In [3]:
```
# Name: Jiajin Liang
# Email: jil904@ucsd.edu

import numpy as np
class Node:
    def __init__(self,f,input_list):
        self.value = f(input_list)
        self.df_values = []
        self.edge_dfvalues = []
    def backprop(self,df_list,backp_input_list,prev_dfvalue_list):
        for df in df_list:
            self.df_values.append(df(backp_input_list))
        #print(self.df_values,df_list)
        self.dfvalue = np.array(self.df_values).dot(np.array(prev_dfvalue_list
))
        for i in range(len(self.df_values)):
            self.edge_dfvalues.append(self.df_values[i] * prev_dfvalue_list[i
])
    def printnode(self):
        print("forward value: ", self.value, "\ndfvalue (the sum of derivative
s of all outgoing edges/the partial derivative of this node with respect to
 f): ", self.dfvalue)
        for i in range(len(self.edge_dfvalues)):
            print("Derivative of outgoing Edge #", i, " of this node: ", self.
edge_dfvalues[i])
class inputNode(Node):
    def __init__(self,inputv):
        self.value = inputv
        self.df_values = []
        self.edge_dfvalues = []
```

In [5]:
```
# Graph # 1

def v1f(input_list):
    return input_list[0] * input_list[1] + input_list[2]
def v3f(input_list):
    return input_list[0]**2
def v2f(input_list):
    return input_list[0] - input_list[1] * input_list[2]
def v4f(input_list):
    return np.exp(input_list[0])
def v5f(input_list):
    return input_list[0] * input_list[1]
def dv3dv1(backp_input_list):
    return 2*backp_input_list[0]
def dfdv5(backp_input_list):
    return backp_input_list[0]
def dv5dv4(backp_input_list):
    return backp_input_list[0]
def dv5dv3(backp_input_list):
    return backp_input_list[1]
def dv4dv2(backp_input_list):
    return np.exp(backp_input_list[0])
def dv3dv1(backp_input_list):
    return 2*backp_input_list[0]
def dv1dx(backp_input_list):
    return backp_input_list[1]
def dv2dx(backp_input_list):
    return 1
def dv1dy(backp_input_list):
    return backp_input_list[0]
def dv2dy(backp_input_list):
    return -backp_input_list[2]
def dv1dz(backp_input_list):
    return 1
def dv2dz(backp_input_list):
    return -backp_input_list[1]
x = inputNode(1)
y = inputNode(2)
z = inputNode(1)
v1 = Node(v1f,[x.value,y.value,z.value])
v3 = Node(v3f,[v1.value])
v2 = Node(v2f,[x.value,y.value,z.value])
v4 = Node(v4f,[v2.value])
v5 = Node(v5f,[v3.value,v4.value])
v5.backprop([dfdv5],[1],[1])
v4.backprop([dv5dv4],[v3.value,v4.value],[v5.dfvalue])
v3.backprop([dv5dv3],[v3.value,v4.value],[v5.dfvalue])
v2.backprop([dv4dv2],[v2.value],[v4.dfvalue])
v1.backprop([dv3dv1],[v1.value],[v3.dfvalue])
x.backprop([dv1dx,dv2dx],[x.value,y.value,z.value],[v1.dfvalue,v2.dfvalue])
y.backprop([dv1dy,dv2dy],[x.value,y.value,z.value],[v1.dfvalue,v2.dfvalue])
z.backprop([dv1dz,dv2dz],[x.value,y.value,z.value],[v1.dfvalue,v2.dfvalue])

#Computing Numerical Derivatives for checking
h = 1e-5
def f(x,y,z):
```

```
    return (x*y+z)**2*np.exp(x-y*z)
dfdx = (f(1+h,2,1) - f(1,2,1))/h
dfdy = (f(1,2+h,1) - f(1,2,1))/h
dfdz = (f(1,2,1+h) - f(1,2,1))/h

print("Outputing Ndoes with their values for Graph 1:")
print("Node x:")
print("Estimate numerical derivative dfdx is: ", dfdx)
x.printnode()
print("Node y:")
print("Estimate numerical derivative dfdy is: ", dfdy)
y.printnode()
print("Node z:")
print("Estimate numerical derivative dfdz is: ", dfdz)
z.printnode()
print("Node v1:")
v1.printnode()
print("Node v2:")
v2.printnode()
print("Node v3:")
v3.printnode()
print("Node v4:")
v4.printnode()
print("Node v5:")
v5.printnode()
```

```
Outputing Ndoes with their values for Graph 1:
Node x:
Estimate numerical derivative dfdx is:  7.725543680381363
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  7.7254682646002895
Derivative of outgoing Edge # 0  of this node:  4.414553294057308
Derivative of outgoing Edge # 1  of this node:  3.310914970542981
Node y:
Estimate numerical derivative dfdy is:  -1.1036401629027637
forward value:  2
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  -1.103638323514327
Derivative of outgoing Edge # 0  of this node:  2.207276647028654
Derivative of outgoing Edge # 1  of this node:  -3.310914970542981
Node z:
Estimate numerical derivative dfdz is:  -4.414527542673241
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  -4.414553294057308
Derivative of outgoing Edge # 0  of this node:  2.207276647028654
Derivative of outgoing Edge # 1  of this node:  -6.621829941085962
Node v1:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  2.207276647028654
Derivative of outgoing Edge # 0  of this node:  2.207276647028654
Node v2:
forward value:  -1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  3.310914970542981
Derivative of outgoing Edge # 0  of this node:  3.310914970542981
Node v3:
forward value:  9
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.36787944117144233
Derivative of outgoing Edge # 0  of this node:  0.36787944117144233
Node v4:
forward value:  0.36787944117144233
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  9
Derivative of outgoing Edge # 0  of this node:  9
Node v5:
forward value:  3.310914970542981
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
```

In [3]:
```
a = [1]
a.append(2)

a = np.array(a)
b = np.array([3, 4])
a.dot(b)
#np.exp(1)
#len([1,3])
print(np.arctan(1)-np.pi/4)
```

```
0.0
```

In [7]:
```
# Graph # 2
def sigmod(a):
    return 1/(1+np.exp(-a))
def v1f(input_list):
    return input_list[0] * input_list[1] + input_list[0] * input_list[2]
def v3f(input_list):
    return sigmod(input_list[0])
def v2f(input_list):
    return input_list[0] * input_list[2] - input_list[1] * input_list[2]
def v4f(input_list):
    return np.arctan(input_list[0])
def v5f(input_list):
    return input_list[0] * input_list[1]
def dv3dv1(backp_input_list):
    return 2*backp_input_list[0]
def dfdv5(backp_input_list):
    return backp_input_list[0]
def dv5dv4(backp_input_list):
    return backp_input_list[0]
def dv5dv3(backp_input_list):
    return backp_input_list[1]
def dv4dv2(backp_input_list):
    return 1/(1+backp_input_list[0]**2)
def dv3dv1(backp_input_list):
    return sigmod(backp_input_list[0])*(1-sigmod(backp_input_list[0]))
def dv1dx(backp_input_list):
    return backp_input_list[1] + backp_input_list[2]
def dv2dx(backp_input_list):
    return backp_input_list[2]
def dv1dy(backp_input_list):
    return backp_input_list[0]
def dv2dy(backp_input_list):
    return -backp_input_list[2]
def dv1dz(backp_input_list):
    return backp_input_list[0]
def dv2dz(backp_input_list):
    return backp_input_list[0] - backp_input_list[1]
x = inputNode(1)
y = inputNode(1)
z = inputNode(1)
v1 = Node(v1f,[x.value,y.value,z.value])
v3 = Node(v3f,[v1.value])
v2 = Node(v2f,[x.value,y.value,z.value])
v4 = Node(v4f,[v2.value])
v5 = Node(v5f,[v3.value,v4.value])
v5.backprop([dfdv5],[1],[1])
v4.backprop([dv5dv4],[v3.value,v4.value],[v5.dfvalue])
v3.backprop([dv5dv3],[v3.value,v4.value],[v5.dfvalue])
v2.backprop([dv4dv2],[v2.value],[v4.dfvalue])
v1.backprop([dv3dv1],[v1.value],[v3.dfvalue])
x.backprop([dv1dx,dv2dx],[x.value,y.value,z.value],[v1.dfvalue,v2.dfvalue])
y.backprop([dv1dy,dv2dy],[x.value,y.value,z.value],[v1.dfvalue,v2.dfvalue])
z.backprop([dv1dz,dv2dz],[x.value,y.value,z.value],[v1.dfvalue,v2.dfvalue])

#Computing Numerical Derivatives for checking
h = 1e-5
```

```
def f(x,y,z):
    return sigmod(x*y+x*z)*np.arctan(x*z-y*z)
dfdx = (f(1+h,1,1) - f(1,1,1))/h
dfdy = (f(1,1+h,1) - f(1,1,1))/h
dfdz = (f(1,1,1+h) - f(1,1,1))/h

print("Outputing Ndoes with their values for Graph 2:")
print("Node x:")
print("Estimate numerical derivative dfdx is: ", dfdx)
x.printnode()
print("Node y:")
print("Estimate numerical derivative dfdy is: ", dfdy)
y.printnode()
print("Node z:")
print("Estimate numerical derivative dfdz is: ", dfdz)
z.printnode()
print("Node v1:")
v1.printnode()
print("Node v2:")
v2.printnode()
print("Node v3:")
v3.printnode()
print("Node v4:")
v4.printnode()
print("Node v5:")
v5.printnode()
```

```
Outputing Ndoes with their values for Graph 2:
Node x:
Estimate numerical derivative dfdx is:  0.8807991778100084
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.8807970779778823
Derivative of outgoing Edge # 0  of this node:  0.0
Derivative of outgoing Edge # 1  of this node:  0.8807970779778823
Node y:
Estimate numerical derivative dfdy is:  -0.8807981278861488
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  -0.8807970779778823
Derivative of outgoing Edge # 0  of this node:  0.0
Derivative of outgoing Edge # 1  of this node:  -0.8807970779778823
Node z:
Estimate numerical derivative dfdz is:  0.0
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.0
Derivative of outgoing Edge # 0  of this node:  0.0
Derivative of outgoing Edge # 1  of this node:  0.0
Node v1:
forward value:  2
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.0
Derivative of outgoing Edge # 0  of this node:  0.0
Node v2:
forward value:  0
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.8807970779778823
Derivative of outgoing Edge # 0  of this node:  0.8807970779778823
Node v3:
forward value:  0.8807970779778823
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.0
Derivative of outgoing Edge # 0  of this node:  0.0
Node v4:
forward value:  0.0
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.8807970779778823
Derivative of outgoing Edge # 0  of this node:  0.8807970779778823
Node v5:
forward value:  0.0
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
```

In [9]:
```
# Graph # 3
def sigmod(a):
    return 1/(1+np.exp(-a))
def v1f(input_list):
    return input_list[0] * 2 + input_list[1]
def v3478f(input_list):
    return sigmod(input_list[0])
def v2f(input_list):
    return input_list[0] - input_list[1] * 2
def v4f(input_list):
    return np.arctan(input_list[0])
def v5f(input_list):
    return input_list[0] - input_list[1]
def v6f(input_list):
    return input_list[0] + input_list[1]
def v9f(input_list):
    return input_list[0] + input_list[1]

def dfdv9(backp_input_list):
    return backp_input_list[0]
def dv9dv78(backp_input_list):
    return 1
def dv3478dv1256(backp_input_list):
    return sigmod(backp_input_list[0])*(1-sigmod(backp_input_list[0]))
def dv5dv4(backp_input_list):
    return -1
def dv5dv3(backp_input_list):
    return 1
def dv6dv3(backp_input_list):
    return 1
def dv6dv4(backp_input_list):
    return 1
def dv1dx(backp_input_list):
    return 2
def dv2dx(backp_input_list):
    return 1
def dv1dy(backp_input_list):
    return 1
def dv2dy(backp_input_list):
    return -2

x = inputNode(1)
y = inputNode(1)

v1 = Node(v1f,[x.value,y.value])
v2 = Node(v2f,[x.value,y.value])
v3 = Node(v3478f,[v1.value])
v4 = Node(v3478f,[v2.value])
v5 = Node(v5f,[v3.value,v4.value])
v6 = Node(v6f,[v3.value,v4.value])
v7 = Node(v3478f,[v5.value])
v8 = Node(v3478f,[v6.value])
v9 = Node(v9f,[v7.value, v8.value])
v9.backprop([dfdv9],[1],[1])
v8.backprop([dv9dv78],[v7.value,v8.value],[v9.dfvalue])
v7.backprop([dv9dv78],[v7.value,v8.value],[v9.dfvalue])
```

```
v6.backprop([dv3478dv1256],[v6.value],[v8.dfvalue])
v5.backprop([dv3478dv1256],[v5.value],[v7.dfvalue])
v4.backprop([dv5dv4,dv6dv4],[v3.value,v4.value],[v5.dfvalue,v6.dfvalue])
v3.backprop([dv5dv3,dv6dv3],[v3.value,v4.value],[v5.dfvalue,v6.dfvalue])
v2.backprop([dv3478dv1256],[v2.value],[v4.dfvalue])
v1.backprop([dv3478dv1256],[v1.value],[v3.dfvalue])
x.backprop([dv1dx,dv2dx],[x.value,y.value],[v1.dfvalue,v2.dfvalue])
y.backprop([dv1dy,dv2dy],[x.value,y.value],[v1.dfvalue,v2.dfvalue])

#Computing Numerical Derivatives for checking
h = 1e-5
def f(x,y):
    return sigmod(sigmod(2*x+y)-sigmod(x-2*y)) + sigmod(sigmod(2*x+y)+sigmod(x
-2*y))
dfdx = (f(1+h,1) - f(1,1))/h
dfdy = (f(1,1+h) - f(1,1))/h
#dfdz = (f(1,1,1+h) - f(1,1,1))/h

print("Outputing Ndoes with their values for Graph 3:")
print("Node x:")
print("Estimate numerical derivative dfdx is: ", dfdx)
x.printnode()
print("Node y:")
print("Estimate numerical derivative dfdy is: ", dfdy)
y.printnode()

print("Node v1:")
v1.printnode()
print("Node v2:")
v2.printnode()
print("Node v3:")
v3.printnode()
print("Node v4:")
v4.printnode()
print("Node v5:")
v5.printnode()
print("Node v6:")
v6.printnode()
print("Node v7:")
v7.printnode()
print("Node v8:")
v8.printnode()
print("Node v9:")
v9.printnode()
```

```
Outputing Ndoes with their values for Graph 3:
Node x:
Estimate numerical derivative dfdx is:  0.026770958805322206
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.026771349914508743
Derivative of outgoing Edge # 0  of this node:  0.03602928228463276
Derivative of outgoing Edge # 1  of this node:  -0.009257932370124015
Node y:
Estimate numerical derivative dfdy is:  0.03653021025673553
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.03653050588256441
Derivative of outgoing Edge # 0  of this node:  0.01801464114231638
Derivative of outgoing Edge # 1  of this node:  0.01851586474024803
Node v1:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.01801464114231638
Derivative of outgoing Edge # 0  of this node:  0.01801464114231638
Node v2:
forward value:  -1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  -0.009257932370124015
Derivative of outgoing Edge # 0  of this node:  -0.009257932370124015
Node v3:
forward value:  0.9525741268224334
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.39875991827678026
Derivative of outgoing Edge # 0  of this node:  0.2229236276672817
Derivative of outgoing Edge # 1  of this node:  0.17583629060949854
Node v4:
forward value:  0.2689414213699951
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  -0.047087337057783146
Derivative of outgoing Edge # 0  of this node:  -0.2229236276672817
Derivative of outgoing Edge # 1  of this node:  0.17583629060949854
Node v5:
forward value:  0.6836327054524383
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.2229236276672817
Derivative of outgoing Edge # 0  of this node:  0.2229236276672817
Node v6:
forward value:  1.2215155481924285
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0.17583629060949854
Derivative of outgoing Edge # 0  of this node:  0.17583629060949854
Node v7:
forward value:  0.6645489967539101
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
Node v8:
forward value:  0.7723301477811472
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
```

```
Node v9:
forward value:  1.4368791445350573
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
```

In [10]:
```
# Graph # 4
def ReLu(x):
    if x > 0:
        return x
    else:
        return 0
def dReLudx(x):
    if x > 0:
        return 1
    else:
        return 0
def v1f(input_list):
    return input_list[0] * 2 + input_list[1]
def v3478f(input_list):
    return ReLu(input_list[0])
def v2f(input_list):
    return input_list[0] - input_list[1] * 2
def v4f(input_list):
    return np.arctan(input_list[0])
def v5f(input_list):
    return input_list[0] - input_list[1]
def v6f(input_list):
    return input_list[0] + input_list[1]
def v9f(input_list):
    return input_list[0] + input_list[1]

def dfdv9(backp_input_list):
    return backp_input_list[0]
def dv9dv78(backp_input_list):
    return 1
def dv3478dv1256(backp_input_list):
    return dReLudx(backp_input_list[0])
def dv5dv4(backp_input_list):
    return -1
def dv5dv3(backp_input_list):
    return 1
def dv6dv3(backp_input_list):
    return 1
def dv6dv4(backp_input_list):
    return 1
def dv1dx(backp_input_list):
    return 2
def dv2dx(backp_input_list):
    return 1
def dv1dy(backp_input_list):
    return 1
def dv2dy(backp_input_list):
    return -2

x = inputNode(1)
y = inputNode(1)

v1 = Node(v1f,[x.value,y.value])
v2 = Node(v2f,[x.value,y.value])
v3 = Node(v3478f,[v1.value])
v4 = Node(v3478f,[v2.value])
```

```
v5 = Node(v5f,[v3.value,v4.value])
v6 = Node(v6f,[v3.value,v4.value])
v7 = Node(v3478f,[v5.value])
v8 = Node(v3478f,[v6.value])
v9 = Node(v9f,[v7.value, v8.value])
v9.backprop([dfdv9],[1],[1])
v8.backprop([dv9dv78],[v7.value,v8.value],[v9.dfvalue])
v7.backprop([dv9dv78],[v7.value,v8.value],[v9.dfvalue])
v6.backprop([dv3478dv1256],[v6.value],[v8.dfvalue])
v5.backprop([dv3478dv1256],[v5.value],[v7.dfvalue])
v4.backprop([dv5dv4,dv6dv4],[v3.value,v4.value],[v5.dfvalue,v6.dfvalue])
v3.backprop([dv5dv3,dv6dv3],[v3.value,v4.value],[v5.dfvalue,v6.dfvalue])
v2.backprop([dv3478dv1256],[v2.value],[v4.dfvalue])
v1.backprop([dv3478dv1256],[v1.value],[v3.dfvalue])
x.backprop([dv1dx,dv2dx],[x.value,y.value],[v1.dfvalue,v2.dfvalue])
y.backprop([dv1dy,dv2dy],[x.value,y.value],[v1.dfvalue,v2.dfvalue])

#Computing Numerical Derivatives for checking
h = 1e-5
def f(x,y):
    return ReLu(ReLu(2*x+y)-ReLu(x-2*y)) + ReLu(ReLu(2*x+y)+ReLu(x-2*y))
dfdx = (f(1+h,1) - f(1,1))/h
dfdy = (f(1,1+h) - f(1,1))/h
#dfdz = (f(1,1,1+h) - f(1,1,1))/h

print("Outputing Ndoes with their values for Graph 3:")
print("Node x:")
print("Estimate numerical derivative dfdx is: ", dfdx)
x.printnode()
print("Node y:")
print("Estimate numerical derivative dfdy is: ", dfdy)
y.printnode()
#print("Node z:")
#z.printnode()
print("Node v1:")
v1.printnode()
print("Node v2:")
v2.printnode()
print("Node v3:")
v3.printnode()
print("Node v4:")
v4.printnode()
print("Node v5:")
v5.printnode()
print("Node v6:")
v6.printnode()
print("Node v7:")
v7.printnode()
print("Node v8:")
v8.printnode()
print("Node v9:")
v9.printnode()
```

```
Outputing Ndoes with their values for Graph 3:
Node x:
Estimate numerical derivative dfdx is:  4.000000000026205
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  4
Derivative of outgoing Edge # 0  of this node:  4
Derivative of outgoing Edge # 1  of this node:  0
Node y:
Estimate numerical derivative dfdy is:  2.0000000000131024
forward value:  1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  2
Derivative of outgoing Edge # 0  of this node:  2
Derivative of outgoing Edge # 1  of this node:  0
Node v1:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  2
Derivative of outgoing Edge # 0  of this node:  2
Node v2:
forward value:  -1
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0
Derivative of outgoing Edge # 0  of this node:  0
Node v3:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  2
Derivative of outgoing Edge # 0  of this node:  1
Derivative of outgoing Edge # 1  of this node:  1
Node v4:
forward value:  0
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  0
Derivative of outgoing Edge # 0  of this node:  -1
Derivative of outgoing Edge # 1  of this node:  1
Node v5:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
Node v6:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
Node v7:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
Node v8:
forward value:  3
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
```

```
Node v9:
forward value:  6
dfvalue (the sum of derivatives of all outgoing edges/the partial derivative
of this node with respect to f):  1
Derivative of outgoing Edge # 0  of this node:  1
```