Transactions importance based on Binary weights for Association Rule Mining

Goal:

1. Finding Frequent item set,
2. Association Rules generation.

Main_idea:

Let say,we have following problem:
Assume min. support as 50%.

Transactions	item's
t1	{a,b}
t2	{a}
t3	{a,c}
t4	{b,d}
t5	{a,b,d,e}
t6	{b,d,e}
t7	{d,e}

..
..
...

min_sup = (50*7)/100 = 3.5 == **4**,
Firstly calculate support for each item

item	support
a	4
b	3
c	1
d	4
e	3

now sort item's in ascending order(acc. to support):
a->d->b->e->c:

We can set transactions importance,,based on binary behavior , it means a item can be present in a transaction or not be.
if it present its value is '1' or '0' if not present.
So,we have to create a cost_matrix(a simple 2D matrix)..
like this...

2^..	4	3	2	1	0
	16		8	4	2	1

T\item	a	d	b	e	c	...|weight
t1	1	0	1	0	0	20
t2	1	0	0	0	0	16
t3	1	0	0	0	1	17
t4	0	1	1	0	0	12
t5	1	1	1	1	0	30
t6	0	1	1	1	0	14
t7	0	1	0	1	0	10

now,
sort the transactions acc. to their weights...
we get: t5->t1->t3->t2->t6->t4->t7 ::
now take 'N' top transactions...
let say N=3;

Transactions	item's
t5	{a,b,d,e}
t1	{a,b}
t3	{a,c}

also we have support list:

item	support
a	4
d	4
b	3
e	3
c	1

we now min_sup is = 4(given)
eliminate item
we get:

item	support
a	4
d	4

now,take very first transaction 't5'
chk its item_set that it contains latest list of item or not...if yes okay the transaction is accepeted otherwise rejected.
take 2nd transaction and so on.....

so,by doing this we get our frequent item set as t5 = (a,b,d,e);
but, **#_MONOTONICITY_PROPERTY** says that , if (a,d) is frequent its superset is also frequent...
so we take (a,d) instead the whole transaction t5(a,b,d,e).

_frequent_item_set = (a,d).

/@@
@@@@@@@@@@@@@@**___Rules_generation____**@@@@@@@@@@@@@@@@@
@@/

_maxRules =(2^a - 2); // a = count of items in _frequent_item_set.

= 2 ^ 2 - 2
= 2

generatedRules:
a->d;
d->a;
Rules are generated by simple permutation login:
*char * generateRules(char *itemList);*

Representation_of_symbols:

Given list of Transactions: T(size_of_T's)

Algo:

#define min_sup = - // min. support value
#define N = - // top transactions...

create a cost_matrix(2d array...) (*in accordance with sorted support value list*) of transactions::item.
fill with 0 | 1
0 = absence of _item.
1 = presence of _item.
Loop:
calculate weight W_i : i in transaction.count T
end:

queue = sort($W_{i.(ascending)}$):
loop:
t = queue.pop();
if t $t \subseteq supportList$:
consider t ;
else
reject t;
