Uncertainty-Aware Program Synthesis with Language Models

Minjune Hwang*
Stanford University
mjhwang@stanford.edu

Adithya Ganesh*
Stanford University
acganesh@stanford.edu

Abstract

In this work we apply language models to the problem of program synthesis: generating programs from text descriptions of their desired behavior. We focus specifically on sampling strategies for the predicted probability distributions $P(s_n|s_1, \ldots, s_{n-1})$ output by the trained language model. In particular, we experimented with greedy sampling (GreedySampler), beam search (BeamSearchSampler), and entropy-based sampling (EntropySampler), which leverages the model's uncertainty to dynamically adjust the number of beams when sampling the distribution. On the introductory problems in the APPS dataset (Hendrycks et al., 2021), we found that EntropySampler had the best test-case accuracy (0.092) and timeout rate (0.039) while BeamSearchSampler at 5 and 10 beams had the largest number of problems solved (22).

1 Introduction

Program synthesis refers to the task of synthesizing a computer program that satisfies given specifications and aligns with the user's intent. In recent years, researchers have shown that language models are capable of generating natural languages and solving language-based reasoning tasks when combined with expressive deep learning models. Thanks to the success of large language models, there has been an increasing number of works on the intersection of language models and program synthesis (Chen et al., 2021; Feng et al., 2020; Austin et al., 2021; Li et al., 2022; Guo et al., 2022; Drain et al., 2021).

In this project, we apply language models to the task of program synthesis: generating full programs from natural language descriptions of code. In particular, we ask the question: can we leverage the model's uncertainty to sample more accurate programs from the model's predicted probability distribution $P(s_n|s_1, \ldots, s_{n-1})$? We hypothesize that using the entropy of this predicted distribution as a signal to sample more candidates can increase the average success rate when evaluating against test cases.

2 Prior Literature

OpenAI Codex (Chen et al., 2021), CodeBert (Feng et al., 2020), and (Austin et al., 2021) consider a fundamental task in program synthesis, which is the problem of generating code from docstrings or text descriptions of what the code should do. Other papers, on the other hand, provide approaches to tackle more specific tasks or use cases related to program synthesis. AthenaTest (Tufano et al., 2020) aims to generate unit-tests for Java methods. DeepDebug (Drain et al., 2021) aims to automatically fix Python bugs from stack traces. GrammFormer (Guo et al., 2022) defines "holes" to designate areas of uncertainty, and its language model generates code completions with sketches, a mix of actual tokens and "holes."

Various works have compared existing sampling strategies and proposed novel sampling methods for language models. The most intuitive and simple method is greedy sampling where a sequence is generated by iteratively selecting a token with a maximum probability. Similarly, beam search finds the most likely sequences, but with breadth-first search in a restricted search space. At each step, beam search keeps the top-k most likely sequences, and expands on them iteratively. However, as suggested in (Li and Jurafsky, 2016), sequences generated by standard beam search only differ in a few last tokens or minor details, resulting in limited diversities among generated sequences and possibly low-quality sequences.

To overcome this issue, various search methods and sampling methods have been proposed to encourage a language model to generate a more diverse set of sequences. Diverse beam search (Vi-

*Equal Contribution

jayakumar et al., 2016) adds an additional term to the score function to penalize similar partial sequences. Cluster-based beam search (Tam, 2020) searches more sequences in the early stage and cluster similar sequences in the later stage to avoid generating sequences that are too close to each other. Iterative beam search (Kulikov et al., 2018) avoids exploring already-explored intermediate sequences from previous iterations in current iteration by iteratively running beam search several times. Nucleus sampling (Holtzman et al., 2020) also expands on beam search in a different direction to ensure quality of generated sequences by truncating less reliable tokens during search.

In this work, we mainly focus on the comparison of our proposed sampling method against greedy sampling and beam search sampling, which are two widely-used methods for language model sampling.

3 Data

We focus this investigation on the Automated Programming Progress Standard (APPS) Dataset (Hendrycks et al., 2021). This dataset consists of problems from different freely accessible coding websites such as Codeforces and Kattis. The APPS dataset attempts to mirror how humans program by providing a natural language prompt as input and using test cases to evaluate solution correctness. In effect, this dataset provides an evaluation benchmark on **(a)** understanding the task description, **(b)** generating code that is syntatically correct, and **(c)** generating algorithms to solve these tasks.

The problems range in difficulty level from introductory (e.g. a student with 1-2 years of programming experience could reliably solve all problems in this category) to advanced (e.g. problems from college programming contests). The APPS dataset consists of 10000 problems in total, with 131777 test cases and 232431 ground truth solutions written by humans. The data is split into a train and test set with 5000 problems each. We focus primarily on the "introductory" level problems, of which there are 3,639 in total and 1,000 in the test set.

Figure 1 shows a sample introductory level problem in the test set. Each problem's text consists of a problem description, constraints, input, and output. The text also provides a pair of sample input and output with explanation. Given the concatenated text of the above elements, a language model is expected to return solution code for the problem.

Problem 4238

Problem: An integer N is a multiple of 9 if and only if the sum of the digits in the decimal representation of N is a multiple of 9. Determine whether N is a multiple of 9.

-----Constraints-----
- $0 \leq N < 10^{200000}$
- N is an integer.

-----Input-----
Input is given from Standard Input in the following format: N

-----Output-----
If N is a multiple of 9, print Yes; otherwise, print No.

-----Sample Input-----
123456789

-----Sample Output-----
Yes

***(Explanation**: the sum of these digits is 1+2+3+4+5+6+7+8+9 =45, which is a multiple of 9, so 123456789 is a multiple of 9.)*

Figure 1: Sample Problem in the APPS dataset

4 Model

We focus our experiments on GPT-2 with 1.5B parameters. GPT-2 formulates the language modeling task as unsupervised distribution estimation from a set of examples $(x_1, x_2, \ldots, x_n)$ each composed of variable length sequences of symbols $(s_1, s_2, \ldots, s_n)$. The joint probabilities over symbols are expressed as a product of conditional probabilities:

$$p(x) = \prod_{i=1}^{n} p(s_i | s_1, \ldots, s_{i-1}).$$

This formulation makes training and inference (sampling) tractable.

Architecturally, GPT-2 uses a Transformer based architecture. The version of GPT-2 used in this work is the largest version described in (Radford et al., 2019), and contains 48 layers, with $d_{model} = 1600$.

4.1 Pretraining and Fine-Tuning

For all of our experiments, we use the GPT-2 model pretrained by (Hendrycks et al., 2021). The authors pretrained GPT-2 on 30GB of Python code obtained from Github. The authors filtered out Github repositories with fewer than one star to ensure that the code trained on was of reasonably high quality. The authors processed all Python dataset in the pretraining dataset by converting from spaces to tabs.

Next, the authors of (Hendrycks et al., 2021) fine-tuned the model on the training set of the APPS dataset. For this task, the objective is to predict the entire code solution given the English text problem statement and the format of the input (generally reading from standard input). For pretraining and fine-tuning, the authors used the AdamW optimizer and fine-tuned for 10 epochs.

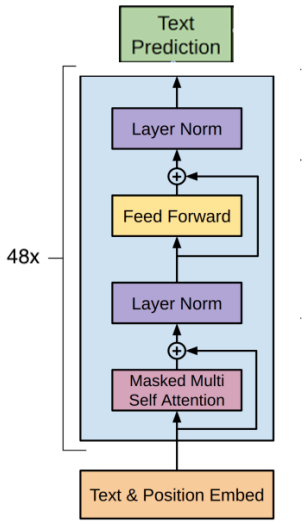


Figure 2: Architecture diagram of transformers used in GPT-2. We use the version with 1.5B parameters, 48 layers, and $d_{model} = 1600$.

5 Methods

Our experiments focused on testing out different sampling methods to postprocess the predicted probability distributions $p(s_n|s_1, \ldots, s_{n-1})$. The baseline was greedy sampling, and we experimented with beam search as we varied the number of beams, and entropy-based sampling, which is described in more detail below.

5.1 GreedySampler

The idea behind the greedy sampler is simple: we just pick the next token that has the highest probability. If x denotes the predicted program, greedy sampling will converge to a local optimum of $p(x)$, but this is unlikely to be the global optimum. We describe the pseudocode of GreedySampler below.

5.2 BeamSearchSampler

The next sampling method we tested was BeamSearchSampler. Here, we use beam search to sample the probability distribution, where we keep k "beams" or k candidates around that are the best partial-programs we have constructed so far. k is a hyperparameter that lets us trade off inference latency with the quality of the predicted program: intuitively, a larger number of beams will result in inference taking longer but will result in a better final solution. Since this is not a greedy algorithm, the results are expected to be generally better than GreedySampler.

5.3 EntropySampler

The final sampler we tested out was EntropySampler. Let $p(s_n|s_1, \ldots, s_{n-1})$ denote the predicted probability distribution over T tokens that the model outputs at a particular point in program generation. Let p_i denote the predicted probability of the i-th token as output by the previously mentioned distribution. We can compute the entropy of the predicted discrete probability distribution as

$$H = \frac{1}{T} \sum_{i=1}^{T} \log \frac{1}{p_i}.$$

In EntropySampler we dynamically increase the number of beams based on the value of the entropy H. If $H > h_{max}$, we set the number of beams to be b_{high} = numBeamsHighEntropy. Otherwise, we will set the number of beams to be b_{low} = numBeamsLowEntropy. The idea is that we will sample more diverse candidates when we are more unsure of our predictions. On the contrary, when we are more confident in our predictions, we will sample fewer candidates.

We present pseudocode for each of the samplers on the next page.

6 Experiments

6.1 Experiment Setup

Using a fine-tuned GPT-2 model, we generated solutions for test case problems from the APPS Dataset with three different sampling methods described above to assess the accuracy and efficiency of them. For the beam search sampler, we used different number of beams (3, 5, and 10).

- GreedySampler
- BeamSearchSampler with 3 beams
- BeamSearchSampler with 5 beams
- BeamSearchSampler with 10 beams
- EntropySampler ($H_{max} = 6.0, b_{high} = 7, b_{low} = 3$)

Algorithm 1 GreedySampler: Pick the next token with highest probability.

```
procedure GreedySampler(N_p, maxLength, inputPrompt)
    Programs ← []
    M ← InitGPT2Model()

    for i in [1..N_p] do
        inputs ← prompt
        while len(input) < maxLength and input[-1] != END do
            logits ← M.Inference(input)
            input += argmax(logits)
        Programs.append(input)
    return Programs
```

Algorithm 2 BeamSearchSampler: Apply beam search to keep k candidates that are the highest probability partial programs seen so far.

```
procedure BeamSearchSampler(N_p, maxLength, inputPrompt, numBeams)
    Programs ← []
    M ← InitGPT2Model()

    for i in [1..N_p] do
        bestCandidates ← [input]
        terminate ← false
        while len(bestCandidates[0]) < maxLength and !terminate do
            logits ← M.Inference(bestCandidates)
            bestCandidates, terminate ← pickBestCandidatesAndPrune(logits, numBeams)
    return Programs
```

Algorithm 3 EntropySampler: Use the entropy of the model to dynamically set the number of beams.

```
procedure EntropySampler(N_p, maxLength, hMax, numBeamsLowEntropy, numBeamsHighEntropy, inputPrompt)
    Programs ← []
    M ← InitGPT2Model()

    for i in [1..N_p] do
        terminate ← false
        bestCandidates ← [input]
        while len(bestCandidates[0]) < maxLength and !terminate do
            logits ← M.Inference(bestCandidates)
            pdf ← softmax(logits)
            entropy ← calculateEntropy(pdf)
            if entropy > hMax then
                bestCandidates, terminate ← pickBestCandidatesAndPrune(logits, numBeamsHighEntropy)
            else
                bestCandidates, terminate ← pickBestCandidatesAndPrune(logits, numBeamsLowEntropy)
    return Programs
=0
```

Section 6.1: Experiment Setup – continued.

We test each sampler to generate a solution for each problem in the 1000 introductory level problems in the test set. Also, since we can generate more solutions for each problem using the beam search sampler with multiple beams, we have the model output as many solutions as possible for each problem. For instance, we generate top 5 solutions for each problem with the beam search sampler with 5 beams, which are used to compute accuracy with different numbers of outputs.

6.2 Metrics

In this section, we list metrics used in this project. The first two metrics are identical to metrics in (Hendrycks et al., 2021), whereas we newly introduce the latter two metrics ("problem set coverage" and "timeout rate"). We want to measure if our uncertainty-driven method (EntropySampler) can efficiently generate programs with high accuracy and a low timeout rate.

6.2.1 Test Case Accuracy

Test case accuracy is computed as the average fraction of test cases passed. Let the number of problems in the test set be P, and set of test cases for problem p be $(x_{p,c}, y_{p,c})_{c=1}^{C_p}$. Then, for the code generated to solve the problem p be denoted $code_p$, we can compute the single-solution test case accuracy as the following.

$$\frac{1}{P}\sum_{p=1}^{P}\frac{1}{C_p}\sum_{c=1}^{C_p} I\left\{eval\left(\langle code_p\rangle, x_{p,c}\right) = y_{p,c}\right\}$$

Now, if we have our model generate N_p different solutions to solve p, we can denote the ith code as $code_{p,i}$. Then, for the multi-solution case, the test case accuracy is

$$\frac{1}{P}\sum_{p=1}^{P}\frac{1}{N_p}\sum_{i=1}^{N_p}\frac{1}{C_p}\sum_{c=1}^{C_p} I\left\{eval\left(\langle code_{p,i}\rangle, x_{p,c}\right) = y_{p,c}\right\}$$

For abbreviation, we name the multi-solution accuracy with k different solutions as Accuracy @ k. Accuracy @ AvgAll refers to the mean of accuracy among multiple solutions for each problem.

6.2.2 Number of Problems Solved

As its name suggests, this metric measures the number of problems for which a corresponding solution passed all test cases. In the single-solution case, it is computed by taking the number of problems with a solution passing every test case.

$$\sum_{p=1}^{P}\prod_{c=1}^{C_p} I\left\{eval\left(\langle code_p\rangle, x_{p,c}\right) = y_{p,c}\right\}$$

Similarly, strict accuracy is computed as the fraction of the fully solved problems in the dataset.

$$\frac{1}{P}\sum_{p=1}^{P}\prod_{c=1}^{C_p} I\left\{eval\left(\langle code_p\rangle, x_{p,c}\right) = y_{p,c}\right\}$$

6.2.3 Problem Set Coverage

Problem set coverage is computed as the fraction of problems in which at least one solution passed every test case. This metric increases as we generate more solutions for each problem. For $N_p = 1$, this value is identical to strict accuracy.

$$\frac{1}{P}\sum_{p=1}^{P}\max_{1\le i\le N_p}\prod_{c=1}^{C_p} I\left\{eval\left(\langle code_{p,i}\rangle, x_{p,c}\right) = y_{p,c}\right\}$$

6.2.4 Timeout Rate

Timeout rate is computed as the fraction of solutions in which at least one of the test cases resulted in a timeout error. For this project, we use the timeout threshold to be one second since each test case is meant to take a very small amount of time for computation. That is, if any of the test case takes more than one second, we consider the solution as timed-out. T_{eval} denotes the inference time of the code given the input.

$$\frac{1}{P}\sum_{p=1}^{P}\frac{1}{N_p}\sum_{i=1}^{N_p}\prod_{c=1}^{C_p} T_{eval}\left(\langle code_{p,i}\rangle, x_{p,c}\right) > 1)$$

7 Analysis

We first walk through example code (shown in Figure 3) generated by the model with BeamSearchSampler (number of beams = 5). Problem 4292 describes N kinds of fruits, Fruit 1, ..., Fruit N at prices of $p_1, \ldots, p_N$ per item. The question asks for the minimum possible total price of fruits if we choose K kinds of fruit and buy one of each chosen kind. In the code from Figure 3, the model starts by parsing the input from standard input. Next, it sorts the prices, and finally returns the sum of the k least-expensive fruits via a while loop. Overall the final algorithm is fairly simple, but it is notable that the model is able to convert the text description to a syntactically valid, semantically correct solution.

8 Results

The below tables contain key metrics from our experiments.

Sampler	**Accuracy @ 1**	Accuracy @ 3	Accuracy @ 5	Accuracy @ 10	Accuracy @ AvgAll
GS	0.085647	N/A	N/A	N/A	0.085647
BSS (3 beams)	0.088945	0.124696	N/A	N/A	0.115977
BSS (5 beams)	0.088951	0.139586	0.160739	N/A	0.146356
BSS (10 beams)	0.074432	0.108706	0.131309	0.152542	0.133934
ES ($H_{max} = 6.0, b_{high} = 7, b_{low} = 3$)	**0.092508**	N/A	N/A	N/A	0.092508

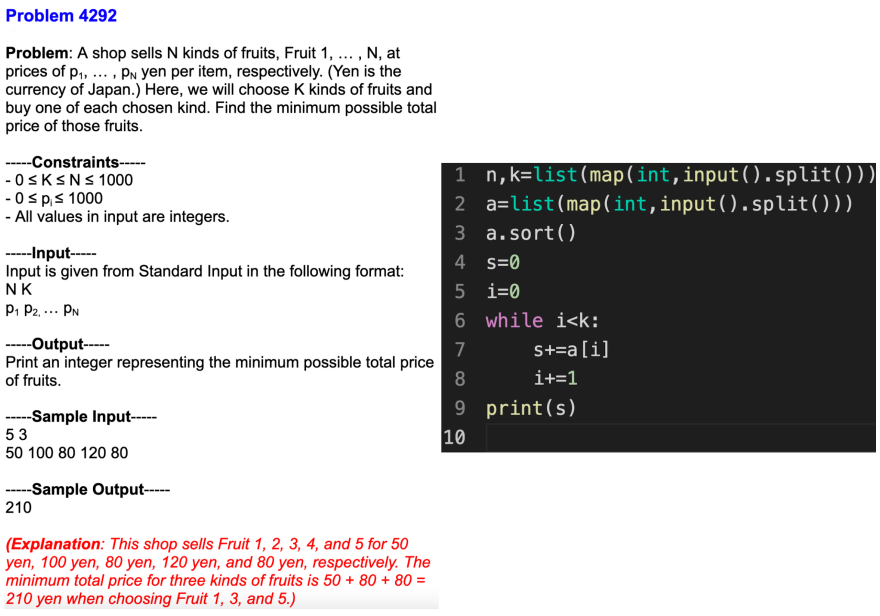
Table 1: Table of mean accuracies. N/A for accuracy @ k means that we generated fewer than k solutions. GS = GreedySampler, BSS = BeamSearchSampler, ES = EntropySampler.

Sampler	NumSolved @ 1	Strict Accuracy @ 1	NumSolved @ Best	Problem Set Coverage
GS	17	1.7%	17	1.7%
BSS (3 beams)	19	1.9%	25	2.5%
BSS (5 beams)	**22**	**2.2%**	**31**	**3.1%**
BSS (10 beams)	18	1.8%	**31**	**3.1%**
ES ($H_{max} = 6.0, b_{high} = 7, b_{low} = 3$)	20	2.0%	20	2.0%

Table 2: Table of number of problems solved. GS = GreedySampler, BSS = BeamSearchSampler, ES = EntropySampler.

Sampler	Timeout Rate
GS	0.16362
BSS (3 beams)	0.12263
BSS (5 beams)	0.15163
BSS (10 beams)	0.34177
ES ($H_{max} = 6.0, b_{high} = 7, b_{low} = 3$)	**0.03963**

Table 3: Table of timeout rates. GS = GreedySampler, BSS = BeamSearchSampler, ES = EntropySampler.



Problem 4292

Problem: A shop sells N kinds of fruits, Fruit 1, … , N, at prices of p_1, … , p_N yen per item, respectively. (Yen is the currency of Japan.) Here, we will choose K kinds of fruits and buy one of each chosen kind. Find the minimum possible total price of those fruits.

-----Constraints-----
- 0 ≤ K ≤ N ≤ 1000
- 0 ≤ p_i ≤ 1000
- All values in input are integers.

-----Input-----
Input is given from Standard Input in the following format:
N K
p_1 p_2 … p_N

-----Output-----
Print an integer representing the minimum possible total price of fruits.

-----Sample Input-----
5 3
50 100 80 120 80

-----Sample Output-----
210

***(Explanation**: This shop sells Fruit 1, 2, 3, 4, and 5 for 50 yen, 100 yen, 80 yen, 120 yen, and 80 yen, respectively. The minimum total price for three kinds of fruits is 50 + 80 + 80 = 210 yen when choosing Fruit 1, 3, and 5.)*

```
n,k=list(map(int,input().split()))
a=list(map(int,input().split()))
a.sort()
s=0
i=0
while i<k:
    s+=a[i]
    i+=1
print(s)
```

Figure 3: Sample problem and sample result generated by BeamSearchSampler (num beams = 5).

Section 7: Analysis – continued. As we can see in Table 1, EntropySampler reported the highest accuracy among all sampling methods. Also, unlike our expectation, BeamSearchSampler with 10 beams resulted in lower accuracy than BeamSearchSampler with 3 beams and 5 beams. GreedySampler reported a low accuracy as well, and this result is expected as a greedy algorithm is not expected to be optimal. As is expected from the definition, accuracy @ k goes up as we increase k.

Table 2 shows that BeamSearch with 5 beams reported the highest number of fully solved problems not only with its first run, but also with its best run. 22 of the first solutions generated by BeamSearch with 5 beam passed all test cases of 22 corresponding problems, resulting in 2.2% strict accuracy. Also, as expected, problem set coverage value is high when we generate more solutions since a problem is considered to be solved if at least one of the solutions passed all test cases.

Table 3 reports the timeout rate of each method. Here we observe that EntropySampler outperforms all other methods with a timeout rate of 0.039. Another result we see is that the timeout rate increases as we add the number of beams. This could reflect a limitation of beam search in which tokens generated from the less reliable tail of the distribution negatively affects the quality of resulting sequences, as suggested in (Holtzman et al., 2020).

9 Conclusion

In this project, we experimented with different sampling algorithms for a GPT-2 language model trained to generate Python code. We tested out greedy sampling, beam search with different numbers of beams, and entropy-based sampling, where the entropy is used to dynamically vary the number of beams. Overall we found that EntropySampler performed best on the test case accuracy metric, BeamSearchSampler with 5 beams had the best "NumSolved@1" value and was tied for the highest "NumSolved@Best" value with BeamSearchSampler with 10 beams.

Given that EntropySampler performed the best on test case accuracy, it appears to be a good sampling method for program synthesis. However, the improvement in test case accuracy was fairly small: EntropySampler has an Accuracy@1 of 0.925 while the next best sampler, BeamSearchSampler with 5 beams, had an accuracy of 0.890. Further experiments or modifications might need to be implemented to substantially improve over BeamSearchSampler. We set the number of beams to be at most 10 for BeamSearchSampler and $b_{high} = 7$ for EntropySampler. In many applications, it may be acceptable if the synthesized program takes a long time to generate. Therefore, increasing the number of beams even further can be a way to get even better performance, when more powerful compute resources are available.

Overall, the GPT-2 model used in this work performed relatively well, solving up to 2.2% of the introductory problems in the APPS dataset. While this is not a very large percentage in absolute terms, it demonstrates that language models show promise on this task, and could perform better when scaled up and pretrained on larger datasets. This subfield of research could enable interesting new technologies like machine-learning assisted autocomplete in the IDE, as well as machine-learning assisted code generation to solve complex programming tasks.

Known Project Limitations

A limitation of this project is that the EntropySampler uses a simple policy where we switch between two values for the number of beams based on the value of the discrete entropy H. This was chosen somewhat arbitrarily, reflecting the intuition that we want to sample more candidates when the model is more uncertain about its predictions. One could consider other policies that are more complicated, such as setting the number of beams based on a more complex function of the entropy.

We are using GPT2-1.5B for these experiments, which was the largest model we could fit on our GPU (NVIDIA RTX 3090). The current state of the art reported in (Chen et al., 2021) is achieved by a larger model (GPT-3). It is unclear whether the findings reported in this analysis would generalize to a language model that is 2 orders of magnitude larger such as GPT-3.

Since the model is pre-trained on publicly available code from Github, users should use caution when deploying this model for real-world use cases. For example, it is possible that code generated by this model has bugs or security issues. Thus, our recommendation is that code generated by this model should be audited by humans before being used for any downstream application.

Authorship Statement

Adithya Ganesh (AG) and Minjune Hwang (MH) adapted the open source code in `https://github.com/hendrycks/apps` to write the model inference loop. AG and MH wrote the paper together.

AG implemented the EntropySampler to dynamically vary the number of beams based on the entropy of the model. AG wrote code to execute generated Python programs in subprocesses and check their correctness.

MH implemented the code to calculate key metrics described in Section 6, such as test case accuracy, problem set coverage, and timeout rate. MH parallelized the inference loop to run across multiple machines to speed up our experimentation.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Dawn Drain, Chen Wu, Alexey Svyatkovskiy, and Neel Sundaresan. 2021. Generating bug-fixes using pretrained transformers. In *MAPS@PLDI 2021: Proceedings of the 5th ACM SIGPLAN International Symposium on Machine Programming, Virtual Event, Canada, 21 June, 2021*, pages 1–8. ACM.

Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. Codebert: A pre-trained model for programming and natural languages. In *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 1536–1547. Association for Computational Linguistics.

Daya Guo, Alexey Svyatkovskiy, Jian Yin, Nan Duan, Marc Brockschmidt, and Miltiadis Allamanis. 2022. Learning to complete code with sketches. In *International Conference on Learning Representations*.

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring coding challenge competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Ilya Kulikov, Alexander H. Miller, Kyunghyun Cho, and Jason Weston. 2018. Importance of a search strategy in neural dialogue modelling. *CoRR*, abs/1811.00907.

Jiwei Li and Dan Jurafsky. 2016. Mutual information and diverse decoding improve neural machine translation. *CoRR*, abs/1601.00372.

Yujia Li, David H. Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. 2022. Competition-level code generation with alphacode. *CoRR*, abs/2203.07814.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Yik-Cheung Tam. 2020. Cluster-based beam search for pointer-generator chatbot grounded by knowledge. *Comput. Speech Lang.*, 64:101094.

Michele Tufano, Dawn Drain, Alexey Svyatkovskiy, Shao Kun Deng, and Neel Sundaresan. 2020. Unit test case generation with transformers. *CoRR*, abs/2009.05617.

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. 2016. Diverse beam search: Decoding diverse solutions from neural sequence models. *CoRR*, abs/1610.02424.

A Appendix

Here we include various generated solutions that passed all of the test cases.

B Problem 4136

B.1 Problem Description

```
A + B is often used as an example of the easiest problem possible to show some
contest platform. However, some scientists have observed that sometimes this
problem is not so easy to get accepted. Want to try?


-----Input-----

The input contains two integers a and b (0   a, b   10^3), separated by a
single space.


-----Output-----

Output the sum of the given integers.


-----Examples-----
Input
5 14

Output
19

Input
381 492

Output
873
```

B.2 Generated Code

```python
a, b = map(int, input().split())
print(a + b)
```

C Problem 4213

C.1 Problem Description

```
You are given an integer sequence A of length N.
Find the maximum absolute difference of two elements (with different indices)
in A.

-----Constraints-----
 - 2 \leq N \leq 100
 - 1 \leq A_i \leq 10^9
 - All values in input are integers.
```

```
-----Input-----
Input is given from Standard Input in the following format:
N
A_1 A_2 ... A_N

-----Output-----
Print the maximum absolute difference of two elements (with different indices)
in A.

-----Sample Input-----
4
1 4 6 3

-----Sample Output-----
5

The maximum absolute difference of two elements is A_3-A_1=6-1=5.
```

C.2 Generated Code

```python
N=int(input())
A=list(map(int,input().split()))
A.sort()
print(A[-1]-A[0])
```

D Problem 4238

D.1 Problem Description

```
An integer N is a multiple of 9 if and only if the sum of the digits in the
decimal representation of N is a multiple of 9.
Determine whether N is a multiple of 9.

-----Constraints-----
 - 0 \leq N < 10^{200000}
 - N is an integer.

-----Input-----
Input is given from Standard Input in the following format:
N

-----Output-----
If N is a multiple of 9, print Yes; otherwise, print No.

-----Sample Input-----
123456789

-----Sample Output-----
Yes

The sum of these digits is 1+2+3+4+5+6+7+8+9=45, which is a multiple of 9, so
```

```
123456789 is a multiple of 9.
```

D.2 Generated Code

```python
N = input()

sum = 0

for each in N:
    sum += int(each)

if(sum%9 == 0):
    print("Yes")
else:
    print("No")
```

E Problem 4246

E.1 Problem Description

```
You will be given a string S of length 3 representing the weather forecast for
three days in the past.
The i-th character (1 \leq i \leq 3) of S represents the forecast for the i-th
day. S, C, and R stand for sunny, cloudy, and rainy, respectively.
You will also be given a string T of length 3 representing the actual weather
on those three days.
The i-th character (1 \leq i \leq 3) of S represents the actual weather on the
i-th day. S, C, and R stand for sunny, cloudy, and rainy, respectively.
Print the number of days for which the forecast was correct.

-----Constraints-----
 - S and T are strings of length 3 each.
 - S and T consist of S, C, and R.

-----Input-----
Input is given from Standard Input in the following format:
S
T

-----Output-----
Print the number of days for which the forecast was correct.

-----Sample Input-----
CSS
CSR

-----Sample Output-----
2

 - For the first day, it was forecast to be cloudy, and it was indeed cloudy.
 - For the second day, it was forecast to be sunny, and it was indeed sunny.
 - For the third day, it was forecast to be sunny, but it was rainy.
```

```
Thus, the forecast was correct for two days in this case.
```

E.2 Generated Code

```python
# cook your dish here

s = input()
t = input()

c = 0
for i in range(3):
    if s[i] == t[i]:
        c += 1

print(c)
```

F Problem 4253

F.1 Problem Description

```
It is known that the area of a regular dodecagon inscribed in a circle of
radius a is 3a^2.
Given an integer r, find the area of a regular dodecagon inscribed in a circle
of radius r.

-----Constraints-----
 - 1 \leq r \leq 100
 - r is an integer.

-----Input-----
Input is given from Standard Input in the following format:
r

-----Output-----
Print an integer representing the area of the regular dodecagon.

-----Sample Input-----
4

-----Sample Output-----
48

The area of the regular dodecagon is 3 \times 4^2 = 48.
```

F.2 Generated Code

```python
r=int(input())
if(r>0):
    print(3*(r**2))
else:
    print(0)
```

G Problem 4257

G.1 Problem Description

```
Compute A \times B.

-----Constraints-----
 - 1 \leq A \leq 100
 - 1 \leq B \leq 100
 - All values in input are integers.

-----Input-----
Input is given from Standard Input in the following format:
A B

-----Output-----
Print the value A \times B as an integer.

-----Sample Input-----
2 5

-----Sample Output-----
10

We have 2 \times 5 = 10.
```

G.2 Generated Code

```python
# cook your dish here
a,b=map(int,input().split())
print(a*b)
```

H Problem 4267

H.1 Problem Description

```
You will turn on the air conditioner if, and only if, the temperature of the
room is 30 degrees Celsius or above.
The current temperature of the room is X degrees Celsius. Will you turn on the
air conditioner?

-----Constraints-----
 - -40 \leq X \leq 40
 - X is an integer.

-----Input-----
Input is given from Standard Input in the following format:
X

-----Output-----
Print Yes if you will turn on the air conditioner; print No otherwise.

-----Sample Input-----
```

```
25

-----Sample Output-----
No
```

H.2 Generated Code

```python
x=int(input())
if(x>=30):
    print("Yes")
else:
    print("No")
```

I Problem 4292

I.1 Problem Description

```
A shop sells N kinds of fruits, Fruit 1, \ldots, N, at prices of p_1, \ldots,
p_N yen per item, respectively. (Yen is the currency of Japan.)
Here, we will choose K kinds of fruits and buy one of each chosen kind. Find
the minimum possible total price of those fruits.

-----Constraints-----
 - 1 \leq K \leq N \leq 1000
 - 1 \leq p_i \leq 1000
 - All values in input are integers.

-----Input-----
Input is given from Standard Input in the following format:
N K
p_1 p_2 \ldots p_N

-----Output-----
Print an integer representing the minimum possible total price of fruits.

-----Sample Input-----
5 3
50 100 80 120 80

-----Sample Output-----
210

This shop sells Fruit 1, 2, 3, 4, and 5 for 50 yen, 100 yen, 80 yen, 120 yen,
and 80 yen, respectively.
The minimum total price for three kinds of fruits is 50 + 80 + 80 = 210 yen
when choosing Fruit 1, 3, and 5.
```

I.2 Generated Code

```python
n,k=list(map(int,input().split()))
a=list(map(int,input().split()))
a.sort()
```

```
s=0
i=0
while i<k:
    s+=a[i]
    i+=1
print(s)
```

J Problem 4294

J.1 Problem Description

```
X and A are integers between 0 and 9 (inclusive).
If X is less than A, print 0; if X is not less than A, print 10.

-----Constraints-----
 - 0 \leq X, A \leq 9
 - All values in input are integers.

-----Input-----
Input is given from Standard Input in the following format:
X A

-----Output-----
If X is less than A, print 0; if X is not less than A, print 10.

-----Sample Input-----
3 5

-----Sample Output-----
0

3 is less than 5, so we should print 0.
```

J.2 Generated Code

```
# cook your dish here
x,a=map(int,input().split())
if(x<a):
    print(0)
else:
    print(10)
```

K Problem 4326

K.1 Problem Description

```
There are N students in a school.
We will divide these students into some groups, and in each group they will
discuss some themes.
You think that groups consisting of two or less students cannot have an
effective discussion, so you want to have as many groups consisting of three
or more students as possible.
```

```
Divide the students so that the number of groups consisting of three or more
students is maximized.

-----Constraints-----
 - 1 \leq N \leq 1000
 - All input values are integers.

-----Input-----
Input is given from Standard Input in the following format:
N

-----Output-----
If you can form at most x groups consisting of three or more students, print x.

-----Sample Input-----
8

-----Sample Output-----
2

For example, you can form a group of three students and another of five
students.
```

K.2 Generated Code

```python
n=int(input())
if n<3:
    print(0)
elif n==3:
    print(1)
else:
    print(n//3)
```

L Problem 4331

L.1 Problem Description

```
Given is a three-digit integer N. Does N contain the digit 7?
If so, print Yes; otherwise, print No.

-----Constraints-----
 - 100 \leq N \leq 999

-----Input-----
Input is given from Standard Input in the following format:
N

-----Output-----
If N contains the digit 7, print Yes; otherwise, print No.

-----Sample Input-----
117
```

```
-----Sample Output-----
Yes

117 contains 7 as its last digit.
```

L.2 Generated Code

```python
# cook your dish here
n=input()
if "7" in n:
    print("Yes")
else:
    print("No")
```

M Problem 4353

M.1 Problem Description

```
As a New Year's gift, Dolphin received a string s of length 19.

The string s has the following format: [five lowercase English letters],[seven
lowercase English letters],[five lowercase English letters].

Dolphin wants to convert the comma-separated string s into a space-separated
string.

Write a program to perform the conversion for him.

-----Constraints-----
 - The length of s is 19.
 - The sixth and fourteenth characters in s are ,.
 - The other characters in s are lowercase English letters.

-----Input-----
The input is given from Standard Input in the following format:
s

-----Output-----
Print the string after the conversion.

-----Sample Input-----
happy,newyear,enjoy

-----Sample Output-----
happy newyear enjoy

Replace all the commas in happy,newyear,enjoy with spaces to obtain happy
newyear enjoy.
```

M.2 Generated Code

```
s=input()
s=s.split(',')
s=list(map(str,s))
print(' '.join(s))
```

N Problem 4388

N.1 Problem Description

```
Cat Snuke is learning to write characters.
Today, he practiced writing digits 1 and 9, but he did it the other way around.
You are given a three-digit integer n written by Snuke.
Print the integer obtained by replacing each digit 1 with 9 and each digit 9
with 1 in n.

-----Constraints-----
 - 111 \leq n \leq 999
 - n is an integer consisting of digits 1 and 9.

-----Input-----
Input is given from Standard Input in the following format:
n

-----Output-----
Print the integer obtained by replacing each occurrence of 1 with 9 and each
occurrence of 9 with 1 in n.

-----Sample Input-----
119

-----Sample Output-----
991

Replace the 9 in the ones place with 1, the 1 in the tens place with 9 and the
1 in the hundreds place with 9. The answer is 991.
```

N.2 Generated Code

```
n=input()
x=list(n)
for i in range(len(x)):
    if x[i]=='1':
        x[i]='9'
    else:
        x[i]='1'
print(''.join(x))
```

O Problem 4494

O.1 Problem Description

```
AtCoder Inc. holds a contest every Saturday.
There are two types of contests called ABC and ARC, and just one of them is
held at a time.
The company holds these two types of contests alternately: an ARC follows an
ABC and vice versa.
Given a string S representing the type of the contest held last week, print the
string representing the type of the contest held this week.

-----Constraints-----
 - S is ABC or ARC.

-----Input-----
Input is given from Standard Input in the following format:
S

-----Output-----
Print the string representing the type of the contest held this week.

-----Sample Input-----
ABC

-----Sample Output-----
ARC

They held an ABC last week, so they will hold an ARC this week.
```

O.2 Generated Code

```python
# cook your dish here
s=input()
if s=='ABC':
    print("ARC")
else:
    print("ABC")
```

P Problem 4541

P.1 Problem Description

```
Given a lowercase English letter c, determine whether it is a vowel. Here,
there are five vowels in the English alphabet: a, e, i, o and u.

-----Constraints-----
 - c is a lowercase English letter.

-----Input-----
The input is given from Standard Input in the following format:
c

-----Output-----
If c is a vowel, print vowel. Otherwise, print consonant.
```

```
-----Sample Input-----
a

-----Sample Output-----
vowel

Since a is a vowel, print vowel.
```

P.2 Generated Code

```python
c=str(input())
if(c.lower() in ['a','e','i','o','u']):
    print('vowel')
else:
    print('consonant')
```

Q Problem 4564

Q.1 Problem Description

```
You are given a string S consisting of lowercase English letters. Determine
whether all the characters in S are different.

-----Constraints-----
 - 2   |S|   26, where |S| denotes the length of S.
 - S consists of lowercase English letters.

-----Input-----
Input is given from Standard Input in the following format:
S

-----Output-----
If all the characters in S are different, print yes (case-sensitive);
otherwise, print no.

-----Sample Input-----
uncopyrightable

-----Sample Output-----
yes
```

Q.2 Generated Code

```python
# cook your dish here
s = input()

if(len(s) == len(set(s))):
    print('yes')
else:
    print('no')
```

R Problem 4622

R.1 Problem Description

```
Given is a sequence of integers A_1, A_2, ..., A_N.
If its elements are pairwise distinct, print YES; otherwise, print NO.

-----Constraints-----
 - 2   N   200000
 - 1   A_i   10^9
 - All values in input are integers.

-----Input-----
Input is given from Standard Input in the following format:
N
A_1 ... A_N

-----Output-----
If the elements of the sequence are pairwise distinct, print YES; otherwise,
print NO.

-----Sample Input-----
5
2 6 1 4 5

-----Sample Output-----
YES

The elements are pairwise distinct.
```

R.2 Generated Code

```python
n=int(input())
a=list(map(int,input().split()))
if len(set(a))==n:
    print("YES")
else:
    print("NO")
```

S Problem 4649

S.1 Problem Description

```
The only difference between easy and hard versions is the size of the input.

You are given a string $s$ consisting of $n$ characters, each character is 'R',
'G' or 'B'.

You are also given an integer $k$. Your task is to change the minimum number of
characters in the initial string $s$ so that after the changes there will be a
string of length $k$ that is a substring of $s$, and is also a substring of the
infinite string "RGBRGBRGB ...".
```

A string a is a substring of string b if there exists a positive integer i such that $a_1 = b_i$, $a_2 = b_{i + 1}$, $a_3 = b_{i + 2}$, ..., $a_{|a|} = b_{i + |a| - 1}$. For example, strings "GBRG", "B", "BR" are substrings of the infinite string "RGBRGBRGB ..." while "GR", "RGR" and "GGG" are not.

You have to answer q independent queries.

-----Input-----

The first line of the input contains one integer q ($1 \le q \le 2000$) — the number of queries. Then q queries follow.

The first line of the query contains two integers n and k ($1 \le k \le n \le 2000$) — the length of the string s and the length of the substring.

The second line of the query contains a string s consisting of n characters 'R', 'G' and 'B'.

It is guaranteed that the sum of n over all queries does not exceed 2000 ($\sum n \le 2000$).

-----Output-----

For each query print one integer — the minimum number of characters you need to change in the initial string s so that after changing there will be a substring of length k in s that is also a substring of the infinite string "RGBRGBRGB ...".

-----Example-----
Input
3
5 2
BGGGG
5 3
RBRGR
5 5
BBBRR

Output
1
0
3

-----Note-----

In the first example, you can change the first character to 'R' and obtain

the substring "RG", or change the second character to 'R' and obtain "BR", or change the third, fourth or fifth character to 'B' and obtain "GB".

In the second example, the substring is "BRG".

S.2 Generated Code

```python
import sys
input = sys.stdin.readline
q = int(input())
for _ in range(q):
    n, k = map(int, input().split())
    word = input()
    ans = 0
    for col in ["RGB", "GBR", "BRG"]:
        cnt = 0
        for i in range(k):
            if word[i] == col[i%3]:
                cnt += 1
        mx = cnt
        for i in range(n-k):
            if word[i+k] == col[(i+k)%3]:
                cnt += 1
            if word[i] == col[i%3]:
                cnt -= 1
            if cnt > mx:
                mx = cnt
        ans = max(ans, mx)
    print(k-ans)
```

T Problem 4686

T.1 Problem Description

```
Let w be a string consisting of lowercase letters.
We will call w beautiful if the following condition is satisfied:
 - Each lowercase letter of the English alphabet occurs even number of times in
w.
You are given the string w. Determine if w is beautiful.

-----Constraints-----
 - 1 \leq |w| \leq 100
 - w consists of lowercase letters (a-z).

-----Input-----
The input is given from Standard Input in the following format:
w

-----Output-----
Print Yes if w is beautiful. Print No otherwise.

-----Sample Input-----
```

```
abaccaba

-----Sample Output-----
Yes

a occurs four times, b occurs twice, c occurs twice and the other letters occur
zero times.
```

T.2 Generated Code

```python
w=input()
w=list(w)
cnt=0
for i in w:
    cnt=cnt+w.count(i)%2

if cnt==0:
    print("Yes")
else:
    print("No")
```

U Problem 4710

U.1 Problem Description

```
Smeke has decided to participate in AtCoder Beginner Contest (ABC) if his
current rating is less than 1200, and participate in AtCoder Regular Contest
(ARC) otherwise.
You are given Smeke's current rating, x. Print ABC if Smeke will participate in
ABC, and print ARC otherwise.

-----Constraints-----
 - 1   x   3{,}000
 - x is an integer.

-----Input-----
The input is given from Standard Input in the following format:
x

-----Output-----
Print the answer.

-----Sample Input-----
1000

-----Sample Output-----
ABC

Smeke's current rating is less than 1200, thus the output should be ABC.
```

U.2 Generated Code

```
x=int(input())
if x<1200:
    print("ABC")
else:
    print("ARC")
```

V Problem 4714

V.1 Problem Description

```
Find the number of palindromic numbers among the integers between A and B
(inclusive).
Here, a palindromic number is a positive integer whose string representation in
base 10 (without leading zeros) reads the same forward and backward.

-----Constraints-----
 - 10000 \leq A \leq B \leq 99999
 - All input values are integers.

-----Input-----
Input is given from Standard Input in the following format:
A B

-----Output-----
Print the number of palindromic numbers among the integers between A and B
(inclusive).

-----Sample Input-----
11009 11332

-----Sample Output-----
4

There are four integers that satisfy the conditions: 11011, 11111, 11211 and
11311.
```

V.2 Generated Code

```
# cook your dish here
a,b=map(int,input().split())
c=0
for i in range(a,b+1):
    d=str(i)
    if(d==d[::-1]):
        c+=1
print(c)
```

W Problem 4715

W.1 Problem Description

```
AtCoDeer the deer recently bought three paint cans.
The color of the one he bought two days ago is a, the color of the one he
bought yesterday is b, and the color of the one he bought today is c.
Here, the color of each paint can is represented by an integer between 1 and
100, inclusive.
Since he is forgetful, he might have bought more than one paint can in the same
color.
Count the number of different kinds of colors of these paint cans and tell him.

-----Constraints-----
 - 1 a,b,c 100

-----Input-----
The input is given from Standard Input in the following format:
a b c

-----Output-----
Print the number of different kinds of colors of the paint cans.

-----Sample Input-----
3 1 4

-----Sample Output-----
3

Three different colors: 1, 3, and 4.
```

W.2 Generated Code

```python
# cook your dish here
a=list(map(int,input().split()))
print(len(set(a)))
```

X Problem 4785

X.1 Problem Description

```
A known problem with some microphones is the “hissing s”. That is, sometimes
the sound of the letter s is particularly pronounced; it stands out from the
rest of the word in an unpleasant way.

Of particular annoyance are words that contain the letter s twice in a row.
Words like amiss, kiss, mississippi and even hiss itself.

-----Input-----
The input contains a single string on a single line. This string consists of
only lowercase letters (no spaces) and has between $1$ and $30$ characters.

-----Output-----
Output a single line. If the input string contains two consecutive occurrences
of the letter s, then output hiss. Otherwise, output no hiss.
```

```
-----Examples-----
Sample Input 1:
amiss
Sample Output 1:
hiss

Sample Input 2:
octopuses
Sample Output 2:
no hiss
```

X.2 Generated Code

```python
# cook your dish here
s=input()
for i in range(len(s)-1):
    if s[i]==s[i+1]:
        print("hiss")
        break
else:
    print("no hiss")
```

Y Problem 4826

Y.1 Problem Description

```
Alice and Bob are playing a new game of stones. There are $N$ stones placed on
the ground, forming a sequence. The stones are labeled from $1$ to $N$.

Alice and Bob in turns take exactly two consecutive stones on the ground until
there are no consecutive stones on the ground. That is, each player can take
stone $i$ and stone $i+1$, where $1 \leq i \leq N - 1$. If the number of stone
left is odd, Alice wins. Otherwise, Bob wins.

Assume both Alice and Bob play optimally and Alice plays first, do you know who
the winner is?

-----Input-----
The input contains an integer $N$ $(1 \leq N \leq 10000000)$, the number of
stones.

-----Output-----
Output the winner, “Alice” or “Bob” (without the quotes), on a line.

-----Examples-----
Sample Input 1:
1
Sample Output 1:
Alice

Sample Input 2:
2
```

```
Sample Output 2:
Bob
```

Y.2 Generated Code

```python
n=int(input())
if n%2==0:
    print('Bob')
else:
    print('Alice')
```