**Assumptions**

* The graph of cities form a connected graph.
  + The current representation makes it a completely connected graph. The Salesman can go from any city to any other city
* The tour always start from city A and return back to city A

**Operations**

* Expanding a node
  + generating all unvisited nodes from that point
  + generating all unvisited cities that can be visited next
* Move to next state
* Goal test on a node
* ClosedList represent the list of visited cities
* OpenList represents the list of cities that can be visited next
* Search function arguments
  + Initial State
    - Which means a state should be a class

**Problem Representation**

* Let each node of the state graph be a path represented by ordered list – city1->city2->………->cityn where n can be 0 to n where n is the total no of cities
* Initial State : A
  + The salesman is at city A
* Goal State : A->x1->x2->…….->xi->A where xi belongs to Set of cities – A
* Goal test: for a state represented by path p the goal test is if path that starts at A and ends at A and contains all cities in between exactly once in any order
* Cost(Path) – returns the cost to cover all the cities as listed in the path in that order
* The Solution that we want : cost(goal State)

**Implementation Details**

* a method for calculating step cost (Euclidean distance) from the 2 cities
* a method that takes a set of nodes and edges and edge costs and returns the MST path sum
* maintain an Open and closed list
* What should each node of the graph represent?
  + The tour so far
  + The current city
  + The path distance so far
* We need a priority queue
  + The priority of a node n should be f(n) = g(n) + h(n)
    - G(n) – path cost from A to n
    - H(n) – MST path of all unvisited nodes from n + nearest distance from an unvisited city to A
* A method to load data from a file
* A dictionary of cities
  + Key is cityId and value is (x,y) tuple
* A dictionary of paths
  + Key is (city1, city2) tuple and value is Euclidean distance
* Graph DS from finding MST
  + Dictionary
    - Key is city
      * Value is a list of tuples – (toCity, distance between city)
* Operations on a node:
  + We need to be able to generate the successor of that node
    - And add it to the queue
  + We also need to maintain a closed list – which keeps track of citites already visited
  + We need to be able to calculate the cost of that path
    - The assumption is that all paths starts from A
    - This will be the value of g(n)
  + H(N) -> The cost to get to that node(cost of adding the node to the current path) + mstCost(of all univisted cities starting from that node n) + lowest cost of getting from one of the univisted cities back to A
    - This H(n) is a heurisitc because
      * H(goal state) = 0
        + adding starting city A to the current path(which already ends at A) + mstPathCost(remaing nodes) + cost of getting back to A = 0
    - This H(n) is an admissible heuristic because
      * 0<=h(n)<=h\*(n)