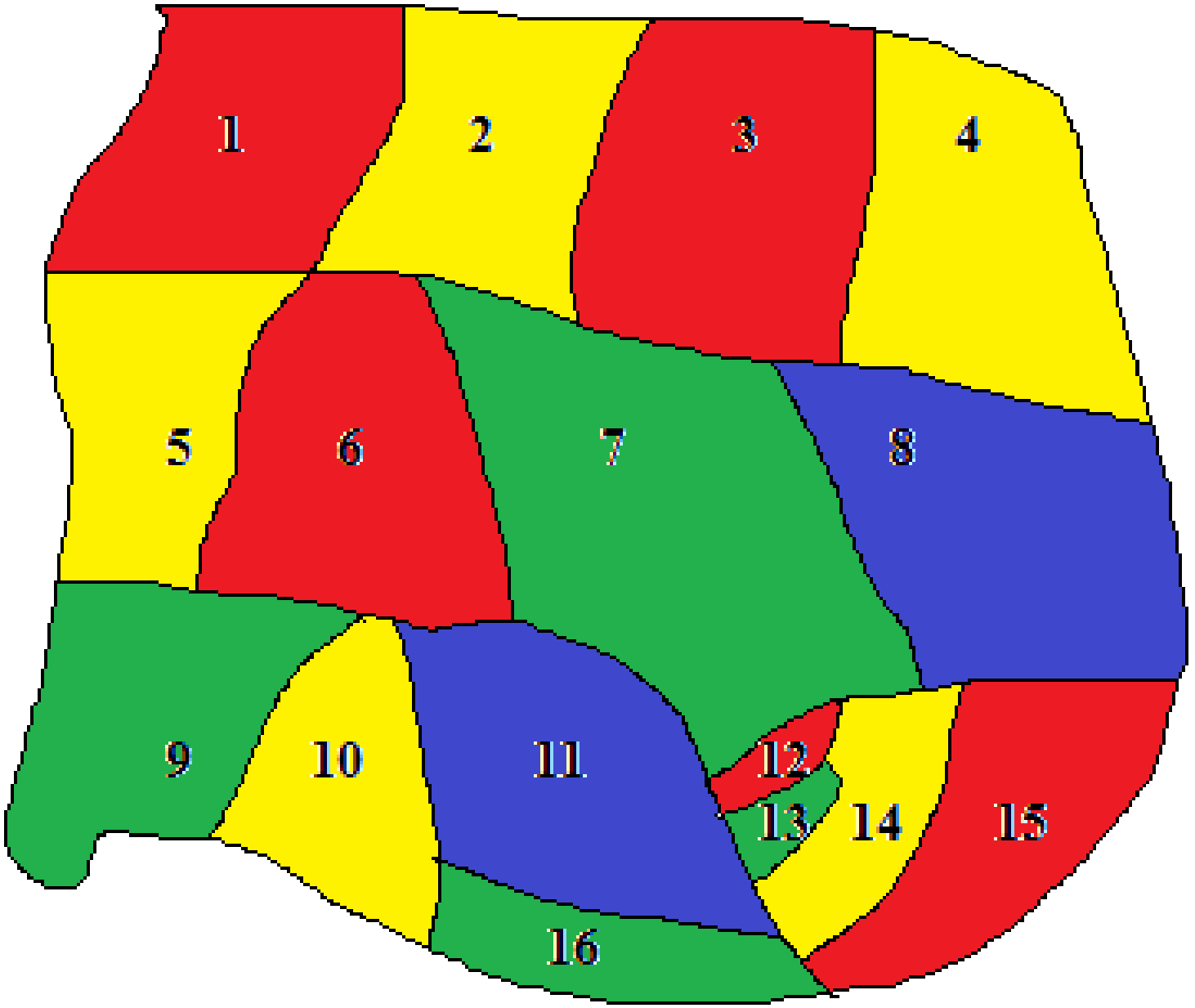


Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1
(13) p=13 c=3
(15) p=14 c=2
(16) p=15 c=1



Recursão:

```
(  1) p=  1 c=1
(  2) p=  2 c=2
(  3) p=  3 c=1
(  4) p=  4 c=2
(  5) p=  5 c=2
(  6) p=  6 c=1
(  7) p=  7 c=3
(  8) p=  8 c=4
(  9) p=  9 c=3
( 10) p=10 c=2
( 11) p=11 c=4
( 12) p=12 c=1
( 13) p=13 c=3
( 15) p=14 c=2
( 16) p=15 c=1
( 17) p=16 c=3
```



Recursão:
(1) p= 1 c=1
(2) p= 2 c=2
(3) p= 3 c=1
(4) p= 4 c=2
(5) p= 5 c=2
(6) p= 6 c=1
(7) p= 7 c=3
(8) p= 8 c=4
(9) p= 9 c=3
(10) p=10 c=2
(11) p=11 c=4
(12) p=12 c=1
(13) p=13 c=3
(15) p=14 c=2
(16) p=15 c=1
(17) p=16 c=3
(18) fim

Encontrar Melhor Solução

- Problema da Mochila Binária

```java
class Objeto{
  int peso;
  int valor;
  Objeto(int p, int v){
    valor = v;
    peso = p;
  }
}

public class MochilaBinaria {
  static int melhorSolucao;
  static void resolveMochilaBinaria(Objeto[] objetos, int atual, int pesoDisponivel,
                                                                    int valorAtual){
    if (atual >= objetos.length) {
      if (valorAtual > melhorSolucao) melhorSolucao = valorAtual;
      return;
    }
     // NAO COLOCAR OBJETO ATUAL
    resolveMochilaBinaria(objetos, atual+1, pesoDisponivel, valorAtual);
     // SE POSSIVEL, COLOCAR OBJETO ATUAL
    Objeto obj = objetos[atual];
    if (obj.peso <= pesoDisponivel){
      resolveMochilaBinaria(objetos, atual+1, pesoDisponivel-obj.peso,
                                                    valorAtual + obj.valor);
    }
  }
}
```