

Project 1 Readme MCR

Version 1 9/11/24

A single copy of this template should be filled out and submitted with each project submission, regardless of the number of students on the team. It should have the name `readme_ "teamname"`. Also change the title of this template to "Project x Readme Team xxx"

1	Team Name: MCR										
2	Cassandra Barco, Rogelio Diaz, Margaret Aiyenero (Cbarco, rdiaz3, maiyener)										
3	Overall project attempted, with sub-projects: Hamiltonian Path/Cycle: Traveling Salesman Problem Programs: Brute Force, Backtracking, Best Case										
4	Overall success of the project: We think the project was successful for the most part, despite having trouble interpreting the approaches. After reading the approaches for our problems, we struggled with how to best approach the test cases and were confused about how the traveling salesman problem would be implemented/checked (based on the example input provided). After discussing, we were able to decide how best to approach these problems. We relied on each of our strengths to complete the code, like having one member work on the recursion since they felt most confident in that, compared to the other members. In the end, we successfully tested the approaches with the input and saw there were no errors processing the test cases.										
5	Approximately total time (in hours) to complete: 18										
6	Link to github repository: https://github.com/rdiaz369/Project1-TOC										
7	<div>List of included files (if you have many files of a certain type, such as test files of different sizes, list just the folder): (Add more rows as necessary)</div> <table border="1"><thead><tr><th>File/folder Name</th><th>File Contents and Use</th></tr></thead><tbody><tr><td colspan="2">Code Files</td></tr><tr><td>hamiltonian_MCR.py/src travelingSalesman_MCR.py/src</td><td>This file contains the implementation for the Hamiltonian Path/Cycle: Traveling Salesman Problems. The backtracking, brute force, and best-case sub-problems were chosen for this. Traveling salesman implements the same algorithms except now we are accounting for weights.</td></tr><tr><td colspan="2">Test Files</td></tr><tr><td>check_MCR.cnf/input check2_MCR.cnf/input hamilton_input.cnf/input</td><td>This file contains the input for our Hamiltonian cycle and path. It has unweighted values for interpreting the best case and the best path to take. We also created check2 file for the traveling salesman problem which involves weights. (included given input file with 3 instances)</td></tr></tbody></table>	File/folder Name	File Contents and Use	Code Files		hamiltonian_MCR.py/src travelingSalesman_MCR.py/src	This file contains the implementation for the Hamiltonian Path/Cycle: Traveling Salesman Problems. The backtracking, brute force, and best-case sub-problems were chosen for this. Traveling salesman implements the same algorithms except now we are accounting for weights.	Test Files		check_MCR.cnf/input check2_MCR.cnf/input hamilton_input.cnf/input	This file contains the input for our Hamiltonian cycle and path. It has unweighted values for interpreting the best case and the best path to take. We also created check2 file for the traveling salesman problem which involves weights. (included given input file with 3 instances)
File/folder Name	File Contents and Use										
Code Files											
hamiltonian_MCR.py/src travelingSalesman_MCR.py/src	This file contains the implementation for the Hamiltonian Path/Cycle: Traveling Salesman Problems. The backtracking, brute force, and best-case sub-problems were chosen for this. Traveling salesman implements the same algorithms except now we are accounting for weights.										
Test Files											
check_MCR.cnf/input check2_MCR.cnf/input hamilton_input.cnf/input	This file contains the input for our Hamiltonian cycle and path. It has unweighted values for interpreting the best case and the best path to take. We also created check2 file for the traveling salesman problem which involves weights. (included given input file with 3 instances)										

	Output Files	
	<p>output_best_case_check_MCR_hamilton_cycle_results.csv/results</p> <p>output_brute_force_check_MCR_hamilton_cycle_results.csv/results</p> <p>output_btracking_check_MCR_hamilton_cycle_results.csv/results</p>	<p>These output files are tables that have the same header as the hamilton_cycle example output has (Instance_ID,Num_Vertices,Num_Edges,Hamiltonian_Path,Hamiltonian_Cycle,Largest_Cycle_Size,Algorithm,Time). Thus, they output this information for each approach.</p>
	Plots (as needed)	
	<p>Graphs/results</p> <p>plots_hamiltonian_backtracking_MCR.png/results</p> <p>plots_hamiltonian_best_case_MCR.png/results</p> <p>plots_hamiltonian_brute_force_MCR.png/results</p>	<p>This includes the links to our excel document where we created the plots. We also included the plot pictures as well, though.</p>
8	Programming languages used, and associated libraries: python, itertools, typing libraries (for set, dict, etc.)	
9	<p>Key data structures (for each sub-project):</p> <p>Brute Force - dictionary of sets, tuple (for checking), list/arrays</p> <p>Best Case - same as backtracking (logic was implemented within the backtracking function)</p> <p>Backtracking - dictionary of sets, list/arrays, sets</p>	
10	<p>General operation of code (for each subproject)</p> <p>Brute Force - Generates all possible city paths, their distances and then gives the shortest route</p> <p>Best Case - fastest route but does not guarantee to be optimal (nearest neighbor algorithm)</p> <p>Backtracking -Goes through inputs step by step. If we see an answer won't beat the current best scenario we backtrack (undo) last choice to try another option.</p>	

11	<p>What test cases you used/added, why you used them, and what did they tell you about the correctness of your code.</p> <p>We used the test case given to us because we wanted to make sure our code ran successfully. This told us clearly if our code was correct or not. We also added test case for Traveling Salesman to account for weights. We tested the original test cases and expanded on them to further test our code by adding more cases.</p>
12	<p>How you managed the code development:</p> <p>We cloned the GitHub repo given to us and each member committed and pushed any changes to the main branch rather than using separate branches, since we thought it would be easier than having to check each of our changes before merging. We made sure to use commit messages that were descriptive. We also split the problems and worked on each case before moving onto the next one. We worked together to find all of the solutions.</p>
13	<p>Detailed discussion of results:</p> <p>When we added more inputs for brute force it became inefficient. When the input hit 9 it significantly declined. This indicates the need for better methods for real world methods. For the case of backtracking it also had an optimal point at 8. Based on the graphs we can see the algorithms at play for each case which further justifies our conclusion that brute is inefficient. Having a more refined case like backtracking with bounds improves the efficiency of brute force. As can be seen when we look at the maximum time of each different plot.</p> <p>Brute force < backtracking < best case (in order of least to most efficient)</p>
14	<p>How team was organized</p> <p>Rogelio: Worked on setting up the repository, outlined functions for brute_force, and BackTracking. Implemented Brute_Force Function. Added Traveling Salesman code—integrated new input file to take on weights. Used Hamiltonian as guidelines for traveling salesman, just added a way to read in weights</p> <p>Margaret: Worked with Cassandra on the backtracking function and implemented the recursion logic and checked the size. Added the graph information and fixed errors when checking our code.</p> <p>Cassandra: Worked with Margaret on the backtracking function. Added the checks for the Hamiltonian cycle and path and updated the return on the simple case (which we weren't implementing). Created some of the plots on excel and fixed naming conventions.</p> <p>The group discussed when issues arose and worked together to solve them (though the person in charge of that part was the only one who committed their changes to ensure consistency)</p>
15	<p>What you might do differently if you did the project again</p> <p>We would probably try to use shortcuts in our code to make the algorithms run more quickly. This might make our graphs look “better” in terms of how the time complexities compare to each approach.</p>
16	<p>Any additional material:</p> <p>To test our input data, we edited constants.py script in the 'helpers' folder and replaced</p>

	<p>the input file with our input data. We also edited <code>hamilton_cycle_helper.py</code> by changing the value of the <code>result_file_name</code> variable to "hamilton_cycle_results" to be more accurate. We tried to follow the naming conventions in the project 1 submission doc, which is why constants.py was edited and the hamilton cycle helper script as well. To test out the traveling salesman you must run <code>uv run src/travelingSalesman_MCR.py</code>.</p>
--	---