

传统算法问答模型

1 引言

1.1 项目介绍

传统算法问答模型/Traditional algorithmic question-answering model(TAQAModel)是一个专门为算法竞赛选手提供算法知识的智能交流模型。TAQAModel区别于基础对话大模型(e.g. ChatGpt4, LGM-4)，它可以提供完全正确的算法模板(e.g. 线段树模板，树上倍增模板)，以及根据正确的模板与用户进行算法知识上的交流。该项目为算法竞赛选手提供了一个强大的辅助工具，帮助他们更好地理解和应用复杂的算法，提高竞赛成绩。

1.2 项目意义

这里引用一下知乎去年用gpt-4辅助参加codeforces中一场比赛的文章：[使用gpt4辅助做Codeforces - 知乎\(zhihu.com\)](#)。

通过该文章发现,让大模型自己解决算法问题，优化代码时间复杂度，目前看来是不实际的。但是如果让大模型辅助选手、程序员去解决算法问题是可以做到的。

作为大学期间参加程序设计大赛(acm/icpc)的一名选手,我深知在解决算法题时找不到相应算法模板，即使找到算法模板，当前大模型给出的算法代码正确性不能得到保证，这些难处。所以借此机会做了一个关于算法问答的模型。根据一名算法竞赛选手的要求，如果该模型可以做到以下几点，这个项目就有存在的意义。

- 提供的算法模板非常精准
- 提供的算法模板注释齐全，解释清楚明了
- 可以根据选手提出的问题智能找到可以解决该问题算法
- 不仅仅可以提供算法模板，还可以与模型交流，学到拓展知识。

这次项目也基本做到上述4点功能，对于我来说它是能拿来使用的，可以帮助我解决算法问题的。

1.3 项目特性

- ✓ 采用 Langchain 作为核心框架，充分利用其模块化和可扩展的特点，确保系统可以灵活地适应不同的算法问答需求。
- ✓ 采用了Sqlite3轻量级关系型数据库, FAISS向量数据库相结合的方法，既做到了持久化存储，节省 token，又做到了提升模型的准确性和响应速度。
- ✓ 通过 Loguru 进行日志管理，确保在开发和运行过程中可以高效地记录和追踪系统的状态和问题。
- ✓ 通过 YAML 文件参数灵活配置。
- ✓ 使用 Gradio实现图形用户界面 (GUI) 以便更易于使用，同时灵活使用Gradio中Block组件以及更改相关css，美化了界面
- ✓ 通过Gradio, 创建一个网络服务或 API，以便在网络应用中使用。
- ✓ 模块化和面向对象的设计，易于定制和扩展。
- ✓ 支持 ChatGLM 和 OpenAI 模型。
- ✓ 文本中去除噪音, 使模型更容易捕捉到有意义的模式和关系。

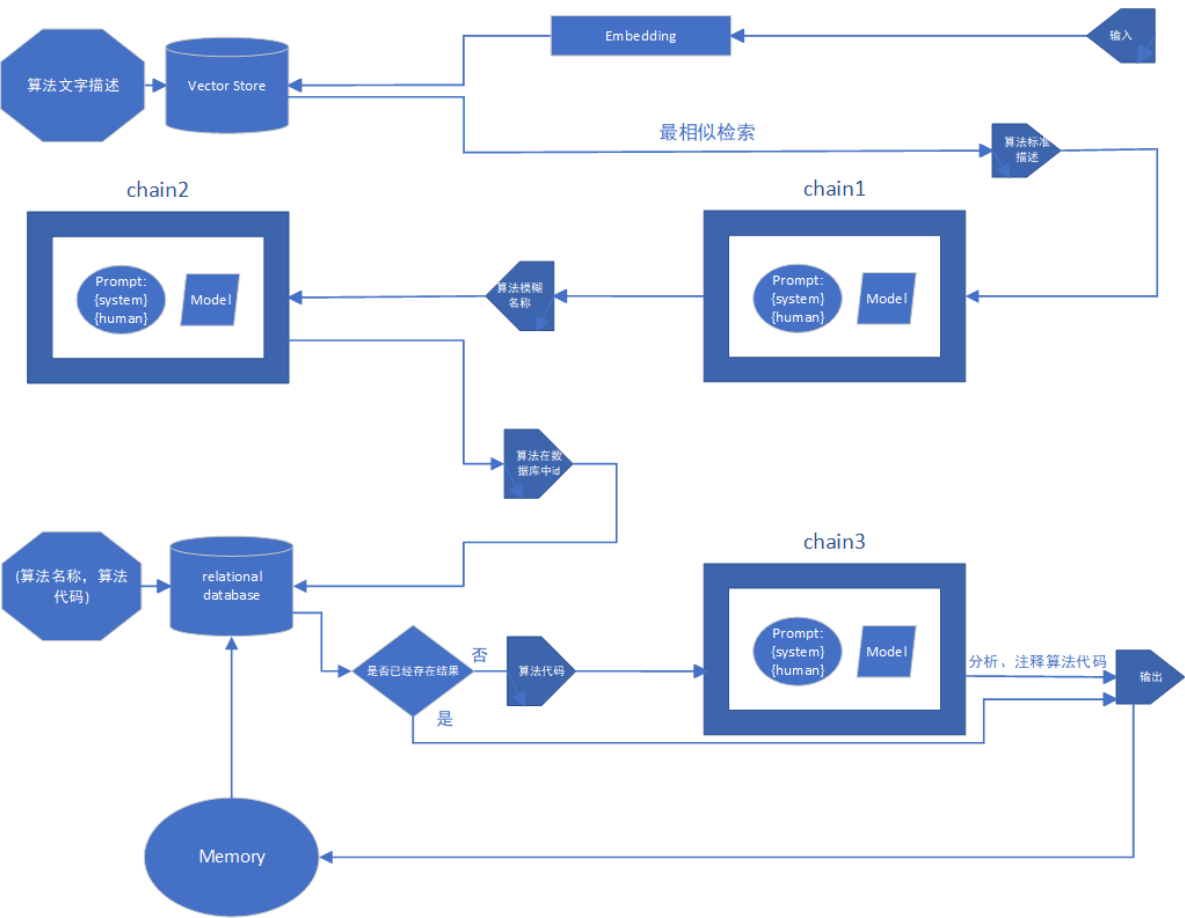
1.4 开发环境

1	python	3.11.5
2	langchain	0.2.0
3	langchain-community	0.2.6
4	langchain-core	0.2.12
5	langchain-openai	0.1.14
6	langchain-text-splitters	0.2.2
7	faiss-cpu	1.8.0.post1
8	gradio	4.37.2
9	pandas	2.0.3
10	numpy	1.24.3
11	loguru	0.7.2
12	sentence-transformers	2.3.0
13	jinja2	3.1.2

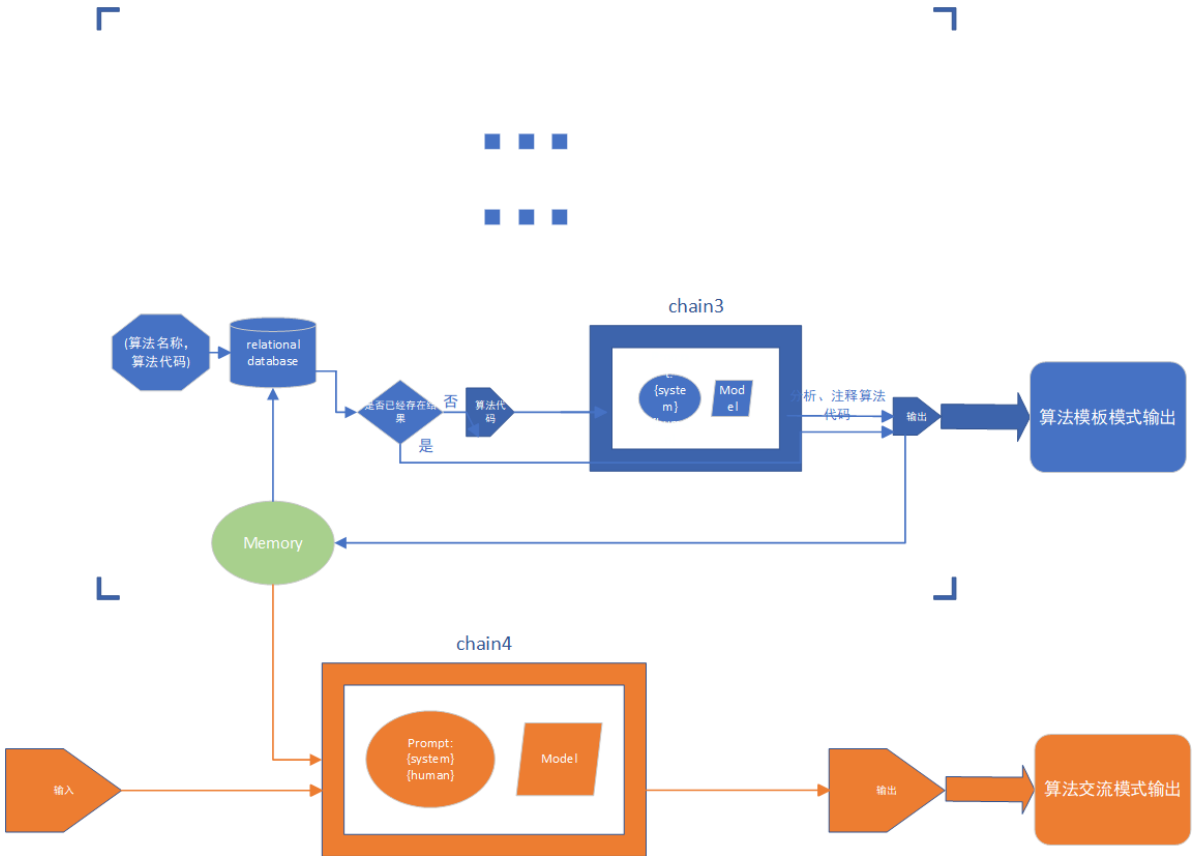
2 项目设计

2.1 系统流程图

算法模板模式流程图

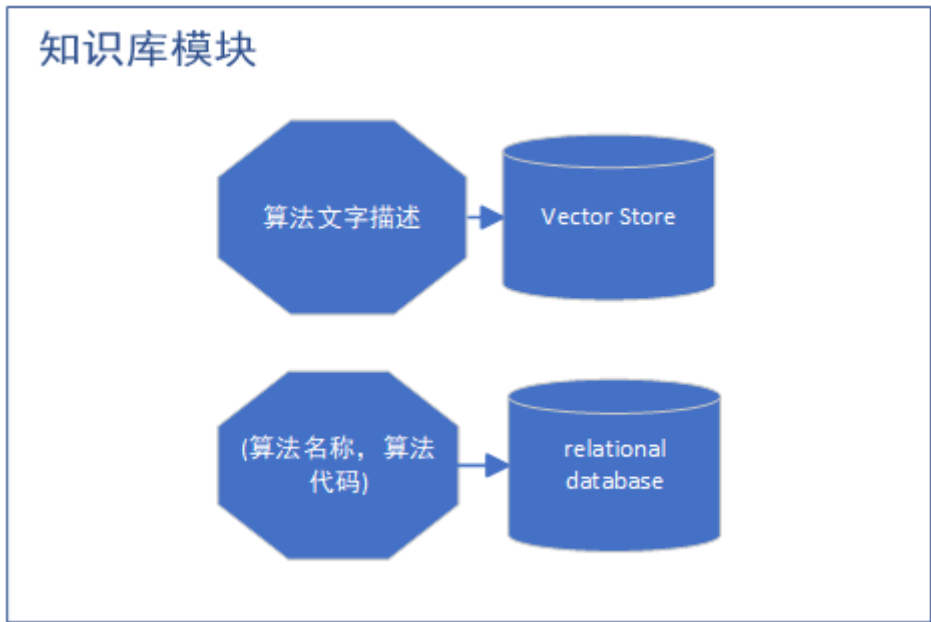


算法交流模式流程图



2.2 系统模块分析

2.2.1 知识库模块



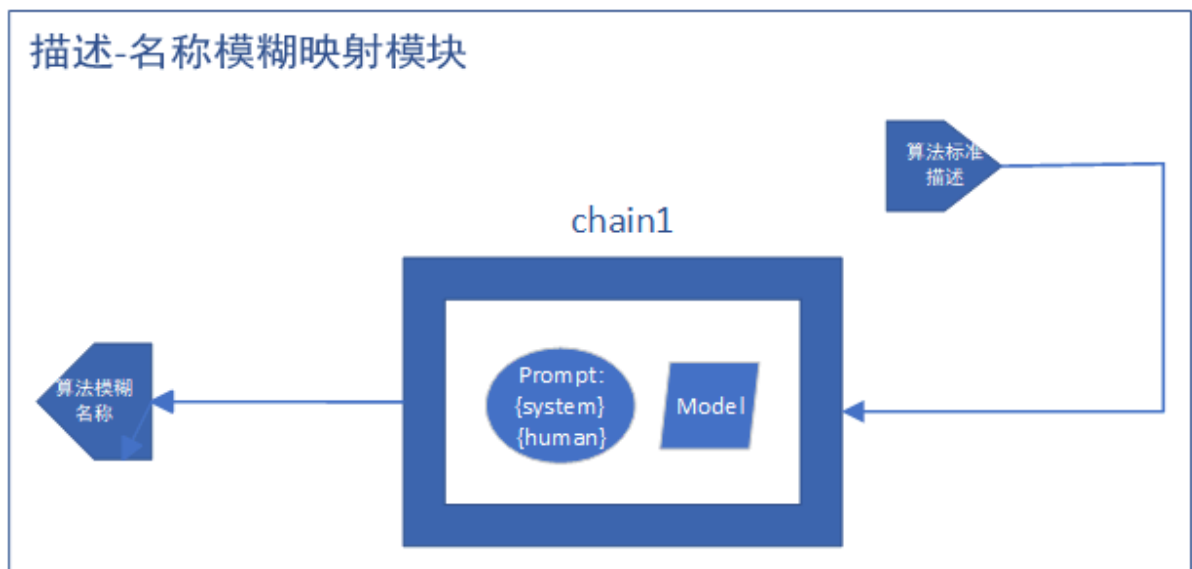
算法知识库分为算法名称，算法代码模板，算法描述。算法名称，算法代码模板存入关系型数据库中，算法描述存入向量数据库中。

2.2.2 用户描述分析模块



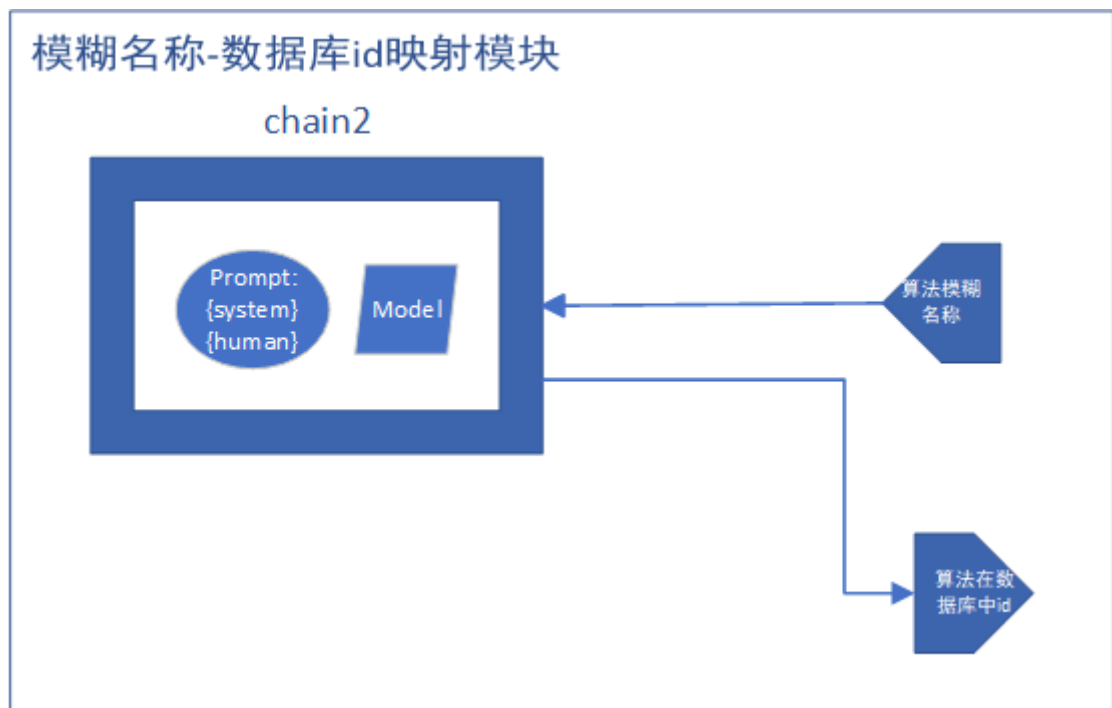
通过Embedding和向量数据库将用户描述，近似检索出最相似的一条算法描述并输出。

2.2.3 描述-名称模糊映射模块



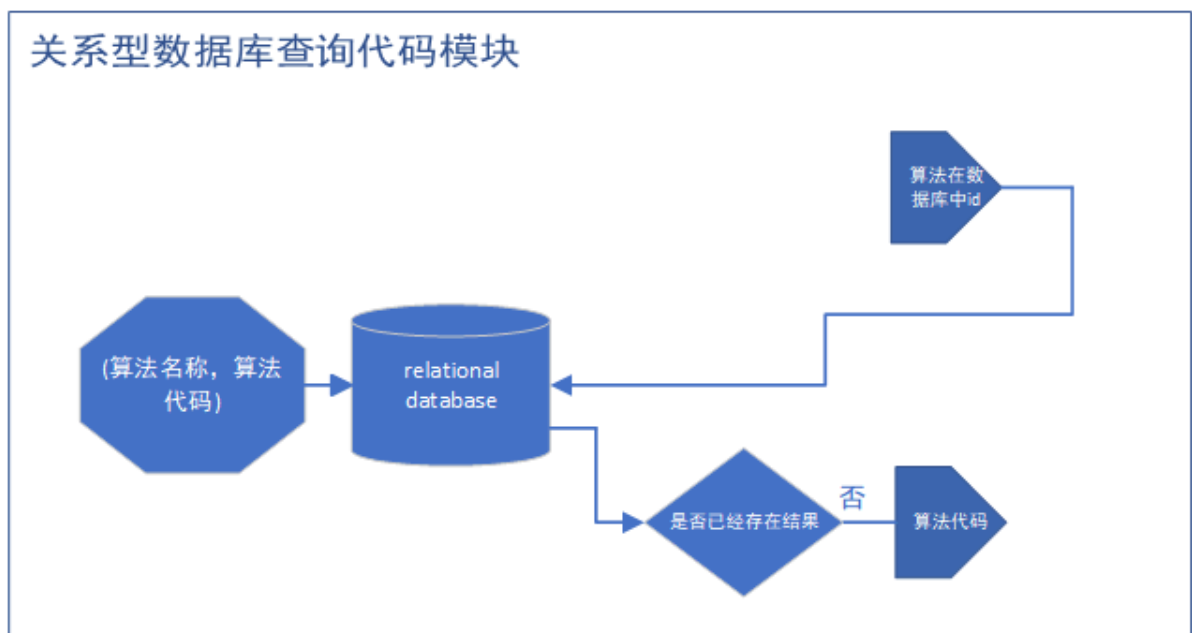
给予chain1中Model合适的prompt，通过chain2，将算法描述映射为算法模糊名称并输出

2.2.4 模糊名称-数据库id映射模块



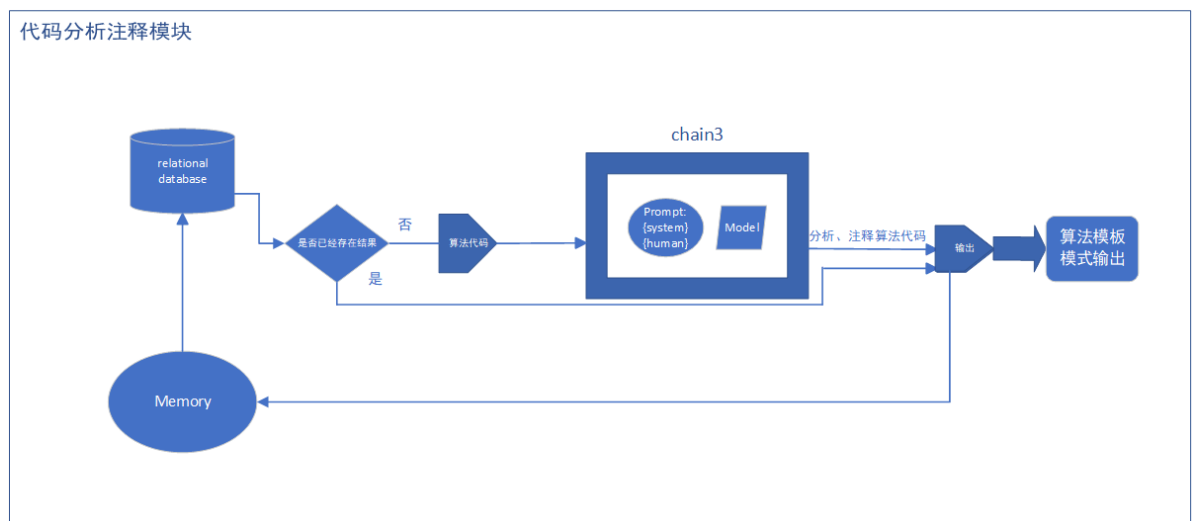
给予chain2中Model合适的prompt，通过chain2，将模糊的算法名称映射为真实算法名称对应数据库表中的id并输出

2.2.5 关系型数据库查询模块



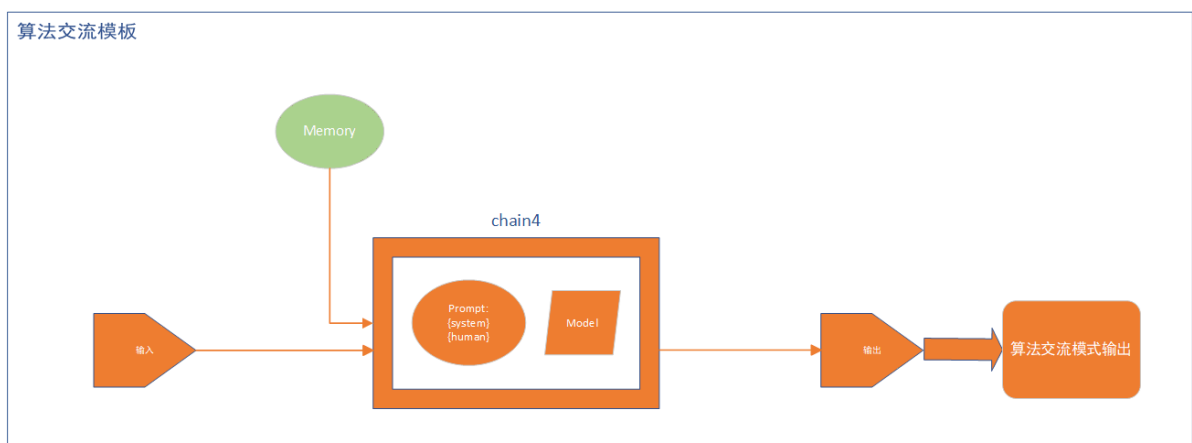
使用id, 通过关系型数据库查到算法代码模板

2.2.6 代码分析注释模块



给予chain3中Model合适的prompt, 通过chain3, 分析算法代码模板, 并输出。同时为了节省tokens, 将获得的结果存到memory中, 从memory中持久化存储到关系型数据库中。

2.2.7 算法交流模板



给予chain4中Model合适的prompt同时将Memory中之间的对话作为prompt，通过chain4，用户可以与TAQAModel进行对话交流。

2.3 遇到的关键问题与解决方法

1. 算法模板匹配不准确。用户忘记算法名称，通过对算法的描述不能找到相关算法模板
解决方法：知识库中，在仅有算法模板的基础上添加该算法的详细描述
2. 算法详细描述 + 算法模板的token太大，超出Embedding模型最大token数
解决方法1：将算法模板与算法详细描述分开存在关系型数据库和向量数据库中。
解决方法2：采用BAAI/bge-small-zh-v1.5模型（一开始采用的是Qianfan模型中的Embeddingv1）
3. 基础大模型通过算法描述得到的算法名称不稳定，导致在关系型数据库中通过算法名称为key查找算法代码时会出错
解决方法：串联两个chain，通过两次大模型的纠正，得到关系型数据库中算法名称对应的id，进而精准的找到算法模板
4. 当TAQAModel切换到算法交流模式时，没有之前记忆难以交流
解决方法：将之前的对话结果存到Memory(history)中，当切换为算法交流模式时，将memory中的内容作为prompt提供给chain

3 项目实现

3.1 前端图形化界面

```
1 import os
2 import gradio as gr
3
4 from utils import LOG
5 from utils import ChatBotConfig
6 from utils import NoiseRemoval
7 from chatbot import ChatBot
8 from sql import DataTODB
9
10 def init():
11     dir_path = os.path.dirname(os.path.abspath(__file__))
12     config = ChatBotConfig()
13     config.init(dir_path=dir_path)
14     global chatbot, noiseremoval, db
15     db = DataTODB()
16
17     chatbot = ChatBot(config.model_name, config.api_key, config.base_url)
18     noiseremoval = NoiseRemoval()
19
20     global slider_value_now, bot_model_now
21     slider_value_now = 0.1
22     bot_model_now = "算法模板模式"
23
```

```

24 def get_slider_value(slider_value):
25     global slider_value_now
26     slider_value_now = slider_value
27     LOG.debug(f"[slider_value_now]{slider_value_now}")
28     return slider_value
29
30 def get_bot_model(bot_model):
31     global bot_model_now
32     bot_model_now = bot_model
33     LOG.debug(f"[bot_model_now]{bot_model_now}")
34     return bot_model
35
36 def algorithm_chat(message, history):
37     try:
38         LOG.debug(f"[message]{message}")
39         LOG.debug(f"[history]{history}")
40         LOG.debug(f"[slider_value_now]{slider_value_now}")
41         LOG.debug(f"[bot_model_now]{bot_model_now}")
42
43         processed_message = noiseremoval.Do_NoiseRemoval(message)
44         LOG.debug(f"[message after noise removal]{processed_message}")
45
46         if bot_model_now == "算法模板模式":
47             result, statue, id = chatbot.CodeAnalysis(processed_message,
48                 slider_value_now)
49
50             if statue == 1:
51                 LOG.debug(f"[result]{result}\n[statue]{statue}\n[id]{id}")
52                 flag = db.Result_Online_ToDB_By_ID(id, result)
53                 if flag:
54                     LOG.debug("Insert successful")
55                 else:
56                     LOG.error("Insert failed")
57
58             return result
59
60         else:
61             return chatbot.AlgorithmExchange(processed_message, history)
62
63     except Exception as e:
64         LOG.error(f"Error invoking retrieval chain: {e}")
65         return "An error occurred. Please try again."
66
67 def launch_gradio():
68     slider = gr.Slider(minimum=0.05, maximum=1, step=0.05, value=0.1,
69         label="匹配阈值")
70     bot_model = gr.Dropdown(choices=["算法模板模式", "算法交流模式"], label="机
71         器人模式", value="算法模板模式")
72     slider_output = gr.Label(label="当前匹配阈值", value=0.1)
73     bot_model_output = gr.Label(label="当前机器人模式", value="算法模板模式")
74     # 创建 ChatInterface 实例
75     with gr.ChatInterface(
76         fn=algorithm_chat,
77         title="算法模板机器人(比赛ak机器人)",
78         chatbot=gr.Chatbot(height=800),
79         retry_btn=None,
80         undo_btn=None,
81         clear_btn=None,

```

```

79     theme="soft",
80     examples=["可以区间查询，区间修改的算法", "线段树区间查询", "可以快速判断字符串是否为子串"],
81     css="""
82     .gradio-container {
83         border: 4px solid #A0C4FF; /* 更浅的外边框 */
84         border-radius: 15px; /* 更大的圆角边框 */
85         padding: 30px; /* 内边距 */
86     }
87     """
88 ) as chat:
89     with gr.Row():
90         with gr.Column():
91             txt1 = gr.Markdown(
92                 """
93                 <span style='color: #6A5ACD; font-size: 18px;
background-color: #E0E7FF; padding: 10px; border-radius: 10px;'>【功能模块】
</span>
94                 """
95             )
96             gr.Interface(fn=get_slider_value, inputs=slider,
outputs=slider_output)
97             gr.Interface(fn=get_bot_model, inputs=bot_model,
outputs=bot_model_output)
98             with gr.Column():
99                 txt1 = gr.Markdown(
100                     """
101                     <span style='color: #6A5ACD; font-size: 18px;
background-color: #E0E7FF; padding: 10px; border-radius: 10px;'>【机器人使用说明】</span>
102                     """
103                 )
104                 with gr.Tab(label="匹配阈值"):
105                     txt2 = gr.Markdown(
106                         """
107                         <span style='font-size: 16px;'>【请将阈值调高】当
您需要让机器人专门匹配某个算法模板时/当机器人语无伦次时</span><br><br>
108                         <span style='font-size: 16px;'>【请将阈值调低】当
您不知道用算法的名称叫什么时/不知道用什么算法时/机器人匹配不到你需要的算法时</span>
109                         """
110                     )
111                     with gr.Tab(label="机器人模式"):
112                         txt3 = gr.Markdown(
113                             """
114                             <span style='font-size: 16px;'>【算法模板模式】当
您仅需要算法模板时</span><br><br>
115                             <span style='font-size: 16px;'>【算法交流模式】当
您需要和机器人对某个算法展开深入探讨时</span>
116                             """
117                         )
118             # 启动聊天接口
119             user_info = [
120                 ("admin", "123456"),
121             ]
122             chat.launch(
123                 share = True,
124                 server_port=9090,
125                 debug=False,
126                 auth=user_info,
127                 auth_message='欢迎登录大模型演示平台！'

```



```

126     )
127
128     if __name__ == "__main__":
129         init()
130         launch_gradio()
131

```

3.2 四个核心chain

3.2.1 chain1 : retrieval_chain

用户描述 ——> 向量数据库中最相似算法描述 ——> 算法模糊名称

```

1     from langchain_community.embeddings import HuggingFaceBgeEmbeddings
2     from langchain_openai import ChatOpenAI
3     from langchain_community.vectorstores import FAISS
4     from langchain.text_splitter import CharacterTextSplitter
5     from langchain.document_loaders import TextLoader
6     from langchain.chains.retrieval import create_retrieval_chain
7     from langchain.chains.combine_documents import
create_stuff_documents_chain
8     from langchain_core.prompts import(
9         SystemMessagePromptTemplate,
10        HumanMessagePromptTemplate,
11        ChatPromptTemplate,
12    )
13    from utils import LOG, get_data_file_path
14    from sql.data_to_db import DataTODB
15    import os
16
17    class AlgorithmDescription:
18        def __init__(self, model_name : str, api_key : str, base_url : str):
19            try:
20                algorithm_describe_faiss_path =
get_data_file_path("algorithms_describe")
21                algorithm_describe_text_path =
get_data_file_path("algorithm_describe")
22                embeddings = HuggingFaceBgeEmbeddings(
23                    model_name="BAAI/bge-small-zh-v1.5",
24                    model_kwargs={"device": "cpu"},
25                    encode_kwargs={"normalize_embeddings": True}
26                )
27
28                chatmodel = ChatOpenAI(
29                    model = model_name,
30                    api_key = api_key,
31                    base_url = base_url
32                )
33                self.chatmodel = chatmodel
34
35                if os.path.exists(algorithm_describe_faiss_path):
36                    # 加载FAISS数据库
37                    db = FAISS.load_local(algorithm_describe_faiss_path,
embeddings, allow_dangerous_deserialization=True)
38                else:
39                    datatodb = DataTODB()

```

```

40         db =
datatodb.Describe_LocalToDB(algorithm_describe_text_path)
41         self.db = db
42     except Exception as e:
43         LOG.error(f"Initialization failed: {e}")
44         raise
45
46     def GetAlgorithmChaoName(self, input : str, score_threshold : int):
47         try:
48             retriever = self.db.as_retriever(
49                 search_type = "similarity_score_threshold",
50                 search_kwargs = {"score_threshold" : score_threshold,
"k" : 1},
51
52             )
53
54             system_template = (
55                 """
56                 你是国际信息学奥林匹克竞赛冠军,并且也是国际大学生程序设计大赛ICPC-
WF冠军。
57
58                 以检索到的内容中'[回答]'的下一行中()中的内容作为你的输出!!
59                 使用以下检索到的内容来回答问题。
60                 {context}
61                 """
62             )
63             system_prompt =
SystemMessagePromptTemplate.from_template(system_template)
64
65             human_template = ("{input}")
66             human_prompt1 =
HumanMessagePromptTemplate.from_template(human_template)
67
68             prompt = ChatPromptTemplate.from_messages([system_prompt,
human_prompt1])
69
70             chain = create_stuff_documents_chain(self.chatmodel, prompt)
71
72             retrieval_chain =
create_retrieval_chain(retriever=retriever, combine_docs_chain=chain)
73             result = retrieval_chain.invoke({'input' : input})
74             LOG.debug(f"[context]{result['context']}")
75             return result['answer'], True
76         except Exception as e:
77             LOG.error(f"Retrieval failed: {e}")
78             return "An error occurred during retrieval", False
79
80
81

```

3.2.2 chain2 : LLMchain

算法模糊名称 ——> 算法名称在数据库中对应的id ——> 通过数据库以及id找到代码模板

```

1  from langchain_openai import ChatOpenAI
2  from sql.mysql import SQLite
3  from chatbot.algorithm_describe import AlgorithmDescription

```

```

4  from langchain.chains import LLMChain
5  from langchain_core.prompts import(
6      SystemMessagePromptTemplate,
7      HumanMessagePromptTemplate,
8      ChatPromptTemplate,
9  )
10
11  from utils import LOG
12
13  class AlgorithmCode:
14      def __init__(self, model_name : str, api_key : str, base_url : str):
15          try:
16              chatmodel = ChatOpenAI(
17                  model = model_name,
18                  api_key = api_key,
19                  base_url = base_url
20              )
21              self.chatmodel = chatmodel
22
23              sqlite = SQLite()
24              self.sqlite = sqlite
25
26              algorithm__list = sqlite.Select_All_Data()
27
28              name_list_prompt = """
29  你是国际信息学奥林匹克竞赛冠军,并且也是国际大学生程序设计大赛ICPC-WF冠军。
30  你将获得一段关于某个算法的大致描述。
31  请你在以下关键词中选取最相关的一个对应的序号整数作为输出:
32  """
33              for row in algorithm__list:
34                  name_list_prompt += row[1] + ":" + str(row[0]) + '\n'
35
36              LOG.debug(f'[name_list_prompt]{name_list_prompt}')
37
38              self.name_list_promt = name_list_prompt
39
40              algorithm_description = AlgorithmDescription(model_name,
41                  api_key, base_url)
42              self.algorithm_description = algorithm_description
43          except Exception as e:
44              LOG.error(f"Initialization failed: {e}")
45              raise
46
47          def GetAlgorithmCode(self, input : str, score_threshold : int):
48              try:
49                  algorithm_chao_name, statue =
50                  self.algorithm_description.GetAlgorithmChaoName(input, score_threshold)
51                  LOG.debug(f"[algorithm_chao_name]{algorithm_chao_name}")
52                  if not statue:
53                      return algorithm_chao_name, statue
54
55                  system_template = (self.name_list_promt)
56                  system_prompt =
57                  SystemMessagePromptTemplate.from_template(system_template)
58
59                  human_template1 = ("请您仅输出序号整数!!!\n以下是我的输入:\n{input}")
60                  human_prompt1 =
61                  HumanMessagePromptTemplate.from_template(human_template1)

```

```

58         prompt = ChatPromptTemplate.from_messages([system_prompt,
59             human_prompt1])
60
61         chain = LLMChain(llm = self.chatmodel, prompt = prompt)
62
63         algorithm_id = chain.invoke({'input' : algorithm_chao_name})
64     ['text']
65     LOG.debug(f"[algorithm_id]{algorithm_id}")
66
67     result, flag = self.sqlite.Select_Data_By_Id(algorithm_id)
68
69     if flag == 2:
70         LOG.debug(f"[algorithm_code : Algorithm analysis results
71             already exist]{result}")
72     elif flag == 0:
73         result = "知识库中无该算法模板"
74         LOG.error(f'[algorithm_code :]{result}')
75     else:
76         LOG.debug(f"[algorithm_code : The algorithm analysis result
77             does not exist, and the return code]{result}")
78
79     return result, flag, algorithm_id
80
81 except Exception as e:
82     LOG.error(f"Retrieval failed: {e}")
83     return "An error occurred during retrieval", 0

```

3.2.3 chain3 : LLMchain

代码模板 ——> 带有注释以及分析的代码模板

```

1  def CodeAnalysis(self, input : str, score_threshold : int):
2      try:
3          chain = LLMChain(llm = self.chatmodel, prompt =
4              self.prompt_analysis)
5          code, statue, id = self.algorithmcode.GetAlgorithmCode(input,
6              score_threshold)
7          LOG.debug(f"chatbot:\n[code]{code}\n[statue]{statue}\n[id]{id}")
8          result = code
9          if statue == 1:#获取代码, 通过大模型分析代码
10             result = chain.invoke({'input' : code})['text']
11             LOG.debug(f"[chatbot : Algorithm analysis results already
12                 exist]{result}")
13         elif statue == 2:#数据库中已经存在分析代码, 直接输出
14             LOG.debug(f"[chatbot : The algorithm analysis result does
15                 not exist, analyze through code]{result}")
16
17         return result, statue, id
18
19 except Exception as e:
20     LOG.error(f"chatbot : Retrieval failed: {e}")
21     return "An error occurred during retrieval"

```

3.2.4 chain4: LLMchain

历史对话作为prompt, 与TAQAModel进行算法交流

```
1     def AlgorithmExchange(self, input : str, history : list):
2         try:
3             prompt_list = [self.system_prompt2]
4             LOG.debug(f"[history len]{len(history)}")
5             if len(history) > 0:
6                 LOG.debug(f"[history message]{history[0][0]}")
7                 human_template0 = (self.escape_braces(history[0][0]))
8                 LOG.debug(f"[history message]{ human_template0}")
9                 human_prompt0 =
HumanMessagePromptTemplate.from_template(human_template0)
10                prompt_list.append(human_prompt0)
11                LOG.debug(f"[history_human]{human_prompt0}")
12
13                LOG.debug(f"[history return]{history[0][1]}")
14                ai_template0 = (self.escape_braces(history[0][1]))
15                ai_prompt0 =
AIMessagePromptTemplate.from_template(ai_template0)
16                prompt_list.append(ai_prompt0)
17                LOG.debug(f"[history_ai]{ai_prompt0}")
18
19                prompt_list.append(self.human_prompt1)
20                prompt_exchange = ChatPromptTemplate.from_messages(prompt_list)
21                chain = LLMChain(llm = self.chatmodel, prompt = prompt_exchange)
22                result = chain.invoke({'input' : input})['text']
23                return result
24
25        except Exception as e:
26            LOG.error(f"chatbot : Retrieval failed: {e}")
27            return "An error occurred during retrieval"
```

3.3 数据库操作集成

```
1     import sqlite3
2     import os
3     from utils import LOG, get_data_file_path
4     from typing import Tuple, List
5     from langchain.text_splitter import CharacterTextSplitter
6     from langchain.document_loaders import TextLoader
7
8     class SQLite:
9         def __init__(self):
10             db_path = get_data_file_path("algorithms_code.db")
11             exist_db = True
12             if not os.path.exists(db_path):
13                 exist_db = False
14
15             conn = sqlite3.connect(db_path, check_same_thread=False)
16
17             cursor = conn.cursor()
18
```

```

19         if exist_db == False:
20             cursor.execute("""
21                 CREATE TABLE IF NOT EXISTS algorithms (
22                     id INTEGER PRIMARY KEY AUTOINCREMENT,
23                     name TEXT NOT NULL,
24                     code TEXT NOT NULL,
25                     result TEXT
26                 )
27             """)
28             LOG.debug("建立表成功")
29             conn.commit()
30             self.conn = conn
31             self.cursor = cursor
32
33             loader = TextLoader(get_data_file_path("algorithm.txt"),
encoding="UTF-8")
34             document = loader.load()
35
36             text_splitter = CharacterTextSplitter(
37                 separator = r'\n\n-----\d+-----',
38                 chunk_size=300,
39                 chunk_overlap=0,
40                 length_function = len,
41                 is_separator_regex = True
42             )
43
44             docs = text_splitter.split_documents(document)
45             for doc in docs:
46                 text = doc.page_content
47                 lines = text.strip().split('\n')
48                 current_section = None
49
50                 for line in lines:
51                     line = line.strip()
52                     if line == "[算法名称]":
53                         current_section = "算法名称"
54                         algorithm_name = ""
55                     elif line == "[算法代码]":
56                         current_section = "算法代码"
57                         algorithm_code = ""
58                     elif current_section == "算法名称":
59                         algorithm_name += line + "\n"
60                     elif current_section == "算法代码":
61                         algorithm_code += line + "\n"
62
63                 algorithm_name = algorithm_name.strip()
64                 algorithm_code = algorithm_code.strip()
65                 LOG.debug(f'[algorithm_name]{algorithm_name}')
66                 LOG.debug(f'[algorithm_code]{algorithm_code}')
67                 statue = True
68                 statue |= self.Insert_Data_name_and_id(algorithm_name,
algorithm_code)
69                 self.conn = conn
70                 self.cursor = cursor
71
72
73         def Select_Data_By_Name(self, algorithm_name : str) -> Tuple[str,
bool]:

```

```

74         self.cursor.execute("SELECT * FROM algorithms WHERE name = ?",
(algorithm_name,))
75         existing_rows = self.cursor.fetchall()
76         if existing_rows:
77             return existing_rows[0][2], True
78         else:
79             return None, False
80
81     def Select_Data_By_Id(self, algorithm_id : int) -> Tuple[str, int]:
82         self.cursor.execute("SELECT * FROM algorithms WHERE id = ?",
(algorithm_id,))
83         existing_rows = self.cursor.fetchall()
84         if existing_rows:
85             if len(existing_rows[0]) > 3 and existing_rows[0][3] is not
None:
86                 return existing_rows[0][3], 2
87             elif len(existing_rows[0]) > 2:
88                 return existing_rows[0][2], 1
89             return None, 0
90         else:
91             return None, 0
92
93     def Insert_Data_name_and_id(self, algorithm_name : str, algorithm_code
: str):
94         LOG.debug(f"[Insert_name]{algorithm_name} \n [Insert_code]
{algorithm_code}")
95         code, statue = self.Select_Data_By_Name(algorithm_name)
96         if statue:
97             return False
98         else:
99             sql = "INSERT INTO algorithms (name, code) VALUES (?, ?)"
100             val = (algorithm_name, algorithm_code)
101             self.cursor.execute(sql, val)
102             self.conn.commit()
103             return True
104
105     def Insert_Date_result_By_ID(self, algorithm_id : int, result : str):
106         code, statue = self.Select_Data_By_Id(algorithm_id)
107         if statue == 1:
108             self.cursor.execute("""
109             UPDATE algorithms
110             SET result = ?
111             WHERE id = ?
112             """, (result, algorithm_id))
113
114             self.conn.commit()
115             return True
116         else:
117             return False
118
119     def Select_All_Data(self): # 定义一个方法，用于查询所有算法数据，返回所有记录
120         self.cursor.execute("SELECT * FROM algorithms")
121         all_rows = self.cursor.fetchall()
122         return all_rows
123
124     def close(self):
125         self.cursor.close()
126         self.conn.close()

```

3.4 utils——工具箱

3.4.1 日志管理

```

1  from loguru import logger
2  import os
3  import sys
4
5  class Logger:
6      def __init__(self, log_dir = "logs"):
7          if not os.path.exists(log_dir):
8              os.makedirs(log_dir)
9
10         log_file_path = os.path.join(log_dir, "algorithm_chatbot.log")
11         logger.remove()
12
13         logger.add(sys.stdout, level="DEBUG")
14         logger.add(log_file_path, rotation="02:00", level="DEBUG")
15         self.logger = logger
16
17 LOG = Logger().logger
18
19 if __name__ == "__main__":
20     log = Logger().logger
21
22     log.debug("This is a debug message.")
23     log.info("This is an info message.")
24     log.warning("This is a warning message.")
25     log.error("This is an error message.")
26
27

```

3.4.2 YAML 文件参数配置管理

```

1  import yaml
2  import os
3
4  class ChatBotConfig:
5
6      _instance = None
7
8      def __new__(cls):
9          if cls._instance is None:
10             cls._instance = super(ChatBotConfig, cls).__new__(cls)
11             cls._instance._config = None
12             return cls._instance
13
14     def init(self, dir_path):
15         with open(os.path.join(dir_path, "config.yaml"), "r") as f:
16             config = yaml.safe_load(f)
17
18         self._instance._config = config

```



```

19
20     def __getattr__(self, name):
21         if self._instance._config and name in self._instance._config:
22             return self._instance._config[name]
23         raise AttributeError(f"ChatBotConfig 对象没有属性'{name}'")

```

3.4.3 文本去除噪音

```

1  import re
2  import jieba
3  from utils.data_path import get_data_file_path
4  class NoiseRemoval:
5      def __init__(self):
6          with open(get_data_file_path("stopwords.txt"), 'r', encoding="UTF-8") as f:
7              stop_words = set(word.strip() for word in f if word.strip())
8              self.stop_words = stop_words
9
10     def Do_NoiseRemoval(self, text : str):
11         text = re.sub(r'^\w\s', '', text)
12         text = re.sub(r'\s+', ' ', text).strip()
13         words = jieba.lcut(text)
14         filtered_words = [word for word in words if word not in
15 self.stop_words]
16         return ' '.join(filtered_words)
17

```

4 界面展示

4.1 登录界面



登录

欢迎登录大模型演示平台!

username
admin

password

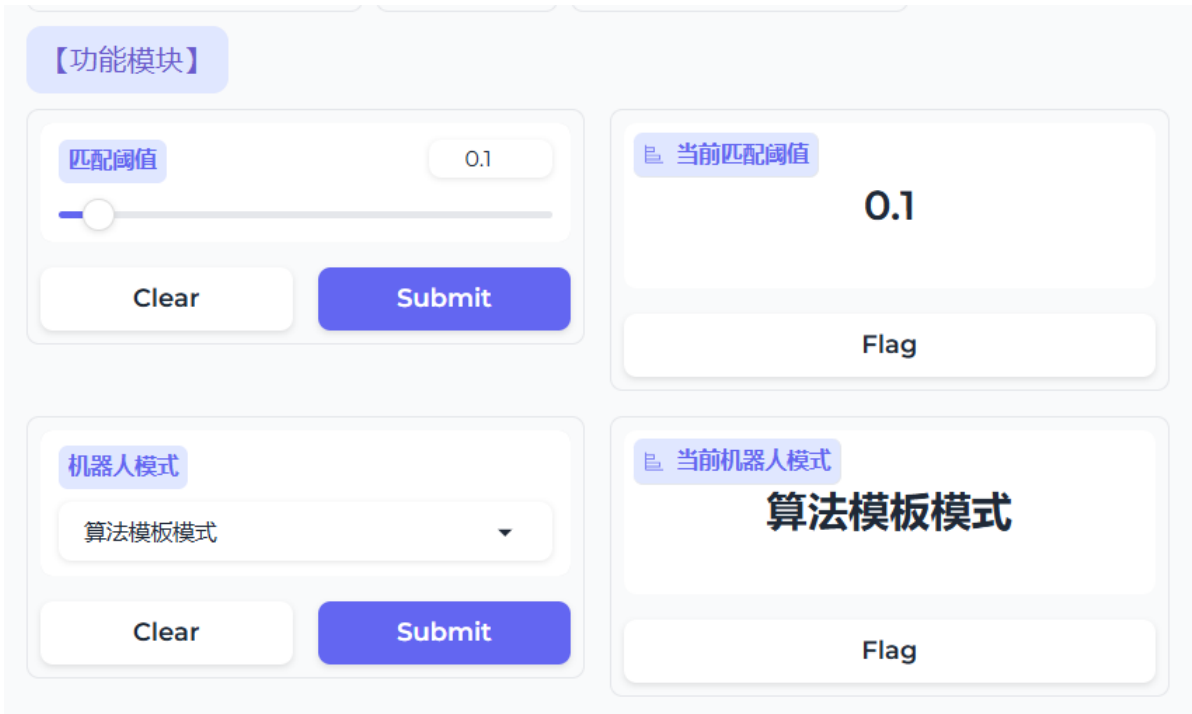
登录

4.2 用户界面

4.2.1 页面



4.2.2 功能模块



【功能模块】

匹配阈值

0.9

ClearSubmit

当前匹配阈值

0.9

Flag

机器人模式

算法交流模式

ClearSubmit

当前机器人模式

算法交流模式

Flag

4.2.3 机器人使用说明

【机器人使用说明】

匹配阈值

机器人模式

【算法模板模式】

当您仅需要算法模板时

【算法交流模式】

当您需要和机器人对某个算法展开深入探讨时

【机器人使用说明】

匹配阈值

机器人模式

【请将阈值调高】

当您需要让机器人专门匹配某个算法模板时/当机器人语无伦次时

【请将阈值调低】

当您不知道用算法的名称叫什么时/不知道用什么算法时/机器人匹配不到你需要的算法时

4.3 算法模式

4.3.1 演示1

算法模板机器人(比赛ak机器人)

Chatbot

二分图的最大匹配问题

以下是重新排版并添加逐行注释的代码。这段代码看起来是一个图的匹配算法，可能是匈牙利算法的一部分，用于寻找最大匹配。

```
// 定义函数find，用于寻找与节点x匹配的节点
bool find(int x) {
    // 从邻接表h[x]开始，遍历所有与节点x相连的边
    for (int i = h[x]; i != -1; i = ne[i]) {
        int j = e[i]; // 获取与边i相连的另一个节点编号
        // 如果节点j没有被访问过
        if (!st[j]) {
            st[j] = true; // 标记节点j为已访问
            // 如果节点j没有被匹配，或者它的匹配节点可以被找到一个新匹配
            if (!match[j] || find(match[j])) {
                match[j] = x; // 将节点j与节点x匹配
                return true; // 返回找到匹配的结果
            }
        }
    }
    // 如果没有找到匹配，返回false
    return false;
}

// 初始化结果计数器
int res = 0;
// 遍历所有节点
for (int i = 1; i <= n; i++) {
    // 每次遍历前重置访问数组
    memset(st, false, sizeof st);
    // 尝试为每个节点找到匹配
    if (find(i)) res++; // 如果找到匹配，结果计数器增加
}
```

以下是每个部分的详细注释：

- o `h[x]`：可能是一个数组，表示节点x在邻接表中的头节点。
- o `i != -1`：使用 -1 作为邻接表中的结束标记。

Type a message...



Submit

Examples

可以区间查询，区间修改的算法 | 线段树区间查询 | 可以快速判断字符串是否为子串

【功能模块】

匹配阈值

0.1



Clear

Submit

当前匹配阈值

0.1

Flag

机器人模式

算法模板模式

Clear

Submit

当前机器人模式

算法模板模式

Flag

【机器人使用说明】

匹配阈值 | 机器人模式

【请将阈值调高】当您需要让机器人专门匹配某个算法模板时/当机器人语无伦次时

【请将阈值调低】当您不知道用算法的名称叫什么时/不知道用什么算法时/机器人匹配不到你需要的算法时

通过 API 使用 使用 Gradio 构建

算法模板机器人(比赛ak机器人)

- Chatbot:** 可能是一个数组，表示与节点 x 相连的边的下一个边。
- $e[i]$: 可能是一个数组，存储边的另一端的节点编号。
 - $st[j]$: 一个布尔数组，用于记录节点是否被访问过。
 - $match[j]$: 一个数组，用于存储每个节点的匹配节点。
 - vis : 可能是一个变量，表示图的一个部分中的节点数量。

这段代码中使用了几个未定义的变量和数组，它们在完整的代码中应该有相应的定义。

详细说明一下match数组的作用

在图论中，尤其是在解决图的最大匹配问题时，`match` 数组扮演着重要的角色。它通常用于存储每个节点当前的匹配状态。具体来说，`match` 数组的作用如下：

- `match[i]`: 这个数组中的每个元素代表节点 i 的匹配状态。
- 如果 `match[i]` 的值为 0 ，表示节点 i 当前没有匹配到任何节点。
- 如果 `match[i]` 的值不为 0 （假设为 j ），表示节点 i 与节点 j 匹配。在这种情况下，通常也有 `match[j] = i`，表示匹配是相互的。

在一个最大匹配算法（如匈牙利算法）中，`match` 数组会随着算法的执行而更新，以找到图中最大数量的节点对，使得每一对节点都是相互匹配的。

在上面的代码示例中，`match` 数组的具体作用如下：

- 初始化:** 在算法开始前，`match` 数组通常初始化为 0 ，表示没有任何节点匹配。
- 查找匹配:** 在 `find` 函数中，`match[j] == 0` 检查节点 j 是否未被匹配。如果未被匹配，或者如果它的匹配节点 `match[j]` 可以通过递归调用 `find` 函数找到一个新匹配（即 `find(match[j])` 返回 `true`），则当前节点 x 可以与节点 j 匹配。
- 更新匹配状态:** 当找到一个匹配时，通过 `match[j] = x` 更新 `match` 数组，表示节点 j 与节点 x 匹配。
- 统计结果:** 通过遍历所有节点并调用 `find` 函数，算法可以统计匹配的节点对数量。每当 `find` 函数为某个节点找到匹配时，`res` 计数器增加。

`match` 数组是跟踪算法过程中图节点匹配状态的关键数据结构，它使得算法能够在图结构中找到最大匹配。

Type a message...



Submit

Examples

可以区间查询，区间修改的算法 线段树区间查询 可以快速判断字符串是否为子串

【功能模块】

匹配阈值 0.1

Clear Submit

当前匹配阈值

0.1

Flag

机器人模式

算法交流模式

Clear Submit

当前机器人模式

算法交流模式

Flag

【机器人使用说明】

匹配阈值 机器人模式

【算法模板模式】当您仅需要算法模板时

【算法交流模式】当您需要和机器人对某个算法展开深入探讨时

通过 API 使用 使用 Gradio 构建

4.3.2 演示2

下面是重新排版并添加了逐行注释的代码。这段代码的目的是生成小于或等于给定数字 n 的所有质数。

```
const int N = 100000; // 假设N是一个足够大的数，以存储所有小于n的质数
int primes[N]; // 存储质数的数组
int cnt = 0; // 质数的计数器

bool st[N]; // 标记数组，st[i]为true表示i不是质数

// 获取小于或等于n的所有质数
void get_primes(int n)
{
    // 从2开始遍历到n，因为1不是质数
    for (int i = 2; i <= n; i++)
    {
        // 如果当前数字i没有被标记为非质数，那么它是一个质数
        if (!st[i]) primes[cnt++] = i;

        // 遍历已有的质数，将它们的倍数标记为非质数
        for (int j = 0; primes[j] <= n / i; j++)
        {
            // 标记当前质数与i的乘积为非质数
            st[primes[j] * i] = true;

            // 如果i能够被当前的质数整除，那么不需要继续循环
            // 因为更大的质数乘以i也会大于n/i
            if (i % primes[j] == 0) break;
        }
    }
}
```

注意：代码中的 N 应该足够大以存储所有小于 n 的质数。这里假设 N 为 100000 ，但是实际使用时应该根据需求来定义这个值。

此外，这段代码并没有初始化标记数组 `st`。在实际使用前，应确保所有的元素都被初始化为 `false`，否则可能导致错误的结果。可以通过在 `get_primes` 函数开始处添加以下代码来实现：

```
memset(st, false, sizeof st);
```

Type a message...



Submit

Examples

可以区间查询，区间修改的算法 | 线段树区间查询 | 可以快速判断字符串是否为子串

【功能模块】

匹配阈值0.1

ClearSubmit

当前匹配阈值0.1

Flag

机器人模式

算法模板模式

ClearSubmit

当前机器人模式

算法模板模式

Flag

【机器人使用说明】

匹配阈值 机器人模式

【请将阈值调高】当您需要让机器人专门匹配某个算法模板时/当机器人语无伦次时

【请将阈值调低】当您不知道用算法的名称叫什么时/不知道用什么算法时/机器人匹配不到你需要的算法时

算法模板机器人(比赛ak机器人)

Chatbot 码中的 N 应该足够大以存储所有小于 n 的质数。这里假设 N 为 100000，但是实际使用时应该根据需求来定义这个值。

此外，这段代码并没有初始化标记数组 `st`。在实际使用前，应确保所有的元素都被初始化为 `false`，否则可能导致错误的结果。可以通过在 `get_primes` 函数开始处添加以下代码来实现：

```
memset(st, false, sizeof st);
```

帮我解释一下为什么这个代码是线性的

这段代码使用了一个被称为埃拉托斯特尼筛法 (Sieve of Eratosthenes) 的优化版本，它之所以被称为线性的，是因为它的运行时间复杂度与 n 成正比，即时间复杂度为 $O(n \log \log n)$ 。这里的“线性”是相对的，因为它比 $O(n^2)$ 或更大的时间复杂度要好得多，但仍然不是严格的线性 (即 $O(n)$)。

以下是为什么这个筛法是线性的几个关键原因：

- 内部循环的优化**：传统的埃拉托斯特尼筛法中，内部循环会检查从2到 \sqrt{n} 的所有质数。在这个优化版本中，内部循环只检查已经找到的质数，并且只检查到 n/i 的质数。这是因为如果 i 是某个质数 p 的倍数，那么 $i \times p$ (其中 $p > i$) 将不会是小于或等于 n 的质数，因为 p 已经在之前被考虑过了。
- 避免重复标记**：每个合数只被它的最小质因数标记一次。例如，如果 i 是质数，那么 $2i, 3i, 4i, \dots$ 都会被标记，但是当 i 是合数时，只会由它的最小质因数来标记，这样可以避免重复工作。
- 提前终止内部循环**：代码中的 `if (i % primes[j] == 0) break;` 这一行是关键。它表示如果当前数 i 能够被当前遍历到的质数 `primes[j]` 整除，那么就没有必要继续遍历更大的质数了，因为 `primes[j]` 已经是最小的质因数，所有更大的质数乘以 i 产生的数都会大于 n/i 。

这些优化确保了每个数最多只被检查一次，而且每个数的最小质因数只被用来标记其倍数一次。因此，尽管代码中有一个嵌套循环，但每个数实际上只参与了一次标记操作，这导致了整体算法的线性行为。

总结来说，这个筛法之所以线性，是因为：

- 每个数只被其最小质因数对应的循环所处理。
- 内部循环的次数大大减少，因为只需要检查到 n/i 。
- 没有重复的标记操作。

这些因素共同作用，使得算法的效率非常高，特别是对于处理大量数据的质数筛选。

Type a message...



Submit

Examples

可以区间查询，区间修改的算法 | 线段树区间查询 | 可以快速判断字符串是否为子串

【功能模块】

匹配阈值

0.1



Clear

Submit

当前匹配阈值

0.1

Flag

机器人模式

算法交流模式

Clear

Submit

当前机器人模式

算法交流模式

Flag

【机器人使用说明】

匹配阈值

机器人模式

【算法模板模式】当您仅需要算法模板时

【算法交流模式】当您需要和机器人对某个算法展开深入探讨时

通过 API 使用 | 使用 Gradio 构建