

Task1：



不断进行循环判断指令是不是跳转/返回指令，如果是返回指令那么直接跳出来即可；如果是跳转指令，那么通过已经构造好的get\_jump\_succ函数来获取后续的跳跃地址然后返回。如果都不是那么继续找下一个指令

Task2：

核心思路：将现有的基本块和该基本块的下一个块的节点连线

下一个块包括：跳跃指令指向的下一个基本块 以及 **为了满足“同一条指令地址不重复出现在不同的基本块中”这一要求而分割开的块（难点）**

整体通过以下形式进行实现，用queue来维护，把当前基本块的后续基本块都加入到queue中，并且用visited来标记已经建立Node的基本块，避免重复访问

While queue:

#获得该基本块的下一个地址的列表successor

#处理当前基本块,后面详细给出，要考虑跳跃和分割两种情况，也会影响上面的successor

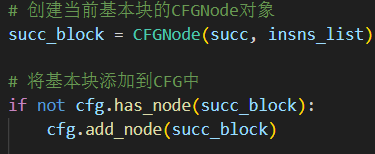
#生成对应的节点



for succ in successors:

#处理下一个块，处理方法一样，也会在后面给出

#生成对应节点



#添加从基本块到下一个块的边



1. 考虑跳跃指令的下一个基本块

会有一个或者两个跳转的地址，这里直接通过

successors = self.disassemble\_block(block\_addr,insns)来获取就行。

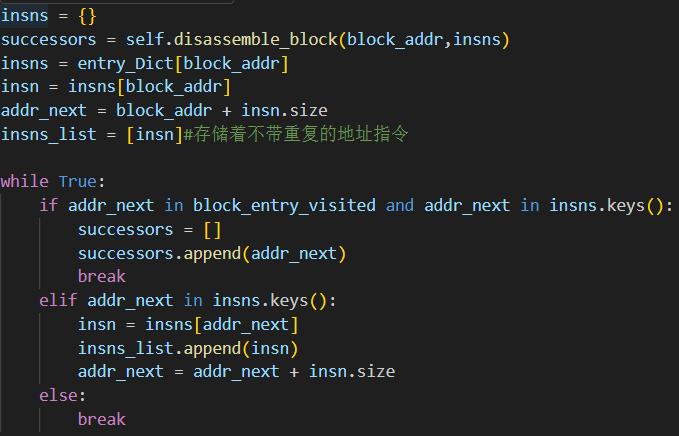
1. 考虑分割开的块

这里实际上就是影响了构建CFGNode对象的时候传入的指令地址。

为了实现这种情况，首先对整体进行遍历找出了每一个基本块的起始地址



之后在创建CFG节点之前写出判断条件：

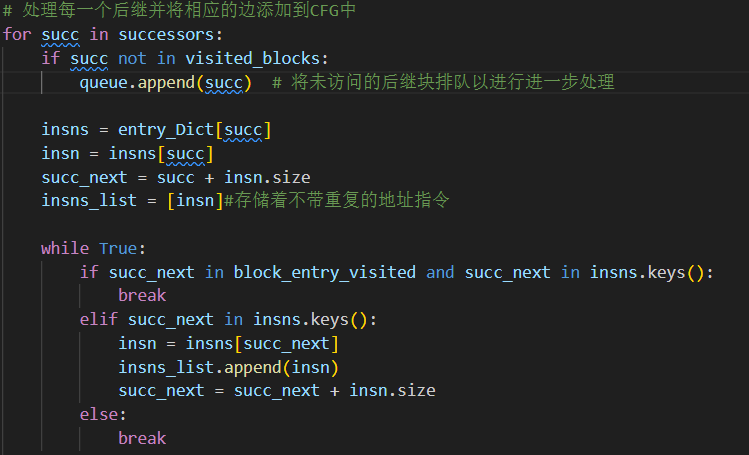


找到分割块的思路：如果一个基本块**非首地址**处有和其他基本块首地址重合的地方，那么就要分割开

在这里需要注意的是，如果是分割的块那么它**要跳转的地方只有一个**，所以必须对successor进行一次初始化之后再进行添加下一条地址（如第三行）。

并且还需要满足条件：保证分割开的地址是在分割前基本块里面的

同样地，在处理下一个基本块也用类似地方法判断：



这里不用考虑successor了，因为是当前基本块到succ\_block的单向连线

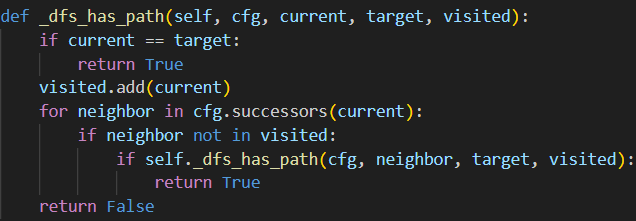
Task3：

不使用networkx来实现。

首先使用find\_node\_by\_addr来将地址转换成基本块



1. 实现can\_reach函数

用dfs来实现，设计一个集合visited = set()来标记，避免重复。

设计跳出条件：①if current == target return True

1. 函数的最后return False

Dfs要做到的事情：找到是否有路

1. 实现find\_paths函数

同样使用dfs来实现。

Dfs要做到的事情：找到两个块（src\_block, dst\_block）之间的所有路

整体思路和上面一致，需要额外处理的就是，dfs里面寻找的路径是node的形式，然后需要把它转换成节点地址

