Assignment 3 (25 points), SE 421, 8/31/2020, due: Wednesday, 9/9/2020

Name (Last, First): Morellas, Stamatios

Electronic Copy Requirement: (a) The answers should be typed. (b) The first page should include the top two lines with your last and the first name. (c) Include each question along with your answer to the question. (d) The file should be named HW3-lastname-firstname. (e) Include each question from here followed by your answer.

Preliminary: Index XINU in Atlas before starting this assignment.

Problem 1 (9 points): Answer the following four questions for the functions *dskqopt* and *ethinter*

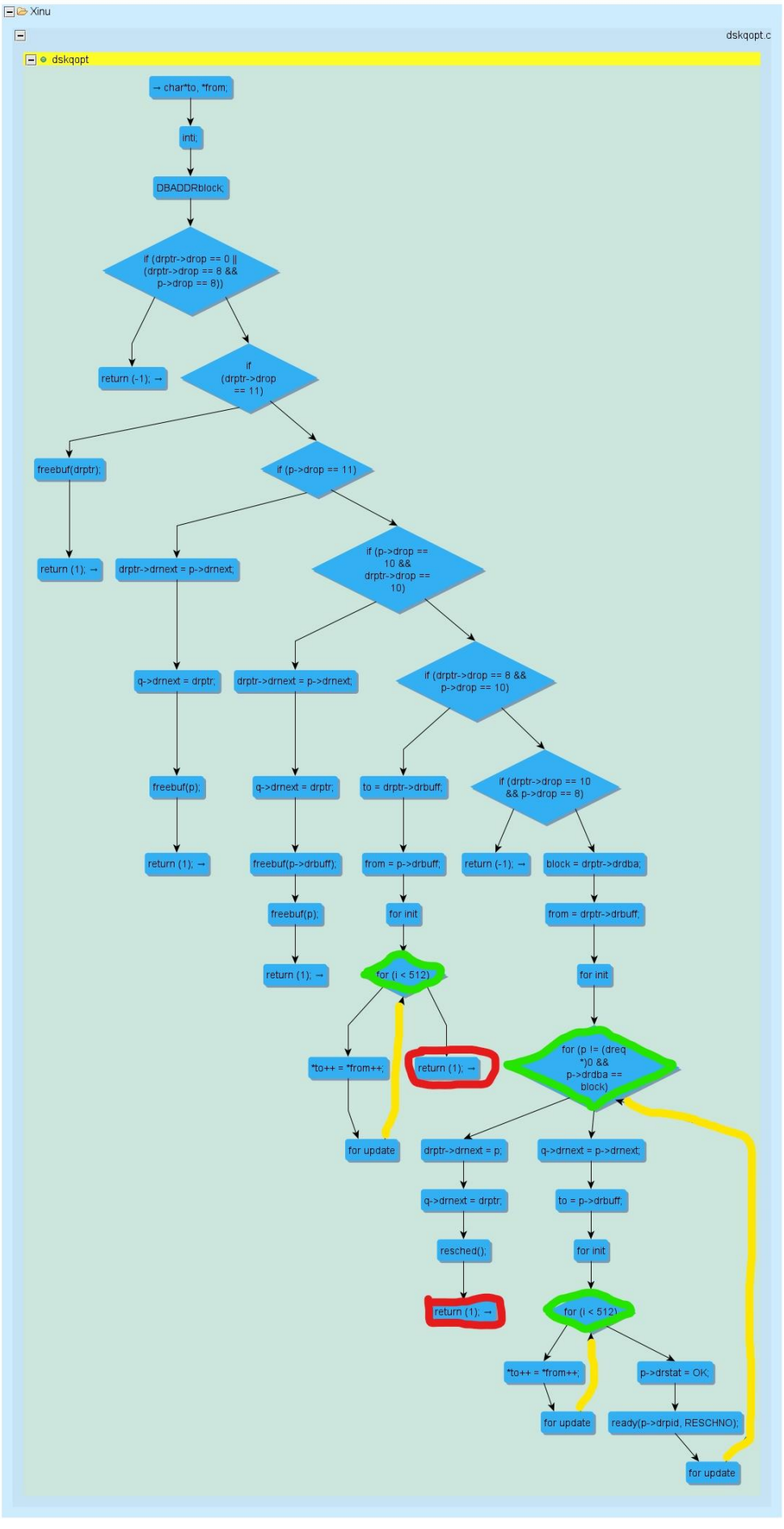
1. Create and save the CFG using Atlas and include them in your answers for the next three questions. (2 points)
2. For each loop in the CFG mark the Entry, Exit, nodes and the Loop-back edges. (3 points)
3. Create the *directed acyclic graph* (DAG) removing the loop back edges (2 points)
4. Give the number of paths for the above DAGs (2 points)

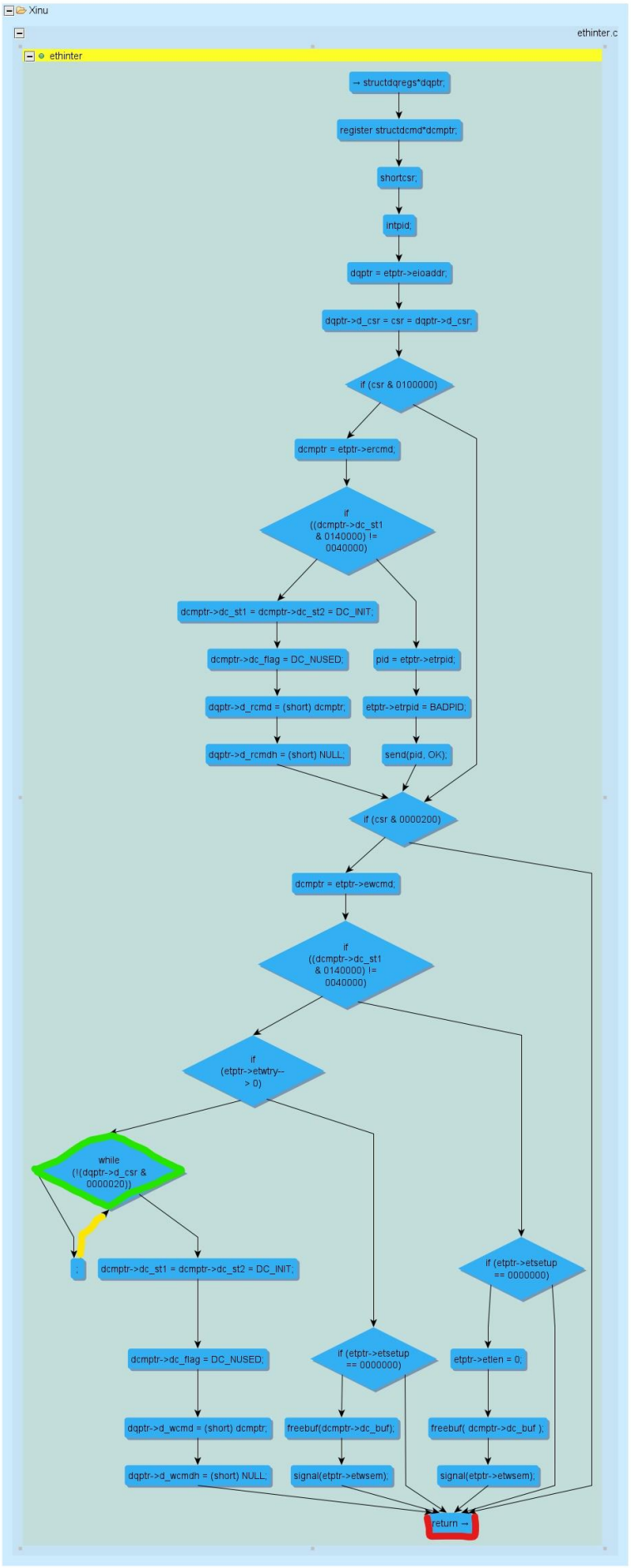
Note: *Refer to lecture notes for similar examples and follow their style for marking graphs.*

Legend/Key:

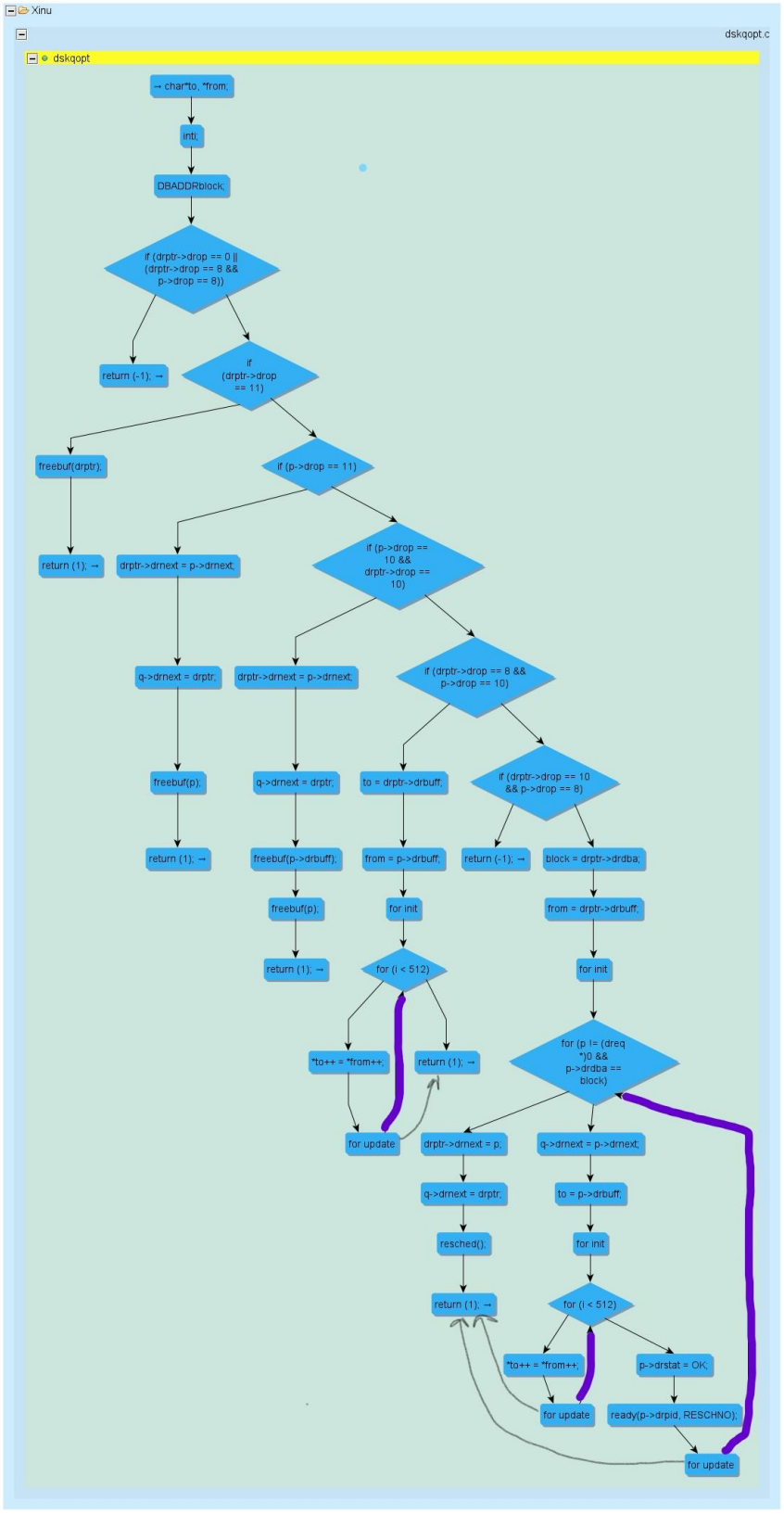
- Green – Loop Entry Node
- Red – Loop Exit Node
- Yellow – Loop Back Edge
- Purple – Remove Edge

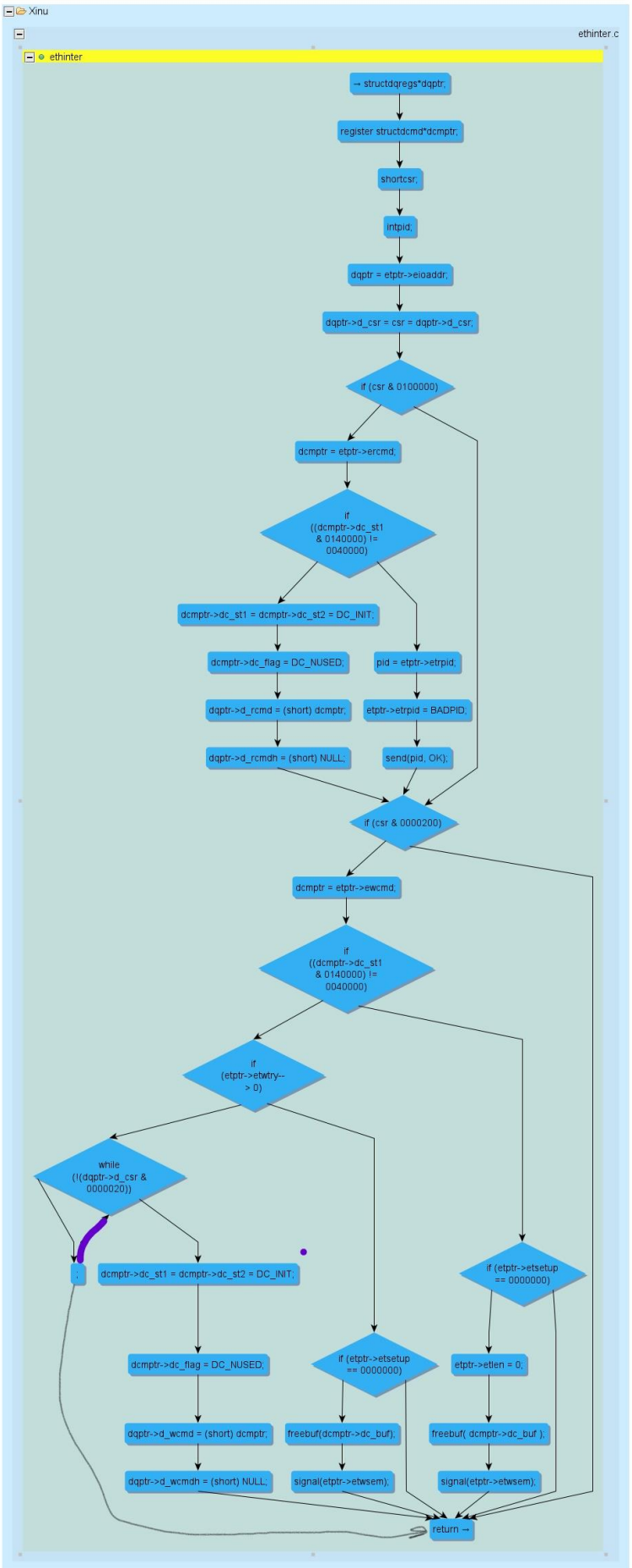
CFGs:


Xinu
dskqopt.c
dskqopt
→ char*to, *from;
int i;
DBADDRblock;
if (drptr->drop == 0 || (drptr->drop == 8 && p->drop == 8))
return (-1); →
if (drptr->drop == 11)
freebuf(drptr);
return (1); →
if (p->drop == 11)
drptr->drnext = p->drnext;
q->drnext = drptr;
freebuf(p);
return (1); →
if (p->drop == 10 && drptr->drop == 10)
drptr->drnext = p->drnext;
q->drnext = drptr;
freebuf(p->drbuff);
freebuf(p);
return (1); →
if (drptr->drop == 8 && p->drop == 10)
to = drptr->drbuff;
from = p->drbuff;
for init
for (i < 512)
*to++ = *from++;
for update
return (1); →
if (drptr->drop == 10 && p->drop == 8)
return (-1); →
block = drptr->drdba;
from = drptr->drbuff;
for init
for (p != (dreq *)0 && p->drdba == block)
drptr->drnext = p;
q->drnext = drptr;
resched();
return (1); →
q->drnext = p->drnext;
to = p->drbuff;
for init
for (i < 512)
*to++ = *from++;
for update
p->drstat = OK;
ready(p->drpid, RESCHNO);
for update


Xinu
ethinter.c
ethinter
→ structdqregs*dqptr;
register structdcmd*dcmptr;
shortcsr;
intpid;
dqptr = etptr->eioaddr;
dqptr->d_csr = csr = dqptr->d_csr;
if (csr & 0100000)
dcmptr = etptr->ercmd;
if ((dcmptr->dc_st1 & 0140000) != 0040000)
dcmptr->dc_st1 = dcmptr->dc_st2 = DC_INIT;
dcmptr->dc_flag = DC_NUSED;
pid = etptr->etrpid;
dqptr->d_rcmd = (short) dcmptr;
etptr->etrpid = BADPID;
dqptr->d_rcmdh = (short) NULL;
send(pid, OK);
if (csr & 0000200)
dcmptr = etptr->ewcmd;
if ((dcmptr->dc_st1 & 0140000) != 0040000)
if (etptr->etwtry-- > 0)
while (!(dqptr->d_csr & 0000020))
;
dcmptr->dc_st1 = dcmptr->dc_st2 = DC_INIT;
if (etptr->etsetup == 0000000)
dcmptr->dc_flag = DC_NUSED;
if (etptr->etsetup == 0000000)
etptr->etlen = 0;
dqptr->d_wcmd = (short) dcmptr;
freebuf(dcmptr->dc_buf);
freebuf(dcmptr->dc_buf);
dqptr->d_wcmdh = (short) NULL;
signal(etptr->etwsem);
signal(etptr->etwsem);
return →

DAGs


Xinu
dskqopt.c
dskqopt
→ char*to, *from;
int i;
DBADDRblock;
if (drptr->drop == 0 || (drptr->drop == 8 && p->drop == 8))
return (-1); →
if (drptr->drop == 11)
freebuf(drptr);
return (1); →
if (p->drop == 11)
drptr->drnext = p->drnext;
if (p->drop == 10 && drptr->drop == 10)
q->drnext = drptr;
freebuf(p);
return (1); →
drptr->drnext = p->drnext;
q->drnext = drptr;
freebuf(p->drbuff);
freebuf(p);
return (1); →
if (drptr->drop == 8 && p->drop == 10)
to = drptr->drbuff;
from = p->drbuff;
for init
for (i < 512)
*to++ = *from++;
return (1); →
for update
if (drptr->drop == 10 && p->drop == 8)
return (-1); →
block = drptr->drdba;
from = drptr->drbuff;
for init
for (p != (dreq *)0 && p->drdba == block)
drptr->drnext = p;
q->drnext = drptr;
resched();
return (1); →
q->drnext = p->drnext;
to = p->drbuff;
for init
for (i < 512)
*to++ = *from++;
for update
p->drstat = OK;
ready(p->drpid, RESCHNO);
for update


Xinu
ethinter.c
ethinter
→ structdqregs*dqptr;
register structdcmd*dcmptr;
shortcsr;
intpid;
dqptr = etptr->eioaddr;
dqptr->d_csr = csr = dqptr->d_csr;
if (csr & 0100000)
dcmptr = etptr->ercmd;
if ((dcmptr->dc_st1 & 0140000) != 0040000)
dcmptr->dc_st1 = dcmptr->dc_st2 = DC_INIT;
dcmptr->dc_flag = DC_NUSED;
pid = etptr->etrpid;
dqptr->d_rcmd = (short) dcmptr;
etptr->etrpid = BADPID;
dqptr->d_rcmdh = (short) NULL;
send(pid, OK);
if (csr & 0000200)
dcmptr = etptr->ewcmd;
if ((dcmptr->dc_st1 & 0140000) != 0040000)
if (etptr->etwtry-- > 0)
while (!(dqptr->d_csr & 0000020))
;
dcmptr->dc_st1 = dcmptr->dc_st2 = DC_INIT;
if (etptr->etsetup == 0000000)
dcmptr->dc_flag = DC_NUSED;
if (etptr->etsetup == 0000000)
etptr->etlen = 0;
dqptr->d_wcmd = (short) dcmptr;
freebuf(dcmptr->dc_buf);
freebuf(dcmptr->dc_buf);
dqptr->d_wcmdh = (short) NULL;
signal(etptr->etwsem);
signal(etptr->etwsem);
return →

Number of paths in DAG (*dskqopt*): 10 paths

Number of paths in DAG (*ethinter*): 21 paths

Problem 2 (7 points): Answer the following questions w.r.t. the dbz function shown below.

```
1  void dbz(int c1, int c2, int c3){
2      int x,d,y,z;
3      x = a1+a2;
4      d = a1;
5      if(c1)
6          x = a1;
7      else d = a2;
8      if(c2){
9          if(c3)
10             y = a1;
11         else d = d-a1;
12     }
13     else
14         d = d+1;
15     z = x/d;
16 }
```

(a) Assume a1 = 10 and a2 = 20. What are the possible values of d at the end of the program? (2 points)

The possible values of d are **10, 0, 11, 20, 21**

(b) Use the information you computed in (a) to find values (c1,c2,c3) that cause the division-by-zero vulnerability. (1 point)

The values of (c1, c2, c3) respectively that cause the division by zero vulnerability are **(1, 1, 0)**

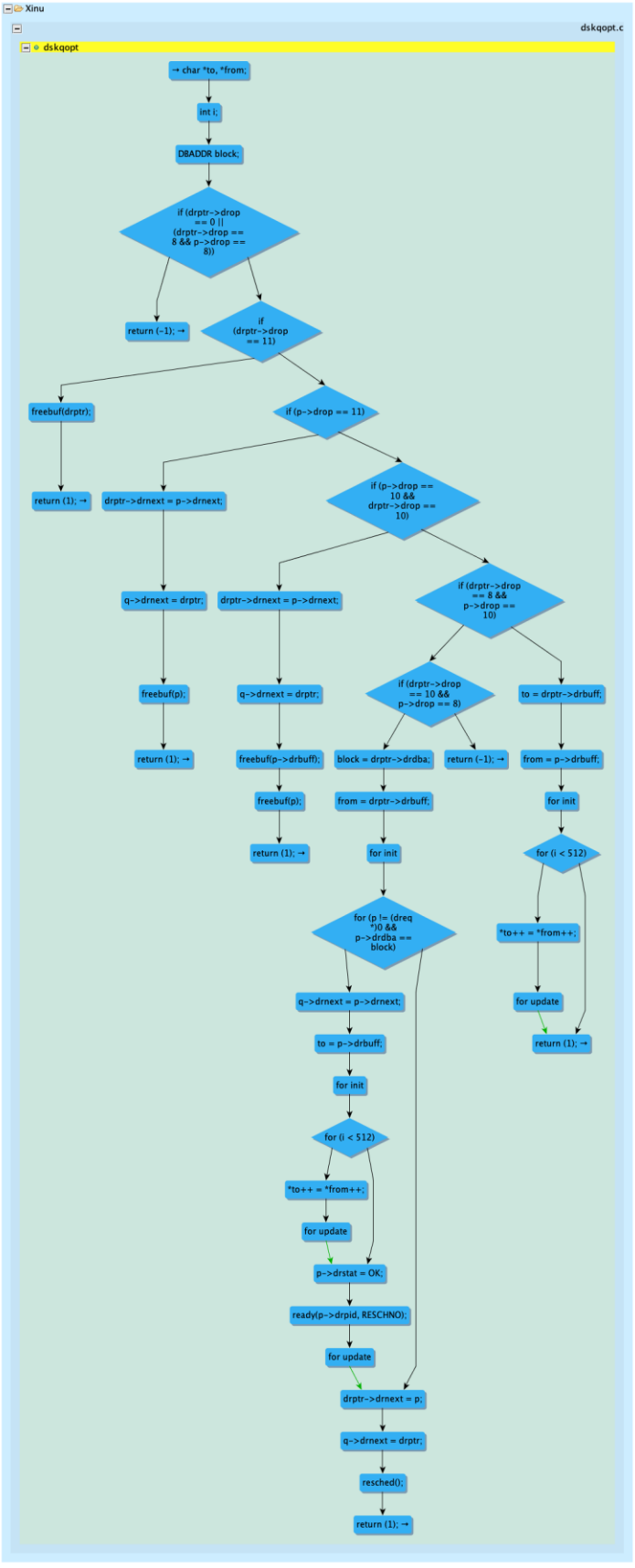
(c) Assume c3 = 1 i.e. if(c3) evaluates to true. Assume a1,a2 to be *non-zero unsigned integers*. Are there values (a1,a2) which can cause the vulnerability? If yes, then what are those? If no, then argue why that can't be the case. (2 points)

No, there should not be any available values for a1 and a2 that would cause a vulnerability. So long as the program can reach the else block for c3, and d = a1 can there be a vulnerability. However, since it is assumed that c3 is true, this else block cannot be executed.

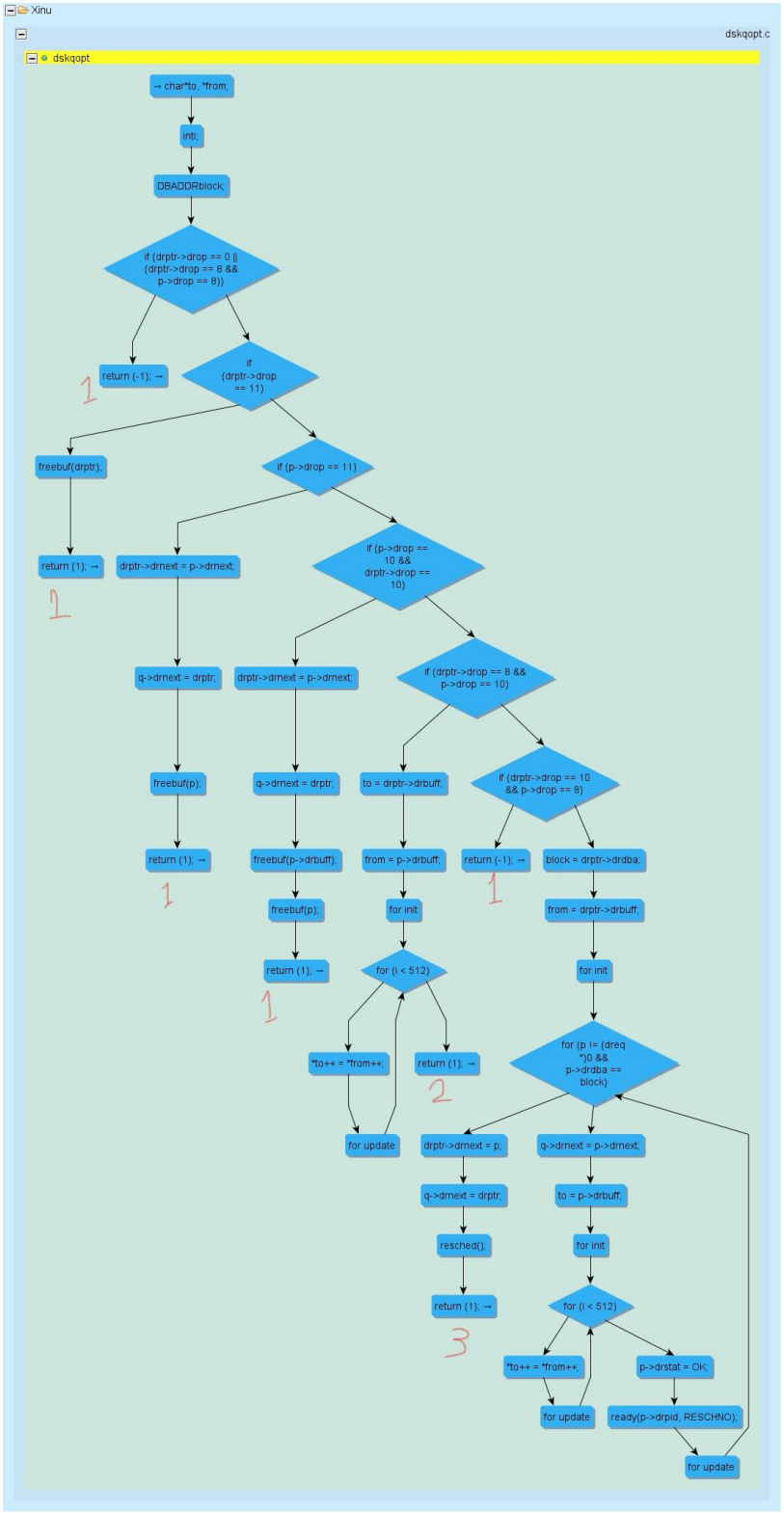
(d) Repeat the exercise (c) with the assumption that a1 and a2 are *non-zero signed integers*. (2 points)

If a1 and a2 are *signed* integers, then **yes**, there will be a vulnerability when a2 < 0 and |a2| < |a1|. When c1 is *false* and the stated conditions are in place, then the value of z will be negative, which will not be read accurately unless z is also a signed integer, but nothing is mentioned about that so we must assume the worst.

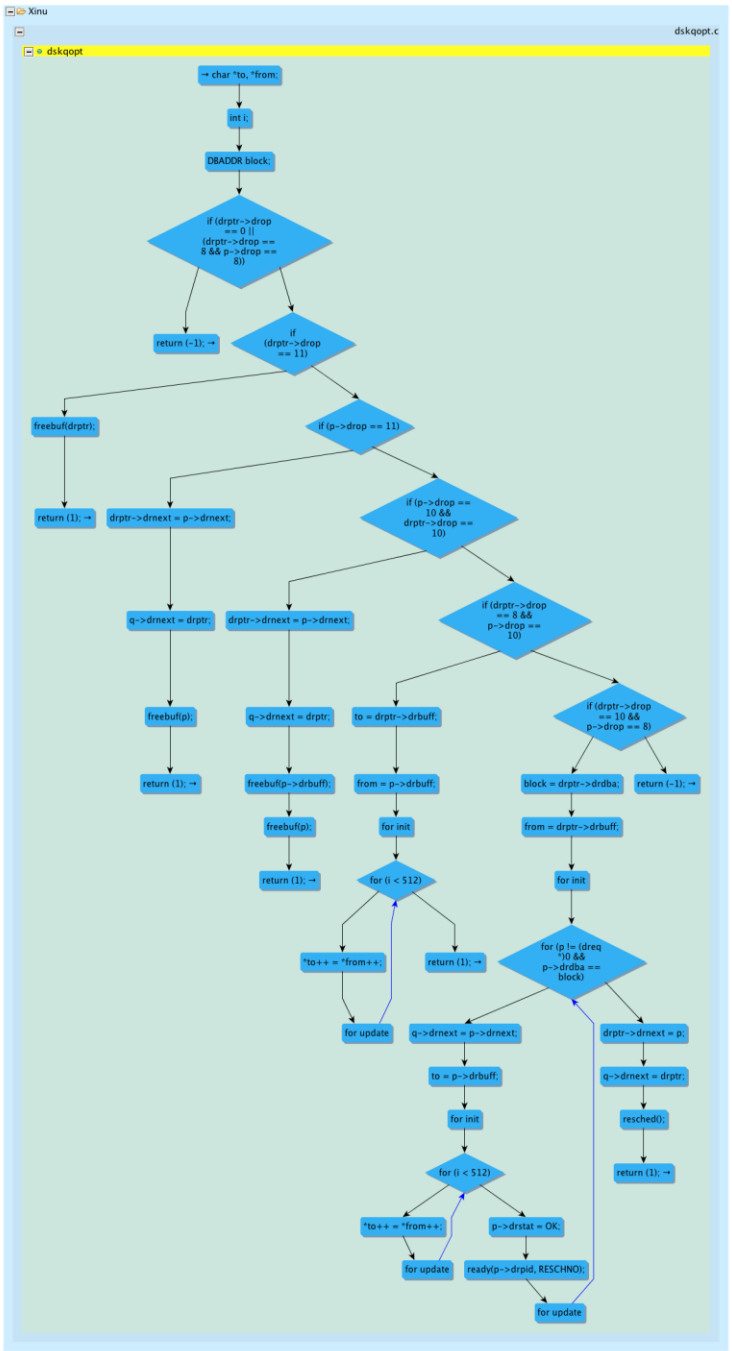
Problem 3 (5 points): What is the total number of paths in the following control flow graph**?** Next to each leaf, write the number of paths reaching that leaf. Note: Each path starts at the root and reaches a leaf.



Answer:


Xinu
dskqopt.c
dskqopt
→ char*to, *from;
int i;
DBADDRblock;
if (drptr->drop == 0 || (drptr->drop == 8 && p->drop == 8))
return (-1); →
1
if (drptr->drop == 11)
freebuf(drptr);
return (1); →
1
if (p->drop == 11)
drptr->drnext = p->drnext;
q->drnext = drptr;
freebuf(p);
return (1); →
1
if (p->drop == 10 && drptr->drop == 10)
drptr->drnext = p->drnext;
q->drnext = drptr;
freebuf(p->drbuff);
freebuf(p);
return (1); →
1
if (drptr->drop == 8 && p->drop == 10)
to = drptr->drbuff;
from = p->drbuff;
for init
for (i < 512)
*to++ = *from++;
for update
return (1); →
2
if (drptr->drop == 10 && p->drop == 8)
return (-1); →
1
block = drptr->drdba;
from = drptr->drbuff;
for init
for (p != (dreq *)0 && p->drdba == block)
drptr->drnext = p;
q->drnext = drptr;
resched();
return (1); →
3
q->drnext = p->drnext;
to = p->drbuff;
for init
for (i < 512)
*to++ = *from++;
for update
p->drstat = OK;
ready(p->drpid, RESCHNO);
for update

Problem 4 (4 points) Answer the following questions for the given CFG. How many loops are there? Is there a nested loop? Suppose you remove all the loop back edges, then how many paths in this control flow graph? Note: As a sample, one loop back edge is marked by X.



In problem 4, there are **3 loops** in total, **1 of which is nested**. If all loop-back edges are removed, then there will be a total of **7 paths** in the control flow graph.

Extra Credit Problem (4 points): Graph-2 is obtained from Graph-1 by removing some edges and adding some new edges. The nodes in both the graphs are the same (no nodes added or removed) but their positions may have changed. Describe the relationship between the two graphs by answering two questions:

1. What edges are removed in Graph-2 compared to Graph-1? In Graph-1, add label X to each edge that is *removed*.
2. What edges are added in Graph-2 compared to Graph-1? In Graph-2, add label A to each edge that is *added*.