Lab - 06

ACT : Advanced Compiler Techniques

AIM : Find dominance frontier and Post-dominator tree

Name : Ambalia Harshit
Roll no : MT001
Date : 06 Oct 2023

Question 01 : Given CFG find dominance frontier

Input : Simple 3 address code

Output : Dominance frontier

Code :

```
def successors(arr, size):
    successor_dict = {}
    for i in range(size):
        for j in range(size):
            if arr[i][j] == 1:
                if i not in successor_dict:
                    successor_dict[i] = [j]
                else:
                    successor_dict[i].append(j)
            else:
                if i not in successor_dict:
                    successor_dict[i] = []
    return successor_dict

def find_paths(graph, start_node, target_node):
    stack = [(start_node, [start_node])]
    paths = []
    while stack:
        current, path = stack.pop()
```

```
        if current == target_node:
            paths.append(path)
        else:
            for succ in graph.get(current, []):
                if succ not in path:
                    stack.append((succ, path + [succ]))
    return paths

def find_dominators(paths):
    common = {}
    for key, lst in paths.items():
        if not lst:
            common[key] = []
        else:
            common[key] = lst[0]
            for tmp in lst[1:]:
                common[key] = list(set(common[key]) & set(tmp))
    return common

def find_join_nodes(arr, length):
    transpose_matrix = [[arr[j][i] for j in range(len(arr))] for i in
range(len(arr[0]))]
    join_nodes = []
    for i in range(length):
        if(transpose_matrix[i].count(1)>1):
            join_nodes.append(i)
    return join_nodes

def dominator_frontier(dominators, join_nodes):
    dominator_frontiers = {node: set() for node in range(11)}
    for node in range(11):
        children = set()
        for child in range(node + 1, 11):
            if child in dominators[node]:
                children.add(child)

        for child in children:
            dom_child = set(dominators[child])
            dom_child.difference_update(dominators[node])
            for ancestor in dom_child:
```

```
                if ancestor not in join_nodes:
                    dominator_frontiers[child].add(ancestor)
    # Adding join nodes to dominator frontiers
    for join_node in join_nodes:
        for node in dominator_frontiers:
            if join_node in dominators[node]:
                dominator_frontiers[node].add(join_node)
    return dominator_frontiers

# def find_immediate_dominator(dominators, x):
#     for _, value in dominators.items():
#         if (x in value):
#             index = value.index(x)
#             return value[index-1]

# def traverse(dominators, immdiate_dominator, pred):
#     for _, value in dominators.items():
#         if pred in value:
#             start_index = value.index(pred) if pred in value else None
#             end_index = value.index(immdiate_dominator) if
immdiate_dominator in value else None
#             sublist = value[start_index:end_index + 1] if start_index is
not None and end_index is not None else []
#             print(sublist)

# for join_node in join_nodes:
#     immdiate_dominator = find_immediate_dominator(dominators, join_node)
#     for pred in predecessor.get(join_node, []):
#         # print(pred, dominators, immdiate_dominator, pred)
#         traverse(dominators, immdiate_dominator, pred)

def main():
    # length = int(input("Enter size : "))
    length = 11
    # arr = []
    # for i in range(size):
    #     a = []
    #     for j in range(size):
    #         a.append(int(input()))
    #     arr.append(a)
```

```
    arr = [
        [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    ]

    successor = successors(arr, length)
    print(f'Successor : {successor}')
    paths = {}
    for i in range(length):
        paths[i] = find_paths(successor, 0, i)
    dominators = find_dominators(paths)
    print(f'Dominators : {dominators}')
    join_nodes = find_join_nodes(arr, length)
    print(f'Join nodes : {join_nodes}')
    df = dominator_frontier(dominators, join_nodes)
    print(f'Dominator Frontier : {df}')

if __name__=="__main__":
    main()
```

Output :

```
hr@Edith:~/Documents/Semester_9/Lab_ACT$ python3 -u "/home/hr/Documents/Semester_9/Lab_ACT/Lab_06/dominance_frontier_m2.py"
Successor : {0: [1, 2], 1: [2], 2: [3], 3: [2, 4, 5], 4: [6], 5: [6], 6: [2, 7], 7: [8, 9], 8: [10], 9: [2, 6, 10], 10: [0]}
Dominators : {0: [0], 1: [0, 1], 2: [0, 2], 3: [0, 2, 3], 4: [0, 2, 3, 4], 5: [0, 2, 3, 5], 6: [0, 2, 3, 6], 7: [0, 2, 3, 6, 7], 8: [0, 2, 3, 6, 7, 8], 9: [0, 2, 3, 6, 7, 9], 10: [0, 2, 3, 6, 7, 10]}
Join nodes : [2, 6, 10]
Dominator Frontier : {0: set(), 1: set(), 2: {2}, 3: {2}, 4: {2}, 5: {2}, 6: {2, 6}, 7: {2, 6}, 8: {2, 6}, 9: {2, 6}, 10: {2, 10, 6}}
hr@Edith:~/Documents/Semester_9/Lab_ACT$
```

Question 02 : Given CFG create post dominator tree

Input : Simple 3 address code

Output : Post Dominance frontier

Code :

```
def successors(arr, size):
    successor_dict = {}
    for i in range(size):
        for j in range(size):
            if arr[i][j] == 1:
                if i not in successor_dict:
                    successor_dict[i] = [j]
                else:
                    successor_dict[i].append(j)
            else:
                if i not in successor_dict:
                    successor_dict[i] = []
    return successor_dict

def find_paths(graph, start_node, target_node):
    stack = [(start_node, [start_node])]
    paths = []
    while stack:
        current, path = stack.pop()
        if current == target_node:
            paths.append(path)
        else:
            for succ in graph.get(current, []):
                if succ not in path:
                    stack.append((succ, path + [succ]))
    return paths

def find_common(paths):
    common = {}
    for key, lst in paths.items():
        if not lst:
            common[key] = []
        else:
            common[key] = lst[0]
```

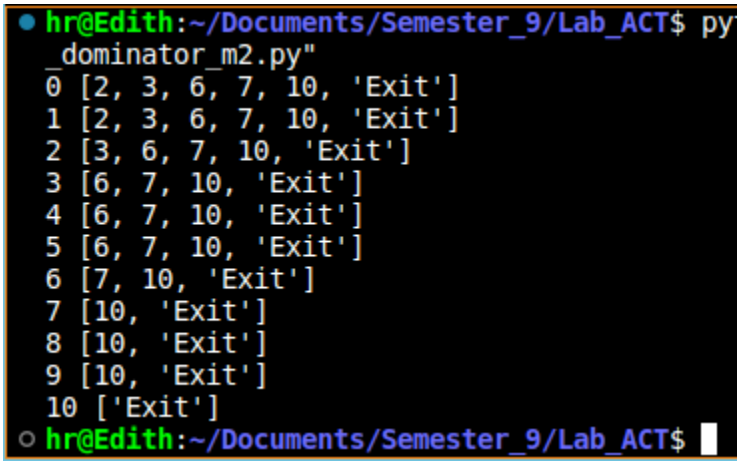
```
            for tmp in lst[1:]:
                common[key] = list(set(common[key]) & set(tmp))
    return common

def main():
    # length = int(input("Enter size : "))
    length = 11
    # arr = []
    # for i in range(size):
    #     a = []
    #     for j in range(size):
    #         a.append(int(input()))
    #     arr.append(a)
    arr = [
        [0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    ]

    successor = successors(arr, length)
    paths = {}
    for i in range(length):
        paths[i] = find_paths(successor, i, length-1)
    post_dominators = find_common(paths)
    for i, j in post_dominators.items():
        j.remove(i)
        j.sort(reverse = False)
        j.append('Exit')
        print(i, j)

if __name__=="__main__":
    main()
```

Output :


hr@Edith:~/Documents/Semester_9/Lab_ACT$ py
_dominator_m2.py"
0 [2, 3, 6, 7, 10, 'Exit']
1 [2, 3, 6, 7, 10, 'Exit']
2 [3, 6, 7, 10, 'Exit']
3 [6, 7, 10, 'Exit']
4 [6, 7, 10, 'Exit']
5 [6, 7, 10, 'Exit']
6 [7, 10, 'Exit']
7 [10, 'Exit']
8 [10, 'Exit']
9 [10, 'Exit']
10 ['Exit']
hr@Edith:~/Documents/Semester_9/Lab_ACT$