|  |  |  |
| --- | --- | --- |
| **Assignment Question number** | **Functions Used** | **Brief of approach taken** |
| **Question 1** |  |  |
| **Question 2** | Main function:  **improveDistance()** -> returns the blocked point requires for question 2 and the ratio of improvement  Function calls within main function:  **getPath()** -> returns all possible paths between two points  **findPaths()** -> finding paths in graph from source to destination  **computerPathDistance()** -> calculates the total distance of a path  **computeDistance()** -> calculates the distance between two adjacent points | Assumptions used:   * the optimum point to be blocked is the first point croc will travel to in the shortest path since if that point is blocked the croc cannot go any further in that path. * New alternative path is the second shortest path, and it is assumed the croc would travel on the second shortest path when the first shortest path is blocked * If there is only one possible path between two points, then that path will be the shortest path and since there is no alternative path new alternative path distance will be 0 which would make the ration of improvement a indefinite very large number (because any number divided by 0 is indefinite and very large) |
| **Question 3** | Main functions:  **countCroc**() - count the number of crocs likely in a x mile radius of a beach.  **locateOptimalBlockage**() - return the point blocked such as A1 and the increase in protection provided using some weighting  **minTime**() - return list of points travelled and the time required  Functions call within main functions:  **addEdge**() - add the edges between start point and end point  **isNotVisited**() - to check if current vertex is already present in path  **findpaths**() - finding paths in graph from source to destination  **getPath**() - returns all possible paths between two points  **computePathDistance**() - calculates the total distance of a path  **computeDistance**() - calculates the distance between two adjacent points | Assumptions used:   * Find the minimum distance by using find the shortest path from source to destination with minimum time consumption. * Provided the number of crocodiles in a certain radius x mile of a beach. Find all crocs within a mile radius of the shore and add them to the list. Find path from beach to all crocs in list * Find the optimum path point to insert a blockage that can make the beach safer. Get the possible path, and find equal to the input value, then append in the next path and s1 will equal to the neighbor value. Find neighbor in path list, block this location. |