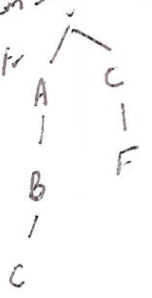
Writing item values in Json

AL.append()
.remove()

FreqCount maintain value set
on push & pop maintain it

errors: followup

- for last Item subsets director has no items.
- suffixSet
- procesTxn round & batchIdx chk.
- freqItemSet don't return root
- freqItSetItr return names.



ISet Tests

num Items = 20.

ISet (18)

superset = ~~18~~. 19, 18

subsets = 19 ; 18.

testSuperSet

is2 (idxToSkip = 0)

↘ ImmedSuper(lastItemId = 19)

↘ ISet ([18])

3

0, 1, 2 0

isSquare (Node ISet)

cNode = root

bld, itemNm

a. → ItemNm ~~[illegible]~~ TreeSet<?>(Comprt)

SortedSet

buildItems~~[illegible]~~ Set

b. build Item ValueMap

c. traverse Patt again to build Basket List

—α—

checkIns

ItemSet
ItemSetItr
ItemSetTest

DynCnt

ItemSetId (sz int[])   is Frequent

Iterator $\overset{\text{OneMore}}{\text{superSets}}$

Iterator $\overset{\text{OneLess}}{\text{subsets}}$

abcd

abc

abd

acd

bcd

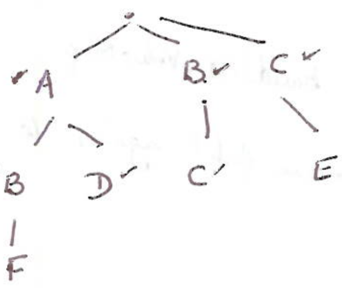
✱ if ids are assigned on 1st scan then b can have id > c's id.

a → 1
c → 2
d → 3

b → 4
c → ~~2~~
d → 3

Increment (Trei, <u>ISet</u>)

ISet: (A, B, CD)



incr Root

T. branch (ISet[0]) ⇒ incr (T.children [ISet[0]] , (B, CD))
A

ISet
    subset (start, end)

Suffix ISet
    start prefix

<u>Suffix ISet</u>

```
Incr ( Trei, ISet )

    trei.cnt ++
    for i in 0..<ISet.sz
        trei.childn [ISet[i]]
            incr ( tr.chldr [ ] , ISet.suffix (. )
```

sz = 5

5 -

	sz	
0	5	(5-0)
1	4	(5-1)
2	3	(5-2)
		[2, 3, 4]

ItemValue   value comparator   val TypeInfo.

- boolean	Boolean
double	Double
- byte	Byte
float	Float
- Int	Integer
Long	Long
- short	Short
Text/String	String

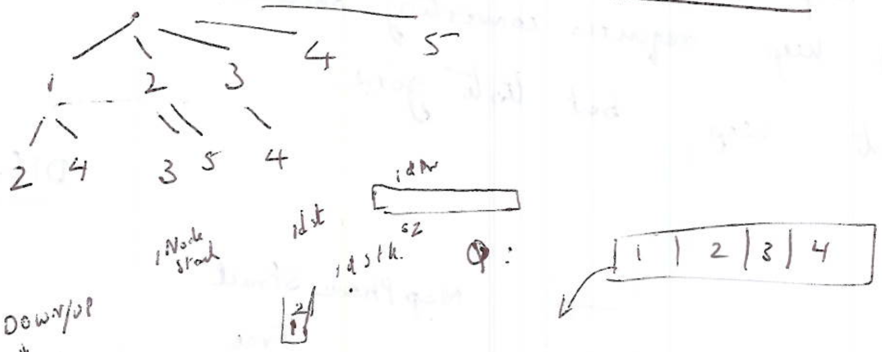
PrimitiveComp < T extends Comparable<T>>

{

cb < T >

of. comp T.

Tree Traversal
distinct
~~currNode~~ itemIds



iNode
    travState   Down/Up
                 down

traverse
    cr node
    iNode stack
    sz = 0
    while   push ~~nt~~Node on stack.

write alg.

1
3
4
5

Current State

ItemSet : represents an IS as an int[]
is writable
convertible to json

probably reusable.

Basket : represents a Basket as AL
is writable
reuse.

ItemSet Iterator : iterate IS
gens ~~[illegible]~~ sub ISs using prefix/suffix

not usable.

MBATest : keep
MBALocalTest : keep requires converting skeleton to json
ItemSet Test : keep but tests gone.

— x —

Plan :

- ItemNode
- Increment Count
- Trei Traversal
- Item Set . Immediate Super Set Iterator
  . Subset of Size N-1 iterator

→ Map Phase Struct
Trei
basket Parts
Map Val → Id
AL<ItemValue>
numItems.
num Baskets.

DIC.groovy.
initialize (IPartition ...)
process ()

ItemValue class
Comparator
based on Prim Type.

initial Processing:

1. read input Partition into a BBList of Baskets → Item Value set idx.

   also create a Sorted ~~[illegible]~~ Map] assume held in memory.

   Item Value → id

2. create partition BndryList each Partition is 1000 entries, except last which can be upto 1500 long.

3. [initialize Trei
   run overall Process on Pg 1

~~[illegible]~~

final Processing

convert ~~[illegible]~~ Sq Nodes in Trei into json repr of Item Set

⇒ need id to value.

-α-

during Basket setup need Item Value → id → maintain HashSet of Item Value comparator on value.

ArrayList sorted on value.

~~resort on id~~.

at end of Basket Building place in AL and sort by idx.

Immed super set of an Item Set

add any id s.t. id > last id in Item Set
is this enough: every IS has items ordered, so enough

Subsets of an Item Set

only consider IS of length n-1 with 1 id removed.

an IS can be SQ when ~~count > SQ~~ support > threshold.

this is wrong

Is this enough?

No counter e.g. A, AC are in tree
don't add ACD since AD not in tree
in tree

~~Recurring~~

If IS is a string, subset Generation technique

subsetGen ('ABCD')

an IS in tree ⇒ all its subsets are SQ
⇒ only check all subsets of length n-1

Gen Subsets (Item Set)

```
get IdList from ItemSet
    device an Iterator that
        returns n IS, where n is length of IS

        ~~for each value of~~
        pos = 0 .. n-1
    for each val of pos skip pos^th elem in IS.
```

```
Item ~~Set~~ Node {
    int itemId
  Map<int, Item ~~Set~~ Node> children
  int   basketIntroducedISet
  int   roundIntroducedISet
   int  support
  state  state

}
```

overall Process

```
    - setup Root Node ( represents empty set, itemId == -1 ) in state.SOLID_SQ
    - setup Node for each ItemId as a child of Root Node in state.
                                                              DOTTED_CIR.

    while ( tree has dotted nodes )
    {
        process Next Batch of Baskets:
        {   for each Basket b :
                invoke  IncrementCount( b, root of Trei, basketPrefix = 0 )

        }
            traverse Tree : 1. convert any DOTTED_CIRCLE -> DOTTED_SQ
                             if they have crossed threshold.
                          2. for a new DOTTED_SQ node
                               for each immed superset S
                                  check that all subsets of S ~~[illegible]~~ is
                                              a SQ
                                  if yes add it to tree as a DOTTED_CIR.

                          3. if a DOTTED item has been counted through
                             all baskets => ItemSet. ~~[illegible]~~ basketIntroducedISet
                                                        ~~[illegible]~~
                                                     is in this Round
                                   & ItemSet.roundIntroduced < current Round

                       make it SOLID.
```

tracks current ItemSet
the set implied
by crnt Node.