lab 5

杨乙　21307130076　信息安全

完成想法与实验步骤

任务 1：识别控制流图基本块

基本块的起始地址 `block_addr` 已经给出。关键在于确定基本块的结束地址。结束位置有两种可能：

- `branch` 语句处（包括 `ret` 语句）
- `label` 语句之前

因为 `label` 语句的位置可能在处理基本块之后才能被确定（例如较大的地址跳转到较小的地址，且较大地址在较小地址后续基本块的情况），所以先定义全局变量 `labels` 记录所有 `label` 语句地址，再定义 `label_ins` 函数获取所有 `label` 语句的地址。`label_ins` 函数采用深度优先搜索的方法，利用 `disassemble_block` 函数获取后续的基本块位置。注意不要重复访问一个基本块，以免 DFS 成环：

```
labels = []                               # 记录所有 label 语句地址

def label_ins(self, block_addr: int) -> list[int]:
    # added by myself
    stack = [block_addr]                  # DFS 栈
    visited = set()                       # 避免重复访问形成环
    labels = []

    while stack:
        current_addr = stack.pop()
        if current_addr in visited:
            continue
        visited.add(current_addr)

        insns = {}
        labels = labels + self.disassemble_block(current_addr, insns)   # 后继地址加入 labels
        stack = stack + self.disassemble_block(current_addr, insns)     # 后继地址加入栈

    return labels
```

接下来实现 `disassemble_block` 函数，识别基本块包含哪些汇编指令，并修改作为参数的字典（对字典的修改会保留），最后返回后续的基本块起始地址：

```
def disassemble_block(
    self, block_addr: int, insns: dict[int, capstone.CsInsn]
) -> list[int]:
    # IMPELMENT ME
    while True:
```

```
            insn = self.disasemble_at(block_addr)
            insns[block_addr] = insn

            if self.is_ret(insn):                # 是 ret 语句
                break
            if self.is_jump(insn):               # 是跳转语句
                return self.get_jump_succs(insn)    # 获取后续基本快起始位置
            block_addr = block_addr + 4
            if block_addr in self.labels:        # 下一个语句是标签语句
                return [block_addr]              # 下一个基本块以这条语句起始
        return []
```

任务 2：构建控制流图

从首地址为 `entry_addr` 的基本块出发，对图进行深度优先搜索，同时维护 `next_addrs` 字典，记录某一节点的所有后续基本块地址。在深度优先搜索的过程中，调用 `disassemble_block` 函数获取当前基本块指令，形成节点并添加到图中。最后根据 `next_addrs` 向图中添加所有边：

```
    def build_cfg(self, entry_addr: int) -> nx.DiGraph:
        # IMPELMENT ME
        cfg = nx.DiGraph()  # the node of the graph must be CFGNode

        stack = [entry_addr]                         # DFS 栈
        next_addrs = {entry_addr:[]}                 # 字典，{当前节点:后续节点列
表}
        addr2node = {}                               # 字典，{起始地址:节点}
        insns = {}                                   # 字典，做参数
        visited = set()                              # 避免重复访问形成环

        self.labels = self.label_ins(entry_addr)     # 调用 label_ins 获取所有
label 语句地址

        while stack:
            current_addr = stack.pop()               # 获取当前要处理的基本块首地
址
            if current_addr in visited:
                continue
            visited.add(current_addr)

            insns = {}                               # 获取后续基本块地址，加入
next_addrs 字典
            next_addrs[current_addr] = self.disassemble_block(current_addr,
insns)
            for next_addr in next_addrs[current_addr]:
                stack.append(next_addr)              # 后续基本块地址加入栈

            cfg_node = CFGNode(current_addr, list(insns.values()))     # 生成节点
            cfg.add_node(cfg_node)                                     # 添加到图
            addr2node[current_addr] = cfg_node                         # 添加到
addr2node 字典

        for current_addr in next_addrs:
            for next_addr in next_addrs[current_addr]:
```

```
                cfg.add_edge(addr2node[current_addr], addr2node[next_addr])
# 添加边

        return cfg
```

任务3：分析控制流（自行编写算法）

首先分析可达性。对于两条指令在同一个基本块内部的情况，若起始指令在结束指令之前则可达，反之不可达。若两条指令不在同一个基本块内部，可以通过深度优先搜索来判断是否可达：

```
    def can_reach(self, cfg: nx.DiGraph, src_addr: int, dst_addr: int) -> bool:
        # IMPELMENT ME
        src_node = self.find_node_by_addr(cfg, src_addr)        # 起始节点
        dst_node = self.find_node_by_addr(cfg, dst_addr)        # 终止节点

        if src_node == dst_node and src_addr <= dst_addr:       # 两条指令在同一个基
本块内部
            return True                                         # 起始指令在结束指令
之前

        stack = [src_node]
        visited = set()
        depth = 0                                               # 循环深度
        while stack:
            current_node = stack.pop()
            if current_node in visited:
                continue
            visited.add(current_node)

            if current_node == dst_node and depth != 0:         # 保证不在同一基本块
                return True                                     # 可达

            stack = stack + list(cfg.adj[current_node])         # 加入相邻节点
            depth = depth + 1                                   # 深度增加

        return False
```

接下来分析所有可能的路径。若两条指令在同一个基本块内部且起始指令在结束指令之前，路径只有一个节点，就是起始节点，因为路径节点不重复；对于其余情况，可以用深度优先搜索来实现。通过循环更新当前路径，若当前节点地址等于目的节点地址，则说明找到一条路径，加入路径列表；反之说明又找到一个路径上的节点，更新栈：

```
    def find_paths(
        self, cfg: nx.DiGraph, src_addr: int, dst_addr: int
    ) -> list[list[int]]:
        # IMPELMENT ME
        src_node = self.find_node_by_addr(cfg, src_addr)            # 起始节点
        dst_node = self.find_node_by_addr(cfg, dst_addr)            # 终止节点

        if self.can_reach(cfg, src_addr, dst_addr):                 # 可达
            if src_node == dst_node and src_addr <= dst_addr:       # 两条指令在同
一个基本块内部
```

```python
        return [[src_node.addr]]                        # 起始指令在结束指令之前
                                                        # 只有一条路径，就是起始节点

    stack = [(src_node, [src_node.addr])]               # DFS 栈
    paths = []                                          # 所有可能的路径

    while stack:
        (current_node, path) = stack.pop()              # 当前节点与已经记录的路径

        lis = [l.addr for l in list(cfg.adj[current_node])] # 所有邻接节点地址

        for next_addr in set(lis) - set(path):          # 避免路径上节点重复
            if next_addr == dst_node.addr:              # 找到一条路径
                paths.append(path + [next_addr])
            else:                                       # 更新 DFS 栈
                stack.append((self.find_node_by_addr(cfg, next_addr), path + [next_addr]))
    return paths

return []
```

运行示例

```
[.] Checking Results...
[.] Checking CFG building results
  [+] datapipe-armel-static_0x040ff8: Correct
  [+] gdbserver-armel-static-8.0.1_0x019c48: Correct
  [+] netstat-armel-static_0x03da84: Correct
  [+] id-armel-static_0x076360: Correct
  [+] gdbserver-armel-static-8.0.1_0x055b90: Correct
  [+] xxd-armel-static_0x057714: Correct
  [+] gdbserver-armel-static-8.0.1_0x06cd5c: Correct
  [+] gdbserver-armel-static-8.0.1_0x086df8: Correct
  [+] id-armel-static_0x01cedc: Correct
  [+] ifconfig-armel-static_0x01c328: Correct
Checking CFG analysis results
  [+] id-armel-static_0x076360_0x0763c8_0x076414: Correct
  [+] gdbserver-armel-static-8.0.1_0x055b90_0x055b9c_0x055bc0: Correct
  [+] gdbserver-armel-static-8.0.1_0x06cd5c_0x06cd68_0x06cd88: Correct
  [+] ifconfig-armel-static_0x01c328_0x01c35c_0x01c3fc: Correct
  [+] xxd-armel-static_0x057714_0x057728_0x0577b8: Correct
  [+] id-armel-static_0x01cedc_0x01cf4c_0x01cf58: Correct
  [+] netstat-armel-static_0x03da84_0x03daf0_0x03dac4: Correct
  [+] gdbserver-armel-static-8.0.1_0x086df8_0x086e20_0x086e3c: Correct
  [+] netstat-armel-static_0x03da84_0x03dacc_0x03db28: Correct
  [+] ifconfig-armel-static_0x01c328_0x01c4e0_0x01c3b0: Correct
  [+] xxd-armel-static_0x057714_0x0578e4_0x0577f8: Correct
  [+] id-armel-static_0x076360_0x076374_0x0763e4: Correct
  [+] gdbserver-armel-static-8.0.1_0x086df8_0x086e0c_0x086e38: Correct
  [+] gdbserver-armel-static-8.0.1_0x019c48_0x019c64_0x019c88: Correct
  [+] gdbserver-armel-static-8.0.1_0x055b90_0x055bd0_0x055c60: Correct
  [-] gdbserver-armel-static-8.0.1_0x06cd5c_0x06cd9c_0x06cd68: Unmatched
  [+] gdbserver-armel-static-8.0.1_0x019c48_0x019c74_0x019c9c: Correct
  [+] datapipe-armel-static_0x040ff8_0x04106c_0x041074: Correct
  [+] datapipe-armel-static_0x040ff8_0x0410b4_0x041070: Correct
  [+] id-armel-static_0x01cedc_0x01cef0_0x01cffc: Correct
[+] 10/10 CFG building tasks are correct.
        The remaining could be wrong, or is better than the groundtruth.
[+] 19/20 CFG analyzing tasks are correct.
        The remaining could be wrong, or is better than the groundtruth.
```

遇到的问题与解决方案

对于任务三中情况 `gdbserver-armel-static-8.0.1_0x06cd5c_0x06cd9c_0x06cd68`，我的答案与参考答案不一致（参考答案认为这种情况可达）。在分析了对应的流程图后，我发现这种情况实际上**两条指令在同一个基本块内部且起始指令在结束指令之后。**对于这种情况应该是不可达的