TASK 4: Turn the result into a text file

Let's look of our possible inputs, examples, how they are made, and what it means (for synchronization purpose, we will use inputs that all represent 1 molecule: a single benzene ring):

1. Final string:
   ['TC5|ic$3A', 'TC5|ic$3A', 'TC5P7;U` r', 'TC5P7;U` r', 'TC5.zAW;Z', 'TC5.zAW;Z']
   They are made from task 3 which map every single atoms to its corresponding bead
   The last 6 characters are randomized to introduce uniqueness to decipher same type of beads on different parts. we will use this array as our main input to produce our text file.
2. mapping:
   [[[0, 'c', 2, [], [(1, np.float64(1.5)), (5, np.float64(1.5))], False],
   [1, 'c', 2, [], [(0, np.float64(1.5)), (2, np.float64(1.5))], False],
   [2, 'c', 2, [], [(1, np.float64(1.5)), (3, np.float64(1.5))], False],
   [3, 'c', 2, [], [(2, np.float64(1.5)), (4, np.float64(1.5))], False],
   [4, 'c', 2, [], [(3, np.float64(1.5)), (5, np.float64(1.5))], False],
   [5, 'c', 2, [], [(0, np.float64(1.5)), (4, np.float64(1.5))], False]]]
   b. Made from analyzing the smiles token, connectivity matrix, and a properties matrix that we discussed in previous tasks
   c. This is an array of arrays of arrays of 6 different values
   d. The most outer array is just the whole molecule
   e. Each inner array is a "section", where a section is classified as any ring or any connected non-ring atoms. Here since we have only benzene, we will only have 1 section which is the whole benzene ring
   f. Each inner-inner array represents an atom and everything about it. The next part will explain what is in this array:
      i. index 0: This is an integer that represents the appearance of this atom when tracing the smiles tokens, where 0 means it is the first atom encountered when tracing the token
      ii. index 1: represent the atom type, we get this from tracing the token
      iii. index 2: represent the section type, 0 means non-ring, 1 means non-benzene ring, and 2 means a benzene ring
      iv. index 3: an array representing the connection from this atom to another atom that is not found in this section. The form of this array is [section index, atom index (this index is not the same as index 0 of the atom, this is the index location of the section array where we can find this atom), bond amount]

v. index 4: an array representing the connection from this atom to another atom that is found in this section. The form of this array is [(atom index (this index is not the same as index 0 of the atom, this is the index location of the section array where we can find this atom), bond amount), (more tuples if needed]
vi. index 5: is this an edge atom, an edge atom is only connected to exactly 0 or 1 other atom within its group (foreign connection does not matter)

g. in Task 3, we have altered the mapping but we will use the original version of this mapping

3. compound_name for saving purposes

Our goal: to create a table called “All beads” into a text file that have column that are labeled: index, beadtype, composed of index, composed of element, # of connections

and then another table below that called “Beads connections” with column that are labeled: index, connection_index, bond type,

First we have to fix the first and second column of the first table, we can do this pretty easily:

REFER TO TASK 4.a here:

After task 4.a we should have 2 arrays that are index and beadtypes, we dont have to put it into the text file yet, we can just finish off the rest

REFER TO TASK 4.b here:

After task 4.b we should have 2 arrays that are coi and coe, now we need to create the elements for the other table

REFER TO TASK 4.c here:

After this, we only have # of connections left and this can be done pretty easily:

# of connections array []

for i in index of the first table

# of connections = 0

trace the index array of the second table, for every i is found increment by 1

append that to # of connections array

now that we have all the arrrays, we can write it into the text file

note that by now, every array in their respective table should have the same amount of element

we now just write the arrays into a text files accoringly, for the sections that are composed of index, composed of elements, we do not need the [] but rather just do comma to separate the elements in the array

we also need to do 1 more thing, fixing the beadtype arrays:

for every element in beadtype_arrays: delete the last 6 characters, then delete every “+” characters, then return back the beadtype_arrays

make sure to indent correctly so everything is pretty.

write the text into a file called {compound_name}.txt with both of the tables

Task 4.a: find the index array and the beadtype array:

input: the final array

create an empty set(), index array [] and beadtype array [], i = 0

trace through the final array:

if a string is not encountered before, put that string into the encountered set

put i into index array

append the final string into beadtype_array

i++

if a string is already encountered, continue

return index, beadtype

TASK 4.b: find the composed of index, composed of element arrays

input, final, mapping, beadtype

create composed of index [], composed of element arrays []

for i in range length(beadtype)

empty array

loop through every j in range length final

check whether final(j) is found in beadtype[i], if it is not continue if it is append j to the empty array

once we reach the end we append this array to composed of index array

for i in composed of index:

empty array

for j in i, check every atom in every section in mapping until we find an atom with index 0 = j

then we find the index 1 of that atom and append it to the empty array

once we reach the end we append this array to composed of element array[]

return coi, coe arrays

Task 4.c: find the index, connection_index, bond type arrays for the other table

input: use any input above and computed from 4.a and 4.b as needed

for i in composed of index

for j in i: check every atom in every section in mapping until we find an atom with index 0 = j

then we check index 4 of that atom and go to every neighboring nodes and check whether we can find the index 0 of the neighboring node in i,

if we cannot find it, that is a new connection.

We will now find the index “c_index” of connection_index where there is an array with an element that is the index 0 of the neighboring node.

We will also check for the connection “connection” between the neighboring node and the current node by checking the second element in every tuple of index 4

append the index of i in index to index, c_index into connection index and “connection” to the bond type array

then we check index 3 of the atom to see if there is a foreign node, if there is we will do the exact same thing where we go to the section and the atom in that section that is the foreign node and check if index 0 of the foreign node in i

if we cannot find it, that is a new connection.

We will now find the index “c_index” of connection_index where there is an array with an element that is the index 0 of the neighboring node.

We will also check for the connection “connection” between the neighboring node and the current node by checking the third element in index 3

append the index of i in index to index, c_index into connection index and “connection” to the bond type array

return index, c_index, bond_type arrays