Exploring Encoding and Prompting Methods for Graph Reasoning in Large Language Models

Meher Banik
Caltech
mbanik@caltech.edu

Rupali Batta
Caltech
rupali@caltech.edu

Madeline Eagan
Caltech
msegan@caltech.edu

1 Introduction

1.1 Motivation

Graph networks are a fundamental representation of complex systems in many different scenarios such as transportation, communication, computational chemistry, biology, and social networks. Navigating and analyzing these networks are crucial for solving problems like finding the shortest path in a transportation network or optimizing routes in a communication network. Recently, large language models (LLMs) like GPT-4, Claude, and Gemini have shown remarkable abilities in natural language understanding and generation. However, their application in graph reasoning tasks is relatively under explored. LLMs face challenges in graph reasoning due to the need to track states of multiple nodes simultaneously. Despite these challenges, enhancing LLMs for graph-related tasks holds significant promise due to the rapid advancements in LLM capabilities. Enhancing the ability of LLMs to understand and process graphs has potential to significantly improve their overall performance, as many tasks that people ask of LLMs today reduce to graph algorithms or have graph algorithms as essential components of their solution. This research aims to investigate and improve the performance of LLMs in navigating and reasoning through directed and weighted graphs by integrating classical graph processing techniques.

1.2 Key Technical Challenges

There are currently many limitations of using LLMs for graph reasoning. Notably, complex reasoning problems, especially where visualization plays a key role in the reasoning, is not a well developed skill of LLMs and little has been explored in this area. When testing GPT-3.5 on sample unconnected graphs, we noticed that the LLM was unable to determine that the graph was not connected and provided incorrect solutions of paths between subgraphs that did not exist. The LLM also attempted to draw a graph with ASCII art; however, this graph was incorrect and included non-existent edges. Therefore, we identified the need for a different way for LLMs to represent graphs, apart from something visual, in order to process them accurately.

1.3 Summary

This paper aims to compare graph reasoning capabilities on two metrics, encoding type and prompting heuristics. It is hypothesized that LLMs will be more capable at processing text-based graphs that are as close to text is has been trained on before, so text than sounds natural and appears very often in training datasets. Therefore, four different sets of graphs encodings were generated: texting, commute, citation, and adjacency. This paper also delves into various modern prompting techniques, namely Few-Shot and varieties of Chain of Thought. These methods were compared against a simple Zero-Shot benchmark, totalling to five different prompting methods: Zero-Shot, Few-Shot, Random Chain of Thought, Matched Chain of Thought, and Adjacency Chain of Thought. To compare the success of these four encodings and five promptings with one another, a bank of ten questions

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

pertaining to directed and weighted graphs was carefully selected. An answer key to these ten questions for each of the fourty generated graphs was created algorithmically.

After comparing the LLM produced answers to the true answers, we found that Random Chain of Thought outperformed the Zero-Shot benchmark by 103%, the best result of our prompting methods. The Adjacency Chain of Thought method, which we proposed to resemble a more computational and quantitative type of prompting, performed unsurprisingly worst. The four encoding types were all marginally similar, with some variation based on subject topic. These results will be fully explored later in this paper.

2 Relevant Links

Here is the link to the Colab notebook that includes all relevant code for this paper: https://tinyurl.com/159-google-colab

Here is the link to the Google Sheets spreadsheet link that includes the custom graph database generated for this project, along with other relevant information mentioned throughout this report: https://tinyurl.com/159-spreadsheet

3 Related Work

In Google Research's 2024 paper, Talk Like a Graph: Encoding Graphs for Large Language Models (Fatemi, 2024), three factors influencing LLMs performance on graph reasoning tasks were explored: (1) the graph encoding method, (2) the nature of the graph task itself, and (3) interestingly, the very structure of the graph considered. They explored different types of encodings and their effect on LLM performance. We took inspiration from this and extended their research to examine weighted and directed graphs and to explore the use of adjacency matrices as an intermediate step in reasoning. This paper's key limitation is that they are only exploring unweighted and undirected graphs, and so we develop this.

In the 2023 paper, "Graph-ToolFormer: To Empower LLMs with Graph Reasoning Ability via Prompt Augmented by ChatGPT," (Zhang, 2023) the methodology centers on improving LLM's graph reasoning capabilities through external API tools. This method involves crafting specific prompt templates and instructional content, which are utilized to augment a dataset with API calls tailored for various graph reasoning tasks. These tasks include "(1) basic graph property reasoning, (2) bibliographic paper topic reasoning, (3) bio-chemical molecular graph function reasoning, (4) recommender system sequential recommendation reasoning, (5) online social network community reasoning, and (6) knowledge graph entity and relation reasoning." The group aims to be able to tune LLMs to take such a graph and classify what subsection falls under what and what leads to what, using specific prompting. These API-enhanced prompts enable the fine-tuning of pre-trained LLMs,incorporating complex graph reasoning tools directly into the model's output generation process. We plan to use a similarly diverse range of graph examples and prompt templates in our work.

The paper titled "Large Language Models on Graphs: A Comprehensive Survey" (Jin, 2024) provides a thorough review of integrating LLMs with graph data, highlighting the potential and methodologies for enhancing graph-based applications using LLMs. It categorizes the applications into three scenarios: pure graphs, text-attributed graphs, and text-paired graphs, and explores three main techniques for applying LLMs to graphs: as predictors, encoders, and aligners. The survey delves into various real-world applications, discusses open-source codes, benchmarks, and future research directions. This work is significant in mapping the current landscape and outlining avenues for future exploration in combining LLM capabilities with complex graph structures. This was useful in scoping out the limitations of our research and deciding to focus exclusively on text based graphs and try testing LLMs both for encoding and prediction.

In the next paper, "Can LLMs perform structured graph reasoning tasks?" (Agrawal, 2023) the authors introduce a more thorough prompting technique, PathCompare. PathCompare enhances LLMs' performance in graph reasoning tasks by guiding models to systematically list all possible paths between two nodes and then compare their costs to identify the optimal path. This method suprisingly outperforms traditional prompting techniques, like Chain-of-Thought, across various

structured graph reasoning challenges, such as multi-hop reasoning and node traversal in weighted and directed graphs. This was just interesting to note how more thorough prompting and extra CoT can make a difference.

4 Research question and approach

4.1 Process for encoding method selection

Prompting an LLM to understand and analyze graphs using a text-based approach rather than a purely numeric or computational representation offers several advantages, primarily because LLMs are trained extensively on natural language rather than on numeric data. They excel in processing and interpreting information presented in linguistic form, leveraging contextual clues and common-sense knowledge embedded in textual descriptions to make inferences. This capability makes the outputs more interpretable and accessible for human users, essential in applications where explaining the model's reasoning is critical. Additionally, text allows for a richer, more nuanced semantic representation of relationships within a graph and provides flexibility in representing complex data more intuitively than numeric formats. Using text reduces the need for numeric preprocessing, enabling LLMs to work directly with familiar data types and avoiding the inaccuracies that can arise from data transformation.

When choosing encoding techniques for this paper, four types of methods were compared, three of which are based on general human-like speech: texting, commute, and citation. For texting, representing graph data as "Alice sent Bob 0 texts" or "Bob sent Claire 39 texts" simply leverages the LLM's proficiency in understanding and processing natural language constructs rather than plain numbers. Similarly, encoding citation networks as "Dr. Lee cited Dr. Yue 59 times" uses familiar linguistic patterns that LLMs are adept at interpreting. This method of presenting information mimics the type of data LLMs encounter during training, such as sentences in books, articles, or conversations, where numerical data is contextualized within text. Then, describing commute distances in a textual format like "The distance from Pennsylvania to D.C. is 216 miles" aligns well with how geographic information is typically discussed in human language. These textual representations are inherently more compatible with the neural architectures of LLMs, which are optimized for text processing, making them more effective at deriving insights from such data compared to facing raw numeric values alone. To compare these encoding methods with a more raw computational format, the fourth encoding of adjacency is introduced. This is far more quantitative and represents the graph as a list of tuples of edges and weights. It would be expected that the three human-like encoding methods would outperform adjacency, due to LLMs being trained on more textual data.

4.2 Process for prompting method selection

The choice of prompting methods such as Zero-Shot, Few-Shot, and three Chain of Thought techniques is designed to explore different capabilities of LLMs in understanding and reasoning about graph structures. Specifically, comparing Few-Shot and Chain-of-Thought prompting provides deep insights into whether LLMs can better understand complex tasks through examples or if they require detailed explanatory reasoning. Few-Shot learning presents the model with a small number of examples (in this case, three) along with their respective tasks and answers, which helps the model learn the task dynamics and generalize to new, unseen graphs. This method tests the model's ability to abstract patterns from limited data. In contrast, the Chain-of-Thought approach not only provides answers but also includes a step-by-step reasoning process. This might help the LLMs in developing a deeper, more structural understanding of how to approach and solve graph-based tasks, potentially leading to better performance when faced with new or more complex graphs.

Further, the exploration between Random-CoT and Matched-CoT prompts intriguing questions about the adaptability and learning transfer of LLMs. Random-CoT involves learning from a CoT explanation paired with a graph of a different encoding type than the test graph, examining the model's ability to transfer learned reasoning across different graph representations. Matched-CoT, however, keeps the encoding type consistent between training and testing scenarios, focusing on the model's ability to deepen its understanding when the training context closely matches the test context. This comparison is interesting to determine if LLMs benefit more from consistency in data representation or if diverse exposures bolster their adaptability. Lastly, the Adjacency-CoT method, which incorporates the adjacency matrix of the graph in the task description, tests whether

integrating explicit structural data (in this case, adjacency matrices) alongside textual descriptions enhances or impedes the LLM's ability to decipher and reason about graph tasks. This can reveal if traditional, numeric graph representations can be effectively integrated with LLMs' primarily text-based processing capabilities, potentially opening up new avenues for combining numerical graph data with natural language processing in LLM applications.

4.3 Prompting Heuristics

We briefly introduce the prompting methods utilized in this project. Full examples of their usages can be found in the appendix.

1. **Zero-Shot** This method involves giving the model a graph and list of graph tasks, requesting it to produce the desired output, all without any prior training specific to the task.
2. **Few-Shot** This method supplies the model with 3 graphs, their lists of graph tasks, and their respective answers to such tasks. The model then learns from these examples to handle a new graph and its respective list of tasks. In this project specifically, each example graph was a distinct encoding type, and the "test" graph was also of distinct encoding.
3. **Random-CoT** This method supplies the model with an graph, its list of graph task questions, and the respective answers, along with a Chain-of-Thought reasoning that explains how each answer was obtained. The model then learns from this example to handle a new graph of a different encoding type and its own list of tasks.
4. **Matched-CoT** This method supplies the model with an graph, its list of graph task questions, and the respective answers, along with a Chain-of-Thought reasoning that explains how each answer was obtained. The model then learns from this example to handle a new graph of the same encoding type and its own list of tasks.
5. **Adjacency-CoT** This method supplies the model with a graph, its adjacency matrix representation, and its list of graph task questions, all without any prior training.

4.4 Graph Question-Answer Tasks

1. **Edge existence**. Determine whether a given edge exists in a graph.
2. **Node degree**. Calculate the degree of a given node in a graph.
3. **Node count**. Count the number of nodes in a graph.
4. **Edge count**. Count the number of edges in a graph.
5. **Connected nodes**. Find all the nodes that are connected to a given node in a graph.
6. **Cycle check**. Determine whether a graph contains a cycle.
7. **Disconnected nodes**. Find all the nodes that are not connected to a given node in a graph.
8. **Reachability**. Determine whether there is a path from one node to another.
9. **Shortest path**. Calculate the length of the shortest path from one node to another.
10. **Weakly Connected**. Determine whether a graph is weakly connected.

4.5 Graph Encoding Types

1. **Texting.** using well-known english first names as node encoding and texting edge encoding.
2. **Commute.** using popular American cities as node encoding and traveling edge encoding.
3. **Citation.** using well-known diverse last names as node encoding and citation edge encoding.
4. **Adjacency.** using integer node encoding and parenthesis edge encoding.

5 Experiments and results

As previously mentioned, we generated a custom dataset of weighted, directed graphs by writing code that generates random weights and edges between nodes in 4 distinct encoding types. Once 40 graphs and the corresponding adjacency matrices were generated (10 graphs per encoding type),

these graphs were saved and organized in a spreadsheet (linked in section 2). Next, an answer key had to be generated in order to evaluate the LLM's performance in answering the graph prompting tasks. These answer keys were generated by writing python methods that could take in a graph's adjacency matrix and correctly answer each graph task that we posed. Such methods were tested with a variety of edge cases, ensuring that only accurate and valid answers were returned. After running each graph in the database through this code, answer keys were generated and saved in the same spreadsheet.

Next, work towards prompting begun. For each prompting method, 10 graphs were randomly assigned for a total of 50 assigned graphs (some graphs were repeated, and some were not used at all). Moreover, 4 graphs were randomly picked to be CoT graphs, which means we wrote thorough chain of thought reasoning paragraphs for each answer obtained, which were used for Random-CoT and Matched-CoT. After this, prompting scripts had to be written, which was automated by writing python scripts that automatically filled in the prompting script based on predetermined elements, such as always asking the LLM to calculate the edge existence between the first and third names in the name bank (name banks are generated to accompany adjacency matrices - refer to the google colab for the implementation).

Lastly, these prompting scripts were inputted in the LLM, in varying formats according to the prompting method that was being utilized. Examples of these various formats can be found in the appendix. After the LLM outputted its answers to the list of graph tasks, the responses were saved to the spreadsheet. Once all responses were recorded, scoring begun. Scoring was conducted by adding a point for each question the LLM correctly answered, for a maximum total of 10 points per graph. This allowed us to see how different prompting methods performed. Additionally, we recorded which questions earned points for every graph to detect possible trends between specific graph tasks and the LLM's performance. Such scoring was completed on "LLM Answers 1" on the linked spreadsheet. Moreover, we wanted to also decipher trends between LLM performance and encoding type.

Table 1: Comparison of Prompting Methods (Score Sum)

Method	Score
Zero-Shot	29
Few-Shot	41
Random CoT	59
Matched CoT	50
Adjacency CoT	39

The Random Chain of Thought (CoT) method achieved the highest score, of 59 points out of 100. This most likely can be attributed to the nature of CoT reasoning, as it better suits the LLM's strengths in processing and generating textual content. Similarly, while still highly effective, the Matched CoT method scored slightly lower than Random CoT. This discrepancy might be due to the LLM overfitting to the particular reasoning patterns presented, leading to less flexibility in handling varied tasks. It appears that the randomness introduced in Random-CoT aids in mitigating any potential biases or confusion, allow for more generalized understanding and learning. Few-Shot prompting demonstrated moderate effectiveness, outperforming Zero-Shot but lagging behind CoT methods. Understandably, providing a few examples helps the LLM understand the task better, yet without the structured reasoning of CoT prompts, it does not reach the same level of performance. The Adjacency CoT method did not perform as well as expected, placing below Few-Shot and the other CoT methods. This may be due to the reliance on numerical adjacency matrices, which are less intuitive for the LLM to interpret compared to textual data. The introduction of numbers instead of natural language text likely contributed to confusion and lower performance. As anticipated, Zero-Shot prompting scored the lowest, serving as the baseline for comparison. Without any examples or structured reasoning to guide it, the LLM struggled to accurately interpret and answer the graph-related tasks, reflecting the inherent difficulty of the tasks without additional context.

The Texting encoding method achieved the highest average score. One possible explanation for this superior performance is the use of alphabetized names, which may aid the LLM in encoding and processing the graph information more effectively. Additionally, it is conceivable that GPT-3, being trained on a substantial corpus of social media data, finds this format familiar and easier to interpret.

The Adjacency encoding method performed well, securing the second highest average score. The inclusion of explicit mentions of "directed and weighted" in the graph explanation might have

Table 2: Comparison of Encoding Methods (Score Avg)

Method	Score
Texting	4.867
Commute	3.456
Citation	4.318
Adjacency	4.333

provided clearer context, enhancing the LLM's understanding. However, this might also have caused slight inconsistencies, as other graphs did not contain this specific terminology, potentially affecting the overall results.

The Citation encoding method demonstrated moderate effectiveness, slightly below the Adjacency method. The use of diverse names in Citation encoding might have posed a challenge for the LLM, leading to difficulties in processing and linking the nodes accurately. The varied nature of the names could have introduced additional complexity, affecting performance.

The Commute encoding method resulted in the lowest average score. It is possible that the use of real city names might have introduced confusion for the LLM. Given that LLMs are often trained on factual information, the presence of real-world city names could have led to context-based misinterpretations, reducing accuracy in solving the graph tasks.

Table 3: Comparison of Graph Task Questions (Score Sum)

Task	Score
Edge Existence	28
Node Degree	7
Node Count	29
Edge Count	3
Connected Nodes	4
Cycle Check	37
Disconnected Node	21
Reachability	40
Shortest Path	18
Weakly Connected	31

The Reachability and Cycle Check tasks received the highest scores, indicating strong performance. This is likely due to both the graph structure and the nature of the graph tasks themselves. The graphs in the custom dataset were typically large and highly connected, often containing cycles. This high connectivity made it easier for the LLM to identify reachability and cycles, as many nodes were only one or two edges apart. Additionally, both tasks are binary (true/false), which inherently reduces the potential for error. Even if the LLM's reasoning was imperfect, it still had a substantially higher chance of providing the correct answer. The regular presence of cycles and reachable nodes in the dataset provided consistent patterns for the LLM to recognize and learn from. Similar to reachability, weakly connected components often involved binary decisions and benefited from the high connectivity of the graphs. Despite its slightly lower performance, it's likely these factors made it easier for the LLM to deduce weak connections between nodes.

Node Count and Edge Existence scored moderately in relation to all graph tasks. Counting nodes is a straightforward task, which the LLM handled relatively well due to the simplicity of the operation. Similarly, determining if an edge exists between two nodes is a basic relational task. However, its performance was not as high as reachability or cycle check, potentially due to the varying formats and contexts in which the question was presented.

The shortest path task had a lower score, likely due to the fact computing this task is more complex, involving multiple steps and considerations. The LLM may have struggled with this complexity, especially in large graphs with many potential paths. Additionally, the LLM struggled in identifying disconnected nodes, as it requires the LLM to effectively assess graph connectivity, a task it handled better than some but not as well as the simpler binary tasks.

Node Degree, Edge Count, and Connected Nodes tasks had the lowest scores. These tasks are highly influenced by whether the graph is directed or undirected, which can cause significant confusion for the LLM. Counting edges or connected nodes requires understanding the directionality and weight of connections, adding layers of complexity. Moreover, variations in how these tasks were presented and the specific details required likely contributed to the LLM's difficulties, resulting in poor performance.

6 Discussion and Conclusion

6.1 Discussion

Improvements in Prompting Methods One of the key aspects that could have improved the LLM's performance is the way we prompted the model. The current study used a variety of prompting methods, but ensuring consistency and clarity across all tasks is crucial. Notably, some tasks, particularly those involving weighted and directed graphs, included specific terminologies such as "weighted and directed" in the prompt, which was not uniformly applied across all graphs. This inconsistency may have led to confusion and impacted the LLM's accuracy. Additionally, the manner in which questions are framed is crucial. Studies, such as those by Google, have shown that contextually framed questions that cater to the language processing strengths of LLMs result in better performance. Hence, structuring questions to align more closely with the LLM's natural language processing capabilities could lead to more accurate and reliable answers.

Distribution of Graph Types In future research, a more balanced representation of all graph types in each prompting scenario could provide a clearer picture of the LLM's capabilities and limitations. Notably, the distribution of graph types in the zero-shot setting leaned heavily towards commute graphs. This uneven distribution might have skewed the results. Additionally, increasing the volume of data used for training and testing the LLM could also enhance performance. More data would allow the model to learn from a wider variety of examples, potentially improving its ability to generalize and perform accurately on different graph tasks. Ensuring that all types of graphs are adequately represented and that the model is exposed to diverse scenarios would likely yield more reliable results.

Specificity in Tasks For certain tasks like cycle checks, providing more specific criteria could both enhance the model's performance and verify its accuracy in responding to such tasks. For instance, specifying the detection of cycles longer than a certain length (e.g., cycles greater than 4 nodes) could help us determine if the LLM can actually detect cycles properly, or if cycle check's high performance was more due to luck in this experiment.

6.2 Conclusion

This project initially aimed to identify specific methods that could enhance an LLM's performance in graph-related tasks. From the results, it appears that ensuring consistency and clarity in prompts, particularly regarding graph properties such as "weighted and directed," is crucial. Contextually framed questions that cater to the LLM's natural language processing strengths, as demonstrated in studies like Google's, tend to perform better. This is demonstrated by the performance in the Random-CoT method, which achieved the highest performance. This performance underscores the importance of leveraging the LLM's strengths in textual processing.

Future research directions should focus on developing sophisticated prompting methods that suit the strengths of LLMs in textual processing. By emphasizing methods like Random CoT, which effectively leverage the LLM's capabilities in understanding and reasoning about textual data, future studies can achieve more effective and interpretable outcomes in graph analysis tasks. Additionally, exploring the application of LLMs to more complex graph types, such as weighted and directed graphs, would provide valuable insights into their capabilities and limitations in handling diverse data formats. This aligns with the aim of extending previous works' understanding to apply more broadly to real-world scenarios, where graphs are often weighted and directed. By addressing these aspects, future studies can build on the findings of this research to achieve more robust and accurate results.

References

[1] Bahare Fatemi, Jonathan Halcrow, Bryan Perozzi (2024) *Talk like a graph: Encoding graps for large language models*, ICLR.

[2] Bowen Jin, Gang Liu, Chi Han, Meng Jiang, Heng Ji, Jiawei Han (2024) *Large Language Models on Graphs: A Comprehensive Survey*, arXiv:2312.02783.

[3] Jiawei Zhang (2023) *Graph-ToolFormer: To Empower LLMs with Graph Reasoning Ability via Prompt Augmented by ChatGPT*, arXiv:2304.11116.

[4] Palaash Agrawal (2023) *Can LLMs perform structured graph reasoning?*, arXiv:2402.01805.

[5] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, Denny Zhou (2023) *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, arXiv:2201.11903.

7 Appendix

7.1 Examples of Graph Encoding

1. Texting:
   ['George sent Alice 215 texts.', 'Alice sent George 0 texts.', 'George sent Charles 434 texts.', 'Charles sent Alice 109 texts.', 'Edward sent Charles 265 texts.', 'Edward sent George 137 texts.', 'Alice sent Edward 435 texts.', 'Edward sent Alice 313 texts.', 'Charles sent Edward 225 texts.', 'Charles sent George 0 texts.', 'George sent Edward 0 texts.', 'Alice sent Charles 169 texts.']
2. Commute:
   ['Flying from Miami to Atlanta takes 0 minutes because this flight is currently unavailable.', 'Flying from Miami to Chicago takes 483 minutes.', 'Flying from Miami to Philadelphia takes 91 minutes.', 'Flying from Atlanta to Chicago takes 291 minutes.', 'Flying from Philadelphia to Atlanta takes 10 minutes.', 'Flying from Chicago to Atlanta takes 94 minutes.', 'Flying from Philadelphia to Miami takes 0 minutes because this flight is currently unavailable.', 'Flying from Atlanta to Philadelphia takes 0 minutes because this flight is currently unavailable.', 'Flying from Philadelphia to Chicago takes 321 minutes.', 'Flying from Chicago to Miami takes 240 minutes.', 'Flying from Atlanta to Miami takes 0 minutes because this flight is currently unavailable.', 'Flying from Chicago to Philadelphia takes 122 minutes.']
3. Citation:
   ["Dr. Jones cited 0 of Dr. Garcia's scientific papers.", "Dr. Lee cited 275 of Dr. Jones's scientific papers.", "Dr. Smith cited 305 of Dr. Jones's scientific papers.", "Dr. Smith cited 0 of Dr. Lee's scientific papers.", "Dr. Lee cited 316 of Dr. Smith's scientific papers.", "Dr. Garcia cited 447 of Dr. Jones's scientific papers.", "Dr. Lee cited 0 of Dr. Garcia's scientific papers.", "Dr. Jones cited 0 of Dr. Smith's scientific papers.", "Dr. Garcia cited 10 of Dr. Lee's scientific papers.", "Dr. Smith cited 86 of Dr. Garcia's scientific papers.", "Dr. Garcia cited 245 of Dr. Smith's scientific papers.", "Dr. Jones cited 179 of Dr. Lee's scientific papers."]
4. Adjacency:
   In a directed graph, (i, j, w) means that node i and node j are connected with a directed edge from i to j with weight w. G describes this graph among nodes 0, 1, 2, 3, 4, 5, 6, 7. The edges in G are: (5, 7, 70) (1, 6, 66) (6, 5, 93) (1, 0, 44) (6, 1, 0) (6, 2, 0).

7.2 Prompt Script

1. Edge existence. Determine whether an edge exists between [1st name] and [3rd name], for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of [4th name]. Output as "2. 123" 3. Node count. Count the number of nodes in a graph. Output as "3. 123" 4. Edge count. Count the number of edges in a graph. Output as "4. 123" 5. Connected nodes. Find all the nodes that are connected to [3rd name] in a graph. Output as "5. 123" 6. Cycle check. Determine whether a graph contains a cycle. Output as "6. (True/False)" 7. Disconnected nodes. Find all the nodes that are not connected to [4th name] in a graph. Output as "7. 123" 8. Reachability. Determine whether there is a path from [1st name] and [3rd name]. Output as "8. (True/False)" 9. Shortest path. Calculate the length of the shortest path from [2nd name] and [4th

name]. Output as “9. (True/False)” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

7.3 Example of Chain-of-Thought

4 graphs were randomly selected, 1 from each encoding type. The following includes an example graph and the respective chain of thought reasoning:

Graph 3 (Texting Encoding)

[’Charles sent Helen 94 texts.’, ’Helen sent Diana 0 texts.’, ’Charles sent George 405 texts.’, ’Charles sent Edward 0 texts.’, ’Alice sent Fiona 378 texts.’, ’Alice sent Edward 0 texts.’, ’Alice sent Helen 23 texts.’, ’Diana sent George 20 texts.’, ’Edward sent Alice 94 texts.’, ’Charles sent Diana 0 texts.’, ’Helen sent George 143 texts.’, ’Charles sent Alice 138 texts.’, ’Diana sent Alice 0 texts.’, ’Helen sent Charles 421 texts.’, ’Charles sent Fiona 333 texts.’, ’George sent Charles 366 texts.’, ’George sent Diana 0 texts.’, ’Helen sent Edward 254 texts.’, ’Fiona sent Helen 406 texts.’, ’Fiona sent George 265 texts.’, ’Alice sent George 42 texts.’, ’Fiona sent Diana 0 texts.’, ’Diana sent Fiona 191 texts.’, ’George sent Edward 468 texts.’, ’George sent Helen 43 texts.’, ’Edward sent Diana 5 texts.’, ’Edward sent Helen 192 texts.’, ’Fiona sent Charles 65 texts.’, ’Edward sent Fiona 251 texts.’, ’Edward sent Charles 338 texts.’, ’Helen sent Fiona 0 texts.’, ’Diana sent Charles 88 texts.’, ’George sent Fiona 0 texts.’, ’Helen sent Alice 51 texts.’, ’Fiona sent Edward 119 texts.’, ’Fiona sent Alice 0 texts.’, ’Alice sent Charles 142 texts.’, ’George sent Alice 141 texts.’, ’Edward sent George 438 texts.’, ’Alice sent Diana 67 texts.’, ’Diana sent Edward 480 texts.’, ’Diana sent Helen 57 texts.’]

1. Edge existence Question: Determine whether an edge exists between Alice and Diana, for both directions. Output as "1. (True/False, True/False)"

Answer: 1. (True, False)

Chain of Thought: To determine if an edge exists between Alice and Diana in both directions, we check the provided list of text messages. An edge from Alice to Diana exists if Alice sent any texts to Diana. From the data, we see that Alice sent 67 texts to Diana, confirming an edge. Conversely, an edge from Diana to Alice exists if Diana sent any texts to Alice. The data shows Diana sent 0 texts to Alice, confirming no edge exists. Therefore, the answer is (True, False).

2. Node degree Question: Calculate the in-degree and out-degree of Edward. Output as “2. (123, 123)”

Answer: 2. (4, 6)

Chain of Thought: The in-degree of Edward is the count of nodes sending texts to Edward. By scanning the data, we find that Charles, Diana, Helen, and Fiona sent texts to Edward, giving an in-degree of 4. The out-degree of Edward is the count of nodes to which Edward sent texts. Edward sent texts to Alice, Diana, Helen, Fiona, Charles, and George, giving an out-degree of 6. Therefore, the answer is (4, 6).

3. Node count Question: Count the number of nodes in this graph. Output as “3. 123”

Answer: 3. 7

Chain of Thought: The number of nodes in the graph is the number of unique individuals involved in sending or receiving texts. From the provided data, the individuals are Alice, Charles, Diana, Edward, Fiona, George, and Helen, totaling 7 nodes. Therefore, the answer is 7.

4. Edge count Question: Count the number of edges in this graph. Output as “4. 123”

Answer: 4. 32

Chain of Thought: To count the edges, we identify all unique sender-recipient pairs with at least one text sent. By examining the data, we count 32 such unique pairs, confirming there are 32 edges in the graph. Therefore, the answer is 32.

5. Connected nodes Question: Find all the nodes that are connected to Diana in this graph. Output as “5. 123”

Answer: 5. [’Charles’, ’Edward’, ’Fiona’, ’George’, ’Helen’]

Chain of Thought: Nodes connected to Diana include those she sent texts to and those who sent texts to her. From the data, Diana sent texts to Charles, Edward, Fiona, and Helen. Additionally, Charles and George sent texts to Diana. Combining these lists, the nodes connected to Diana are Charles, Edward, Fiona, George, and Helen. Therefore, the answer is ['Charles', 'Edward', 'Fiona', 'George', 'Helen'].

6. Cycle check Question: Determine whether this graph contains a cycle. Output as "6. (True/False)"

Answer: 6. True

Chain of Thought: To check for cycles, we use Depth-First Search (DFS) with cycle detection. By performing DFS on each node and keeping track of visited nodes, we detect back edges indicating cycles. Applying this algorithm to the graph reveals the presence of cycles.

For example:

Start DFS from node 0. Traverse to nodes 3 and then 2. From node 2, traverse to node 6. From node 6, traverse back to node 3. Node 3 is already in the recursion stack, indicating a cycle (0 -> 3 -> 2 -> 6 -> 3).

This confirms the presence of cycles in the graph.Therefore, the answer is True.

7. Disconnected nodes Question: Find all the nodes that are not connected to Edward in this graph. Output as "7. 123"

Answer: 7. None

Chain of Thought: To identify nodes not connected to Edward, we look for nodes with no direct or indirect paths to or from Edward. Using Breadth-First Search (BFS) from Edward, we reach all nodes, indicating no disconnection. Therefore, the answer is None.

8. Reachability Question: Determine whether there is a path from Alice and Diana. Output as "8. (True/False)"

Answer: 8. (True, True)

Chain of Thought: To check reachability from Alice to Diana and vice versa, we use BFS or DFS. Starting from Alice, we find a direct path to Diana. Similarly, starting from Diana, we find a path to Alice through other nodes.

For example:

From node 0, traverse to nodes 3, 6, and 2. Since node 2 is reached, there is a path from node 0 to node 2.

Similarly, start from node 2 and check if node 0 can be reached.

By running BFS from node 2, we find a direct edge from node 2 to node 0, confirming the reverse path.

9. Shortest path Question: Calculate the length of the shortest path from Charles and Edward. Output as "9. 123"

Therefore, the answer is (True, True).

Answer: 9. (2, 1)

Chain of Thought: To find the shortest path from Charles to Edward, we apply Dijkstra's algorithm or BFS for unweighted graphs. We find the shortest path from Charles to Edward is through another node, giving a length of 2. The path from Edward to Charles is direct, giving a length of 1.

For example:

From node 1, traverse to nodes 4, 2, and 6. From node 4, traverse to nodes 0 and 5. From node 0, traverse to node 3, giving a total distance of 3 edges. To find the shortest path from node 3 to node 1:

Start from node 3 and find a direct edge to node 1, giving a distance of 1 edge.

Therefore, the answer is (2, 1).

10. Weakly Connected Question: Determine whether this graph is weakly connected. Output as “10. (True/False)”

Answer: 10. True

Chain of Thought: A graph is weakly connected if all nodes are connected when considering undirected edges. By treating all directed edges as undirected and using BFS, we traverse the entire graph without encountering disconnected components.

For example:

Start BFS from node 0. Traverse to nodes 3, 6, 2, 4, 5, and 1. Since all nodes are reachable and visited, the graph is weakly connected.

Therefore, the graph is weakly connected, and the answer is True. ability from Brown to Khan and vice versa, we use BFS or DFS. From the data, Dr. Brown cited Dr. Khan’s papers, and Dr. Khan cited Dr. Brown’s papers, indicating direct paths in both directions.

7.4 Example of Zero-Shot Prompting

Example of LLM Prompting for Zero-Shot:

Here is a graph:

[’Charles sent Helen 94 texts.’, ’Helen sent Diana 0 texts.’, ’Charles sent George 405 texts.’, ’Charles sent Edward 0 texts.’, ’Alice sent Fiona 378 texts.’, ’Alice sent Edward 0 texts.’, ’Alice sent Helen 23 texts.’, ’Diana sent George 20 texts.’, ’Edward sent Alice 94 texts.’, ’Charles sent Diana 0 texts.’, ’Helen sent George 143 texts.’, ’Charles sent Alice 138 texts.’, ’Diana sent Alice 0 texts.’, ’Helen sent Charles 421 texts.’, ’Charles sent Fiona 333 texts.’, ’George sent Charles 366 texts.’, ’George sent Diana 0 texts.’, ’Helen sent Edward 254 texts.’, ’Fiona sent Helen 406 texts.’, ’Fiona sent George 265 texts.’, ’Alice sent George 42 texts.’, ’Fiona sent Diana 0 texts.’, ’Diana sent Fiona 191 texts.’, ’George sent Edward 468 texts.’, ’George sent Helen 43 texts.’, ’Edward sent Diana 5 texts.’, ’Edward sent Helen 192 texts.’, ’Fiona sent Charles 65 texts.’, ’Edward sent Fiona 251 texts.’, ’Edward sent Charles 338 texts.’, ’Helen sent Fiona 0 texts.’, ’Diana sent Charles 88 texts.’, ’George sent Fiona 0 texts.’, ’Helen sent Alice 51 texts.’, ’Fiona sent Edward 119 texts.’, ’Fiona sent Alice 0 texts.’, ’Alice sent Charles 142 texts.’, ’George sent Alice 141 texts.’, ’Edward sent George 438 texts.’, ’Alice sent Diana 67 texts.’, ’Diana sent Edward 480 texts.’, ’Diana sent Helen 57 texts.’]

1. Edge existence. Determine whether an edge exists between Alice and Diana, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of Edward. Output as “2. (123, 123)” 3. Node count. Count the number of nodes in this graph. Output as “3. 123” 4. Edge count. Count the number of edges in this graph. Output as “4. 123” 5. Connected nodes. Find all the nodes that are connected to Diana in this graph. Output as “5. 123” 6. Cycle check. Determine whether this graph contains a cycle. Output as “6. (True/False)” 7. Disconnected nodes. Find all the nodes that are not connected to Edward in this graph. Output as “7. 123” 8. Reachability. Determine whether there is a path from Alice and Diana. Output as “8. (True/False)” 9. Shortest path. Calculate the length of the shortest path from Charles and Edward. Output as “9. 123” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

Adhere to the example formatting and do not add extraneous text.

7.5 Example of Few-Shot Prompting

Here is a graph: [’Flying from Miami to Atlanta takes 0 minutes because this flight is currently unavailable.’, ’Flying from Miami to Chicago takes 483 minutes.’, ’Flying from Miami to Philadelphia takes 91 minutes.’, ’Flying from Atlanta to Chicago takes 291 minutes.’, ’Flying from Philadelphia to Atlanta takes 10 minutes.’, ’Flying from Chicago to Atlanta takes 94 minutes.’, ’Flying from Philadelphia to Miami takes 0 minutes because this flight is currently unavailable.’, ’Flying from Atlanta to Philadelphia takes 0 minutes because this flight is currently unavailable.’, ’Flying from Philadelphia to Chicago takes 321 minutes.’, ’Flying from Chicago to Miami takes 240 minutes.’, ’Flying from Atlanta to Miami takes 0 minutes because this flight is currently unavailable.’, ’Flying from Chicago to Philadelphia takes 122 minutes.’]

Here are the questions we asked: 1. Edge existence. Determine whether an edge exists between Atlanta and Miami, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of Philadelphia. Output as “2. (123, 123)” 3. Node count. Count the number of nodes in this graph. Output as “3. 123” 4. Edge count. Count the number of edges in this graph. Output as “4. 123” 5. Connected nodes. Find all the nodes that are connected to Miami in this graph. Output as “5. 123” 6. Cycle check. Determine whether this graph contains a cycle. Output as “6. (True/False)” 7. Disconnected nodes. Find all the nodes that are not connected to Philadelphia in this graph. Output as “7. 123” 8. Reachability. Determine whether there is a path from Atlanta and Miami. Output as “8. (True/False)” 9. Shortest path. Calculate the length of the shortest path from Chicago and Philadelphia. Output as “9. 123” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

Here are the answers: 1. (False, False) 2. (2, 2) 3. 4 4. 8 5. [’Chicago’, ’Philadelphia’] 6. True 7. None 8. (True, True) 9. (1, 1) 10. True

Here is a graph: ["Dr. Johnson cited 0 of Dr. Khan’s scientific papers.", "Dr. Khan cited 493 of Dr. Johnson’s scientific papers.", "Dr. Garcia cited 136 of Dr. Johnson’s scientific papers.", "Dr. Nguyen cited 275 of Dr. Yue’s scientific papers.", "Dr. Garcia cited 0 of Dr. Khan’s scientific papers.", "Dr. Khan cited 348 of Dr. Yue’s scientific papers.", "Dr. Yue cited 10 of Dr. Nguyen’s scientific papers.", "Dr. Nguyen cited 0 of Dr. Khan’s scientific papers.", "Dr. Nguyen cited 450 of Dr. Garcia’s scientific papers.", "Dr. Yue cited 135 of Dr. Khan’s scientific papers.", "Dr. Khan cited 255 of Dr. Nguyen’s scientific papers.", "Dr. Garcia cited 273 of Dr. Yue’s scientific papers.", "Dr. Johnson cited 380 of Dr. Garcia’s scientific papers.", "Dr. Johnson cited 0 of Dr. Nguyen’s scientific papers.", "Dr. Garcia cited 0 of Dr. Nguyen’s scientific papers.", "Dr. Nguyen cited 337 of Dr. Johnson’s scientific papers.", "Dr. Johnson cited 93 of Dr. Yue’s scientific papers.", "Dr. Yue cited 427 of Dr. Garcia’s scientific papers.", "Dr. Yue cited 270 of Dr. Johnson’s scientific papers.", "Dr. Khan cited 0 of Dr. Garcia’s scientific papers."]

Here are the questions we asked: 1. Edge existence. Determine whether an edge exists between Garcia and Khan, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of Nguyen. Output as “2. (123, 123)” 3. Node count. Count the number of nodes in this graph. Output as “3. 123” 4. Edge count. Count the number of edges in this graph. Output as “4. 123” 5. Connected nodes. Find all the nodes that are connected to Khan in this graph. Output as “5. 123” 6. Cycle check. Determine whether this graph contains a cycle. Output as “6. (True/False)” 7. Disconnected nodes. Find all the nodes that are not connected to Nguyen in this graph. Output as “7. 123” 8. Reachability. Determine whether there is a path from Garcia and Khan. Output as “8. (True/False)” 9. Shortest path. Calculate the length of the shortest path from Johnson and Nguyen. Output as “9. 123” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

Here are the answers: 1. (False, False) 2. (2, 3) 3. 5 4. 14 5. [’Johnson’, ’Nguyen’, ’Yue’] 6. True 7. None 8. (True, True) 9. (2, 1) 10. True

Here is a graph: In a directed graph, (i, j, w) means that node i and node j are connected with a directed edge from i to j with weight w. G describes this graph among nodes 0, 1, 2, 3, 4, 5, 6. The edges in G are: (0, 1, 0) (2, 0, 72) (0, 3, 60) (5, 6, 12) (4, 5, 79) (6, 5, 5) (1, 0, 0) (1, 4, 87) (2, 4, 0) (6, 2, 0) (3, 4, 95) (4, 2, 0) (2, 3, 93) (6, 1, 8) (3, 6, 4) (5, 2, 51) (6, 4, 53) (4, 3, 0) (4, 0, 17) (0, 6, 80) (1, 6, 0) (5, 0, 67) (1, 5, 82) (0, 5, 77) (4, 6, 0) (5, 1, 44).

Here are the questions we asked: 1. Edge existence. Determine whether an edge exists between 0 and 2, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of 3. Output as “2. (123, 123)” 3. Node count. Count the number of nodes in this graph. Output as “3. 123” 4. Edge count. Count the number of edges in this graph. Output as “4. 123” 5. Connected nodes. Find all the nodes that are connected to 2 in this graph. Output as “5. 123” 6. Cycle check. Determine whether this graph contains a cycle. Output as “6. (True/False)” 7. Disconnected nodes. Find all the nodes that are not connected to 3 in this graph. Output as “7. 123” 8. Reachability. Determine whether there is a path from 0 and 2. Output as “8. (True/False)” 9. Shortest path. Calculate the length of the shortest path from 1 and 3. Output as “9. 123” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

Here are the answers: 1. (False, True) 2. (2, 2) 3. 7 4. 18 5. [0, 3] 6. True 7. None 8. (True, True) 9. (3, 2) 10. True

Here is your graph: ['Diana sent Bob 474 texts.', 'Charles sent George 118 texts.', 'Bob sent Diana 201 texts.', 'Alice sent Fiona 0 texts.', 'Charles sent Fiona 1 texts.', 'Fiona sent Bob 432 texts.', 'Alice sent Charles 471 texts.', 'Charles sent Alice 0 texts.', 'Bob sent Fiona 127 texts.', 'George sent Alice 0 texts.', 'Alice sent Bob 0 texts.', 'George sent Diana 120 texts.', 'Bob sent Alice 86 texts.', 'Diana sent Charles 103 texts.', 'George sent Fiona 369 texts.', 'Alice sent George 0 texts.', 'Charles sent Bob 0 texts.', 'George sent Charles 461 texts.', 'Alice sent Diana 0 texts.', 'Diana sent Alice 15 texts.', 'Charles sent Diana 377 texts.', 'Fiona sent George 438 texts.', 'George sent Bob 393 texts.', 'Diana sent Fiona 386 texts.', 'Fiona sent Diana 138 texts.', 'Diana sent George 417 texts.', 'Bob sent George 196 texts.', 'Fiona sent Alice 38 texts.', 'Bob sent Charles 143 texts.', 'Fiona sent Charles 396 texts.']

1. Edge existence. Determine whether an edge exists between Alice and Charles, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of Diana. Output as "2. (123, 123)" 3. Node count. Count the number of nodes in this graph. Output as "3. 123" 4. Edge count. Count the number of edges in this graph. Output as "4. 123" 5. Connected nodes. Find all the nodes that are connected to Charles in this graph. Output as "5. 123" 6. Cycle check. Determine whether this graph contains a cycle. Output as "6. (True/False)" 7. Disconnected nodes. Find all the nodes that are not connected to Diana in this graph. Output as "7. 123" 8. Reachability. Determine whether there is a path from Alice and Charles. Output as "8. (True/False)" 9. Shortest path. Calculate the length of the shortest path from Bob and Diana. Output as "9. 123" 10. Weakly Connected. Determine whether this graph is weakly connected. Output as "10. (True/False)"

Adhere to the example formatting and do not add extraneous text.

7.6 Example of Random/Matched-CoT Prompting

Here is a graph: ['Flying from Miami to Atlanta takes 0 minutes because this flight is currently unavailable.', 'Flying from Miami to Chicago takes 483 minutes.', 'Flying from Miami to Philadelphia takes 91 minutes.', 'Flying from Atlanta to Chicago takes 291 minutes.', 'Flying from Philadelphia to Atlanta takes 10 minutes.', 'Flying from Chicago to Atlanta takes 94 minutes.', 'Flying from Philadelphia to Miami takes 0 minutes because this flight is currently unavailable.', 'Flying from Atlanta to Philadelphia takes 0 minutes because this flight is currently unavailable.', 'Flying from Philadelphia to Chicago takes 321 minutes.', 'Flying from Chicago to Miami takes 240 minutes.', 'Flying from Atlanta to Miami takes 0 minutes because this flight is currently unavailable.', 'Flying from Chicago to Philadelphia takes 122 minutes.'] 1. Edge existence Question: Determine whether an edge exists between Atlanta and Miami, for both directions. Output as "1. (True/False, True/False)"

Answer: 1. (False, False)

Chain of Thought: To determine if an edge exists between Atlanta and Miami in both directions, we check the flight availability. From the data, there are flights listed from Atlanta to Miami and Miami to Atlanta, but they are currently unavailable, indicating no edges exist. Therefore, the answer is (False, False).

2. Node degree Question: Calculate the in-degree and out-degree of Philadelphia. Output as "2. (123, 123)"

Answer: 2. (2, 2)

Chain of Thought: To find the in-degree of Philadelphia, we count the number of flights arriving in Philadelphia. From the data, flights from Miami and Chicago land in Philadelphia, giving an in-degree of 2. For the out-degree, we count the flights departing from Philadelphia to other cities. Flights from Philadelphia go to Atlanta and Chicago, giving an out-degree of 2. Therefore, the answer is (2, 2).

3. Node count Question: Count the number of nodes in this graph. Output as "3. 123"

Answer: 3. 4

Chain of Thought: The number of nodes in the graph corresponds to the number of unique cities in the flight data. The cities involved are Miami, Atlanta, Chicago, and Philadelphia, totaling 4 nodes. Therefore, the answer is 4.

4. Edge count Question: Count the number of edges in this graph. Output as "4. 123"

Answer: 4. 8

Chain of Thought: To count the edges, we identify all unique available flights between cities. By examining the data, we count 8 such flights: Miami to Chicago, Miami to Philadelphia, Atlanta to Chicago, Philadelphia to Atlanta, Chicago to Atlanta, Philadelphia to Chicago, Chicago to Miami, and Chicago to Philadelphia. Therefore, the answer is 8.

5. Connected nodes Question: Find all the nodes that are connected to Miami in this graph. Output as "5. 123"

Answer: 5. ['Chicago', 'Philadelphia']

Chain of Thought: Nodes connected to Miami include those with direct flights to or from Miami. From the data, there are flights from Miami to Chicago and Philadelphia. Additionally, there is a flight from Chicago to Miami. Therefore, the nodes connected to Miami are ['Chicago', 'Philadelphia'].

6. Cycle check Question: Determine whether this graph contains a cycle. Output as "6. (True/False)"

Answer: 6. True

Chain of Thought: To check for cycles, we use Depth-First Search (DFS) with cycle detection. By performing DFS on each node and keeping track of visited nodes, we detect back edges indicating cycles. Applying this algorithm to the graph reveals the presence of cycles (e.g., Miami -> Philadelphia -> Atlanta -> Chicago -> Miami).

For example:

Start DFS from Miami. Traverse to Chicago, then Atlanta, and finally back to Miami, indicating a cycle (Miami -> Chicago -> Atlanta -> Miami).

Therefore, the answer is True.

7. Disconnected nodes Question: Find all the nodes that are not connected to Philadelphia in this graph. Output as "7. 123"

Answer: 7. None

Chain of Thought: To identify nodes not connected to Philadelphia, we look for nodes with no direct or indirect paths to or from Philadelphia. Using Breadth-First Search (BFS) or Depth-First Search (DFS) starting from Philadelphia, we can reach all other nodes. Therefore, there are no nodes disconnected from Philadelphia. Thus, the answer is None.

8. Reachability Question: Determine whether there is a path from Atlanta and Miami. Output as "8. (True/False)"

Answer: 8. (True, True)

Chain of Thought: To check reachability from Atlanta to Miami and vice versa, we use BFS or DFS. Even though direct flights are unavailable, we can find paths through other cities. For example, Atlanta -> Chicago -> Miami and Miami -> Philadelphia -> Chicago -> Atlanta are possible paths.

For example:

From Miami, traverse to Philadelphia directly. BFS Traversal from Philadelphia: Similarly, start from Philadelphia and check if Miami can be reached. By running BFS from Philadelphia, we find an indirect path to Miami through other nodes, confirming the reverse path.

Therefore, the answer is (True, True).

9. Shortest path Question: Calculate the length of the shortest path from Chicago and Philadelphia. Output as "9. 123"

Answer: 9. (1, 1)

Chain of Thought: To find the shortest path from Chicago to Philadelphia and vice versa, we apply BFS for unweighted graphs or Dijkstra's algorithm for weighted graphs. From the data, we find direct flights between Chicago and Philadelphia with a single flight leg. Therefore, the length of the shortest path is 1 for both directions.

For example:

From Chicago, traverse directly to Philadelphia, giving a distance of 1 edge.

Thus, the answer is (1, 1).

10. Weakly Connected Question: Determine whether this graph is weakly connected. Output as “10. (True/False)”

Answer: 10. True

Chain of Thought: A graph is weakly connected if all nodes are connected when considering undirected edges. By treating all directed edges as undirected and using BFS or DFS, we traverse the entire graph without encountering disconnected components.

For example:

Start BFS from Miami. Traverse to nodes Chicago, Atlanta, and Philadelphia. Since all nodes are reachable and visited, the graph is weakly connected.

Therefore, the graph is weakly connected, and the answer is True. Here is your graph: ’Flying from D.C. to Phoenix takes 336 minutes.’, ’Flying from D.C. to Houston takes 158 minutes.’, ’Flying from Phoenix to Columbus takes 304 minutes.’, ’Flying from Columbus to D.C. takes 458 minutes.’, ’Flying from Columbus to Phoenix takes 144 minutes.’, ’Flying from D.C. to Philadelphia takes 0 minutes because this flight is currently unavailable.’, ’Flying from Philadelphia to Houston takes 0 minutes because this flight is currently unavailable.’, ’Flying from Houston to Philadelphia takes 0 minutes because this flight is currently unavailable.’, ’Flying from D.C. to Columbus takes 11 minutes.’, ’Flying from Houston to Phoenix takes 0 minutes because this flight is currently unavailable.’, ’Flying from Columbus to Houston takes 282 minutes.’, ’Flying from Phoenix to D.C. takes 265 minutes.’, ’Flying from Philadelphia to D.C. takes 267 minutes.’, ’Flying from Philadelphia to Columbus takes 0 minutes because this flight is currently unavailable.’, ’Flying from Houston to Columbus takes 329 minutes.’, ’Flying from Phoenix to Philadelphia takes 63 minutes.’, ’Flying from Phoenix to Houston takes 79 minutes.’, ’Flying from Columbus to Philadelphia takes 158 minutes.’, ’Flying from Philadelphia to Phoenix takes 341 minutes.’, ’Flying from Houston to D.C. takes 0 minutes because this flight is currently unavailable.’] 1. Edge existence. Determine whether an edge exists between Columbus and Houston, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of Philadelphia. Output as “2. (123, 123)” 3. Node count. Count the number of nodes in this graph. Output as “3. 123” 4. Edge count. Count the number of edges in this graph. Output as “4. 123” 5. Connected nodes. Find all the nodes that are connected to Houston in this graph. Output as “5. 123” 6. Cycle check. Determine whether this graph contains a cycle. Output as “6. (True/False)” 7. Disconnected nodes. Find all the nodes that are not connected to Philadelphia in this graph. Output as “7. 123” 8. Reachability. Determine whether there is a path from Columbus and Houston. Output as “8. (True/False)” 9. Shortest path. Calculate the length of the shortest path from D.C. and Philadelphia. Output as “9. 123” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

Adhere to the example formatting and do not add extraneous text.

7.7 Example of Adjacency-CoT Prompting

Adjacency prompting

Here is a graph:

[’Flying from Miami to Atlanta takes 0 minutes because this flight is currently unavailable.’, ’Flying from Miami to Chicago takes 483 minutes.’, ’Flying from Miami to Philadelphia takes 91 minutes.’, ’Flying from Atlanta to Chicago takes 291 minutes.’, ’Flying from Philadelphia to Atlanta takes 10 minutes.’, ’Flying from Chicago to Atlanta takes 94 minutes.’, ’Flying from Philadelphia to Miami takes 0 minutes because this flight is currently unavailable.’, ’Flying from Atlanta to Philadelphia takes 0 minutes because this flight is currently unavailable.’, ’Flying from Philadelphia to Chicago takes 321 minutes.’, ’Flying from Chicago to Miami takes 240 minutes.’, ’Flying from Atlanta to Miami takes 0 minutes because this flight is currently unavailable.’, ’Flying from Chicago to Philadelphia takes 122 minutes.’]

Here is its adjacency matrix:

[[0 291 0 0] [94 0 240 122] [0 483 0 91] [10 321 0 0]]

1. Edge existence. Determine whether an edge exists between Atlanta and Miami, for both directions. Output as "1. (True/False, True/False)" 2. Node degree. Calculate the in degree and out degree of Philadelphia. Output as “2. (123, 123)” 3. Node count. Count the number of nodes in this graph. Output as “3. 123” 4. Edge count. Count the number of edges in this graph. Output as “4. 123” 5. Connected nodes. Find all the nodes that are connected to Miami in this graph. Output as “5. 123” 6. Cycle check. Determine whether this graph contains a cycle. Output as “6. (True/False)” 7. Disconnected nodes. Find all the nodes that are not connected to Philadelphia in this graph. Output as “7. 123” 8. Reachability. Determine whether there is a path from Atlanta and Miami. Output as “8. (True/False)” 9. Shortest path. Calculate the length of the shortest path from Chicago and Philadelphia. Output as “9. 123” 10. Weakly Connected. Determine whether this graph is weakly connected. Output as “10. (True/False)”

Adhere to the example formatting and do not add extraneous text.