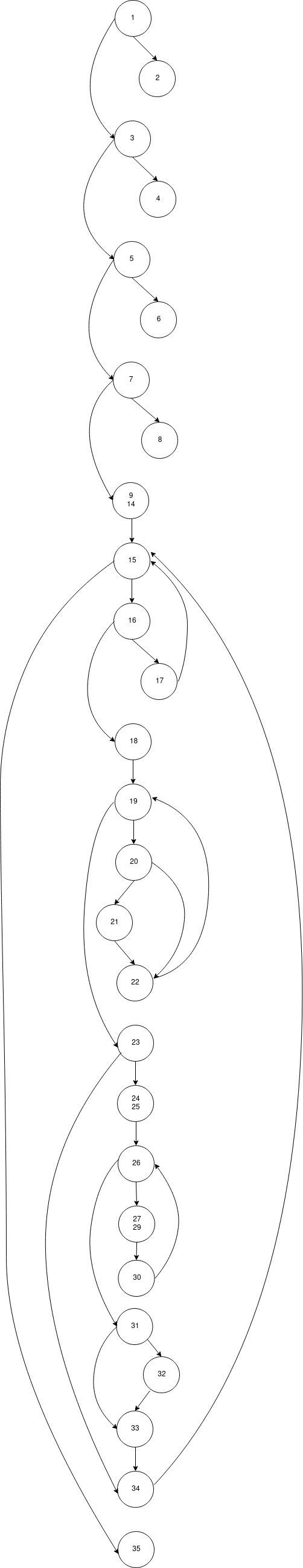
2 h) Avem numerotate liniile astfel, pentru a reprezenta nodurile grafului:

1. if ( k < 0 )
2. throw new IllegalArgumentException("K is negative.");
3. if ( s < 0 )
4. throw new IllegalArgumentException("S is negative.");
5. if ( a < 0 || b < 0)
6. throw new IllegalArgumentException("Range is negative.");
7. if ( a > b )
8. throw new IllegalArgumentException("Range is reversed.");
9. List<Integer> primes = new ArrayList<>();
10. int copy;
11. int digit;
12. int sum;
13. boolean found;
14. int number, divisor;
15. for(number = a; primes.size() < k && number <= b; number++){
16. if(number == 0 || number == 1)
17. continue;
18. found = false;
19. for(divisor = 2; !found && divisor <= sqrt(number); divisor++){
20. if(number % divisor == 0)
21. found = true;
22. }
23. if(!found){
24. copy = number;
25. sum = 0;
26. while(copy != 0){
27. digit = copy % 10;
28. copy = copy / 10;
29. sum += digit;
30. }
31. if(sum == s)
32. primes.add(number);
33. }
34. }
35. return primes;

}

Pe baza acestei notari, rezulta graful urmator:



Expresia regulta obtinuta:

9.15.(16.((null + 17).15)\*.18.19.(20.((21+null).22.19)\*.23.((34.15)\* + 24.26.(27.30.26)\*.31.(32.33+33).34.15)\*.35

Pentru n=0 si n=1 avem:

1.1.(1.((1 + 1).1 + 1).1.1.(1.((1+ 1).1.1 + 1).1.((1.1 + 1) + 1.1.(1.1.1 + 1).1.(1.1+1).1.1 + 1).1

= (2+1) \* (2+1) \* (1+1) \* (1+1) \* (1+1) = 3 \* 3 \* 2 \* 2 \* 2 = 72 cai

2 b)

Am numerotat deciziile astfel:

public static List<Integer> findPrimes(int k, int a, int b, int s) throws IllegalArgumentException {

1. if ( k < 0 ) throw new IllegalArgumentException("K is negative.");
2. if ( s < 0 ) throw new IllegalArgumentException("S is negative.");
3. if ( a < 0 || b < 0) throw new IllegalArgumentException("Range is negative.");
4. if ( a > b ) throw new IllegalArgumentException("Range is reversed.");

List<Integer> primes = new ArrayList<>();

int copy;

int digit;

int sum;

boolean found;

int number, divisor;

1. for(number = a; primes.size() < k && number <= b; number++)

{

1. if(number == 0 || number == 1)

continue;

found = false;

1. for(divisor = 2; !found && divisor <= sqrt(number); divisor++)

{

1. if(number % divisor == 0)

found = true;

}

1. if(!found)

{

copy = number;

sum = 0;

1. while(copy != 0)

{

digit = copy % 10;

copy = copy / 10;

sum += digit;

}

1. if(sum == s)

primes.add(number);

}

}

return primes;

}

Pentru a acoperi toate cazurile, atat cu adevarat cat si cu fals, avem urmatoarele teste:

1 → A - findPrimes(-1, 0, 0, 0)

→ F - findPrimes(0, 0, 0, -1)

2 → A - findPrimes(0, 0, 0, -1)

→ F - findPrimes(0, -1, 0, 0)

3 → A - findPrimes(0, -1, 0, 0)

→ F - findPrimes(0, 2, 1, 0)

4 → A - findPrimes(0, 2, 1, 0)

→ F - findPrimes(1, 1, 3, 2)

5 → A - findPrimes(1, 1, 3, 2)

→ F - findPrimes(1, 1, 3, 2)

6 → A - findPrimes(1, 1, 3, 2)

→ F - findPrimes(1, 1, 3, 2)

7 → A - findPrimes(5, 1, 5, 2)

→ F - findPrimes(1, 1, 3, 2)

8 → A - findPrimes(5, 1, 5, 2)

→ F - findPrimes(5, 1, 5, 2)

9 → A - findPrimes(1, 1, 3, 2)

→ F - findPrimes(5, 1, 5, 2)

10 → A - findPrimes(1, 1, 3, 2)

→ F - findPrimes(1, 1, 3, 2)

11 → A - findPrimes(1, 1, 3, 2)

→ F - findPrimes(5, 1, 5, 2)