**Author**

Han Deng, Xinran Hu

**System Description**

We build an application that stores information about US cities and road distance between cities. The application provides a console interface allows a user to perform the following operations

(1) List all cities

(2) Add a city

(3) Add a road

(4) Search cities within a radius of an origin city, and sort result by either name or distance

For sorting in (4), we will use two sorting algorithms and compare them from the aspect of runtime and number of comparisons made

**Data Structure and Algorithm**

**Data Structures**

1. Undirected, Weighted graph

* City information is modeled by a undirected, weighted graph. Each city is a vertex and each road connection is an edge. The weight of each edge is the distance

1. Queue

* During the search of shortest path originated from the origin, paths are stored in a queue

1. HashMap

* A Hashmap is used to store (city, shortest distance)

1. Array

* Cities and their shortest distances are store in an array and passed to sorting algorithm

Algorithms

1. BFS

* BFS is used to search cities within a radius

1. Quicksort (Compared with MergeSort)

* Quicksort is used to sort cities by user-provided criteria

1. Mergesort (Compared with QuickSort)

* Mergesort is used to sort cities by user-provided criteria

**Algorithm Compared**

We compared two sorting algorithms: QuickSort and MergeSort.

We investigated three main aspects: Runtime, Number of Comparison and in-place-ness.

**Algorithm Comparison Result:**

**CPU: Core-i5 1.7 Ghz**

**Case 1: Sort cities by distance within 2000 miles from Chicago**

|  |  |  |
| --- | --- | --- |
|  | **Quick Sort** | **Merge Sort** |
| Run-time | *84.0507 ms* | *69.0467 ms* |
| # of Comparison | *47* | *11* |
| In-place | *Yes* | *No* |

**Case 2: Sort cities by name within 5000 miles from New York**

|  |  |  |
| --- | --- | --- |
|  | **Quick Sort** | **Merge Sort** |
| Run-time | *270.1602 ms* | *520.3815 ms* |
| # of Comparison | *99* | *25* |
| In-place | *Yes* | *No* |

**Test**

We have taken multiple measures to ensure the correctness of the application.

(1) Unit-test:

We ran unit test for individual functions

(2) Integration-test:

We tested various choices of origin cities: cities in the Midwest, Easter Coast and West Coast. We also tested with

**Description of Result**

**Advantage of quick sort over merge sort:**

Quicksort is, in general, a faster algorithm than merge sort. Since quicksort has expected run-time complexity of O(nlogn) and its constant is generally small compare to other algorithms.

However, if the pivot selection is simple and data is largely sorted, the number of comparison is large and the algorithm degraded into O(n^2) run time complexity. This is a realistic problem for our use case since our search algorithm tends to return cities that already sorted.

One important advantage of quicksort over merge sort is that quicksort can be implemented in-place much easier than merge sort. This gives quicksort an large advantage of sorting long arrays.

Advantage of merge sort over quick sort

The biggest advantage of merge sort is its worst case run time is same as average run time. They are both O(nlogn), which is also the best complexity any comparison based sort can achieve. This means merge sort is robust to the form of input.

The number comparisons use by merge sort is less than that of quick sort. This may due to the fact data is likely sorted in our application.

The out-of-place implementation of merge sort is also simpler than quick sort. However, in-place implementation is difficult. This makes merge sort inefficient in sorting long list.