CECS-545: Project #1

Traveling Salesperson Prob.

Raudel

Using C programming Language,
MacBook Pro 16gb Ram, i7 Core, and 500gb SSD

Raudel Valdes
CECS
Speed School of Engineering
University of Louisville, USA
r0vald06@louisville.edu

1. **Introduction** (What did you do in this project and why?)

In this project we attempted to find a solution to an iteration of the Traveling Salesperson Problem. The traveling salesperson is given a list of cities to which it must visit, but there is a catch. The salesperson must be able to find the optimal path (least amount of total distance) in which it can visit all of its cities and end at its starting city. This city path is also known as the Hamiltonian Cycle which, in technical terms, it is a path in an undirected or directed graph that visits each vertex exactly once.

The reason we did this project was to practice different approaches for solving problems and to understand how each approach can have different results. The way that we targeted this problem was by using brute force and it allowed us to realize that this is not an optimal way of finding a solution. With brute force it is very likely that our solution method can require a non-feasible amount of computation time as well as memory. As computer engineers we want to create algorithms that take shortcuts and optimal approaches to produce a correct answer within a reasonable amount of time and computing resources.

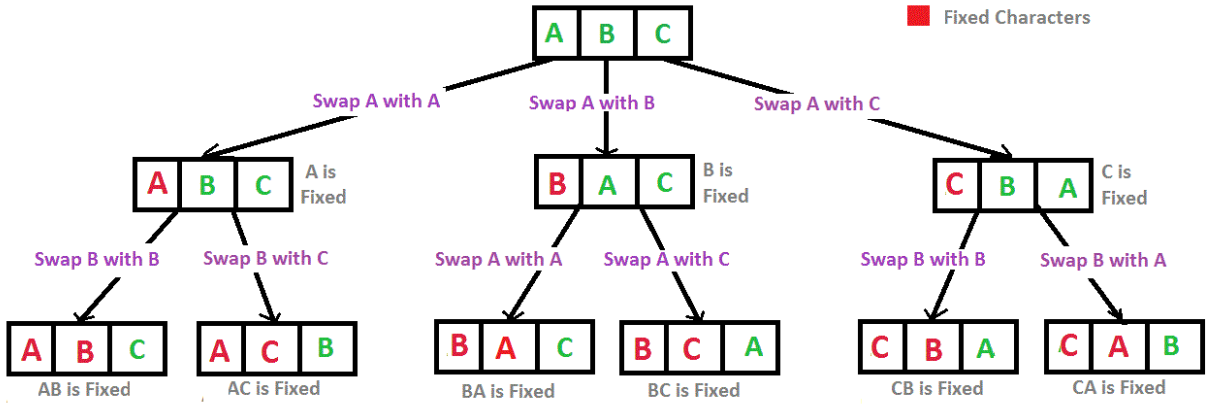
2. **Approach** (Describe algorithm you are using for this project)

As part of the assignment, I had to tackle the traveling salesman problem using a brute force approach. The brute force approach calculates every single permutation from the list of cities that the salesperson must visit. This approach was taken in order to understand how there are efficient and inefficient ways of solving problems.

My algorithm scans a file and accesses the list of cities with their X and Y coordinates. With the coordinates, I am able to calculate the distance between every two possible cities and store it within a dynamic 2D-array (matrix). Once the matrix is generated, a dynamic array with the int size of the total number of cities is passed to a function called permuteCities(). This function is in charge of permuting the array which contains in each of its index an int value that corresponds to a city.

The function permuteCity(), calls the function swapCities() which is in charge of moving the cities within the array by using pointers. PermuteCity() uses recursion in order to implement a backtracking algorithm of time complexity O((n)n!). The image below is from a website called geeksforgeeks.com and they implemented a similar approach which is depicted in the image. With this algorithm the program is able to generate every possible path and at the same time access the distance matrix to calculate the total distance for every new path that is generated.

Every path, its path number, and its total distance is currently being stored inside of an array of structs that is dynamically allocated. Inside of the struct you will find an attribute with names and types of int *path, double totalDist, and int pathNumb. With this array of structs I am able to loop through and find which paths contain the smallest distance and I save those to another array of structs that only contains the optimal paths.



Recursion Tree for Permutations of String "ABC"

3. **Results** (How well did the algorithm perform?)

My algorithm was probably not the most efficient in terms of time and computational resources, but it delivers the correct total distance as well as the paths every time in an average of five and a half minutes.

3.1 Data (Describe the data you used.)

The data that I used were .tsp files which contained various information about the cities. The file contained the number of cities in the list as well as the X and Y coordinate for each city. I was able to use the X and Y coordinates with the distance formula to calculate the distance between any two possible cities.

3.2 Results (Numerical results and any figures or tables.)

Please provide a file:
random4.tsp

Optimal Dist: 215.085533
Paths with Optimal Dist: 2
1 -> 4 -> 2 -> 3 -> 1
3 -> 2 -> 4 -> 1 -> 3

Please provide a file:
random5.tsp

Optimal Dist: 139.133542
Paths with Optimal Dist: 9

1 -> 2 -> 5 -> 3 -> 4 -> 1
1 -> 4 -> 3 -> 5 -> 2 -> 1
2 -> 1 -> 4 -> 3 -> 5 -> 2
2 -> 5 -> 3 -> 4 -> 1 -> 2
3 -> 4 -> 1 -> 2 -> 5 -> 3
3 -> 5 -> 2 -> 1 -> 4 -> 3
4 -> 3 -> 5 -> 2 -> 1 -> 4
4 -> 1 -> 2 -> 5 -> 3 -> 4
5 -> 3 -> 4 -> 1 -> 2 -> 5

Please provide a file:
random6.tsp

Optimal Dist: 118.968914
Paths with Optimal Dist: 12
1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 1
1 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1
2 -> 1 -> 6 -> 5 -> 4 -> 3 -> 2
2 -> 3 -> 4 -> 5 -> 6 -> 1 -> 2
3 -> 2 -> 1 -> 6 -> 5 -> 4 -> 3
3 -> 4 -> 5 -> 6 -> 1 -> 2 -> 3
4 -> 3 -> 2 -> 1 -> 6 -> 5 -> 4
4 -> 5 -> 6 -> 1 -> 2 -> 3 -> 4
5 -> 4 -> 3 -> 2 -> 1 -> 6 -> 5
5 -> 6 -> 1 -> 2 -> 3 -> 4 -> 5
6 -> 5 -> 4 -> 3 -> 2 -> 1 -> 6
6 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6

Please provide a file:
random7.tsp

Optimal Dist: 63.863032
Paths with Optimal Dist: 14

1 -> 2 -> 7 -> 3 -> 6 -> 5 -> 4 -> 1
1 -> 4 -> 5 -> 6 -> 3 -> 7 -> 2 -> 1
2 -> 1 -> 4 -> 5 -> 6 -> 3 -> 7 -> 2
2 -> 7 -> 3 -> 6 -> 5 -> 4 -> 1 -> 2
3 -> 6 -> 5 -> 4 -> 1 -> 2 -> 7 -> 3
3 -> 7 -> 2 -> 1 -> 4 -> 5 -> 6 -> 3
4 -> 1 -> 2 -> 7 -> 3 -> 6 -> 5 -> 4
4 -> 5 -> 6 -> 3 -> 7 -> 2 -> 1 -> 4
5 -> 4 -> 1 -> 2 -> 7 -> 3 -> 6 -> 5
5 -> 6 -> 3 -> 7 -> 2 -> 1 -> 4 -> 5
6 -> 3 -> 7 -> 2 -> 1 -> 4 -> 5 -> 6
6 -> 5 -> 4 -> 1 -> 2 -> 7 -> 3 -> 6
7 -> 2 -> 1 -> 4 -> 5 -> 6 -> 3 -> 7
7 -> 3 -> 6 -> 5 -> 4 -> 1 -> 2 -> 7

Please provide a file:
random8.tsp

Optimal Dist: 310.982080
Paths with Optimal Dist: 4

4 -> 5 -> 2 -> 3 -> 7 -> 1 -> 6 -> 8 -> 4
4 -> 8 -> 6 -> 1 -> 7 -> 3 -> 2 -> 5 -> 4
8 -> 4 -> 5 -> 2 -> 3 -> 7 -> 1 -> 6 -> 8
8 -> 6 -> 1 -> 7 -> 3 -> 2 -> 5 -> 4 -> 8

Please provide a file:
random9.tsp

Optimal Dist: 131.028366
Paths with Optimal Dist: 1

6 -> 7 -> 1 -> 8 -> 4 -> 9 -> 2 -> 5 -> 3 -> 6

Please provide a file:
random10.tsp

Optimal Dist: 106.785820
Paths with Optimal Dist: 3

1 -> 2 -> 7 -> 6 -> 8 -> 5 -> 9 -> 10 -> 4 -> 3 -> 1
4 -> 3 -> 1 -> 2 -> 7 -> 6 -> 8 -> 5 -> 9 -> 10 -> 4
9 -> 5 -> 8 -> 6 -> 7 -> 2 -> 1 -> 3 -> 4 -> 10 -> 9

Please provide a file:
random11.tsp

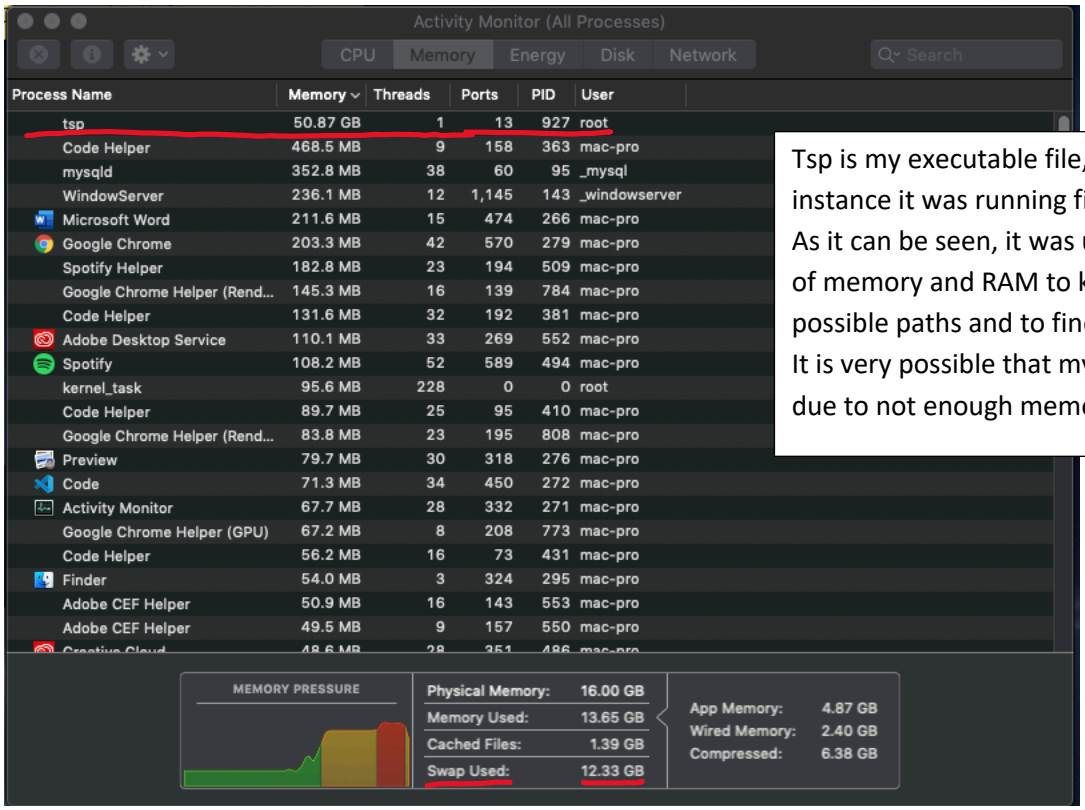
Optimal Dist: 252.684434
Paths with Optimal Dist: 15

1 -> 6 -> 10 -> 11 -> 8 -> 9 -> 7 -> 5 -> 3 -> 4 -> 2 -> 1
2 -> 1 -> 6 -> 10 -> 11 -> 8 -> 9 -> 7 -> 5 -> 3 -> 4 -> 2
2 -> 4 -> 3 -> 5 -> 7 -> 9 -> 8 -> 11 -> 10 -> 6 -> 1 -> 2
3 -> 5 -> 7 -> 9 -> 8 -> 11 -> 10 -> 6 -> 1 -> 2 -> 4 -> 3
4 -> 2 -> 1 -> 6 -> 10 -> 11 -> 8 -> 9 -> 7 -> 5 -> 3 -> 4
4 -> 3 -> 5 -> 7 -> 9 -> 8 -> 11 -> 10 -> 6 -> 1 -> 2 -> 4
5 -> 7 -> 9 -> 8 -> 11 -> 10 -> 6 -> 1 -> 2 -> 4 -> 3 -> 5
6 -> 1 -> 2 -> 4 -> 3 -> 5 -> 7 -> 9 -> 8 -> 11 -> 10 -> 6
6 -> 10 -> 11 -> 8 -> 9 -> 7 -> 5 -> 3 -> 4 -> 2 -> 1 -> 6
7 -> 5 -> 3 -> 4 -> 2 -> 1 -> 6 -> 10 -> 11 -> 8 -> 9 -> 7
7 -> 9 -> 8 -> 11 -> 10 -> 6 -> 1 -> 2 -> 4 -> 3 -> 5 -> 7
8 -> 9 -> 7 -> 5 -> 3 -> 4 -> 2 -> 1 -> 6 -> 10 -> 11 -> 8
9 -> 8 -> 11 -> 10 -> 6 -> 1 -> 2 -> 4 -> 3 -> 5 -> 7 -> 9
10 -> 11 -> 8 -> 9 -> 7 -> 5 -> 3 -> 4 -> 2 -> 1 -> 6 -> 10
11 -> 8 -> 9 -> 7 -> 5 -> 3 -> 4 -> 2 -> 1 -> 6 -> 10 -> 11

Please provide a file:
random12.tsp

Optimal Dist: 66.084844
Paths with Optimal Dist: 5

1 -> 8 -> 2 -> 3 -> 12 -> 4 -> 9 -> 5 -> 10 -> 6 -> 7 -> 11 -> 1
3 -> 12 -> 4 -> 9 -> 5 -> 10 -> 6 -> 7 -> 11 -> 1 -> 8 -> 2 -> 3
5 -> 9 -> 4 -> 12 -> 3 -> 2 -> 8 -> 1 -> 11 -> 7 -> 6 -> 10 -> 5
10 -> 5 -> 9 -> 4 -> 12 -> 3 -> 2 -> 8 -> 1 -> 11 -> 7 -> 6 -> 10
12 -> 4 -> 9 -> 5 -> 10 -> 6 -> 7 -> 11 -> 1 -> 8 -> 2 -> 3 -> 12



Tsp is my executable file, and in this instance it was running file Random12.tsp . As it can be seen, it was using a large chunk of memory and RAM to keep track of all the possible paths and to find the optimal ones. It is very possible that my program can crash due to not enough memory.

4. **Discussion** (Talk about the results you got and answer any specific questions mentioned in the assignment.)

The results that I am getting for the optimal path distance seem to be very valid numbers since they aren't extremely low or high and seem to fall within margins. Also, every time a calculation is ran on a specific file the result stays consistent.

From file Random4.tsp to file Random11.tsp the computations take within less than 10 seconds. When the computation is running on Random12.tsp the program can take up to five minutes to finish executing. Random12.tsp has 12! (12! = 479001600) permutations which must get calculated and parsed through in order to compute an answer. The exponential explosion can be very large for cities larger than 11 in which brute force may not suffice or purposes.

It has also been taken in consideration that the computation for 12 cities can take up 51gb of RAM which comes very close to crashing the program specially in older computers with less resources. These calculations have been generated on a MacBook Pro that has 16gb RAM and is fairly new. In systems with 8gb of RAM or other lack of resources, 12 cities might not be able to be computed. In order to tackle this issue, I came up with an idea in which I no longer dynamically allocated all of the permutations but instead only the optimal ones. By doing this it is possible that calculations can be sped up and the computer can process larger number of cities.

5. **Execution (**Give the necessary steps and resources needed in order to be able to run the program)

To run this program, it is needed to have GCC installed in order to compile the .C file. At least 16gb of RAM is recommended in order to get to 12 cities.

1) gcc src.c -Wall -g -o <choose a name for your execution file>
2) ./<chosen execution file name>

3) Enter path to .tsp file or name of the file with its extension type (.c) if it is in the same directory.

6. References (If you used any sources in addition to lectures please include them here.)

http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95

http://www.tsp.gatech.edu/concorde/index.html

https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/

https://www.cs.fsu.edu/~myers/c++/notes/pointers1.html

http://scc-forge.lancaster.ac.uk/open/char/memory/dynamic

https://www.tutorialspoint.com/gnu_debugger/gdb_commands.htm#

https://www.tutorialspoint.com/c_standard_library/c_function_fscanf.htm

https://www.cs.swarthmore.edu/~newhall/unixhelp/C_arrays.html