Cell record

Cell:

- id: Int. Unique cell identifier.
- lineage: String. The branch of the cell in the quadtree. It provides position and depth of the cell.
- envelope: Polygon. Geometric representation of the cell.

Lineage example

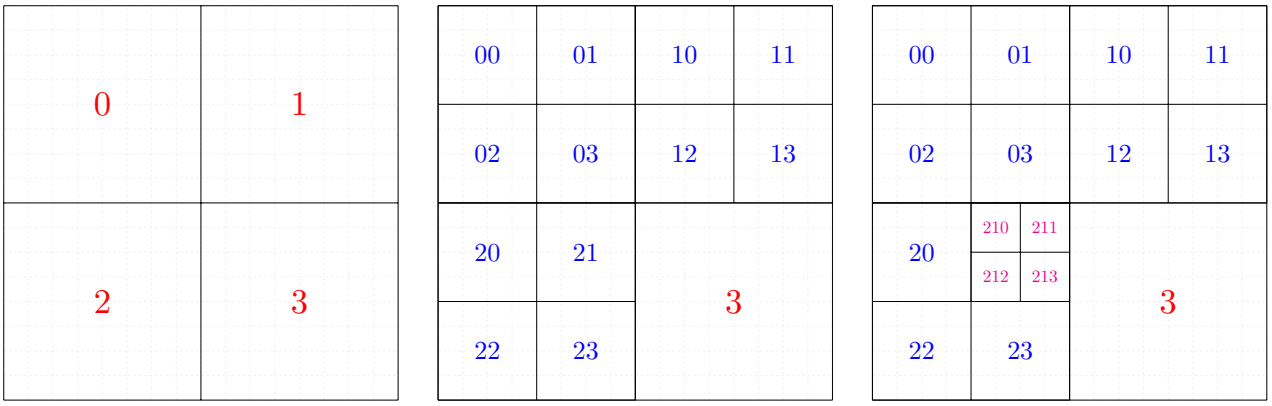

Figure 1: Lineage can provide the cell's position (string's last character) and its depth (string's length).

Algorithms

Lemma. *Four cells at the same level can not be empty. At least one of them must have edges in order to force the split.*

Proof. The GETCELLSINCORNER function will query the interior corner of a cell according to its position, that is the centroid of its cell parent. The only cells which can intersect that point are cells at the same level of the current cell or their children. If the 3 cells returned by GETCELLSINCORNER are empty, at least one of them must have a deeper level that the current cell. Following that cell guarantees that the search space will be shrank at each iteration. Eventually, the algorithm will reach the maximum level of the quadtree where all the involved cells will have the same level and, therefore, at least one of them must have edges. □

Algorithms example

Algorithm 1 getNextCellWithEdges algorithm

Require: a quadtree with cell envelopes $\mathcal{Q}$ and map of cells and their edge count $\mathcal{M}$.

```
1: function getNextCellWithEdges ( Q, M )
2:     C ← list of empty cells in M
3:     for each emptyCell in C do
4:         initialize cellList with emptyCell
5:         nextCellWithEdges ← null
6:         referenceCorner ← null
7:         done ← false
8:         while not done do
9:             c ← last cell in cellList
10:            cells, corner ← getCellsAtCorner(Q, c)        ▷ return 3 cells and the reference corner
11:            for each cell in cells do
12:                nedges ← get edge count of cell in M
13:                if nedges > 0 then
14:                    nextCellWithEdges ← cell
15:                    referenceCorner ← corner
16:                    done ← true
17:                else
18:                    add cell to cellList
19:                end if
20:            end for
21:        end while
22:        for each cell in cellList do
23:            output(cell, nextCellWithEdges, referenceCorner)
24:            remove cell from C
25:        end for
26:    end for
27: end function
```

Algorithm 2 getCellsAtCorner algorithm

Require: a quadtree with cell envelopes $\mathcal{Q}$ and a cell c.

```
1: function getCellsInCorner ( Q, c )
2:     region ← last character in c.lineage
3:     switch region do
4:         case '0'
5:             corner ← left bottom corner of c.envelope
6:         case '1'
7:             corner ← right bottom corner of c.envelope
8:         case '2'
9:             corner ← left upper corner of c.envelope
10:        case '3'
11:            corner ← right upper corner of c.envelope
12:    cells ← cells which intersect corner in Q
13:    cells ← cells − c          ▷ Remove the current cell from the intersected cells
14:    cells ← sort cells on basis of their depth          ▷ using cell.lineage
15:    return (cells, corner)
16: end function
```