1. The Grid TreeMap layout algorithm for set grids

In this document, we demonstrate the algorithm steps using examples of three sets—Set1, Set2, and Set3—and their exclusive intersections.

1.1 Group the elements based on exclusive intersections:
All of the elements in each set grid are organized and assigned into different subsets based on their exclusive intersections among the sets on the screen. We assume that three sets: Set1, Set2, and Set3 are added onto the Main view by the user.

- Three sets and their set grids:

<table>
<thead>
<tr>
<th>Set Name</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set1</td>
<td>A, B, D, F, I, K, N, P, Q</td>
</tr>
<tr>
<td>Set2</td>
<td>B, C, D, E, F, H, L, N, O</td>
</tr>
<tr>
<td>Set3</td>
<td>E, F, G, J, M, N, O, P, Q</td>
</tr>
</tbody>
</table>

- Decide setIndex for the sets in the order they are added to the Main view (refer to setIndex in Algorithm1 and 2):

<table>
<thead>
<tr>
<th>Set Name</th>
<th>setIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set1</td>
<td>0</td>
</tr>
<tr>
<td>Set2</td>
<td>1</td>
</tr>
<tr>
<td>Set3</td>
<td>2</td>
</tr>
</tbody>
</table>
1.2 Compute all the exclusive Intersections of the Sets in Main View, assuming Set1, Set2, Set3 are added to the Main View

First, the algorithm identifies the exclusive Intersections and required parameters to construct binary trees for the three sets present in the Main View.

<table>
<thead>
<tr>
<th>Exclusive Intersections</th>
<th>Associated Sets</th>
<th>IntersectionLabel</th>
<th>Degree</th>
<th>Cardinality</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only Set1</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Only Set2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Only Set3</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Only Set1 &amp; Set2</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Only Set1 &amp; Set3</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Only Set2 &amp; Set3</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Only Set1 &amp; Set2 &amp; Set3</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
1.3 Construct a tree structure for set intersections:
Once all of the exclusive intersections have been identified and grouped according to their associated elements in each set, these element subsets could be arranged into three different types of tree structures: (1) set memberships of the intersecting elements (common sets to which elements belong in the intersections), (2) degrees of the intersecting elements (the number of sets to which the intersecting elements belong), and (3) cardinalities of the intersecting elements (the number of elements in the intersections). These three tree structures generate different layouts of subdivisions within the set grid. The user can select different subdivision layouts by using one of the following three types of trees. (Note, however, that we used the same algorithms for both degrees and cardinalities of intersections.)

1.3.1 Generate a binary tree based on set memberships of the intersecting elements (common sets to which elements belong in the intersections):
To divide elements in the set grid, we first generate a binary tree based on the set memberships of the intersecting elements. The following tree $T$ is based on set memberships for Set1 with its exclusive intersections with Set2 and Set3.
To generate a binary tree structure based on the set memberships of the intersecting elements, we create a strict hierarchy of exclusive intersections with respect to their associated sets. Algorithm 1 generates the above tree structure based on the set memberships of the elements in a recursive way.

```
Algorithm 1 Generate a Tree Structure based on Set Memberships
1:  S ← all sets in the Main view
2:  for each s ∈ S do
3:      s.setIndex ← an ordered index value for sets in the Main view
4:  end for
5:  for each s ∈ S do
6:      E ← all the Nonempty exclusive intersections of s
7:      //IntersectionLabel: denotes if a set is present in the intersection
8:      for each e ∈ E do
9:          if a set r ∈ S is a part of e then
10:             e.IntersectionLabel[r.setIndex] ← 1
11:          else
12:             e.IntersectionLabel[r.setIndex] ← 0
13:      end if
14:  end for
15:  //Node: Node structure containing the exclusive intersections
16:  //arrayIndex: Index of IntersectionLabel to compare;
17:  //setIndex: set Index of current set;
18:  function GENERATETREEBYSET(Node, arrayIndex, setIndex)
19:      // Skip the current set’s array index
20:      if arrayIndex = setIndex then
21:          return GENERATETREEBYSET(Node, arrayIndex + 1, setIndex)
22:  end if
23:  // Recursion termination condition
24:  if Node.length = 1 then
25:      return Node
26:  end if
27:  // Initializing Tree and Child arrays
28:  Tree ← new []
29:  Child ← new []
30:  Append Node[0] into Child
31:  for each e ∈ Node till Node.length - 1 do
32:      // Comparing the intersections using IntersectLabel
33:      if e.IntersectionLabel[arrayIndex] = e.next().IntersectLabel[arrayIndex] then
34:          Append e.next() into Child
35:      else
36:          Append GENERATETREEBYSET(child, arrayIndex + 1, setIndex) into Tree
37:          Child ← new []
38:          Append e.next() into Child
39:  end if
40:  end for
41:  Append GENERATETREEBYSET(child, arrayIndex + 1, setIndex) into Tree
42:  return Tree
43: end function
```
Arrange and organize element glyphs in the set grid based on the tree structures (set memberships of the intersecting elements): After the tree structures for the sets on screen are formed, they can then be visualized and converted to the treemap layout in each set grid. We use the Grid TreeMap algorithm (GTM) [1], which recursively partitions and allocates a sequence of element glyphs into multiple subdivisions within a set grid; this process is based on the tree $T$ built from the previous step (S2).

As illustrated in the following examples, GridSet divides and arranges element glyphs in different subdivisions of the set grid according to the tree $T$:

**RootNode:**

All the elements contained in the root node are first filled in the set grid vertically—either from top to bottom or from bottom to top.

**Level 1:**

At Level 1 of $T$, the algorithm allocates elements in the left child to the vertical subdivision on the top (yellow), and elements in the right child to the bottom (light blue) vertical subdivision by scanning elements in rows.

**Level 2:**

At level 2 (the last level) of $T$, the algorithm divides and lays out subdivisions recursively within the two subdivisions (yellow and light blue subdivisions) created during the previous step, filling the grid slots in each subdivision with associated elements, either vertically or horizontally.
1.3.2 Generate a tree based on degrees of the intersecting elements (the number of sets to which the intersecting elements belong)

GridSet technique supports two alternative layouts in the set grid. Specifically, the algorithm generates a sorted binary tree based on either the degree or cardinality of intersections. The exclusive intersections are sorted by the intersection degree or cardinality, after which the median of the degree (or cardinality) is then computed and utilized. Any subset with degrees (or cardinalities) that are less than the median is then added to the left child, while those that are more than the median are added to the right child.
Algorithm 2 provides details for generating a tree structure based on either degree or cardinality of sets.

```plaintext
Algorithm 2 Generate a Tree Structure based on Degree or Cardinality
1: //Node structures to represent child nodes
2: Initialize Tree, leftNode, rightNode as empty arrays
3: S ← all sets in the Main view
4: for each s ∈ S do
5:      E ← all the exclusive intersections of set s
6:      Sort E in increasing order of Degree/Cardinality
7:      s.tree ← GENERATE_TREE_BY_DEG(E)
8: end for
9: function GENERATE_TREE_BY_DEG(intersections)
10:      if intersection.length = 1 then
11:          return intersections
12:      end if
13:      if all of i ∈ intersections have the same Degree/Cardinality then
14:          return intersections
15:      end if
16:      M ← a median of the Degrees/Cardinalities of intersections
17:      for each i ∈ intersections do
18:          if Degree/Cardinality of i < M then
19:              Append i into leftNode
20:          else
21:              Append i into rightNode
22:      end if
23: end for
24: Append GENERATE_TREE_BY_DEG(leftNode) into Tree
25: Append GENERATE_TREE_BY_DEG(rightNode) into Tree
26: return Tree
27: end function
```
Arrange and group element glyphs in the set grid using the tree structures based on the degrees of the intersecting elements (arrows indicate scanning direction in each set division):

**Root Node:**

```
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Q</td>
<td>F</td>
</tr>
<tr>
<td>K</td>
<td>P</td>
<td>N</td>
</tr>
</tbody>
</table>
```

**Level 1:**

```
<table>
<thead>
<tr>
<th>A</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Q</td>
</tr>
<tr>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
</tr>
</tbody>
</table>
```

**Level 2:**

```
<table>
<thead>
<tr>
<th>A</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>I</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>K</td>
<td>F</td>
</tr>
</tbody>
</table>
```

**Level 3:**

```
<table>
<thead>
<tr>
<th>A</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q</td>
<td>I</td>
</tr>
<tr>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>K</td>
<td>F</td>
</tr>
</tbody>
</table>
```
1.3.3 Generate a tree based on the set cardinalities of the intersecting elements (the number of elements in the intersections)

A tree based on the degrees of the set cardinalities of intersecting elements
Similarly, arrange and group element glyphs in the set grid using the tree structures based on set cardinality (arrows indicate scanning direction in each set division):

**RootNode:**

```
Set1
P  N  A
Q  F  I
B  D  K
```

**Level 1:**

```
Set1
P  Q  B
N  F  D
A  I  K
```

**Level 2:**

```
Set1
P  D  F
Q  B  N
A  I  K
```
1.4 Comparison of the final layouts generated with the three different trees:
A color represents the common subdivisions and subsets of elements across three sets.

Set Membership

Set Degree

Set Cardinality

Reference