Output: A R1CS r consisting of w three vectors, and the length of each vector is l
Input: the list res of RNode in the created graph

```
    res ← ∅
  for i ← 1 to w do
    res.append(new rightnode)
  end for
  for z ← 1 to l do
    node_a ← None
    cons ← z^th constraint in r
    for i ← 0 to l do
      node_a ← None
      if cons.a[i] ≠ 0 then
        if i == 0 then
          node_a ← CreateConstNode(cons.a[i])
          res.append(node_a)
        else
          tmp ← CreateConstNode(cons.a[i])
          node_a ← Multiple(tmp, res[i])
          res.append(node_a)
          res.append(tmp)
        end if
        for j ← 0 to l do
          node_b ← None
          if cons.b[j] ≠ 0 then
            if j == 0 then
              node_b ← CreateConstNode(cons.b[j])
              res.append(node_b)
            else
              tmp ← CreateConstNode(cons.b[j])
              node_a ← Multiple(tmp, res[j])
              res.append(node_b)
              res.append(tmp)
            end if
            if i and j are the indices corresponding to the last non-zero elements in the constraint. then
              if only one element in the cons.c is not zero then
                if leftnode == None then
                  result node ← corespondingnodeofnone − zeroelementinc
                  result node.operation ← Multiple
                  result node.father ← {node_a, node_b}
                  node_a.child.append{result node}
                  node_b.childappend{result node}
                else
                  rightnode ← Multiple(node_a, node_b)
                  res.append(rightnode)
                  result node ← corespondingnodeofnone − zeroelementinc
                  result node.operation ← Add
                  result node.father ← {leftnode, rightnode}
                  leftnode.child.append{result node}
                  rightnode.child.append{result node}
                end if
              else
                for k ← 0 to l do
                  if cons.c[k] ≠ 0 then
                    if i == 0 then
                      node_c ← CreateConstNode(cons.c[k])
                      res.append(node_c)
                    else
                      tmp ← CreateConstNode(cons.c[k])
                      node_c ← Multiple(tmp, res[k])
                      res.append(node_c)
                      res.append(tmp)
                    end if
                    if rightnode == None then
                      rightnode ← node_c
                    else
```

$rightnode \leftarrow Add(rightnode, node_c)$;
$res.append(rightnode)$
end if
end if
end for
if $leftnode ==$ **None then**
$leftnode \leftarrow Multiple(node_a, node_b)$
$res.append(leftnode)$
else
$tmp \leftarrow Multiple(node_a, node_b)$
$leftnode \leftarrow Add(leftnode, tmp)$
$res.append(leftnode)$
$res.append(tmp)$
end if
end if
$gotonextconstraint$
else
if $leftnode ==$ **None then**
$leftnode \leftarrow Multiple(node_a, node_b)$
$res.append(leftnode)$
else
$tmp \leftarrow Multiple(node_a, node_b)$
$leftnode \leftarrow$ Add(leftnode, tmp)
$res.append(leftnode)$
$res.append(tmp)$
end if
end if
end if
end for
end if
end for
end for

Output: the root RNode of the tile to be chosen and a flag
Input: set of the edges of the chosen tile, s

function GetTile(r, flag, s)
  if r has no predecessor nodes **then**
    return
  end if
  if flag == False & r.opretion == Multiple & both father nodes of r represents constant value **then**
    return
  end if
  if flag == True & r.opretion == Multiple & both father nodes of r represent variables **then**
    $s.add(< r, father_{left} >)$
    $s.add(< r, father_{right} >)$
    return
  end if
  if r.operation == Multiple **then**
    $s.add(< r, father_{left} >)$
    $s.add(< r, father_{right} >)$
    if $father_{left}$ represents constant value **then**
      $GetTile(r, false, father_{left})$
    end if
    if $father_{right}$ represents constant value **then**
      $GetTile(r, false, father_{right})$
    end if
  else
    $s.add(< r, father_{left} >)$
    $s.add(< r, father_{right} >)$
    $GetTile(r, false, father_{left})$
    $GetTile(r, false, father_{right})$
  end if
end function