# 1. Get Started!

This document is an introduction to GPT repository v0.4.2.

The repository contains python codes translation of the APL codes from **the textbook** *Grenander, U. General Pattern Theory: A Mathematical Study of Regular Structures. Oxford Mathematical Monographs. Clarendon Press, 1993. ISBN 9780198536710*.

Before proceeding, you should read the textbook’s chapter 1 and 4. Chapter 4 is where APL codes start to be used for pattern generation, but you need chapter 1 to understand this whole framework. Chapter 4 codes are the focus of this document. And we will refer to *tutorial\_getting\_started* series in notebook folder.

Check out also the condensed version of the textbook “A note on pattern theory.pdf” in this [link](https://www.researchgate.net/publication/349318945_A_Note_on_General_Pattern_Theory?channel=doi&linkId=602a72a9a6fdcc37a82ab8ae&showFulltext=true) (still work in progress, but sufficient for current repo).

# 2. Quick Check

You may like to go straight to the python code. If you choose to do so, ignore the APL codes here, but at least read the description of what each function like COMPENV does. The python codes will name the functions similarly so that it is easy to read.

To learn APL, we recommend use https://tryapl.org/ for the bare basics. It is incomplete, but useful for beginner. I used the software from https://www.dyalog.com/ on Windows to run APL. It works fine.

The python codes we are looking at here are in the folder src\APL\_to\_python.

Let us start.

**What is this about?**

In summary, we have generators from *generator space* placed in a configuration (e.g. a square lattice) and we are evolving them using growth functions to form patterns.

## CODE function

From textbook p.198. We call it *code\_generator(i,k,nBSG)* in python. We are often dealing with , but sometimes we want to know what species it is and which transformation it went through. Formally, it takes elements and return .

Check the following against the python version, CODE in tutorial\_getting\_started0001.ipynb

|  |  |
| --- | --- |
| **APL code** | **Result** |
| NBSG←4  x←2 2 ⍴ ⍳4  y←2 2 ⍴ (2 × ⍳4)  ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  x CODE y | 2 8 14 20 |

This is how we will proceed. We present APL code and refer you to the python code in the *notebooks* folder for double checking.

## COMPENV

From textbook p.199. It computes the environment from the configuration by placing the bond values due to the generators in the configuration to their respective positions.

Compare to python version in tutorial\_getting\_started0001.ipynb COMPENV.

|  |  |
| --- | --- |
| J←4 2⍴1 0 0 1 ¯1 0 0 ¯1  L←5 ⋄ NJ←4  GE←8 4 ⍴⍳32  CE←L L ⍴ 1 ⋄ CE[3;2]←5  ∇ Z←OPPOSITE K [1] Z←1+NJ|¯1+K+NJ÷2 [2] ∇  ∇ COMPENV;K [1] ⍝ COMPUTES ENV ARRAY FROM GIVEN GLOBAL VARIABLE CE [2] ⍝ RESULT CALLED ENV IS J-ARRAY WHOSE FIRST SUBSCRIPT IS BOND COORDINATE [3] ENV←(NJ,L,L)⍴0 [4] K←1 [5] LOOP: ENV[K;;]←GE[J[K;1]⊖J[K;2]⌽CE; OPPOSITE K] [6] K←K+1 [7] →(K≤NJ)/LOOP [8] ∇  COMPENV  ENV | 3 3 3 3 3  3 19 3 3 3  3 3 3 3 3  3 3 3 3 3  3 3 3 3 3    4 4 4 4 4  4 4 4 4 4 20 4 4 4 4  4 4 4 4 4  4 4 4 4 4    1 1 1 1 1  1 1 1 1 1  1 1 1 1 1  1 17 1 1 1  1 1 1 1 1 |
| Some remarks:   1. The APL code here is standalone, including all the initialization. 2. You may see functions like OPPOSITE being repeated in the following codes too. 3. OPPOSITE computes the index of the bond spatially at its opposite\*. So, if we have top/right/bottom/left bonds of a generator, inputting index for top into OPPOSITE will give you index of bottom.   \*We confine the descriptions to the contexts used in that section of the textbook. | |

## GROWTH1

From textbook p.205. This is the dynamic rule used in the first few examples of the textbook.

Compare to python version in tutorial\_getting\_started0001.ipynb GROWTH1.

|  |  |
| --- | --- |
| NBSG←4  L←5⋄ NJ←4  AGE←L L⍴0  ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  ∇ B←GROWTH1 A [1] CHANGE←(A≤NBSG) ∧1=+/[1]ENV>0 [2] D←+/[1]ENV [3] E←(⍳NJ)+.×ENV>0 [4] B←D CODE E [5] B←(A×1-CHANGE)+CHANGE×B [6] AGE←AGE×1-CHANGE [7] ∇  ENV←(NJ,L,L)⍴0  ENV[1;2;2]←ENV[2;3;3]←ENV[4;2;3]←2 ⋄ ENV  CE←L L ⍴ 1 ⋄ CE[2;3]←5 ⋄ CE  B←GROWTH1 CE  B  'AGE'⋄AGE | B 1 1 1 1 1 1 5 5 1 1 1 1 6 1 1 1 1 1 1 1 1 1 1 1 1 'AGE'⋄AGE AGE 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 |

## COMPNONZERO

From textbook p. 200. This is a display utility.

Compare to python version in tutorial\_getting\_started0001.ipynb COMPNONZERO.

|  |  |
| --- | --- |
| ∇ COMPNONZERO M;M1;N;P;Q [1] ⍝ REDUCES MATRIX M … [2] M1←M [3] M←M≠0 [4] N←(⍴M)[1] [5] P←( ∨/M) ⍳1 [6] Q←(⌽ ∨/M)⍳1 [7] V1←¯1+P+⍳2+N-P+Q [8] N←(⍴M)[2] [9] P←( ∨/[1]M)⍳1 [10] Q←(⌽ ∨/[1]M)⍳1 [11] V2←¯1+P+⍳2+N-P+Q [12] Z←M1[V1;V2] [13] ∇  M← 5 5 ⍴ 0 0 2 ⋄M ⋄ COMPNONZERO(M) ⋄' >'⋄ Z  M←4 3 ⍴ 0 1 0⋄M ⋄ COMPNONZERO(M) ⋄' >'⋄ Z  M[1;1]←3⋄M ⋄ COMPNONZERO(M) ⋄' >'⋄ Z | M← 5 5 ⍴ 0 0 2 ⋄M ⋄ COMPNONZERO(M) ⋄' >'⋄ Z 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0  > 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 0 2 0 M←4 3 ⍴ 0 1 0⋄M ⋄ COMPNONZERO(M) ⋄' >'⋄ Z  0 1 0 0 1 0 0 1 0 0 1 0  > 1 1 1 1 M[1;1]←3⋄M ⋄ COMPNONZERO(M) ⋄' >'⋄ Z 3 1 0 0 1 0 0 1 0 0 1 0  > 3 1 0 1 0 1 0 1 |

## DEVELOP

From textbook p. 200. This is the function that uses COMPENV and GROWTH (the first few examples use GROWTH1) to evolve a configuration.

Compare to python version in tutorial\_getting\_started0001.ipynb DEVELOP.

|  |  |
| --- | --- |
| ∇ Z←OPPOSITE K [1] Z←1+NJ|¯1+K+NJ÷2 [2] ∇  ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  ∇ Z←DECODE N;I;K [1] I←1+⌊(N-1)÷NBSG [2] K←N-NBSG×I-1 [3] Z←I,[0.5]K [4] ∇  ∇ COMPENV;K [1] ⍝ COMPUTES ENV ARRAY FROM GIVEN GLOBAL VARIABLE CE [2] ⍝ RESULT CALLED ENV IS J-ARRAY WHOSE FIRST SUBSCRIPT IS BOND COORDINATE [3] ENV←(NJ,L,L)⍴0 [4] K←1 [5] LOOP: ENV[K;;]←GE[J[K;1]⊖J[K;2]⌽CE; OPPOSITE K] [6] K←K+1 [7] →(K≤NJ)/LOOP [8] ∇    ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  ∇ B←GROWTH1 A [1] CHANGE←(A≤NBSG) ∧1=+/[1]ENV>0 [2] D←+/[1]ENV [3] E←(⍳NJ)+.×ENV>0 [4] B←D CODE E [5] B←(A×1-CHANGE)+CHANGE×B [6] AGE←AGE×1-CHANGE [7] ∇  ∇ COMPNONZERO M;M1;N;P;Q [1] ⍝ REDUCES MATRIX M … [2] M1←M [3] M←M≠0 [4] N←(⍴M)[1] [5] P←( ∨/M) ⍳1 [6] Q←(⌽ ∨/M)⍳1 [7] V1←¯1+P+⍳2+N-P+Q [8] N←(⍴M)[2] [9] P←( ∨/[1]M)⍳1 [10] Q←(⌽ ∨/[1]M)⍳1 [11] V2←¯1+P+⍳2+N-P+Q [12] Z←M1[V1;V2] [13] ∇  ∇ DEVELOP MORE; T1 [1] ⍝ computes more iterations and displays every pth [2] T1←0 [3] LOOP1:COMPENV [4] CE←⍎GROW,' CE' [5] AGE←AGE+1 [6] T←T+1 [7] T1←T1+1 [8] →(0≠P|T1)/LOOP2 [9] '' [10] 'ITERATION NO. ',⍕T1 [11] COMPNONZERO CE>NBSG [12] ALPH[(DECODE CE[V1;V2])[1;;]] [13] ⍳0 [14] LOOP2:→(T1<MORE)/LOOP1 [15] ∇  NBSG←4  J←4 2⍴1 0 0 1 ¯1 0 0 ¯1  GE←4 4 ⍴ 0 ⋄ GE2←4 4 ⍴ 2 0 2 0 0 2 0 2 ⋄ GE← GE,[1]GE2  L←5 ⋄ NJ←4  CE←L L ⍴ 1 ⋄ CE[3;2]←5  AGE←L L⍴0  MORE←4⋄P←1  T←0 ⍝ not sure where this T is used though  ALPH←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'  GROW←'GROWTH1'  DEVELOP MORE | ITERATION NO. 1 A A A   ITERATION NO. 2 A A A A A   ITERATION NO. 3 A A A A A |

## Other related codes

The above are the few essential codes we need. We leave here some of the related codes with many verbose details that you may want to skip for now.

|  |
| --- |
| This is a niladic function (no argument), with I being the only variable specified as local. NJ and J are not local. Hence, if we run SETTOPOLOGY and input the following successively (separated by //) 4// 1 0// 0 1// 0 ¯1// ¯1 0, the variable NJ is set to 4 and J will be the specified matrix. |
| ∇ SETTOPOLOGY; I  'HOW MANY NEIGHBORS?'  I ←1  NJ←⎕  J←(NJ,2)⍴0  LOOP: 'ENTER (X,Y) COORDS OF NEIGHBOR NO. ',⍕I  J[I;] ←⎕  I←I+1  →(I≤NJ)/LOOP  ∇ |

G0 is set to size (NG0+1,NJ). There are +1 extra generator for a zero row (they call it the empty generator). NJ is the number of neighbors per generator, so for each generator (each row), NJ bond values are stored to link up to that NJ neighbors.

Empty generator in G0 is indexed by 1, in GE denoted as 1, 2, …, NBSG (after applying all group elements).

In the following, we set G0 by specifying no. of generator=1! This means the empty generator is auto-generated.

The first row 0 0 0 0 is the empty generator, which looks like ( 0 ) in the diagram form. The second row, 2 – (1) – 2 is generator 2 in G0. “Recall that the row subscript is one bigger”, “thus the bond value is 2 not 1”. The entry of G0 is 1 bigger than the actual index, so that 0 can represent no bond.

|  |  |
| --- | --- |
| ∇ SETG0; ANS; I  'NO. OF GENERATORS IN GO?'  NG0←⎕  AGE←(L, L) ⍴0  'ARE BOND VALUES IDENTICAL TO GENERATORS?'  ANS←⎕  G0←((NG0+1),NJ) ⍴0  I←2  ALPHA←(NG0+1) ⍴0  LOOP1: →(~ANS)/LOOP2  G0[I;] ←NJ⍴I  'ALPHA VALUE FOR GENERATOR NO. ',⍕I  ALPHA[I] ←⎕  I←I+1  →(I≤NGO+1)/LOOP1  →0  LOOP2: 'BOND VALUE VECTOR FOR GENERATOR NO. ' ,⍕I  G0[I;] ←⎕  'ALPHA VALUE FOR GENERATOR NO. ' ,⍕I-1  ALPHA[I] ←⎕  I←I+1  →(I ≤NG0+1)/LOOP2  ∇ | Let’s try reading that code.  Get number of generators NG0.  Get Boolean ANS. # bond values identical  Get zero matrix G0 size (NG0+1, NJ)  Get zero vector ALPHA size (NG0+1)  If ANS: # bond values identical  # First loop:  for i in [2,3,…,NGO+1]:  Set G0 row I **to all I**  Set ALPHA row I to user input  I++  else  # Second loop:  for i in [2,3,…,NGO+1]:  Set G0 row I **to user input**  Set ALPHA row I to user input  I++    Remark: the only difference between the two loops are the input to G0, which presumably stores the “bond values". |

|  |
| --- |
| ∇ SETBSG;I  'ORDER OF BOND STRUCTURE GROUP? '  I←2  NBSG←⎕  BSG←(NBSG,NJ) ⍴ ⍳NJ  →(NBSG=1)/OUT  LOOP: 'PERMUTATION VECTOR FOR GROUP ELEMENT NO. ',⍕I  BSG[I;] ←⎕  I←I+1  →(I≤NBSG)/LOOP  OUT:  ∇ |

|  |  |
| --- | --- |
| ∇ Z←OPPOSITE K [1] Z←1+NJ|¯1+K+NJ÷2 [2] ∇ | Don’t worry, this is not wrong. It is NOT the inverse element in the group sense. It is OPPOSITE in the spatial sense (up vs down, left vs right)  NJ=4 will return 3 4 1 2 for 1 2 3 4 respectively because the arrangement is like  1  4 2  3 |
| Other example, set NJ=8  OPPOSITE ⍳8 returns  5 6 7 8 1 2 3 4  So, compared to the input  1 2 3 4 5 6 7 8  the opposite in cyclical arrangement is indeed correct  8 1 2  7 3  6 5 4 | Python code  for i in range(1,8+1):  print(opposite(i,8))  yields the same output (for double checking)  For NJ=6 we get  4 5 6 1 2 3  1 2 3 4 5 6  1  6 2  5 3  4 |

|  |  |
| --- | --- |
| ∇ SETEXTG0;I;K  [1] ⍝ EXTENSION  [2] NGE←NBSG×NG0+1  [3] GE←(NGE,NJ)⍴0  [4] K←I←1  [5] LOOP:GE[K+(¯1+I)×NBSG;]←G0[I;BSG[K;]]  [6] K←K+1  [7] →(K≤NBSG)/LOOP  [8] K←1  [9] I←I+1  [10] →(I≤NG0+1)/LOOP  ∇ | Let’s try reading the code  NGE = NBSG\*(NG0+1)  GE = zeros of shape (NGE,NJ)  for I in [1,…,NG0+1]:  for K in [1,…,NBSG]:  # Apply all group elements to all groups in G0  GE[K+(I-1)\*NBSG;] = G0[I;BSG[K;]] |
| An example is  The no. of rows is the number elements in the initial generator space.  The no. of columns is the number neighbors or arity. |  |

|  |  |
| --- | --- |
| ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇ | |
| ∇ Z←DECODE N;I;K [1] I←1+⌊(N-1)÷NBSG [2] K←N-NBSG×I-1 [3] Z←I,[0.5]K [4] ∇ | Line [3] might look strange. But [0.5] is to specify which axis to concatenate I and K along. See wiki; the notation is unusual. |

Section 4.2.1.2

***It’s not defined in the textbook***, so let’s just define one.

UPDATE: by inspecting COMPENV code, indeed the number 5 in CE in the textbook corresponds to the 5th row in GE, or, in another words, ***the 5th generators out of all the available 8 generators***.

|  |
| --- |
| ∇ SETCINIT [1] CE← L L ⍴ 1 [2] LOOP: 'ANY (MORE) NON-EMPTY SITES?' [3] MORE←⎕ [4] →(~MORE)/0 [5] '(X,Y) COORDS OF GENERATOR?' [6] XY←⎕ [7] ⍝ THIS PART NOT SURE [8] 'GENERATOR NO?' [9] GEN\_NUM←⎕ [10] 'ELEMENT NUMBER IN BSG?' [11] BSG\_NUM←⎕ [12] CE[XY[2];XY[1]]←(NBSG× GEN\_NUM)+BSG\_NUM [13] →LOOP [14] ∇ |

# 3. Examples

After going through the quick checks, let us replicate more examples in **section 4.2.2.1 and 4.2.2.7** of the textbook. The python code related to this is in src/APL\_to\_python/simple\_algo\_section4.2.2.py.

Choose one of the APL initialization code below, and then run the **block of DEVELOP code** in the next table. Compare this with results obtained using the python code in the right-most column. Note that we shortcut the variables initialization that needs manual input below.

|  |  |  |
| --- | --- | --- |
|  | **APL Initialization code** |  |
| 1. | G0← 2 4 ⍴ 0 0 0 0 2 0 2 0⋄ TMAX ← 4 ⋄L←10  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | python simple\_algo\_section4.2.2.py --example\_n 1 |
| 2. | G0←4 4 ⍴ 0 0 0 0 3 0 3 0 4 0 4 0 0 0 0 0 ⋄ TMAX ← 6 ⋄L←10  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | python simple\_algo\_section4.2.2.py --example\_n 2 |
| 3. | G0←6 4 ⍴ 0 0 0 0 3 0 3 0 4 0 4 0 5 0 5 0 0 6 0 6 6 0 6 0 ⋄ TMAX ← 7 ⋄L←10  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | python simple\_algo\_section4.2.2.py --example\_n 3 |
| 4. | G0←2 4 ⍴ 0 0 0 0 2 2 2 2 ⋄ TMAX ← 16 ⋄L←32  CE←L L ⍴ 1 ⋄ CE[16;16]←5 | python simple\_algo\_section4.2.2.py --example\_n 4 |
| 5. | G0←3 4 ⍴ 0 0 0 0 3 3 3 3 0 0 3 3 ⋄ TMAX ← 4 ⋄L←10  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | python simple\_algo\_section4.2.2.py --example\_n 5 |
| 6. | G0←6 4 ⍴ 0 0 0 0 3 0 4 0 0 0 0 5 0 6 0 0 0 3 0 0 0 0 0 4 ⋄ TMAX ← 8 ⋄L←10  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | python simple\_algo\_section4.2.2.py --example\_n 6 |
| 7. | G0←7 4 ⍴ 0 0 0 0 3 0 4 7 0 0 0 5 0 6 0 0 0 3 0 0 0 0 0 4 7 0 7 0 ⋄ TMAX ← 4 ⋄L←10  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | python simple\_algo\_section4.2.2.py --example\_n 7 |
| 8. | G0←5 4 ⍴ 0 0 0 0 3 3 3 3 3 0 4 0 4 0 5 0 0 3 0 0 ⋄ TMAX ← 9⋄L←15  CE←L L ⍴ 1 ⋄ CE[8;8]←5 | python simple\_algo\_section4.2.2.py --example\_n 8 |
| 9. | G0←5 4 ⍴ 0 0 0 0 3 0 3 0 4 0 4 0 3 5 3 5 5 0 5 0 ⋄ TMAX ← 6⋄L←15  CE←L L ⍴ 1 ⋄ CE[8;8]←5 | python simple\_algo\_section4.2.2.py --example\_n 9 |

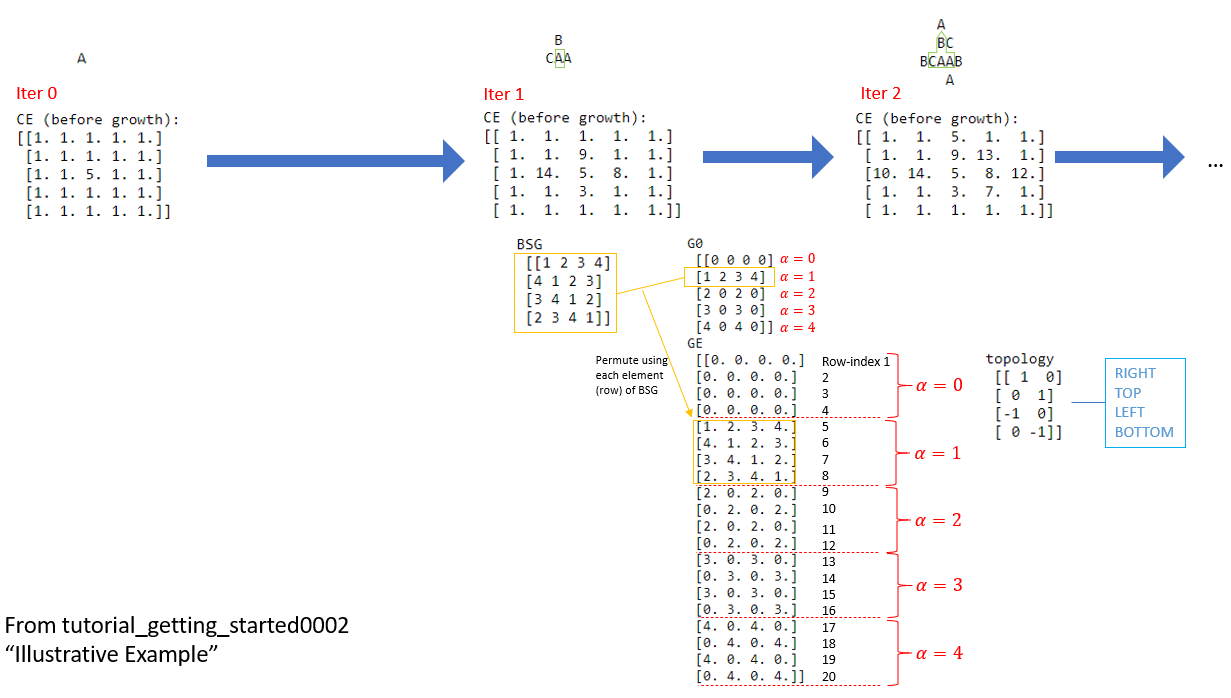
|  |  |
| --- | --- |
| **Block of DEVELOP code** | Some results |
| J←4 2⍴1 0 0 1 ¯1 0 0 ¯1  NJ←4 ⋄ NBSG←4  BSG ← 4 4 ⍴ 1 2 3 4 4 1 2 3 3 4 1 2 2 3 4 1  AGE←L L⍴0  NG0← (⍴ G0)[1] - 1  ∇ Z←OPPOSITE K [1] Z←1+NJ|¯1+K+NJ÷2 [2] ∇  ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  ∇ Z←DECODE N;I;K [1] I←1+⌊(N-1)÷NBSG [2] K←N-NBSG×I-1 [3] Z←I,[0.5]K [4] ∇  ∇ SETEXTG0;I;K  [1] ⍝ EXTENSION  [2] NGE←NBSG×NG0+1  [3] GE←(NGE,NJ)⍴0  [4] K←I←1  [5] LOOP:GE[K+(¯1+I)×NBSG;]←G0[I;BSG[K;]]  [6] K←K+1  [7] →(K≤NBSG)/LOOP  [8] K←1  [9] I←I+1  [10] →(I≤NG0+1)/LOOP  ∇  ∇ COMPENV;K [1] ⍝ COMPUTES ENV ARRAY FROM GIVEN GLOBAL VARIABLE CE [2] ⍝ RESULT CALLED ENV IS J-ARRAY WHOSE FIRST SUBSCRIPT IS BOND COORDINATE [3] ENV←(NJ,L,L)⍴0 [4] K←1 [5] LOOP: ENV[K;;]←GE[J[K;1]⊖J[K;2]⌽CE; OPPOSITE K] [6] K←K+1 [7] →(K≤NJ)/LOOP [8] ∇  T←0 ⍝ not sure where this T is used though  ALPH←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'  ∇ DEVELOP MORE; T1 [1] ⍝ computes more iterations and displays every pth [2] T1←0 [3] LOOP1:COMPENV [4] CE←⍎GROW,' CE' [5] AGE←AGE+1 [6] T←T+1 [7] T1←T1+1 [8] →(0≠P|T1)/LOOP2 [9] '' [10] 'ITERATION NO. ',⍕T1 [11] COMPNONZERO CE>NBSG  [12] ⍝ CE ⍝ UNCOMMENT THIS FOR VERBOSE [13] ALPH[(DECODE CE[V1;V2])[1;;]] [14] ⍳0 [15] LOOP2:→(T1<MORE)/LOOP1 [16] ∇  ∇ COMPNONZERO M;M1;N;P;Q [1] ⍝ REDUCES MATRIX M  [2] M1←M [3] M←M≠0 [4] N←(⍴M)[1] [5] P←( ∨/M) ⍳1 [6] Q←(⌽ ∨/M)⍳1 [7] V1←¯1+P+⍳2+N-P+Q [8] N←(⍴M)[2] [9] P←( ∨/[1]M)⍳1 [10] Q←(⌽ ∨/[1]M)⍳1 [11] V2←¯1+P+⍳2+N-P+Q [12] Z←M1[V1;V2] [13] ∇  ∇ B←GROWTH1 A [1] CHANGE←(A≤NBSG) ∧1=+/[1]ENV>0 [2] D←+/[1]ENV [3] E←(⍳NJ)+.×ENV>0 [4] B←D CODE E [5] B←(A×1-CHANGE)+CHANGE×B [6] AGE←AGE×1-CHANGE [7] ∇  GROW←'GROWTH1'  P←1  SETEXTG0  DEVELOP TMAX | From --example\_n 3  ITERATION NO. 1 B A B  …  ITERATION NO. 7 EEEEDEEEE  C   B   A   B   C  EEEEDEEEE |
| From --example\_n 4  ITERATION NO. 1  A  AAA  A  …  ITERATION NO. 6  A   A   AAAAA   A   A AAA A   A A A A A  AAAAAAAAAAAAA  A A A A A   A AAA A   A   AAAAA   A   A  … |
| From --example\_n 6  ITERATION NO. 1  C  A  B  …  ITERATION NO. 5  C  CE  CE  A  BD  BD  B  ITERATION NO. 6  CE  CE  CE  A  BD  BD  BD |
| From --example\_n 9  ITERATION NO. 1  B  A  B  ITERATION NO. 2  C  B  A  B  C  …  ITERATION NO. 6  C  B  DDCDD  B  DDDDCDDDD  B  A  B  DDDDCDDDD  B  DDCDD  B  C |

# 4. More details

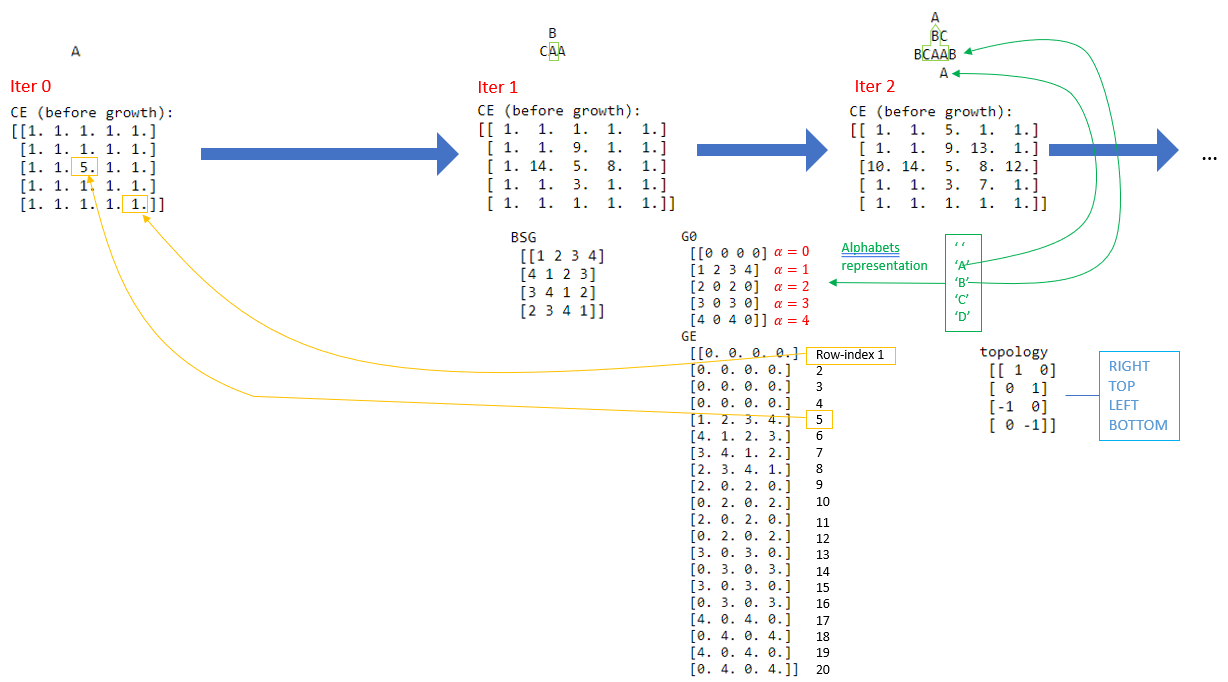
After seeing the codes in quick check and the examples, we may want to look a little bit more into the details. In particular, we see what ENV computed by COMPENV looks like, and other intermediate values computed.

## Illustrative Example

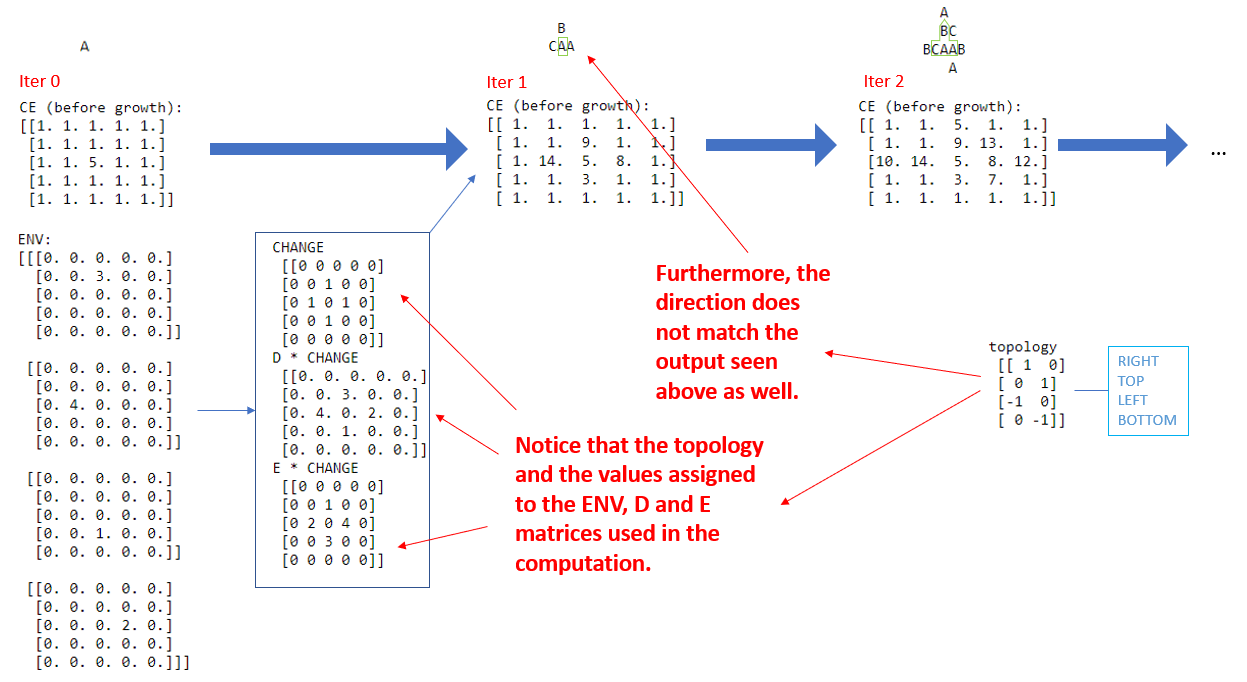
The python code is run in tutorial\_getting\_started0002.ipynb Illustrative Example. We see the following. First, these are the GE generated from G0 and BSG and their indices. Topology matrix is also shown.



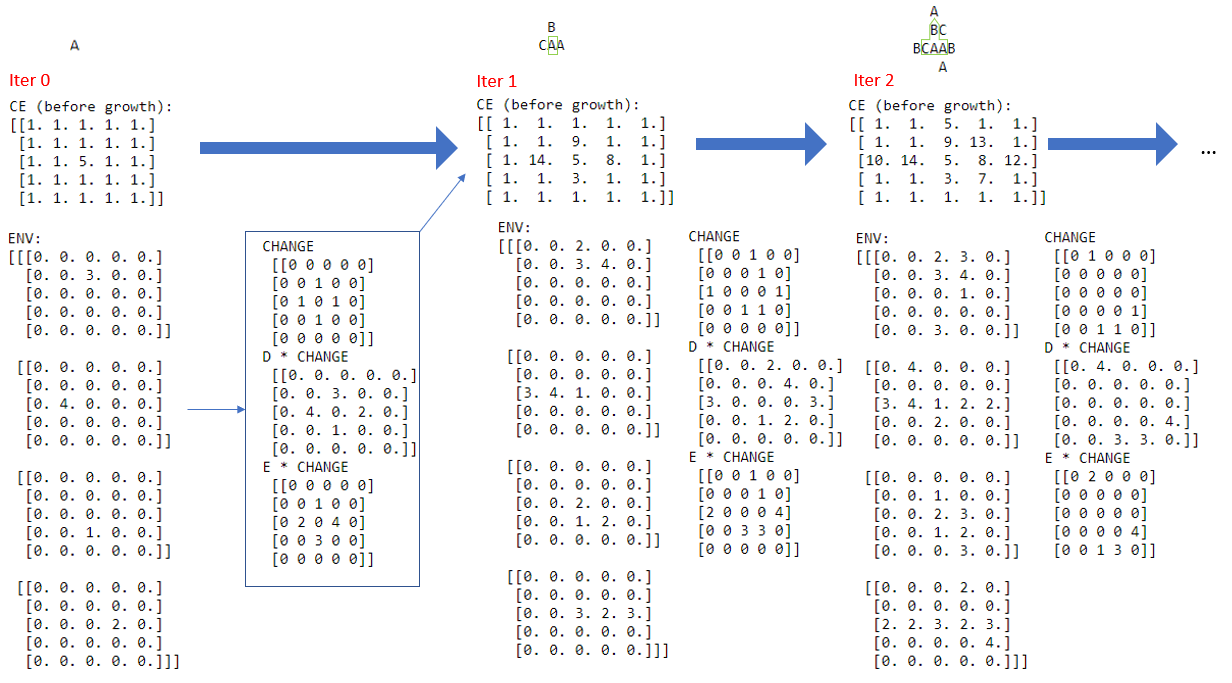
See how the elements in G are placed in the initial configuration CE and then evolved.



Alright, it seems like the RIGHT/TOP/LEFT/BOTTOM placement of bond values do not correspond to what we see. They are actually right, because the directions are defined w.r.t array indices which go from top-left to bottom right.



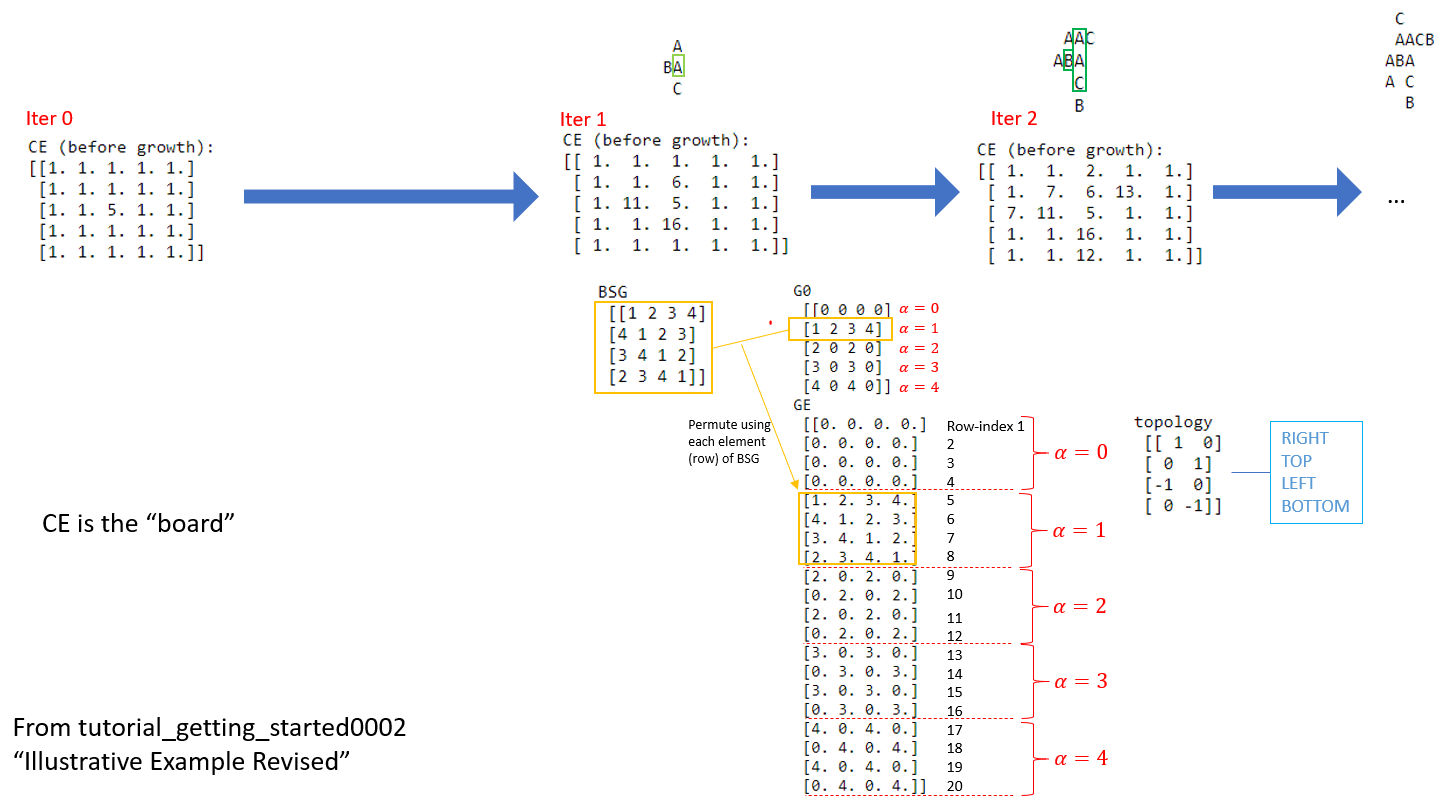
Anyway, here is the more complete version.

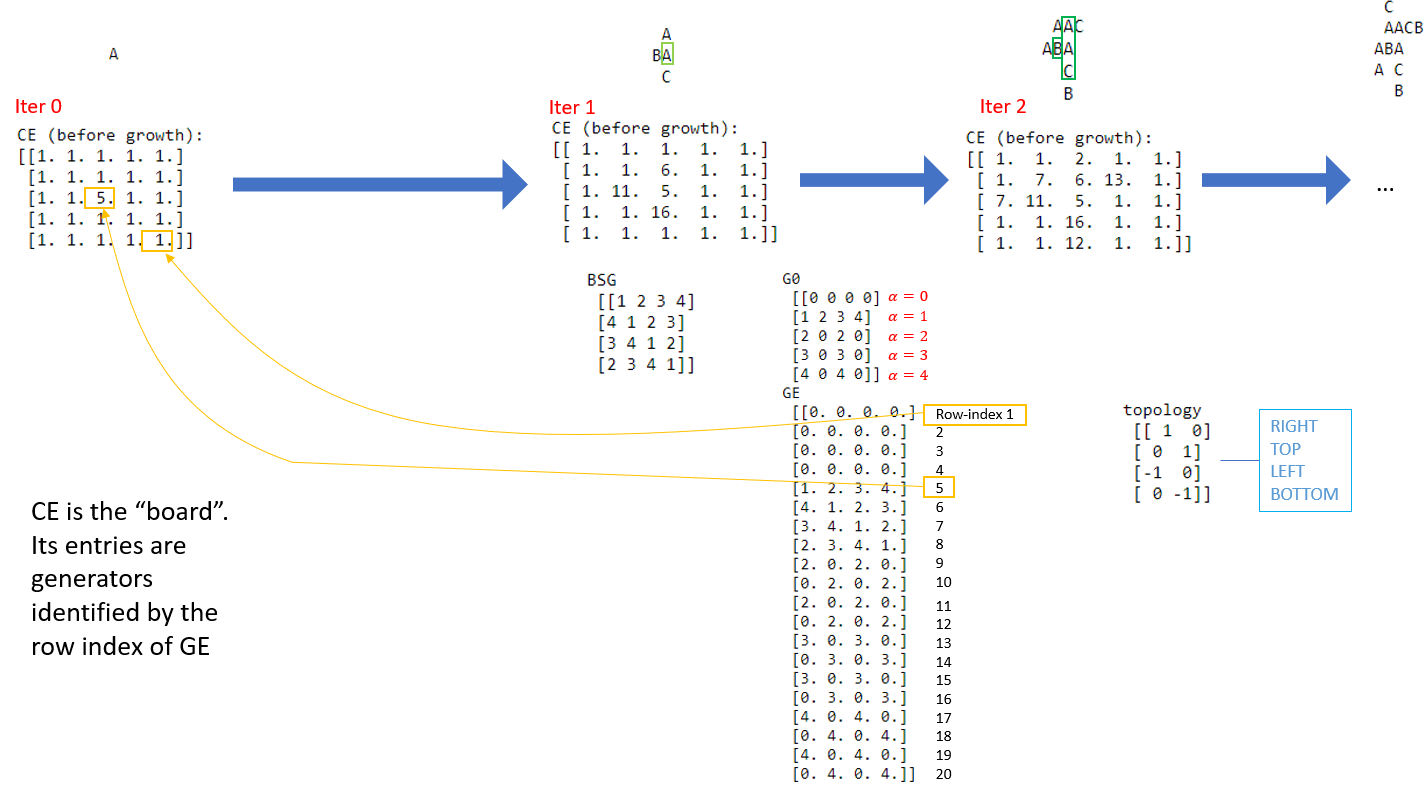


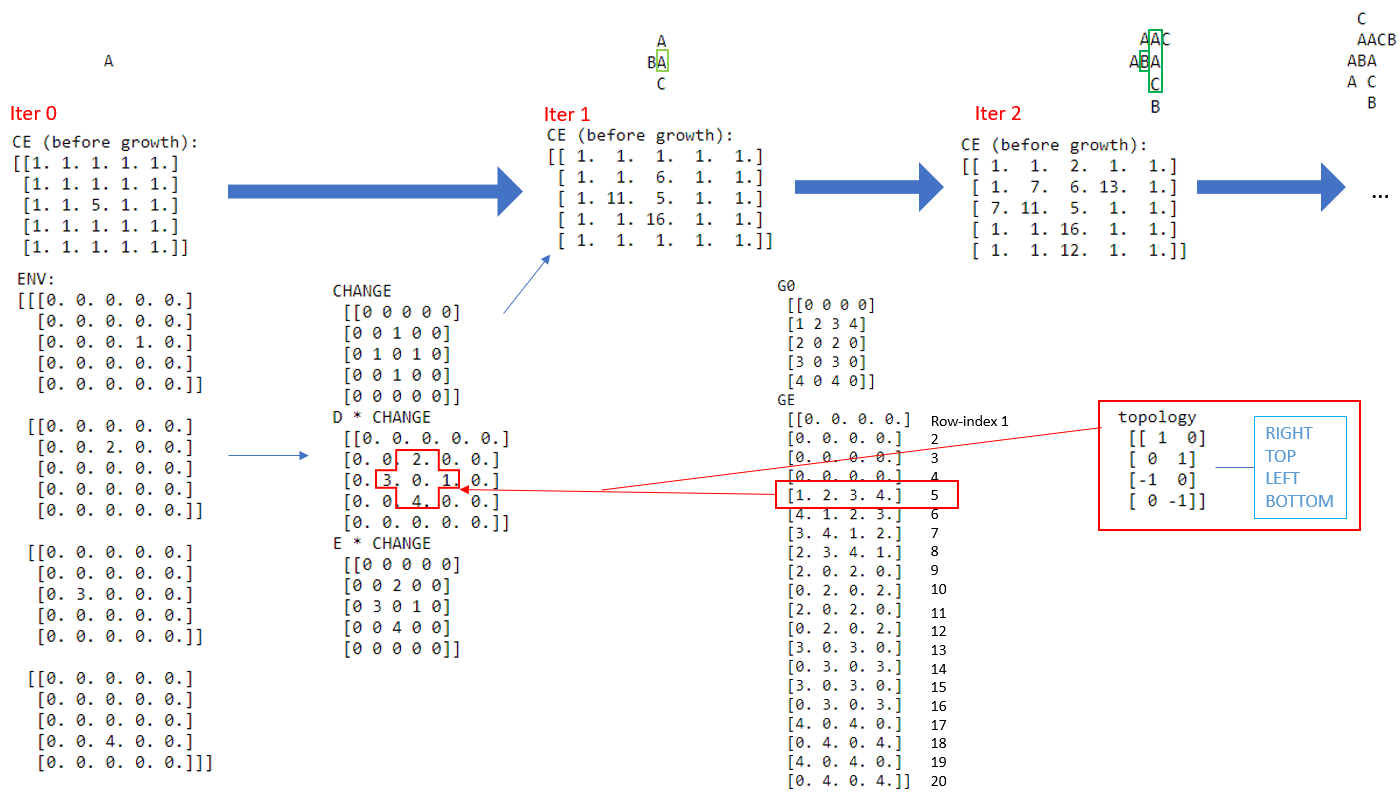
## Revised version

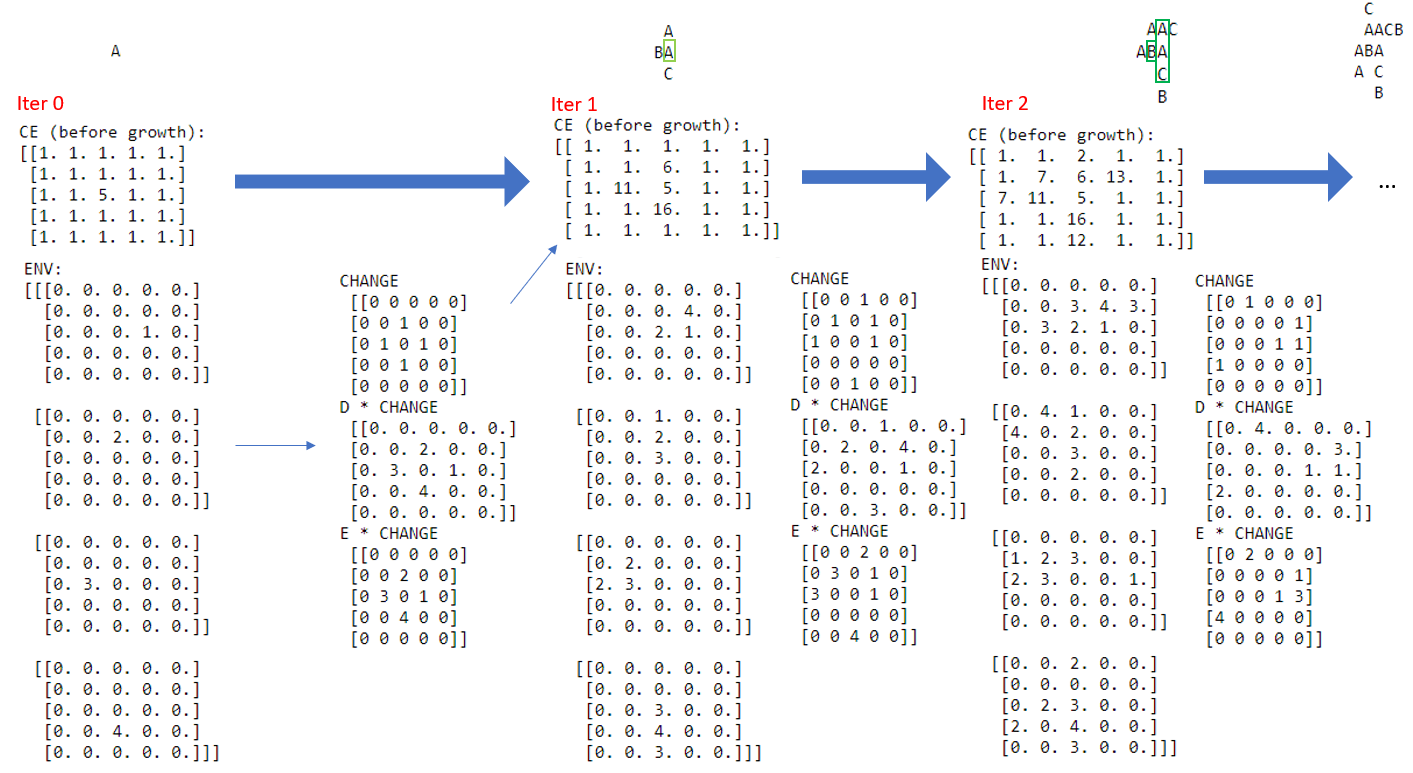
From what we see in the previous section, we create the revised version so that TOP/RIGHT/ BOTTOM/LEFT directions at least match what we see when the array is printed. This will be the one we use in research part of GPT repository (e.g. gpt\_mnist), though we will not use it when we go through the examples in the textbook.

tutorial\_getting\_started0002.ipynb Illustrative Example Revised. Here we have:









# More examples

Python codes not available yet. Please update

## Example with GROWTH 2

From p. 219. Similar to the previous examples but using GROWTH2. This is in **section 4.2.2.8** of the textbook. The python code related to this is in src/APL\_to\_python/simple\_algo\_section4.2.2.py. In the python version, simply run

python simple\_algo\_section4.2.2.py --example\_n 10

|  |  |
| --- | --- |
| J←4 2⍴1 0 0 1 ¯1 0 0 ¯1  L←10 ⋄ NJ←4 ⋄ NBSG←4  AGE←L L⍴0  BSG ← 4 4 ⍴ 1 2 3 4 4 1 2 3 3 4 1 2 2 3 4 1  G0←4 4 ⍴ 0 0 0 0 3 3 3 3 0 0 3 3 0 0 0 0  NG0←3  CE←L L ⍴ 1 ⋄ CE[5;5]←5  ∇ Z←OPPOSITE K [1] Z←1+NJ|¯1+K+NJ÷2 [2] ∇  ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  ∇ Z←DECODE N;I;K [1] I←1+⌊(N-1)÷NBSG [2] K←N-NBSG×I-1 [3] Z←I,[0.5]K [4] ∇  ∇ SETEXTG0;I;K  [1] ⍝ EXTENSION  [2] NGE←NBSG×NG0+1  [3] GE←(NGE,NJ)⍴0  [4] K←I←1  [5] LOOP:GE[K+(¯1+I)×NBSG;]←G0[I;BSG[K;]]  [6] K←K+1  [7] →(K≤NBSG)/LOOP  [8] K←1  [9] I←I+1  [10] →(I≤NG0+1)/LOOP  ∇  ∇ COMPENV;K [1] ⍝ COMPUTES ENV ARRAY FROM GIVEN GLOBAL VARIABLE CE [2] ⍝ RESULT CALLED ENV IS J-ARRAY WHOSE FIRST SUBSCRIPT IS BOND COORDINATE [3] ENV←(NJ,L,L)⍴0 [4] K←1 [5] LOOP: ENV[K;;]←GE[J[K;1]⊖J[K;2]⌽CE; OPPOSITE K] [6] K←K+1 [7] →(K≤NJ)/LOOP [8] ∇  T←0 ⍝ not sure where this T is used though  ALPH←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'  ∇ DEVELOP MORE; T1 [1] ⍝ computes more iterations and displays every pth [2] T1←0 [3] LOOP1:COMPENV [4] CE←⍎GROW,' CE' [5] AGE←AGE+1 [6] T←T+1 [7] T1←T1+1 [8] →(0≠P|T1)/LOOP2 [9] '' [10] 'ITERATION NO. ',⍕T1 [11] COMPNONZERO CE>NBSG [12] ALPH[(DECODE CE[V1;V2])[1;;]] [13] ⍳0 [14] LOOP2:→(T1<MORE)/LOOP1 [15] ∇  ∇ COMPNONZERO M;M1;N;P;Q [1] ⍝ REDUCES MATRIX M  [2] M1←M [3] M←M≠0 [4] N←(⍴M)[1] [5] P←( ∨/M) ⍳1 [6] Q←(⌽ ∨/M)⍳1 [7] V1←¯1+P+⍳2+N-P+Q [8] N←(⍴M)[2] [9] P←( ∨/[1]M)⍳1 [10] Q←(⌽ ∨/[1]M)⍳1 [11] V2←¯1+P+⍳2+N-P+Q [12] Z←M1[V1;V2] [13] ∇  ∇ B←GROWTH1 A [1] CHANGE←(A≤NBSG) ∧1=+/[1]ENV>0 [2] D←+/[1]ENV [3] E←(⍳NJ)+.×ENV>0 [4] B←D CODE E [5] B←(A×1-CHANGE)+CHANGE×B [6] AGE←AGE×1-CHANGE [7] ∇    ∇ B ← GROWTH2 A [1] ⍝ SPECIAL GROWTH DYNAMICS WITH AGE DEPENDENCE [2] B ←GROWTH1 A [3] CHANGE1←(B>NBSG)∧AGE>AGE1 [4] CHANGE2←(B>NBSG)∧AGE=AGE1 [5] B←((1-CHANGE1+CHANGE2)×B)+CHANGE1+13×CHANGE2 [6] ∇  P←1  SETEXTG0  GROW←'GROWTH2'  AGE1←1  DEVELOP 4 | **Not sure if there is any typo in ITER 1 and 2 of the textbook.**  ITERATION NO. 1  B  BAB  B    ITERATION NO. 2  B   BCB  BCCCB  BCB   B    ITERATION NO. 3  B   BCB   BC CB  BC CB  BC CB   BCB   B    ITERATION NO. 4  B   BCB   BC CB   BC CB  BC CB  BC CB   BC CB   BCB   B |

## Example with GROWTH 3 (part 1)

Similar to the previous examples but using GROWTH3 with a few extra functions. This is in **section 4.2.3.1 p. 223** of the textbook. The python code related to this is in src/APL\_to\_python/ simple\_algo\_section4.2.3.py. In the python version, simply run

python simple\_algo\_section4.2.3.py

**Growth 3 part 1 APL code.**

Run these codes successively. The first part is the initial setting, the second part main source code and the third part executes the code.

|  |  |  |
| --- | --- | --- |
| 1. | J←4 2⍴1 0 0 1 ¯1 0 0 ¯1  L←10 ⋄ NJ←4 ⋄ NBSG←4  AGE←L L⍴0  BSG ← 4 4 ⍴ 1 2 3 4 4 1 2 3 3 4 1 2 2 3 4 1  G0←4 4 ⍴ 0 0 0 0 3 3 3 3 0 0 3 3 0 0 0 0  NG0←3  CE←L L ⍴ 1 ⋄ CE[5;5]←5 | ITERATION NO. 1  B  BAB  B    ITERATION NO. 2  B   B  BBABB  B   B    ITERATION NO. 3  B   B   B  BBBABBB  B   B   B |
| 2. | ∇ Z←OPPOSITE K [1] Z←1+NJ|¯1+K+NJ÷2 [2] ∇  ∇ N ← I CODE K [1] N←K+NBSG×I-1 [2] ∇  ∇ Z←DECODE N;I;K [1] I←1+⌊(N-1)÷NBSG [2] K←N-NBSG×I-1 [3] Z←I,[0.5]K [4] ∇  ∇ SETEXTG0;I;K  [1] ⍝ EXTENSION  [2] NGE←NBSG×NG0+1  [3] GE←(NGE,NJ)⍴0  [4] K←I←1  [5] LOOP:GE[K+(¯1+I)×NBSG;]←G0[I;BSG[K;]]  [6] K←K+1  [7] →(K≤NBSG)/LOOP  [8] K←1  [9] I←I+1  [10] →(I≤NG0+1)/LOOP  ∇  ∇ COMPENV;K [1] ⍝ COMPUTES ENV ARRAY FROM GIVEN GLOBAL VARIABLE CE [2] ⍝ RESULT CALLED ENV IS J-ARRAY WHOSE FIRST SUBSCRIPT IS BOND COORDINATE [3] ENV←(NJ,L,L)⍴0 [4] K←1 [5] LOOP: ENV[K;;]←GE[J[K;1]⊖J[K;2]⌽CE; OPPOSITE K] [6] K←K+1 [7] →(K≤NJ)/LOOP [8] ∇  T←0 ⍝ not sure where this T is used though  ALPH←' ABCDEFGHIJKLMNOPQRSTUVWXYZ'  ∇ DEVELOP MORE; T1 [1] ⍝ computes more iterations and displays every pth [2] T1←0 [3] LOOP1:COMPENV [4] CE←⍎GROW,' CE' [5] AGE←AGE+1 [6] T←T+1 [7] T1←T1+1 [8] →(0≠P|T1)/LOOP2 [9] '' [10] 'ITERATION NO. ',⍕T1 [11] COMPNONZERO CE>NBSG [12] ALPH[(DECODE CE[V1;V2])[1;;]] [13] ⍳0 [14] LOOP2:→(T1<MORE)/LOOP1 [15] ∇  ∇ COMPNONZERO M;M1;N;P;Q [1] ⍝ REDUCES MATRIX M  [2] M1←M [3] M←M≠0 [4] N←(⍴M)[1] [5] P←( ∨/M) ⍳1 [6] Q←(⌽ ∨/M)⍳1 [7] V1←¯1+P+⍳2+N-P+Q [8] N←(⍴M)[2] [9] P←( ∨/[1]M)⍳1 [10] Q←(⌽ ∨/[1]M)⍳1 [11] V2←¯1+P+⍳2+N-P+Q [12] Z←M1[V1;V2] [13] ∇  ∇ B←GROWTH3 A [1] ⍝ GROWTH WITH MOVE OF SPLIT CELLS [2] NEW←⍎SPLIT,' CE' [3] J1←1 [4] LOOP1:→(0=∨/∨/NEW[J1;;]>NBSG)/LOOP2 [5] M←(,NEW[J1;;]>NBSG)⍳1 [6] X←⌈M÷L [7] Y←M-L×X-1 ⍝ ⋄'M X Y'⋄M⋄X⋄Y [8] J2←NEW[J1;X;Y] [9] NEW[J1;X;Y]←0 [10] DIR←J[J1;]  [11] CE←⍎'(X,Y,DIR) ', MOVE,' CE'  [12] CE[X+DIR[1];Y+DIR[2]]←J2  [13] →LOOP1 [14] LOOP2:J1←J1+1  [15] →(J1≤NJ)/LOOP1  [16] B←CE [17] ∇  ∇ B←PAR MOVE1 A [1] ⍝ MOVES RAY OF LXL MATRIX ALONG RAY [2] ⍝ STARTING AT POINT Z=PAR[1 2] IN DIRECTION [3] ⍝DIR = PAR[3 4] [4] Z←PAR[1 2] [5] DIR ← PAR[3 4] [6] SET←((⍳L)∘.×DIR)+(L⍴1)∘.×Z  [7] ANS←TEST SET  [8] SET←SET[ANS;]  [9] V←1 1 ⍉A[SET[;1];SET[;2]]  [10] NONEMPTY←V>NBSG [11] LAST←NONEMPTY⍳0  [12] B←A [13] I←1 [14] NEW1←NEW  [15] LOOP:B[SET[I+1;1];SET[I+1;2]]←A[SET[I;1];SET[I;2]] [16] NEW1[;SET[I+1;1];SET[I+1;2]]←NEW[;SET[I;1];SET[I;2]] [17] NEW1[;SET[1;1];SET[1;2]]←NJ⍴0 [18] I←I+1 [19] →(I<LAST)/LOOP  [20] NEW←NEW1  ∇  ∇ ANS←TEST Z [1] ⍝ Z IS 2 COLUMN MATRIX OF POINTS [2] ⍝ ANS IS VECTOR OF ROW SUBSCRIPTS FOR WHICH POINTS ARE INSIDE LXL LATTICE [3] ANS←(1≤Z[;1])∧(L≥Z[;1])∧(1≤Z[;2])∧L≥Z[;2] [4] ANS←ANS/⍳⍴ANS [5] ∇  ∇ B←SPLIT1 A  [1] B←(NJ⍴1)∘ . ×9×A∊ 5 6 7 8  [2] ∇ |
| 3. | SPLIT←'SPLIT1'  MOVE←'MOVE1'  P←1  SETEXTG0  GROW←'GROWTH3'  DEVELOP 3 |  |

## Example with GROWTH 3 (part 2)

Similar to the previous examples but using GROWTH3 with different G0 and functions like tau1 etc. This is in **section 4.2.3.1 p. 225** of the textbook. The python code related to this is in src/APL\_to\_python/ simple\_algo\_section4.2.3.py. In the python version, simply run

python simple\_algo\_section4.2.3.py --example\_n 2

**Growth 3 part 2 APL code.**

Run these codes successively. The first part is the initial setting, the second part main source code that is the same as the previous section, the third part executes the code, at the end of which there are a few extra observations from manual “surgery” of CE arrays.

|  |  |  |
| --- | --- | --- |
| 1. | J←4 2⍴1 0 0 1 ¯1 0 0 ¯1  L←20 ⋄ NJ←4 ⋄ NBSG←4  AGE←L L⍴0  BSG ← 4 4 ⍴ 1 2 3 4 4 1 2 3 3 4 1 2 2 3 4 1  G0← 10 4 ⍴ 0 0 0 0 2 0 0 0 3 0 3 0 4 0 4 0 5 0 5 0 6 0 6 0 7 0 7 0 8 0 8 0 9 0 9 0 0 0 10 0  NG0←9  CE←L L ⍴ 1 ⋄CE[11;5,6,7,8,9]←37 | ITERATION NO. 1  HHHHH  IIIII  ITERATION NO. 2  GGGGG  HHHHH  IIIII  ITERATION NO. 3  FFFFF  GGGGG  HHHHH  IIIII  ...  ITERATION NO. 8  AAAAA  BBBBB  CCCCC  DDDDD  EEEEE  FFFFF  GGGGG  HHHHH  IIIII  ITERATION NO. 9  AAAAA  BBBBB  CCCCC  DDDDD  EEEEE  FFFFF  GGGGG  HHHHH  IIIII  'APPLY SURGERY HERE'  EEEEE  GGGGG  HHHHH  IIIII  ITERATION NO. 1  DDDDD  EEEEE  FFFFF  FFFFF  GGGGG  HHHHH  IIIII  ITERATION NO. 2  CCCCC  DDDDD  EEEEE  FFFFF  FFFFF  GGGGG  HHHHH  IIIII  'APPLY MORE SURGERY HERE'  EEEEE  FFFFF  GGGGG  ITERATION NO. 1  DDDDD  EEEEE  FFFFF  GGGGG  FFFFF  ITERATION NO. 2  CCCCC  DDDDD  EEEEE  FFFFF  GGGGG  FFFFF  EEEEE  'LAST PART'  ITERATION NO. 1  BB B  BCCBCB  CDCCDC  DEDDED  EFEEFE  FEFFGF  GDGGFG  FCFFGF  B F |
| 2. | Copy the second part of Growth 3 part 1 APL code. |
| 3. | ∇ NEW←SPLIT2 A [1] ⍝ SPLIT TRANSFORMATION USING DEFINED TAU [2] I←(DECODE A)[1;;] [3] K←(DECODE A)[2;;]  [4] ⍎TAU [5] NEW←B [6] NEW[;2;;]←1+NBSG|NEW[;2;;]+(NBSG⍴1)∘.×K-2  [7] NEW←NEW[;1;;]CODE NEW[;2;;] [8] ∇  ∇ TAU1  ⍝ TAU-TRANSFORMATION FOR PAIR-WISE INTERATION (?)  ⍝ ASSUME BSG CYCLIC AND STANDARD POSITION K=1  ⍝ MATRIX I SHOULD BE IN G0-FORM  B←(NJ,2,L,L)⍴0  J1←1  LOOP:B[J1;1;;]←⍎'I ',NEWG,' ENV[J1;;]'  B[J1;2;;]←(L,L)⍴J1  J1←J1+1  →(J1≤NJ)/LOOP  CE←⍎CHANGEG,' CE'  ∇  ∇ Z←I NEWG2 BETA [1] Z←(L,L)⍴0 [2] COND←(I>1)∧1<|BETA-I [3] COND2←COND∧I=2 [4] COND10←COND∧I=10 [5] CONDINT←COND-COND2+COND10 [6] Z←Z+3×COND2×(J1=1)×BETA>3 [7] Z←Z+9×COND10×(J1=3)×BETA<9 [8] Z←Z+CONDINT×((J1=1)∨J1=3)×I+×BETA-I [9] ∇  MOVE←'MOVE1'  P←1  SETEXTG0  ∇ X←CHANGEI X [1] ∇  CHANGEG←' CHANGEI'  GROW←'GROWTH3'  NEWG←'NEWG2'  SPLIT←'SPLIT2'  TAU←'TAU1'  DEVELOP 9  'APPLY SURGERY HERE'  CE[3 4 5 6 7; 4+⍳5]←1  CE[8; 4+⍳5]←21 ⋄ CE  ALPH[(DECODE CE[V1;V2])[1;;]]  DEVELOP 2  'APPLY MORE SURGERY HERE'  CE[5 6 10 11 12;4+⍳5]←1  CE[9;4+⍳5]←29  ALPH[(DECODE CE[V1;V2])[1;;]]  DEVELOP 2  'LAST PART'  CE←20 20⍴1  CE[5;4+⍳6]←13 13 14 13 13 13  CE[6;4+⍳6]←19 ⋄ CE[7;4+⍳6]←21 ⋄ CE[8;4+⍳6]←27 ⋄ CE[9;4+⍳6]←29  CE[7;9]←29 ⋄ CE[9;6]←13  ALPH[(DECODE CE[V1;V2])[1;;]]  DEVELOP 1 |