Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Science and Engineering
Session: 2021-2022 [EVEN SEM]

Compiler Design Lab

Name : Kush Munot

Sec : A

Roll no. : 47

Batch : A3

Subject : Compiler Design

PRACTICAL No. 6

Aim: Write a program to detect loop in the Three Address code given

Code :

```
def find_leaders(statements):
    leaders = set()
    leaders.add(1)

    for i, statement in enumerate(statements):
        if "GOTO" in statement:
            target = int(statement.split()[-1])
            leaders.add(target)
            if i + 2 <= len(statements):
                leaders.add(i + 2)

    return leaders


def create_basic_blocks(statements, leaders):
    basic_blocks = {}
    current_block = None

    for i, statement in enumerate(statements, start=1):
        if i in leaders:
            current_block = i
            basic_blocks[current_block] = []

        basic_blocks[current_block].append(statement)

    return basic_blocks


def program_flow_graph(statements, basic_blocks):
    edges = set()
```

```
    for i, statement in enumerate(statements):
        if "GOTO" in statement:
            source = [k for k, v in basic_blocks.items() if statement in v][0]
            target = int(statement.split()[-1])
            edges.add((source, target))

            if i + 2 <= len(statements):
                edges.add((source, i + 2))

    return edges


def dominators(basic_blocks, pfg):
    dominators = {}

    for block in basic_blocks:
        if block == 1:
            dominators[block] = set()
        else:
            dominators[block] = set(basic_blocks.keys())

    while True:
        updated_dominators = dominators.copy()
        for block in basic_blocks:
            if block != 1:
                preds = {pred for pred, succ in pfg if succ == block}
                if preds:
                    updated_dominators[block] = {block} |
set.intersection(*[dominators[pred] for pred in preds])

        if dominators == updated_dominators:
            break
        else:
            dominators = updated_dominators

    return dominators


def natural_loop(pfg):
    loops = set()

    for source, target in pfg:
        if target < source:
            loops.add((target, source))

    return loops


statements = [
```

```
    "sum = 0",
    "i = 0",
    "If i > n GOTO 12",
    "t1 = addr(a)",
    "t2 = i * 4",
    "t3 = t1[t2]",
    "t4 = sum + t3",
    "sum = t4",
    "t5 = i + 1",
    "i = t5",
    "GOTO 3",
    "end",
]

leaders = find_leaders(statements)
basic_blocks = create_basic_blocks(statements, leaders)
pfg = program_flow_graph(statements, basic_blocks)
dominators_data = dominators(basic_blocks, pfg)
loops = natural_loop(pfg)

print("Leader statements:", leaders)
print("Basic blocks:", basic_blocks)
print("Program Flow Graph:", pfg)
print("Dominators of all basic blocks:", dominators_data)
print("Natural loop:", loops)
```

Input:

```
statements = [
    "sum = 0",
    "i = 0",
    "If i > n GOTO 12",
    "t1 = addr(a)",
    "t2 = i * 4",
    "t3 = t1[t2]",
    "t4 = sum + t3",
    "sum = t4",
    "t5 = i + 1",
    "i = t5",
    "GOTO 3",
    "end",
]
```

Output:

```
Leader statements: {1, 3, 12, 4}
```

```
Basic blocks: {1: ['sum = 0', 'i = 0'], 3: ['If i > n GOTO 12'], 4:
['t1 = addr(a)', 't2 = i * 4', 't3 = t1[t2]', 't4 = sum + t3', 'sum =
t4', 't5 = i + 1', 'i = t5', 'GOTO 3'], 12: ['end']}
Program Flow Graph: {(3, 12), (4, 12), (3, 4), (4, 3)}
Dominators of all basic blocks: {1: set(), 3: {1, 3, 4, 12}, 4: {1,
3, 4, 12}, 12: {1, 3, 4, 12}}
Natural loop: {(3, 4)}
```