GridSet: Visualizing Individual Elements and Attributes for Analysis of Set-Typed Data (Supplementary Material 1—the Layout Algorithm Examples)

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:**

Set Name	Elements
Set1	A, B, D, F, I, K, N, P, Q
Set2	B, C, D, E, F, H, L, N, O
Set3	E, F, G, J, M, N, O, P, Q

Set1		
A	B	D
F	I	K
N	P	Q

Set2		
B	C	D
E	F	H
L	N	O

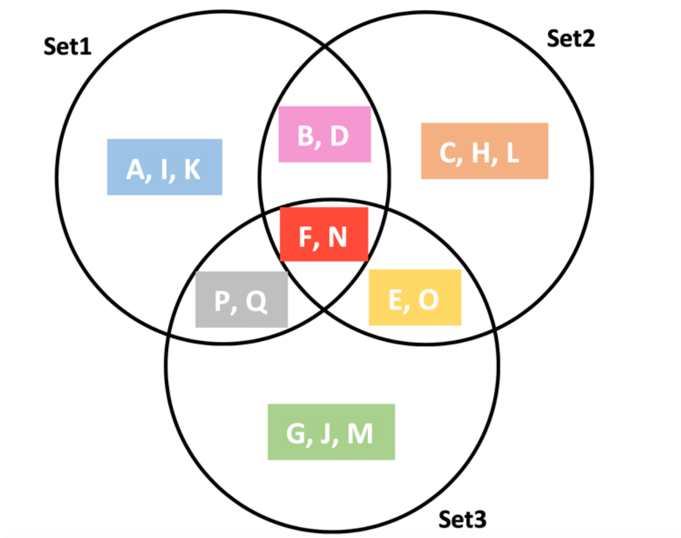
Set3		
E	F	G
J	M	N
O	P	Q

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

Set Name	setIndex
Set1	0
Set2	1
Set3	2

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.



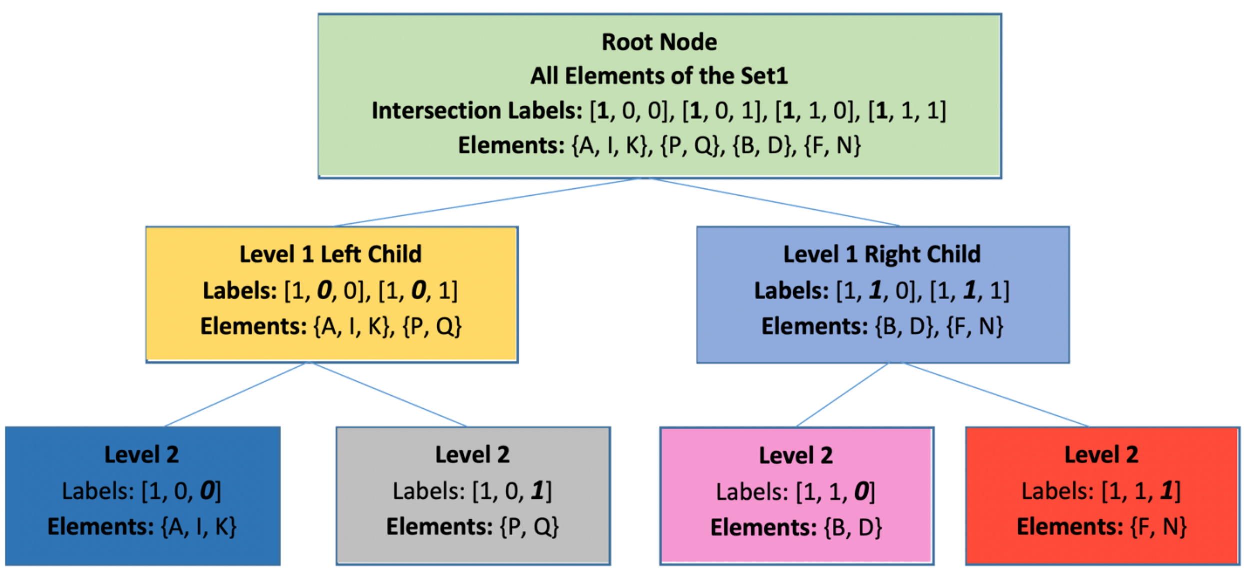
Exclusive Intersections	Associated Sets			IntersectionLabel	Degree	Cardinality	Elements
	0	1	2				
Only Set1	Yes	No	No	[1, 0, 0]	1	3	{A, I, K}
Only Set2	No	Yes	No	[0, 1, 0]	1	3	{C, H, L}
Only Set3	No	No	Yes	[0, 0, 1]	1	3	{G, J, M}
Only Set1 & Set2	Yes	Yes	No	[1, 1, 0]	2	2	{B, D}
Only Set1 & Set3	Yes	No	Yes	[1, 0, 1]	2	2	{P, Q}
Only Set2 & Set3	No	Yes	Yes	[0, 1, 1]	2	2	{E,O}
Only Set1 & Set2 & Set3	Yes	Yes	Yes	[1, 1, 1]	3	2	{F,N}

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



A binary tree based on the set memberships of the intersecting elements

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

```
S ← all sets in the Main view
for each s ∈ S do
    s.setIndex ← an ordered index value for sets in the Main view
end for
for each s ∈ S do
    E ← all the Nonempty exclusive intersections of s
    //IntersectionLabel: denotes if a set is present in the intersection
    for each e ∈ E do
        if a set r ∈ S is a part of e then
            e.IntersectionLabel[r.setIndex] ← 1
        else
            e.IntersectionLabel[r.setIndex] ← 0
        end if
    end for
    Sort E in ascending order considering IntersectionLabels as S.length bit binary number
    //RootNode: Node structure representing the root of the tree
    RootNode ← E
    s.tree ← GenerateTreeBySet(RootNode, 0, s.setIndex)
end for
// Node: Node structure containing the exclusive intersections
// arrayIndex: Index of IntersectionLabel to compare;
// setIndex: set Index of current set;
function GenerateTreeBySet(Node, arrayIndex, setIndex)
    // Skip the current set's array index
    if arrayIndex =setIndex then
        return GenerateTreeBySet(Node, arrayIndex + 1, setIndex)
    end if
    // Recursion termination condition
    if Node.length = 1 then
        return Node
    end if
    // Initializing Tree and Child arrays
    Tree ← new []
    Child ← new []
    Append Node[0] into Child
    for each e ∈ Node till Node.length - 1 do
        // Comparing the intersections using IntersectLabel
        if e.IntersectLabel[arrayIndex] = e.next().IntersectLabel[arrayIndex] then
            Append e.next() into Child
        else
            Append GenerateTreeBySet(child, arrayIndex + 1, setIndex) into Tree
            Child ← new []
            Append e.next() into Child
        end if
    end for
    Append GenerateTreeBySet(child, arrayIndex + 1, setIndex) into Tree
    return Tree
end function
```

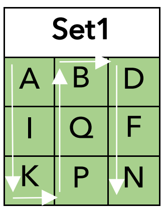
Arrange and organize element glyphs in the set grid based on the tree structures (set memberships of the intersecting elements): After the tree structures for the sets on screen are formed, they can then be visualized and converted to the treemap layout in each set grid. We use the Grid TreeMap algorithm (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).

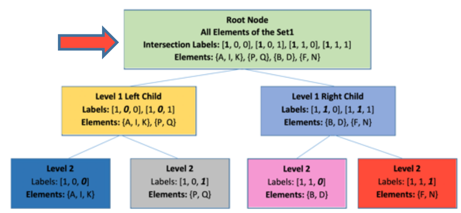
As illustrated in the following examples, GridSet divides and arranges element glyphs in different subdivisions of the set grid according the tree *T*:

Arrows in the set grid indicate scanning direction in each set division.

RootNode:

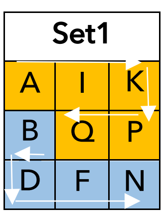
All the elements contained in the root node are first filled in the set grid vertically—either from top to bottom or from bottom to top.

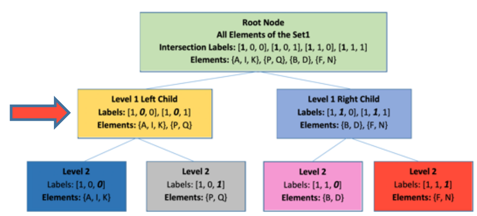




Level 1:

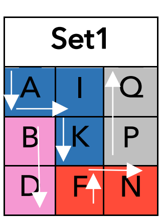
At Level 1 of *T*, the algorithm allocates elements in the left child to the vertical subdivision on the top (yellow), and elements in the right child to the bottom (light blue) vertical subdivision by scanning elements in rows.

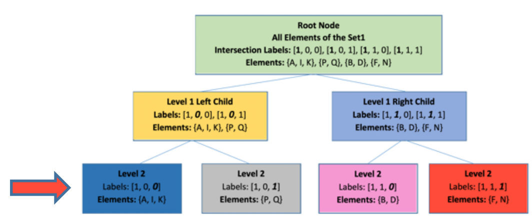




Level 2:

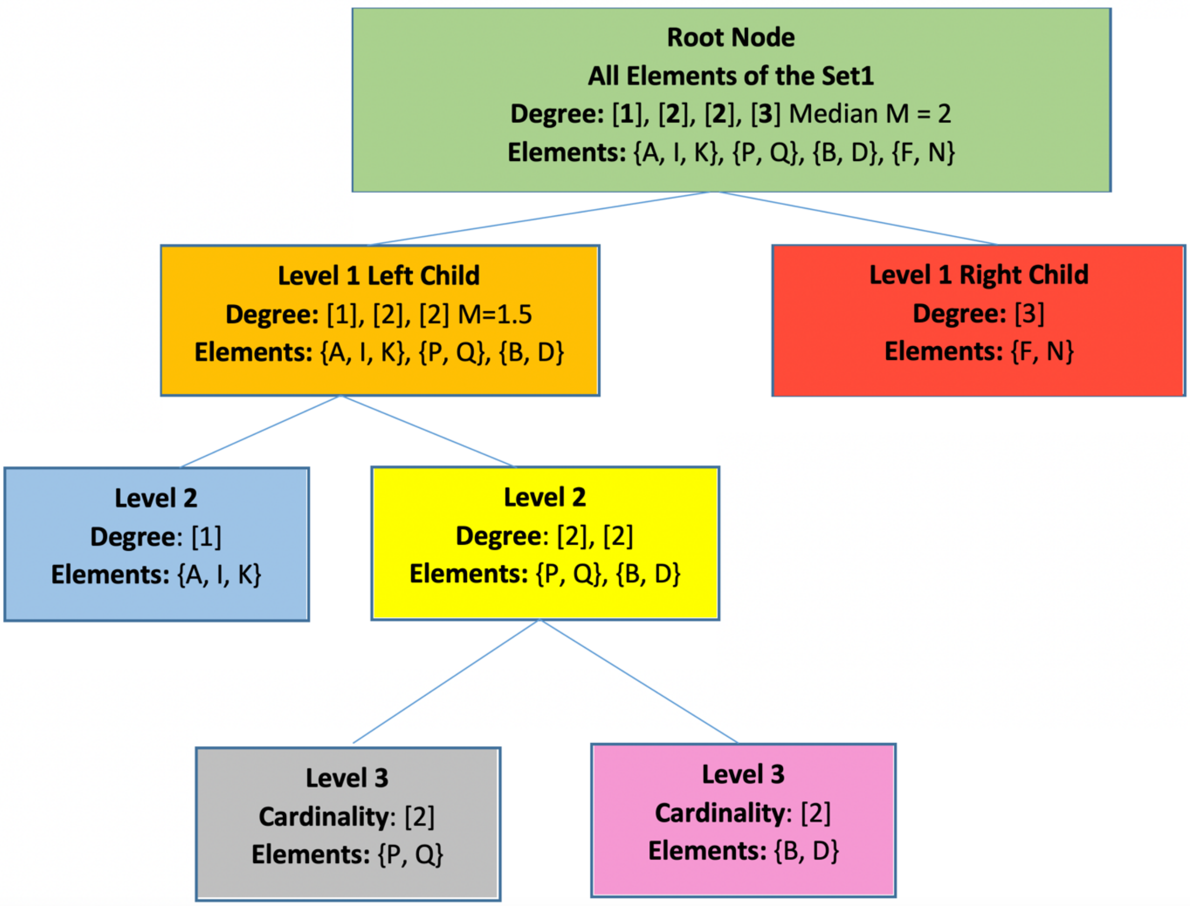
At level 2 (the last level) of *T*, the algorithm divides and lays out subdivisions recursively within the two subdivisions (yellow and light blue subdivisions) created during the previous step, filling the grid slots in each subdivision with associated elements, either vertically or horizontally.





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.



A tree based on the degrees of the intersecting elements

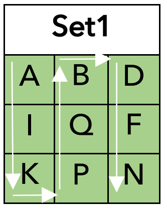
Algorithm 2 provides details for generating a tree structure based on either degree or cardinality of sets.

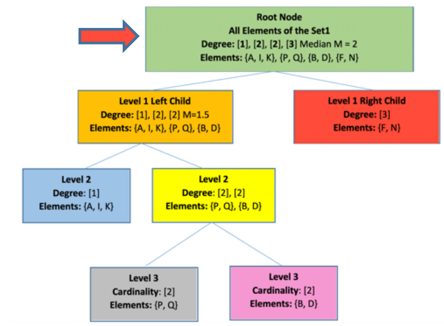
Algorithm 2 Generate a Tree Structure based on Degree or Cardinality

```
//Node structures to repersent child nodes
Initialize Tree,leftNode, rightNode as empty arrays
S ← all sets in the Main view
for each s ∈ S do
    E ← all the exclusive intersections of set s
    Sort E in increasing order of Degree/Cardinality
    s.tree ←GenerateTreeByDeg(E)
end for
function GenerateTreeByDeg(intersections)
    if intersection.length = 1 then
        return intersections
    end if
    if all of i ∈ intersections have the same Degree/Cardinality then
        return intersections
    end if
    M ← a median of the Degrees/Cardinalities of intersections
    for each i ∈ intersections do
        if Degree/Cardinality of i < M then
            Append i into leftNode
        else
            Append i into rightNode
        end if
    end for
    Append GenerateTreeByDeg(leftNode) into Tree
    Append GenerateTreeByDeg(rightNode) into Tree
    return Tree
end function
```

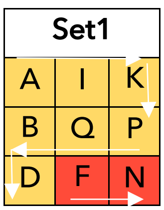
Arrange and group element glyphs in the set grid using the tree structures based on the degrees of the intersecting elements (arrows indicate scanning direction in each set division):

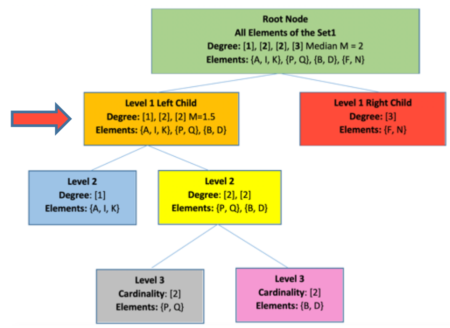
Root Node:



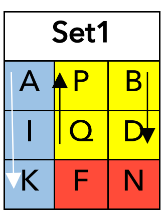


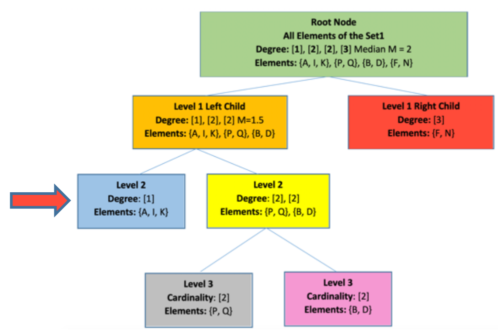
Level 1:



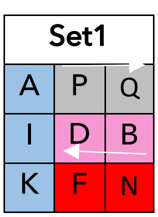


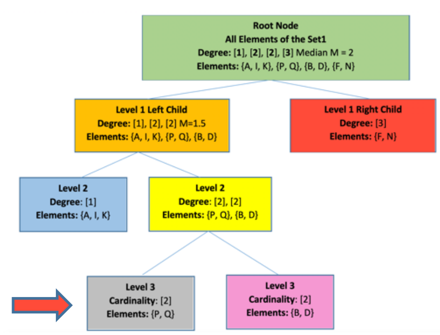
Level 2:



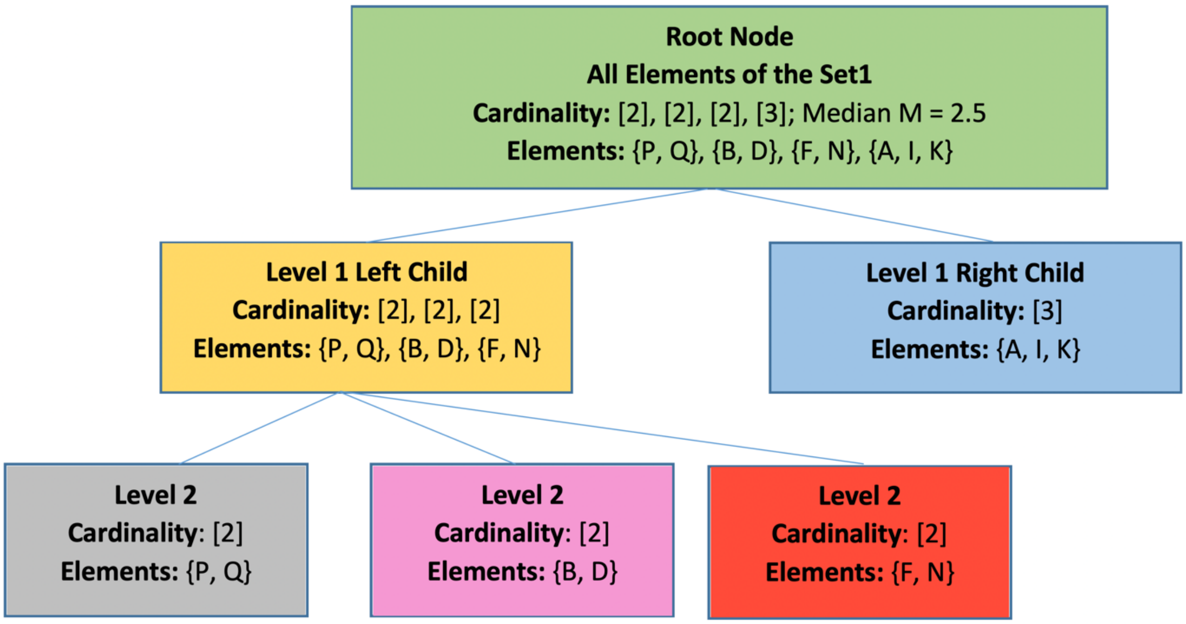


Level 3:





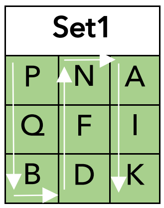
1.3.3 Generate a tree based on the set cardinalities of the intersecting elements (the number of elements in the intersections)

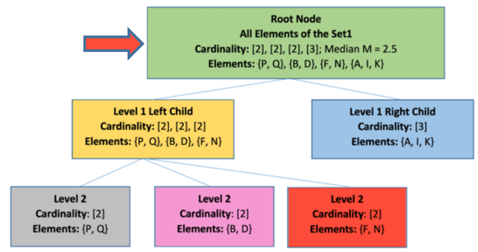


A tree based on the degrees of the set cardinalities of intersecting elements

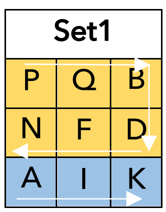
Similarly, arrange and group element glyphs in the set grid using the tree structures based on set cardinality (arrows indicate scanning direction in each set division):

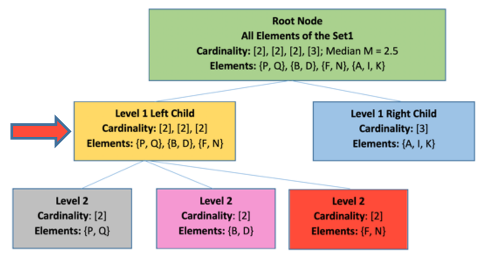
RootNode:



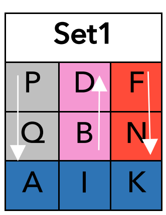


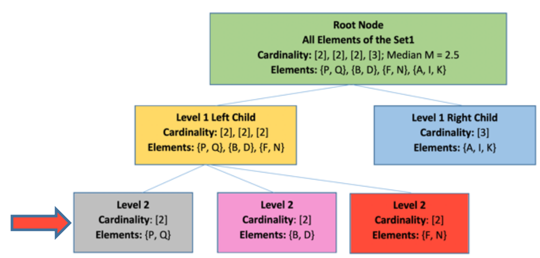
Level 1:





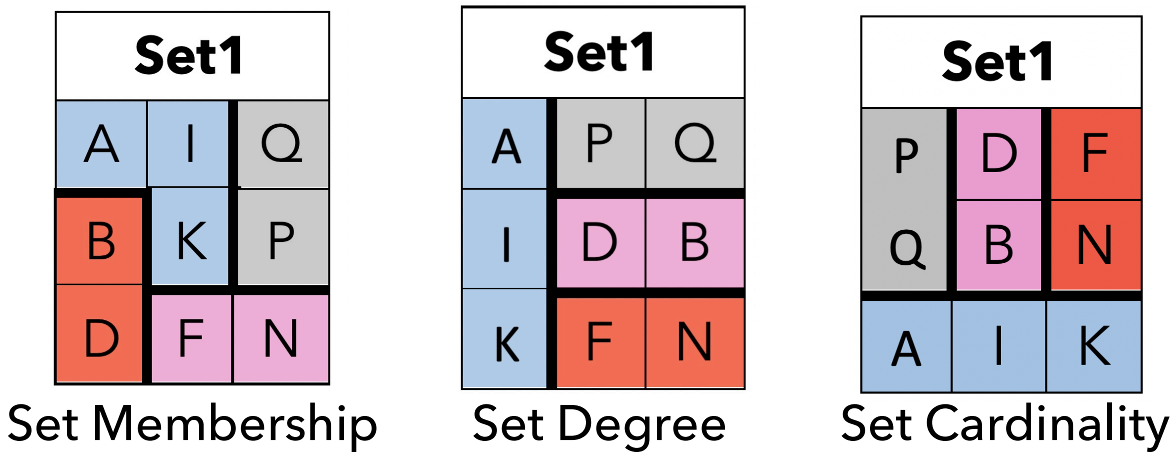
Level 2:





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.



Reference

[1] T. Schreck, D. Keim, and F. Mansmann, "Regular treemap layouts for visual analysis of hierarchical data," in *Proceedings of the 22nd Spring Conference on Computer Graphics*, 2006, pp. 183-190: ACM.