Brian Aguirre
ba5bx
postlab11.pdf

Time:

Time is a huge factor when trying to solve both the pre lab and the in lab. As mentioned in the post lab instructions, the inefficiency of the solution for the traveling salesman problem is greatly seen through the time it would take to solve the following input: 20 20 20 14 20 as command line parameters (20 route tour through Middle Earth), over 300 thousand years! Thus, something we see as a huge factor is the number of cities and the number within the map (as it takes time to populate) and thus increased our biggest second factor which are the cities one must visit.

The number of cities is possibly one of the most time consuming of these parameters or operations since the iteration between having to recalculate from one city to the next takes a lot of time and computational power. Based on factorials, since there are x number of cities to visit, then the possibilities and worst case scenario is O((x-1)!). Then since this information is stored in a vector, same as what was done in the pre-lab portion. Filling up the vector and traversing through it ends up being a linear operation dependent on the size of x, which as stated before is the number of cities to visit. The combination of these two, after some simplifying (O((x-1)!) *O(x)) = O(x!).

Space:

The space, or memory used by the algorithms seen in the pre-lab and in-lab mostly revolve around the vectors used to store the information. When the map is created, the cities, names and position are stored within one vector. This vector serves as the master information holder for the map. There exists another vector, which is talked about in the optimization for the algorithms which contains the distances between the cities. This information are held as constants within this second vector, which add up to the total space for the program. In terms of time in accessing each one, since the first vector contains the cities' positions, there are used as coordinates, there being a horizontal component and vertical one. Thus in order to access both, which is the product of the two linear models is O(n) * O(n) = O(n^2). Then the other vector is one dimensional, resulting in O(n).

Optimization:

Note that in order to optimize all of the following possible solutions for the problem, was was done is already calculate the distances between the cities and place them within a vector (as previously mentioned). Accessing the vector results in a constant time rather than having to recalculate the distance between two cities based on coordinates.

Solution 1: Branch and Bound

This method involves by first selecting a node and setting up upper bounds cub that the algorithm then chooses a minimum. Unless another bound is found, then you continue going through all the edges until all of them have been visited. Apparently it does well and is faster but since it's an approximation technique (according to wikipedia), then there is a lot of debate whether it will hold true for different quantities or complexities of input.

Solution 2: Linear Solution ("Greedy Algorithm")

The concept behind this one is that once you are at your starting node, you then move one by one based on shortest distance then continue to recycle through the starting point until you have gone to every single node and have come back to the true starting point. Again, technically it is faster and less complicated than the brute force method but ultimately there can be a lot of resource that go behind this algorithm because you are moving through nodes and eventually checking the amount of nodes whether you have been there and approximating values can hurt the overall efficiency.

Solution 3: Branch and Cut:

This algorithm involves going through an already filed set (in the form of a vector in our case) and seeing the edges between the nodes. Once the edges between the nodes have been found, the corresponding distance is then retrieved from the constant vector. If the connections between certain nodes results in a much more efficient step (for example. starting from node a, then traveling to node b is distance 1, and from node c to d is distance 2, what if the overall distance from node a to d < a to b, and b to c < c to d, then that path should be considered.) Once again, due to the approximation that goes behind all of these algorithms, then there can be a high efficiency cost resulting in the computations of this one as well. Thus not all the time is it worth doing something simpler as it could be more taxing in terms of resources.

Sources:
http://en.wikipedia.org/wiki/Branch_and_bound
http://en.wikipedia.org/wiki/Linear_programming
http://en.wikipedia.org/wiki/Branch_and_cut

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Traveling Salesman*. John Wiley and Sons, 1986.
Emanuel Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley and Sons, 1998.