|  |  |
| --- | --- |
| CLIENT | Message enterprice |
| USER | Delivery courier |
| FUNCTIONAL REQUIREMENT | R1: determínate better rute  R2: add place represented by vertex.  R3: add route represented by edges.  R4: add weight to the Edges.  R5: minimize the distance of the Delivery Courier for the deliveries.  R6: Allow the messenger to start at any intersection. |
| CONTEXTO DEL PROBLEMA |  |
| NO FUNCTIONAL REQUIREMENT | RN1: Accuracy: Although an approximate approach is used, the algorithm should provide solutions as close as possible to the optimal route. The goal is to minimize the total distance traveled by the courier to deliver all packages.  RN2: Scalability: The algorithm must be able to efficiently handle large sets of intersections and streets. It must be scalable to adapt to cities with a large number of destinations.  RN3: Flexibility: The system must allow the addition or removal of destinations in real time without affecting the overall performance. This implies that the algorithm must be able to update the optimal route efficiently when changes are made to the set of destinations. |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | RF1: determínate better rute | | |
| Summary | Determine the optimal route to deliver packages to different destinations in a city. | | |
| Inputs | **input name** | **Datatype** | **Selection or repetition condition** |
| destination | T value |  |
| General activities necessary to obtain the results | 1. type the packet destination,  2. the program calculate the optimal route of the deliver.  3. the program show the designed route in console. | | |
| Result or postcondition | Optimal route made an showed. | | |
| Outputs | **output name** | **Datatype** | **Selection or repetition condition** |
| toString | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | RF2: add place represented by vertex. | | |
| Summary | Add place represented by a vertex in a graph. | | |
| Inputs | **input name** | **Datatype** | **Selection or repetition condition** |
| vertex | int |  |
| General activities necessary to obtain the results | 1. type the adress of the place  2. the program save the vertex in the list.  3. confirmation message. | | |
| Result or postcondition | New place saved. | | |
| Outputs | **output name** | **Datatype** | **Selection or repetition condition** |
| msj | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | RF3: add route represented by edges. | | |
| Summary | Add route represented by a edge in a graph. | | |
| Inputs | **input name** | **Datatype** | **Selection or repetition condition** |
| intialAdress | int |  |
| finalAdress | int |  |
| General activities necessary to obtain the results | 1. type the initial and the final destination.  2. the system assing the route  3. confirmation message | | |
| Result or postcondition | New route saved | | |
| Outputs | **output name** | **Datatype** | **Selection or repetition condition** |
| msj | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | RF4: add weight to the Edges. | | |
| Summary | Assign weights to edges representing the distance between intersections. | | |
| Inputs | **input name** | **Datatype** | **Selection or repetition condition** |
| weight | int |  |
| inicialValue | int |  |
| finalValue | int |  |
| General activities necessary to obtain the results | 1. type the inicialValue and the finalValue to find the Edge.  2. the program search the Edge.  3.1 if the program not find the Edge, the program dont print a error message.  3.2 if the program find the Edge, the user Will type the weight to asing to the respetive Edge.  4 the program show a confirmation message. | | |
| Result or postcondition | Weight assigned. | | |
| Outputs | **output name** | **Datatype** | **Selection or repetition condition** |
| toString | String |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Name or identifier | RF5: minimize the distance of the Delivery Courier for the deliveries. | | |
| Summary | Find the shortest route to deliver all packages. | | |
| Inputs | **input name** | **Datatype** | **Selection or repetition condition** |
| inicialValue | int |  |
| finalValue | int |  |
| General activities necessary to obtain the results | 1. type the inicialValue and the finalValue to find the initial and final point to design the route.  2. the program Will calculate the optimal route.  3 the program show the order of the vertex in the console. | | |
| Result or postcondition | Optimal designed route | | |
| Outputs | **output name** | **Datatype** | **Selection or repetition condition** |
| toString | String |  |

SEARCH FOR CREATIVE SOLUTIONS:

To tackle the problem of finding the optimal parcel delivery route in a city represented by a network, you can use the "traveling salesman" algorithm (also known as the TSP, Traveling Salesman Problem). Here are some solution ideas:

**Nearest Neighbor Algorithm:** This algorithm starts at an initial node and, at each step, selects the nearest unvisited node as the next destination. It continues this process until all nodes have been visited and then returns to the starting point. This approach is simple and fast, but does not guarantee the optimal path.

**Closest insertion algorithm:** This algorithm starts with an initial path containing two nodes and, at each step, selects the closest unvisited node and inserts it at the position that minimizes the total path distance. It repeats this process until all nodes have been visited and then returns to the starting point. This approach can produce better solutions than the nearest neighbor algorithm, but still does not guarantee the optimal path.

**Branching and pruning algorithm:** This algorithm uses an exhaustive search strategy to explore all possible routes. It starts at an initial node and, at each step, generates all possible extensions of the current path and evaluates them. Only those extensions that have the potential to lead to an optimal solution are further explored. A pruning technique is used to avoid exploring branches that clearly will not lead to the optimal solution. This approach guarantees finding the optimal path, but can be computationally expensive for large instances of the problem.

**Dynamic programming algorithm:** This approach is useful when the number of nodes is relatively small. A table is used to store and compute the minimum distances between all combinations of nodes. The table is iteratively filled and used to construct the optimal route at the end of the process. This algorithm is also guaranteed to find the optimal path, but can require a significant amount of time and memory, especially for larger problems.

**Genetic algorithm:** This approach is based on principles of biological evolution and uses heuristic search techniques. It starts by generating an initial population of possible routes (chromosomes) and applies genetic operators such as selection, crossover and mutation to evolve the population in each generation. The quality of each pathway is evaluated by a fitness function that considers the total distance traveled. Over time, the population tends to converge to more optimal solutions. This approach can be effective in finding good, but not necessarily optimal, solutions and is useful when there are time or computational resource constraints.

**Ant colony optimization algorithm:** This algorithm is inspired by the behavior of ants as they search for the shortest route between their colony and a food source. Ants deposit pheromones on the ground, and other ants follow these clues to find efficient paths. In the context of the packet delivery problem, the nodes of the graph would be the intersections and the pheromones would be represented as additional information on the edges. The ants would perform exploratory traverses and, based on the number of pheromones encountered, make decisions to construct the optimal route. This approach is especially useful when there are multiple packages and additional constraints, such as time windows for delivery, must be considered.