PP01: Divide and Conquer

Name: Manohara Rao Penumala

ID: 1209455582

Python Code:

1. *# -\*- coding: utf-8 -\*-*
2. **import** numpy
3. *# STOCK\_PRICES  = [100,113,110,85,105,102,86,63,81,101,94,106,101,79,94,90,97]*
4. STOCK\_PRICE\_CHANGES = [13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]

7. *# Implement pseudo code from the book*
8. **def** find\_maximum\_sub\_array\_brute(A, low=0, high=-1):
9. """
10. Return a tuple (i,j) where A[i:j] is the maximum subarray.
11. Implement the brute force method from chapter 4
12. time complexity = O(n^2)
14. >>> STOCK\_PRICE\_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
15. >>> find\_maximum\_sub\_array\_brute(STOCK\_PRICE\_CHANGES, 0, 15)
16. (7, 10)
17. """
18. left\_position = low
19. right\_position = low
20. maximum = 0
21. **for** i **in** range(0, high):
22. current\_max = 0
23. **for** j **in** range(i, high):
24. current\_max = current\_max + A[j]
25. **if** maximum < current\_max:
26. maximum = current\_max
27. left\_position = i
28. right\_position = j
29. **return** (left\_position, right\_position)

32. *# Implement pseudocode from the book*
33. **def** find\_maximum\_crossing\_sub\_array(A, low, mid, high):
34. """
35. Find the maximum subarray that crosses mid
36. Return a tuple (i,j) where A[i:j] is the maximum subarray.
38. >>> STOCK\_PRICE\_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
39. >>> find\_maximum\_crossing\_sub\_array(STOCK\_PRICE\_CHANGES, 0, 7, 15)
40. (7, 10, 43)
41. """
42. left\_max = 0
43. maximum = 0
44. left\_position = mid
45. **for** i **in** range(mid-1, low-1, -1):
46. maximum = maximum + A[i]
47. **if** left\_max < maximum:
48. left\_max = maximum
49. left\_position = i
50. right\_max = 0
51. maximum = 0
52. right\_position = mid
53. **for** j **in** range(mid, high):
54. maximum = maximum + A[j]
55. **if** right\_max < maximum:
56. right\_max = maximum
57. right\_position = j
58. **return** (left\_position, right\_position, left\_max+right\_max)

61. **def** find\_maximum\_sub\_array\_recursive(A, low=0, high=-1):
62. """
63. Return a tuple (i,j) where A[i:j] is the maximum subarray.
64. Recursive method from chapter 4
66. >>> STOCK\_PRICE\_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
67. >>> find\_maximum\_sub\_array\_recursive(STOCK\_PRICE\_CHANGES, 0, 15)
68. (7, 10, 43)
69. """
70. **if** high == low:
71. **return** (low, high, A[0])
72. **else**:
73. mid = ((low + high)/2)
74. left\_tuple = find\_maximum\_sub\_array\_recursive(A, low, mid)
75. right\_tuple = find\_maximum\_sub\_array\_recursive(A, mid+1, high)
76. cross\_tuple = find\_maximum\_crossing\_sub\_array(A, low, mid, high)
77. **if** left\_tuple[2] >= right\_tuple[2] **and** left\_tuple[2] >= cross\_tuple[2]:
78. **return** (left\_tuple[0], left\_tuple[1], left\_tuple[2])
79. **elif** right\_tuple[2] >= left\_tuple[2] **and** right\_tuple[2] >= cross\_tuple[2]:
80. **return** (right\_tuple[0], right\_tuple[1], right\_tuple[2])
81. **else**:
82. **return** (cross\_tuple[0], cross\_tuple[1], cross\_tuple[2])

85. **def** find\_maximum\_sub\_array\_iterative(A, low=0, high=-1):
86. """
87. Return a tuple (i,j) where A[i:j] is the maximum subarray.
88. Do problem 4.1-5 from the book.
89. Assuming that at least one of the input values will be positive.
91. >>> STOCK\_PRICE\_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]
92. >>> find\_maximum\_sub\_array\_iterative(STOCK\_PRICE\_CHANGES, 0, 15)
93. (7, 10)
94. """
95. left\_position = 0
96. right\_position = 0
97. current\_position = 0
98. positive\_element\_exists = 0
99. maximum = A[low]
100. S = [0]\*len(A)
101. **if** A[low] > 0:
102. S[0] = A[low]
103. positive\_element\_exists = 1
104. **for** i **in** range(low+1, high):
105. **if** A[i] > 0:
106. positive\_element\_exists = 1
107. S[i] = S[i-1] + A[i]
108. **if** S[i] > S[i-1] **and** S[i-1] <= 0:
109. current\_position = i
110. **if** S[i] < 0:
111. S[i] = 0
112. **if** S[i] > maximum:
113. maximum = S[i]
114. left\_position = current\_position
115. right\_position = i
116. **if** positive\_element\_exists == 1:
117. **return** (left\_position, right\_position)
118. **else**:
119. **return** (0, 0)

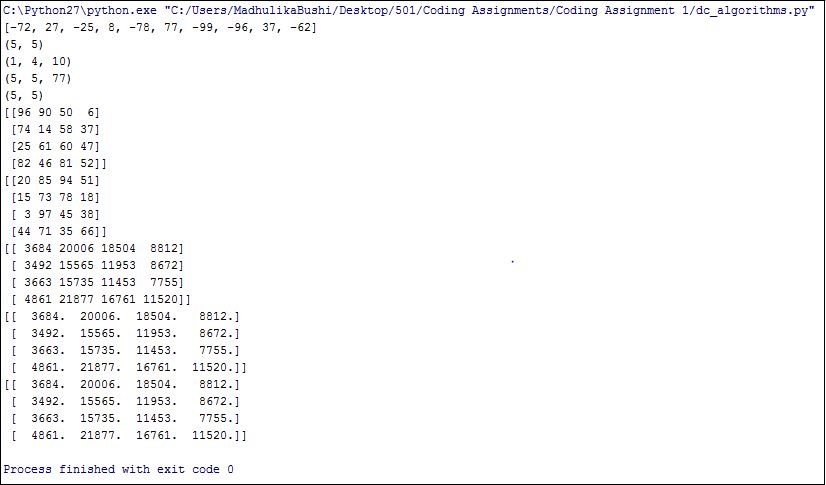
122. **def** square\_matrix\_multiply(A, B):
123. """
124. Return the product AB of matrix multiplication.
126. >>> A = [[36, 54, 24, 38], [54, 50, 19, 68], [26, 79, 57, 49], [94, 59, 20, 97]]
127. >>> B = [[46, 68, 27, 38], [57, 94, 74, 20], [46, 0, 52, 69], [20, 65, 37, 26]]
128. >>> square\_matrix\_multiply(A, B)
129. array([[  6598.,   9994.,   7622.,   5092.],
130. [  7568.,  12792.,   8662.,   6131.],
131. [  9301.,  12379.,  11325.,   7775.],
132. [ 10547.,  18243.,  11533.,   8654.]])
133. """
134. A = numpy.asarray(A)
135. B = numpy.asarray(B)
136. **assert** A.shape == B.shape
137. **assert** A.shape == A.T.shape
138. dimensions = A.shape
139. C = numpy.zeros(dimensions)
140. **for** i **in** range(0, dimensions[0]):
141. **for** j **in** range(0, dimensions[0]):
142. **for** k **in** range(0, dimensions[0]):
143. C[i][j] = C[i][j] + (A[i][k]\*B[k][j])
144. **return** C

147. **def** square\_matrix\_multiply\_strassens(A, B):
148. """
149. Return the product AB of matrix multiplication.
150. Assume len(A) is a power of 2
152. >>> A = [[36, 54, 24, 38], [54, 50, 19, 68], [26, 79, 57, 49], [94, 59, 20, 97]]
153. >>> B = [[46, 68, 27, 38], [57, 94, 74, 20], [46, 0, 52, 69], [20, 65, 37, 26]]
154. >>> square\_matrix\_multiply(A, B)
155. array([[  6598.,   9994.,   7622.,   5092.],
156. [  7568.,  12792.,   8662.,   6131.],
157. [  9301.,  12379.,  11325.,   7775.],
158. [ 10547.,  18243.,  11533.,   8654.]])
159. """
160. A = numpy.asarray(A)
161. B = numpy.asarray(B)
162. **assert** A.shape == B.shape
163. **assert** A.shape == A.T.shape
164. **assert** (len(A) & (len(A) - 1)) == 0, "A is not a power of 2"
165. dimensions = A.shape
166. C = numpy.zeros(shape=(dimensions[0],dimensions[0]))
167. **if** dimensions[0] == 1:
168. C[0][0] = A[0][0] \* B[0][0]
169. **else**:
170. *# Partition the given 2 matrices*
171. partition = dimensions[0]/2
172. A11 = A[:partition, :partition]
173. A12 = A[:partition, partition:]
174. A21 = A[partition:, :partition]
175. A22 = A[partition:, partition:]
176. B11 = B[:partition, :partition]
177. B12 = B[:partition, partition:]
178. B21 = B[partition:, :partition]
179. B22 = B[partition:, partition:]
181. *# Evaluate P*
182. P1 = square\_matrix\_multiply\_strassens(A11, B12 - B22)
183. P2 = square\_matrix\_multiply\_strassens(A11 + A12, B22)
184. P3 = square\_matrix\_multiply\_strassens(A21 + A22, B11)
185. P4 = square\_matrix\_multiply\_strassens(A22, B21 - B11)
186. P5 = square\_matrix\_multiply\_strassens(A11 + A22, B11 + B22)
187. P6 = square\_matrix\_multiply\_strassens(A12 - A22, B21 + B22)
188. P7 = square\_matrix\_multiply\_strassens(A11 - A21, B11 + B12)
190. *# Evaluate the product matrix*
191. C[:partition, :partition] = P5 + P4 - P2 + P6
192. C[:partition, partition:] = P1 + P2
193. C[partition:, :partition] = P3 + P4
194. C[partition:, partition:] = P1 + P5 - P3 - P7
196. **return** C
197. **pass**

200. **def** test():
201. C = [numpy.random.randint(-99, 99)]\*1
202. array\_length = numpy.random.randint(1, 20)
203. **for** x **in** range(1, array\_length):
204. C.append(numpy.random.randint(-99, 99))
206. brute\_force\_sub\_array = find\_maximum\_sub\_array\_brute(C, 0, len(C)-1)
207. crossing\_sub\_array = find\_maximum\_crossing\_sub\_array(C, 0, (len(C)-1)/2, len(C)-1)
208. recursive\_sub\_array = find\_maximum\_sub\_array\_recursive(C, 0, len(C)-1)
209. iterative\_sub\_array = find\_maximum\_sub\_array\_iterative(C, 0, len(C)-1)
210. **print**(C)
211. **print**(brute\_force\_sub\_array)
212. **print**(crossing\_sub\_array)
213. **print**(recursive\_sub\_array)
214. **print**(iterative\_sub\_array)
216. matrix\_size = 2\*\*numpy.random.randint(1, 3)
217. A = numpy.random.randint(99, size=(matrix\_size, matrix\_size))
218. B = numpy.random.randint(99, size=(matrix\_size, matrix\_size))
219. square\_matrix = square\_matrix\_multiply(A, B)
220. strassens\_matrix = square\_matrix\_multiply\_strassens(A, B)
221. **print**(A)
222. **print**(B)
223. **print**(A.dot(B))
224. **print**(square\_matrix)
225. **print**(strassens\_matrix)
227. **pass**

230. **if** \_\_name\_\_ == '\_\_main\_\_':
231. test()

Output for one of the random inputs:



Doctest and flake8 outputs:

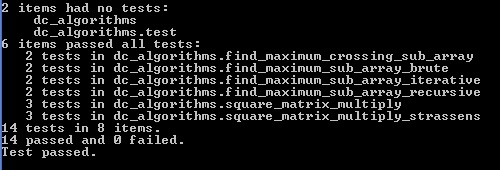
Input used for doctests:

*STOCK\_PRICE\_CHANGES =[13, -3, -25, 20, -3, -16, -23, 18, 20, -7, 12, -5, -22, 15, -4, 7]*

Expected Output:

*(7, 10)*

Doctest passed for all the functions as seen in below screenshot.



Output of flake8 (complexity 9):

