Amber Liu
Al7bf
12/06/2018
postlab11.pdf

Complexity Analysis

Prelab

Space Complexity

Node:
String = 4 bytes
Vector for adjacent = 4 bytes * n
Int indegree = 4 bytes
Total: 4 bytes * n + 8

Main:
Vector of n elements = 4 bytes * n
2 boolean = 2 bits
2 int = 2 * 4 bytes = 8 bytes
Total: 4*n bytes + 8 bytes + 0.2

Sorting:
queue = sizeof(n)*2 byte

Time Complexity

Main function reads in string from the file and searches adjacent list, this is a O(n) operation for each step search through out list of n nodes. The push back is also performed on the node when the string is already in the list vector. This is O(n), and when the two operations are repeated it ultimately has a run time of O(n^2).

inlab

space complexity

middle earth class
3 int variables = 4 bytes * 3 = 12
vector string cities = 4 *num cities
vector float xpos = 4 * xpos
vector float ypos = 4 * ypos
float distance = 4 bytes
map indicies = 4 bytes * 4 bytes * indicies

main
middleearth = total of above
vector<string> dests = 4 bytes * num cities
2 string variables = 8 bytes

vector<string> = 4 bytes * num cities
float distance = 8 bytes
float small dist = 8 bytes
Total = 2(4*num cities) + 24

Computedistance
Float sum = 8 bytes
Vector<string> dests[i] = 4 bytes
String start = 4 bytes
Total= 16

Time Complexity
Elements in the vector is set by total cities in Middle earth, so we don't need to resize the vectors. Push_back() operation would have a running time of O(1). The array accesses the vectors, xpos, ypos, that are all values already set and known so the 2d array also runs at O(1). The constructor operates at O(n). The next_permutation will have n! permutations where n is the number of cities, so the overall running time is O(n!)

Acceleration techniques
Minimum Spanning Tree
This method uses MST to reduce number of vertices visited. A node on the trip is removes and MST is constructed with the beginning city as the root of the tree. Then smallest vertices between nodes are set to most efficient path. It is ½ of what the total cost of a full walk through is because MST structure constraining the edges to be visited at most 2 times. It is a 2-approximate algorithm since the runtime is at most twice from its best possible output.

Christofide's Algorithm
The algorithm looks for shortest spanning tree, where it has shortest branches linking all cities together. The algorithm would find shortest distance between the cities, then find another short distant to a new city where one of the two is added to the tree. It then converts into a round trip after going through all branches two times. At worst, it will be twice as long as its best solution.

Nearest Neighbor
The technique starts from the first city and finds the shortest edge from the start city to another city, and marks that the visited city. The next city is dependent on the previous one and requires keeping track of visited and unvisited cities. After all cities are visited, there is a fairly short route created. Although not the fastest algorithm, is has a running time of O(log n).

https://www.geeksforgeeks.org/travelling-salesman-problem-set-2-approximate-using-mst/
https://www.wired.com/2013/01/traveling-salesman-problem/
https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm
https://www.mathworks.com/matlabcentral/fileexchange/25542-nearest-neighbor-algorithm-for-the-travelling-salesman-problem