

```
1 # cs224FinalProject
2 A shared repository for a group project in CS224
3
4 Authors: Seth Campbell, Noah Shirey, Drew Ellery
5
6 The purpose of the project is to create a 2D, circular, doubly-linked list that can be used as a library
   for exact cover problems.
7
8
9
10 ---IMPORTANT INFORMATION---
11
12 1) Although this library is intended for exact cover problems, it was written to be robust enough to
    handle all sorts of matrix implementations.
13 This should be evident in our test cases.
14
15 2) One of our test cases demonstrated our implementation on a large scale. As is, the test takes about 30
    seconds to complete.
16 In order to shorten this time you may do one of the following:
17     a) Remove, or comment out lines 567-580
18     b) Replace the values on lines 571, and 572 with smaller values
19
20 3) All tests can be run either through terminal, by executing the file, or through an IDE with built in
    test support, such as PyCharm.
```

```
1 import unittest
2 import random
3
4
5 class Node(object):
6     """ This node class is intended for a 2-D, circular, doubly liked list"""
7
8     def __init__(self, up, down, left, right, x, y, value=None):
9         """ Creates a node that stores values and pointers to nodes adjacent to the node.
10
11         :param up:the pointer to the node above the current node
12         :param down:the pointer to the node below the current node
13         :param right:the pointer to the node to the right of the current node
14         :param left:the pointer to the node to the left of the current node
15         :param x:the indexed row number starting from the top of the matrix down
16         :param y:the indexed column number starting from the left of the matrix down
17         :param value:a numerical value stored by the node
18
19         :type up: node
20         :type down: node
21         :type left: node
22         :type right: node
23         :type x: int
24         :type y: int
25         :type value: int
26     """
27
28     self.up = up
29     self.down = down
30     self.left = left
31     self.right = right
32
33     if up is None:
34         self.up = self
35     if down is None:
```

File - /Applications/MAMP/htdocs/cs224FinalProject/seth_noah_drew_matrix_project.py

```
36         self.down = self
37     if left is None:
38         self.left = self
39     if right is None:
40         self.right = self
41
42     self.x = x
43     self.y = y
44     self.value = value
45
46
47 class Header(Node):
48     """The header is a subclass of node.
49     It also has functionality to keep track of the total value of all node in the given column
50     """
51
52     def __init__(self, up, down, left, right, primary):
53         """ Creates a subclass of node that stores values and pointers to nodes adjacent to that node and
54         keeps track of the sum of the values of the nodes of the column.
55
56         :param up:the pointer to the node above the current node
57         :param down:the pointer to the node below the current node
58         :param right:the pointer to the node to the right of the current node
59         :param left:the pointer to the node to the left of the current node
60         :param primary:a boolean marking the column as primary or not primary.
61
62         :type up:node
63         :type down:node
64         :type left:node (header)
65         :type right:node (header)
66         :type primary:boolean
67
68     """
69
70     super(Header, self).__init__(up, down, left, right, None, None)
```

```
70         self.is_primary = primary
71
72     def tally_values(self):
73         """Sets the instance variable value to the sum of the values in the column the header is in."""
74
75         current_node = self.down
76         self.value = 0
77
78         while type(current_node) is not Header:
79             self.value += current_node.value
80             current_node = current_node.down
81
82
83
84     class Matrix:
85         """The matrix class contains a reference to the first node in the entire structure.
86         In its entirety, the matrix structure is a circular, doubly-linked, 2D list.
87         """
88
89         def __init__(self, values):
90             """Creates a matrix with headers for each column and nodes for each entry in the matrix.
91
92             :param values: a two dimensional array where each element of the array is the value of the node
93             at that position in the matrix
94             :type values: collections iterable
95
96             """
97
98             self.removed_nodes = []
99             self.first_header = None
100            self.total_rows = 0
101            self.zero_columns = False
102
103            for row in values:
104                self.add_row(row)
```

```
104
105     def add_row(self, values):
106         """ adds a row to the matrix and if no rows have been created it creates headers for the matrix
107
108         :param values: an array containing the values for the nodes that are being created
109         :type values: collections iterable
110
111         """
112
113         if self.first_header is None:
114             self.first_header = Header(None, None, None, None, False)
115             current_header = self.first_header
116             for i in range(1, len(values)):
117                 new_header = Header(None, None, current_header, self.first_header, False)
118                 self.first_header.left = new_header
119                 current_header.right = new_header
120                 current_header = new_header
121
122             headers = self.get_number_of_headers()
123             if len(values) != headers:
124                 raise ValueError("Invalid number of values for given number of rows. Matrix has "+str(
headers)+" headers, but "+str(len(values))+" values were given.")
125
126             current_node = Node(self.first_header.up, self.first_header, None, None, self.total_rows, 0,
values[0])
127             self.first_header.up = current_node
128             current_node.up.down = current_node
129
130             for i in range(1, len(values)):
131                 new_node = Node(current_node.up.right, current_node.down.right, current_node, self.
first_header.up, self.total_rows, i, values[i])
132
133                 new_node.up.down = new_node
134                 new_node.down.up = new_node
```

File - /Applications/MAMP/htdocs/cs224FinalProject/seth_noah_drew_matrix_project.py

```
135         new_node.left.right = new_node
136         new_node.right.left = new_node
137
138         current_node = new_node
139
140         self.tally_all_values()
141         self.total_rows += 1
142         array_to_add = []
143         for i in range(0, headers):
144             array_to_add.append(None)
145         self.removed_nodes.append(array_to_add)
146
147     def get_number_of_headers(self):
148         """ returns total number of headers.
149
150         :return: the number of headers in the matrix.
151         :rtype: int
152         """
153
154         header = self.first_header.right
155         total = 1
156         while header is not self.first_header:
157             total += 1
158             header = header.right
159         return total
160
161     def tally_all_values(self):
162         """loops through every header node and then sets the tally_values function on them"""
163
164         current_header = self.first_header
165
166         first = True
167         while first or (current_header is not self.first_header):
168             first = False
169             current_header.tally_values()
```

```
170         current_header = current_header.right
171
172     def remove_row(self, index):
173         """removes a row at a specified index and returns the first node of the row that was removed
174
175         :param index: the position in the matrix at which the row resides
176         :type index: int
177
178         :return: returns the first node in the row.
179         :rtype: node
180         """
181
182         self.remove_row_overlap(index)
183
184         current_node = self.first_header
185         for i in range(0, index+1):
186             current_node = current_node.down
187
188         first_node = current_node
189
190         first = True
191         while first or (current_node is not first_node):
192             first = False
193
194             current_node.up.down = current_node.down
195             current_node.down.up = current_node.up
196             current_node = current_node.right
197             self.removed_nodes[index][current_node.y] = current_node
198
199         self.tally_all_values()
200         return first_node
201
202     def remove_row_overlap(self, index):
203         """if you are removing a row that overlaps a column it will remove that node
204         that had already been removed along with the other nodes in the row.
```

```
205
206     :param index:the position in the matrix at which the row resides
207     :type index:int
208     """
209
210     for i in range(0, len(self.removed_nodes[index])):
211         current_node = self.removed_nodes[index][i]
212         if current_node is not None:
213             current_node.up.down = current_node.down
214             current_node.down.up = current_node.up
215
216     def restore_row(self, first_node):
217         """restores a row that has been removed by restoring all pointers for each node in the row
218
219         :param first_node: the first node in the row that was deleted.
220         :type first_node: node
221         """
222
223         current_node = first_node
224
225         first = True
226         while first or (current_node is not first_node):
227             first = False
228
229             current_iterative_node = current_node.up
230             while current_iterative_node.up.down is not current_iterative_node:
231                 current_iterative_node.down = current_node
232                 current_iterative_node = current_iterative_node.up
233             current_iterative_node.down = current_node
234             current_node.up = current_iterative_node
235
236             current_iterative_node = current_node.down
237
238             while current_iterative_node.down.up is not current_iterative_node:
239                 current_iterative_node.up = current_node
```

```
240         current_iterative_node = current_iterative_node.down
241         current_iterative_node.up = current_node
242         current_node.down = current_iterative_node
243
244         current_node = current_node.right
245         self.removed_nodes[current_node.x][current_node.y] = None
246
247         self.restore_row_overlap(first_node.x)
248         self.tally_all_values()
249
250     def restore_row_overlap(self, index):
251         """restores a row and if a node has been removed as a result of a column removal it will restore
252         that node as well
253
254         :param index: the position in the matrix at which the row resides
255         :type index: int
256         """
257
258         for i in range(0, len(self.removed_nodes[index])):
259             current_node = self.removed_nodes[index][i]
260             if current_node is not None:
261                 current_node.up.down = current_node
262                 current_node.down.up = current_node
263
264     def remove_column(self, index):
265         """removes a column at a specified index.
266
267         :param index: the position in the matrix at which the row resides
268         :type index: int
269         """
270
271         self.remove_column_overlap(index)
272
273         current_node = self.first_header
274         for i in range(0, index):
275             current_node = current_node.right
```

```
274     first_node = current_node.down
275
276     if current_node is self.first_header:
277         if current_node.right is current_node:
278             self.zero_columns = True
279             return first_node
280         self.first_header = current_node.right
281
282     first = True
283     while first or (type(current_node) is not Header):
284         current_node.left.right = current_node.right
285         current_node.right.left = current_node.left
286
287         if not first:
288             self.removed_nodes[current_node.x][index] = current_node
289             current_node = current_node.down
290             first = False
291
292     current_node.tally_values()
293     return first_node
294
295
296     def remove_column_overlap(self, index):
297         """removes a column and if a node has been removed as a result of a row removal it will remove
298 that node as well as the
299 other nodes in that row.
300
301     :param index:the position in the matrix at which the row resides
302     :type index:int
303     """
304
305     for i in range(0, len(self.removed_nodes)):
306         current_node = self.removed_nodes[i][index]
307         if current_node is not None:
308             current_node.left.right = current_node.right
```

```
308         current_node.right.left = current_node.left
309
310     def restore_column(self, first_node):
311         """restores a column
312
313         :param first_node:the node in the first position of the column that was deleted.
314         :type first_node:node
315         """
316
317     current_node = first_node
318
319     if current_node.y < self.first_header.down.y:
320         self.first_header = current_node.up
321
322     first = True
323     while first or (current_node is not first_node):
324         first = False
325
326         current_iterative_node = current_node.left
327         while current_iterative_node.left.right is not current_iterative_node:
328             current_iterative_node.right = current_node
329             current_iterative_node = current_iterative_node.left
330         current_iterative_node.right = current_node
331         current_node.left = current_iterative_node
332
333         current_iterative_node = current_node.right
334
335         while current_iterative_node.right.left is not current_iterative_node:
336             current_iterative_node.left = current_node
337             current_iterative_node = current_iterative_node.right
338         current_iterative_node.left = current_node
339         current_node.right = current_iterative_node
340
341     if type(current_node) is not Header:
342         self.removed_nodes[current_node.x][current_node.y] = None
```

```
File - /Applications/MAMP/htdocs/cs224FinalProject/seth_noah_drew_matrix_project.py
343         current_node = current_node.down
344
345     self.restore_column_overlap(first_node.y)
346     self.tally_all_values()
347
348     def restore_column_overlap(self, index):
349         """restores a column and if a node has been added as a result of a row addition it will exclude
350         that node rather than
351         add it twice
352
353         :param index:the position in the matrix at which the row resides
354         :type index:int
355         """
356
357         for i in range(0, len(self.removed_nodes)):
358             current_node = self.removed_nodes[i][index]
359             if current_node is not None:
360                 current_node.left.right = current_node
361                 current_node.right.left = current_node
362
363     def get_array_representation(self):
364         """returns the array representation of the matrix by traversing the rows left to right and
365         appending each
366         nodes value to an array."""
367
368         if self.zero_columns:
369             return []
370         array = []
371         current_node = self.first_header.down
372         while type(current_node) is not Header:
373             inner_array = []
374             first_node_in_row = current_node
375             first = True
376             while first or (current_node is not first_node_in_row):
377                 first = False
378                 inner_array.append(current_node.value)
379                 current_node = current_node.down
380             array.append(inner_array)
381             current_node = current_node.right
```

```
376         inner_array.append(current_node.value)
377         current_node = current_node.right
378         array.append(inner_array)
379         current_node = current_node.down
380
381     return array
382
383
384 class UnitTest(unittest.TestCase):
385
386     def test_node_creation(self):
387         node = Node(None, None, None, None, 1, 2, 5)
388         self.assertEqual(node.value, 5)
389         self.assertEqual(node.x, 1)
390         self.assertEqual(node.y, 2)
391
392     def test_circular_node_creation(self):
393         node = Node(None, None, None, None, None, None)
394         self.assertEqual(node.up, node)
395         self.assertEqual(node.down, node)
396         self.assertEqual(node.left, node)
397         self.assertEqual(node.right, node)
398
399     def test_header_creation(self):
400         node = Header(None, None, None, None, False)
401         self.assertEqual(node.value, None)
402         self.assertEqual(node.x, None)
403         self.assertEqual(node.y, None)
404         self.assertEqual(node.is_primary, False)
405
406     def test_circular_header_creation(self):
407         node = Header(None, None, None, None, None)
408         self.assertEqual(node.up, node)
409         self.assertEqual(node.down, node)
410         self.assertEqual(node.left, node)
```

```
411         self.assertEqual(node.right, node)
412
413     def test_header_tally_values(self):
414         header = Header(None, None, None, None, False)
415         node1 = Node(header, None, None, None, 1, 1, 1)
416         node2 = Node(node1, None, None, None, 2, 1, 2)
417         node3 = Node(node2, header, None, None, 3, 1, 3)
418
419         header.up = node3
420         header.down = node1
421         node1.down = node2
422         node2.down = node3
423
424         header.tally_values()
425
426         self.assertEqual(header.value, 6)
427
428     def test_header_tally_values_when_negative(self):
429         header = Header(None, None, None, None, False)
430         node1 = Node(header, None, None, None, 1, 1, 1)
431         node2 = Node(node1, None, None, None, 2, 1, 2)
432         node3 = Node(node2, header, None, None, 3, 1, -3)
433
434         header.up = node3
435         header.down = node1
436         node1.down = node2
437         node2.down = node3
438
439         header.tally_values()
440
441         self.assertEqual(header.value, 0)
442
443     def test_header_tally_values_after_change(self):
444         header = Header(None, None, None, None, False)
445         node1 = Node(header, None, None, None, 1, 1, 1)
```

```
446     node2 = Node(node1, None, None, None, 2, 1, 2)
447     node3 = Node(node2, header, None, None, 3, 1, 3)
448
449     header.up = node3
450     header.down = node1
451     node1.down = node2
452     node2.down = node3
453
454     header.tally_values()
455
456     self.assertEqual(header.value, 6)
457
458     node3.value += 10
459     header.tally_values()
460
461     self.assertEqual(header.value, 16)
462
463 def test_matrix_creation_no_values(self):
464     matrix = Matrix([])
465     self.assertEqual(matrix.removed_nodes, [])
466     self.assertEqual(matrix.first_header, None)
467     self.assertEqual(matrix.total_rows, 0)
468
469 def test_matrix_add_row(self):
470     matrix = Matrix([])
471     matrix.add_row([1, 2, 3])
472
473     header1 = Header(None, None, None, None, False)
474     header2 = Header(None, None, header1, None, False)
475     header3 = Header(None, None, header2, header1, False)
476
477     header1.left = header3
478     header1.right = header2
479     header2.right = header3
480
```

```
481     node1 = Node(header1, header1, None, None, 1, 1, 1)
482     node2 = Node(header2, header2, node1, None, 1, 2, 2)
483     node3 = Node(header3, header3, node2, node1, 1, 3, 3)
484
485     header1.down = node1
486     header2.down = node2
487     header3.down = node3
488
489     header1.up = node1
490     header2.up = node2
491     header3.up = node3
492
493     node1.left = node3
494     node1.right = node2
495     node2.right = node3
496
497     self.assertEqual(node1.value, matrix.first_header.down.value)
498     self.assertEqual(node2.value, matrix.first_header.down.right.value)
499     self.assertEqual(node3.value, matrix.first_header.down.left.value)
500
501     self.assertEqual(node1.value, matrix.first_header.down.value)
502     self.assertEqual(node2.value, matrix.first_header.down.right.value)
503     self.assertEqual(node3.value, matrix.first_header.down.left.value)
504
505 def test_matrix_add_multiple_rows(self):
506     matrix = Matrix([])
507     matrix.add_row([1, 2, 3])
508     matrix.add_row([4, 5, 6])
509     matrix.add_row([7, 8, 9])
510
511     header1 = Header(None, None, None, None, False)
512     header2 = Header(None, None, header1, None, False)
513     header3 = Header(None, None, header2, header1, False)
514
515     header1.left = header3
```

```
516     header1.right = header2
517     header2.right = header3
518
519     node1 = Node(header1, None, None, None, 1, 1, 1)
520     node2 = Node(header2, None, node1, None, 1, 2, 2)
521     node3 = Node(header3, None, node2, node1, 1, 3, 3)
522
523     node1.left = node3
524     node1.right = node2
525     node2.right = node3
526
527     node4 = Node(node1, None, None, None, 2, 1, 4)
528     node5 = Node(node2, None, node4, None, 2, 2, 5)
529     node6 = Node(node3, None, node5, node4, 2, 3, 6)
530
531     node4.left = node6
532     node4.right = node5
533     node2.right = node6
534
535     node7 = Node(node4, header1, None, None, 3, 1, 7)
536     node8 = Node(node5, header2, node7, None, 3, 2, 8)
537     node9 = Node(node6, header3, node8, node7, 3, 3, 9)
538
539     header1.down = node1
540     header2.down = node2
541     header3.down = node3
542
543     header1.up = node7
544     header2.up = node8
545     header3.up = node9
546
547     node1.down = node4
548     node2.down = node5
549     node3.down = node6
550
```

```
551     node4.down = node7
552     node5.down = node8
553     node6.down = node9
554
555     self.assertEqual(node1.value, matrix.first_header.down.value)
556     self.assertEqual(node2.value, matrix.first_header.down.right.value)
557     self.assertEqual(node3.value, matrix.first_header.down.left.value)
558
559     self.assertEqual(node4.value, matrix.first_header.down.down.value)
560     self.assertEqual(node5.value, matrix.first_header.down.down.right.value)
561     self.assertEqual(node6.value, matrix.first_header.down.down.left.value)
562
563     self.assertEqual(node7.value, matrix.first_header.down.down.down.value)
564     self.assertEqual(node8.value, matrix.first_header.down.down.down.right.value)
565     self.assertEqual(node9.value, matrix.first_header.down.down.down.left.value)
566
567 def test_add_large_number_rows(self):
568     print("----THIS TEST TAKES ABOUT 30 SECONDS TO COMPLETE----")
569     print("----TO SKIP THIS TEST, COMMENT OUT LINES 567-580----")
570     matrix = Matrix([])
571     numberOfRows = 500
572     numberOfColumns = 500
573
574     for x in range(0,numberOfRows):
575         row = []
576         for x in range(0, numberOfColumns):
577             row.append(random.randint(0,100))
578         matrix.add_row(row)
579     print("----LONG TEST COMPLETE----")
580     self.assertEqual(matrix.total_rows, numberOfRows)
581
582 def test_get_array_representation(self):
583     matrix = Matrix([])
584     matrix.add_row([1, 2, 3])
585     matrix.add_row([4, 5, 6])
```

File - /Applications/MAMP/htdocs/cs224FinalProject/seth_noah_drew_matrix_project.py

```
586     matrix.add_row([7, 8, 9])
587
588     self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
589
590     def test_matrix_creation_with_values(self):
591         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
592
593         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
594
595     def test_matrix_tally_all_ones(self):
596         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
597
598         matrix.tally_all_values()
599
600         self.assertEqual(12, matrix.first_header.value)
601         self.assertEqual(15, matrix.first_header.right.value)
602         self.assertEqual(18, matrix.first_header.left.value)
603
604     def test_matrix_add_row_over_bounds(self):
605         matrix = Matrix([[1, 2, 3]])
606         self.assertRaises(ValueError, matrix.add_row, [4, 5, 6, 7])
607
608     def test_matrix_add_row_under_bounds(self):
609         matrix = Matrix([[1, 2, 3]])
610         self.assertRaises(ValueError, matrix.add_row, [4, 5])
611
612     def test_matrix_get_number_of_headers(self):
613         matrix = Matrix([[1, 2, 3]])
614         self.assertEqual(3, matrix.get_number_of_headers())
615
616     def test_matrix_remove_row_no_overlap(self):
617         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
618
619         matrix.remove_row(1)
```

```
621     self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [7, 8, 9]])
622
623 def test_matrix_remove_multiple_rows_non_successive(self):
624     matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]])
625
626     matrix.remove_row(1)
627     matrix.remove_row(2)
628
629     self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [7, 8, 9], [13, 14, 15]])
630
631 def test_matrix_remove_multiple_rows_successive(self):
632     matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], [13, 14, 15]])
633
634     matrix.remove_row(1)
635     matrix.remove_row(1)
636
637     self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [10, 11, 12], [13, 14, 15]])
638
639 def test_matrix_remove_all_rows(self):
640     matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
641
642     matrix.remove_row(0)
643     matrix.remove_row(0)
644     matrix.remove_row(0)
645
646     self.assertEqual(matrix.get_array_representation(), [])
647
648 def test_matrix_remove_column_no_overlap(self):
649     matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
650
651     matrix.remove_column(0)
652
653     self.assertEqual(matrix.get_array_representation(), [[2, 3], [5, 6], [8, 9]])
654
655 def test_matrix_remove_multiple_columns_non_successive(self):
```

```
656     matrix = Matrix([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])  
657  
658     matrix.remove_column(1)  
659     matrix.remove_column(2)  
660  
661     self.assertEqual(matrix.get_array_representation(), [[1, 3, 5], [6, 8, 10], [11, 13, 15]])  
662  
663 def test_matrix_remove_multiple_columns_successive(self):  
664     matrix = Matrix([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])  
665  
666     matrix.remove_column(1)  
667     matrix.remove_column(1)  
668  
669     self.assertEqual(matrix.get_array_representation(), [[1, 4, 5], [6, 9, 10], [11, 14, 15]])  
670  
671 def test_matrix_remove_column_with_first_header(self):  
672     matrix = Matrix([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])  
673  
674     new_header = matrix.first_header.right  
675  
676     matrix.remove_column(0)  
677  
678     self.assertEqual(matrix.first_header, new_header)  
679     self.assertEqual(matrix.get_array_representation(), [[2, 3, 4, 5], [7, 8, 9, 10], [12, 13, 14,  
15]])  
680  
681 def test_matrix_remove_column_with_first_header_where_next_column_is_non_adjacent(self):  
682     matrix = Matrix([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])  
683  
684     new_header = matrix.first_header.right.right  
685  
686     matrix.remove_column(1)  
687     matrix.remove_column(0)  
688  
689     self.assertEqual(matrix.first_header, new_header)
```

```
690     self.assertEqual(matrix.get_array_representation(), [[3, 4, 5], [8, 9, 10], [13, 14, 15]])  
691  
692     def test_matrix_remove_all_columns(self):  
693         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
694  
695         matrix.remove_column(0)  
696         matrix.remove_column(0)  
697         matrix.remove_column(0)  
698  
699         self.assertEqual(matrix.get_array_representation(), [])  
700  
701     def test_matrix_remove_row_remove_column(self):  
702         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
703  
704         matrix.remove_row(1)  
705         matrix.remove_column(1)  
706  
707         self.assertEqual(matrix.get_array_representation(), [[1, 3], [7, 9]])  
708  
709     def test_matrix_remove_column_remove_row(self):  
710         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
711  
712         matrix.remove_column(1)  
713         matrix.remove_row(1)  
714  
715         self.assertEqual(matrix.get_array_representation(), [[1, 3], [7, 9]])  
716  
717     def test_matrix_restore_non_overlapping_row(self):  
718         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
719  
720         node_removed = matrix.remove_row(1)  
721         matrix.restore_row(node_removed)  
722  
723         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
724
```

```
725     def test_matrix_restore_multiple_rows_same_order(self):
726         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
727
728         node_removed = matrix.remove_row(1)
729         node_removed2 = matrix.remove_row(1)
730         matrix.restore_row(node_removed)
731         matrix.restore_row(node_removed2)
732
733         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
734
735     def test_matrix_restore_multiple_rows_different_order(self):
736         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
737
738         node_removed = matrix.remove_row(1)
739         node_removed2 = matrix.remove_row(1)
740         matrix.restore_row(node_removed2)
741         matrix.restore_row(node_removed)
742
743         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
744
745     def test_matrix_restore_non_overlapping_column(self):
746         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
747
748         node_removed = matrix.remove_column(1)
749         matrix.restore_column(node_removed)
750
751         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
752
753     def test_matrix_restore_multiple_columns_same_order(self):
754         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
755
756         node_removed = matrix.remove_column(1)
757         node_removed2 = matrix.remove_column(1)
758         matrix.restore_column(node_removed)
759         matrix.restore_column(node_removed2)
```

```
760
761     self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
762
763     def test_matrix_restore_multiple_columns_different_order(self):
764         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
765
766         node_removed = matrix.remove_column(1)
767         node_removed2 = matrix.remove_column(1)
768         matrix.restore_column(node_removed2)
769         matrix.restore_column(node_removed)
770
771         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
772
773     def test_matrix_restore_multiple_columns_difficult_order(self):
774         matrix = Matrix([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])
775
776         node_removed = matrix.remove_column(1)
777         node_removed2 = matrix.remove_column(1)
778         node_removed3 = matrix.remove_column(1)
779         node_removed4 = matrix.remove_column(1)
780         matrix.restore_column(node_removed)
781         matrix.restore_column(node_removed3)
782         matrix.restore_column(node_removed4)
783         matrix.restore_column(node_removed2)
784
785         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12,
786             13, 14, 15]])
787
788     def test_matrix_remove_column_with_first_header_backtracking_header(self):
789
790         matrix = Matrix([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]])
791
792         new_header = matrix.first_header
793
794         removed_node = matrix.remove_column(0)
```

```
File - /Applications/MAMP/htdocs/cs224FinalProject/seth_noah_drew_matrix_project.py
794     matrix.remove_column(0)
795     matrix.restore_column(removed_node)
796
797     self.assertEqual(matrix.first_header, new_header)
798     self.assertEqual(matrix.get_array_representation(), [[1, 3, 4, 5], [6, 8, 9, 10], [11, 13, 14,
15]])
799
800     def test_matrix_remove_and_restore_rows_and_columns_backtracking_order(self):
801         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
802
803         node_removed = matrix.remove_row(1)
804         node_removed2 = matrix.remove_column(1)
805         matrix.restore_column(node_removed2)
806         matrix.restore_row(node_removed)
807
808         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
809
810     def test_matrix_remove_and_restore_rows_and_columns_respective_order(self):
811         matrix = Matrix([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
812
813         node_removed = matrix.remove_row(1)
814         node_removed2 = matrix.remove_column(1)
815         matrix.restore_row(node_removed)
816         matrix.restore_column(node_removed2)
817
818         self.assertEqual(matrix.get_array_representation(), [[1, 2, 3], [4, 5, 6], [7, 8, 9]])
819
820
821     def main():
822         unittest.main()
823
824
825     if __name__ == '__main__':
826         main()
827
```

File - /Applications/MAMP/htdocs/cs224FinalProject/seth_noah_drew_matrix_project.py

828

829