Algorithm Description:

Overview:

This program was designed based on the Held-Karp algorithm BUT with a tweak for intermediate optimal path calculations. While the Held-Karp algorithm has a disadvantage as it utilizes a lot of space when computing, the program tries to tackle this by utilizing a specialized optimal path generator (**get_opt_path_and_cost()**) for intermediate iterations prior to the final iteration where all of the subproblems have been computed and stored which allows the final optimal solution to be computed by the Held-Karp algorithm directly. This is done by taking the intermediate calculated subproblems and using a set of conditions to move around the nodes in the path and check if this increase optimality.

Specifically, for intermediate iterations, the program contains a function, **get_opt_path_and_cost()** that computes an updated optimal path based on the just-computed subproblem. If the resulting optimal path results in a decreased total path cost, the changes are kept, otherwise, they are reversed. The algorithm updates both the list variable holding the optimal path and optimal cost but also a "weighted" optimal path which includes information about the current best from_node and the cost from it directly and attaches this information to the node inside the "weighted" optimal path. According to tests done, the **get_opt_path_and_cost()** allows for the intermediate optimal path and cost to converge to the true optimal path and cost, resulting in it being equal or very close to the true values respectively.

Algorithm Description:

Prior to the algorithm executing:

- Begins by processing all user input (input file, output file, time limit)
- The cost matrix (**cost_matrix**) is built using the using the nodes' x and y coordinates by computing the distance between all nodes using the Euclidean distance.
- The starting optimal path (**opt_PATH**) is initialized going from the starting node and back while traversing all nodes in order of magnitude. Note that this path will be updated consistently during the running of the program using **get_opt_path_and_cost()**.
- A variant of the optimal path that is used to make choices in terms of changing the optimal path is also initialized (**weighted_opt_PATH**). This matches the optimal path except that for every node id it also has next to it the current best cost to it and the immediate parent that led to this cost. Initially, this portion for each node is initialized to None and is updated as subproblems are calculated via **get_best_cost()** and then passed over to **get_opt_path_and_cost()**.

*The main algorithm function (**held_karp_variant()**) then begins*:

- A dictionary (**traversal_order**) for key-value pairs is initialized. This will store the best cost for each subproblem going from smallest to largest (which is ultimately the original problem)
- A single for loop gets all of the costs from the start node to every other node directly and stores these results in the dictionary.

- Then, for each problem size (more specifically, the size of the subset) a subset of nodes is chosen and a node that is not part of this subset is also chosen as the target node (denoted as **to_node**) and the best cost to this node is calculated within the helper method (**get_best_cost()**) and returned.
    - **Ex:** given key: [2, {3,4}] (**to_node** would be 2 in this case) you would use (**get_best_cost()**) to look at all possible scenarios that could result in the arrival to node 2 with the given subset {3,4} which ultimately leads back to the start node 1. This could include a scenario of choosing 3 as the immediate parent and then adding it to the cost to get from 4 to 3 and then adding it to the cost to get from 1 to 4. This would then result in the returning of the value pair of the optimal cost and the immediate parent that led to this optimal cost (**ex**: [445, 3] so the best cost is 445 and the immediate parent that led to this is 3 (denoted as (**from_node**))).
- Upon returning to the main algorithm function (**held_karp_variant()**) the newly found value is set for the key (i.e: the current subproblem) and then the result is also passed into the optimal path and cost generator function (**get_opt_path_and_cost()**).
- **get_opt_path_and_cost()** is the function that updates the current best optimal path (**opt_PATH**) based on the key-value pair that was just calculated using **get_best_cost()** and calculates the new cost based on these changes made to the optimal path. If these changes result in an improvement or no change to the current best cost, the changes are kept, otherwise the changes are reversed.
    - More specifically, there are 3 cases that the function deals with:
    - Note that below the best parent node is the immediate parent that was returned along with the best cost as part of the value from **get_best_cost()**.
        - 1. If the weighted optimal path has no value for the best cost and immediate parent pair (this happens for all nodes at the beginning): just add the costparent pair to the weighted opt path.
        - 2. If both the **to_node** and **from_node** have cost-parent pairs in the weighted opt path then:
- 2.1 If the previous value for cost in the cost-parent pair of the target node within the weighted optimal path is more than the cost just found in the subproblem:
    - If the target node and parent node (i.e.: the one that is now the best immediate parent as part of the solution from **get_best_cost()**) are next to each other:
        - swap the positions of the target node and best immediate parent in the optimal path and weighted optimal path and update the target node's best cost and immediate parent pair to be the value returned from **get_best_cost()** in the weighted optimal path.
    - Else:
        - Put the target node right before the newly found best immediate parent in the optimal path and weighted optimal path and update

the target node's best cost and immediate parent pair to be the value returned from **get_best_cost()**.

- ✦ If any of the above cases are not valid then the target node's best cost and immediate parent pair in the weighted optimal path are set to be the value returned from **get_best_cost().**

- As stated before, any changes that result in a decrease in optimality will be reversed.
- For the final subproblem, both the true cost and true optimal path will be calculated by **get_best_cost()** and simply retuned by the **get_opt_path_and_cost()** function without any changes being done.

Notes:

- The program prints the current best cost on one line and the current best path on the next line. Please comment the line with the note: **# NOTE: COMMENT FOR ALL INTERMEDIATE OPT. PATH & COST** to see all intermediate solutions
- There are several commented out print statements in order to see more detailed outputs for each iteration. They are listed with the comment starting off with **#TEST:** ...
- For more information please check the source code.