PINAPL (& PINALP2) STANDARD HERRINGBONE PINEAPPLE KNOTS

Lengthy user's tips but may be you will be rewarded by what the 5 programs can do for you when you will have learned to correctly use them .

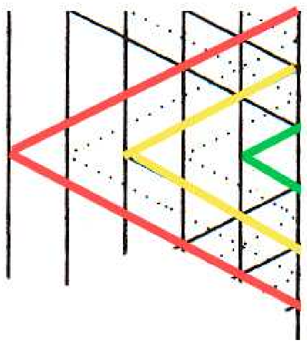
The programs PGR1 .....PGR5 in **PINAPL take the load of calculation off your brain**.
NONETHELESS you will need to set yourself in 'BRAIN ENABLED MODE' when tackling the coding of those knots.

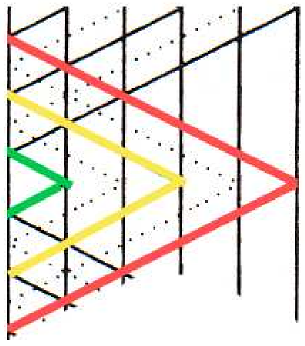
A POINT MUST BE MADE CLEAR : ALL THIS REST ON SCHAAKE AND TURNER's MATHEMATICAL GROUND BREAKING WORK :
No disrespect intended to all grade of 'expert' but…… you will be better for having read ***THE BRAIDER*** and the ***Pamphlets*** if you want to be anything else than a spider** making knots, relying on mere ganglia rather than on grey matter, or rather on handed down recipes not really explained and not fully understood - (at least in the full meaning usually attributed to those terms.)
**the sort of spider that is "constrained" to specially tailor cores for the knots they know how to make but are unable to reciprocate and tailor their knots to a given core, those poor souls are "constrained" to remain into a small "over-trodden" fenced territory.
As a 16th century French writer, MONTAIGNE, said " knowing by rote is not knowing, it is holding what has been put in the care of one's memory .

First a small bit of sketchy NOMENCLATURE (read SCHAAKE & TURNER to get the full story or a very 'gauche' summary in my Turkshead-pages)

Those knots are part of a bigger CLASS : the REGULAR NESTED CYLINDRICAL KNOTS. (**BUT** despite what has always seem to me -- I have met like-thinking in S & T now ! feel less lonely for it -- to be the too often encountered mistaken usual opinion , what knot has 'nested bight' is **NOT** necessarily a **PINEAPPLE** knot.**)**

The nested bight leads to distinguish **BIGHT BOUNDARIES** (I will think PINS LINES rather) :
Here 5 Bights in each NEST , so FIVE BOUDARIES on which you put pins to make the knot. (note that two adjacent pins lines are separated by 2 columns)





You need to know all about :
The **NUMBER OF BIGHT NESTS (B*)** on each rim
The **NUMBER OF BIGHT PER NEST** which leads to **the NUMBER OF BIGHT BOUNDARIES** (I prefer to call them pins lines or pins rim as I used the cylinder frame of reference mostly).Those boundaries are vital components in the calculation and for approaching an understanding of the inner nature of those knots.

You also need the NUMBER of LEADS in EACH STANDARD THK COMPONENT of the COMPLETE PINEAPPLE. (this number of LEADS in **each** component THK will be **denoted by ' S ' in the calculation**.)

The so-called **NUMBER OF PASS (or of STEPS in the making) will be denoted by letter "A".**
Note that if A = 5 then there is 5! ways of making the pineapple.
That is factorial, that is 5 * 4 * 3 * 2 * 1 = 120 different ways to make the pineapple.
You already know from using the others HP48GX programs in EMU48 that we can compute the columns in which a crossing exist.

This is relating to the component THK seen in isolation.
It is what figure the upper part of each illustrations of calculations that will be

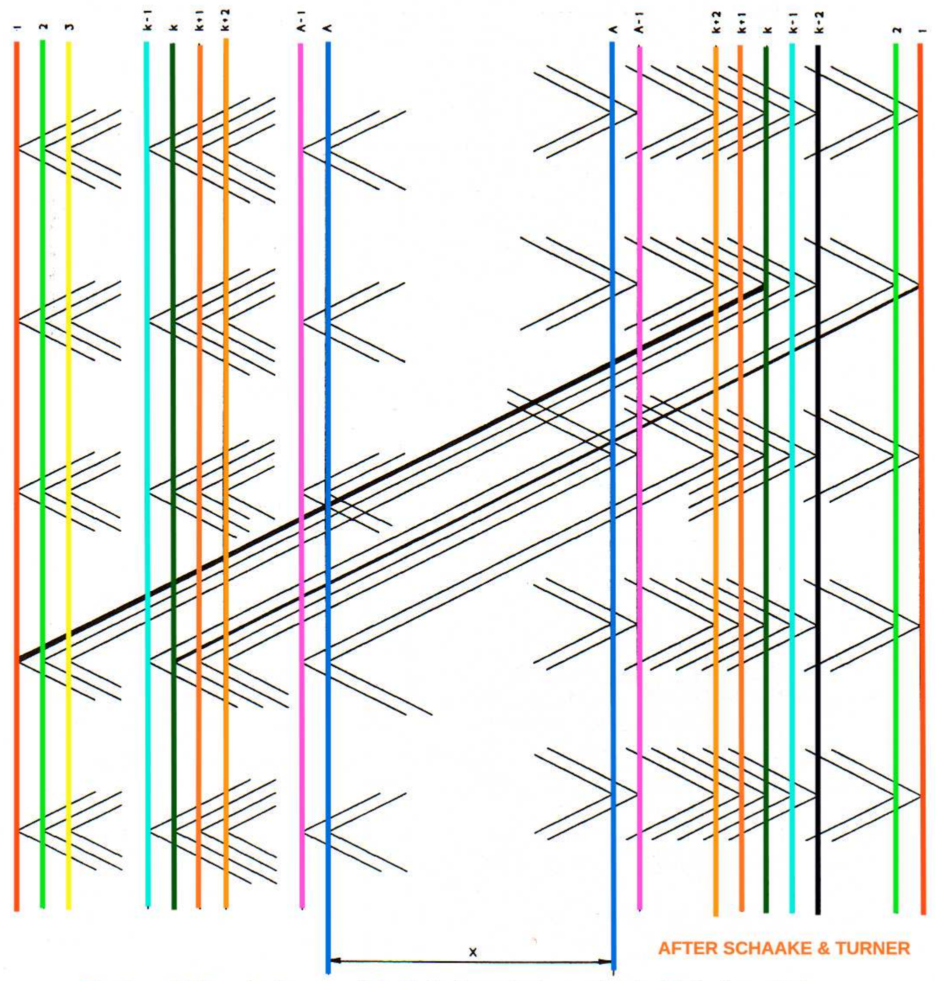


Fig. 3 — Half-cycles between left Bight-boundaries and right Bight-boundaries.

following.

When you will be doing the next (immediately following) PASS then the crossings of the component THK itself will be added to the crossings collection made by the preceding component(s) installed.
So the "recipe" is to **calculate the "base component THK matrix " and the "already laid components matrix" , adding the two will give you another matrix** that is indeed quite easy to read to get the code for each H-P (Half-Period) in the PASS.

We will from this point work on an example inspired by **SCHAAKE AND TURNER** and entirely done (with some others as verification) on my **HP48GX.**

A **FIVE PASS** **HERRINGBONE ¨PINEAPPLE**
ALL TOLD 29 LEAD for 20 BIGHT [(5*3) + (7*2) L (4*5)B

The programs are set for a max of 30L 30B (GDC permitting) and 15 PASS
I do believe you will NEVER BE ABLE TO TIE A 15 pass PINEAPPLE of 27L 30B and 29L 30B component THK ; just imagine 421 L 450 B
{(27*7) + (29*8)= 421 L and the Bight to go with that (30 * 15) = 450 B}

THK BASE COMPONENTS IN THIS ONE COME IN TWO GROUPS. THE 2 GROUPS ARE WITH THE SAME NUMBER OF BIGHTS BUT WITH NUMBER OF LEADS (the "S") THAT ARE SEPARATED BY 2 UNITS (ex : 5 and 7)

One group of two will be 7 L (so **S**=7) and 4 **B***

The second group will be 5 L (so **S**=5) and 4 **B***

4 B* mean 4 NESTS PER BIGHT RIM

5 BIGHTS PER NEST so 5 BIGHT BOUNDARIES (PINS LINES) on each border

For each component you will have to enter the
LEFT BIGHT BOUNDARY
and the
RIGHT BIGHT BOUNDARY

PRACTICAL WORKED EXAMPLE

29 LEAD 20 BIGHT 5 PASS

The order of the **FIVE PASS** will be chosen as : (5! manners to order the passes)

PASS ONE (one layer added on nothing pre-existing)
5L 4B THK component so
S=5 B* =4 A = 1 Left boundary **= 1 Right** b **=1**

PASS TWO (one layer added to ONE pre-existing layer of crossings)
7L 4B THK component so
S=7 B* =4 A = 2 Left boundary **= 1 Right** b**=1**

PASS THREE (one layer added to TWO pre-existing intermingled layers of crossings)
5L 4B THK component so
S=5 B* =4 A = 3 Left boundary **= 3 Right** b **=2**

PASS FOUR (one layer added to THREE pre-existing layers)
7L 4B THK component so
S=7 B* =4 A = 4 Left boundary **= 2 Right** b **=1**

PASS FIVE (one layer added to FOUR pre-existing layers)
5L 4B THK component so
S=5 B* =4 A = 5 Left boundary **= 4 Right** b **=4**

After you have understood this page and the following illustration (modified from **SCHAAKE & TURNER** for the edification of the not "abstract" minded) we will then look at how to use the **FIVE** programs I wrote : **PINAPL.HP**

«PGR1»

You will have to enter '**S**' and '**B***'
(remember to break the HALT with a CONT command)
The calculation will put on the stack (and STOre in a series of HP1 ……HP(b*2) variables) the Columns where a crossing happen. You can exploit it as such with the aid of a precise diagram of the knot in its finished state.

You have now to RUN **PGR2**

«PGR2»

PGR2 ask for a digit identifying the LIST that will be made .

(remember to break the HALT with a CONT command)

The computation will produce a LINU(index) file STOring a MATRIX that is the result.
This is : the Basic component *as seen in isolation*.

Now you need to find the resulting CROSSING due to the *component THK* being ADDED on **ALREADY LAID** THK COMPONENT(S)

In fact what you will exploit as matrices is the result of

«PGR1» immediately followed by **«PGR2»**
I let persist the intermediate step of PGR1 results persist as visible rather that transparent for the user as they are practically useful in some situations and are good teaching.
Resulting matrices are STOred in variables named **LINU(index)** index being the number identifying the LIST (that is converted to MATRIX form at the end stage of PGR2)

You may decide to make at one go the whole series of 'THK-component-in-isolation' matrices for the whole of the number PASS in the intended knot (so reiterate **[PGR1 + PGR2]** as often needed to cover all the PASS)
or
you may decide to immediately follow with doing the calculation for the "reference" or the already laid layers or PASS then you have to **RUN PGR3** (to do all the PASS you will do reiteration of PGR1 + PGR2 followed by PGR3 and….)

«PGR3»

You will have to enter '**S**' and '**B***' (identical to the one entered for PGR1 for the corresponding PASS, but as I have for didactic reason "disjoined" the use of PGR3 from the preceding calculation phase it must be entered again)

Again the **LIST NUMBER** (this is for user's usage so can be arbitrary as long as you remember and stay coherent and congruent – better use the real number == pass number)
Enter **NUMBER OF THE PASS** (this must be conform to "reality" as it is an essential part of the calculation)
ENTER **NUMBER OF THE LEFT BOUNDARY**
ENTER **NUMBER OF THE RIGHT BOUNDARY**

THE RESULTS are STOred in **LREF(index)** files as a MATRIX (these are easier for calculation than LIST are) Later we will add **LINU(index)** and **LREF(index)** by matched PAIRS (match according to identical (index) the result being STOred in **PASS(index)** files

As in : **LINU1 + LREF1 = PASS1**

NOT as

LINU1 + LREF2 = PASS3
or

LINU2 + LREF4 = PASS5

«PGR4»

This one is making the 'synthesis" : the addition of the calculated matrices to obtain the final matrix (in **PASSn** files) of the considered PASS (with index **n**)
Bright users will be rather easily able to directly use on their own the results put in files.

«PGR5»

This one is for those feeling like "don't want to feel my brain work", it will help the lazy by directly making the code for each half-period of the **PASSn** variable studied.

Those coding will be put on the STACK and a LIST of those will be STOred in the **PASSn** file

«PGR6»

Will pace you though the FIVE proceeding PGR and let you get the calculation of one PASS at one go (you still have to make entries !). Another version of **PINAPL** will be (may be ; if feedback is given on that) written allowing to treat a full PASS with entries to make only at the start.

DO NOT FORGET TO USE : «CLEAN» to erase the used variables before closing down the HP48GX or the EMU unless you have a use of them again ("eat" quite a lot of memory on a "real" HP48GX)

PINALP2 This one a slight modification of PINALP

«PGR»

This one will SEAMLESSLY (once you have made the necessary entries) calculate a whole PASS.
Of course you will have to COMPLY WITH THE NATURAL ORDER of the PASSes in you projected knot (or it will amount to building the third level of a building without having built the street level, 1 floor, 2 floor)
So you will have to reiterate **PGR** for PASS 1 , PASS (1 + 1 =2) PASS (2+1 = 3) PASS (3+ 1 = 4)……………PASS ((n-1) , PASS (n)

Otherwise just refer to the same documentation as for **PINAPL**

BEWARE : DO NOT use «CLEAN» BETWEEN RUN of this **«PGR»** or you will erase variables that are necessary for RUN to come, but DO NOT FORGET TO RUN «CLEAN» when you are finished and just before the very first RUN of **«PGR»**

Say you have as “end result for the PASS you intend to lay :

[
[3 4 4 4 3]………………………………….correspond to FIRST H-P
[3 4 4 5 3]………………………………….correspond to SECOND H-P
[3 4 4 5 3]………………………………….correspond to THIRD H-P
[3 4 5 5 3]………………………………….
[3 4 5 5 3]………………………………….
[3 5 5 5 3]………………………………….
[3 5 5 5 3]………………………………….
[4 5 5 5 3]………………………………….correspond to EIGHTH H-P
]

Remember that the coding is
U – O – U – O ………… U – O – U – O ….

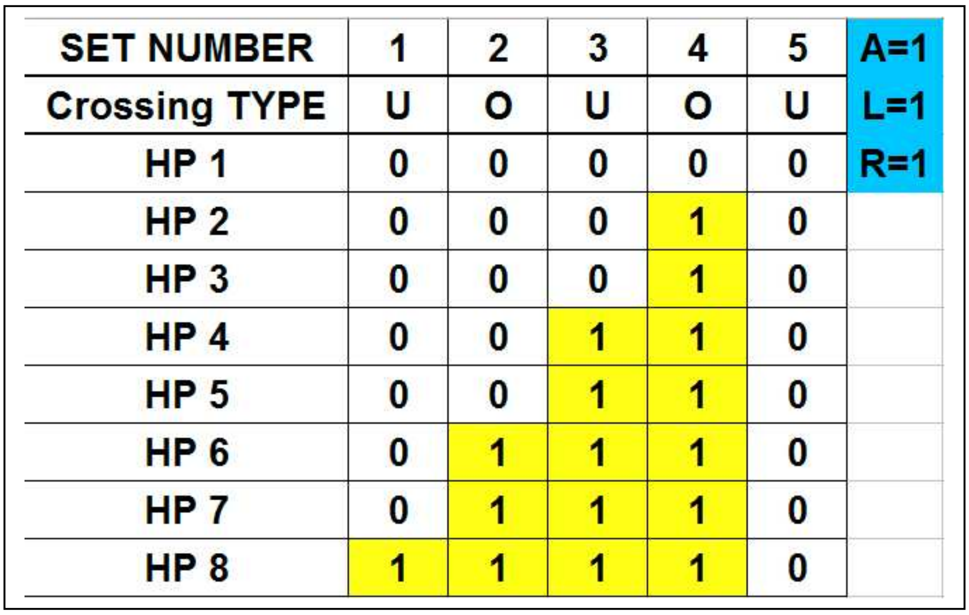
So you have for the N^th PASS

[3 4 4 4 3] that is HP1 == [U3 O4 U4 O4 U3]

[3 4 4 5 3] that is HP2 == [U3 O4 U4 O5 U3]

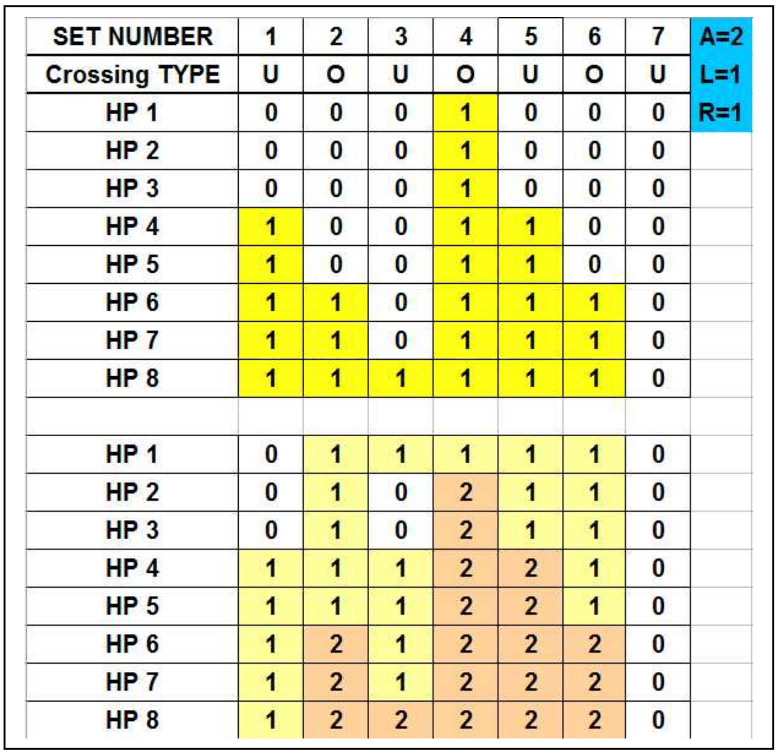
AND SO ON…

DO THAT FOR PASS ONE, ……,THEN LAST PASS and you have all that is needed to put cordage on the pins !
Now for the suite of PASSes in the WORKED EXAMPLE :



SET NUMBER	1	2	3	4	5	A=1
Crossing TYPE	U	O	U	O	U	L=1
HP 1	0	0	0	0	0	R=1
HP 2	0	0	0	1	0	
HP 3	0	0	0	1	0	
HP 4	0	0	1	1	0	
HP 5	0	0	1	1	0	
HP 6	0	1	1	1	0	
HP 7	0	1	1	1	0	
HP 8	1	1	1	1	0	

This is simply the coding of the very first THK component laid. No mystery in that.



SET NUMBER	1	2	3	4	5	6	7	A=2
Crossing TYPE	U	O	U	O	U	O	U	L=1
HP 1	0	0	0	1	0	0	0	R=1
HP 2	0	0	0	1	0	0	0	
HP 3	0	0	0	1	0	0	0	
HP 4	1	0	0	1	1	0	0	
HP 5	1	0	0	1	1	0	0	
HP 6	1	1	0	1	1	1	0	
HP 7	1	1	0	1	1	1	0	
HP 8	1	1	1	1	1	1	0	
HP 1	0	1	1	1	1	1	0	
HP 2	0	1	0	2	1	1	0	
HP 3	0	1	0	2	1	1	0	
HP 4	1	1	1	2	2	1	0	
HP 5	1	1	1	2	2	1	0	
HP 6	1	2	1	2	2	2	0	
HP 7	1	2	1	2	2	2	0	
HP 8	1	2	2	2	2	2	0	

Now we are doing **PASS TWO** so the crossing are piling up in each columns : those already there and the one added.
Upper part is the added THK component as seen "in isolation"
The lower part is the addition of "reference" coding already laid and the newly added one in the last PASS done.

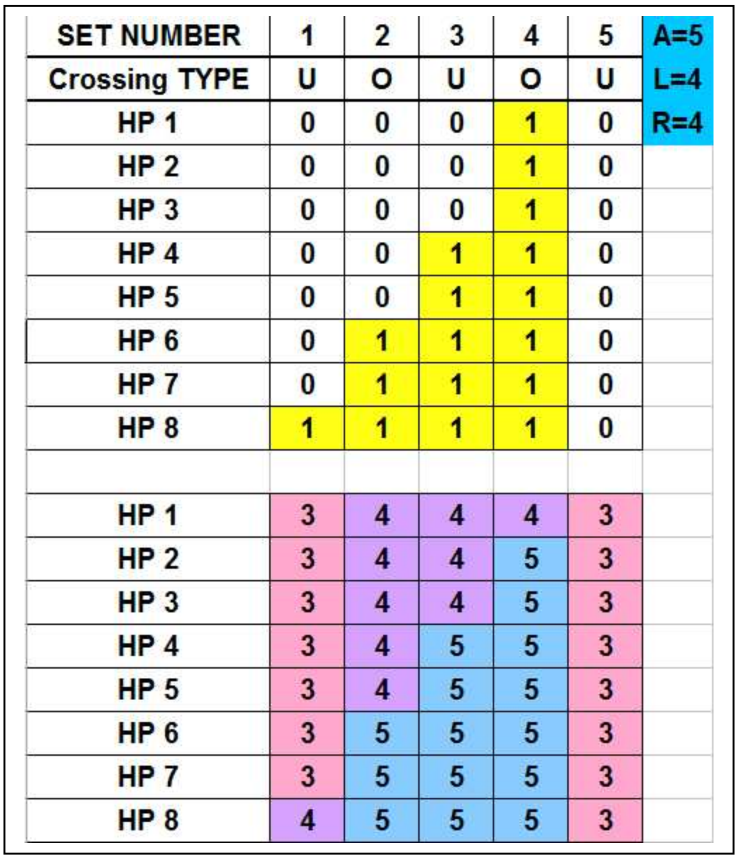
Before using the **PINAPL** you need to be **VERY CLEAR ABOUT YOUR PROJECT** : you will need a **VERY PRECISE DIAGRAM** :

*** NUMBER OF NEST**
**** NUMBER OF BIGHT PER NEST ==NUMBER OF BOUNDARY LINES**
**** *NUMBER OF LEAD IN EACH COMPONENT THK THAT GIVES THE 'S' parameter.**
****** THE PRECISE STARING LEFT BOUNDARY and ARRIVING RIGHT BOUNDARY FOR THE FIRST H-P OF EACH COMPONENT THK.**
******* THE ORDER IN WHICH YOU WILL PROCEED** (remember 'A' PASS means A! ways of going about doing the knot
A = 2 2 * 1 = 2 ways
A = 3 3*2*1=6 ways
A = 4 4*3*2*1= 24 ways
A = **5**
5*4*3*2*1=**120** ways

SET NUMBER	1	2	3	4	5	A=3
Crossing TYPE	U	O	U	O	U	L=3
HP 1	0	0	0	1	0	R=2
HP 2	0	0	0	1	0	
HP 3	0	0	0	1	0	
HP 4	0	0	1	1	0	
HP 5	0	0	1	1	0	
HP 6	0	1	1	1	0	
HP 7	0	1	1	1	0	
HP 8	1	1	1	1	0	
HP 1	2	2	2	2	1	
HP 2	1	2	2	3	2	
HP 3	2	2	2	3	1	
HP 4	1	2	3	3	2	
HP 5	2	2	3	3	1	
HP 6	1	3	3	3	2	
HP 7	2	3	3	3	1	
HP 8	2	3	3	3	2	

WITHOUT THOSE ELEMENTS YOU CANNOT MEANINGFULLY COMPUTE THE CODING EITHER MANUALLY OR WITH THE PROGRAM

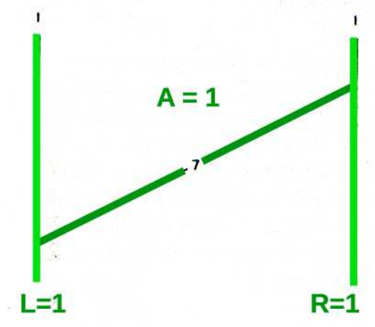
SET NUMBER	1	2	3	4	5	6	7	A=4
Crossing TYPE	U	O	U	O	U	O	U	L=2
HP 1	0	0	0	1	0	0	0	R=1
HP 2	0	0	0	1	0	0	0	
HP 3	0	0	0	1	0	0	0	
HP 4	1	0	0	1	1	0	0	
HP 5	1	0	0	1	1	0	0	
HP 6	1	1	0	1	1	1	0	
HP 7	1	1	0	1	1	1	0	
HP 8	1	1	1	1	1	1	0	
HP 1	1	3	3	3	3	3	0	
HP 2	0	3	3	4	3	3	1	
HP 3	1	3	3	4	3	3	0	
HP 4	1	3	3	4	4	3	1	
HP 5	2	3	3	4	4	3	0	
HP 6	1	4	3	4	4	4	1	
HP 7	2	4	3	4	4	4	0	
HP 8	1	4	4	4	4	4	1	

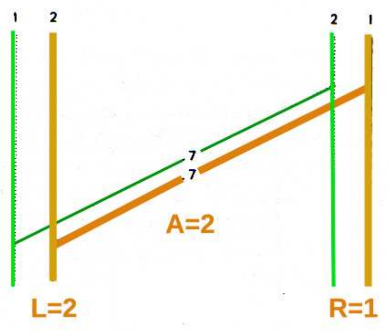


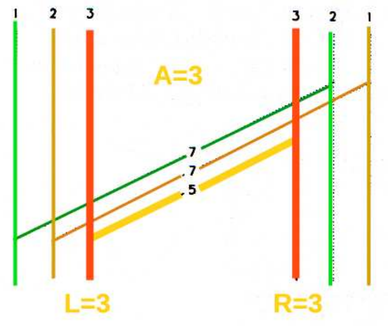
SET NUMBER	1	2	3	4	5	A=5
Crossing TYPE	U	O	U	O	U	L=4
HP 1	0	0	0	1	0	R=4
HP 2	0	0	0	1	0	
HP 3	0	0	0	1	0	
HP 4	0	0	1	1	0	
HP 5	0	0	1	1	0	
HP 6	0	1	1	1	0	
HP 7	0	1	1	1	0	
HP 8	1	1	1	1	0	
HP 1	3	4	4	4	3	
HP 2	3	4	4	5	3	
HP 3	3	4	4	5	3	
HP 4	3	4	5	5	3	
HP 5	3	4	5	5	3	
HP 6	3	5	5	5	3	
HP 7	3	5	5	5	3	
HP 8	4	5	5	5	3	

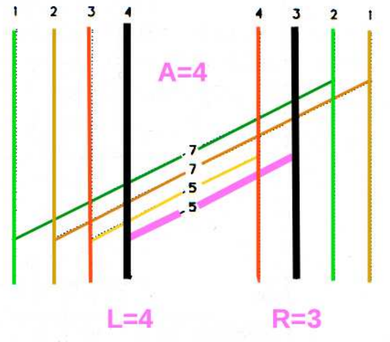
/…

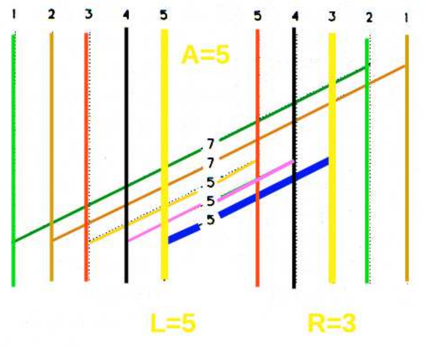
IF YOU WANT SOME TRAINING, in fact you cannot escape it if you want to be proficient ;-)   : DO THE EXERCICE PROPOSED JUST UNDER :











JUST SOLVE THIS ONE WHICH IS ONLY ANOTHER OF THE WAYS AVAILABLE

B* = 4

2 groups one with 7L and the other with 5L just as in the worked example.

Make the CODE FOR EACH FIVE PASS with **PINAPL** **and then with PINAPL2**

AS ILLUSTRATIONS HERE UNDER ARE, DIRECT FROM MY HP48GX, PRACTICAL RESULTS OF THE WORKED EXAMPLE

R5
```
[[ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]
 [ 3 4 4 4 3 ]]
```

R4
```
[[ 1 3 3 3 3 3 0 ]
 [ 0 3 3 3 3 3 1 ]
 [ 1 3 3 3 3 3 0 ]
 [ 0 3 3 3 3 3 1 ]
 [ 1 3 3 3 3 3 0 ]
 [ 0 3 3 3 3 3 1 ]
 [ 1 3 3 3 3 3 0 ]
 [ 0 3 3 3 3 3 1 ]]
```

R3
```
[[ 2 2 2 2 1 ]
 [ 1 2 2 2 2 ]
 [ 2 2 2 2 1 ]
 [ 1 2 2 2 2 ]
 [ 2 2 2 2 1 ]
 [ 1 2 2 2 2 ]
 [ 2 2 2 2 1 ]
 [ 1 2 2 2 2 ]]
```

R2
```
[[ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]
 [ 0 1 1 1 1 1 0 ]]
```

R1
```
[[ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]
 [ 0 0 0 0 0 ]]
```

N4
```
[[ 0 0 0 0 0 0 0 ]
 [ 0 0 0 1 0 0 0 ]
 [ 0 0 0 1 0 0 0 ]
 [ 1 0 0 1 1 0 0 ]
 [ 1 0 0 1 1 0 0 ]
 [ 1 1 0 1 1 1 0 ]
 [ 1 1 0 1 1 1 0 ]
 [ 1 1 1 1 1 1 0 ]]
```

N2
```
[[ 0 0 0 0 0 0 0 ]
 [ 0 0 0 1 0 0 0 ]
 [ 0 0 0 1 0 0 0 ]
 [ 1 0 0 1 1 0 0 ]
 [ 1 0 0 1 1 0 0 ]
 [ 1 1 0 1 1 1 0 ]
 [ 1 1 0 1 1 1 0 ]
 [ 1 1 1 1 1 1 0 ]]
```

N5
```
[[ 0 0 0 0 0 ]
 [ 0 0 0 1 0 ]
 [ 0 0 0 1 0 ]
 [ 0 0 1 1 0 ]
 [ 0 0 1 1 0 ]
 [ 0 1 1 1 0 ]
 [ 0 1 1 1 0 ]
 [ 1 1 1 1 0 ]]
```

N3
```
[[ 0 0 0 0 0 ]
 [ 0 0 0 1 0 ]
 [ 0 0 0 1 0 ]
 [ 0 0 1 1 0 ]
 [ 0 0 1 1 0 ]
 [ 0 1 1 1 0 ]
 [ 0 1 1 1 0 ]
 [ 1 1 1 1 0 ]]
```

N1
```
[[ 0 0 0 0 0 ]
 [ 0 0 0 1 0 ]
 [ 0 0 0 1 0 ]
 [ 0 0 1 1 0 ]
 [ 0 0 1 1 0 ]
 [ 0 1 1 1 0 ]
 [ 0 1 1 1 0 ]
 [ 1 1 1 1 0 ]]
```

To get the “end result” that you want it is only necessary to ADD THE NEW PASS that you intend to make to ALL THE PREVIOUS LAYERS laid (PASSes already done) and this will lead you to the coding to use in order to do without too much trouble that intended new PASS.

For example adding N4 & R4

N4	R4	RESULT
[[0 0 0 0 0 0 0]	[[1 3 3 3 3 3 0]	[[1 3 3 3 3 3 0]
[0 0 0 1 0 0 0]	[0 3 3 3 3 3 1]	[0 3 3 4 3 3 1]
[0 0 0 1 0 0 0]	[1 3 3 3 3 3 0]	[1 3 3 4 3 3 0]
[1 0 0 1 1 0 0]	[0 3 3 3 3 3 1]	[1 3 3 4 4 3 1]
[1 0 0 1 1 0 0]	[1 3 3 3 3 3 0]	[2 3 3 4 4 3 0]
[1 1 0 1 1 1 0]	[0 3 3 3 3 3 1]	[1 4 3 4 4 4 1]
[1 1 0 1 1 1 0]	[1 3 3 3 3 3 0]	[2 4 3 4 4 4 0]
[1 1 1 1 1 1 0]]	[0 3 3 3 3 3 1]]	[1 4 4 4 4 4 1]]
SECOND 7L 4B to be added on…….	**the already laid PASS**	**will get you this.**

THAT WILL DECIPHER AS (essential coding is U - O - U - O…U - O - U - O…U - O…)

RESULT for the FOURTH PASS

```
[[ U1   O3   U3   O3   U3   O3   U0 ]  [Under One Over Two…….Under NONE]
 [ U0   O3   U3   O4   U3   O3   U1 ] = second Half-Period
 [ U1   O3   U3   O4   U3   O3   U0 ] = third Half-Period
 [ U1   O3   U3   O4   U4   O3   U1 ] = fourth Half-Period
 [ U2   O3   U3   O4   U4   O3   U0 ] = fifth Half-Period
 [ U1   O4   U3   O4   U4   O4   U1 ] = sixth Half-Period
 [ U2   O4   U3   O4   U4   O4   U0 ] = seventh Half-Period
 [ U1   O4   U4   O4   U4   O4   U1 ]] = eighth Half-Period
```

U0 Under NONE
O0 Over NONE
Those **U0 and O0** are there only for the sake of being able to use matrices and equal length sub-lists inside LIST and have to be discarded when you write down the coding on a paper.

Happy moments !
I hope you enjoy using this one billionth of what I enjoyed writing those programs on my old faithful HP48GX and savouring **SCHAAKE & TURNER** CRISTAL CLEAR **THINKING**.
You should take to making the effort of reading ***THE BRAIDER*** ”