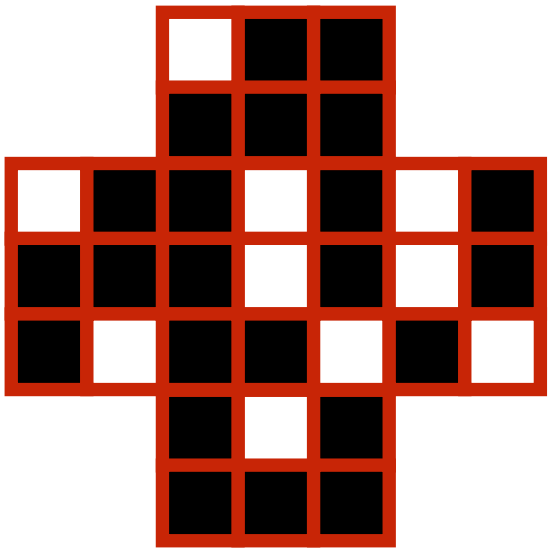
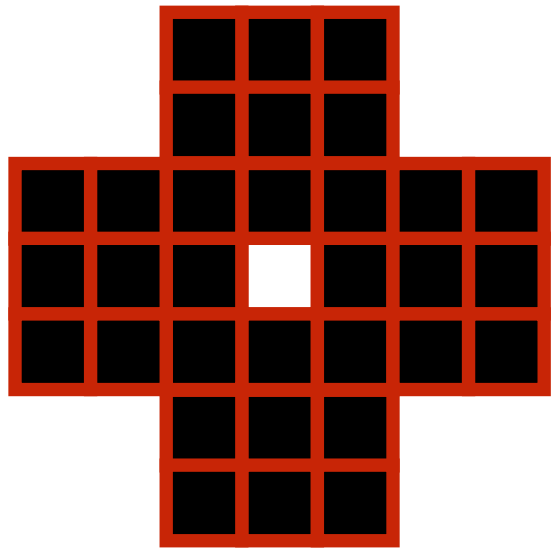
A "**HIЯIQ**" (pronounced higher-I.Q.) puzzle is an array of 33 black or white pixels (bits), organized in 7 rows, 4 of which contain 3 pixels each and the middle 3 rows contain 7 pixels each. You are given a particular configuration of these pixels as in (A).



(A)

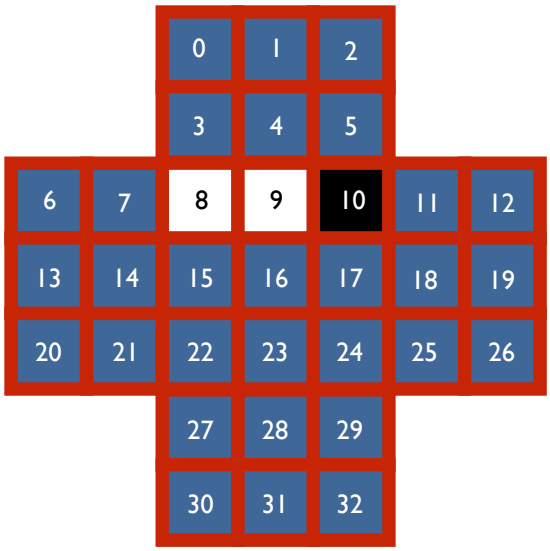


(B)

Your task is to write a JAVA program that will transform this configuration into the *solved* configuration (B). To modify your configuration you are allowed to apply one of two substitution rules whenever the rule is applicable: anywhere in your configuration if one of these patterns is found (horizontally or vertically), you can replace it with the other:

We recommend that you use a tree/graph data structure to represent each reachable configuration of the "**HIЯIQ**" puzzle. The children of a configuration should be the possible configurations that can be reached applying a single substitution rule. Your algorithm should traverse the tree so to locate the *solved* configuration. Once the solution is found, your algorithm should return a **String** containing the sequence of substitutions necessary to solve the puzzle. For convenience we will name the "BBW to WWB" substitution a **B-substitution** and the "WWB to BBW" substitution a **W-substitution**.

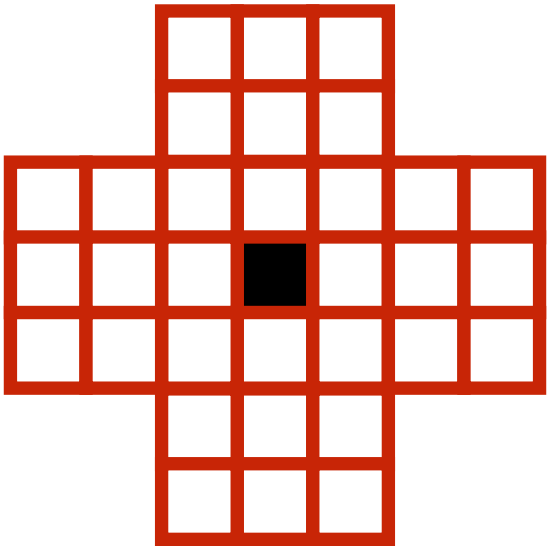
We will use the following notation to represent the locations on the "**HIℜIQ**" board:



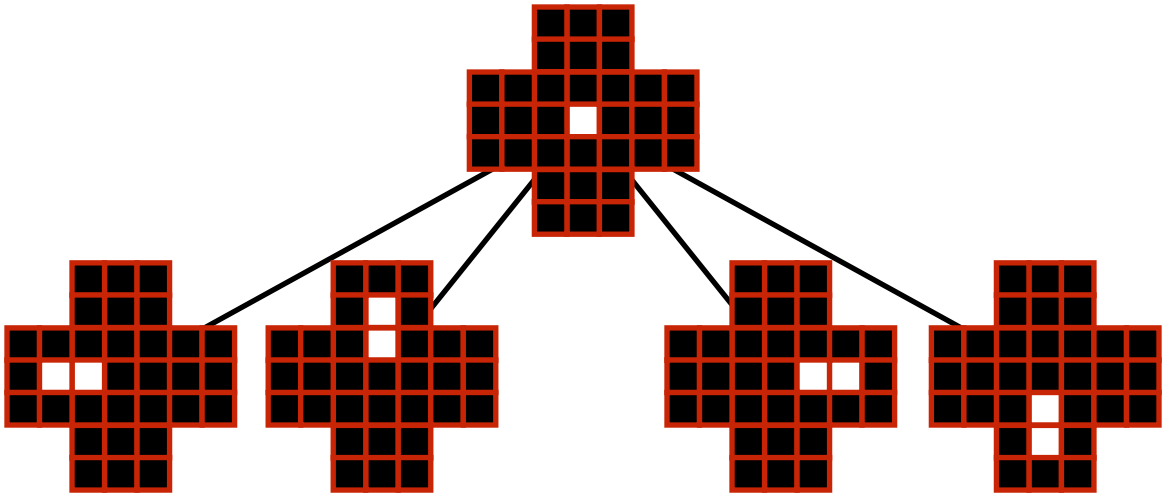
For instance "8W10" means a W-substitution may be applied to pixels at positions 8,9,10. Your output should be something like : "18W16, 5B17,…, 16W28". We will provide you with an Object type **HiRiQ** that you must use for uniformity reasons. This object will contain methods for testing if the puzzled is solved, and outputting a board configuration. This will be available through the course web page very soon.

In order to make your program (space) efficient, you must implement some sort of memory management policy to avoid overflowing the virtual machine memory carelessly. You will be evaluated on the ability of your program to solve various puzzles: maybe your program cannot find a solution from ALL configurations, but maybe it can find a solution for a configuration with only 20 white pixels on it, or only 10 white pixels,… Examples on various configurations to be solved will be provided by us on the course web page as well. The more complex puzzles you can solve, the better grade you will get. **Note:** if you choose to solve this problem *without explicitly exploring a tree*, you must explain where a *virtual* tree is being explored and *how* it is explored (DFS, BFS, etc).

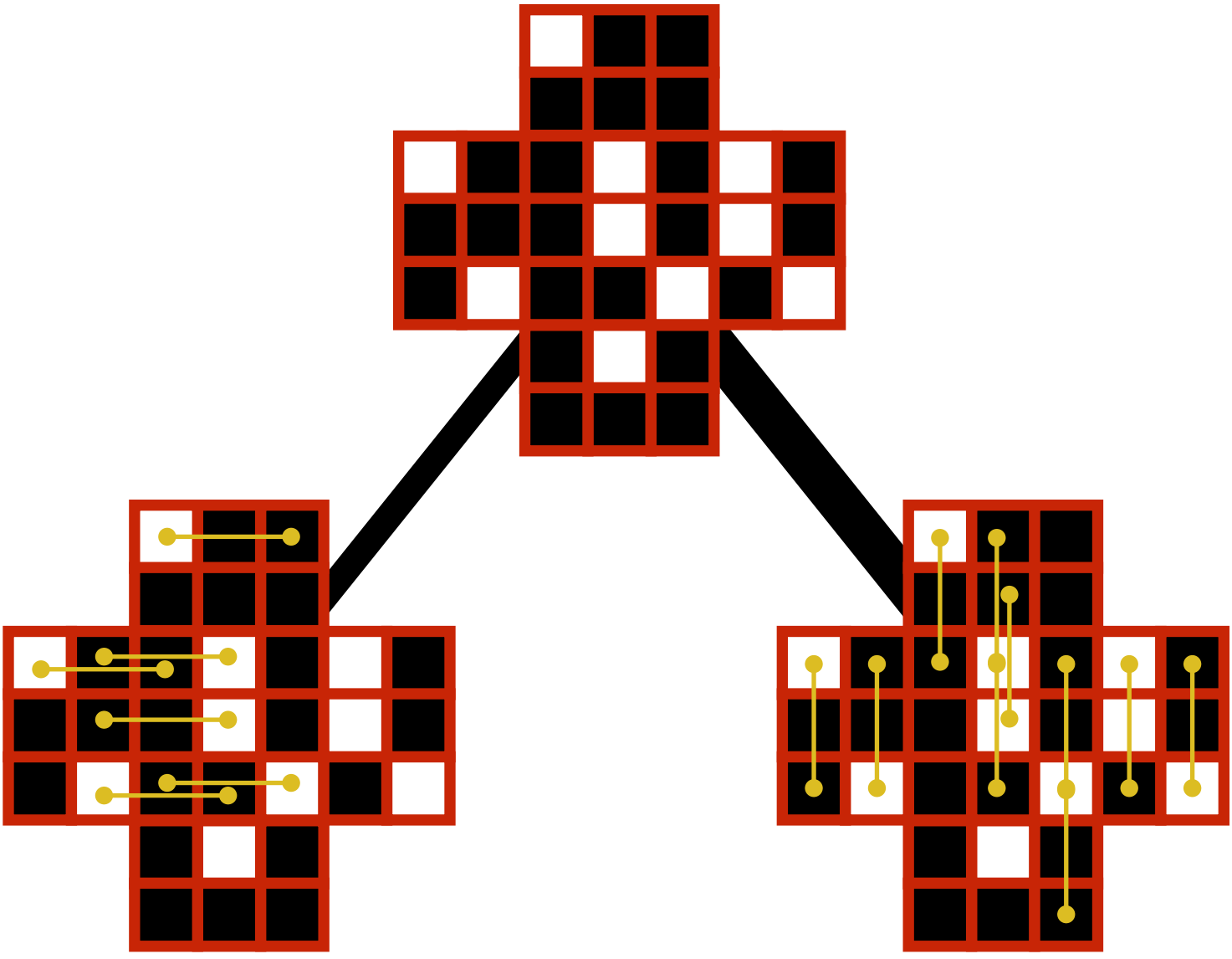
The standard "HI-Q™" puzzle has a unique start-configuration (with 32 white pixels) *opposite* to the *solved* state and only allows W-substitutions:



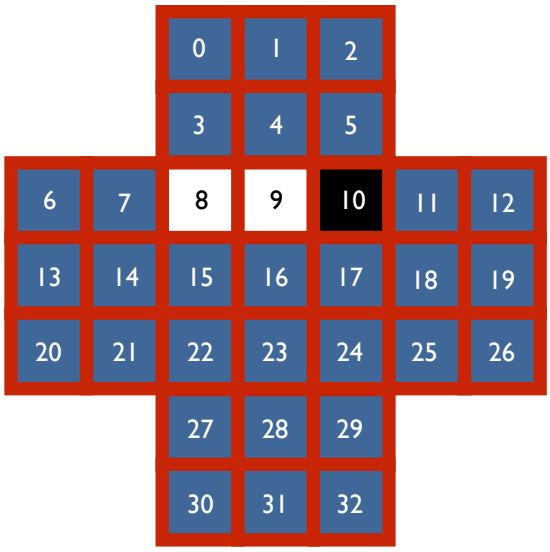
In a first example, we have the *solved* state with its four children :



In a second example, we have the configuration (A) from above with the horizontal substitutions to get 6 of its children and the vertical substitutions to get its other 10 :



List of 38 triplets:



In general, when looking for the children of a given configuration, the following 38 triplets should be considered for applying the substitution rules:

[0 , 1 , 2]
[3 , 4 , 5]

[6 , 7 , 8]
[7, 8, 9]
[8 , 9 ,10]
[9 ,10,11]
[10,11,12]

[13,14,15]
[14,15,16]
[15,16,17]
[16,17,18]
[17,18,19]

[20,21,22]
[21,22,23]
[22,23,24]
[23,24,25]
[24,25,26]

[27,28,29]
[30,31,32]

[12,19,26]
[11,18,25]

[2 , 5 ,10]
[5 ,10,17]
[10,17,24]
[17,24,29]
[24,29,32]

[1 , 4 , 9]
[4 , 9 ,16]
[9 ,16,23]
[16,23,28]
[23,28,31]

[0 , 3 , 8]
[3 , 8 ,15]
[8 ,15,22]
[15,22,27]
[22,27,30]

[7 ,14,21]
[6 ,13,20]