def topological\_sorting(self):  
 *"""verify if the corresponding graph is a DAG and performs a topological sorting of the  
 activities using the algorithm based on predecessor counters"""* sorted = list()  
 queue = deque()  
 count = {}  
 # see how many inbound edges each vertex has, if they don't have then add it to the queue  
 for vertex in self.parse\_vertices():  
 count[vertex] = self.in\_degree(vertex)  
 if count[vertex] == 0:  
 queue.append(vertex)  
  
 # start algorithm  
 # we take a vertex with no predecessors, we put it on the sorted list, and we eliminate it from the graph.  
 # Then, we take a vertex with no predecessors from the remaining graph and continue the same way.  
 while len(queue) != 0:  
 vertex = queue.popleft()  
 sorted.append(vertex)  
 for y in self.parse\_vertex\_out(vertex):  
 count[y] -= 1  
 if count[y] == 0:  
 queue.append(y)  
  
 # this mean we don;t have DAG  
 if len(sorted) < self.get\_number\_vertices():  
 sorted = []  
 return sorted  
  
def get\_times(self):  
 *"""earliest and the latest starting time for each activity and the total time of the project"""* sorted\_list = self.topological\_sorting()  
 # earliest time when activity begins and ends  
 for x in sorted\_list:  
 if self.in\_degree(x) == 0:  
 self.tmb[x] = 0  
 else:  
 # we check all the inbound vertexes and activity begins when all of them are done(max)  
 for y in self.parse\_vertex\_in(x):  
 self.tmb[x] = max(self.tmb[x], self.tme[y])  
 # activity ends = activity begins + duration  
 self.tme[x] = self.tmb[x] + self.duration[x]  
  
 # latest time when activity begins and ends  
 sorted\_list.reverse()  
 for x in sorted\_list:  
 if self.out\_degree(x) == 0:  
 # TME[X] is the duration of the project  
 self.TME[x] = self.tme[sorted\_list[0]]  
 for y in self.parse\_vertex\_out(x):  
 # we check all the outbound vertexes and activity ends when one of them starts(min)  
 self.TME[x] = min(self.TME[x], self.TMB[y])  
 # activity begins = activity ends - duration  
 self.TMB[x] = self.TME[x] - self.duration[x]  
  
 # return the 4 lists and the duration  
 return self.tme, self.tmb, self.TMB, self.TME, self.TME[sorted\_list[0]]  
  
def critical\_activities(self):  
 *"""A critical activity has a total time reserve of 0"""* a = self.get\_times() # just to calculate  
 arr = []  
 for i in range(self.get\_number\_vertices()):  
 # if latest time when activity begins is also the earliest time => critical activity  
 if self.tmb[i] == self.TMB[i]:  
 arr.append(i)  
 return arr

def topological\_sorting(self):  
 *"""verify if the corresponding graph is a DAG and performs a topological sorting of the  
 activities using the algorithm based on predecessor counters"""* sorted = list()  
 queue = deque()  
 count = {}  
 # see how many inbound edges each vertex has, if they don't have then add it to the queue  
 for vertex in self.parse\_vertices():  
 count[vertex] = self.in\_degree(vertex)  
 if count[vertex] == 0:  
 queue.append(vertex)  
  
 # start algorithm  
 # we take a vertex with no predecessors, we put it on the sorted list, and we eliminate it from the graph.  
 # Then, we take a vertex with no predecessors from the remaining graph and continue the same way.  
 while len(queue) != 0:  
 vertex = queue.popleft()  
 sorted.append(vertex)  
 for y in self.parse\_vertex\_out(vertex):  
 count[y] -= 1  
 if count[y] == 0:  
 queue.append(y)  
  
 # this mean we don;t have DAG  
 if len(sorted) < self.get\_number\_vertices():  
 sorted = []  
 return sorted  
  
def get\_times(self):  
 *"""earliest and the latest starting time for each activity and the total time of the project"""* sorted\_list = self.topological\_sorting()  
 # earliest time when activity begins and ends  
 for x in sorted\_list:  
 if self.in\_degree(x) == 0:  
 self.tmb[x] = 0  
 else:  
 # we check all the inbound vertexes and activity begins when all of them are done(max)  
 for y in self.parse\_vertex\_in(x):  
 self.tmb[x] = max(self.tmb[x], self.tme[y])  
 # activity ends = activity begins + duration  
 self.tme[x] = self.tmb[x] + self.duration[x]  
  
 # latest time when activity begins and ends  
 sorted\_list.reverse()  
 for x in sorted\_list:  
 if self.out\_degree(x) == 0:  
 # TME[X] is the duration of the project  
 self.TME[x] = self.tme[sorted\_list[0]]  
 for y in self.parse\_vertex\_out(x):  
 # we check all the outbound vertexes and activity ends when one of them starts(min)  
 self.TME[x] = min(self.TME[x], self.TMB[y])  
 # activity begins = activity ends - duration  
 self.TMB[x] = self.TME[x] - self.duration[x]  
  
 # return the 4 lists and the duration  
 return self.tme, self.tmb, self.TMB, self.TME, self.TME[sorted\_list[0]]  
  
def critical\_activities(self):  
 *"""A critical activity has a total time reserve of 0"""* a = self.get\_times() # just to calculate  
 arr = []  
 for i in range(self.get\_number\_vertices()):  
 # if latest time when activity begins is also the earliest time => critical activity  
 if self.tmb[i] == self.TMB[i]:  
 arr.append(i)  
 return arr