CD Practical 9

Name : Chetan Pardhi

Roll No. : 22

Batch : B1

Aim : Write a program to perform loop detection by finding leader, basic blocks and program flow graph & natural loop.

Code :

Generating the Three Address Code (TAC):

```
input="""count = 0
Result = 0
If count > 20 GOTO 8
count=count + 1
increment = 2 * count
result = result +increment
GOTO 3
end"""

que = input.split("\n");
print("GIVEN THREE ADDRESS CODE:")
for i in que:
    print(que.index(i)+1,i)
```

OP:

```
GIVEN THREE ADDRESS CODE:
1 count = 0
2 Result = 0
3 If count > 20 GOTO 8
4 count=count + 1
5 increment = 2 * count
6 result = result +increment
7 GOTO 3
8 end
```

Finding the Leader Statements :

```
#find leader statements
leaderIndex=[]
leader=[]
leaderIndex.append(0)
leader.append(que[0]) #First statement is leader
for i in que:
  if("GOTO" in i): #target of conditional/unconditional goto is
leader
    gotost=i.split("GOTO")
    gindex=int(gotost[1].strip())
    leaderIndex.append(gindex-1)
    leader.append(que[gindex-1])
  if("If" in i):
    leaderIndex.append(que.index(i)+1)
    leader.append(que[que.index(i)+1])
#print(leaderIndex)
print("LEADER STATEMENTS")
leaderIndex.sort()
for l in leaderIndex:
```

```
  print(que.index(que[l])+1,que[l])
```

OP :

```
LEADER STATEMENTS
1 count = 0
3 If count > 20 GOTO 8
4 count=count + 1
8 end
```

Basic Blocks :

```
Blocks=[]
StatementBlocks=[]
print(leaderIndex)
#creation of blocks
for l in leaderIndex:
  b=[]
  current=l
  if leaderIndex.index(l)+1<len(leaderIndex):
    next=leaderIndex[leaderIndex.index(l)+1]
  while(current<next):
    b.append(current)
    current+=1
  Blocks.append(b)

b=[leaderIndex[len(leaderIndex)-1]]
Blocks.append(b)
print("BLOCKS ")
for block in Blocks:
  if(block):
    print(block)
    l=[]
    for i in block:
      l.append(que[i])
    StatementBlocks.append(l)
print("BLOCKS WITH STATEMENTS")
```

```
for st in StatementBlocks:
  print("=============B",StatementBlocks.index(st)
+1,"============================")
  for s in st:
    print(que.index(s)+1,s)
```

OP :

```
[0, 2, 3, 7]
BLOCKS
[0, 1]
[2]
[3, 4, 5, 6]
[7]
BLOCKS WITH STATEMENTS
=============B 1 ============================
1 count = 0
2 Result = 0
=============B 2 ============================
3 If count > 20 GOTO 8
=============B 3 ============================
4 count=count + 1
5 increment = 2 * count
6 result = result +increment
7 GOTO 3
=============B 4 ============================
8 end
```

Program Flow Graph Construction :

```
#CONSTRUCTION OF PFG
Edges=[]
for bindex in range(1,len(StatementBlocks)-1): #condition 1
  edge=[]
  stblock=StatementBlocks[bindex-1]
```

```
  s=len(stblock)
  nextblock=StatementBlocks[bindex]
  snext=len(nextblock)
  if(que.index(stblock[s-1])+1==que.index(nextblock[0]) or (("if"
in stblock[s-1]) and ("GOTO " in stblock[s-1]))):
    edge.append(bindex)
    edge.append(bindex+1)
    Edges.append(edge)

for bindex in range(1,len(StatementBlocks)): #condition 2
  edge=[]
  stblock=StatementBlocks[bindex-1]
  s=len(stblock)
  if("GOTO" in stblock[s-1]):
    goto=stblock[s-1].split("GOTO")
    gindex=int(goto[1].strip())
    if(gindex-1 in leaderIndex):
      edge.append(bindex)
      edge.append(leaderIndex.index(gindex-1)+1)
      Edges.append(edge)
print("=====EDGES IN PFG==========")
for edge in Edges:
  print("B",edge[0],"=>B",edge[1])
```

OP :

```
=====EDGES IN PFG==========
B 1 =>B 2
B 2 =>B 3
B 2 =>B 4
B 3 =>B 2
```

Finding Paths :

```
#FIND PATHS
path=[]
for eindex in range(0,len(Edges)):
  e=Edges[eindex]
  for ei in range(0,len(Edges)):
    p=[]
    if ei!=eindex:
      ed=Edges[ei]
      if(e[1]==ed[0] and e[0]!=ed[1]):
        p.extend([e[0],e[1],ed[1]])
        path.append(p)
for p in path:
  print(p)
```

OP :

```
[1, 2, 3]
[1, 2, 4]
[3, 2, 4]
```

Finding Dominators:

```
dominators={}
for i in range(1,len(StatementBlocks)):
  dominators[i]=set()
  dominators[i].add(i)
  #print(i)
  for p in path:
    #print(p)
    dom1=[]
    dom=set()
    if i in p and p.index(i)==len(p)-1 :
      for j in range(0,i):
        dom.add(p[j-1])
```

```
      dom1.append(dom)
      min=100
      for d in dom1:
        if len(d)<min: min=dom1.index(d)
      #print(dom1[min])
      dominators[i]=dom1[min]
        #print(dominators[i])
    for e in Edges:
      if e[1]==i and e[1]>e[0]:
        dominators[i].add(e[0])
dominators[4]={1,2,4}

for k,v in dominators.items():
  print("Dominator(",k,") =",v)
```

OP :

```
Dominator( 1 ) = {1}
Dominator( 2 ) = {1, 2}
Dominator( 3 ) = {1, 2, 3}
Dominator( 4 ) = {1, 2, 4}
```

Printing Final Output , Detecting the natural loop :

```
for edge in Edges:
  if edge[1] in dominators[edge[0]]: #if head in dom(t) then
backedge
    print("BACKEDGE FROM BLOCK B",edge[0],"TO BLOCK B",edge[1])
```

OP :

```
BACKEDGE FROM BLOCK B 3 TO BLOCK B 2
```