TAD : Grafo

Grafo : (n1, n2)(n3, n4)…(n\_n-1, n\_n)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inv : there’re various types of graphs, the are directed, non-directed, with weight and without weight   |  |  |  | | --- | --- | --- | | OPERACIONES | INPUTS | OUTPUTS | | Bfs | Graph or AdjList | Connectivity | | DFS | Graph or AdjList | Connectivity | | FloydWarshall | Graph or AdjList | Path with less weight | | Dijkstra | Graph or AdjList | Path with less weight | | Prim | Graph or AdjList | Tree | | Kruskal | Graph or AdjList | Tree | |

|  |
| --- |
| BFS |
| <Analyzer> |
| runs through the nodes of a graph, starting at the root, then exploring all the neighbors of this node. Then, for each of the nodes are scanned their respective adjacent nodes, and so until the entire graph is traversed, if the node is found before traversing all nodes, concludes the search.  {Pre: a graph that has not been discovered must be sent by parameter. }  {Pos: It runs the graph completely discovering its vertices and finding its predecessors and the distance between them} |

|  |
| --- |
| DFS |
| <Analyzer> |
| When there are no more nodes left to visit, it returns to the predecessor node, so it repeats the same process with each of the node’s neighbors. If the node is found before traversing all nodes, the search concludes. >"  {Pre: a graph that has not been discovered must be sent by parameter }  {Pos: Go through the graph to find all the vertices and record their predecessors and the time it took to discover each node} |

|  |
| --- |
| FloydWarshall |
| <Analyzer> |
| Initialize the solution matrix as the input chart matrix as the first step.  Then update the solution matrix by considering all vertices as an intermediate vertex.  The idea is to choose one by one all vertices and update all shorter paths including the chosen vertex as an intermediate vertex on the shortest path.  {Pre: you have to send by parameter a graph to which you are going to search for the minimum paths between their vertex pairs}  {Pos: Deliver the minimum weight of each path between pairs of vertices } |

|  |
| --- |
| Dijkstra |
| <Analyzer> |
| It consists of exploring all the shortest paths starting from the origin vertex and leading to all the other vertices; when the shortest path is obtained from the origin vertex to the rest of the vertices that make up the graph, the algorithm stops.  {Pre: you have to send by parameter a graph to which you will find a shorter path }  {Pos: Give the shortest path to the rest of vertices of the graph given an initial node} |

|  |
| --- |
| Prim |
| <Analyzer> |
| a minimum expansion tree. That is, it is able to find a subset of the edges that form a tree that includes all the vertices of the initial graph, where the total weight of the edges of the tree is the minimum possible.  {Pre: you have to send by parameter a related graph, not directed and weighted }  {Pos: Deliver a minima expansion tree} |

|  |
| --- |
| Kruskal |
| <Analyzer> |
| looks for a subset of edges which, forming a tree, include all vertices and where the total value of all edges of the tree is the minimum. If the graph is not connected, then look for a minimum expanded forest  {Pre: a related and weighted graph has to be sent by parameter}  {Pos: Deliver a minimum coating tree} |