MMCODE: BENCHMARKING MULTIMODAL LARGE LANGUAGE MODELS IN CODE GENERATION WITH VISUALLY RICH PROGRAMMING PROBLEMS

Kaixin Li[1] **Yuchen Tian**[2] **Qisheng Hu**[3] **Ziyang Luo**[2] **Zhiyong Huang**[1] **Jing Ma**[2]
[1]National University of Singapore [2]Hong Kong Baptist University
[3]Nanyang Technological University
likaixin@u.nus.edu

ABSTRACT

Programming often involves converting detailed and complex specifications into code, a process during which developers typically utilize visual aids to more effectively convey concepts. While recent developments in Large Multimodal Models have demonstrated remarkable abilities in visual reasoning and mathematical tasks, there is little work on investigating whether these models can effectively interpret visual elements for code generation. To this end, we present MMCode, the first multi-modal coding dataset for evaluating algorithmic problem-solving skills in visually rich contexts. MMCode contains 3,548 questions and 6,620 images collected from real-world programming challenges harvested from 10 code competition websites, presenting significant challenges due to the extreme demand for reasoning abilities. Our experiment results show that current state-of-the-art models struggle to solve these problems. The results highlight the lack of powerful vision-code models, and we hope MMCode can serve as an inspiration for future works in this domain. The data and code are publicly available.

1 INTRODUCTION

Programming is primarily aimed at fulfilling requirements, frequently entailing the translation of detailed and intricate specifications into executable code (Nuseibeh & Easterbrook, 2000). In this endeavor, human developers regularly employ visual aids such as images and diagrams to facilitate effective communication and a better understanding of concepts (Agarwal & Sinha, 2003).

Recently, automated code generation tools have attracted significant attention, largely attributing to the substantial advance in Code Large Language Models (Code LLMs) (Chen et al., 2021; Nijkamp et al., 2023; Roziere et al., 2023; Luo et al., 2023b; Li et al., 2023a; Guo et al., 2024). These models demonstrated unprecedentedly remarkable coding abilities, potentially assist to enhance productivity, reduce human error and democratize coding skills. Nevertheless, these models are limited to processing text-only inputs, lacking the ability to interpret rich information presented through images.

In a closely related development, the field has also observed the emergence of many powerful Large Multimodal Models (LMMs), marked by GPT-4V (OpenAI, 2023b) and Gemini (Team Gemini et al., 2023), representing a significant step forward in bridging the modality of text and images. While there are multiple works evaluating these models in mathematical reasoning (Lu et al., 2023), perception and reasoning (Liu et al., 2023) and instruction-following (Ye et al., 2023), there is a notable gap in evaluating LMMs for code generation.

To this end, we present MMCode, the first multi-modal benchmark for rigorously evaluating the code generation ability of Large Multimodal Models. It comprises 3,548 questions with 6,620 images collated from 10 programming-related websites encompassing a broad spectrum of subjects, extending from fundamental coding concepts to the application of code for solving mathematical problems. The generated code is rigorously checked by test cases. The overall framework is illustrated in Figure 1.

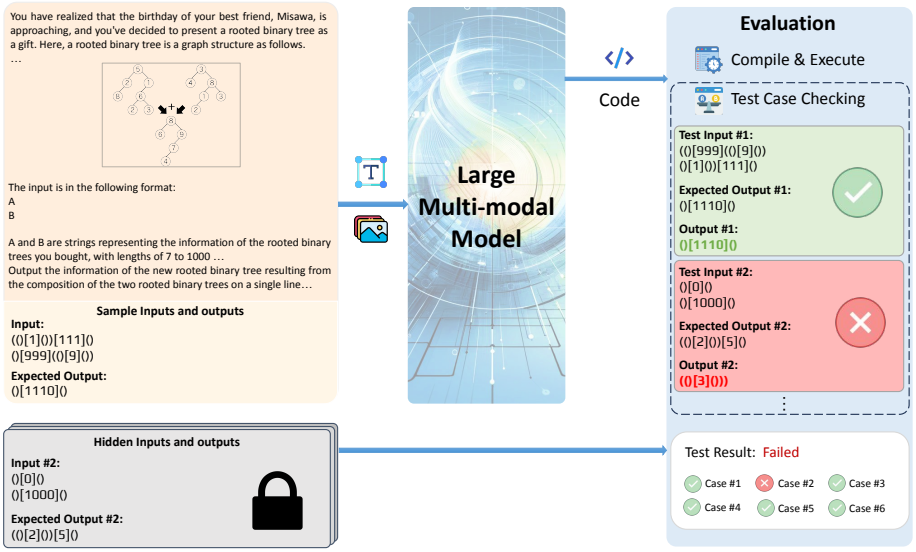


Figure 1: An illustration of an example question and the automatic testing pipeline of MMCode. The tests in the judger are selected for display. The actual test cases are harder than the sample inputs and outputs.

Our experiments revealed that current LMMs struggle significantly to solve the tasks in MMCode. The most powerful LMMs, GPT-4V and Gemini, scored unsatisfactory pass rates as low as 19.4% and 5.0%, potentially due to the requirement of intense reasoning on the text descriptions and images. Open-source LMMs (Liu et al., 2024a; Bai et al., 2023) yield negligible pass rates because of their inability to understand the abstract meaning of the images. The findings reveal a significant deficiency in current LMMs' ability to interpret and utilize multimodal information for code generation, highlighting an imperative need for further advancements in this area. We believe MMCode will serve as a pivotal benchmark for evaluating the forthcoming evolution of Code LMMs and inspire research in this area.

2 Related Works

2.1 Code Large Language Models

Large Language Models (LLMs) have experienced significant advancements in recent years, demonstrating remarkable progress in their capabilities and applications that were previously unattainable (Ouyang et al., 2022; Brown et al., 2020; OpenAI, 2022; 2023a; Touvron et al., 2023a;b; Chowdhery et al., 2022; Anil et al., 2023; Hoffmann et al., 2022; Scao et al., 2022). Building on their increasing proficiency at understanding and generating human-like text, a set of specialized models known as Code LLMs have emerged, focusing specifically on programming code (Chen et al., 2021; Nijkamp et al., 2023; Roziere et al., 2023; Li et al., 2023a; Luo et al., 2023b; Guo et al., 2024). Trained on large corpora of code data, these models have acquired the capacity to comprehend programming contexts and generate syntactically correct and logically sound code snippets. However, a significant limitation of these tools is their inability to process image inputs, restricting their application to environments where interaction is solely text- or code-based. Such a deficiency precludes their use in scenarios requiring the interpretation of visual data.

2.2 Coding Benchmarks

Accompanying the rapid development of Code Large Language Models, numerous benchmarks and datasets have witnessed the astonishing advancements of Code LLMs. These benchmarks cover a wide area of code-related tasks, such as code completion (Chen et al., 2021; Zheng et al., 2023;

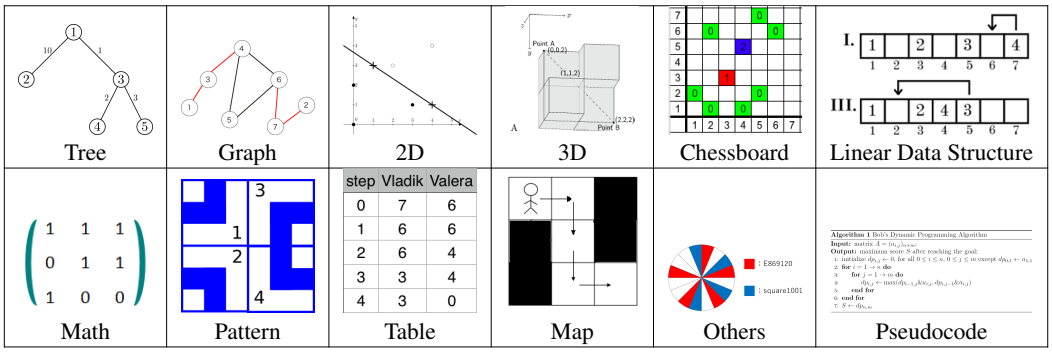


Table 1: Examples of images from each category. Some images are cropped for better visualization.

Austin et al., 2021; Yan et al., 2023a), editing (Li et al., 2024; Tian et al., 2024) and translation (Yan et al., 2023b). Most relevant to our work, APPS (Hendrycks et al., 2021) and CodeContests (Li et al., 2022) leveraged coding problems from real-world practice and contest coding websites as benchmarks. Recently, TACO (Li et al., 2023b) contributed a comprehensive collection of contest problems. However, it aims to cluster the problems by the programming skills needed (e.g. Dynamic Programming and Tree Algorithms), while MMCode focuses on image-augmented questions to assess the question-solving skills of multi-modal language models.

2.3 Reasoning-Intense Visual Question Answering

Several works have emerged to assess the reasoning capabilities of LMMs with visual contexts. ScienceQA (Lu et al., 2022) consists of multimodal multiple-choice questions across scientific topics, designed to measure the multi-hop reasoning ability. MMMU (Yue et al., 2023) features college-level questions with multi-disciplinary subjects. MathVista (Lu et al., 2023) emphasizes mathematical problem-solving with multi-modal input, involving tasks that require diverse math reasoning skills. OlympiadBench (He et al., 2024) offers a set of challenging Olympiad-level mathematics and physics contest questions. PuzzleVQA (Chia et al., 2024) benchmarks LMMs on patterns in order to evaluate if the models' reasoning ability generalizes to abstract figures. Our work distinguishes itself by necessitating the generation of solution code of complex problems, which benchmarks LMMs for long-horizon reasoning.

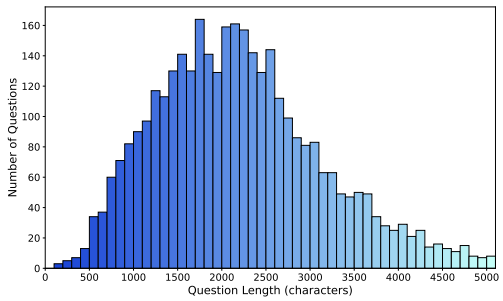
3 MMCode

In this section, we introduce the source and collection pipeline of MMCode. The collection pipeline comprises four stages: 1) Raw data collection; 2) automatic filtering; 3) human filtering and 4) annotation. This pipeline to be introduced in the following sections guarantees the quality and diversity of the data collected for MMCode.

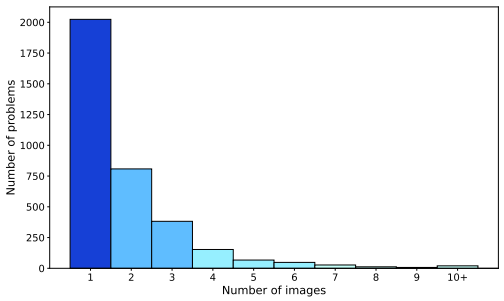
3.1 Data Sources

The questions of MMCode are collected from 10 coding platforms, including AtCoder, Aizu, CodeChef, CodeForces, CodeWars, Project Euler, Geeksforgeeks, HackerRank, Leetcode and Open Kattis. More information can be found in Appendix A.

The data sources exhibit a wide range of characteristics and purposes, including competitions, job interviews, and tutorials, etc. Notably, Project Euler is distinguished by its collection of challenges that necessitate a combination of mathematical and computer programming skills to solve. As a result, MMCode benefits from the diversity of these sources, offering programming problems with varying difficulties, styles, and skill requirements.



(a) The distribution of lengths of the question statements, measured by the number of characters.



(b) The distribution of numbers of images per question. Questions with 10 or more images are combined into the last bin "10+".

Figure 2: Data statistics of the questions in MMCode.

3.2 Data Collection Pipeline

Raw Data Collection. For each of the 10 platforms, distinct web crawlers were developed to retrieve the problem statements. The HTML elements were then converted to plain texts following unified rules to ensure cleanliness and readability. Furthermore, the metadata of these questions was collected conditionally on availability, e.g. problem name, time limit, and memory limit. It is noteworthy that we also included the raw HTML code in our dataset for further flexible use.

If there were images (`<img>` tags) encapsulated within the statements, we saved them and converted them to PNG format. The tags were replaced with markdown tags to insert them in the text (e.g. `![image](1.png)`). It is essential to maintain the locations of the images in the text because a question may encompass multiple images, and the images can be closely related to the text sections around them. This practice ensures the cohesion and coherence of the contents, where visual and textual elements are harmoniously integrated for better understanding.

Due to the difficulty of obtaining the automated test cases as a result of the changes in platforms' designs and policies, we also reused the rich information from the TACO dataset (Li et al., 2023b) where feasible. We matched the crawled questions with those existing in TACO by URLs[1]. Specifically, we crawled all questions from the largest two data sources, CodeForces and Aizu, including problem statements and test cases. Additionally, we included a new platform Project Euler that is not present in previous datasets. For other platforms, we reused the data from TACO and fetched the question statements to add the images.

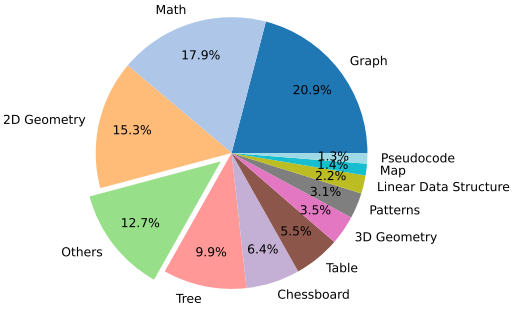
An initial data analysis revealed that 18.8% of the obtained questions contained images, corroborating our motivation for creating a multi-modal coding benchmark.

Automated Filtering. In this phase, our initial step involved excluding questions that do not include associated images. Subsequently, we applied various post-processing steps to ensure the quality of the data. We filtered questions with images unable to load using the PILLOW library[2]. Additionally, we converted PNGs with alpha channels to pure RGB format by painting the background to pure white, which is critical for discerning the texts on the images. This avoids distinct behaviors of different models interpreting the transparent color. Finally, a strict 5-gram similarity is conducted on every pair of question statements in the dataset to remove similar problems with a similarity score greater than 0.80. This process eliminated 33 questions from the dataset.

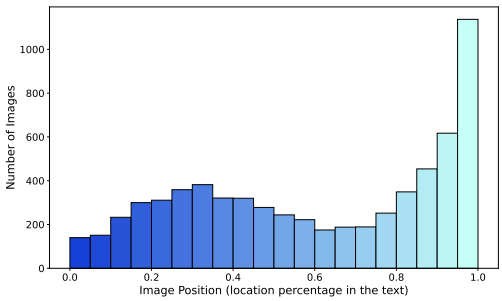
Human Filtering. At this stage, a preliminary inspection of sampled questions was first conducted to scope the quality of the collected data. The primary source of noise was found to be teaser images that try to interest the readers but do not provide information or implications to help solve

[1]TACO comprises questions obtained from mixed sources including CodeContests, APPS, and by crawling the websites. As a result, not all questions are provided with URLs. We abandoned the questions without URL metadata to ensure the overall quality of our data.

[2]https://github.com/python-pillow/Pillow



(a) The distribution of lengths of the question statements, measured by the number of characters.



(b) The distribution of the position of the images, measured by the percentage of their placement in the question statement.

Figure 3: Data statistics of the images in MMCode.

the questions. These images mostly originate from Open Kattis and CodeForces, consisting of photographs about the background of the question, anime screenshots, etc. An example is presented in Appendix D.10, where the question is about developers' cooperation, but the image is a humorous comic about the daily work of programmers. We also spotted some mixture of website logos and UI elements in the images, probably due to mistakes of the question creators in typesetting.

To address this problem, a convenient solution is to employ large LMMs such as GPT-4V and Gemini to determine if the image(s) are useful in addressing the question. Nonetheless, such a method may potentially introduce bias into the data. Therefore, we decided to opt for human labor to filter out these unrelated images. We manually examined every image in the dataset to remove the noisy ones. Note that when an image was deemed irrelevant but was not the sole image in the question, we exclusively removed this image and its corresponding markdown tag from the text. The question itself is only eliminated if there are no images remaining after this process.

Annotation. In this stage, we annotate the images in MMCode into distinct categories in order to facilitate a more detailed analysis of model performance across various types of images. The images were examined and discussed by expert human coders who have rich experience in solving coding contest problems. Following this deliberation, the images were meticulously categorized into 12 types: `Linear Data Structure`, `Tree`, `Graph`, `2D Geometry`, `3D Geometry`, `Chessboard`, `Map`, `Patterns`, `Math`, `Table`, `Pseudocode`, and `Others`. Gemini Pro Vision is leveraged to generate the coarse labels. Detailed descriptions of the categories are listed in Appendix B.

This detailed categorization facilitates a focused analysis on how different types of visual information are processed and interpreted by models, thereby potentially aiding in the identification and improvement of their abilities in coding contexts.

3.3 Data Splits

After performing the previous procedures, we acquired a dataset with 3,548 questions with 6,620 images. Considering the lengthy nature of the questions and additional tokens needed to represent the images, evaluating on the full dataset can be expensive. Following MathVista (Lu et al., 2023), a conscious decision was made to keep the test set small. As a result, we sampled 263 questions as the test set, and applied careful human inspection to correct the image category labels.

3.4 Testing Pipeline

An execution-based testing pipeline is adopted in MMCode for rigorous answer checking, following Hendrycks et al. (2021); Li et al. (2023b); Chen et al. (2021). As demonstrated in Figure 1, the judger attempts to compile the code generated by models, followed by a timed execution in a sandbox. The programs' outputs are checked against the ground truth answers in the test cases, and the solution is judged as correct only if it passes all hidden test cases.

Model	Task Type												Average
	Linear	Tree	Graph	2D	3D	Chessboard	Map	Math	Patterns	Table	Pseudocode	Others	
Language Only Inputs													
LLaVA-1.5-7B	8.0	0.0	0.0	0.0	0.0	6.7	0.0	0.0	0.0	0.0	0.0	0.0	1.1
LLaVA-1.5-13B	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.7	0.0	10.0	0.0	1.5
QWEN-VL	4.0	0.0	0.0	0.0	0.0	6.7	0.0	0.0	0.0	0.0	10.0	0.0	1.1
CodeGemma-7b-Instruct	12.0	0.0	0.0	0.0	3.8	6.7	3.6	0.0	3.7	0.0	20.0	0.0	3.4
CodeLLaMA-7b-instruct	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	0.0	1.1
CodeLLaMA-13b-instruct	8.0	0.0	0.0	0.0	0.0	6.7	0.0	0.0	0.0	0.0	10.0	0.0	1.5
DeepSeekCoder-7b-instruct	16.0	0.0	4.3	3.3	3.8	20.0	3.6	0.0	3.7	7.1	10.0	3.8	5.7
DeepSeekCoder-33b-instruct	16.0	0.0	8.7	10.0	7.7	20.0	17.9	8.0	11.1	7.1	30.0	11.5	11.4
LLaMA3-instruct	12.0	0.0	4.3	6.7	3.8	0.0	3.6	0.0	3.7	14.3	0.0	0.0	4.2
MagiCoder-6.7b	20.0	0.0	8.7	0.0	0.0	6.7	7.1	0.0	7.4	7.1	20.0	0.0	5.7
StarCoder-15b-instruct	12.0	0.0	0.0	6.7	0.0	0.0	0.0	4.0	7.4	0.0	10.0	0.0	3.4
WizardCoder-15b	8.0	0.0	0.0	3.3	0.0	6.7	0.0	0.0	3.7	0.0	20.0	0.0	2.7
Gemini Pro	16.0	0.0	4.3	3.3	0.0	0.0	3.6	0.0	14.8	0.0	20.0	7.7	5.7
GPT-3.5 (gpt-3.5-turbo-1106)	28.0	6.9	4.3	6.7	7.7	13.3	10.7	4.0	18.5	14.3	20.0	7.7	11.0
GPT-4 (gpt-4-1106-preview)	28.0	6.9	13.0	10.0	7.7	13.3	17.9	16.0	29.6	21.4	40.0	26.9	17.9
GPT-4V (gpt-4-vision-preview)	40.0	10.3	17.4	10.0	7.7	26.7	7.1	12.0	22.2	21.4	50.0	23.1	18.3
GPT-4o (gpt-4o-2024-05-13)	32.0	6.9	8.7	3.3	11.5	20.0	10.7	16.0	18.5	7.1	40.0	15.4	14.8
Vision + Language Inputs													
LLaVA-1.5-7B	12.0	0.0	0.0	0.0	0.0	6.7	0.0	0.0	0.0	0.0	0.0	0.0	1.5
LLaVA-1.5-13B	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.7	0.0	0.0	0.0	1.1
QWEN-VL	8.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8
Qwen2VL-2B	12.0	0.0	0.0	0.0	3.8	0.0	0.0	0.0	3.8	0.0	20.0	0.0	2.7
Qwen2VL-7B	16.0	3.4	0.0	6.7	0.0	13.3	3.6	0.0	3.8	0.0	30.0	3.8	5.8
Qwen2VL-72B	24.0	0.0	4.3	6.7	0.0	20.0	11.1	4.3	7.7	7.1	30.0	15.4	10.0
Gemini Pro Vision	12.5	0.0	4.3	0.0	3.8	6.7	7.1	0.0	7.4	0.0	30.0	0.0	5.0
GPT-4V (gpt-4-vision-preview)	40.0	6.9	13.0	13.8	3.8	21.4	24.0	9.5	25.9	21.4	40.0	20.8	19.4
GPT-4o (gpt-4o-2024-05-13)	36.0	6.9	8.7	3.4	7.7	21.4	24.0	14.3	25.9	14.3	50.0	8.3	17.0

Table 2: Pass@1 (%) results grouped by different image categories. The dashed lines separate open-source models (above) and proprietary models (below).

4 Data Analysis

In this section, we undertake a comprehensive exploration of MMCode, introducing its and statistical attributes to provide a nuanced understanding of MMCode.

Problem Length. The diversity of data sources incorporated into MMCode results in significant variance in problem length, as can be seen in Figure 2a. The mean length of the questions reaches 2,256 characters, with the 25th, 50th, and 75th percentile at 1,516, 2,127, and 2,791. This can be ascribed to the distinct style and difficulty of the questions presented in MMCode. Certain questions articulate the instructions succinctly and directly, whereas others elaborate on the contextual background of the problem in detail.

Image Count per Problem. A notable characteristic that differentiates MMCode from previous datasets is its inclusion of multiple images per question. On average, each question is associated with 1.87 images, with the 25th percentile having 1 image and the 75th percentile having 2 images. These figures are interleaved with the text contents, and the understanding of them frequently depends on their order, posing great difficulty to the models.

Image Position. As Figure 3b illustrates, the images in the problems of MMCode can appear at any position in the text, but concentrate at the tail. This is because many images are drawn to intuitively depict and explain sample inputs and outputs, which are mostly located at the end of the text.

Image Type. Figure 3a illustrates the portion of the categories of images following the classification criteria introduced in Section 3.2. `Graph`, `Math` and `2D Geometry` form the majority comprising more than half of the dataset, taking up 20.9% 17.9%, and 15.3% respectively. Miscellaneous images classified under `Others` account for roughly one-tenth of the dataset, representing a high level of heterogeneity. `Tree` follows up with 9.9%. The remaining groups sum up to approximately a quarter, demonstrating the diversity of MMCode.

Model	Task Type												Average
	Linear	Tree	Graph	2D	3D	Chessboard	Map	Math	Patterns	Table	Pseudocode	Others	
Gemini Pro	16.0	0.0	0.0	6.7	0.0	6.7	3.6	0.0	11.1	7.1	20.0	7.7	6.1
GPT-4 (gpt-4-vision-preview)	32.0	10.3	17.4	6.7	3.8	33.3	25.0	12.0	33.3	21.4	40.0	19.2	19.0

Table 3: The performance of closed-source models with Image Replacement. Results are measured by Pass@1 (%).

Model	Task Type												Average
	Linear	Tree	Graph	2D	3D	Chessboard	Map	Math	Patterns	Table	Pseudocode	Others	
Gemini Pro Vision	8.0	0.0	0.0	6.7	0.0	13.3	3.7	0.0	3.7	0.0	20.0	0.0	3.8
GPT-4V (gpt-4-vision-preview)	28.0	6.9	8.7	6.9	7.7	7.1	28.0	9.5	33.3	14.3	40.0	12.5	16.6

Table 4: The performance of closed-source models with Captioning Chain of Thought. Results are measured by Pass@1 (%).

5 EXPERIMENTS

In this section, we benchmark several Language-Only models and Vision-Language models with MMCode. A comparative analysis of the experimental results for these models is conducted, providing a thorough examination of their capabilities.

5.1 EXPERIMENTAL SETUP

We evaluate the models by prompting with fixed templates (see Appendix C) using greedy decoding and extracting their generated codes, which are executed by the testing framework to check their correctness. Pass@1 (Chen et al., 2021) is reported. The following three setups are compared:

Language-Only Models. We evaluate several powerful and Language-Only models, including GPT-3.5 (OpenAI, 2022), GPT-4 OpenAI (2023a), and Gemini Pro Team Gemini et al. (2023). The images in the problem statement are removed in this setup.

Large Multi-modal Models. Some popular LMMs are selected as testees on MMCode. This includes proprietary models such as Gemini Pro Vision Team Gemini et al. (2023), GPT-4V OpenAI (2023b). Additionally, open-source models such as the LLaVA series (Liu et al., 2024a), QWEN-VL Bai et al. (2023) and Qwen2-VL (Wang et al., 2024) are assessed to track the advancements of the more accessible LMMs. The first image in the problem is kept for models that are not trained to support multiple-image inputs, i.e. the LLaVA series. For fairer comparison, text-only inputs performance of these models are also reported whenever applicable.

Caption-augmented Models. We investigate whether the inclusion of captions can help the model better understand the image contexts. In our early experiments, the open-source models yielded inferior captions, frequently containing hallucinations and failing to interpret the abstract meaning of the images. Thus, we only benchmark the proprietary models. We explored two methods of leveraging the captions: **(a) Image Replacement**, where the image slots are replaced by the captions. **(b) Captioning Chain of Thought**, where we explicitly prompt the models to generate captions for the images first, and then work out the questions, resembling the Chain of Thought prompting (Wei et al., 2022).

6 EVALUATION RESULTS

6.1 RESULTS AND FINDINGS

MMCode poses a great challenge to all models. As Table 2 depicts, all models except for the GPT family scored a Pass@1 rate under 10%, whereas the best of the models tested, GPT-4V, yielded a mere 19.4% when equipped with all image contexts. Test case pass rates, as a fine-grained

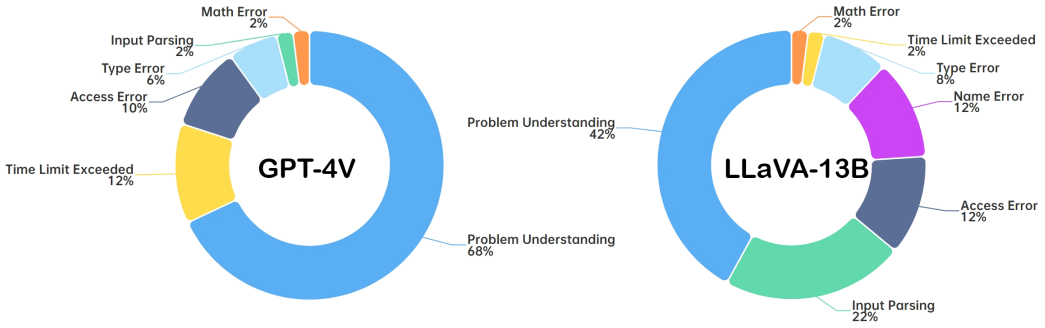


Figure 4: Error distribution of GPT-4V and LLaVA-13B on a sampled subset of 50 problems.

measure, show a similar trend in Tabel 7 in Appendix E.1. This renders MMCode a challenging benchmark for the development of coding LMMs.

Proprietary models take a huge lead on MMCode. The GPTs yield superior results, leaving a huge gap between other models. Gemini Pro, though underperforms the GPTs, beats all tested open-source models. Many open-source models generally demonstrate the inability to solve the questions with negligible pass rates of around 1% and a majority of zeros in many categories. Notably, Qwen2-VL achieves 10%, but is still behind the GPT family. A plausible reason is that these open-source LMMs are not trained on such reasoning-heavy code generation tasks nor to understand abstract diagrams. The coding ability is only inherited from the base LLMs, but can be impaired due to catastrophic forgetting (Luo et al., 2023a).

Visual context helps, but requires advanced comprehending capability. Interestingly, unlike previous works such as OlympiadBench (He et al., 2024) where the text-only inputs beat multi-modal inputs, the best performance of all experiments is produced by GPT-4V with vision contexts. The observation confirms that the images contain critical information that can be mined to assist problem-solving. However, Gemini Pro Vision often fails to leverage the hints from the images, and the performance drops compared with the language-only Gemini Pro.

GPT-4V performs better than GPT-4 counterparts on less visually-cluttered image types. Comparing GPT-4V with multi-modal input to text-only GPT-4 and GPT-4V on problems with different types of images, it is observed that improvements are achieved on simpler image types, e.g. `Linear Data Structure`, `Tree`, `2D`, and `Map`. On other visually cluttered categories such as `Graph`, `Chessboard` and `Patterns`, the addition of images hurts the performance. GPT-4V also produces worse results on `Others`, which consists of miscellaneous cases including complex annotations, which are challenging for the model to interpret.

Image replacement with generated captions helps, but Captioning CoT does not. Table 3 and 4 lists the results with the two caption prompting strategies. The vision models can generate informative captions (though often inaccurate; see case studies in Section 6.3.1), as the text-only models all improve from their caption-free settings using the Image Replacement strategy. However, interestingly, all LMMs prompted with Captioning Chain of Thought suffer a decline in the pass rates. A possible explanation is that the captions lengthen the context, while the images still remain in the context, causing trouble for the models to determine where to attend.

6.2 Error Analysis

To facilitate the understanding of the models' bottleneck in solving MMCode problems, an identical subset of 50 questions are randomly selected from the failure cases of GPT-4V and LLaVA-13B and reviewed. Figure 4 presents the results. The majority of errors arise in the wrong understanding of the problems, where executable codes are generated but with wrong results. GPT-4V produces fewer runtime errors than LLaVA-13B, including Access Errors (e.g. IndexError, KeyError), Type Errors (e.g. calling non-existing methods of an object), and Math Errors (e.g. ZeroDivisionError).

Notably, LLaVA-13B makes many elementary mistakes such as wrong Input Parsing and NameError (e.g. usage of variables undefined or defined afterward). These errors prevent the programs from producing outputs that can be checked, resulting in a decrease in Problem Understanding errors.

6.3 Case Study

6.3.1 Caption Quality

Figures 5 to 16 in Appendix F showcase the captions generated by GPT-4V and Gemini Pro Vision of 12 images from different categories. Generally, GPT-4V generates more accurate and more insightful captions than Gemini Pro Vision. However, both models can hallucinate the images, especially on visually complex elements such as `Graph` (Figure 10). On the easier image of a `Tree` with fewer nodes and edges, both models produce correct explanations (Figure 9).

6.3.2 Code Quality

We examined solutions generated by GPT-4V in section G in the Appendix. Apart from complex logic errors and inefficient implementations (Section G.1, it still makes trivial mistakes, e.g. naming variables after built-in functions (Section G.2), reading inputs when the problem does not ask it to (Section G.3).

7 Conclusion

In this paper, we present MMCode, the first multi-modal coding dataset for evaluating algorithmic problem-solving skills in image-text interwoven contexts. We benchmarked a range of state-of-the-art LLMs and LMMs on MMCode and provide a detailed analysis. Despite their advanced capabilities, these models demonstrate a significant challenge in leveraging visual contexts for code generation. We believe that MMCode will catalyze further research and innovation, paving the way for the creation of AI systems capable of handling sophisticated visual and textual reasoning in programming and beyond.

References

Ritu Agarwal and Atish P Sinha. Object-oriented modeling with uml: a study of developers' perceptions. *Communications of the ACM*, 46(9):248–256, 2003.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL `https://arxiv.org/pdf/2107.03374.pdf`.

Yew Ken Chia, Vernon Toh Yan Han, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. Puzzlevqa: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. *arXiv preprint arXiv:2403.13315*, 2024.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022. URL https://arxiv.org/pdf/2204.02311.pdf.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938*, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Kaixin Li, Qisheng Hu, James Zhao, Hui Chen, Yuxi Xie, Tiedong Liu, Michael Shieh, and Junxian He. InstructCoder: Instruction tuning large language models for code editing. In Xiyan Fu and Eve Fleisig (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 50–70, Bangkok, Thailand, August 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-srw.6.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023a.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023b.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097, 2022.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024a.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024b.

Yuan Liu, Haodong Duan, Yuanhan Zhang, Bo Li, Songyang Zhang, Wangbo Zhao, Yike Yuan, Jiaqi Wang, Conghui He, Ziwei Liu, et al. Mmbench: Is your multi-modal model an all-around player? *arXiv preprint arXiv:2307.06281*, 2023.

Pan Lu, Swaroop Mishra, Tanglin Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. *arXiv preprint arXiv:2310.02255*, 2023.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023a.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*, 2023b.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=iaYcJKpY2B_.

Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pp. 35–46, 2000.

OpenAI. Introducing ChatGPT. https://openai.com/blog/chatgpt, 2022.

OpenAI. Gpt-4 technical report. https://arxiv.org/pdf/2303.08774, 2023a.

OpenAI. Gpt-4v(ision) system card. https://cdn.openai.com/papers/GPTV_System_Card.pdf, 2023b. Accessed: 2024-02-03.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022. URL https://arxiv.org/pdf/2211.05100.pdf.

Team Gemini, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Runchu Tian, Yining Ye, Yujia Qin, Xin Cong, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Debugbench: Evaluating debugging capability of large language models. *arXiv preprint arXiv:2401.04621*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a. URL https://arxiv.org/pdf/2302.13971.pdf.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Weixiang Yan, Haitian Liu, Yunkun Wang, Yunzhe Li, Qian Chen, Wen Wang, Tingyu Lin, Weishan Zhao, Li Zhu, Shuiguang Deng, et al. Codescope: An execution-based multilingual multitask multidimensional benchmark for evaluating llms on code understanding and generation. *arXiv preprint arXiv:2311.08588*, 2023a.

Weixiang Yan, Yuchen Tian, Yunzhe Li, Qian Chen, and Wen Wang. Codetransocean: A comprehensive multilingual benchmark for code translation. *arXiv preprint arXiv:2310.04951*, 2023b.

Qinghao Ye, Haiyang Xu, Guohai Xu, Jiabo Ye, Ming Yan, Yiyang Zhou, Junyang Wang, Anwen Hu, Pengcheng Shi, Yaya Shi, et al. mplug-owl: Modularization empowers large language models with multimodality. *arXiv preprint arXiv:2304.14178*, 2023.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. *arXiv preprint arXiv:2311.16502*, 2023.

Qinkai Zheng, Xiao Xia, Xu Zou, Yuxiao Dong, Shan Wang, Yufei Xue, Lei Shen, Zihan Wang, Andi Wang, Yang Li, et al. Codegeex: A pre-trained model for code generation with multilingual benchmarking on humaneval-x. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5673–5684, 2023.

A Data Sources

The data in MMCode is collected from the following websites[3]:

- AtCoder: `https://atcoder.jp`
- Aizu: `https://judge.u-aizu.ac.jp/onlinejudge/`
- CodeChef: `https://www.codechef.com`
- CodeForces: `https://codeforces.com`[4]
- CodeWars: `https://www.codewars.com`
- Project Euler: `https://projecteuler.net`[5]
- GeeksForGeeks: `https://www.geeksforgeeks.org`
- HackerRank: `https://www.hackerrank.com`
- LeetCode: `https://leetcode.com`
- Open Kattis: `https://open.kattis.com/`

The statistical data of the quantity of questions and images retained from each platform can be found in Table 5. In total, MMCode comprises 3548 programming questions. Among the sources, CodeForces and Aizu contribute the most questions and images in MMCode.

Platform	**# Questions**	**# Images**
AtCoder	139	234
Aizu	694	1349
CodeChef	90	134
CodeForces	1941	3837
Project Euler	132	176
GeeksForGeeks	128	192
Open Kattis	145	195
HackerRank	169	316
CodeWars	33	46
LeetCode	77	141
Total	3548	6620

Table 5: The data sources of MMCode and the number of questions and images from each source.

[3]The license is Apache 2.0 from TACO unless specifically stated.

[4]No license found.

[5]CC BY-NC-SA 4.0.

B DEFINITION OF IMAGE CATEGORIES

- **Linear Data Structure:** This category includes diagrams that illustrate sequential data structures such as arrays, linked lists, and queues, where data elements are arranged in a linear order.
- **Tree:** Dedicated to the data structure of trees, focusing on hierarchical representations.
- **Graph:** Includes visuals of graph data structures where nodes are connected by edges, e.g., directed and undirected graphs. If the problem description is about graphs but the image depicts a tree (e.g. after pruning), it is still classified under this category.
- **2D Geometry:** Focuses on two-dimensional geometric shapes and properties, including points, lines, polygons, etc., emphasizing spatial relationships in a plane.
- **3D Geometry:** Comprises images that depict three-dimensional objects and structures, such as 3D coordinate systems, orthographic projections, and nets of cubes, showcasing the complexity and characteristics of three-dimensional space.
- **Chessboard:** This category includes images showing a chessboard, where the model is expected to solve a problem with respect to some rules of playing.
- **Map:** Pertains to images displaying maps that show positions. If the image features a graph functioning as a map, it falls into this category.
- **Patterns:** Covers images that involve recognizing, generating, or solving puzzles and patterns, which could be numerical, geometrical, or based on character arrangements.
- **Table:** Dedicated to tabular data presentations.
- **Pseudocode:** Includes images that contain pseudocode or simplified code, posing challenges to the dense OCR ability of the models.
- **Others:** Serves as a miscellaneous category for visual content that does not fit into other categories, e.g. bar graphs, pie charts, and Venn diagrams.

C PROMPTS

All prompts used in this work are listed in Table 6.

Type	Prompt
Problem Solving	**System Prompt (if applicable):** You are a professional programming contester trying to solve algorithmic problems. The problems come with a description and some images, and you should write a Python solution. **User Prompt:** You are required to solve a programming problem. Please enclose your code inside a ```` ```python``` ```` block. Do not write a main() function. If a Call-Based format is used, return the result in an appropriate place instead of printing it. {*problem statement*}
Caption Generation	Please describe and explain the images in the programming problem. The readers will not be able to see the image, so make sure you include all important information for solving the problem. Please enclose your explanations inside ```` ```plain``` ```` blocks, one for each image. Your output should look like: Caption: ```` ```plain ```` The image shows...```` ``` ```` {*problem statement with only one image*}

Table 6: The prompts used in this study.

D Data Samples

D.1 An example of a question with a pseudo code image

Bob is playing a game named "Walk on Matrix". In this game, player is given an $n \times m$ matrix $A = (a_{i,j})$, i.e. the element in the i-th row in the j-th column is $a_{i,j}$. Initially, player is located at position $(1, 1)$ with score $a_{1,1}$. To reach the goal, position (n, m), player can move right or down, i.e. move from (x, y) to $(x, y + 1)$ or $(x + 1, y)$, as long as player is still on the matrix. However, each move changes player's score to the bitwise AND of the current score and the value at the position he moves to. Bob can't wait to find out the maximum score he can get using the tool he recently learnt — dynamic programming. Here is his algorithm for this problem:

Algorithm 1 Bob's Dynamic Programming Algorithm

Input: matrix $A = (a_{i,j})_{n \times m}$;
Output: maximum score S after reaching the goal;

initialize $dp_{i,j} \leftarrow 0$, for all $0 \le i \le n$, $0 \le j \le m$ except $dp_{0,1} \leftarrow a_{1,1}$
for $i = 1 \to n$ **do**
  for $j = 1 \to m$ **do**
    $dp_{i,j} \leftarrow \max(dp_{i-1,j} \& a_{i,j}, dp_{i,j-1} \& a_{i,j})$
  end for
end for
$S \leftarrow dp_{n,m}$

However, he suddenly realizes that the algorithm above fails to output the maximum score for some matrix A. Thus, for any given non-negative integer k, he wants to find out an $n \times m$ matrix $A = (a_{i,j})$ such that:

- $1 \leq n, m \leq 500$ (as Bob hates large matrices);
- $0 \leq a_{i,j} \leq 3 \cdot 10^5$ for all $1 \leq i \leq n, 1 \leq j \leq m$ (as Bob hates large numbers);
- the difference between the maximum score he can get and the output of his algorithm is exactly k. It can be shown that for any given integer k such that $0 \leq k \leq 10^5$, there exists a matrix satisfying the above constraints.

Input
The only line of the input contains one single integer k ($0 \leq k \leq 10^5$).
Output
Output two integers n, m ($1 \leq n, m \leq 500$) in the first line, representing the size of the matrix. Then output n lines with m integers in each line, $a_{i,j}$ in the $(i+1)$-th row, j-th column.
Examples
Input

```
0
```

Output

```
1 1
300000
```

Input

```
1
```

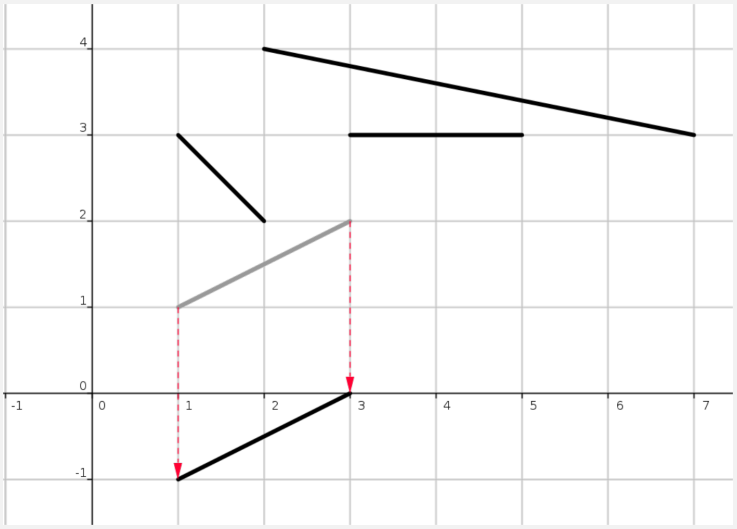
Output

```
3 4
7 3 3 1
4 8 3 6
7 7 7 3
```

Note
In the first example, the maximum score Bob can achieve is 300000, while the output of his algorithm is 300000.

D.2 An example of a question with a 2D geometry image

You have most definitely heard the legend of King Arthur and the Knights of the Round Table. Almost all versions of this story proudly point out that the roundness of the Round Table is closely related to Arthur's belief of equality among the Knights. That is a lie! In fact, Arthur's choice of table is conditioned by his childhood traumas. In fact, Arthur was forced to clean up quadratic tables from a young age after a tournament in pick-up sticks had been played on them. After the tournament, typically there would be a bunch of sticks on the table that do not touch each other. In the spirit of the game, the organizers issued strict regulations for the table cleaners. More precisely, the sticks on the table need to be removed one by one in a way that the cleaners pull them in the shortest way towards the edge of the table closest to where they are currently sitting. They also mustn't rotate or touch the other sticks while doing this (not even in the edge points). In this task, we will represent the table in the coordinate system with a square that has opposite points in the coordinates $(0, 0)$ and $(10\,000, 10\,000)$, whereas the sticks will be represented with straight line segments that lie within that square. We will assume that Arthur is sitting at the edge of the table lying on the x-axis. Then the movement of the stick comes down to translating the line segment along the shortest path towards the x-axis until the stick falls off the table (as shown in the image). It is your task to help Arthur determine the order of stick movements that meets the requirements from the previous paragraph.



Input
The first line of input contains the integer N $(1 \leq N \leq 5\,000)$, the number of sticks on the table. Each of the following N lines contains four integers x_1, y_1, x_2, y_2 $(0 \leq x_1, y_1, x_2, y_2 \leq 10\,000)$ that denote the edge points of a stick.

Output
The first and only line of output must contain space-separated stick labels in the order which they need to be taken off the table. A stick's label corresponds to its position in the input sequence. If there are multiple possible solutions, output any of them.

Sample Input 1

```
4
1 3 2 2
1 1 3 2
2 4 7 3
3 3 5 3
```

Sample Output 1

```
2 4 1 3
```

Sample Input 2

```
4
0 0 1 1
1 2 0 3
2 2 3 3
4 0 3 1
```

Sample Output 2

```
4 3 1 2
```

Sample Input 3

```
3
4 6 5 5
2 1 15 1
3 2 8 7
```

Sample Output 3

```
2 3 1
```

D.3 AN EXAMPLE OF A QUESTION WITH A 3D GEOMETRY IMAGE

In AD 3456, the earth is too small for hundreds of billions of people to live in peace. Interstellar Colonization Project with Cubes (ICPC) is a project that tries to move people on the earth to space colonies to ameliorate the problem. ICPC obtained funding from governments and manufactured space colonies very quickly and at low cost using prefabricated cubic blocks.

The largest colony looks like a Rubik's cube. It consists of $3 \times 3 \times 3$ cubic blocks (Figure J.1A). Smaller colonies miss some of the blocks in the largest colony.

When we manufacture a colony with multiple cubic blocks, we begin with a single block. Then we iteratively glue a next block to existing blocks in a way that faces of them match exactly. Every pair of touched faces is glued.

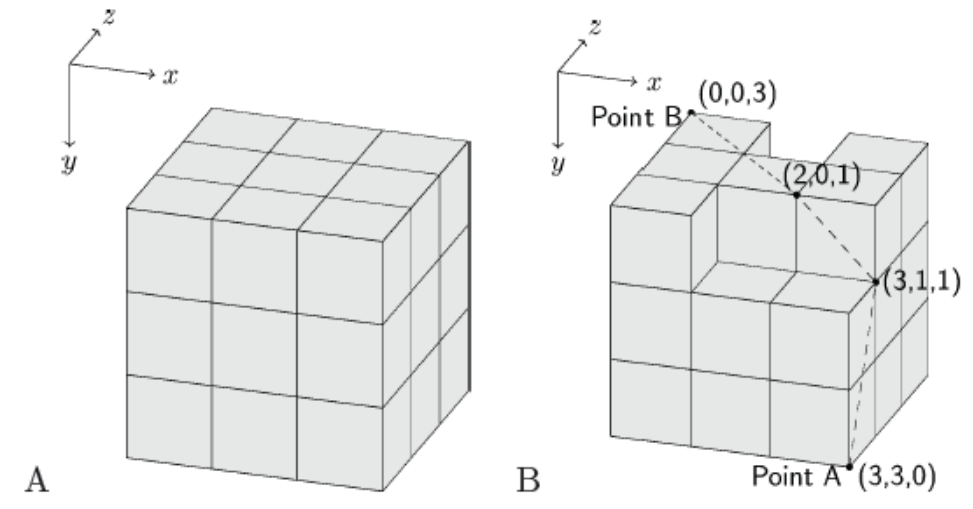


Figure J.1: Largest colony

However, just before the first launch, we found a design flaw with the colonies. We need to add a cable to connect two points on the surface of each colony, but we cannot change the inside of the prefabricated blocks in a short time. Therefore we decided to attach a cable on the surface of each colony. If a part of the cable is not on the surface, it would be sheared off during the launch, so we have to put the whole cable on the surface. We would like to minimize the lengths of the cables due to budget constraints. The dashed line in Figure J.1B is such an example.

Input

The input contains a series of datasets. Each dataset describes a single colony and the pair of the points for the colony in the following format.

$x_1 y_1 z_1 x_2 y_2 z_2$
$b_{0,0,0} b_{1,0,0} b_{2,0,0}$
$b_{0,1,0} b_{1,1,0} b_{2,1,0}$
$b_{0,2,0} b_{1,2,0} b_{2,2,0}$
$b_{0,0,1} b_{1,0,1} b_{2,0,1}$
$b_{0,1,1} b_{1,1,1} b_{2,1,1}$
$b_{0,2,1} b_{1,2,1} b_{2,2,1}$
$b_{0,0,2} b_{1,0,2} b_{2,0,2}$
$b_{0,1,2} b_{1,1,2} b_{2,1,2}$
$b_{0,2,2} b_{1,2,2} b_{2,2,2}$

(x_1, y_1, z_1) and (x_2, y_2, z_2) are the two distinct points on the surface of the colony, where $x_1, x_2, y_1, y_2, z_1, z_2$ are integers that satisfy $0 \leq x_1, x_2, y_1, y_2, z_1, z_2 \leq 3$. $b_{i,j,k}$ is '#' when there is a cubic block whose two diagonal vertices are (i, j, k) and $(i+1, j+1, k+1)$, and $b_{i,j,k}$ is '.' if there is no block. Figure J.1A corresponds to the first dataset in the sample input, whereas Figure J.1B corresponds to the second. A cable can pass through a zero-width

gap between two blocks if they are touching only on their vertices or edges. In Figure J.2A, which is the third dataset in the sample input, the shortest cable goes from the point A (0, 0, 2) to the point B (2, 2, 2), passing through (1, 1, 2), which is shared by six blocks. Similarly, in Figure J.2B (the fourth dataset in the sample input), the shortest cable goes through the gap between two blocks not glued directly. When two blocks share only a single vertex, you can put a cable through the vertex (Figure J.2C; the fifth dataset in the sample input).
You can assume that there is no colony consisting of all $3 \times 3 \times 3$ cubes but the center cube. Six zeros terminate the input.

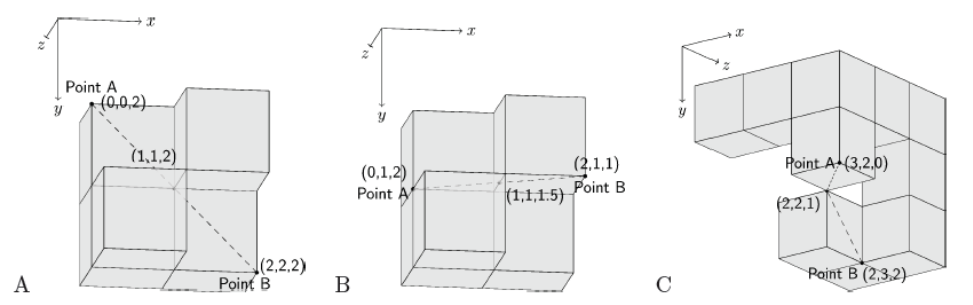


Figure J.2: Largest colony

Output
For each dataset, output a line containing the length of the shortest cable that connects the two given points. We accept errors less than 0.0001. You can assume that given two points can be connected by a cable.
Sample Input

```
0 0 0 3 3 3
###
###
###
###
###
###
###
###
###
3 3 0 0 0 3
#..
###
###
###
###
###
#.#
###
###
0 0 2 2 2 2
...
...
...
.#.
#..
...
##.
##.
...
0 1 2 2 1 1
...
```

```
...
...
.#.
#..
...
##.
##.
...
3 2 0 2 3 2
###
..#
...
..#
...
.#.
..#
..#
.##
0 0 0 0 0 0
```

Output for the Sample Input

```
6.70820393249936941515
6.47870866461907457534
2.82842712474619029095
2.23606797749978980505
2.82842712474619029095
```

D.4 AN EXAMPLE OF A QUESTION WITH A TREE IMAGE

Let's define the Eulerian traversal of a tree (a connected undirected graph without cycles) as follows: consider a depth-first search algorithm which traverses vertices of the tree and enumerates them in the order of visiting (only the first visit of each vertex counts). This function starts from the vertex number 1 and then recursively runs from all vertices which are connected with an edge with the current vertex and are not yet visited in increasing numbers order. Formally, you can describe this function using the following pseudocode:

```
next_id = 1
id = array of length n filled with -1
visited = array of length n filled with false

function dfs(v):
    visited[v] = true
    id[v] = next_id
    next_id += 1
    for to in neighbors of v in increasing order:
        if not visited[to]:
            dfs(to)
```

You are given a weighted tree, the vertices of which were enumerated with integers from 1 to n using the algorithm described above.
A leaf is a vertex of the tree which is connected with only one other vertex. In the tree given to you, the vertex 1 is not a leaf. The distance between two vertices in the tree is the sum of weights of the edges on the simple path between them.
You have to answer q queries of the following type: given integers v, l and r, find the shortest distance from vertex v to one of the leaves with indices from l to r inclusive.
Input
The first line contains two integers n and q ($3 \le n \le 500\,000$, $1 \le q \le 500\,000$) — the number of vertices in the tree and the number of queries, respectively.

The $(i-1)$-th of the following $n-1$ lines contains two integers p_i and w_i ($1 \le p_i < i$, $1 \le w_i \le 10^9$), denoting an edge between vertices p_i and i with the weight w_i.
It's guaranteed that the given edges form a tree and the vertices are enumerated in the Eulerian traversal order and that the vertex with index 1 is not a leaf.
The next q lines describe the queries. Each of them contains three integers v_i, l_i, r_i ($1 \le v_i \le n$, $1 \le l_i \le r_i \le n$), describing the parameters of the query. It is guaranteed that there is at least one leaf with index x such that $l_i \le x \le r_i$.

Output

Output q integers — the answers for the queries in the order they are given in the input.

Examples

Input

```
5 3
1 10
1 1
3 2
3 3
1 1 5
5 4 5
4 1 2
```

Output

```
3
0
13
```

Input

```
5 3
1 1000000000
2 1000000000
1 1000000000
1 1000000000
3 4 5
2 1 5
2 4 5
```

Output

```
3000000000
1000000000
2000000000
```

Input

```
11 8
1 7
2 1
1 20
1 2
5 6
6 2
6 3
5 1
9 10
9 11
5 1 11
1 1 4
9 4 8
6 1 4
9 7 11
9 10 11
```

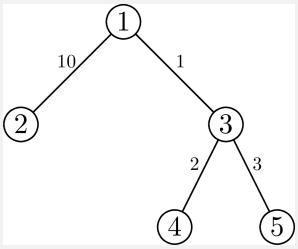
```
8 1 11
11 4 5
```

Output

```
8
8
9
16
9
10
0
34
```

Note

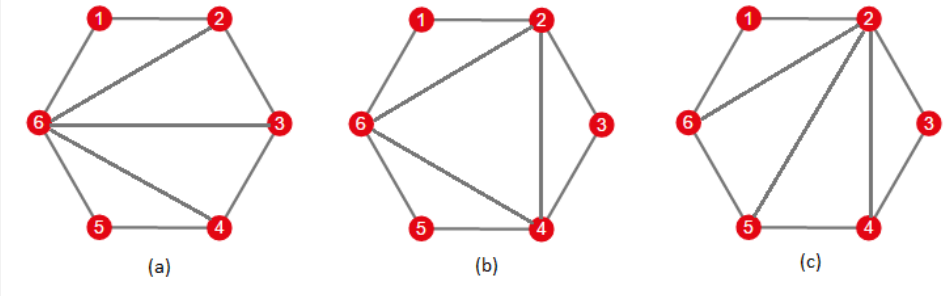
In the first example, the tree looks like this:



In the first query, the nearest leaf for the vertex 1 is vertex 4 with distance 3. In the second query, the nearest leaf for vertex 5 is vertex 5 with distance 0. In the third query, the nearest leaf for vertex 4 is vertex 4; however, it is not inside interval $[1, 2]$ of the query. The only leaf in interval $[1, 2]$ is vertex 2 with distance 13 from vertex 4.

D.5 AN EXAMPLE OF A QUESTION WITH A GRAPH IMAGE

Fox Ciel just designed a puzzle game called "Polygon"! It is played using triangulations of a regular n-edge polygon. The goal is to transform one triangulation to another by some tricky rules.



Triangulation of an n-edge polygon is a set of $n - 3$ diagonals satisfying the condition that no two diagonals share a common internal point.

For example, the initial state of the game may look like (a) in the figure. And your goal may look like (c). In each step, you can choose a diagonal inside the polygon (but not one of the edges of the polygon) and flip this diagonal.

Suppose you are going to flip a diagonal $a - b$. There always exist two triangles sharing $a - b$ as a side, let's denote them as $a - b - c$ and $a - b - d$. As a result of this operation, the diagonal $a - b$ is replaced by a diagonal $c - d$. It can be easily proven that after the flip operation, the resulting set of diagonals is still a triangulation of the polygon.

So in order to solve the above case, you may first flip diagonal $6 - 3$, it will be replaced by diagonal $2 - 4$. Then you flip diagonal $6 - 4$ and get figure (c) as a result.
Ciel just proved that for any starting and destination triangulations, this game has a solution. She wants you to solve it in no more than $20,000$ steps for any puzzle satisfying $n \leq 1000$.
Input
The first line contains an integer n ($4 \leq n \leq 1000$), the number of edges of the regular polygon.
Then follows two groups of $(n - 3)$ lines describing the original triangulation and goal triangulation.
Description of each triangulation consists of $(n - 3)$ lines. Each line contains 2 integers a_i and b_i ($1 \leq a_i, b_i \leq n$), describing a diagonal $a_i - b_i$.
It is guaranteed that both original and goal triangulations are correct (i.e., no two diagonals share a common internal point in both of these triangulations).
Output
First, output an integer k ($0 \leq k \leq 20,000$): the number of steps.
Then output k lines, each containing 2 integers a_i and b_i: the endpoints of a diagonal you are going to flip at step i. You may output a_i and b_i in any order.
If there are several possible solutions, output any of them.
Examples
Input

```
41 32 4
```

Output

```
11 3
```

Input

```
62 63 64 66 25 24 2
```

Output

```
26 36 4
```

Input

```
87 12 77 36 34 66 16 26 36 46 8
```

Output

```
37 37 27 1
```

Note
Sample test 2 is discussed above and shown on the picture.

D.6 AN EXAMPLE OF A QUESTION WITH AN UNRELATED IMAGE

Bash got tired on his journey to become the greatest Pokemon master. So he decides to take a break and play with functions.
Bash defines a function $f_0(n)$, which denotes the number of ways of factoring n into two factors p and q such that $\gcd(p, q) = 1$. In other words, $f_0(n)$ is the number of ordered pairs of positive integers (p, q) such that $p \cdot q = n$ and $\gcd(p, q) = 1$.
But Bash felt that it was too easy to calculate this function. So he defined a series of functions, where f_{r+1} is defined as:

$$f_{r+1}(n) = \sum_{u \cdot v = n} \frac{f_r(u) + f_r(v)}{2},$$

Where (u, v) is any ordered pair of positive integers, they need not to be co-prime.
Now Bash wants to know the value of $f_r(n)$ for different r and n. Since the value could be huge, he would like to know the value modulo $10^9 + 7$. Help him!
Input

The first line contains an integer q ($1 \leq q \leq 10^6$) — the number of values Bash wants to know.
Each of the next q lines contains two integers r and n ($0 \leq r \leq 10^6$, $1 \leq n \leq 10^6$), which denote Bash wants to know the value $f_r(n)$.
Output
Print q integers. For each pair of r and n given, print $f_r(n)$ modulo $10^9 + 7$ on a separate line.
Example
Input

```
50
301 253
652 54
48
```

Output

```
85254630
```

D.7 An example of a question with a table image

At a regular competition, Vladik and Valera won a and b candies respectively. Vladik offered 1 his candy to Valera. After that, Valera gave Vladik 2 his candies, so that no one thought that he was less generous. Vladik for the same reason gave 3 candies to Valera in the next turn.
More formally, the guys take turns giving each other one candy more than they received in the previous turn.
This continued until the moment when one of them couldn't give the right amount of candy. Candies, which guys got from each other, they don't consider as their own. You need to know who is the first who can't give the right amount of candy.
Input
A single line of input data contains two space-separated integers a, b ($1 \leq a, b \leq 10^9$) — the number of Vladik and Valera candies respectively.
Output
Print a single line "Vladik" if Vladik is the first who can't give the right amount of candy, or "Valera" otherwise.
Examples
Input

```
1 1
```

Output

```
Valera
```

Input

```
7 6
```

Output

```
Vladik
```

Note
Illustration for the first test case:

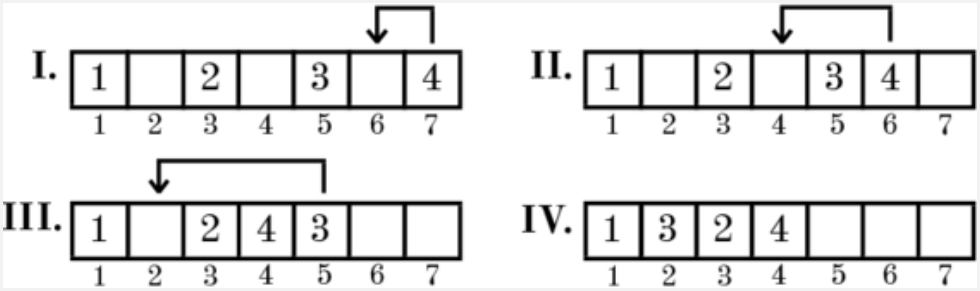
step	Vladik	Valera
0	1	1
1	0	1

Illustration for the second test case:

step	Vladik	Valera
0	7	6
1	6	6
2	6	4
3	3	4
4	3	0

D.8 AN EXAMPLE OF A QUESTION WITH AN LINEAR DATA STRUCTURE IMAGE

Dima is a beginner programmer. During his working process, he regularly has to repeat the following operation again and again: to remove every second element from the array. One day he has been bored with easy solutions of this problem, and he has come up with the following extravagant algorithm.
Let's consider that initially, the array contains n numbers from 1 to n and the number i is located in the cell with the index $2i - 1$ (Indices are numbered starting from one) and other cells of the array are empty. Each step Dima selects a non-empty array cell with the maximum index and moves the number written in it to the nearest empty cell to the left of the selected one. The process continues until all n numbers will appear in the first n cells of the array. For example if $n = 4$, the array is changing as follows:



You have to write a program that allows you to determine what number will be in the cell with index x $(1 \leq x \leq n)$ after Dima's algorithm finishes.

Input

The first line contains two integers n and q $(1 \leq n \leq 10^{18}, 1 \leq q \leq 200,000)$, the number of elements in the array and the number of queries for which it is needed to find the answer. Next q lines contain integers x_i $(1 \leq x_i \leq n)$, the indices of cells for which it is necessary to output their content after Dima's algorithm finishes.

Output

For each of q queries, output one integer number, the value that will appear in the corresponding array cell after Dima's algorithm finishes.

Examples

Input

```
4 3234
```

Output

```
324
```

Input

```
13 410548
```

Output

```
13389
```

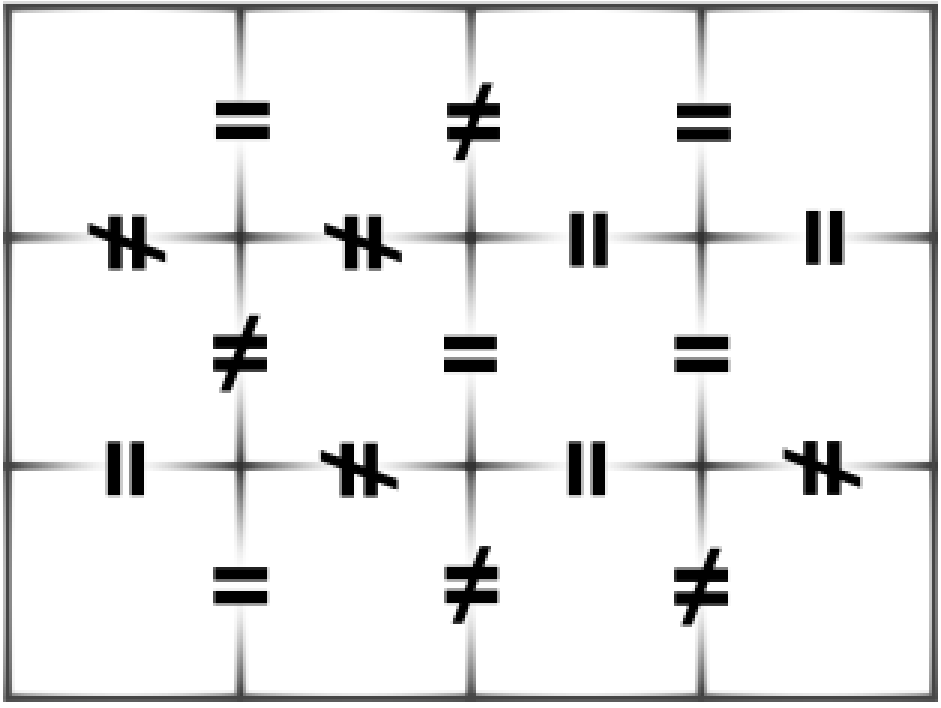
Note
The first example is shown in the picture.
In the second example, the final array is $[1, 12, 2, 8, 3, 11, 4, 9, 5, 13, 6, 10, 7]$.

D.9 AN EXAMPLE OF A QUESTION WITH AN OTHER IMAGE

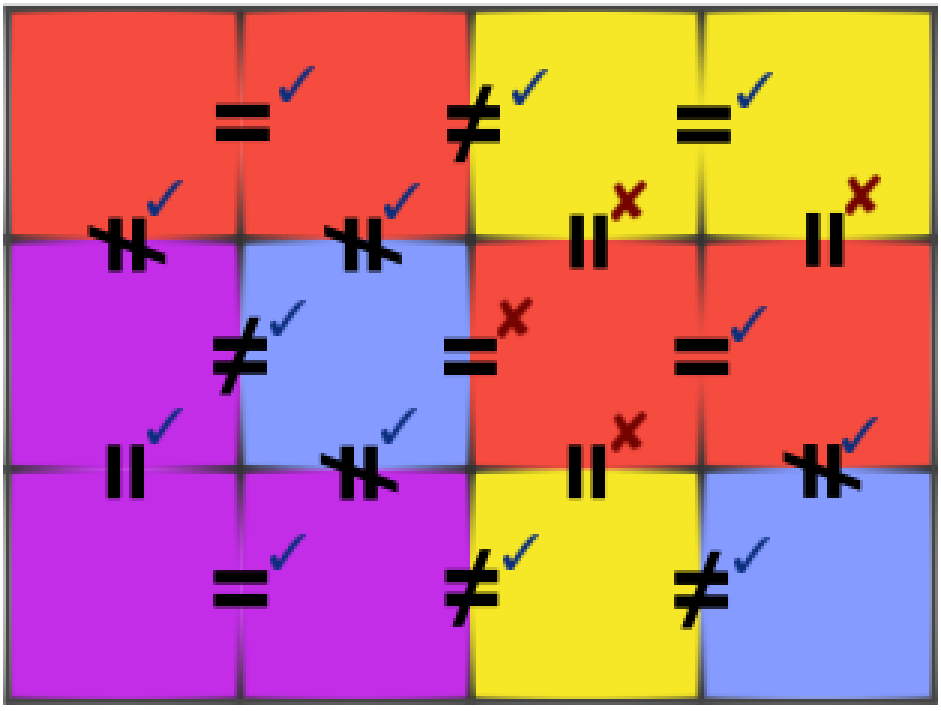
Even polar bears feel cold when lying on the ice. Therefore, a polar bear Alice is going to make a carpet. The carpet can be viewed as a grid with height h and width w. Then the grid is divided into $h \times w$ squares. Alice is going to assign one of k different colors to each square. The colors are numbered from 1 to k. She may choose not to use all of the colors. However, there are some restrictions. For every two adjacent squares (squares that share an edge) x and y, there is a color constraint in one of the forms:

- color(x) = color(y), or
- color(x) $\neq$ color(y).

Example of the color constraints:



Ideally, Alice wants to satisfy all color constraints. But again, life in the Arctic is hard. It is not always possible to satisfy all color constraints. Fortunately, she will still be happy if at least $\frac{3}{4}$ of the color constraints are satisfied.
If she has 4 colors she can color the carpet in the following way:



And she is happy because $\frac{13}{17}$ of the color constraints are satisfied, and $\frac{13}{17} > \frac{3}{4}$. Your task is to help her color the carpet.

Input

The first line contains three integers h, w, k ($2 \leq h, w \leq 1000, 1 \leq k \leq w \cdot h$). The next $2h - 1$ lines describe the color constraints from top to bottom, left to right. They contain $w-1, w, w-1, w, \ldots, w-1$ characters respectively. Each color constraint is represented by a character "E" or "N", where "E" means " $=$ " and "N" means " $\neq$ ". The color constraints are listed in the order they are depicted in the picture.

Output

If there is a coloring that satisfies at least $\frac{3}{4}$ of the color constraints, print "YES" (without quotes) on the first line. In each of the next h lines, print w integers describing the coloring. Otherwise, print "NO" (without quotes).

Examples

Input

```
3 4 4ENENNEENEEENENENN
```

Output

```
YES
1 1 2 2
3 4 1 1
3 3 2 4
```

D.10 An example of a question with an unrelated image

It's another day in the office, and you're a mastermind of not doing any work yourself. Instead, you'll go to your coworkers for "help," but secretly have them do all the work.

You've determined that the more one of your coworkers helps you, the more annoyed they become. You've also been able to determine how much more annoyed a coworker gets every time you ask them for help. At the beginning of the day, a coworker is initially a annoyed at you. That's their annoyance level. Every time you ask them for help though, they become d more annoyed at you – their annoyance level a increases by a constant amount d so that $a = a + d$.
You want to complete a project of h tasks solely with "help" from your coworkers, but you need to be careful not to annoy any of them too much. What's the best you can do?

Model	Linear	Tree	Graph	2D	3D	Chess-board	Map	Math	Patterns	Table	Pseudo-code	Others	Average
Language Only Inputs													
LLaVA-1.5-7B (text-only)	8.1	2.1	1.3	0.6	0.1	6.7	0.0	0.2	2.9	1.8	0.0	2.7	2.2
LLaVA-1.5-13B (text-only)	9.8	0.4	3.9	0.4	0.6	0.1	0.1	4.2	4.0	1.8	11.1	1.7	2.9
QWEN-VL (text-only)	4.3	1.9	0.2	2.5	3.0	9.3	3.8	1.7	0.0	1.8	10.2	2.2	2.7
CodeGemma-7b-Instruct	20.5	7.1	4.8	6.6	10.3	14.3	10.8	7.5	18.1	8.3	20.0	7.5	11.1
CodeLLaMA-7b-instsruct	12.1	3.9	5.4	5.6	0.4	3.6	1.7	5.9	8.7	6.5	10.3	0.9	5.4
CodeLLaMA-13b-instruct	9.3	7.9	5.0	5.3	2.8	6.8	2.1	1.7	3.8	5.4	10.0	1.4	4.9
DeepSeekCoder-7b-instruct	27.4	10.8	13.8	6.4	9.8	38.0	20.9	17.4	15.7	12.7	12.4	12.7	16.5
DeepSeekCoder-33b-instruct	31.1	7.2	19.7	22.8	13.5	31.0	26.6	18.7	18.5	15.5	31.1	25.3	21.5
LLaMA3-instruct	17.8	4.2	10.9	10.4	7.3	11.1	11.6	7.7	7.9	22.8	1.2	6.0	9.9
MagiCoder	24.9	8.4	17.1	8.6	3.0	20.4	17.5	10.4	21.5	12.8	21.7	19.5	15.3
StarCoder-15b-instruct	16.3	5.0	4.9	12.6	4.4	2.5	3.6	7.2	10.3	5.7	10.7	2.9	7.5
WizardCoder	11.2	5.4	6.1	8.2	2.2	10.7	6.3	10.1	15.3	5.9	21.0	6.2	8.6
Gemini Pro	25.0	3.9	4.9	7.5	6.0	13.6	13.2	4.3	21.1	8.1	20.4	17.1	12.0
GPT-3.5-turbo-1106	38.3	18.6	17.9	17.1	14.6	25.4	20.8	16.8	26.9	19.0	21.9	18.6	21.0
GPT-4	40.3	22.0	20.9	19.5	17.7	29.6	25.3	24.7	48.4	24.1	33.8	37.2	28.0
GPT-4V (text-only)	52.4	17.9	23.1	20.0	17.4	37.9	23.5	20.3	34.4	30.4	44.9	39.6	28.5
GPT-4o (text-only)	40.8	10.1	15.8	18.3	14.6	31.5	16.2	28.7	30.8	8.8	45.1	29.2	23.3
Vision + Language Inputs													
LLaVA-1.5-7B	12.6	4.5	0.6	3.4	2.7	6.7	0.8	0.1	0.2	2.6	0.0	0.4	3.2
LLaVA-1.5-13B	8.2	3.9	0.1	0.1	0.6	0.7	3.2	1.0	3.7	1.8	0.7	0.3	2.3
QWEN-VL	11.1	2.3	0.2	0.7	0.0	2.8	0.1	2.3	4.9	3.7	0.0	1.6	2.5
Gemini Pro Vision	20.6	4.9	5.9	7.1	6.9	10.8	15.0	5.0	16.2	7.9	31.2	7.5	10.7
GPT-4V	59.7	22.9	21.3	19.1	19.8	37.2	26.5	16.2	39.2	24.8	43.3	29.8	29.5
GPT-4o	44.7	12.8	20.0	13.0	15.9	34.9	34.2	31.6	37.3	15.2	50.0	25.5	27.0

Table 7: Test case average grouped by different image categories.

E MORE EXPERIMENTS

E.1 TEST CASE AVERAGE PASS RATES

We also report partial success metrics measured by the test case average following APPS[1], presented in Table 7. We observed that it aligns well with the pass@1 reported in Table 2.

E.2 IMAGE REPLACEMENT CAPTIONING

In this experiment, language-only models are prompted with Image Replacement captioning, but the captions are generated by different models. The results are showcased in Table 8. GPT-4's accuracy drops when using Gemini Pro Vision's captions, while Gemini Pro yields identical results.

Model	Caption Model	Linear	Tree	Graph	2D	3D	Chess-board	Map	Math	Patterns	Table	Pseudo-code	Others	Average
Gemini Pro	Gemini Pro Vision	16.0	0.0	0.0	6.7	0.0	6.7	3.6	0.0	11.1	7.1	20.0	7.7	6.1
Gemini Pro	GPT-4V (gpt-4-1106-preview)	16.0	0.0	0.0	6.7	0.0	6.7	3.6	0.0	11.1	7.1	20.0	7.7	6.1
GPT-4 (gpt-4-1106-preview)	Gemini Pro Vision	32.0	3.4	17.4	6.7	7.7	33.3	25.0	12.0	25.9	21.4	40.0	19.2	18.6
GPT-4 (gpt-4-1106-preview)	GPT-4V (gpt-4-1106-preview)	32.0	10.3	17.4	6.7	3.8	33.3	25.0	12.0	33.3	21.4	40.0	19.2	19.0

Table 8: Image Replacement captioning performance measured by Pass@1 (%) of models with different caption sources.

E.3 IMAGE POSITIONS

Since the problems are typically long, it is uncertain if the images receive sufficient attention of the model. Motivated by the findings of Liu et al. (2024b), we explored whether the position of the images affects the performance.

The results in Table 9 illustrate the impact of image positioning in the problem statements. Specifically, for Gemini Pro Vision, maintaining images in their original positions results in the highest pass rates. Grouping the images at either the beginning or the end of the texts hurt performance. No-

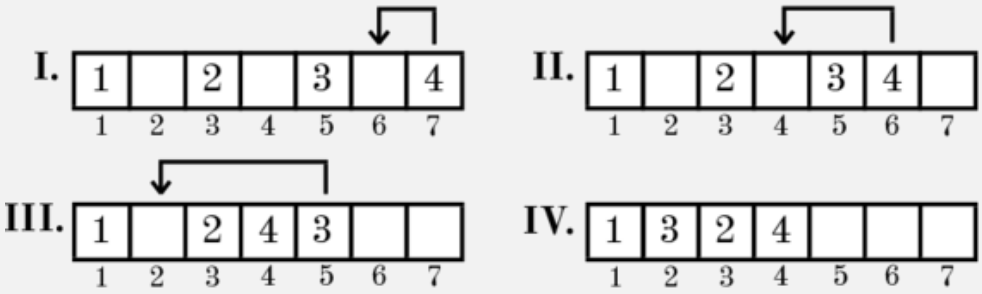
tably, GPT-4V demonstrates significant robustness, with its overall accuracy remaining unaffected.

Model	Image position	Linear	Tree	Graph	2D	3D	Chess-board	Map	Math	Patterns	Table	Pseudo-code	Others	Total
Gemini Pro Vision	in-place	12.5	0.0	4.3	0.0	3.8	6.7	7.1	0.0	7.4	0.0	30.0	0.0	5.0
	front	8.7	0.0	0.0	0.0	0.0	0.0	7.1	0.0	3.7	0.0	30.0	3.8	3.4
	end	16.0	0.0	0.0	0.0	3.8	0.0	3.6	0.0	7.4	0.0	30.0	3.8	4.6
GPT-4V (gpt-4-1106-preview)	in-place	40.0	6.9	13.0	13.8	3.8	21.4	24.0	9.5	25.9	21.4	40.0	20.8	19.4
	front	36.0	6.9	8.7	6.9	7.7	50.0	24.0	0.0	33.3	14.3	40.0	20.8	19.4
	end	40.0	6.9	13.0	10.3	7.7	21.4	36.0	14.3	18.5	14.3	40.0	20.8	19.4

Table 9: The impact of image positioning. “in-place” refers to images interspersed within the text, while “front” and “end” indicate images grouped at the start or end of the problem statement.

F CASE STUDY: CAPTIONS

Linear Data Structure



Gemini Vision Pro

The image shows an array of 4 elements, with indices ranging from 1 to 7. The elements are arranged as follows:
[1, 2, 3, 4, _, _, _]

GPT-4V

The image shows a sequence of four diagrams labeled I. to IV., representing the steps of Dima's algorithm applied to an array with n = 4 elements. Each diagram depicts an array with seven cells, numbered from 1 to 7 at the bottom. Initially, the numbers 1 to 4 are placed in cells 1, 3, 5, and 7, respectively. The diagrams illustrate the process of moving the numbers to the left into the nearest empty cells until all numbers are in the first four cells of the array. Arrows indicate the movement of numbers from one cell to another.

Figure 5: An example of a `Linear Data Structure` image. Gemini Pro Vision only sees one subfigure, and generates unusable captions. GPT-4V's caption is correct, but the details of step II, III and IV are not included.

Pseudocode

Algorithm 1 Bob's Dynamic Programming Algorithm

Input: matrix $A = (a_{i,j})_{n \times m}$;
Output: maximum score S after reaching the goal;

initialize $dp_{i,j} \leftarrow 0$, for all $0 \leq i \leq n$, $0 \leq j \leq m$ except $dp_{0,1} \leftarrow a_{1,1}$
for $i = 1 \rightarrow n$ **do**
    for $j = 1 \rightarrow m$ **do**
        $dp_{i,j} \leftarrow \max(dp_{i-1,j} \& a_{i,j}, dp_{i,j-1} \& a_{i,j})$
    end for
end for
$S \leftarrow dp_{n,m}$

Gemini Vision Pro

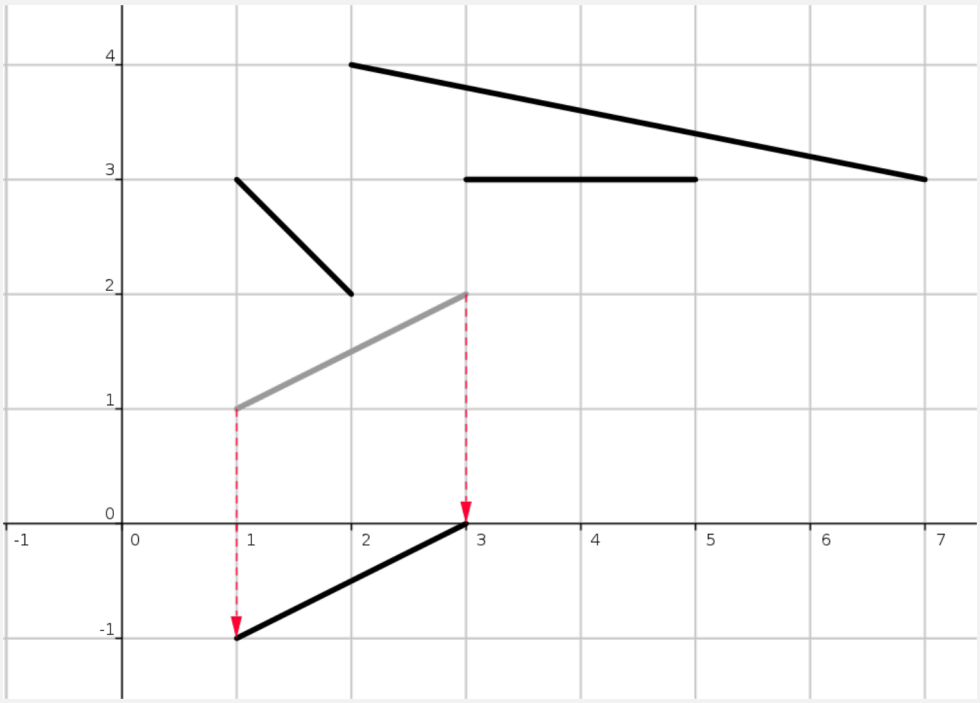
The image shows Bob's dynamic programming algorithm. It takes a matrix A as input and outputs the maximum score S after reaching the goal. The algorithm initializes a 2D array dp with all zeros, except for dp[0, 0] which is set to A[1, 1]. Then, for each row i from 1 to n, the algorithm iterates through each column j from 1 to m and updates dp[i, j] to the maximum of dp[i - 1, j] & A[i, j], dp[i, j - 1] & A[i, j], and dp[i - 1, j - 1] & A[i, j]. Finally, the algorithm returns dp[n, m].

GPT-4V

The image shows a text box with the title "Algorithm 1 Bob's Dynamic Programming Algorithm" and contains pseudocode for a dynamic programming approach to solve a problem related to a game named "Walk on Matrix". The algorithm takes as input an n by m matrix A with elements $a_{i,j}$ and outputs the maximum score S after reaching the goal. The pseudocode outlines the steps of initializing a 2D array dp with zeros, except for dp[0,1] which is set to $a_{1,1}$. It then iterates through the matrix, updating the dp array with the maximum of two bitwise AND operations involving the current element and its adjacent elements to the left and above. The final score S is set to the value of dp[n,m].

Figure 6: An example of a `Pseudocode` image. Gemini Pro Vision makes a mistake in the array index and hallucinates, while GPT-4V's caption is correct.

2D Geometry



Gemini Vision Pro

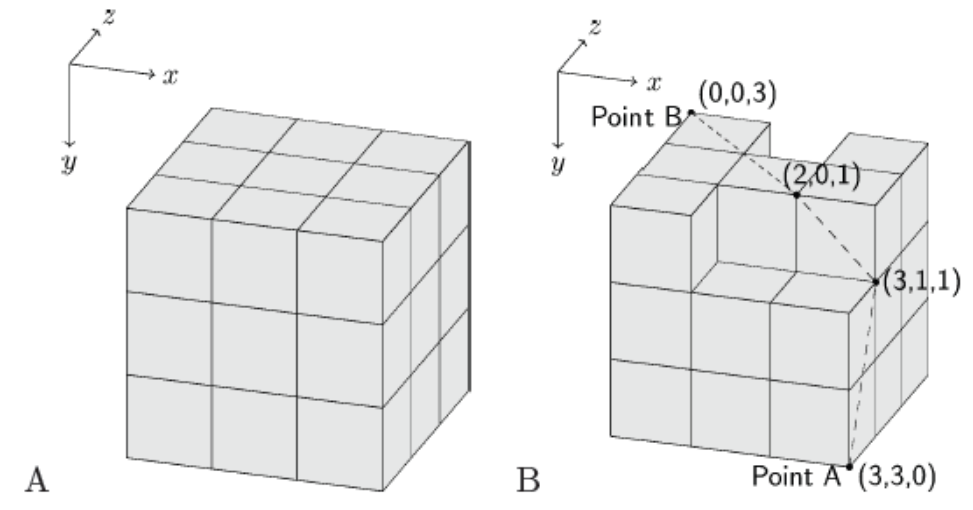
The image shows a square representing the table with four sticks on it. The sticks are represented by line segments. The first stick is a diagonal line from (1, 3) to (2, 2). The second stick is a horizontal line from (1, 1) to (3, 2). The third stick is a diagonal line from (2, 4) to (7, 3). The fourth stick is a vertical line from (3, 3) to (5, 3). The red dotted lines show the shortest paths for removing the sticks. The first stick is removed by moving it down to the x-axis. The second stick is removed by moving it left to the x-axis. The third stick is removed by moving it down to the x-axis. The fourth stick is removed by moving it left to the x-axis.

GPT-4V

The image shows a Cartesian coordinate system with a grid and three line segments representing sticks on a table. The x-axis is labeled from -1 to 7, and the y-axis is labeled from -1 to 4. There are three sticks: one is almost horizontal at y=3, another is slanted and crosses the y-axis at y=1, and the third is vertical at x=3. Red dashed arrows indicate the direction in which the sticks should be moved towards the x-axis. The vertical stick has an arrow pointing directly downward, while the slanted stick has an arrow pointing diagonally down to the nearest point on the x-axis. The horizontal stick does not have an arrow, but it is implied that it would move straight down as well. This illustrates the rule that sticks must be moved towards the x-axis along the shortest path without rotation or touching other sticks.

Figure 7: An example of a `2D Geometry` image. Both models struggle to discern the segments and extract their attributes.

3D Geometry



Gemini Vision Pro

Figure J.1A shows the largest colony, which consists of $3 \times 3 \times 3$ cubic blocks. Figure J.1B shows a smaller colony with some blocks missing. The dashed line in Figure J.1B is an example of a cable connecting two points on the surface of the colony.

GPT-4V

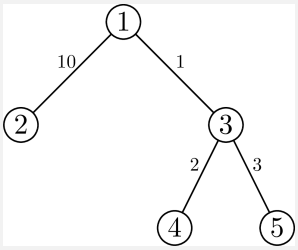
The image contains two parts labeled as Figure J.1A and Figure J.1B, illustrating examples of space colonies made up of cubic blocks. Both parts show a three-dimensional coordinate system with axes labeled x, y, and z.

Figure J.1A depicts a large colony that resembles a Rubik's cube, consisting of $3 \times 3 \times 3$ cubic blocks, forming a perfect cube.

Figure J.1B shows a smaller colony with some cubic blocks missing. The colony is irregularly shaped due to the missing blocks. There are dashed lines indicating the shortest path for a cable on the surface of the colony, connecting two points labeled as Point A (3,3,0) and Point B (0,0,3). The path of the cable wraps around the surface of the blocks, illustrating the requirement that the cable must remain on the surface.

Figure 8: An example of a `3D` image. Gemini Vision Pro only saw one of the segmented lines (called "cables" in the problem). GPT-4V interprets the image correctly.

Tree



Gemini Vision Pro

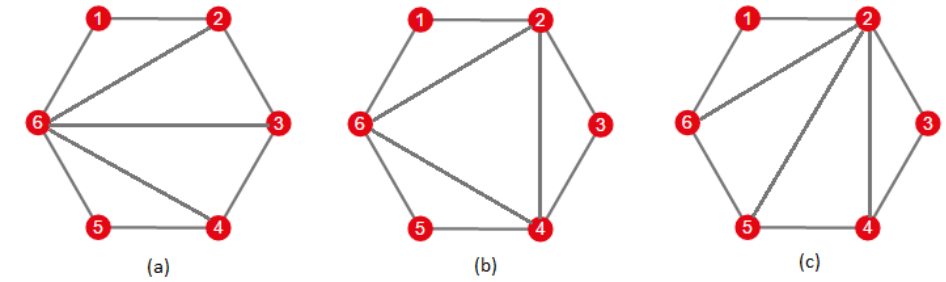
The image shows a tree with 5 vertices and 4 edges. The vertices are numbered from 1 to 5. The edges are: (1, 2) with weight 10, (1, 3) with weight 1, (3, 4) with weight 2, and (3, 5) with weight 3.

GPT-4V

The image shows a tree graph with 5 vertices labeled 1 through 5. Vertex 1 is connected to vertex 2 with an edge weight of 10 and to vertex 3 with an edge weight of 1. Vertex 3 is further connected to vertices 4 and 5 with edge weights of 2 and 3, respectively. The graph is drawn with vertices represented as circles containing their labels, and the edges are lines connecting these circles with the weights written alongside.

Figure 9: An example of a `Tree` image. Both models generate correct captions.

Graph



Gemini Vision Pro

The image shows a regular hexagon with 6 vertices labeled 1 through 6. There are 3 diagonals drawn in the hexagon, forming a triangulation.

GPT-4V

The image shows three figures labeled (a), (b), and (c), each representing a different triangulation of a hexagon (a six-edged polygon). Each vertex of the hexagon is numbered from 1 to 6 in a clockwise direction. The figures illustrate the process of transforming one triangulation into another by flipping diagonals.

Figure (a) shows the initial triangulation with diagonals connecting vertices 1-4, 1-5, and 5-3.
Figure (b) shows an intermediate step after flipping the diagonal 6-3 to 2-4.
Figure (c) shows the goal triangulation with diagonals connecting vertices 1-4, 2-4, and 2-3.

Figure 10: An example of a `Graph` image. Gemini Vision Pro fails to recognize all three hexagons and the caption is vague, missing key information of how the hexagons are constructed. GPT-4V summarizes all subfigures but hallucinates on the vertices connected.

Table

step	Vladik	Valera
0	7	6
1	6	6
2	6	4
3	3	4
4	3	0

Gemini Vision Pro

The image shows a 2x3 table with the following values:

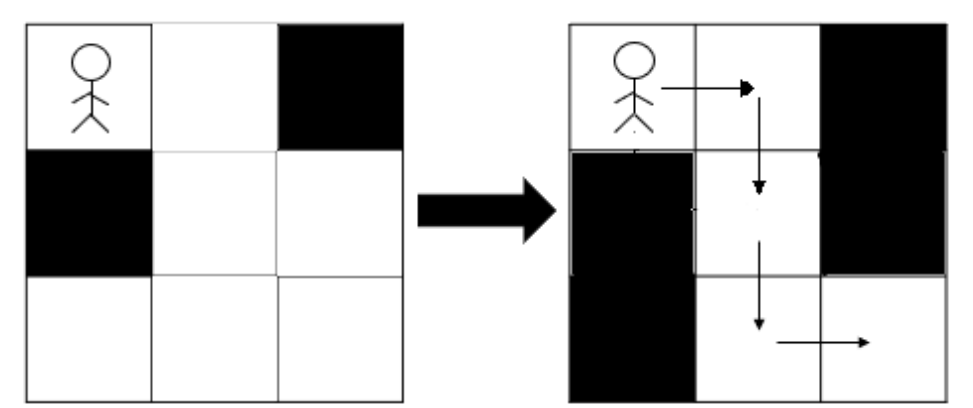
step — Vladik — Valera
———-———————
0 — 7 — 6
1 — 6 — 7
2 — 5 — 8
3 — 4 — 9
4 — 3 — 10
5 — 2 — 11
6 — 1 — 12
7 — 0 — 13

GPT-4V

The image shows a table with three columns and three rows. The columns are labeled ”step”, ”Vladik”, and ”Valera”. The rows represent the steps of the candy exchange process. At step 0, Vladik has 1 candy, and Valera also has 1 candy. At step 1, after Vladik gives 1 candy to Valera, Vladik has 0 candies left, and Valera has 2 candies.

Figure 11: An example of a `Table`. Surprisingly, neither of the models were able to transcript the table correctly.

Map



Gemini Vision Pro

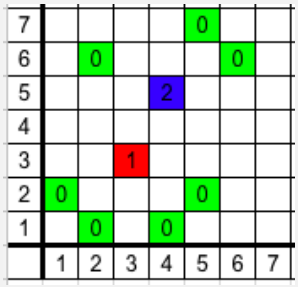
The image shows a 3x3 grid with a character called Kenus at square (1, 1). The squares are painted black or white. Kenus can move up, down, left or right by one square. The game is completed when Kenus reaches square (3, 3) passing only white squares. Snuke can change the color of some of the white squares to black before the beginning of the game. However, he cannot change the color of square (1, 1) and (3, 3).

GPT-4V

The image shows a 3x3 grid with a stick figure representing Kenus at the top left corner (square 1,1). The grid has black and white squares, with the black squares represented by a solid fill and the white squares by an absence of fill. The left side of the image shows the initial state of the grid, with the top right square (1,3) and the middle left square (2,1) being black. The right side of the image shows the same grid after two white squares have been changed to black, specifically the middle square (2,2) and the bottom middle square (3,2). Arrows indicate the path Kenus can take to reach the bottom right corner (3,3), moving right, down, down, and right, avoiding black squares.

Figure 12: An example of a `Map` image. Gemini Pro Vision's caption is correct overall, but important details about the path is missing. GPT-4V recognizes the switched squares, but grounds them to wrong positions.

Chessboard



Gemini Vision Pro

The image shows a 7x7 chessboard with 7 green knights placed on it. The green knights are placed in the following cells: (1, 2), (2, 1), (4, 1), (5, 2), (5, 7), (6, 6), and (7, 6). There is a red knight in cell (3, 3), and a blue knight in cell (4, 5).

GPT-4V

The image shows a 7x7 section of an infinite chessboard with rows labeled 1 to 7 from bottom to top and columns labeled 1 to 7 from left to right. There are five cells marked with green zeroes, representing the initial knights' positions. These cells are located at coordinates (1,2), (2,1), (2,6), (5,2), and (5,7). There is one cell marked with a red one at coordinates (3,3), indicating a knight placed during the process because it is under attack by at least four knights. Another cell is marked with a blue two at coordinates (4,5), showing that it will also have a knight placed in it after the knight in the red cell is added, as it will then be under attack by four knights. The rest of the cells are empty.

Figure 13: An example of a `Chessboard` image. The descriptions are mostly correct, but both models fail to list all green squares.

Math

$$f_{r+1}(n) = \sum_{u \cdot v = n} \frac{f_r(u) + f_r(v)}{2},$$

Gemini Vision Pro

The image shows a screenshot of a programming problem. The problem statement is as follows:
...(Repeats the problem statement) f_{r+1} is defined as:

$$f_{r+1}(n) = \sum_{u^2 - v^2 = n} \frac{f_r(u) + f_r(v)}{2}$$

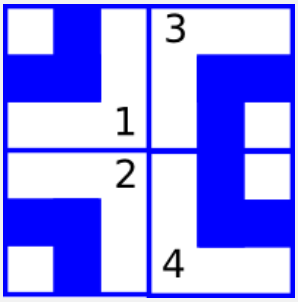
...(The rest of the problem statement)

GPT-4V

The image shows a mathematical function definition for $f_{r+1}(n)$ which is equal to the sum over all ordered pairs (u, v) such that $u * v = n$ of the average of $f_r(u)$ and $f_r(v)$. The function f_r is defined recursively, and the base function $f_0(n)$ counts the number of ways of factoring n into two co-prime factors p and q. The image is a part of a programming problem statement that involves calculating the value of $f_r(n)$ for given values of r and n, modulo $10^9 + 7$.

Figure 14: An example of a `Math` image. Gemini Pro Vision's output contains unnecessary parts, and the subscript of the recognized equation is wrong. GPT-4V's caption is accurate.

Pattern



Gemini Vision Pro

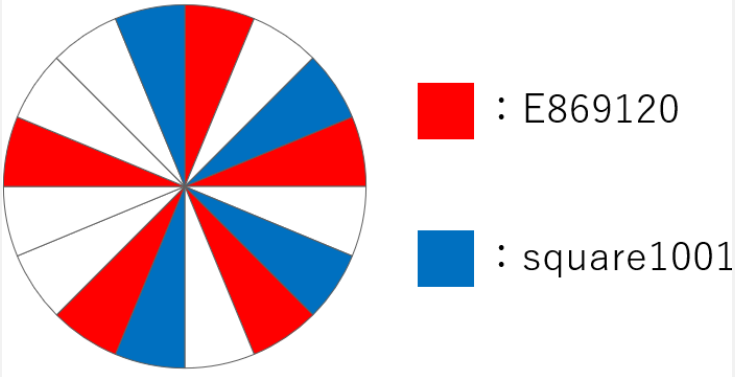
The image shows a 2x2 grid with 4 pipe segments. The segments are numbered 1, 2, 3, and 4.
Segment 1 is a horizontal segment at the top of the grid.
Segment 2 is a vertical segment at the left side of the grid.
Segment 3 is a horizontal segment at the bottom of the grid.
Segment 4 is a vertical segment at the right side of the grid.

GPT-4V

The image shows four types of pipe segments, each represented by a square with two arrows indicating the ends of the pipe. Segment 1 has ends at the top and left, segment 2 has ends at the bottom and left, segment 3 has ends at the top and right, and segment 4 has ends at the bottom and right.

Figure 15: An example of a `Pattern` image. GPT-4V succeeds in describing two of the four blocks, whereas Gemini Vision Pro fails to interpret any.

Others



Gemini Vision Pro

The image shows a round cake cut into 16 equal fan-shaped pieces.
The pieces are arranged in a circle, and each piece is labeled with a number from 1 to 16.
The colors of the pieces alternate between red and blue.

GPT-4V

The image shows a round cake cut into 16 equal fan-shaped pieces. The pieces are alternately colored in red and blue to represent the pieces taken by E869120 and square1001, respectively. There are 5 red pieces and 4 blue pieces, with no two adjacent pieces of the same color, indicating that the instruction in the note has been followed.

Figure 16: An example of a `Others` image. Both models are correct.

G CASE STUDY: CODE ANALYSIS

In this section, some samples of machine-generated solution code are presented and studied. Incorrect segments are colored in red and bolded in the code listed.

G.1 INCORRECT AND INEFFICIENT SOLUTION

Sometimes GPT-4V can generate inefficient code that takes too long to finish execution, leading to a time-out. An demonstration is given in Figure 17 and Figure 18. Moreover, it implements a wrong method of calculating the required quantity.

The problem statement is listed in Figure 17. It asks to find the number of faces in a minimum 3D shape made of unit cubes that cover all integer coordinates within a sphere of radius $\sqrt{n}$. In Figure 18, GPT-4V's solution loops over all integer points inside $-\sqrt{n} \leq x, y, z \leq \sqrt{n}$. A cleverer method is to leverage the symmetry and only count the faces perpendicular to an axis, reducing the operations to $\frac{1}{6}$ of the original. Moreover, the algorithm for calculating the connected faces in the code is incorrect.

G.2 TYPEERROR CAUSED BY NAMING CONFLICT

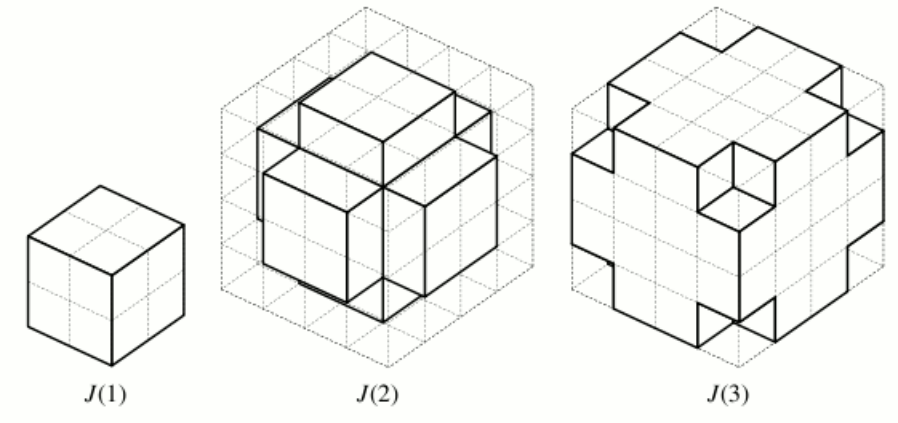
As mentioned in Section 6.3.2, despite the advance in understanding complex questions and generating syntactically correct solutions, GPT-4V still makes basic errors. In this example code shown in Figure 19, GPT-4V assigns the name `len` to a local variable, which conflicts with and overrides the Python built-in `len()` function. Consequently, when the code attempts to obtain the length of a list on line 14, it raises a `TypeError`. This is because `len` now refers to the variable, not the function it originally intended to use.

Problem

Let $J(n)$ be a three-dimensional body that:

- is a union of unit cubes whose all vertices lie on integer coordinates,
- contains all points that are closer than the distance of $\sqrt{n}$ to the origin, and
- is the smallest of all such bodies.

Figure 1 below shows how $J(1)$, $J(2)$, and $J(3)$ look.



Your task is to calculate how many faces $J(n)$ have. Here, we define two squares belong to the same face if they are parallel and share an edge, but don't if they share just a vertex.

Input

The input consists of multiple data sets, each of which comes with a single line containing an integer n ($1 \leq n \leq 1000000$). The end of input is indicated by $n = 0$.

Output

For each data set, print the number of faces $J(n)$ have.

Sample Input

```
1
2
3
4
0
```

Output for the Sample Input

```
6
30
30
6
```

Figure 17: The description of the problem of the case study.

```python
import math

def count_faces(n):
    # Calculate the radius of the sphere
    r = math.sqrt(n)

    # Calculate the range for x, y, z
    limit = math.ceil(r)

    # Initialize the count of faces
    faces = 0

    # Iterate through all possible cubes and count the
    ↪ faces
    for x in range(-limit, limit + 1):
        for y in range(-limit, limit + 1):
            for z in range(-limit, limit + 1):
                # Check if the cube is on the surface of
                ↪ J(n)
                if r - 1 < math.sqrt(x**2 + y**2 + z**2) <=
                ↪ r:
                    # Check each face of the cube
                    for dx, dy, dz in [(1, 0, 0), (0, 1,
                    ↪ 0), (0, 0, 1)]:
                        # If the neighboring cube is not
                        ↪ part of J(n), count the face
                        if math.sqrt((x+dx)**2 + (y+dy)**2
                        ↪ + (z+dz)**2) > r:
                            faces += 1

    return faces

# Read input and process each case
while True:
    n = int(input())
    if n == 0:
        break
    print(count_faces(n))
```

Figure 18: An inefficient and incorrect solution generated by GPT-4V.

GPT-4V's solution

```
def calculate_optimal_sum(n, len, a, k):
    # Helper function to calculate the sum of a subarray
    def subarray_sum(i):
        return sum(a[i:i+len])

    # Calculate the initial optimal sum without any operations
    optimal_sum = max(subarray_sum(i) for i in range(n - len + 1))

    # Sort the array elements by their absolute values in descending order
    # This will help us to choose which elements to flip to maximize the sum
    negative_elements = sorted((x for x in a if x < 0), key=abs, reverse=True)

    # Perform up to k operations, flipping the most negative elements
    for i in range(min(k, len(negative_elements))):
        # Flip the negative element to positive
        negative_elements[i] *= -1
        # Update the array with the flipped element
        a[a.index(negative_elements[i] * -1)] = negative_elements[i]

    # Calculate the new optimal sum after performing the operations
    optimal_sum = max(optimal_sum, max(subarray_sum(i) for i in range(n - len + 1)))

    return optimal_sum

# Read input
n, len = map(int, input().split())
a = list(map(int, input().split()))
k = int(input())

# Calculate and print the result
print(calculate_optimal_sum(n, \textbf{len}, a, k))
```

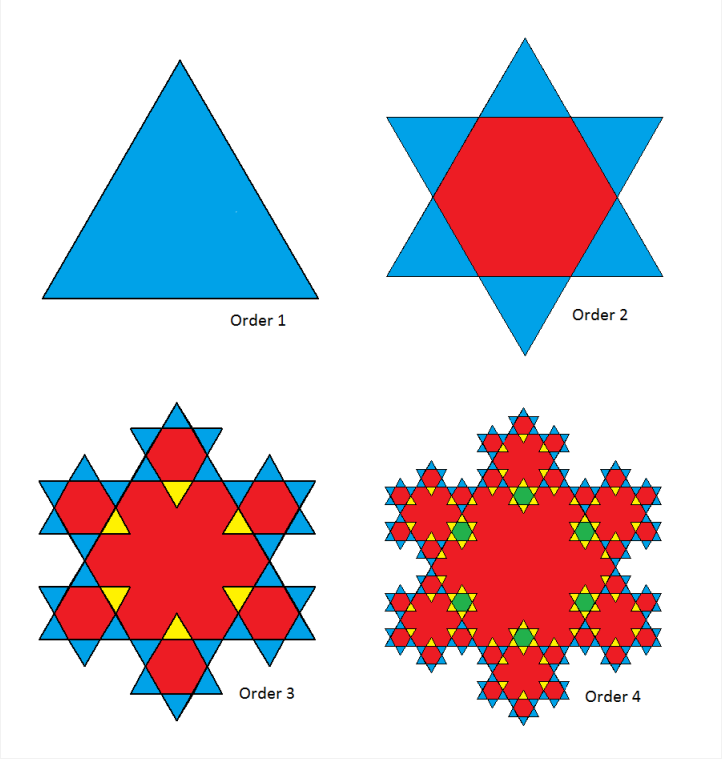
Figure 19: A sample of GPT-4V's wrong solution caused by a name conflict.

G.3 INPUT PARSING ERROR

The problem shown in Figure 20 asks the summation of a certain function with input from 3 to 10^7. However, GPT-4V tries to read the upper limit from user input in line 24 in Figure 21, raising an EOFError (End of File Error) because the input is empty.

Problem

A snowflake of order n is formed by overlaying an equilateral triangle (rotated by 180 degrees) onto each equilateral triangle of the same size in a snowflake of order $n-1$. A snowflake of order 1 is a single equilateral triangle.



Some areas of the snowflake are overlaid repeatedly. In the above picture, blue represents the areas that are one layer thick, red two layers thick, yellow three layers thick, and so on. For an order n snowflake, let $A(n)$ be the number of triangles that are one layer thick, and let $B(n)$ be the number of triangles that are three layers thick. Define $G(n) = \gcd(A(n), B(n))$.
E.g. $A(3) = 30$, $B(3) = 6$, $G(3) = 6$.
$A(11) = 3027630$, $B(11) = 19862070$, $G(11) = 30$.
Further, $G(500) = 186$ and $\sum_{n=3}^{500} G(n) = 5124$.
Find $\sum_{n=3}^{10^7} G(n)$.

Figure 20: The description of the problem that causes an input parsing error.

```python
from math import gcd

def calculate_snowflake_layers(n):
    # Base values for A(3) and B(3)
    A = 30
    B = 6
    G = gcd(A, B)

    # Calculate A(n) and B(n) using the recursive
    ↪ relationship
    for i in range(4, n + 1):
        A = A * 3 + 12 * (i - 2)
        B = B * 3 + 6 * (i - 2)
        G = gcd(A, B)

    return G

def sum_of_gcds(n):
    total_sum = 0
    for i in range(3, n + 1):
        total_sum += calculate_snowflake_layers(i)
    return total_sum

# Read the upper limit from standard input
upper_limit = int(input())

# Calculate and print the sum of GCDs
print(sum_of_gcds(upper_limit))
```

Figure 21: GPT-4V tries to read the upper limit from user input but it should be hardcoded 10^7.