The Lengauer Tarjan Algorithm for Computing the Immediate Dominator Tree of a Flowgraph

by

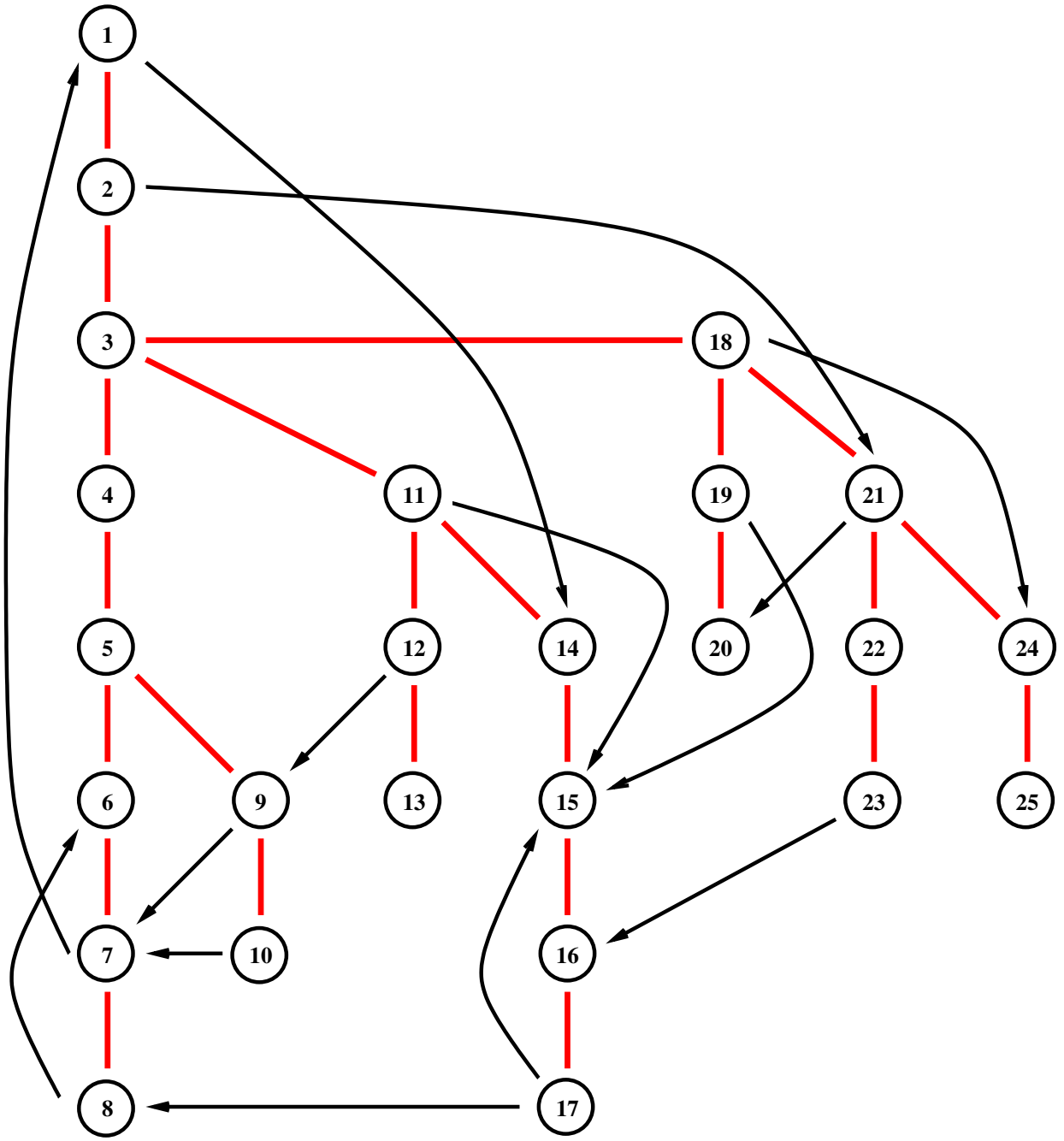
Martin Richards

mr@cl.cam.ac.uk

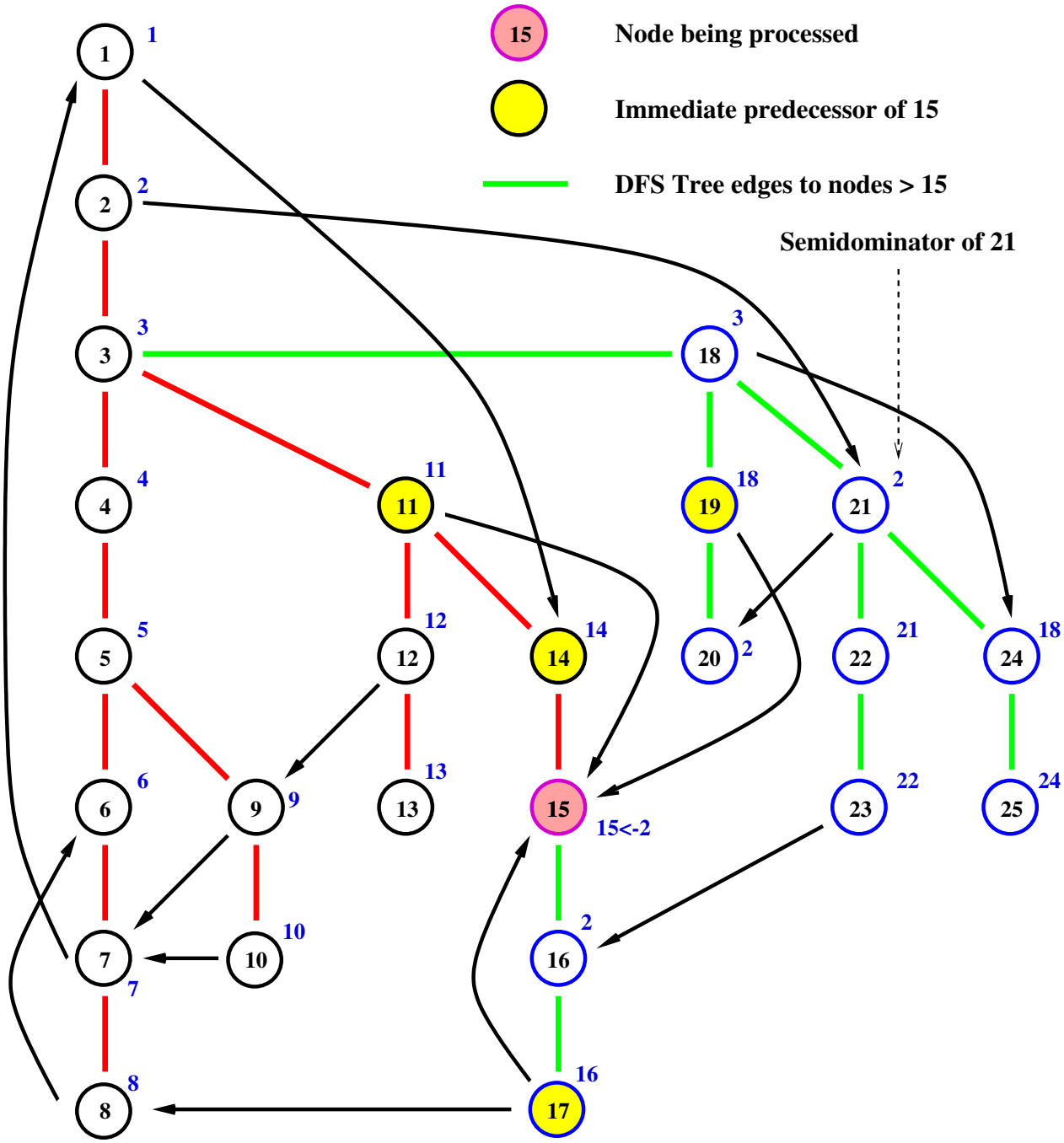
http://www.cl.cam.ac.uk/~mr

University Computer Laboratory
New Museum Site
Pembroke Street
Cambridge, CB2 3QG

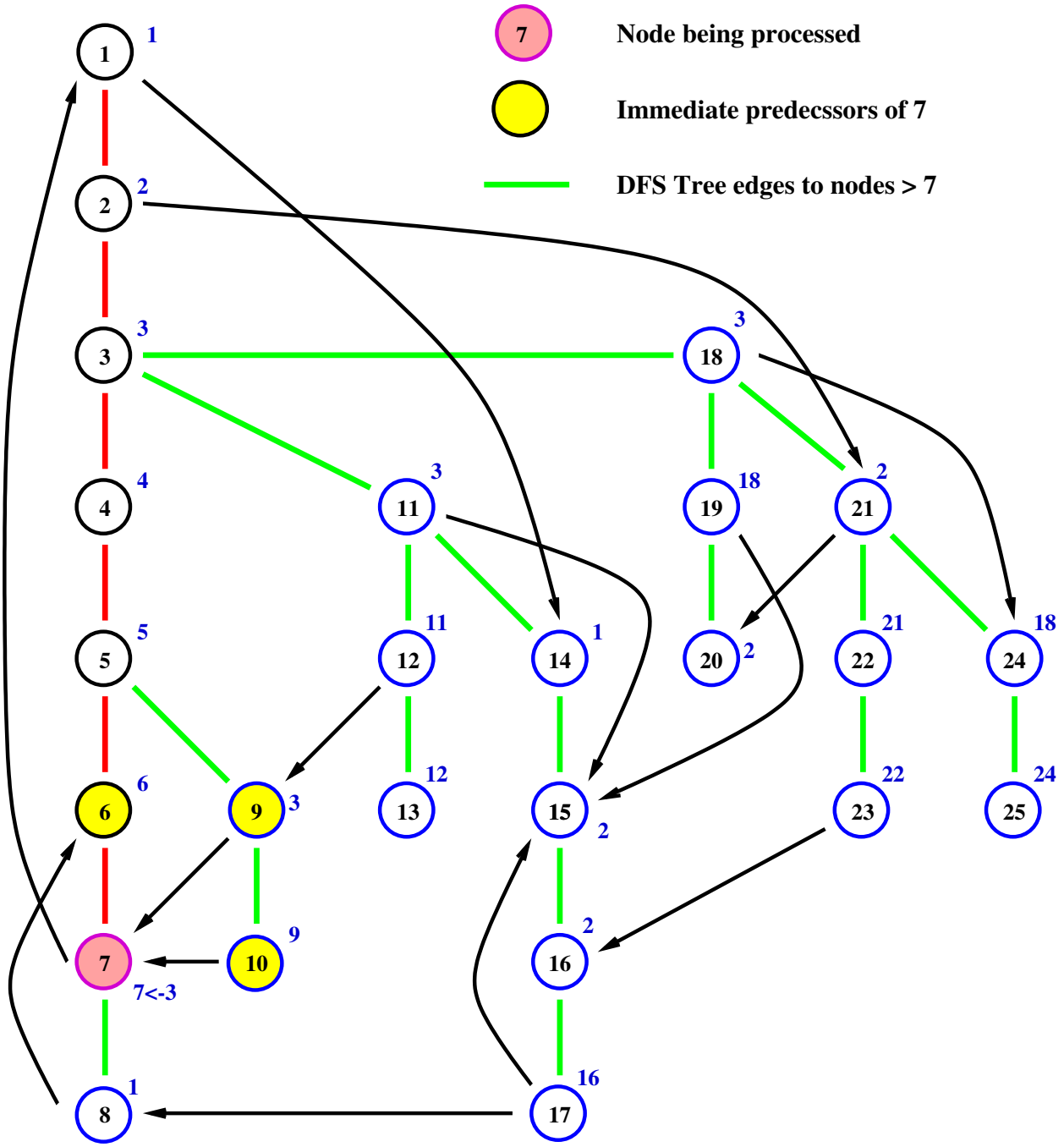
DFS of the Flowgraph



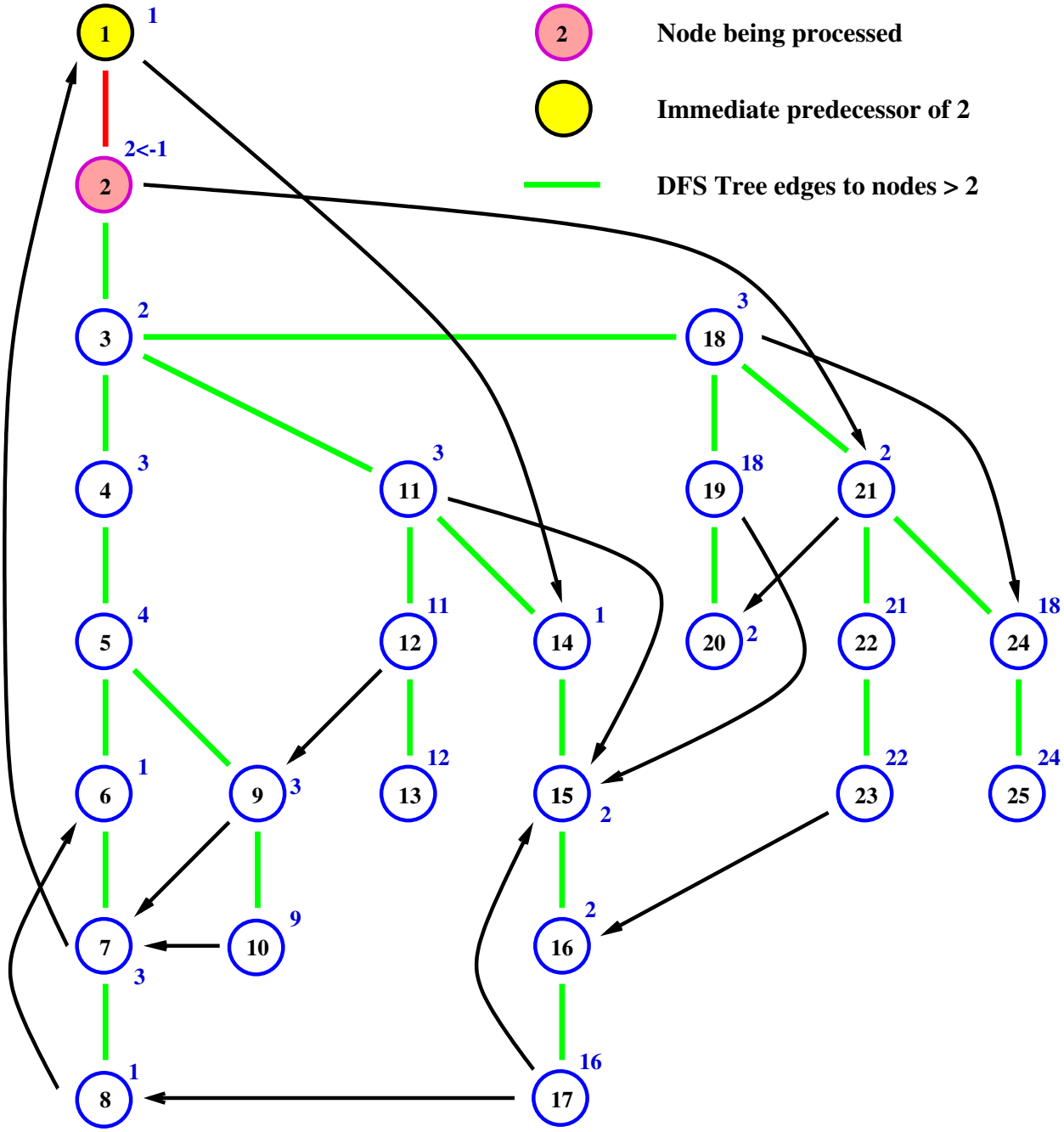
Processing Node 15



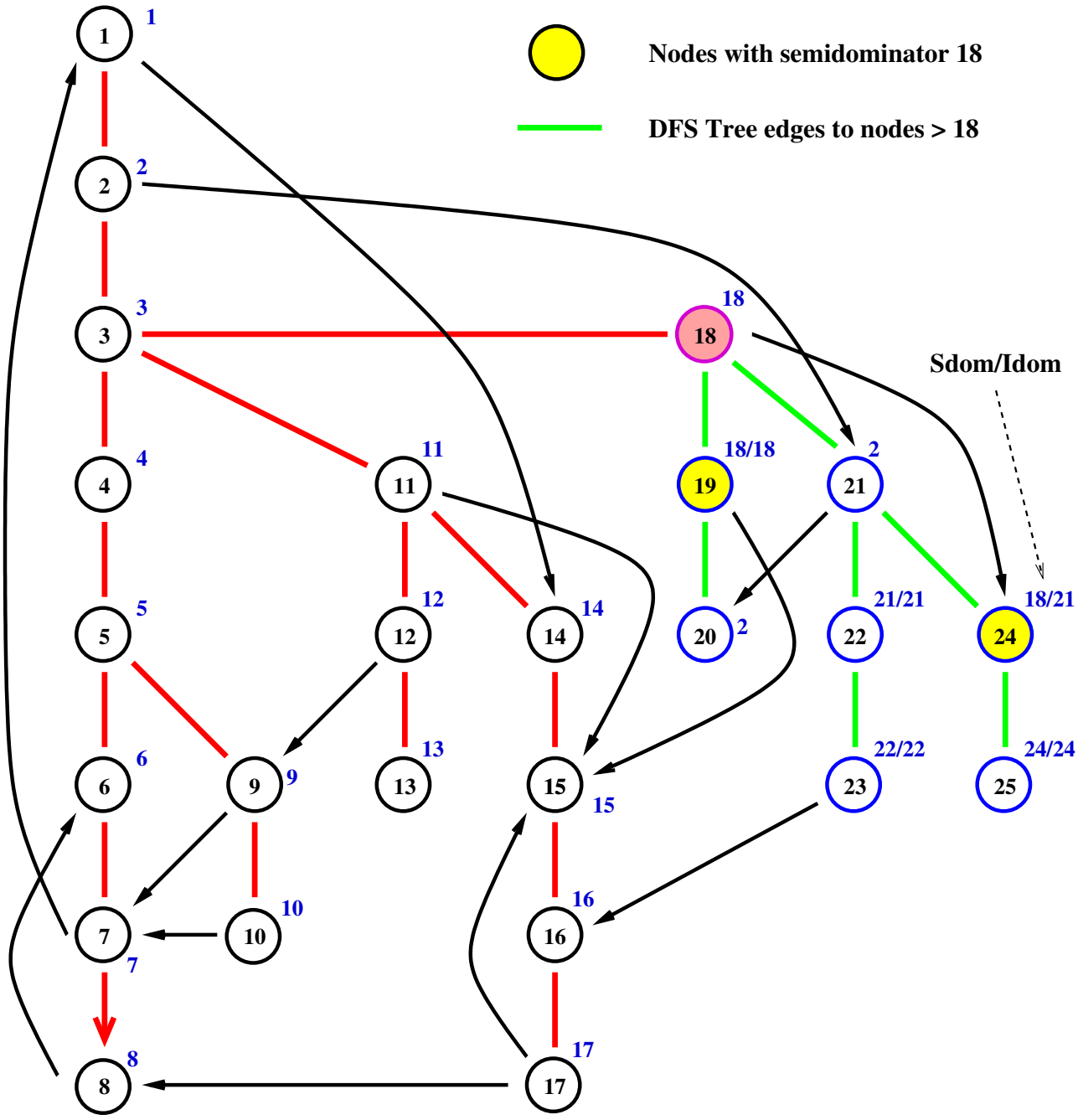
Processing Node 7



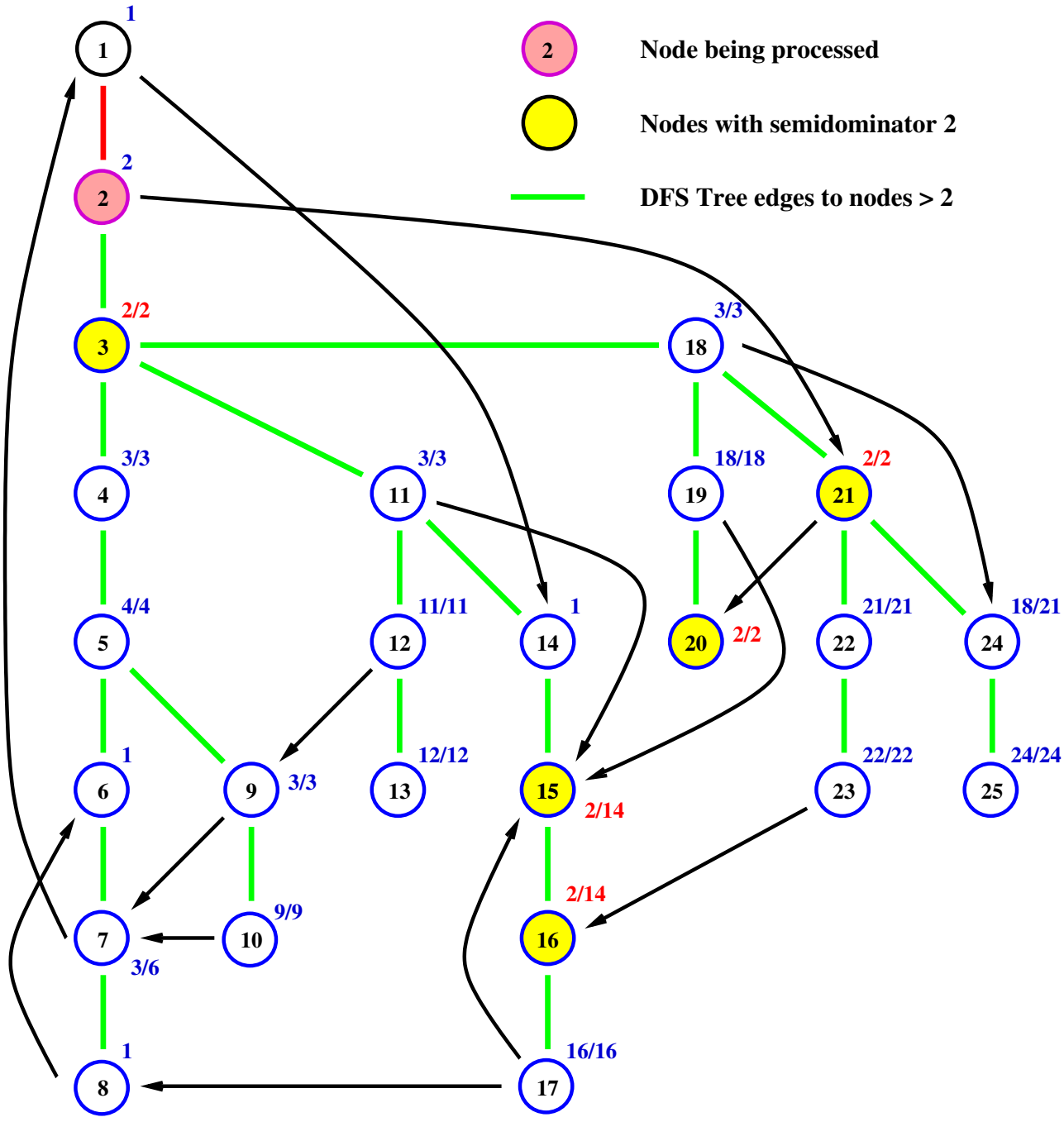
Processing Node 2



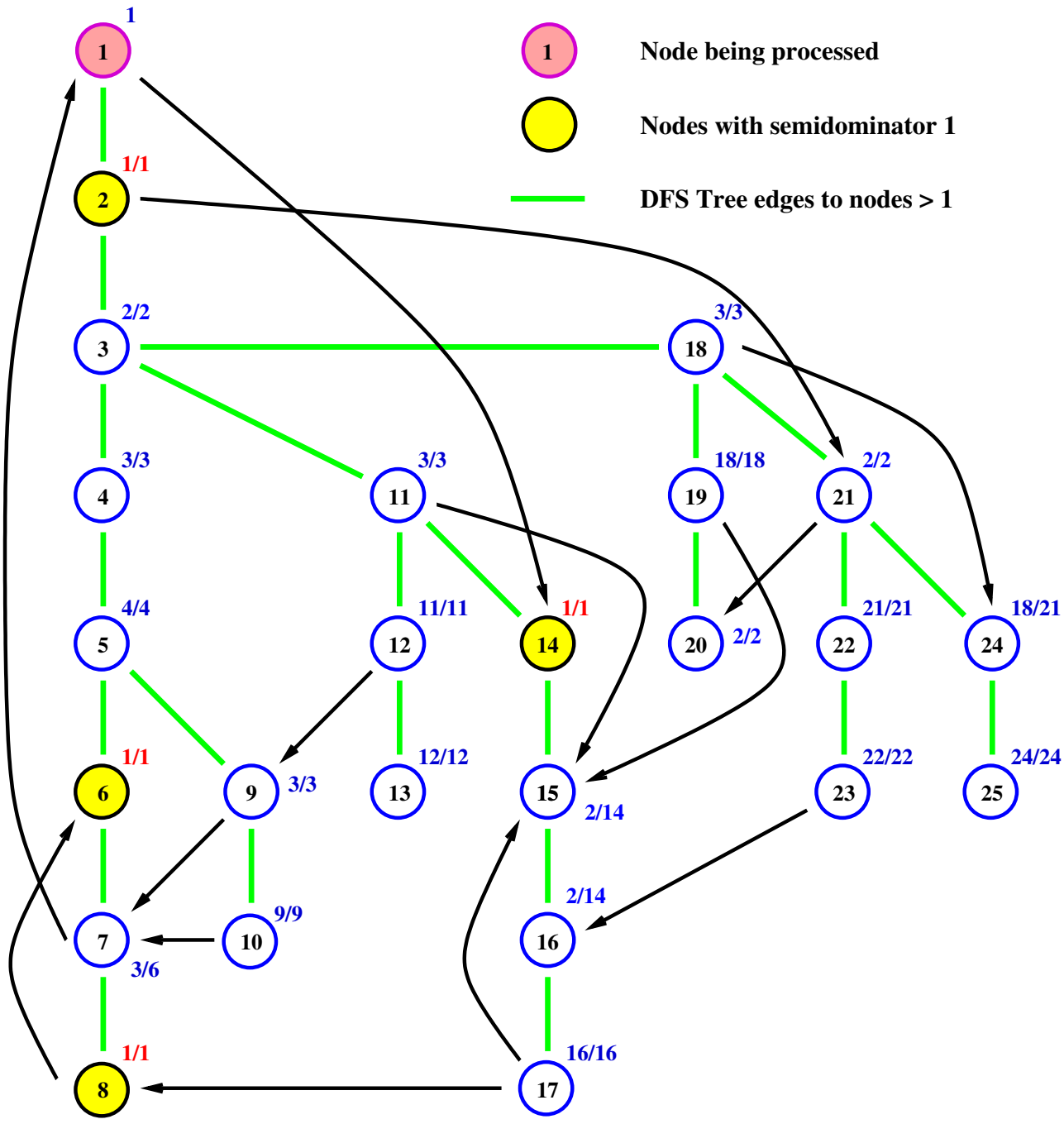
Nodes with Semidominator 18



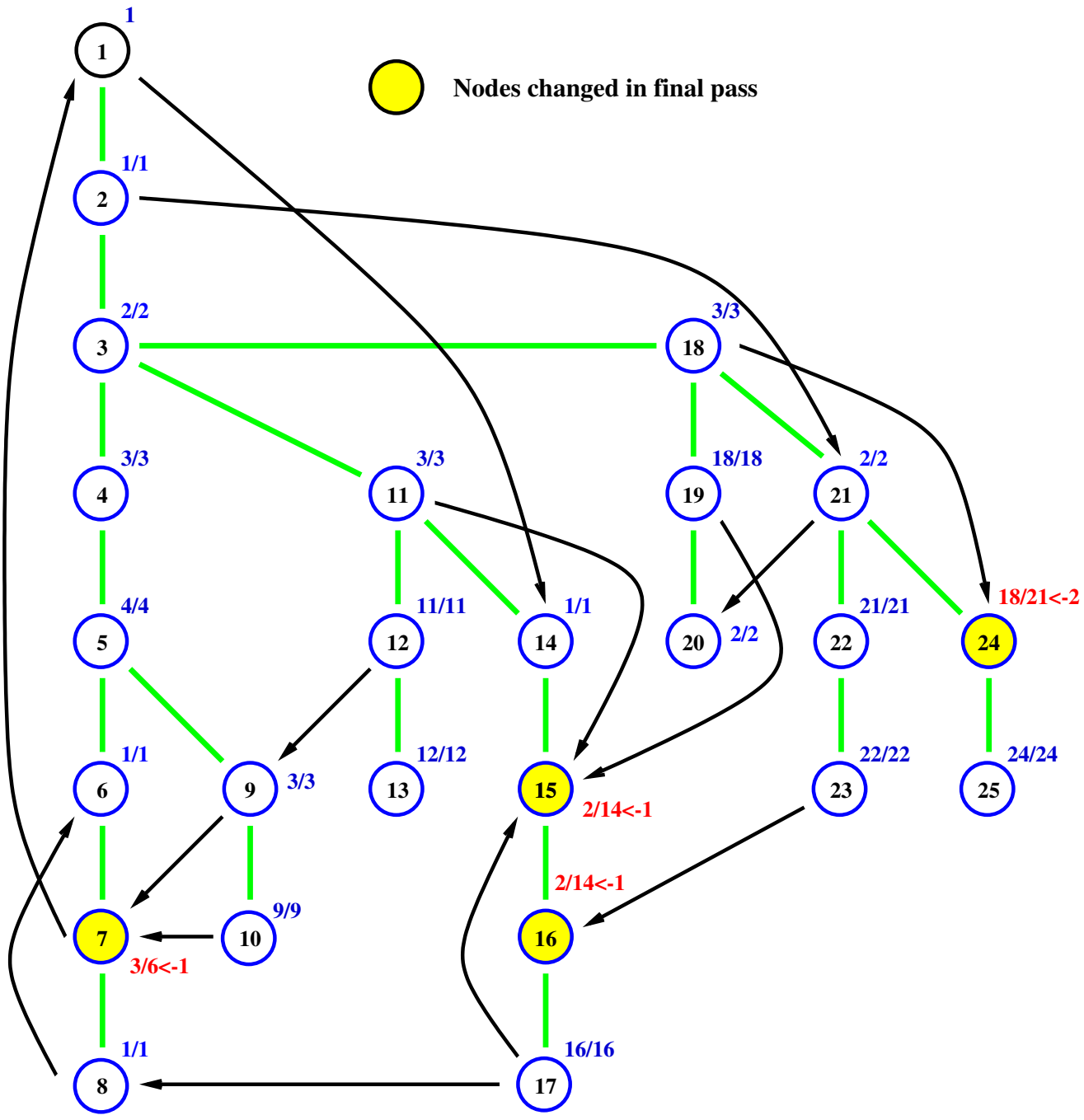
Nodes with Semidominator 2



Nodes with Semidominator 1



Final Phase


Nodes changed in final pass
1
1
1/1
2
2/2
3
3/3
18
3/3
4
3/3
11
18/18
19
2/2
21
4/4
5
11/11
12
1/1
14
2/2
20
21/21
22
18/21<-2
24
1/1
6
9
3/3
12/12
13
15
2/14<-1
22/22
23
24/24
25
2/14<-1
9/9
7
10
3/6<-1
16
1/1
8
16/16
17

Step 1: Initialisation

Number the vertices is depth first search order from 1 to `n`.

For each vertex `v` from 1 to `n` set:

```
parent[v]     :=   DFS tree parent of v
succs[v]      :=   list of successors
preds[v]      :=   list of predecessors

semi[v]       :=   v
idom[v]       :=   0

ancestor[v]   :=   0
best[v]       :=   v
bucket[v]     :=   0
```

Steps 2 and 3

```
FOR w = n TO 2 BY -1 DO
{       LET p = parent[w]

step2:  FOR each v in preds[w] DO
        { LET u = EVAL(v)
          IF semi[w] >  semi[u] DO
             semi[w] := semi[u]
        }
        add w to bucket[semi[w]]
        LINK(p, w)

step3:  FOR each v in bucket[p]
        { LET u = EVAL(v)
          // Note:   semi[v] is p
          idom[v] := semi[u]<p -> u, p
        }
        bucket[p] := 0
}
```

Steps 4

```
step4: FOR w = 2 TO n DO
         UNLESS idom[w] = semi[w] DO
                idom[w] := idom[idom[w]]
       idom[1] := 0
```

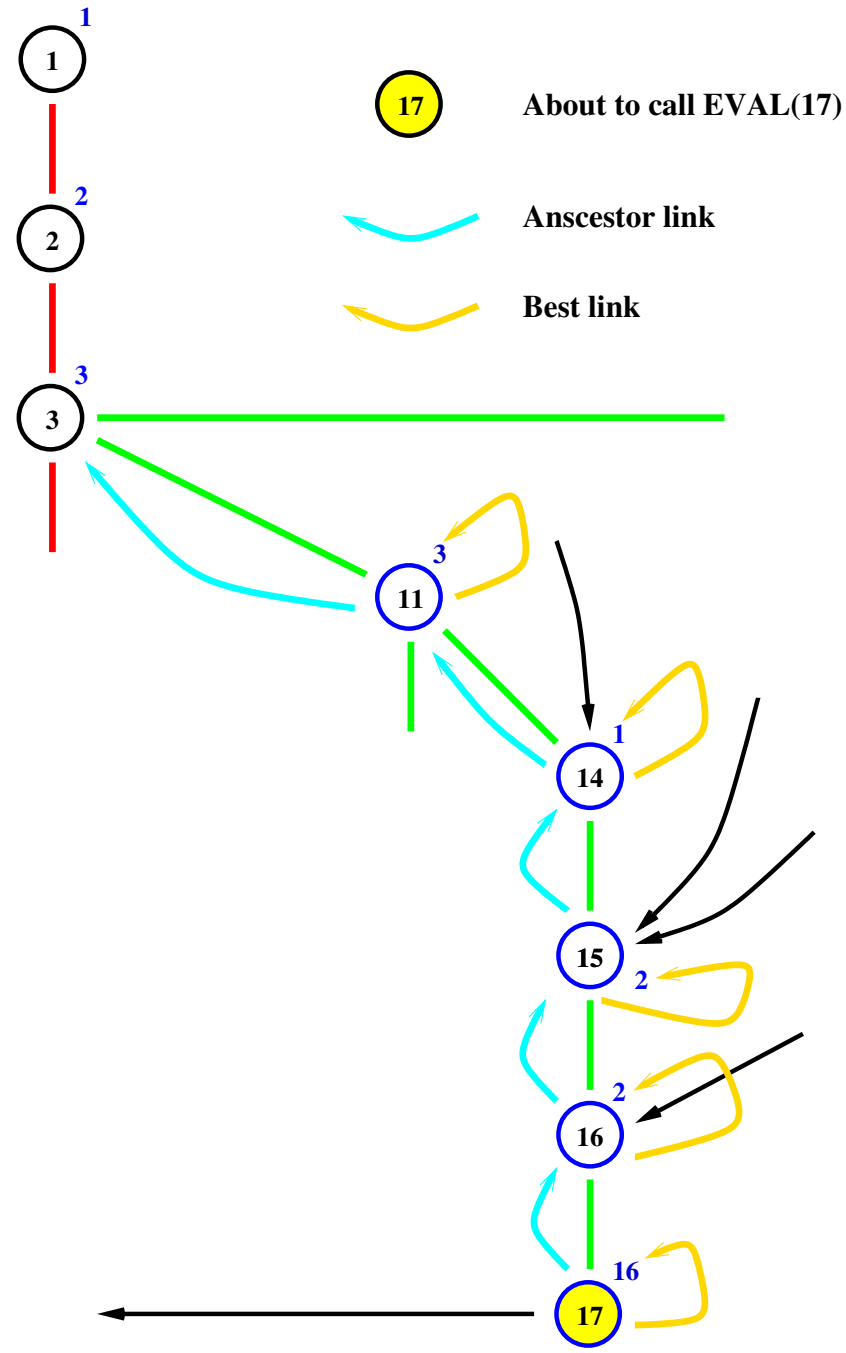
Very Simple LINK and EVAL

```
LET LINK(v, w) BE ancestor[w] := v

LET EVAL(v) = VALOF
{ LET a = ancestor[v]

  WHILE ancestor[a] DO
  { IF semi[v] > semi[a] DO v := a
    a := ancestor[a]
  }

  // v is now a vertex
  //   with smallest semidominator
  //   of any in the ancestor chain.

  RESULTIS v
}
```

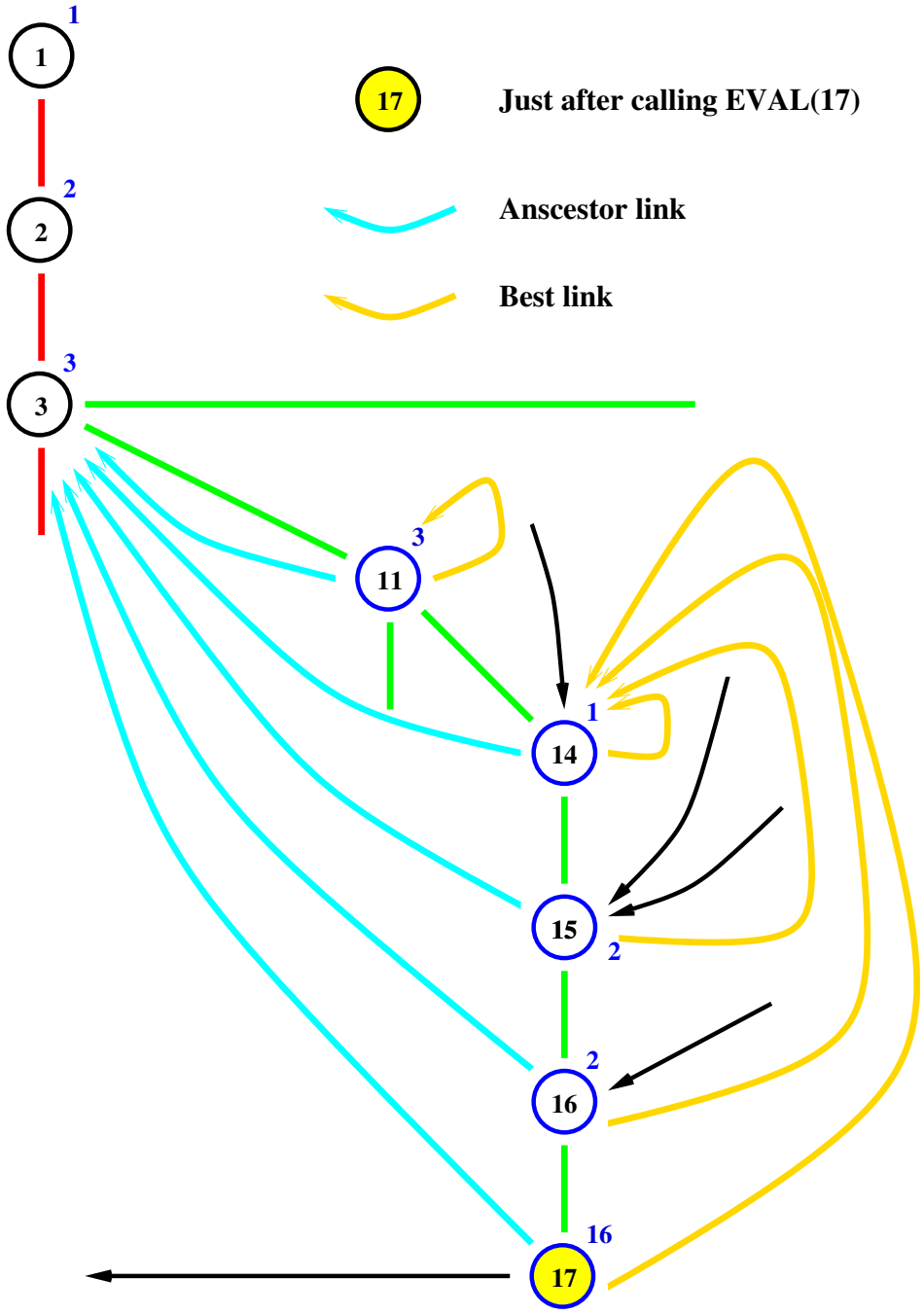
Simple LINK and EVAL

```
LET LINK(v, w) BE ancestor[w] := v

LET EVAL(v) = VALOF
{ UNLESS ancestor[v] RESULTIS v
  COMPRESS(v)
  RESULTIS best[v]
}

AND COMPRESS(v) BE
{ LET a = ancestor[v]

  UNLESS ancestor[a] RETURN

  COMPRESS(a)

  IF semi[best[v]]  >  semi[best[a]] DO
          best[v]   :=      best[a]

  ancestor[v] := ancestor[a]
}
```

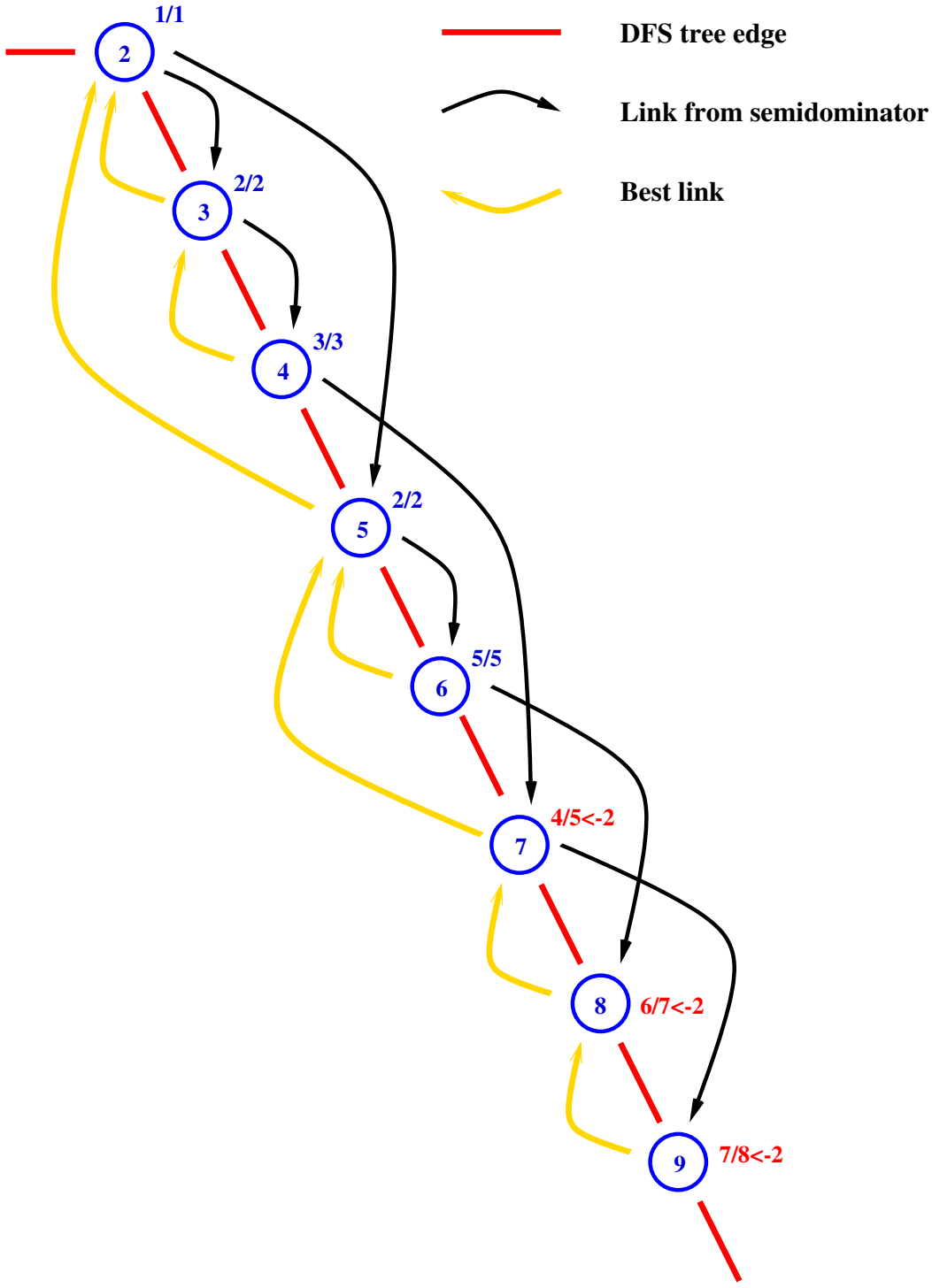
Before calling EVAL(17)



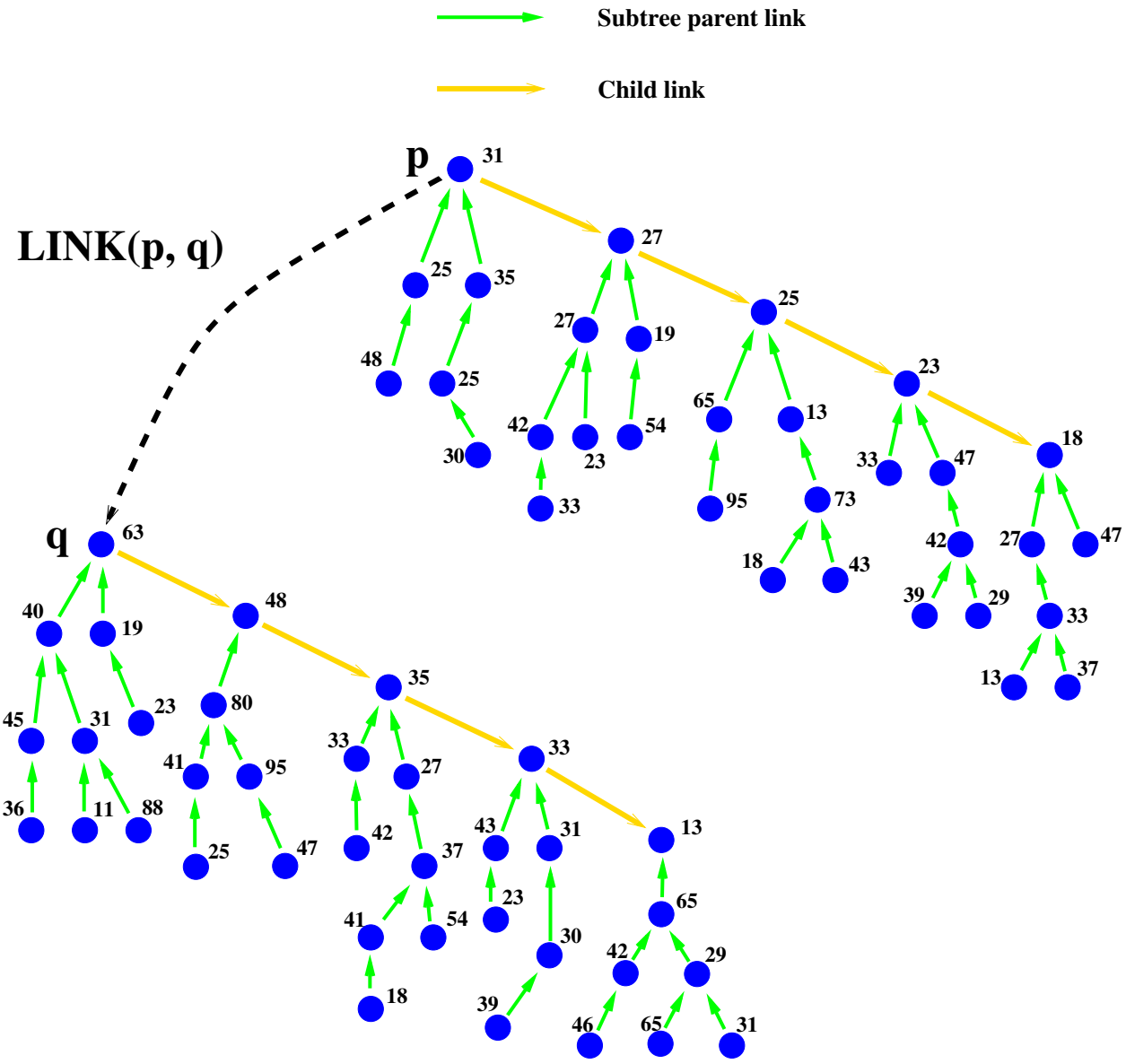
After calling EVAL(17)



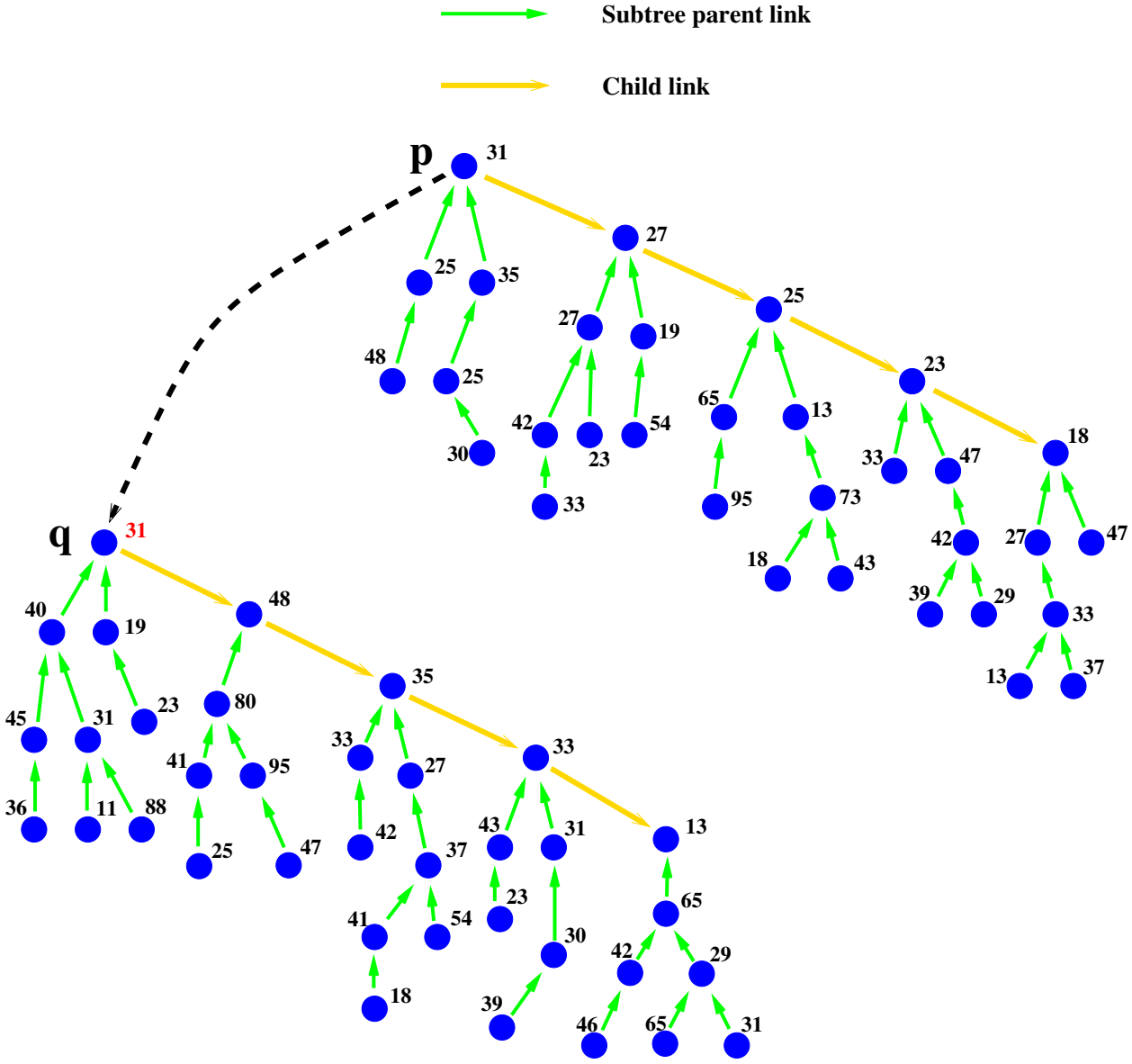
Step 4 Example



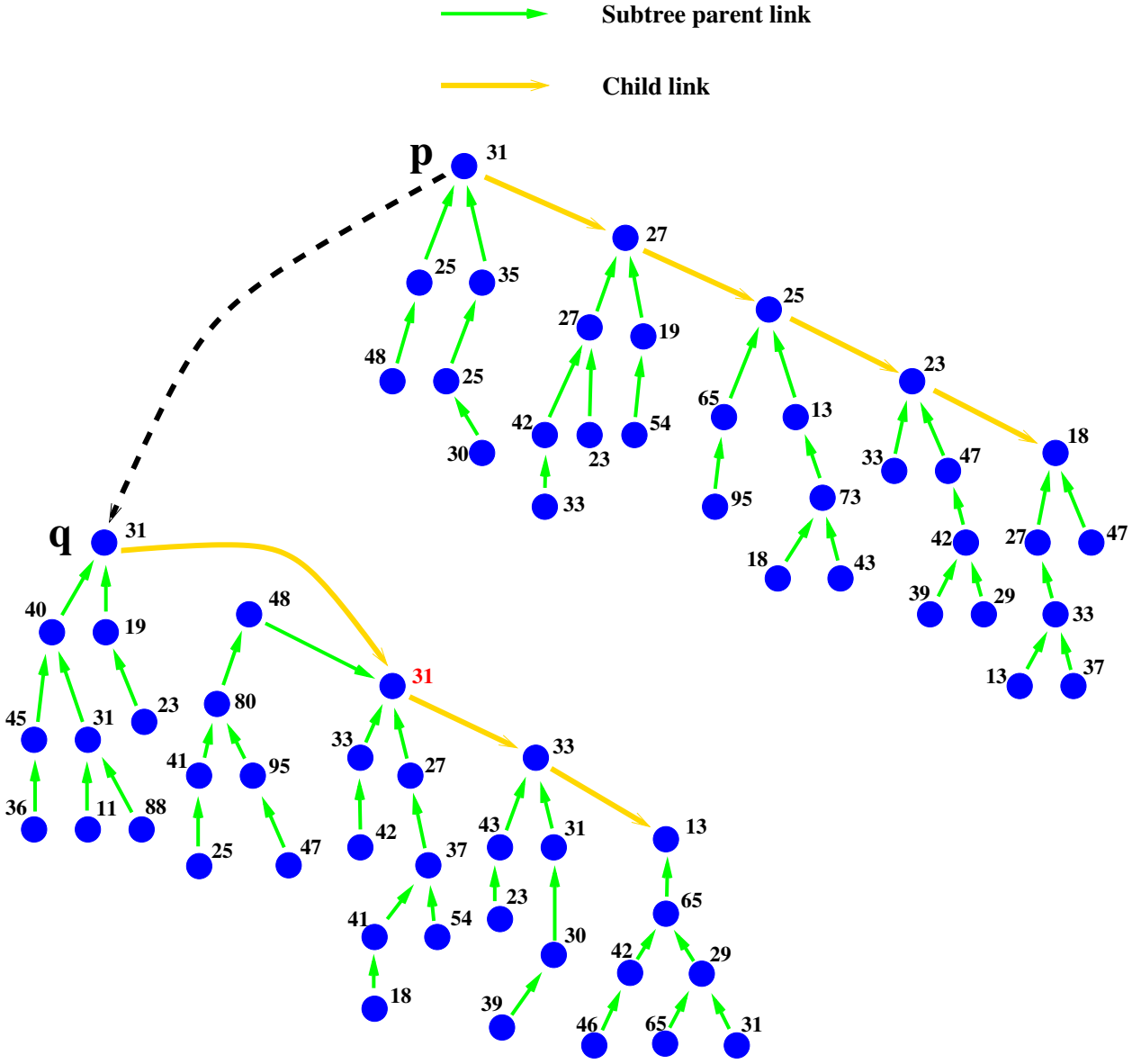
Balanced Trees



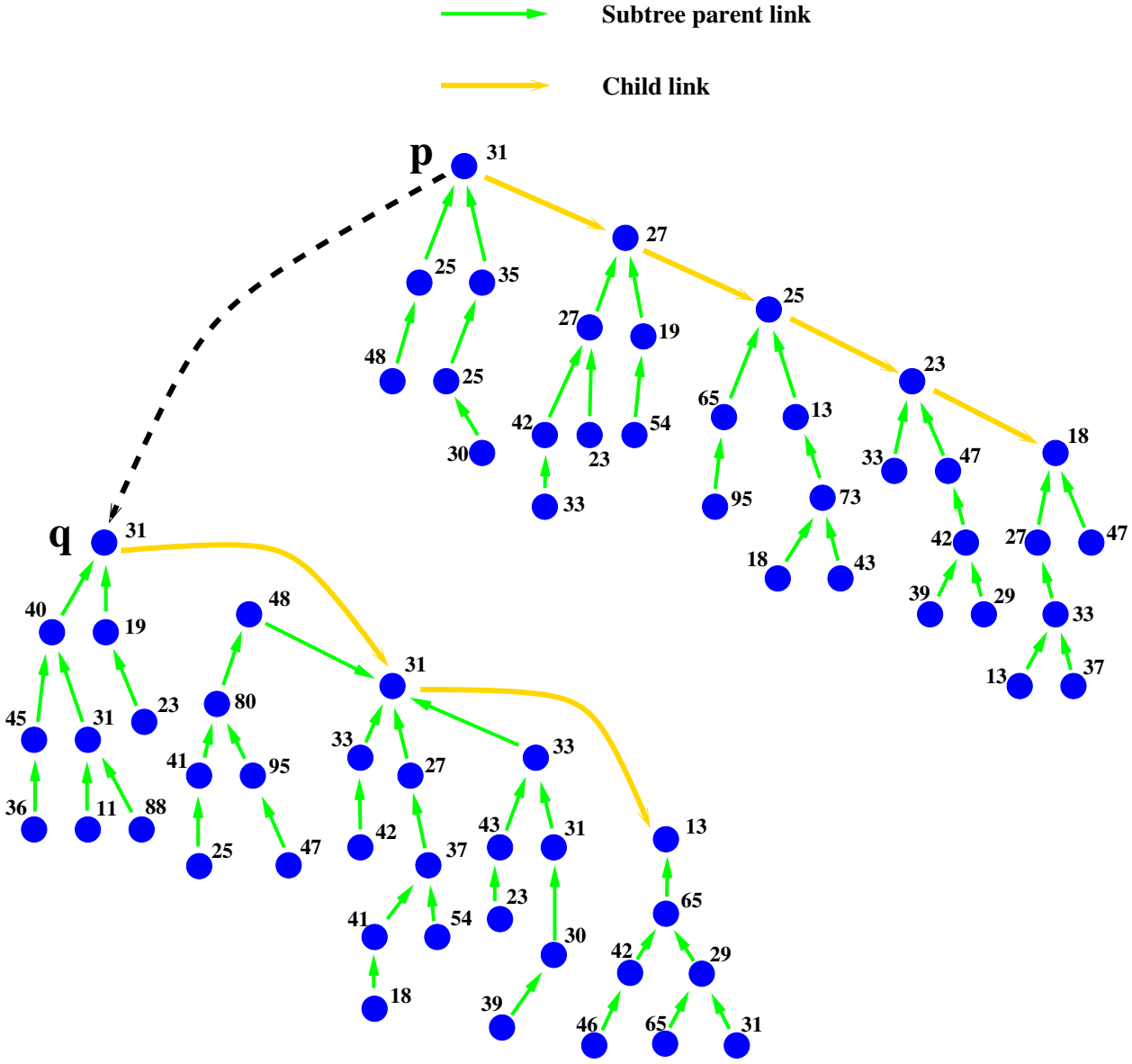
Balanced Trees 1



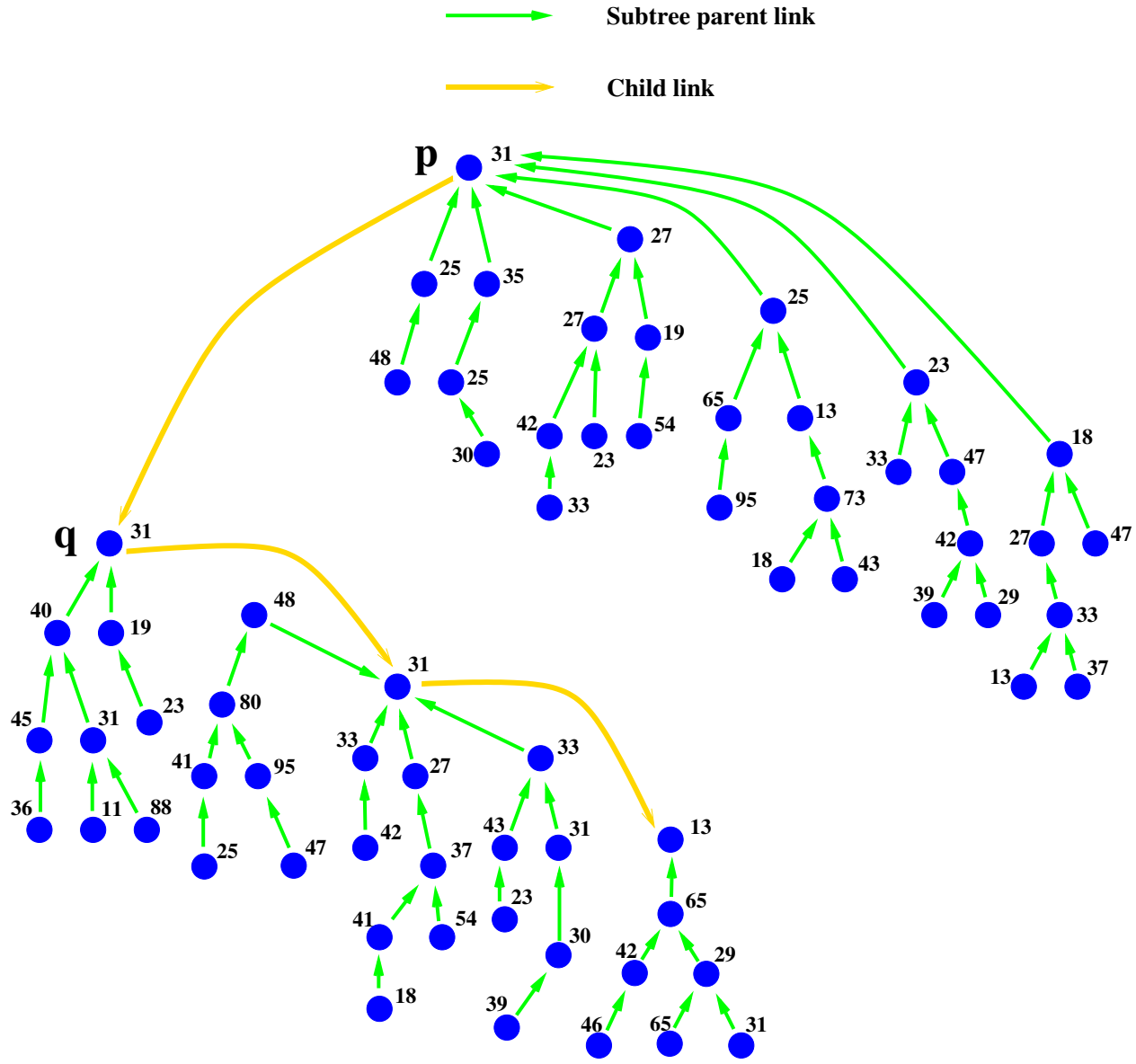
Balanced Trees 2



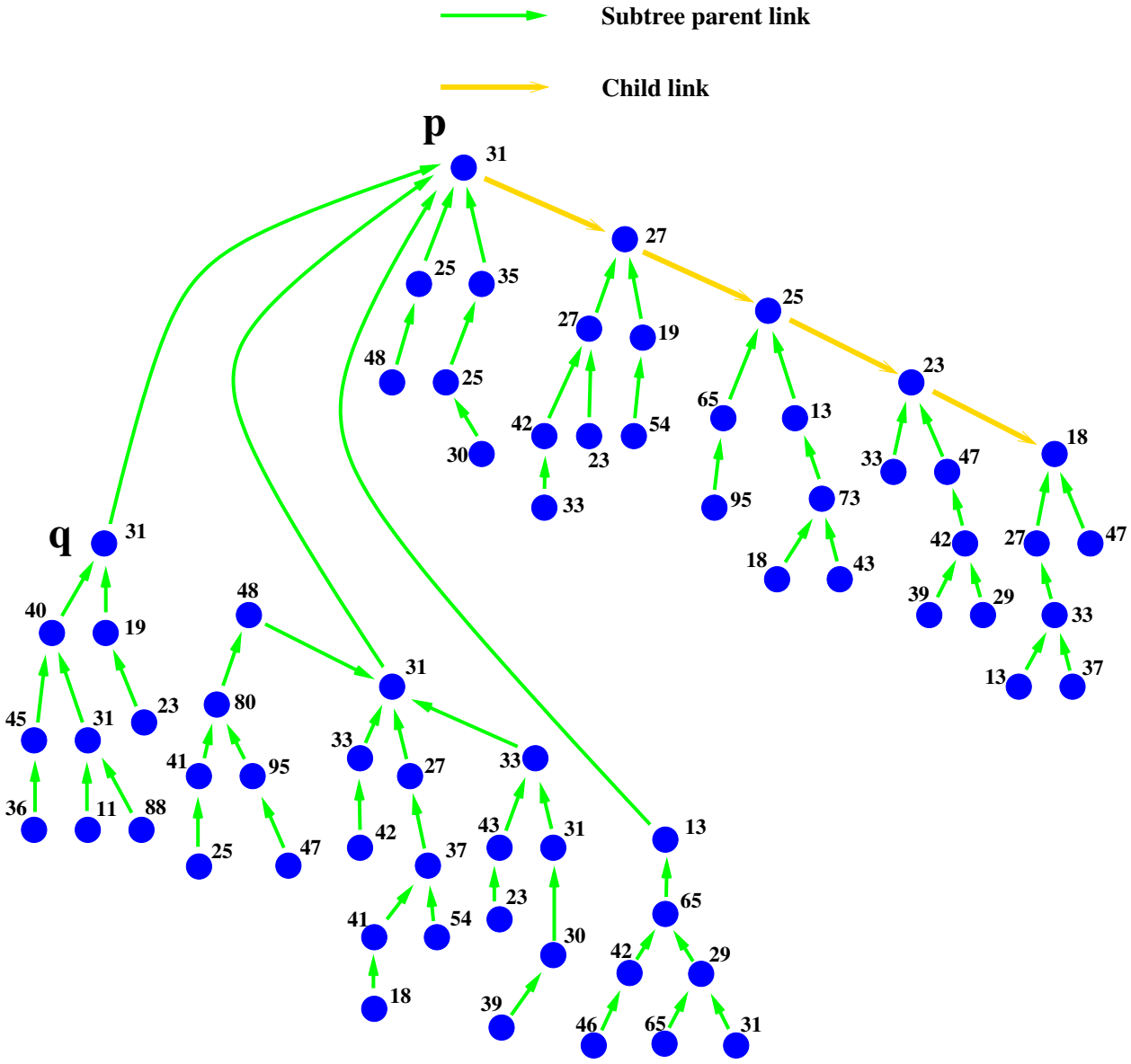
Balanced Trees 3



Balanced Trees 4a



Balanced Trees 4b



Experimental Results

Results from running the BCPL program `bcplprogs/dom/lt.b` which applies the three variants of the algorithm to random graphs.

Random Graph				Cintcode Instruction Counts		
Nodes	Edges	Seed		v.simple	simple	sophisticated
1000	1500	1		311671	285439	328346
1000	2000	1		543460	333994	369395
1000	2500	1		1568707	398925	404413
1000	3000	1		3357486	473709	434642
1000	5000	1		7942067	675828	570509
1000	10000	1		18072476	1131823	905586
10000	50000	1		475843115	7083489	5736513
10000	100000	1		1353711323	11785784	9103018
100000	400000	1		-	60774694	51198153
100000	123289	1	f	-	26591295	33179341