The Nearest Neighbor Algorithm

Algorithm Description

The nearest neighbor algorithm is a greedy algorithm that solves the traveling salesman problem by continually choosing the nearest unvisited city to the current city until all cities have been visited. True to its nature as a greedy algorithm, the algorithm runs quickly and effectively. When given randomly generated city data the greedy algorithm will return a solution that is 20-25% longer than the optimal solution.

Test File	**Algorithm Solution**	**Optimal Solution**	**Ratio**
tsp_example_1.txt	130,921	108,159	1.21
tsp_example_2.txt	3,115	2,579	1.21
tsp_example_3.txt		1,573,084	

While the greedy algorithm does run quickly and is easy to implement, there are some arrangments of cities which can make the nearest neighbor algorithm give the worst route. For example, it has been shown that "for every $n \geq 2$ there is an instance of ATSP (STSP) on n vertices for which [the greedy algorithm] finds the worst tour." [1]

Justifications

We chose the nearest neighbor (greedy) algorithm because it is relatively easy to implement and still works quickly and efficiently. With an average solution of 20-25% worse than the optimal solution, the algorithm also meets the requirments for this project.

Pseudocode

```
def distance_squared(c1, c2):
  return (c1['x'] - c2['x'])**2 + (c1['y'] - c2['y'])**2

# cities is an array of city objects which have an id, x-coordinate, and y-coordina
def get_nearest_neighbor(cities, city):
  # Dictionary for selecting nearest neighbor
  neighbors = {}

  # Add all distances_squared to neighboring cities to dictionary
  for neighbor in cities:
    neighbors[distance_squared(city, neighbor)] = neighbor

  # Return neighbor with least distance
  nearest_neighbor = neighbors[min(neighbors)]
  distance = int(round(sqrt(min(neighbors))))

  return nearest_neighbor, distance

def TSP_nearest_neighbor(cities):
  tour = []
  min_distance = infinity

  for city in cities:
    total_distance = 0

    # Start on arbitrary vertex.
    visited = [city]
    unvisited = []

    # Add all cities to unvisited list except the starting city.
    for city in cities:
      if city is not city:
        unvisited.append(city)

    # find an unvisited nearest neighbor, marked it visited, and add it's distance.
    while len(unvisited) > 0:
      nearest_neighbor, neighbor_distance = get_nearest_neighbor(unvisited, visited[-
      visited.append(nearest_neighbor)
      unvisited.remove(nearest_neighbor)
      total_distance += neighbor_distance

    # add the distance between the first and last city to complete the tour.
    total_distance += round(sqrt(distance_squared(visited[0], visited[-1])))

    if total_distance < min_distance:
      tour = visited
      min_distance = total_distance

  return tour, min_distance
```