Infusing Computational Thinking into a Computer Science Gateway Course*

Younes Benkarroum, Mohammad Q. Azhar
Computer Information Systems
Borough of Manhattan Community College
New York, NY 10007
{ybenkarroum,mazhar}@bmcc.cuny.edu

Abstract

Computational thinking (CT) stands as a universal problem-solving approach applicable across diverse disciplines, transcending the domain of computer science. It embodies the mental process of structuring a problem to enable a computational solution feasible for both humans and machines. This methodology involves dissecting problems into smaller parts that are easier to understand and solve. This study delineates a meticulously designed series of CT activities within an introductory computer science course and explores their profound impact on student engagement and problem-solving proficiency. Our findings underscore the pivotal role of hands-on CT practice in augmenting students' ability to decompose problems, recognize patterns, and abstract complexities, and employ algorithms effectively. Notably, this infusion of CT not only cultivates theoretical understanding but also bridges the gap between conceptual knowledge and real-world application through the use of computational tools like Python programming. As CT continues to emerge as a cornerstone skill in diverse domains, this research presents compelling evidence advocating for its integration into introductory courses, laying a robust foundation for students to navigate the evolving technological landscape with enhanced problem-solving capabilities.

1 Introduction

Promoting computational thinking (CT) has emerged as a pivotal educational objective spanning both STEM and non-STEM domains. CT stands as an inclusive problem-solving approach, extending beyond computer science to become as fundamental as core competencies like literacy and numeracy, as articulated by Professor Wing [10]. Understanding the scope and constraints of CT, as underscored by Lu and Fletcher [4], holds significant relevance, even for individuals outside the STEM sphere, contributing substantially to professional and everyday contexts.

Computational thinking encapsulates the cognitive process of framing problems to accommodate solutions executable by humans, machines, or their amalgamation [11]. This methodology involves breaking problems down into separate parts (decomposition), looking for similarities or common differences (pattern recognition), filtering out information that is not necessary to solve the problems (abstraction), and developing step-by-step instructions for solving problems (algorithms) [9].

Between Fall 2021 and Fall 2023, as part of our NSF-funded project "Bridging the Gap: Designing a Technology Learning Community Integrating Computational Thinking to Improve STEM Engagement across Disciplines", we curated a series of five computational activities aimed at honing students' skills in practicing and perfecting their computational thinking abilities. The students who participated in the activities were from the introductory course CSC 101 - Principles in Information Technology. This article details the approach and outcomes observed among a total of 142 CSC 101 students in addressing these computational thinking activities.

2 Related Work

Previous studies have underscored the synergy between creative thinking and computational thinking (CT), advocating for their combined integration into educational frameworks. Miller et al. [5] demonstrated the efficacy of introducing creative thinking exercises within introductory computer science courses, a concept echoed in our approach. The introduction of our CT activities not only mirrors this integration but also provides a practical platform where students engage in creative problem-solving scenarios that revolve around computational challenges.

Moreover, the study by the same research team [7] that introduced Computational Creativity Exercises (CCE) into an introductory computer science course tailored for engineering students aligns closely with our methodologies. Just as their outcomes highlighted an enhanced grasp of fundamental CT prin-

ciples, our meticulously designed CT exercises aimed to not only introduce core CT concepts but also engage students in practical problem-solving tasks, observing similar positive impacts on students' understanding and application of CT methodologies.

Furthermore, the work by Shell et al. [6], emphasizing the correlation between self-regulation, creative competence, and CT skills, aligns with our approach of providing a scaffolded learning environment. Our activities aimed to scaffold the development of CT skills by progressively guiding students through the elements of decomposition, pattern recognition, abstraction, and algorithm design, promoting a structured approach to problem-solving similar to that advocated by prior research.

Lu and Fletcher [4] proposed that the mental models and patterns fostered by computational thinking create an environment conducive to teaching higher-level computational processes and abstraction. Csizmadia et al. [3] presented a guide for integrating computational thinking into curricula, emphasizing its fundamental role in education. Swaid [8] promoted the universal relevance of computational thinking as an underlying mindset in STEM projects and education discussions. Castro et al. [2] analyzed the impact of a first-year engineering course on the acquisition of computational thinking across various student profiles.

In essence, these studies collectively underscore the significance of merging creative thinking with CT, scaffolding learning experiences, and reinforcing theoretical concepts through practical engagement — principles that resonate strongly with the integrative CT activities outlined in this study.

3 CT Exercise Design

Our activities introduced students to the problem of finding the shortest path between multiple cities on a road map. The activities we designed aim to cultivate computational thinking, which is foundational for problem-solving in computer science. By breaking down complex problems into manageable parts and devising algorithms, students develop crucial problem-solving strategies applicable across various domains within computer science.

When traveling from one place to another, there are several factors to consider; unless the intent is to visit some points and landmarks in a specific order, people are often interested in the most efficient way to get somewhere. In our activities, students used CT to experiment with different ways of creating paths between multiple points to effectively travel through cities in a region. This problem is also known as the Traveling Salesman Problem (TSP) [1] where the salesman must take the shortest path that passes through each city exactly once and returns back to the start. Thus, the problem statement of our exer-

cise is: Given a list of cities in South Africa, the goal is to reduce the cost for a vendor whose duty is to distribute the COVID-19 vaccines in those cities. The problem is therefore to find the shortest possible route that starts from Cape Town and visits each city exactly once.

As stated earlier in the introduction, CT is a problem-solving process that includes four elements: decomposition, pattern recognition, abstraction, and algorithms. In the following, we focus on one CT element at a time.

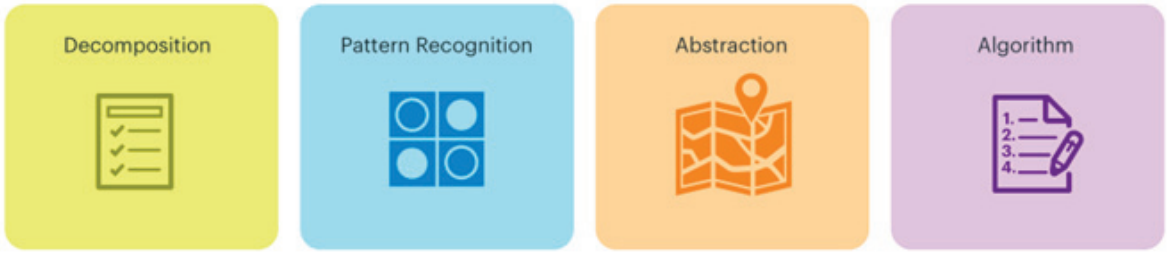


Figure 1: Elements of Computational Thinking Process

3.1 Decomposition

Breaking problems down into smaller parts can make complex challenges more manageable. This allows other elements of CT to be applied more effectively to complex challenges. Applying the decomposition element to our problem leads to the following sub-tasks:

1. Given the list of cities, determine their geographic coordinates (i.e., latitude and longitude) to find their exact positions on the map.
2. Find the distances between each pair of cities in the list.
3. Find the shortest path that passes through each city exactly once.

The first two sub-tasks are quite straightforward and easy to accomplish using the available online maps and their geographic tools; but the last sub-task is the main section of our activity. In the following, we will only describe how the remaining CT elements can be applied to this section.

3.2 Pattern Recognition

Recognizing if a pattern exists and determining its sequence can simplify the solution. For the task at hand, we noticed that the problem can be solved recursively as follows.

Let the letter O (Origin) be the label of the departure city (Cape Town). If we only need to visit two cities (A and B), then the strategy we use to

determine the shortest path will choose either the path O-A-B or the path O-B-A. Thus, the number of possible paths to choose from is 2.

If the number of cities to visit is three (A, B, and C), then the first city to be visited can be either O-A, O-B or O-C, then the remaining two cities must be visited. Note that from the previous case, the strategy knows how to determine the shortest path when visiting two cities, so the number of possible paths in this case is $3 \times 2 = 6$.

In general, if the number of cities to visit is n, then the number of possible paths to choose from is $n \times P(n\text{-}1)$, where $P(n\text{-}1)$ is the number of possible paths when visiting $n\text{-}1$ cities.

3.3 Abstraction

Stepping back from the specific details of a given problem and focusing on the big picture allows us to create a more generic solution. Applying this CT element in our case requires us to analyze the problem to leave out unnecessary information such as the mode of transportation chosen, the purpose of the trip, the time needed to visit all cities, whether the vendor is traveling with coworkers, etc. Once done, we can start brainstorming a solution to the problem. The important details to focus on are the list of cities, their exact locations, and the distances between them.

3.4 Algorithms

An algorithm is a step-by-step strategy for solving a problem. It can be written in plain language, with flowcharts, or pseudocode. In our activity, students from CSC 101 and BUS 104 tried four different algorithms and compared their outputs. All algorithms have been applied to the list of the given cities from South Africa; the starting point was always Cape Town, and then the algorithms produced routes that visit all other cities. The algorithms we considered in this activity are listed below.

1. **Longitude sorting:** The task is to sort the cities by their longitudes, then to visit them all in ascending order of this geographic coordinate. By doing so, the path traveled will visit all cities from left to right (from the farthest city in the west to the farthest city in the east). The total route distance is the sum of the distances between each pair of cities on the route.
2. **Latitude sorting:** The task now is to sort the cities by their latitudes, and then visit them all in ascending order of that coordinate. In doing so, the path traveled will visit all cities from the farthest city in the south to the farthest city in the north.

3. **Nearest neighbor:** The nearest neighbor algorithm starts at Cape Town and connects to the nearest unvisited city. It repeats until every city has been visited.
4. **Greedy algorithm:** The greedy algorithm is an iterative algorithm that builds a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit. In our activity, the edges are the roads between the pairs of cities. All possible edges are sorted by distance in ascending order; then at each iteration, we add the shortest edge which will not make a city with more than 2 edges, nor create a cycle.

4 Results and Discussion

It's important to highlight that due to the staggered offering of activities across various semesters and days, not all students engaged in every activity. While some students participated in all activities, others took part in only specific ones. Overall, among the various activities offered, a total of 142 unique students participated in at least one. Figure 2 illustrates the distribution per semester of the students who participated in the CT activities, while Table 1 and Figure 3 display the counts and rates of students who successfully completed each of the four algorithms.

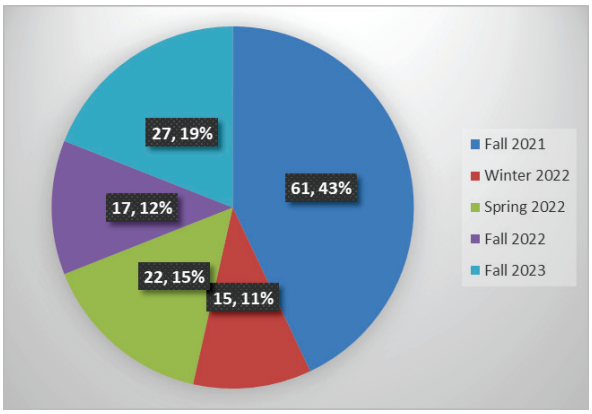


Figure 2: Distribution per Semester of the Participants

From the description of each of the algorithms above, we can observe that the initial two activities present relatively straightforward tasks. The initial low success rate for the Longitude Sorting (75.3%) might indicate a lack of clarity in instructions or unfamiliarity with the activity format. This could be attributed to it being the first task. The lesson plan regarding the Longitude Sorting activity is appended at the end of this article. The high success rate for

Table 1: Number of Participants and Success Rate by Activity

Algorithm	Participants	Successful Completion	Success Rate
Longitude sorting	77	58	75.3%
Latitude sorting	69	61	88.4%
Nearest neighbor	85	66	77.6%
Greedy algorithm	41	23	56.1%

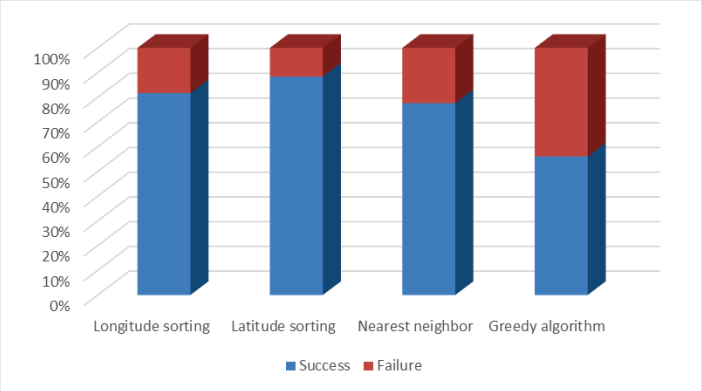


Figure 3: Success Rate by Activity

the Latitude Sorting (88.4%) indicates that once students understood the format and requirements (thanks to the prior task), they performed significantly better. Familiarity likely contributed to this high success rate. The Nearest Neighbor task had a relatively moderate success rate (77.6%) which seems reasonable considering the complexity of the algorithm. The Greedy Algorithm task had the lowest success rate (56.1%) as anticipated due to its difficulty. Being a challenging algorithm, it's expected that a smaller percentage of students would successfully complete this task, especially in an introductory course.

The varying success rates signify the students' level of engagement and understanding of CT concepts. Higher success rates, such as those observed in tasks following simpler algorithms after initial exposure, indicate a growing familiarity and grasp of CT principles. These observed rates offer pedagogical insights, guiding educators in designing future CT activities. Understanding which CT elements pose challenges or foster better comprehension can aid in refining teaching methodologies and curriculum design to optimize student learning experiences.

It's worth noting that the participating students did not utilize programming skills to determine the shortest routes for the four explored algorithms. Instead, they manually executed each algorithm's steps and solely employed the Google Sheet *SUM* function to compute the total distance for each path. As

depicted in Table 2, there has been a progressive enhancement in the distance of the shortest path generated by these algorithms.

Table 2: Distance of Shortest Path by Algorithm

Algorithm	**Distance**
Longitude sorting	4,941 km
Latitude sorting	4,004 km
Nearest neighbor	3,846 km
Greedy algorithm	3,586 km

Before engaging in the computational thinking activity, students were prompted with questions such as, *'Imagine planning a road trip with multiple stops; how would you determine the route?'* Interestingly, some students independently devised steps similar to those employed in the nearest neighbor algorithm, unaware of its formal designation. Notably, many perceived this approach as highly efficient for solving the traveling salesman problem. Subsequently, upon exposure to the greedy algorithm, students exhibited fascination, particularly upon realizing the considerable distance saved by this algorithm. Their reaction reflected an enthusiastic appreciation for the algorithm's optimization capabilities.

5 Brute Force Algorithm

To evaluate whether the CT activity generated the optimal route, students were tasked with employing the brute force method. This algorithm represents the initial and most straightforward strategy when confronted with a problem. In technical terms, it involves exhaustively considering all available possibilities to resolve a problem. In our case, the brute force algorithm would generate every conceivable path among the cities, subsequently computing the distance for each path and selecting the shortest one. This approach seems to be simple and achievable. But practically this is not the case unless you get help from a computing device. The number of all possible paths in our situation is exceedingly large; we have already demonstrated that the number of possible paths to choose from is $P(n) = n \times P(n\text{-}1)$. This sequence can be simplified as follows:

$$P(n) = n(n-1)(n-2)\ldots 1 = n!$$

With eight cities to visit in our activity, the total count of potential paths amounts to $8!$ (i.e., 40,320). Manually calculating all these paths is impractical for students. Hence, they employed a computer program to generate and evaluate all paths, subsequently selecting the one with the shortest distance. To

address this challenge, students were provided with a concise Python program utilizing a *for-loop* to systematically generate and display the various paths for visiting three cities (labeled as B, C, and D) from the initial city A and computing their respective distances. The Python program listing is provided below, and Figure 5 showcases the program's output.

```
import itertools

Cities = "ABCD"

Distances = [[0, 40, 20, 90],
             [40, 0, 30, 70],
             [20, 30, 0, 55],
             [90, 70, 55, 0]]

AllPaths = list(itertools.permutations(sorted(Cities[1:])))

for p in AllPaths:
  path = list(p)
  path.insert(0,"A")
  path = tuple(path)
  print(path, end = ' ')

  sum = 0
  for x in range(len(path) - 1):
    sum += Distances[Cities.find(path[x])][Cities.find(path[x + 1])]

  print("distance =", sum)

print("\nNumber of paths =", len(AllPaths))
```

Figure 4: Brute Force Python Program

```
('A', 'B', 'C', 'D') distance = 125
('A', 'B', 'D', 'C') distance = 165
('A', 'C', 'B', 'D') distance = 120
('A', 'C', 'D', 'B') distance = 145
('A', 'D', 'B', 'C') distance = 190
('A', 'D', 'C', 'B') distance = 175

Number of paths = 6
```

Figure 5: Brute Force Python Program

Building upon the above Python program, the students' initial task involved modifying the code to accommodate the traversal of eight cities in South Africa, considering the actual distances between each pair of cities.

Due to the large number of paths presented on the program's output (40,320), students encountered difficulty identifying the path with the shortest distance. To overcome this challenge, students were instructed to enhance the Python program by integrating a decision-making statement (if-statement) and the necessary variables to capture the smallest distance and its corresponding path, and subsequently displaying this information on the screen. Among the 91 participating students, 76 successfully completed this task, resulting in a 83.5% success rate. Remarkably, the shortest path distance recorded aligned precisely with the distance computed using the greedy algorithm. This alignment intensified the students' fascination with the efficiency and accuracy of the greedy algorithm, solidifying their appreciation for its problem-solving prowess.

It's important to note that while the optimal shortest path distance aligned with the output of the greedy algorithm in this specific instance, this outcome may not necessarily hold true for all scenarios. The apparent optimality observed could be coincidental, and further investigation is warranted to determine the general applicability of the greedy algorithm across various problem instances.

6 Conclusion

In summary, the infusion of computational thinking (CT) into the Computer Science Gateway Course was instrumental in enhancing problem-solving skills among students. Through a series of meticulously designed activities, participants were not only introduced to the core principles of CT but also engaged in practical problem-solving scenarios. The findings from this study highlight the progressive enhancement in students' comprehension and application of CT methodologies across diverse algorithms, with varying complexities.

The analysis of students' performance revealed insightful patterns, showcasing the impact of familiarity and task complexity on success rates. Notably, the success rates varied across algorithms, shedding light on the challenges students encountered and conquered. Despite the complexities, students demonstrated a commendable grasp of CT elements, showcasing their ability to decompose problems, recognize patterns, abstract details, and employ algorithms effectively.

Moreover, the integration of CT wasn't solely confined to theoretical understanding; rather, it required practical implementation, necessitating creativity and critical thinking. The utilization of computational tools like Python programming served as a bridge between theoretical concepts and real-world problem-solving, enhancing students' ability to navigate complex scenarios.

In conclusion, this study has significant implications for educational strategies, emphasizing the importance of hands-on engagement and practical ap-

plications in computational thinking education. As CT continues to emerge as a foundational skill in various domains, its integration into introductory courses lays a robust foundation for students, equipping them with essential problem-solving abilities crucial in today's evolving technological landscape.

As a part of future work, the authors aim to conduct a comparative study between students enrolled in CS introductory courses where computational thinking (CT) was infused and those where it was not. This comparative analysis will focus on evaluating the performance of students when confronted with novel problems demanding solution methodologies. It is hypothesized that students exposed to CT methodologies will showcase enhanced problem-solving capabilities in tackling new challenges. This comparative analysis holds the potential to offer valuable insights into the tangible impact of integrating CT into introductory CS courses. The aim is to discern the influence of CT infusion on students' adaptability and efficacy in addressing novel problems, ultimately gauging the efficacy of CT as a foundational educational approach.

Acknowledgements

This work was supported by the U.S. National Science Foundation under Award #2122690. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Science Foundation.

References

[1] David L. Applegate et al. *The Traveling Salesman Problem: A Computational Study*. Princeton, New Jersey: Princeton University Press, 2006.

[2] Laura Melissa Cruz Castro et al. "Analyzing Students' Computational Thinking Practices in a First-Year Engineering Course". In: *IEEE Access* 9 (2021), pp. 33041–33050.

[3] Andrew Csizmadia et al. "Computational Thinking - a Guide for Teachers". In: *Computing at School* (2015), p. 18.

[4] James J. Lu and George H.L. Fletcher. "Thinking About Computational Thinking". In: *Proceedings of the 40th ACM technical symposium on Computer science education* (2009), pp. 260–264.

[5] L. Dee Miller et al. "Improving Learning of Computational Thinking Using Creative Thinking Exercises in CS-1 Computer Science Courses". In: *43rd ASEE/IEEE Frontiers in Education Conference Proceeding* (2013), pp. 1426–1432.

[6] Duane F. Shell et al. “Associations of Students’ Creativity, Motivation, and Self-Regulation with Learning and Achievement in College Computer Science Courses”. In: *IEEE Frontiers in Education Conference* (2013).

[7] Duane F. Shell et al. “Improving Learning of Computational Thinking Using Computational Creativity Exercises in a College CS1 Computer Science Course for Engineers”. In: *IEEE Frontiers in Education Conference* (2014).

[8] Samar I. Swaid. “Bringing Computational Thinking to STEM Education”. In: *Procedia Manufacturing* 3 (2015), pp. 3657–3662.

[9] Kristen Thorson. “Early Learning Strategies for Developing Computational Thinking Skills”. In: *Getting Smart* (2018).

[10] Jeannette M. Wing. “Computational Thinking”. In: *Communications of the ACM* 49 (2006), pp. 33–35.

[11] Jeannette M. Wing. “Computational Thinking: What and Why?” In: *The Link Magazine* (2010), pp. 20–23.

Appendix – Longitude Sorting Lesson Plan

Finding the Shortest Path Challenge – Activity 1

There are several factors to consider when traveling from one place to another. Unless the intent is to visit some points and landmarks in a specific order, people are often interested in the most efficient way to get somewhere. In this activity, students will use computational thinking to experiment with different ways of creating paths between multiple points to effectively travel through cities in a region.

Factors involved when choosing a route:

Question: If you are traveling from one place to another, how do you decide what route to take? What are the factors that influence your decision?

Question: Imagine you were asked to create a road trip with multiple stops, how would you decide what route to take?

Traveling Salesman Challenge:

Given a list of cities, their geographic coordinates, and the distances between each pair of them, the goal is to reduce the cost for a salesperson whose duty is to distribute COVID-19 vaccines in those cities. The task is therefore to find the shortest possible route that visits each city exactly once. The list we consider in this activity includes the following cities in South Africa:

- A - Cape Town

- B - Bhisho
- C - Pietermaritzburg
- D - Bloemfontein
- E - Kimberley
- F - Johannesburg
- G - Mahikeng
- H - Nelspruit
- I - Polokwane

The list of cities above along with their geographic coordinates are also included in the Shortest Path Challenge workbook that you will use in this computational thinking activity. The exact locations of the cities are shown on the map below.

The distances in kilometers (km) between each pair of cities are shown in the following matrix:

CITY	A	B	C	D	E	F	G	H	I
A	0	991	1,558	1,004	941	1,398	1,318	1,743	1,719
B	991	0	590	557	659	890	966	1,198	1,241
C	1,558	590	0	556	714	490	746	616	796
D	1,004	557	556	0	165	397	445	743	718
E	941	659	714	165	0	478	365	814	789
F	1,398	890	490	397	478	0	291	327	331
G	1,318	966	746	445	365	291	0	632	545
H	1,743	1,198	616	743	814	327	632	0	301
I	1,719	1,241	796	718	789	331	545	301	0

Note: The above matrix is symmetric since the distance to travel from $\boldsymbol{X}$ to $\boldsymbol{Y}$ is the same as the distance to travel from $\boldsymbol{Y}$ to $\boldsymbol{X}$.

Developing a strategy for traversing all points: In this activity, you will compare four different algorithms and choose the best one. Algorithms will be applied to all the above cities (from South Africa); the starting point is always city $\boldsymbol{A}$, then the algorithms will produce routes that visit all the other cities. The algorithms we are considering are:

- Longitude sorting
- Latitude sorting
- Nearest neighbor
- Greedy algorithm

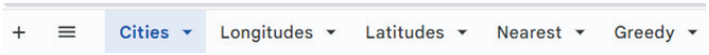
Longitude sorting algorithm:

The task is to sort the cities by their longitudes, and then to visit them all in ascending order of this geographic coordinate. By doing so, the path traveled will visit all cities from left to right (from the farthest city in the west to the farthest city in the east).

Create your local copy of the Shortest Path Challenge workbook. To do this, click on the **File** tab and then select **"Make a copy"** from the drop-down menu.

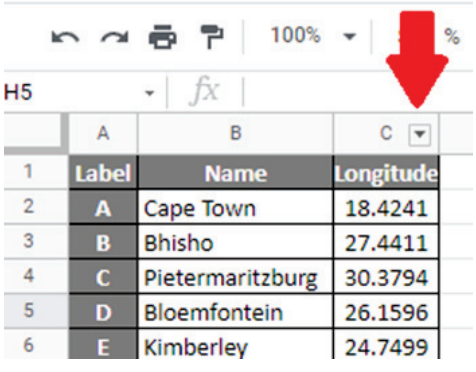
Give your local copy a name, then click **OK**.

The workbook has five worksheets.



The first worksheet (**Cities**) includes the list of cities along with their geographic coordinates and the distance matrix.

Click on the second worksheet (**Longitudes**) that we will be using in this task. This page has the list of cities and their longitudes. Use the sort function to sort cities by longitude in ascending order. To do this, click on the arrow in column **C**, then select **"Sort sheet A –> Z"**.



Once sorted, fill in the second column (**City**) of the Order table in the same worksheet using the resulting order.

Use the distance matrix to fill in the third column (**Distance from Previous City**) of the **Order** table.

Once all the distances are entered, the spreadsheet will automatically calculate and display the total distance at the bottom of the **Order** table. The total distance is the sum of the distances between each pair of cities on the route.

Answer the following questions using this form.

Question 1: What is the order of each city in the route produced by the longitude sorting algorithm?

Question 2: What is the total distance of the route?

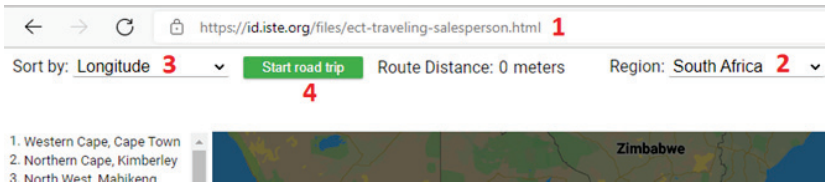
Question 3: Does it look like an efficient route?

Question 4: What is the difficulty of applying this algorithm? Use 1 for the easiest and 5 for the most difficult.

Traveling salesperson simulator:

Now you can use an online mapping tool to check the results of the longitude sorting algorithm. The steps are listed below.

1. Open the Traveling salesperson simulation. Click **OK** in the message window that appears.
2. Use the drop-down menu in the upper-right to set the region to **South Africa**. Click **OK** again.
3. Use the "**Sort by**" drop-down menu in the upper-left to select to sort the cities by **Longitude**.
4. Press the green "**Start road trip**" button to begin the trip.



Once the road trip is complete, review the route on the map and compare the accumulated distance traveled with the total distance you found (your answer to question 2). Note that the distance returned by the online mapping tool uses the meter (m) as the unit of measure. The distance you calculated in the worksheet uses kilometers (km).

Well done! You've successfully completed the first computational thinking activity and are ready to move on to the second activity (**Latitude Sorting**).