Curtin University

LECTURE 4: GRAPHS (PART 2)

Software Engineering Testing (CMPE3008/CMPE4001/CMPE5000)

Created by Arlen Brower

OUTCOMES

You should be able to do the following:

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
    - Node & Edge Coverage

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
  - Node & Edge Coverage
  - Prime Path Coverage

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
  - Node & Edge Coverage
  - Prime Path Coverage
  - All-DU-Paths, All-Uses, All-defs coverage

OUTCOMES

You should be able to do the following:

- Create control flow graphs from source code
- Identify test requirements for:
    - Node & Edge Coverage
    - Prime Path Coverage
    - All-DU-Paths, All-Uses, All-defs coverage
- Identify test paths for the above

BUT FIRST: LET'S REVIEW

Prime paths!

BUT FIRST: LET'S REVIEW

Prime paths!

- 'Simple paths'

BUT FIRST: LET'S REVIEW

Prime paths!

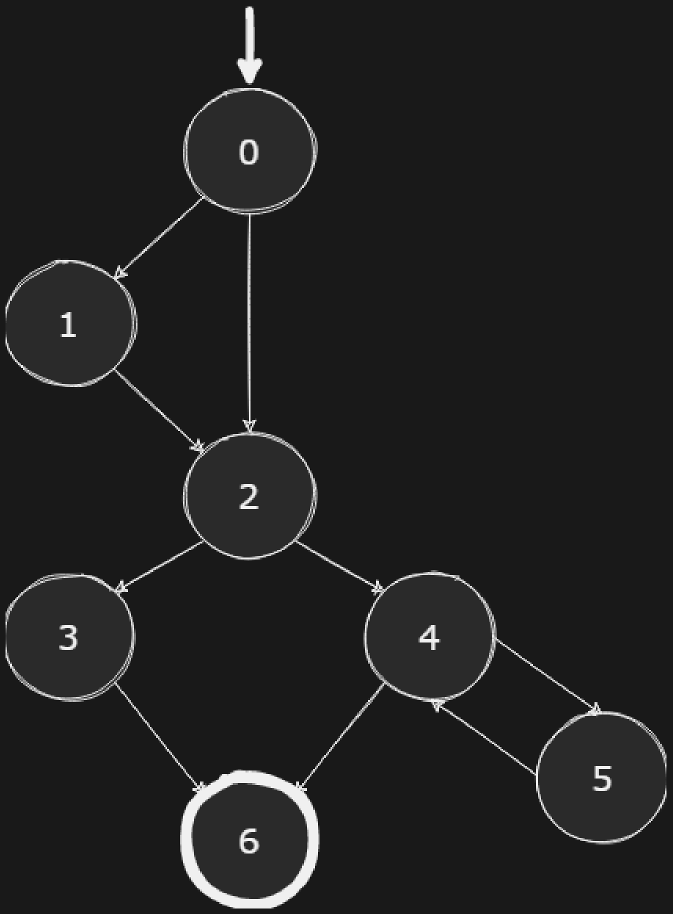
- 'Simple paths'
- No duplicates... except maybe at the start/end

BUT FIRST: LET'S REVIEW

Prime paths!

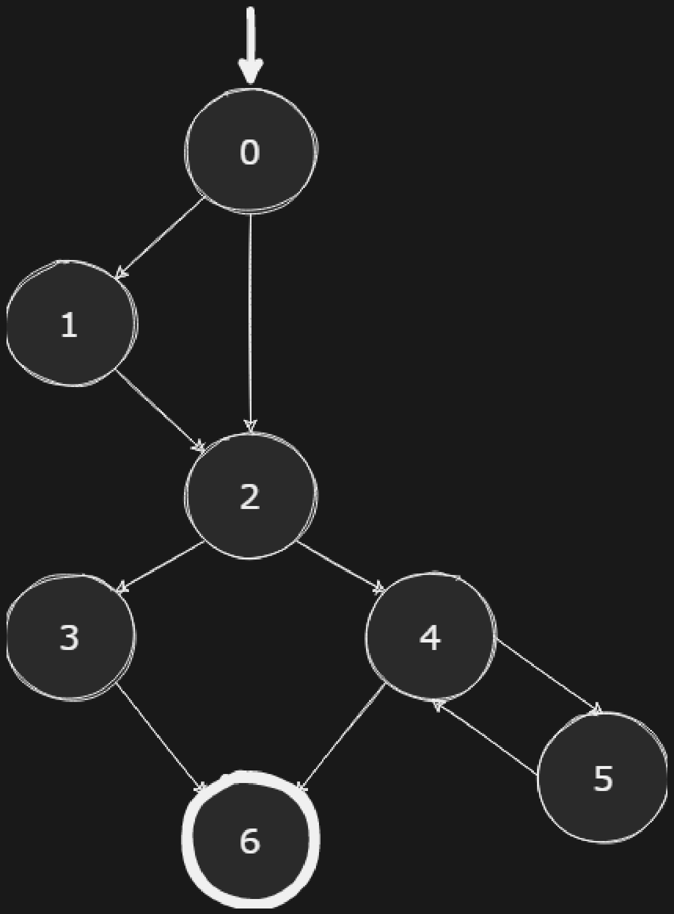
- 'Simple paths'
- No duplicates... except maybe at the start/end
- Used to formally define a means to tackle loops.

BUT FIRST: LET'S REVIEW



What are the prime paths?

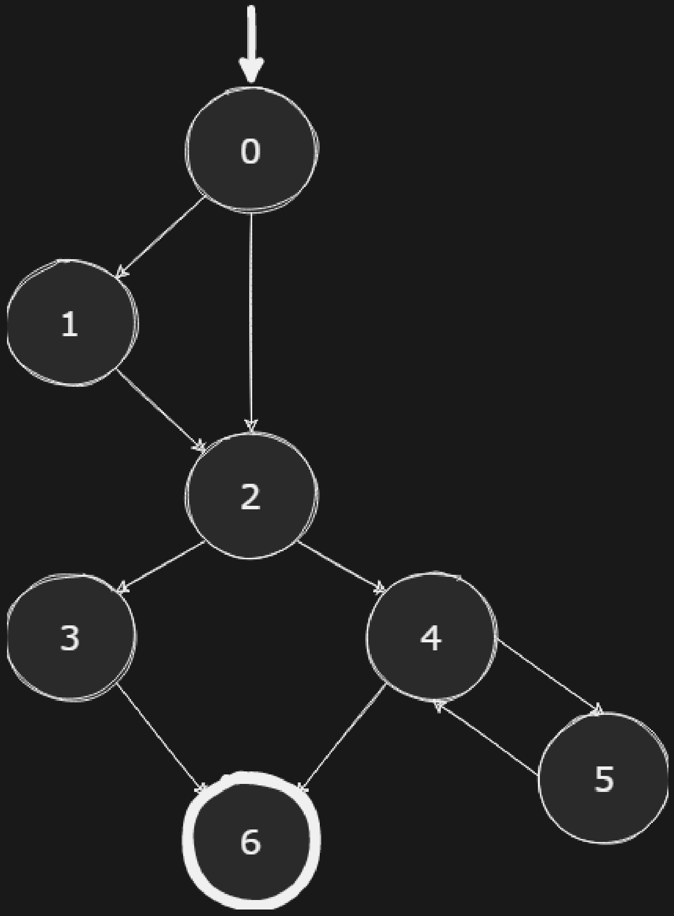
BUT FIRST: LET'S REVIEW



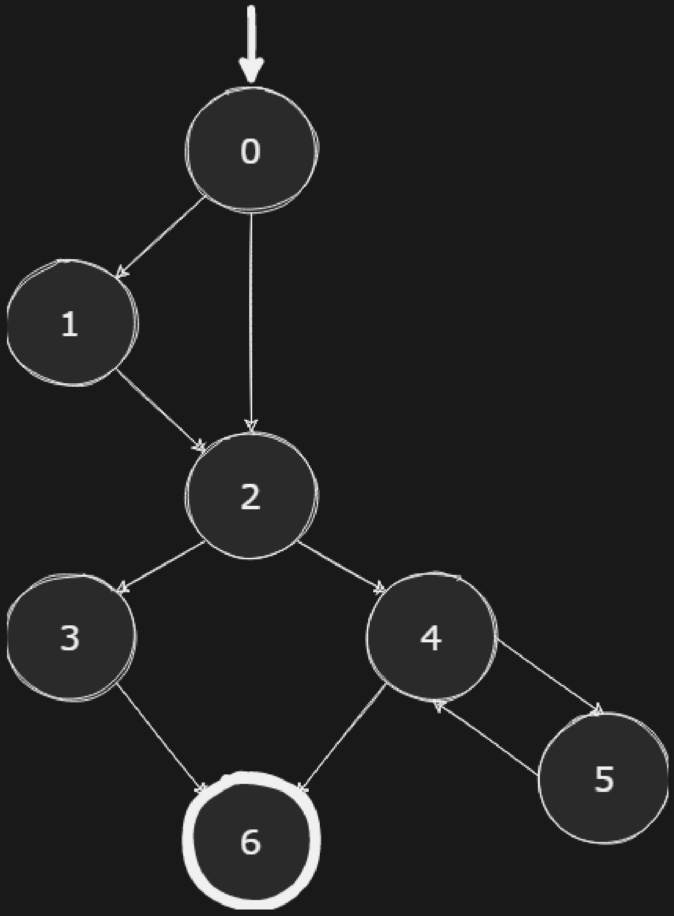
What are the prime paths?

There are nine...

BUT FIRST: LET'S REVIEW

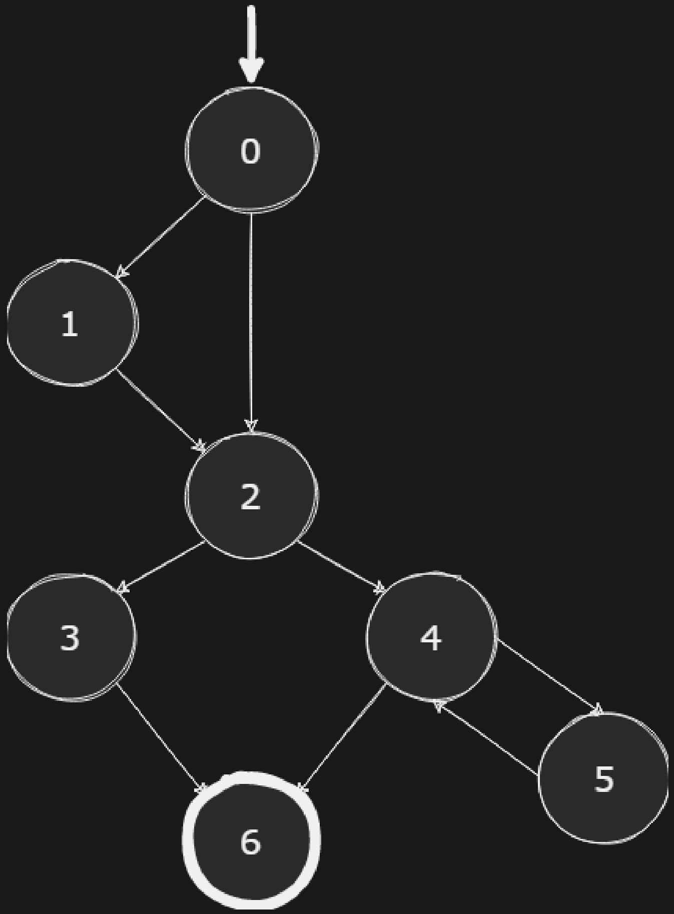


BUT FIRST: LET'S REVIEW



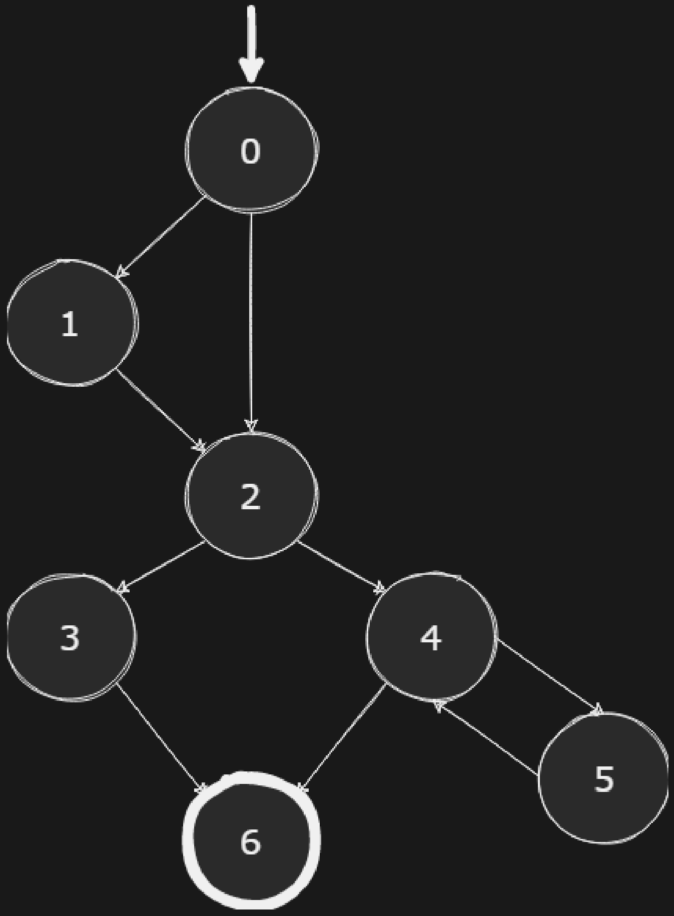
[0 1 2 3 6]

BUT FIRST: LET'S REVIEW



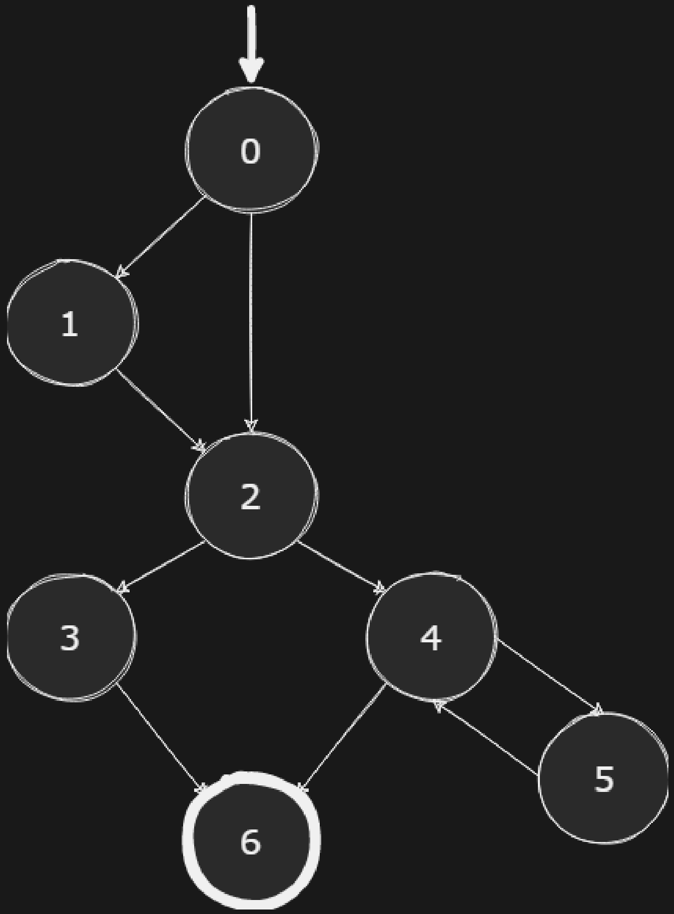
[0 1 2 3 6] [0 1 2 3 6]

BUT FIRST: LET'S REVIEW



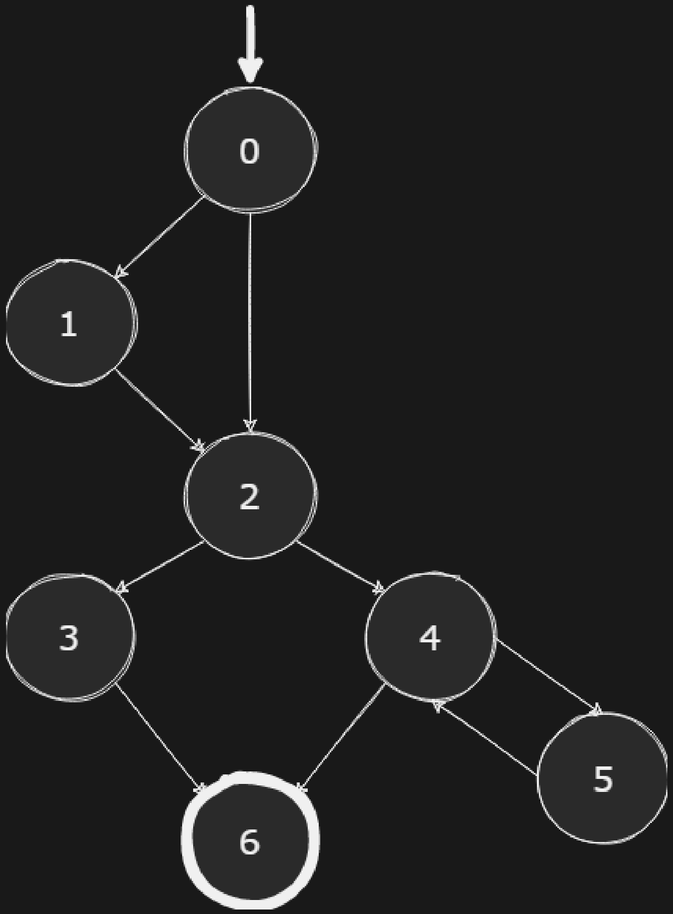
```
[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5]
```

BUT FIRST: LET'S REVIEW



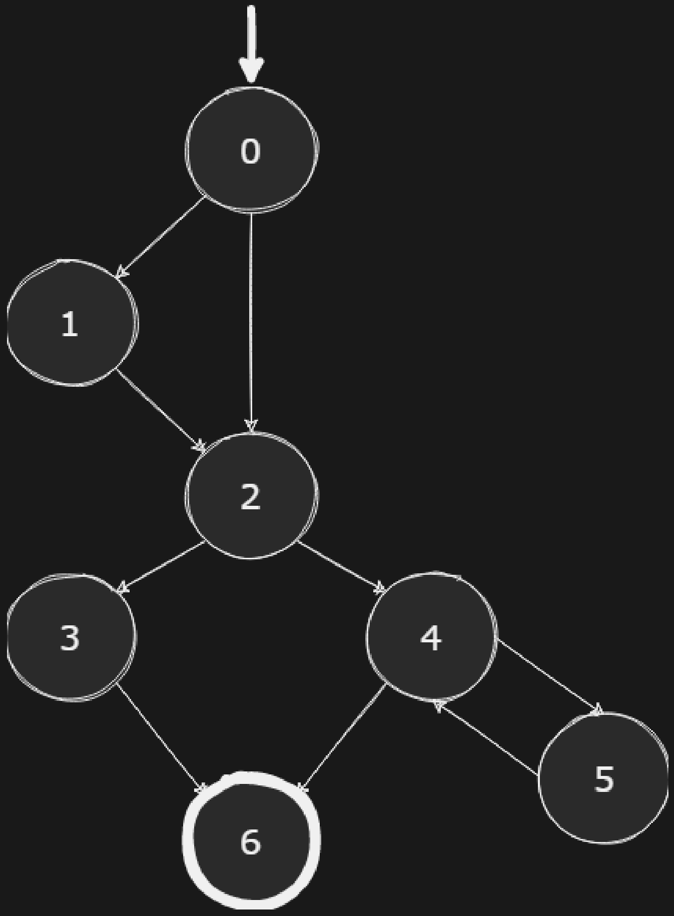
[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5]

BUT FIRST: LET'S REVIEW



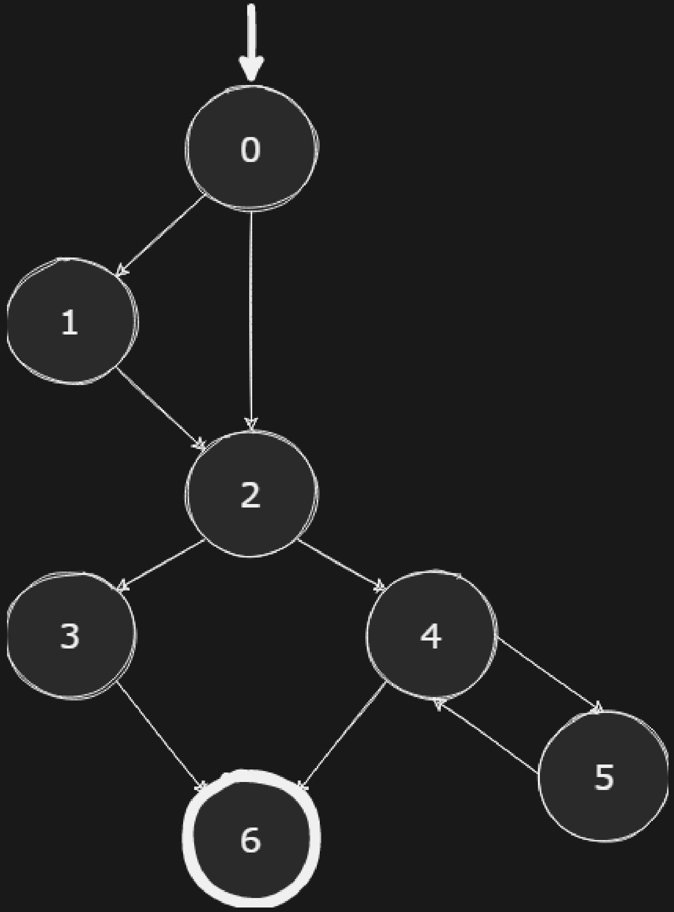
[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5] [0 2 3 6]

BUT FIRST: LET'S REVIEW



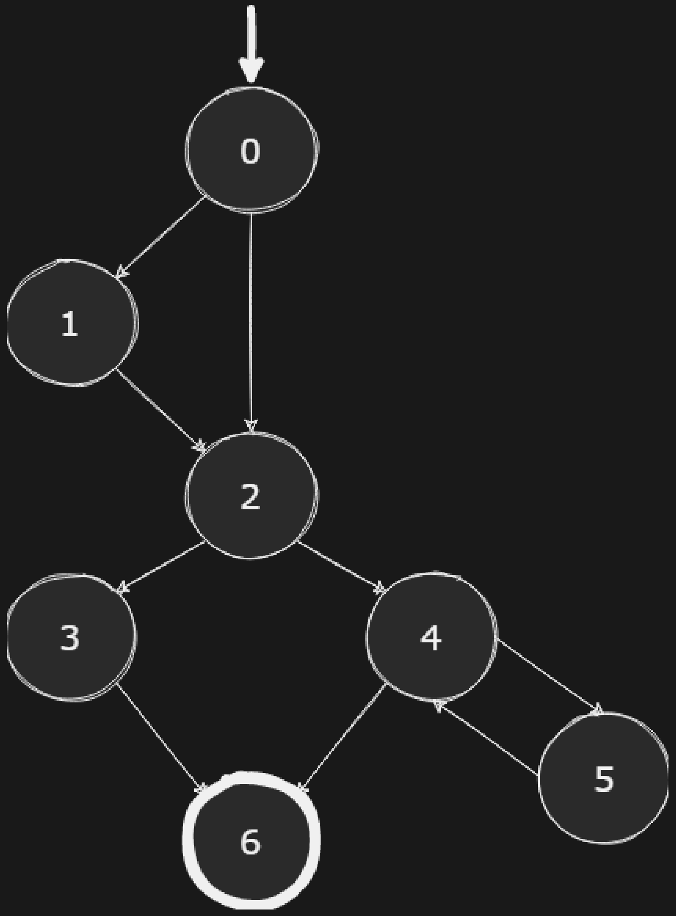
[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5] [0 2 3 6] [0 2 4 6]

BUT FIRST: LET'S REVIEW



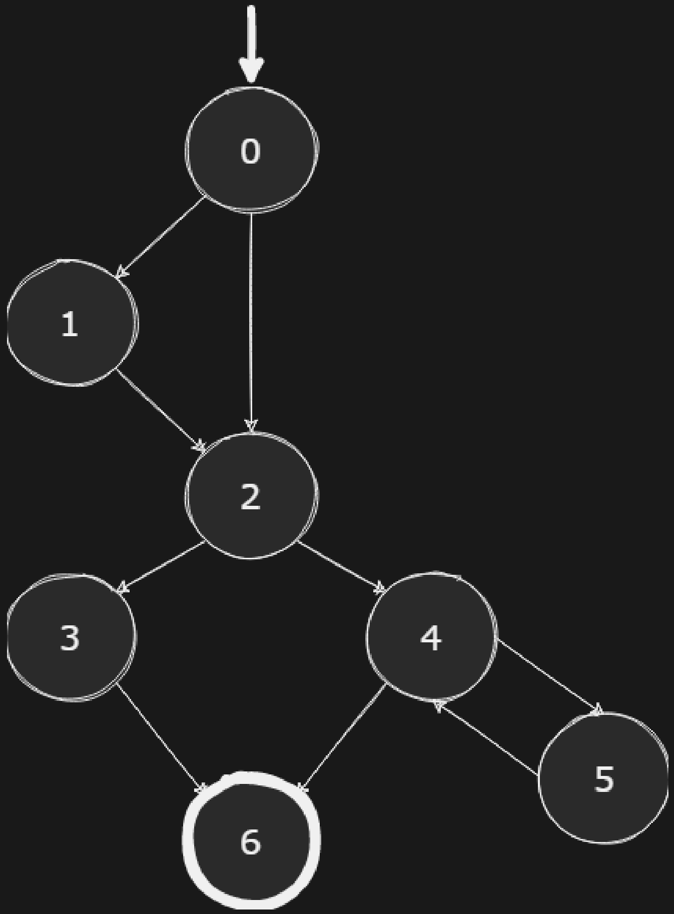
[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5] [0 2 3 6] [0 2 4 6] [0 2 4 5]

BUT FIRST: LET'S REVIEW



```
[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5] [0 2 3 6] [0 2 4 6] [0 2 4 5]
                    [4 5 4]
```

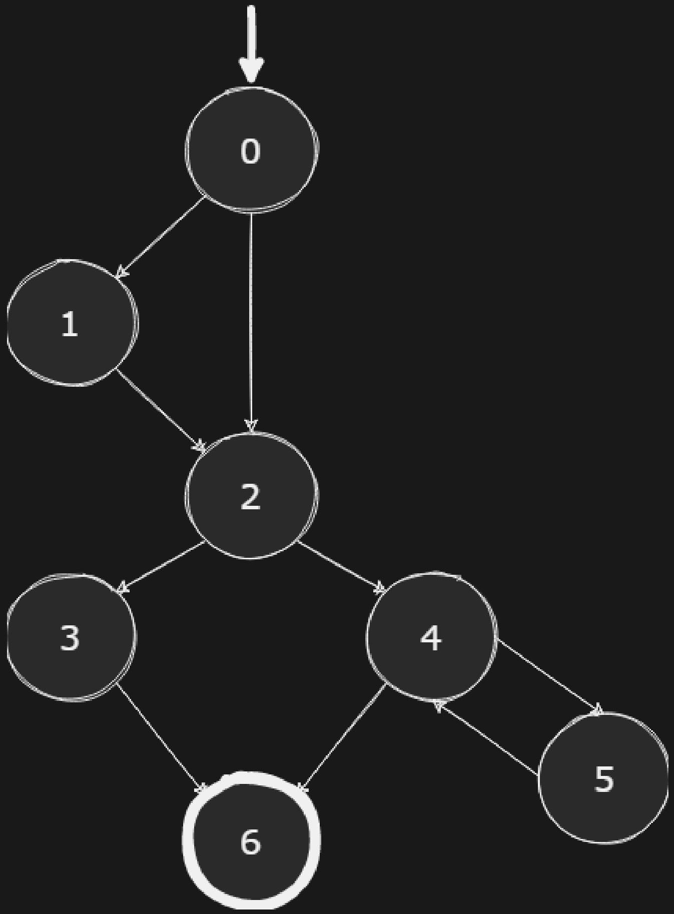
BUT FIRST: LET'S REVIEW



[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5] [0 2 3 6] [0 2 4 6] [0 2 4 5]

[4 5 4] [5 4 5]

BUT FIRST: LET'S REVIEW



[0 1 2 3 6] [0 1 2 3 6] [0 1 2 3 5] [0 2 3 6] [0 2 4 6] [0 2 4 5]
[4 5 4] [5 4 5] [5 4 6]

DATA FLOW COVERAGE CRITERIA

DATA FLOW COVERAGE CRITERIA

DATA FLOW COVERAGE CRITERIA

- Definition (def):A location where a variable is given a value (i.e. stored in memory)

DATA FLOW COVERAGE CRITERIA

- Definition (def):A location where a variable is given a value (i.e. stored in memory)
- Use: A location where a variable is accessed

DATA FLOW COVERAGE CRITERIA

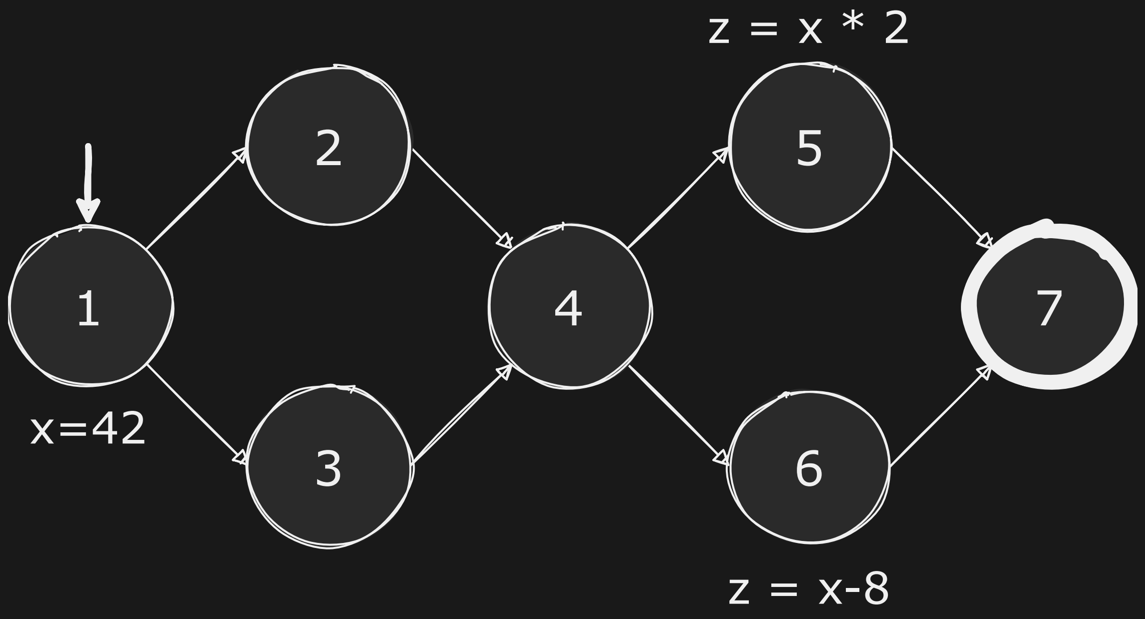
- Definition (def):A location where a variable is given a value (i.e. stored in memory)
- Use: A location where a variable is accessed
- def(n) or def(e): The set of variables that are defined by node n or edge e

DATA FLOW COVERAGE CRITERIA

- Definition (def):A location where a variable is given a value (i.e. stored in memory)
- Use: A location where a variable is accessed
- def(n) or def(e): The set of variables that are defined by node n or edge e
- use(n) or use(e): The set of variables that are used by node n or edge e

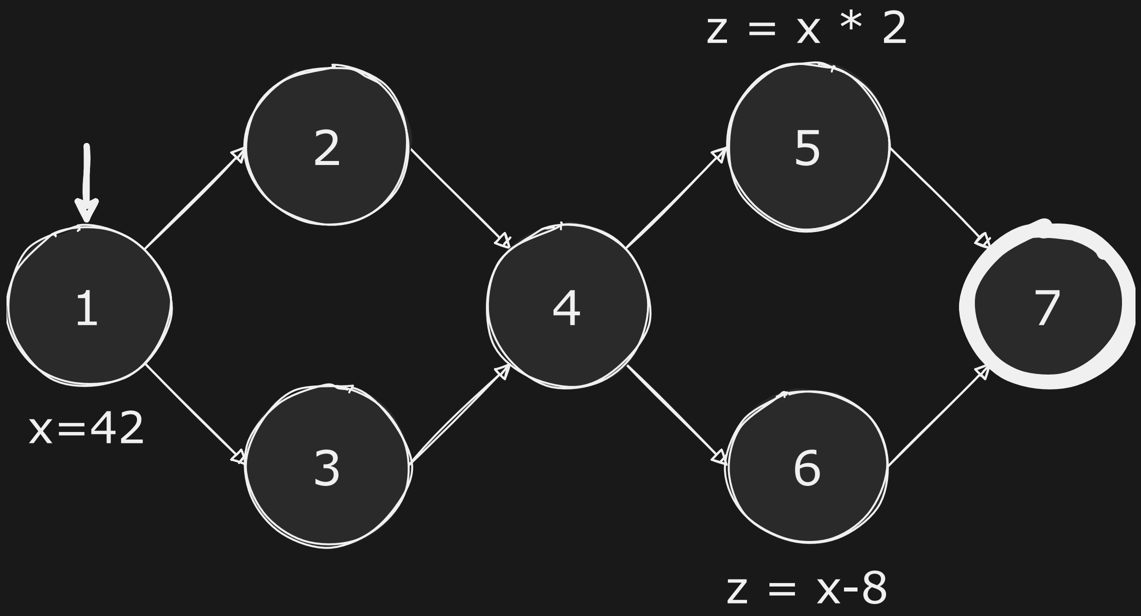
DATA FLOW COVERAGE CRITERIA

For example...



DATA FLOW COVERAGE CRITERIA

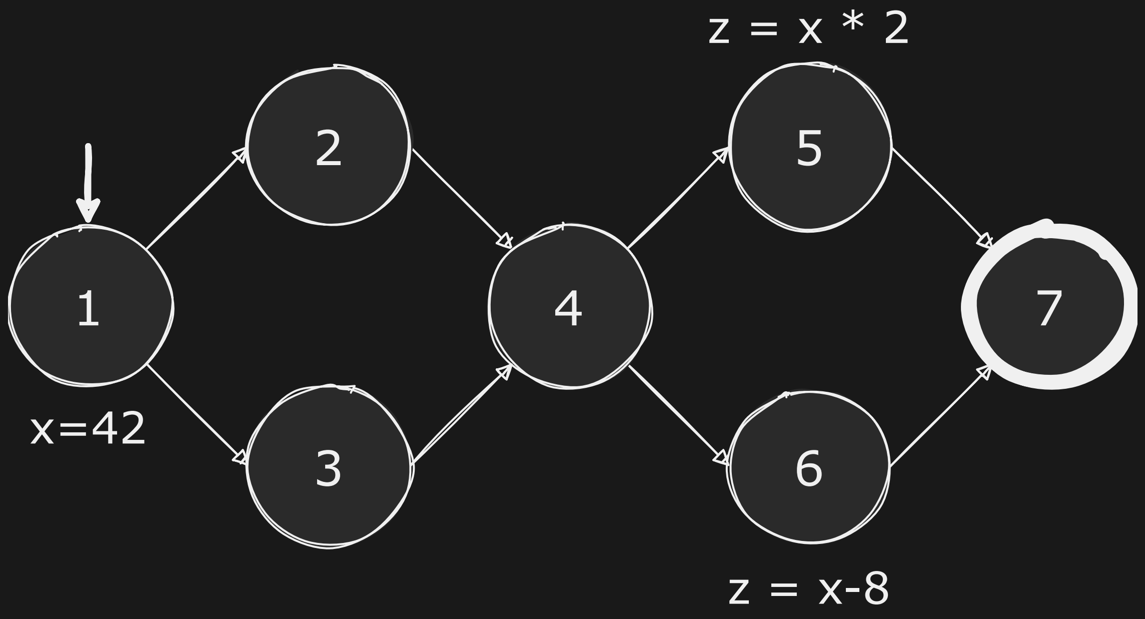
For example...



- Defs:
  - def(1) = {x}
  - def(5) = {z}
  - def(6) = {z}

DATA FLOW COVERAGE CRITERIA

For example...



- Defs:
  - def(1) = {x}
  - def(5) = {z}
  - def(6) = {z}
- Uses:
  - use(5) = {x}
  - use(6) = {x}

DATA FLOW COVERAGE CRITERIA

Prepare for more terminology!

DATA FLOW COVERAGE CRITERIA

DATA FLOW COVERAGE CRITERIA

- DU pair: A pair of locations; one def and one use.

DATA FLOW COVERAGE CRITERIA

- DU pair: A pair of locations; one def and one use.
- Def-clear: A path from one location to another is *def-clear* if there are no other defs between them.

DATA FLOW COVERAGE CRITERIA

- DU pair: A pair of locations; one def and one use.
- Def-clear: A path from one location to another is *def-clear* if there are no other defs between them.
- Reach: If there is a def-clear path between a def and a use, that def *reaches* the use.

DATA FLOW COVERAGE CRITERIA

- DU pair: A pair of locations; one def and one use.
- Def-clear: A path from one location to another is *def-clear* if there are no other defs between them.
- Reach: If there is a def-clear path between a def and a use, that def *reaches* the use.
- du-path: A *simple* subpath that is def-clear with respect to a DU pair

DATA FLOW COVERAGE CRITERIA

DATA FLOW COVERAGE CRITERIA

- $$du(n_i, n_j, v)$$

  The set of du-paths from one location to another.

DATA FLOW COVERAGE CRITERIA

- $$du(n_i, n_j, v)$$

  The set of du-paths from one location to another.

- $$du(n_i, v)$$

  The set of du-paths that start at one location

DATA FLOW COVERAGE CRITERIA

DATA FLOW COVERAGE CRITERIA

- A test path *p* du-tours subpath *d* with respect to a variable if *p* tours *d* and that subpath is def-clear

DATA FLOW COVERAGE CRITERIA

- A test path *p* du-tours subpath *d* with respect to a variable if *p* tours *d* and that subpath is def-clear
- Importantly, most of these are with respect to a particular variable.

DATA FLOW COVERAGE CRITERIA

- A test path *p* du-tours subpath *d* with respect to a variable if *p* tours *d* and that subpath is def-clear
- Importantly, most of these are with respect to a particular variable.
- e.g. We have defs, uses, du-pairs, du-paths for X.

DATA FLOW COVERAGE CRITERIA

- A test path *p* du-tours subpath *d* with respect to a variable if *p* tours *d* and that subpath is def-clear
- Importantly, most of these are with respect to a particular variable.
- e.g. We have defs, uses, du-pairs, du-paths for X.
- e.g. We have defs, uses, du-pairs, du-paths for Y.

DATA FLOW COVERAGE CRITERIA

- A test path *p* du-tours subpath *d* with respect to a variable if *p* tours *d* and that subpath is def-clear
- Importantly, most of these are with respect to a particular variable.
- e.g. We have defs, uses, du-pairs, du-paths for X.
- e.g. We have defs, uses, du-pairs, du-paths for Y.
- e.g. We have defs, uses, du-pairs, du-paths for etc, etc.

DATA FLOW COVERAGE CRITERIA

We have three criteria following from this

DATA FLOW COVERAGE CRITERIA

We have three criteria following from this

- All-defs coverage

DATA FLOW COVERAGE CRITERIA

We have three criteria following from this

- All-defs coverage
- All-uses coverage

DATA FLOW COVERAGE CRITERIA

We have three criteria following from this

- All-defs coverage
- All-uses coverage
- All-du-paths coverage

- All-defs coverage
- For each set of du-paths *S* = du(n,v), TR contains at least one path *d* in *S*

- All-defs coverage
- For each set of du-paths *S* = du(n,v), TR contains at least one path *d* in *S*
- … or, make sure your test requirements include all defs.

- All uses coverage
- For each set of du-paths to uses *S* = du (ni, nj, v), TR contains at least one path *d* in *S*

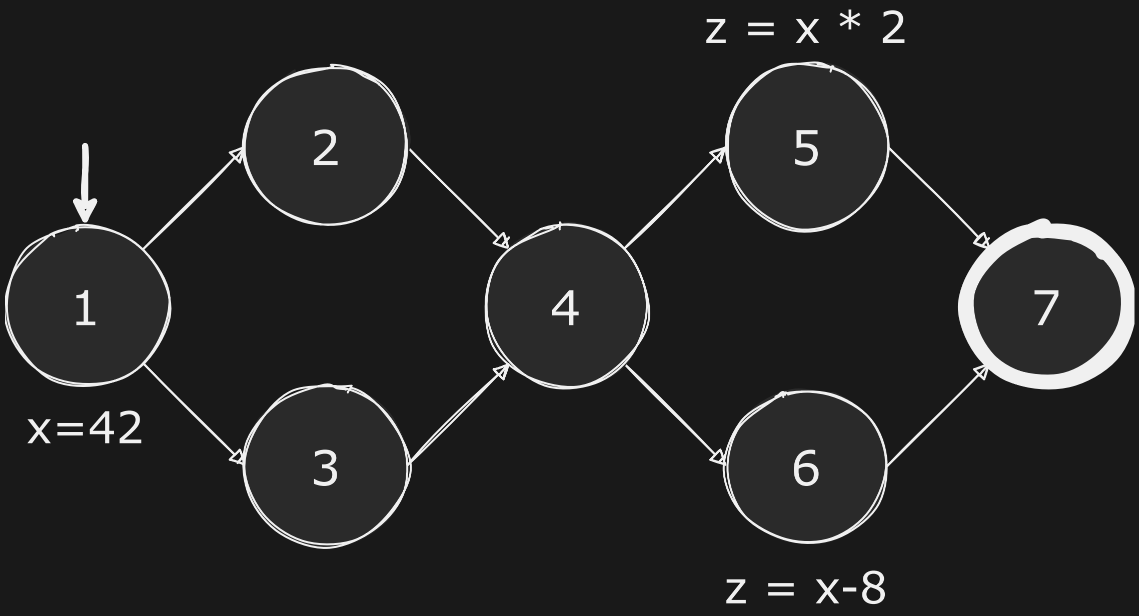
- All uses coverage
- For each set of du-paths to uses *S* = du (ni, nj, v), TR contains at least one path *d* in *S*
- … or, for each def, make sure you have a path to all uses (for that variable)

- All du-paths coverage
- For each set *S* = du (ni, nj, v), TR contains every path *d* in *S*.

- All du-paths coverage
- For each set *S* = du (ni, nj, v), TR contains every path *d* in *S*.
- … or, make sure for each def, you explore every du-path to every use.

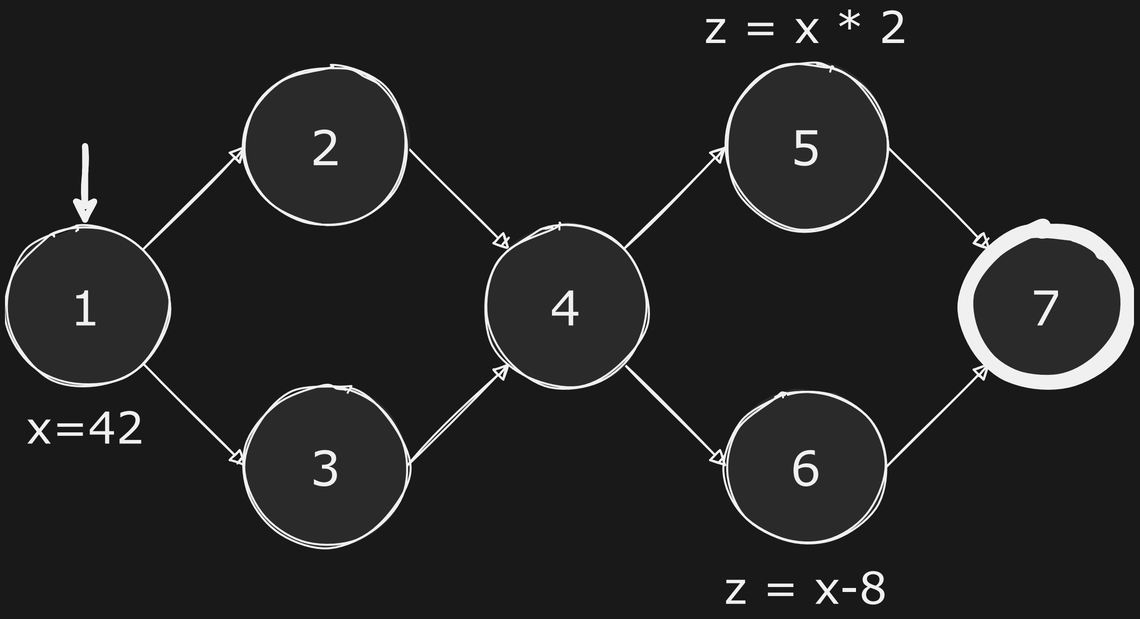
- All du-paths coverage
- For each set *S* = du (ni, nj, v), TR contains every path *d* in *S*.
- … or, make sure for each def, you explore every du-path to every use.
- Remember: du-paths are def-clear!

DATA FLOW TESTING EXAMPLE

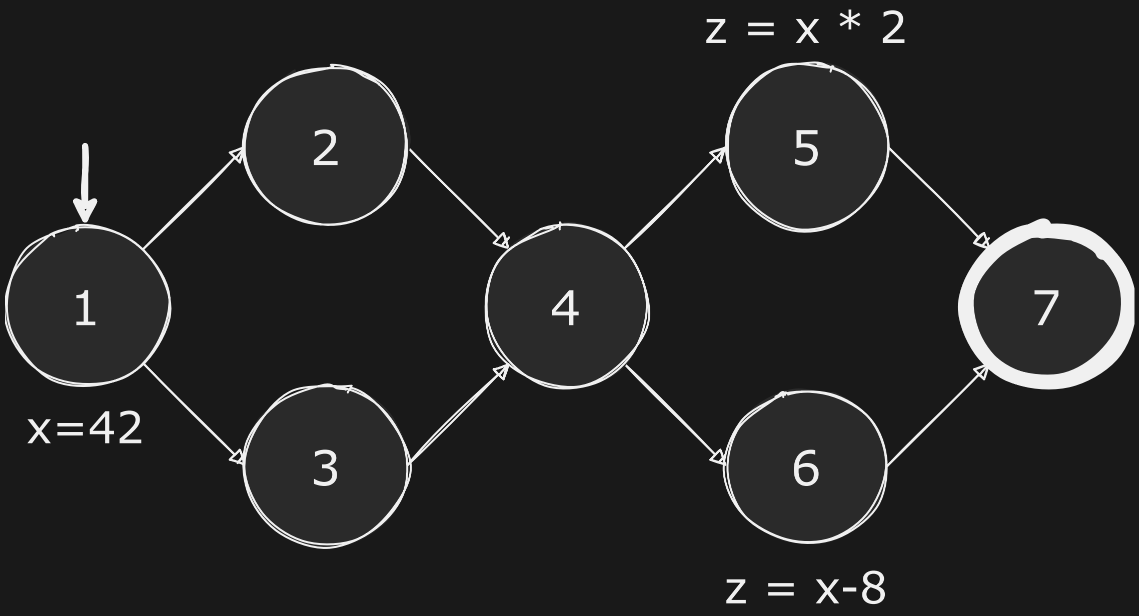


DATA FLOW TESTING EXAMPLE

- All Defs for x:
  - [1 2 4 5]

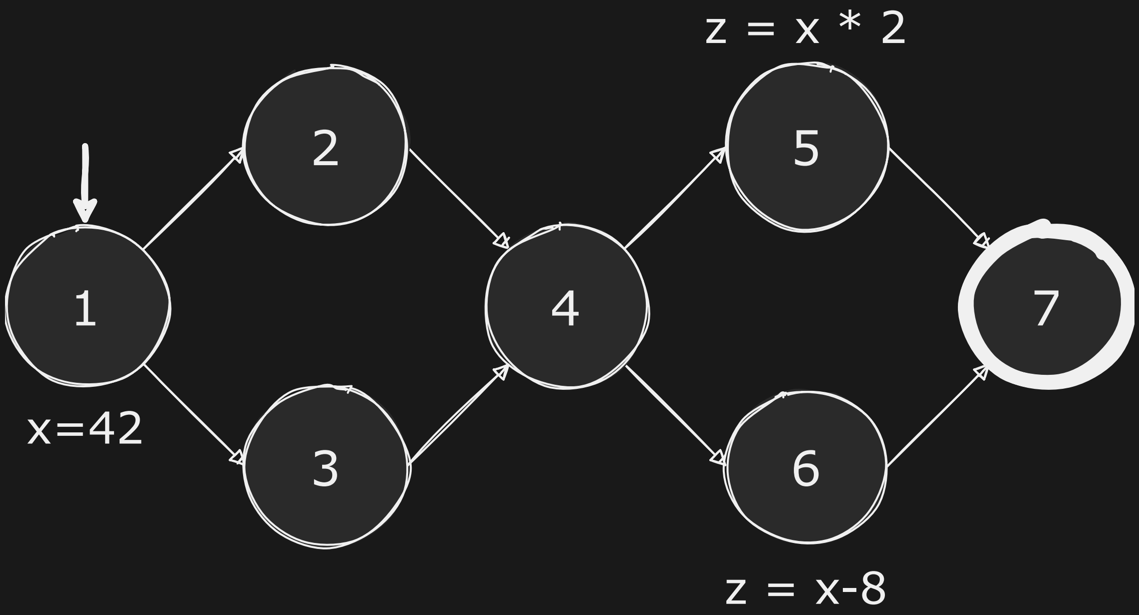


DATA FLOW TESTING EXAMPLE



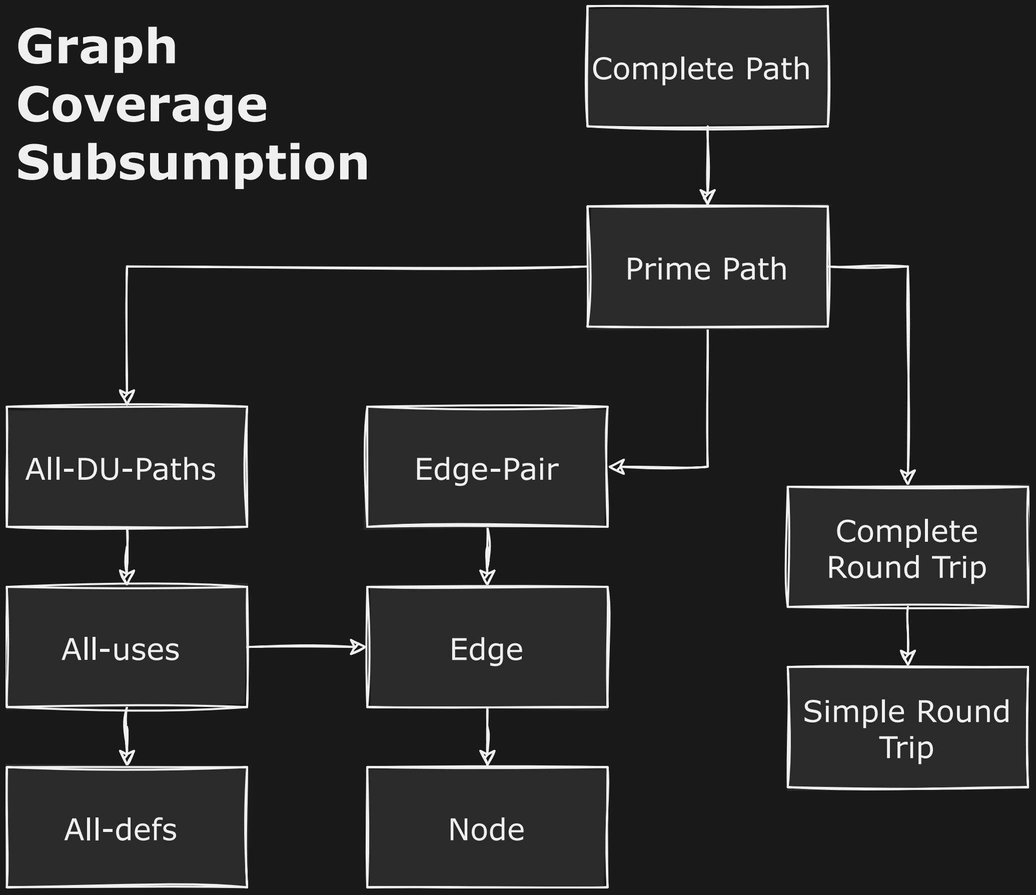
- All Defs for x:
  - [1 2 4 5]
- All Uses for x:
  - [1 2 4 5]
  - [1 2 4 6]

DATA FLOW TESTING EXAMPLE



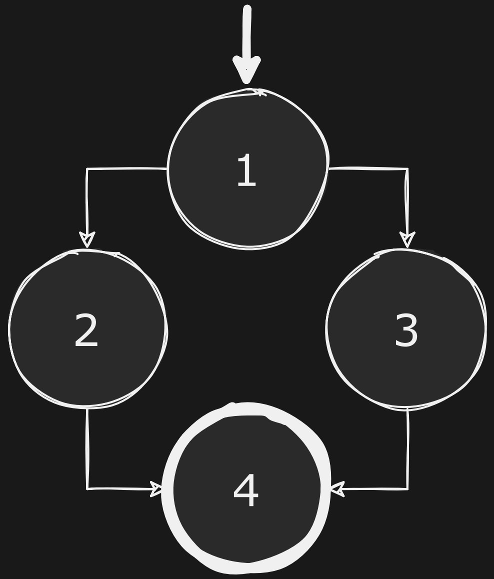
- All Defs for x:
  - [1 2 4 5]
- All Uses for x:
  - [1 2 4 5]
  - [1 2 4 6]
- All Du-Paths for x:
  - [1 2 4 5]
  - [1 3 4 5]
  - [1 2 4 6]
  - [1 3 4 6]

Graph Coverage Subsumption

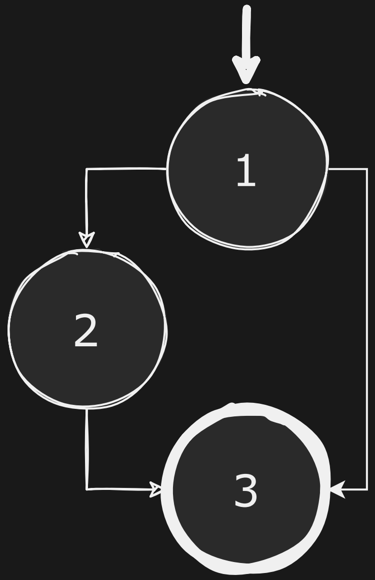


TRANSLATING SOURCE CODE TO GRAPHS

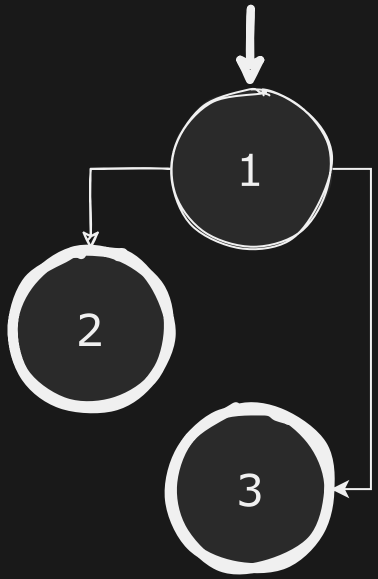
```
if( x < y)
{
   y =0;
   x = x+1;
}
else
{
   x = y;
}
```



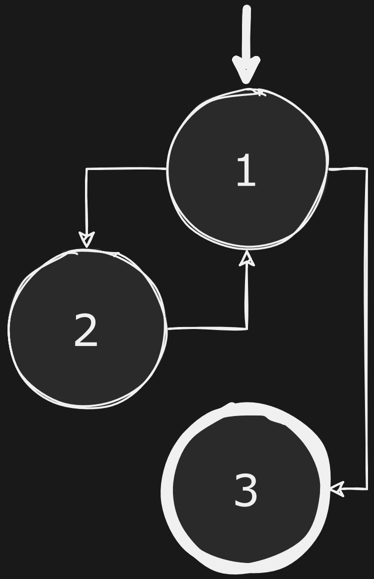
```
if( x < y)
{
   y =0;
   x = x+1;
}
```



```
if( x < y)
{
   return;
}

print(x);
return;
```

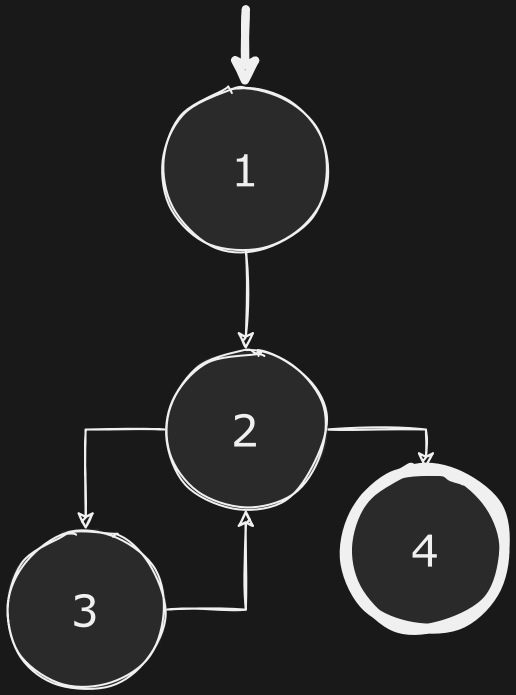


```
x = 0;
while (x<y)
{
   y = f(x,y);
   x = x + 1;
}
```

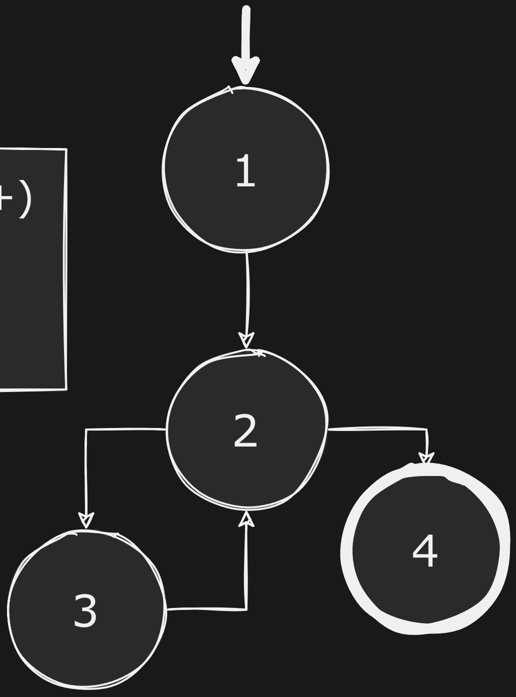


Or...

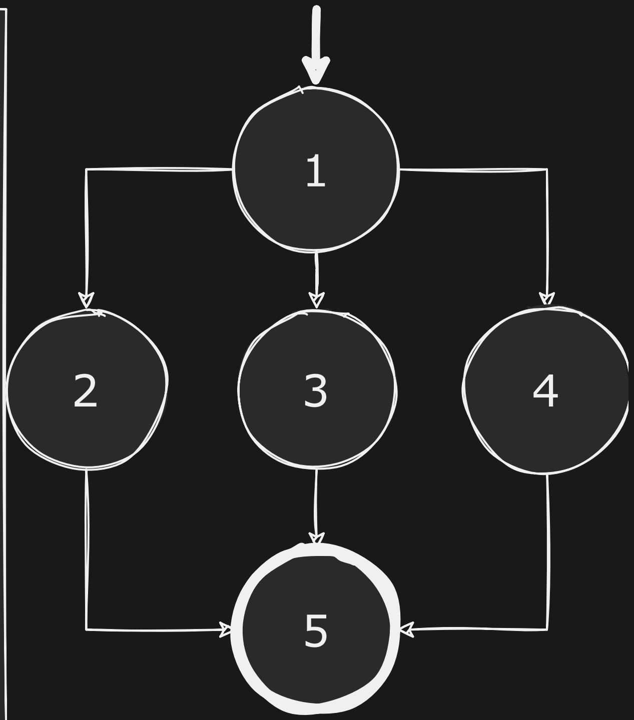
```
x = 0;
while (x<y)
{
   y = f(x,y);
   x = x + 1;
}
```

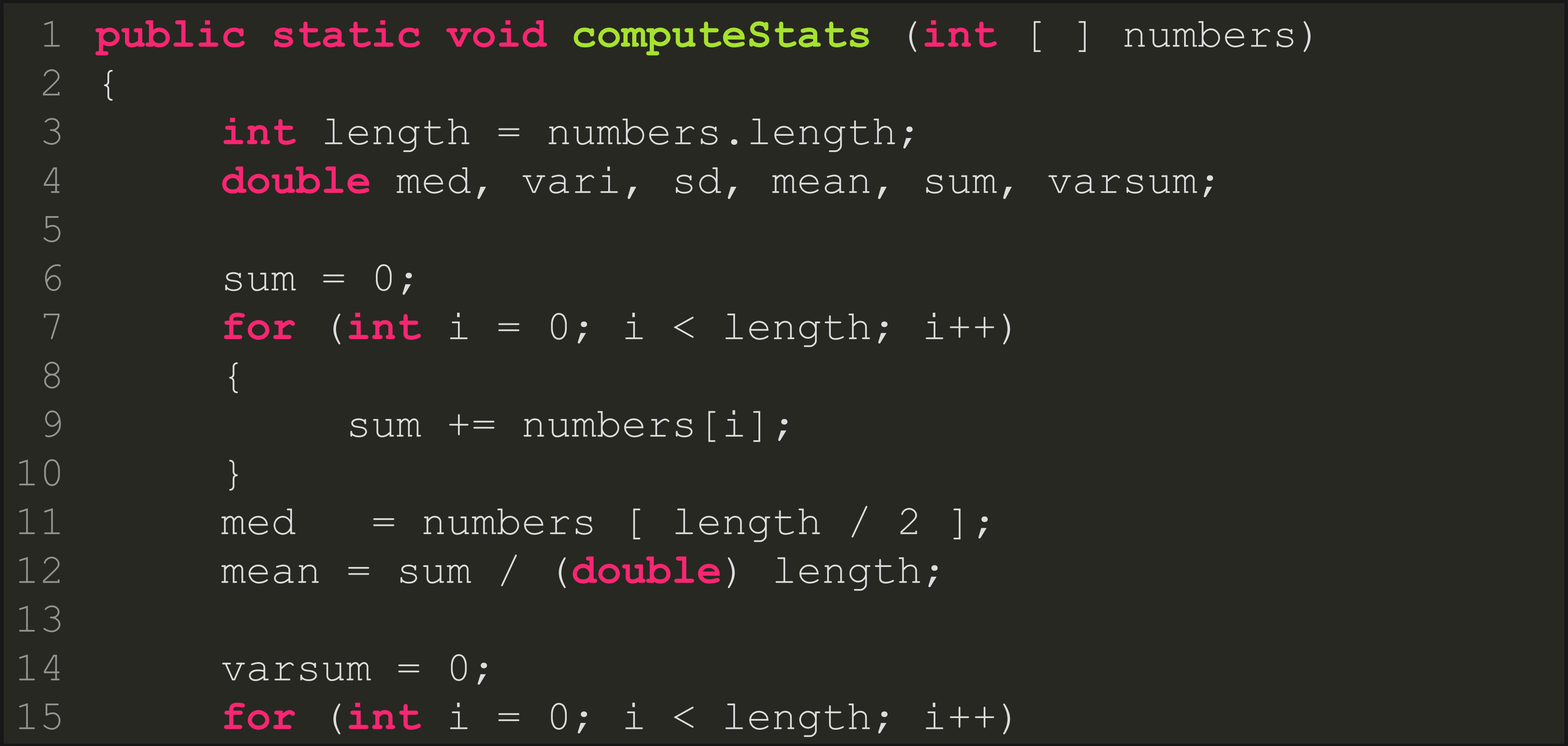


```
for(x=0;x<y;x++)
{
   y = f(x,y);
}
```

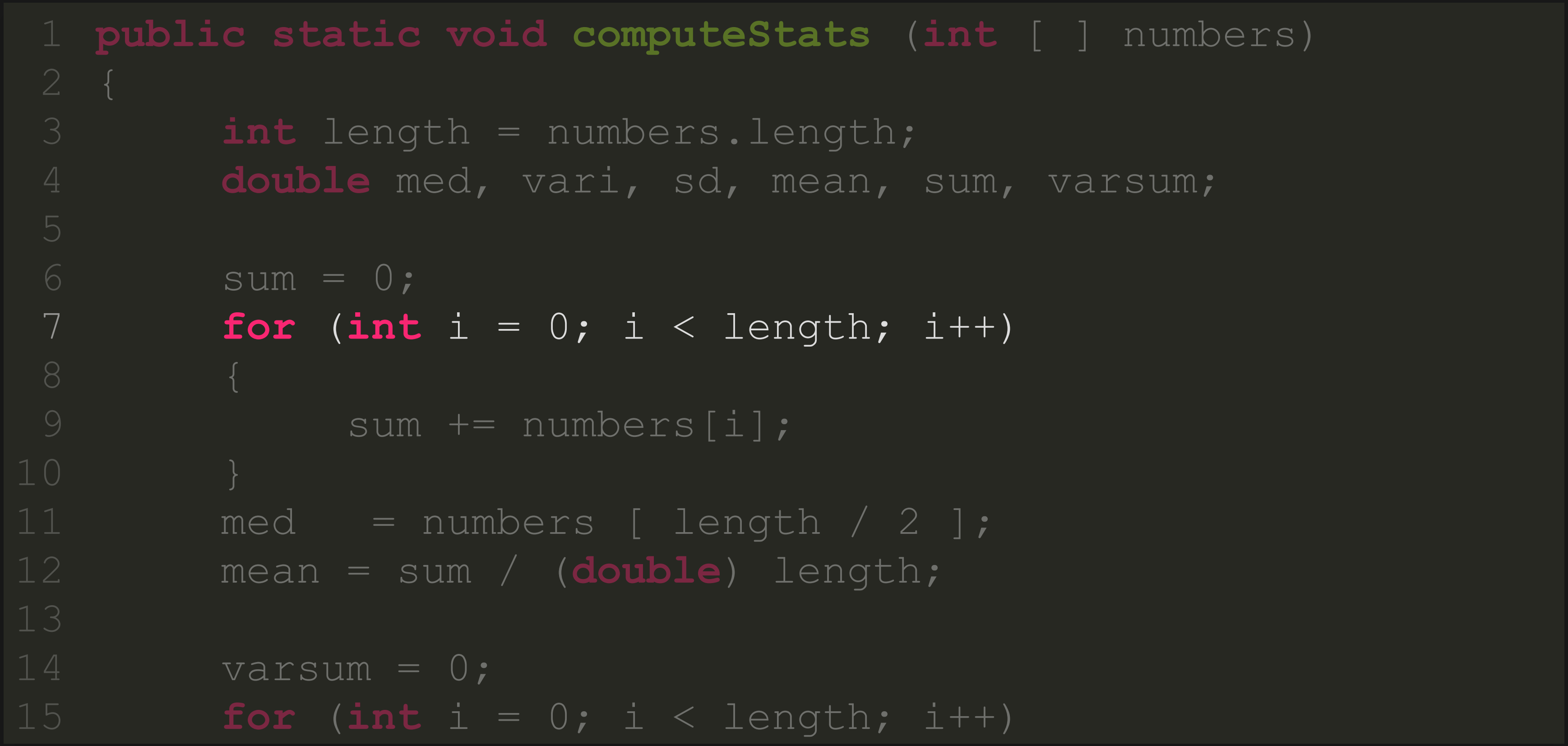


```
read(c);
switch(c)
{
   case 'N':
      y=25;
      break;
   case 'Y':
      y=50;
   default:
      y=0;
      break;
}
print(y);
```



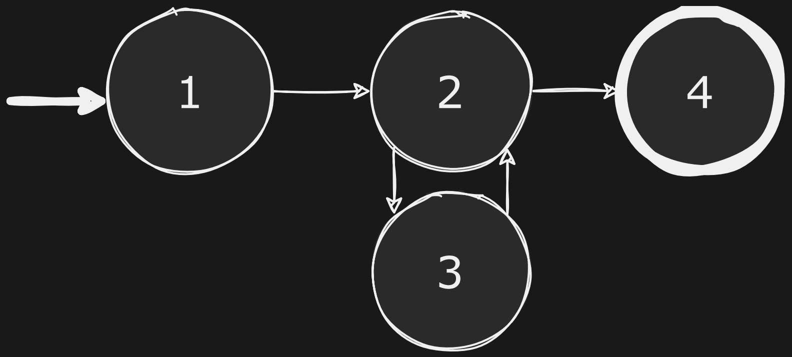


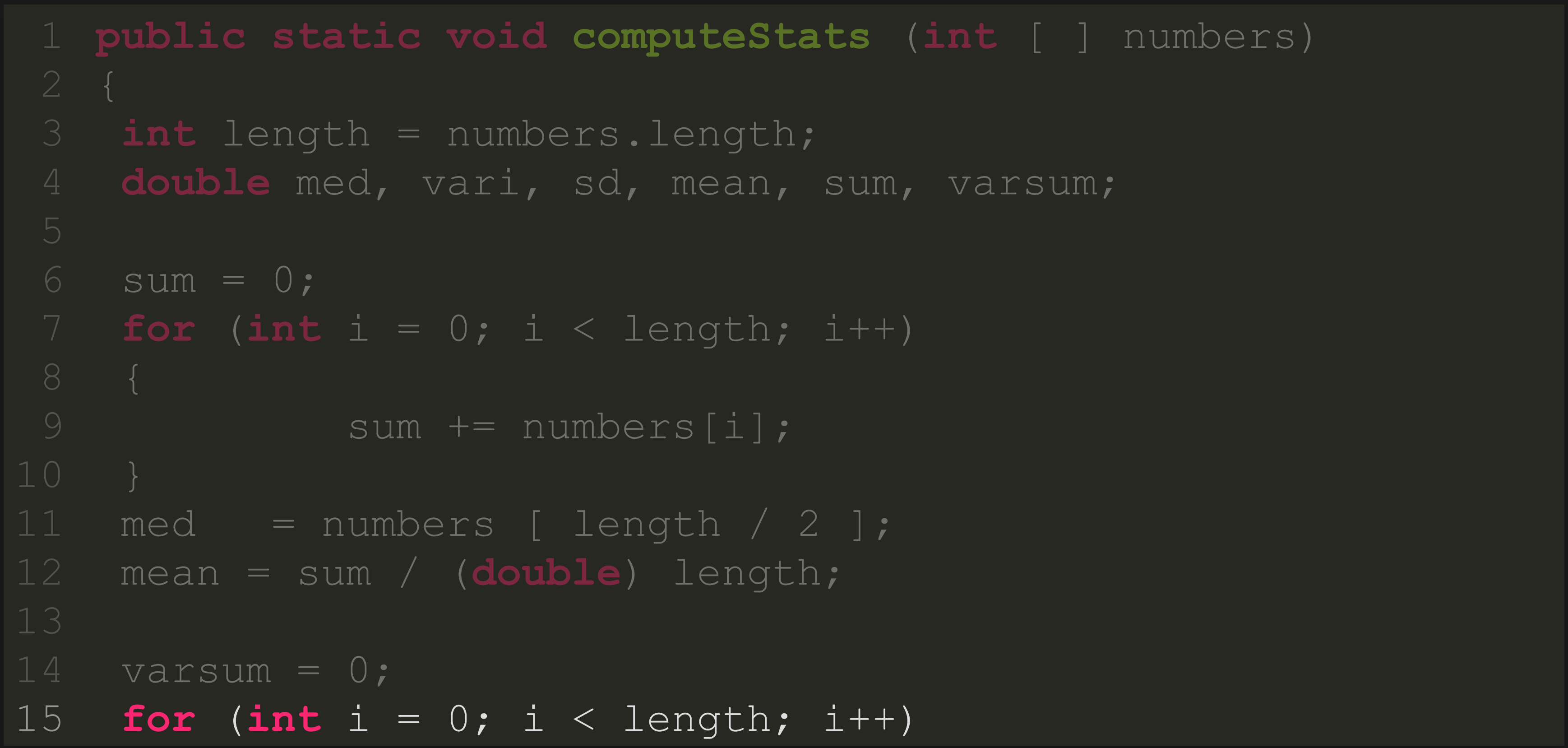
```java
public static void computeStats (int [ ] numbers)
{
     int length = numbers.length;
     double med, vari, sd, mean, sum, varsum;

     sum = 0;
     for (int i = 0; i < length; i++)
     {
          sum += numbers[i];
     }
     med   = numbers [ length / 2 ];
     mean = sum / (double) length;

     varsum = 0;
     for (int i = 0; i < length; i++)
```



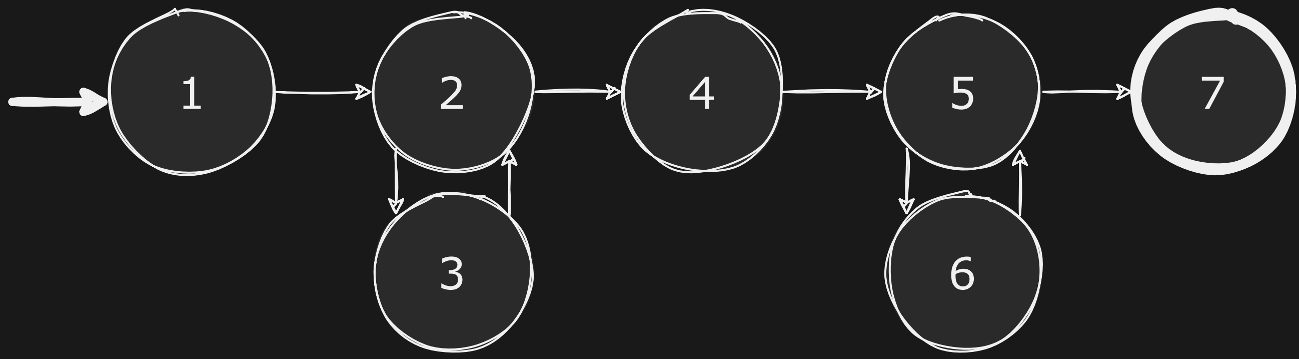
```
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

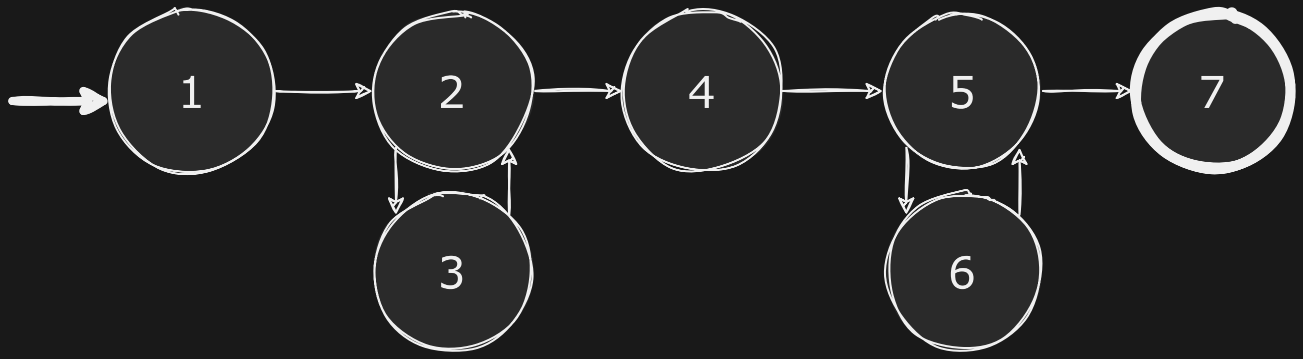
Remember what the for loop looked like in isolation?


1
2
3
4

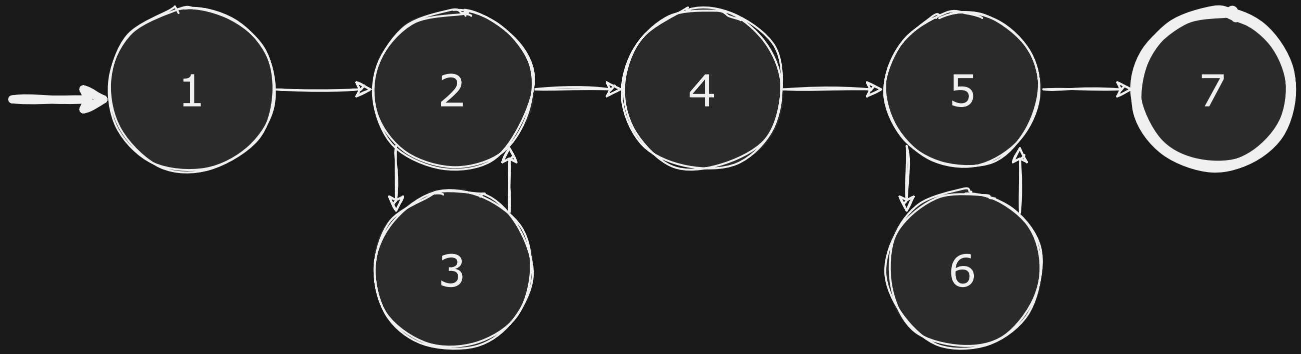


```java
public static void computeStats (int [ ] numbers)
{
 int length = numbers.length;
 double med, vari, sd, mean, sum, varsum;

 sum = 0;
 for (int i = 0; i < length; i++)
 {
          sum += numbers[i];
 }
 med   = numbers [ length / 2 ];
 mean = sum / (double) length;

 varsum = 0;
 for (int i = 0; i < length; i++)
```




1
2
4
5
7
3
6

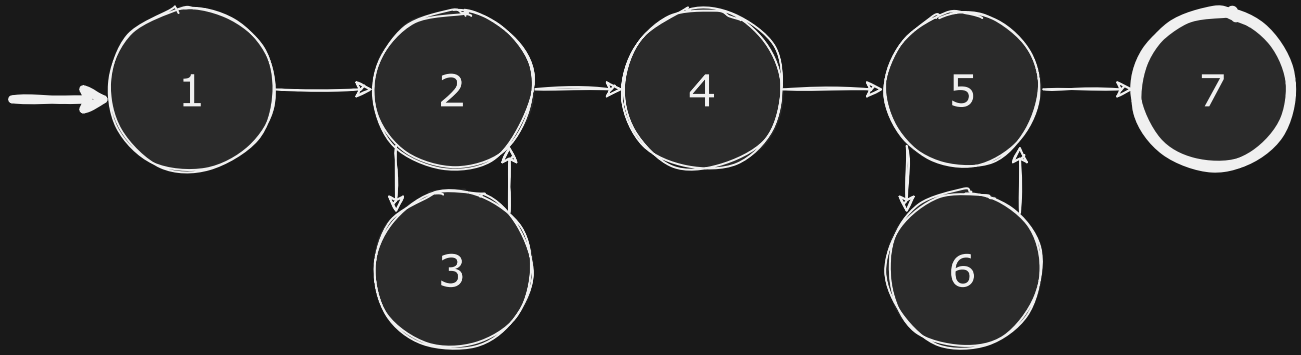
PRIME PATH COVERAGE



Test Requirements

-

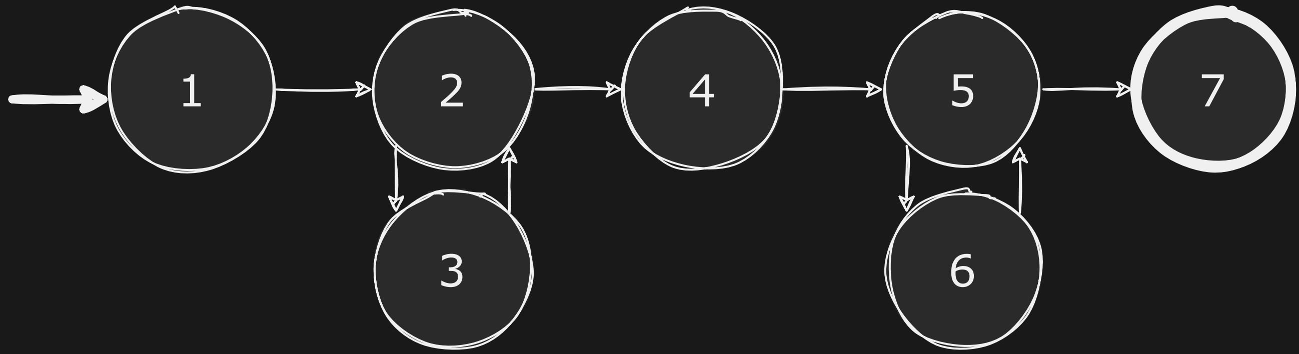
PRIME PATH COVERAGE



Test Requirements

- [2 3 2]

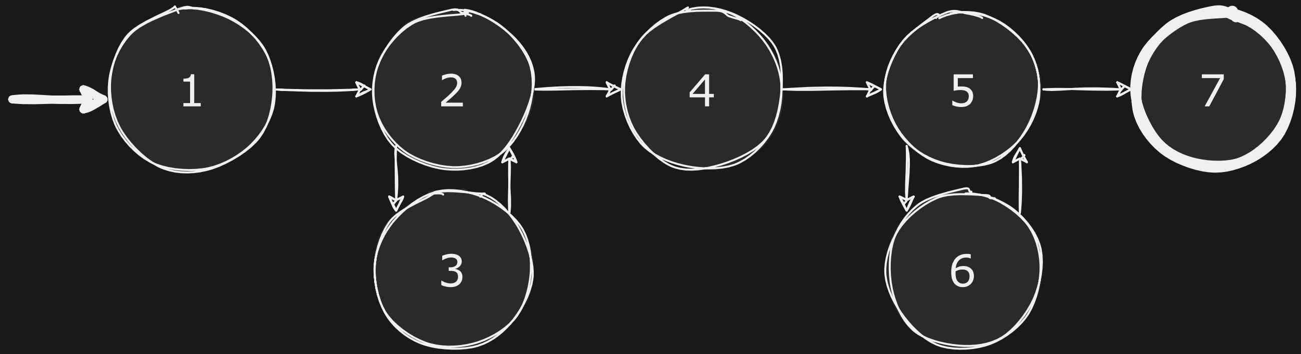
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3]

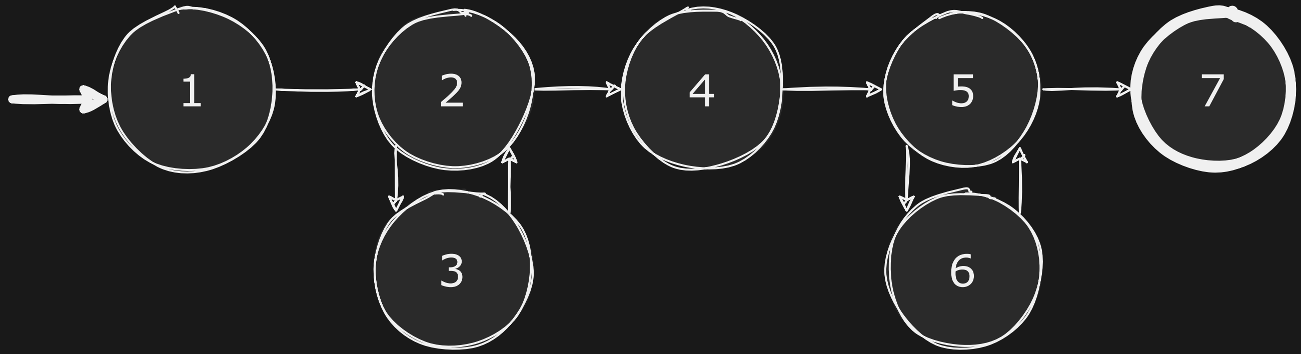
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5]

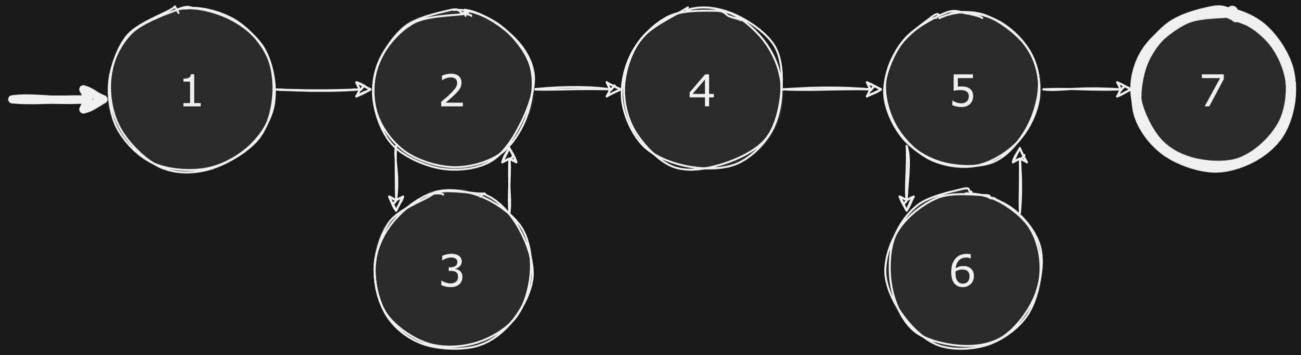
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6]

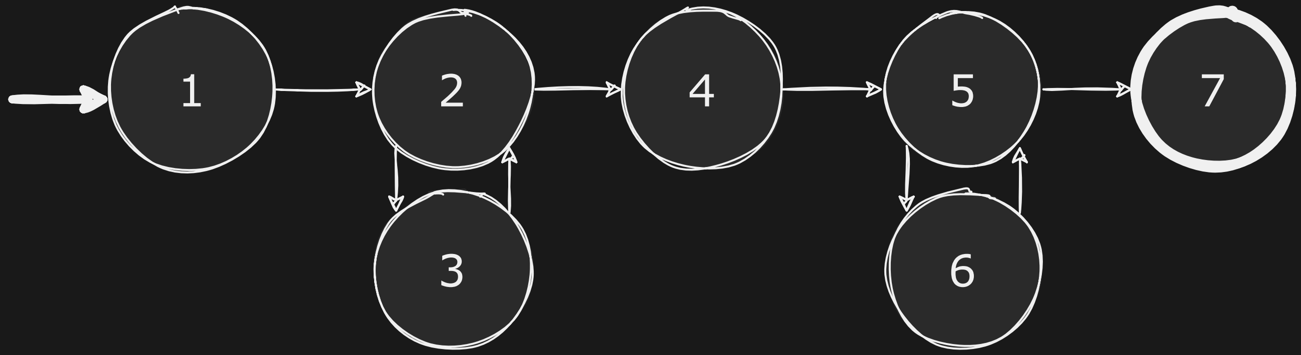
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6] [6 5 7]

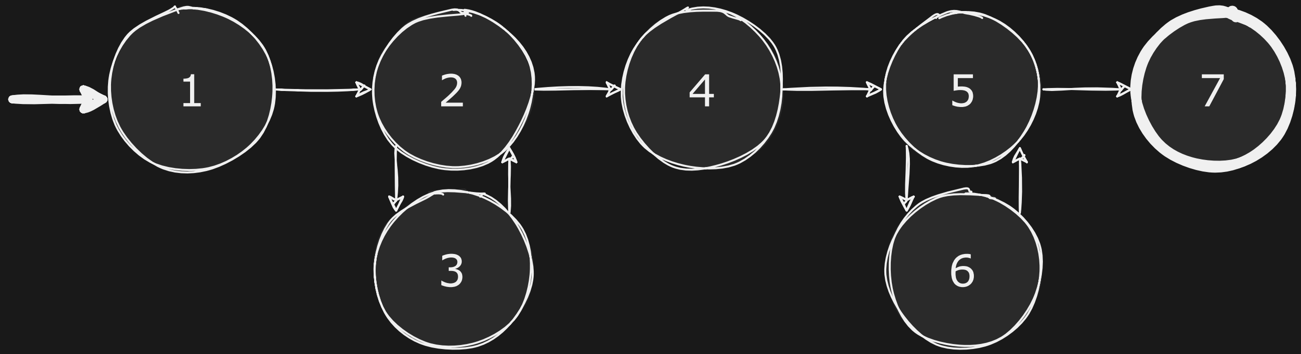
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6] [6 5 7] [1 2 4 5 7]

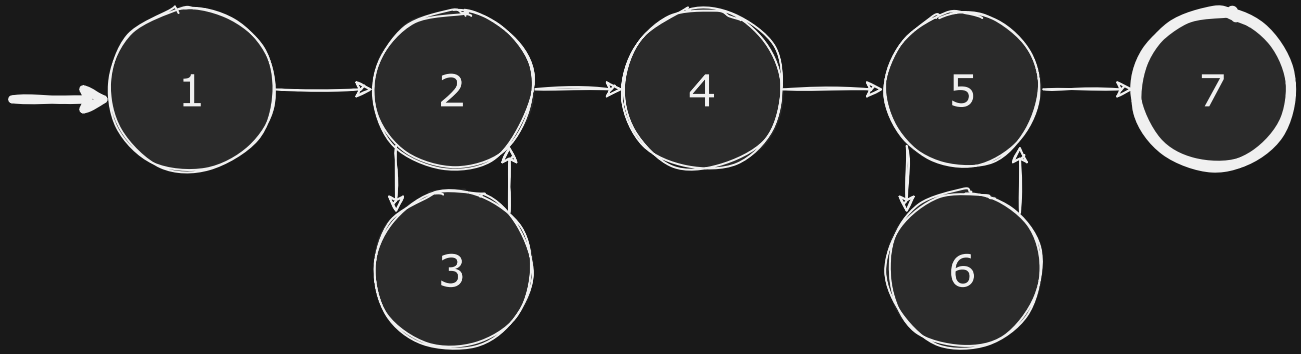
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6] [6 5 7] [1 2 4 5 7] [3 2 4 5 7]

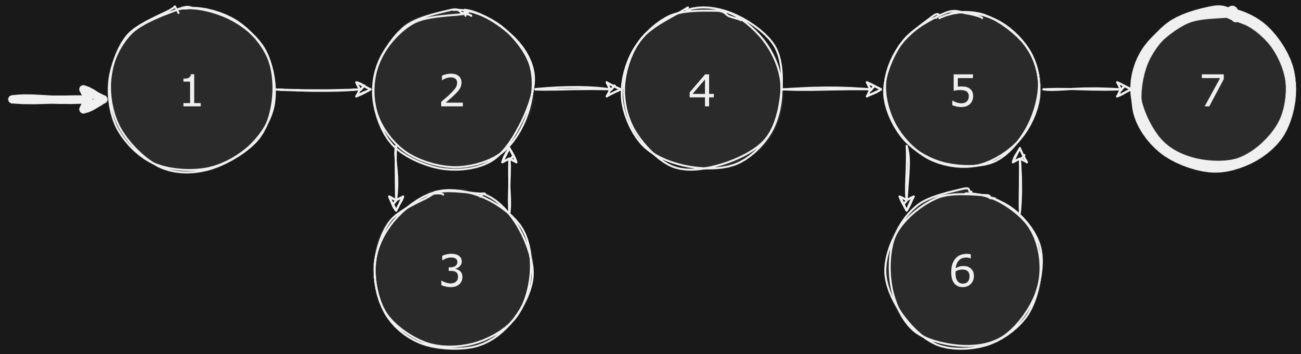
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6] [6 5 7] [1 2 4 5 7] [3 2 4 5 7] [1 2 3]

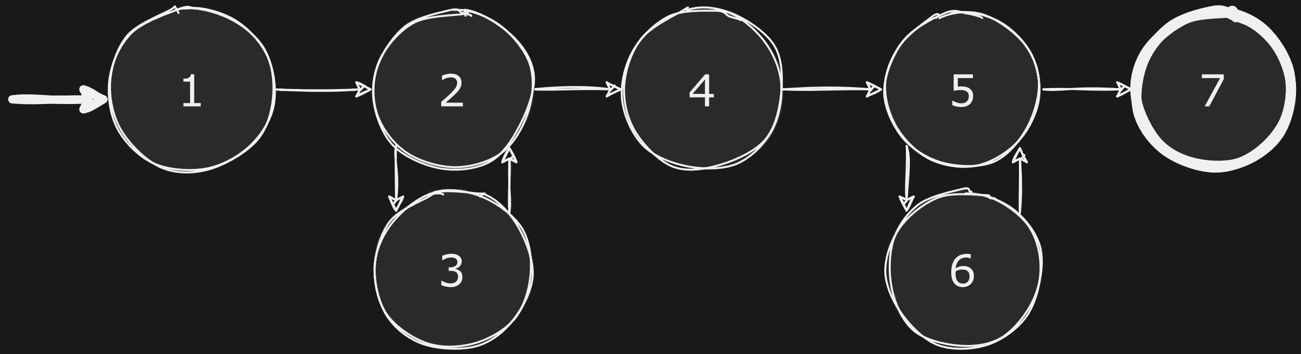
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6] [6 5 7] [1 2 4 5 7] [3 2 4 5 7] [1 2 3] [1 2 4 5 6]

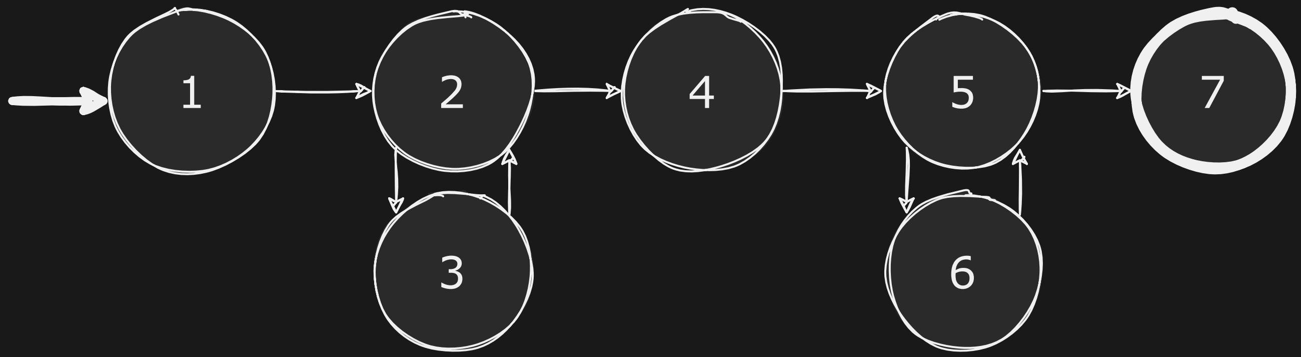
PRIME PATH COVERAGE



Test Requirements

- [2 3 2] [3 2 3] [5 6 5] [6 5 6] [6 5 7] [1 2 4 5 7] [3 2 4 5 7] [1 2 3] [1 2 4 5 6] [3 2 4 5 6]

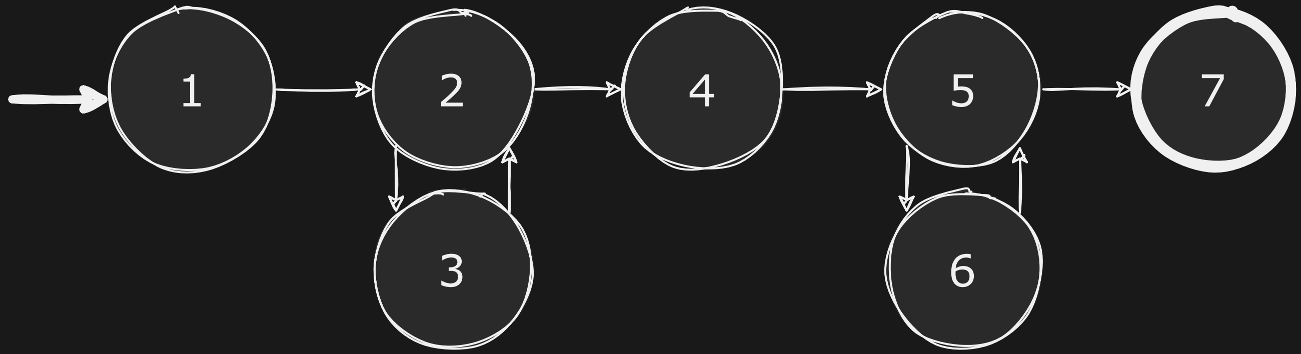
PRIME PATH COVERAGE



Test Paths

-

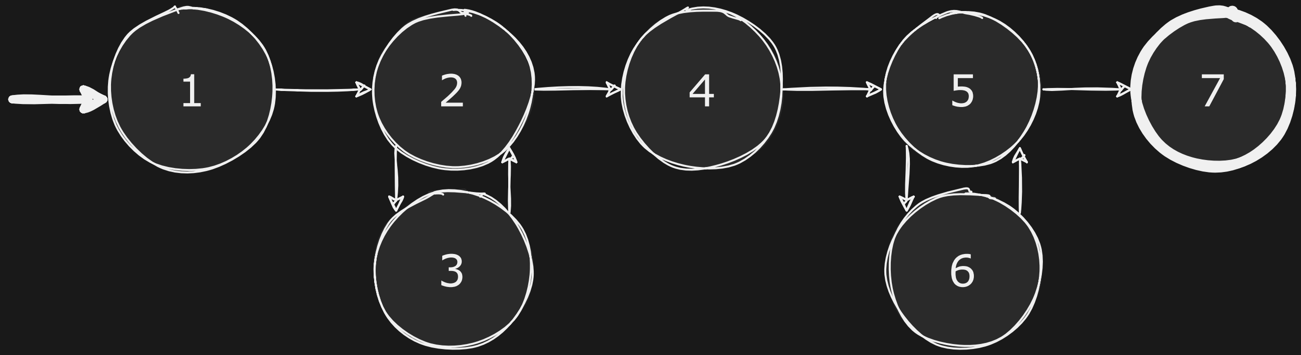
PRIME PATH COVERAGE



Test Paths

- [1 2 3 2 4 5 6 5 7]

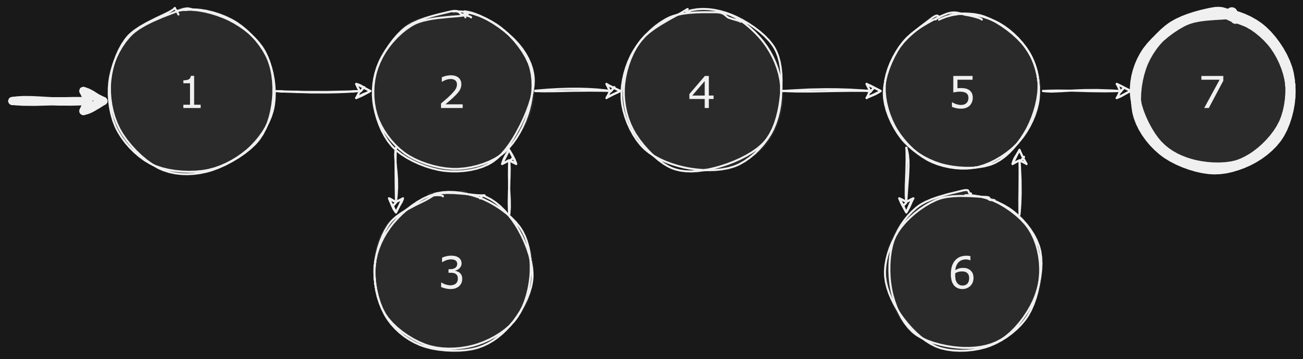
PRIME PATH COVERAGE



Test Paths

- [1 2 3 2 4 5 6 5 7] [1 2 4 5 7]

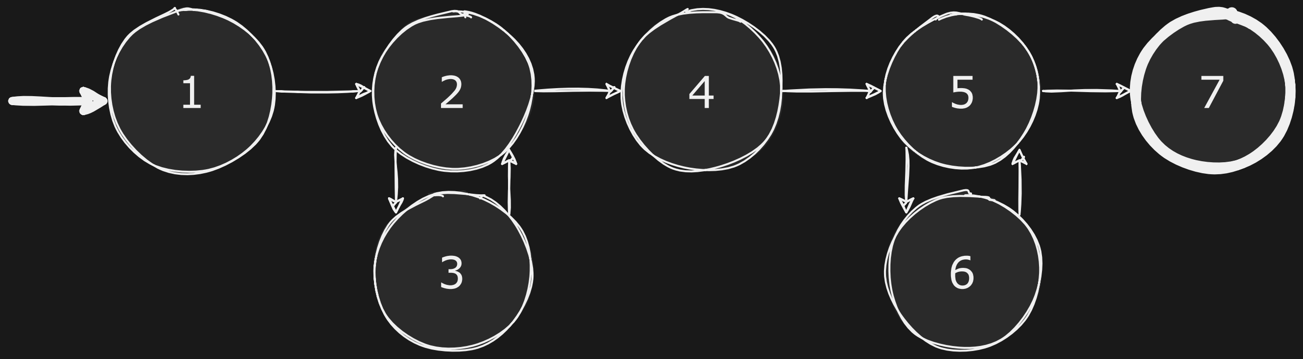
PRIME PATH COVERAGE



Test Paths

- [1 2 3 2 4 5 6 5 7] [1 2 4 5 7] [1 2 4 5 6 5 7]

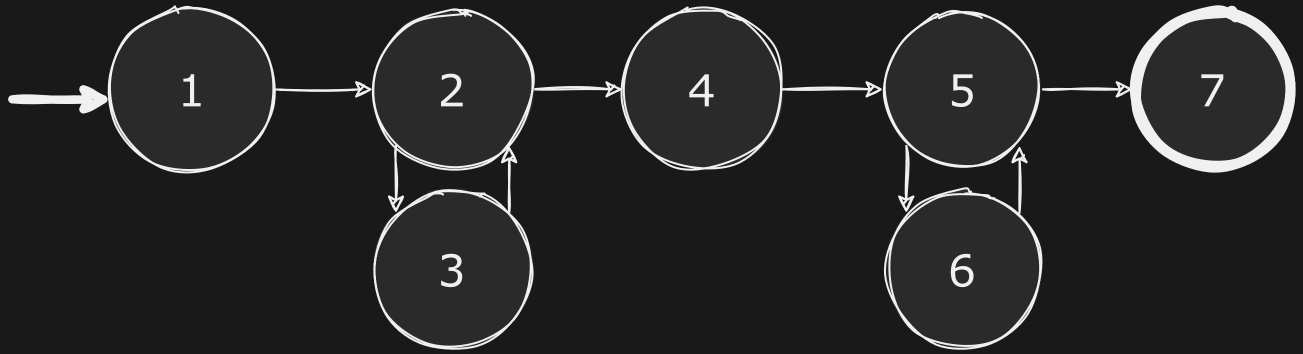
PRIME PATH COVERAGE



Test Paths

- [1 2 3 2 4 5 6 5 7] [1 2 4 5 7] [1 2 4 5 6 5 7] [1 2 3 2 4 5 7]

PRIME PATH COVERAGE



Test Paths

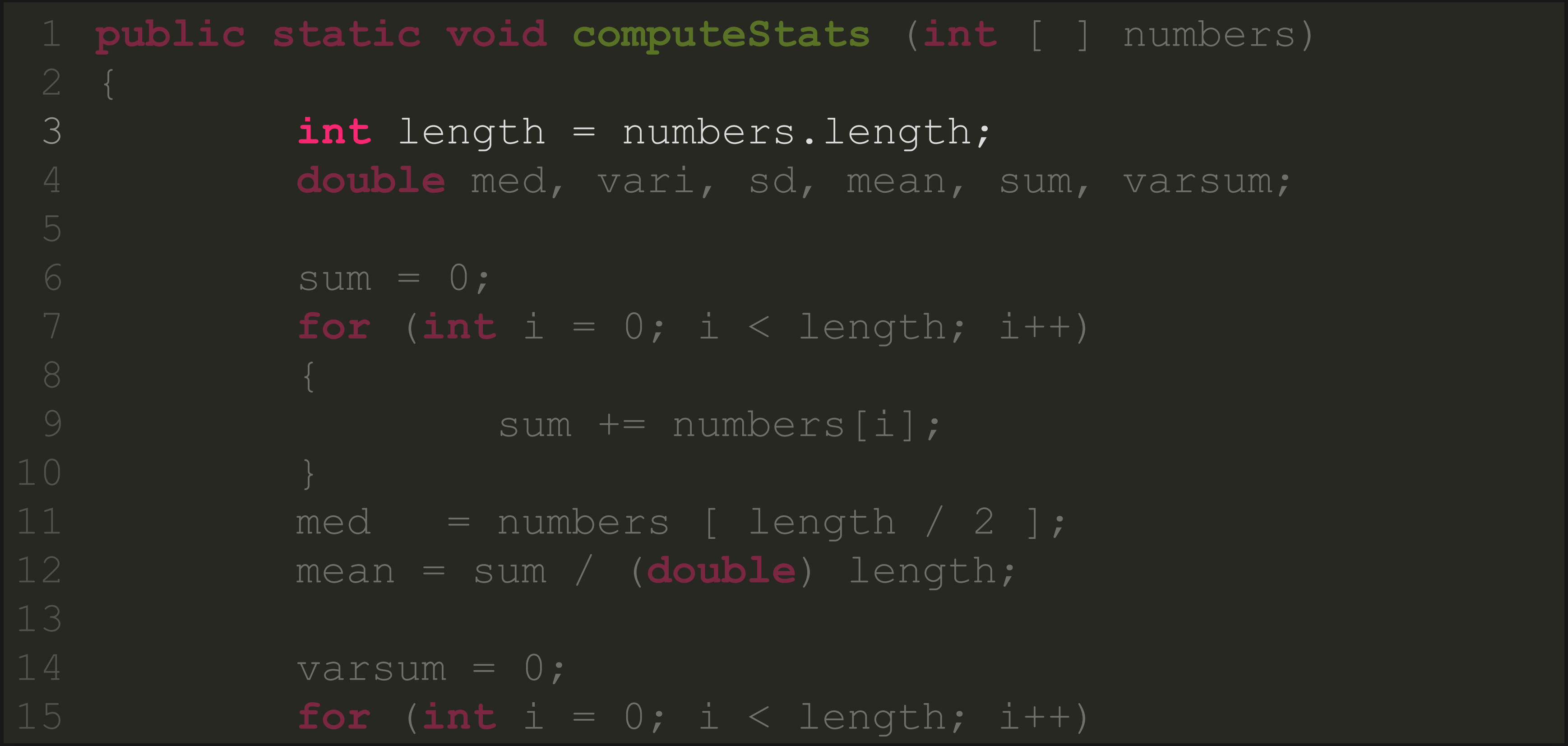
- [1 2 3 2 4 5 6 5 7] [1 2 4 5 7] [1 2 4 5 6 5 7] [1 2 3 2 4 5 7] [1 2 3 2 3 2 4 5 6 5 6 5 7]

DATA FLOW AND SOURCE

DEFS

```
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

DEFS



```java
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

DEFS

```java
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

DEFS

```
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

DEFS

```
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
        {
```

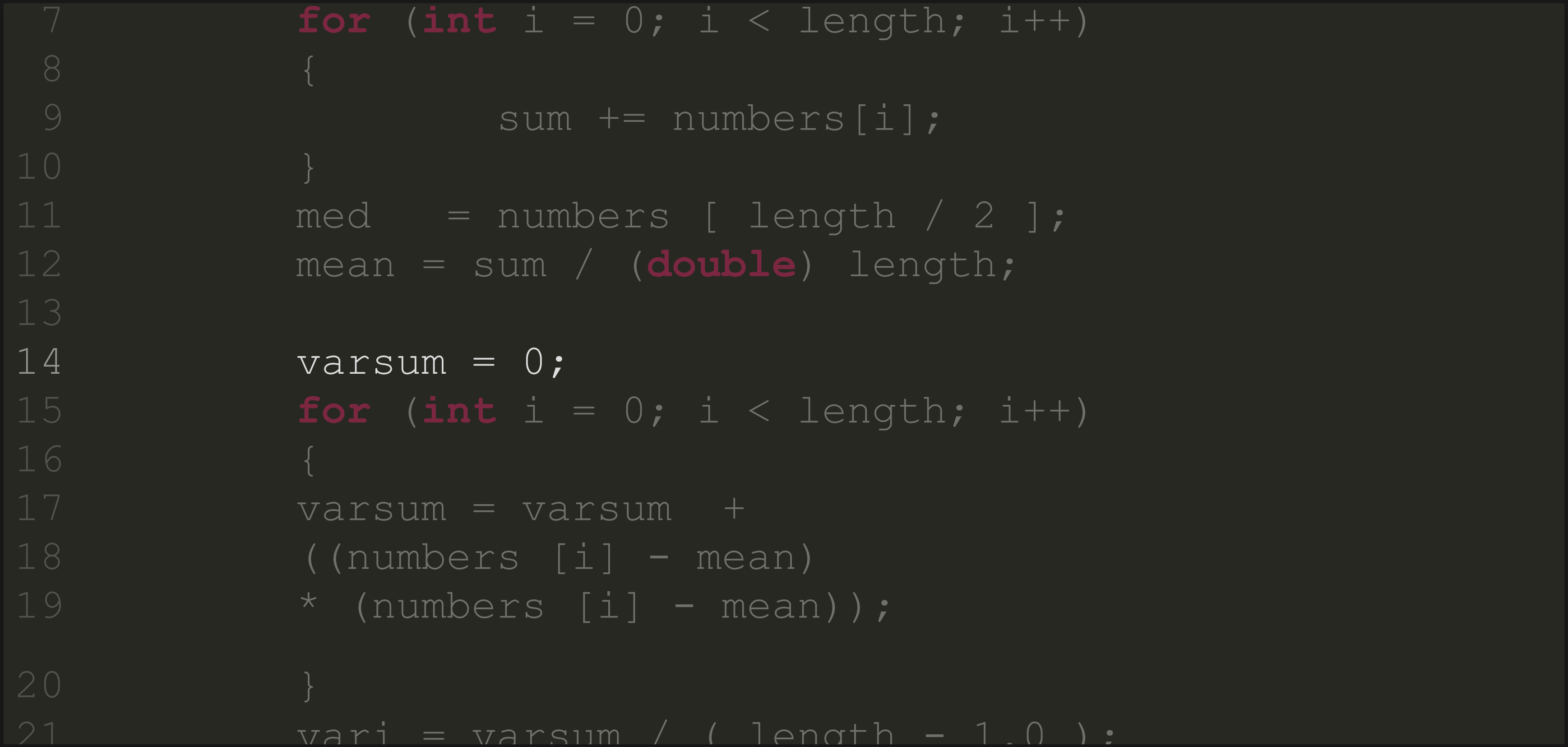
DEFS

```
double med, vari, sd, mean, sum, varsum;

sum = 0;
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
```

DEFS

```

sum = 0;
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
*  (numbers [i] - mean));
```

DEFS

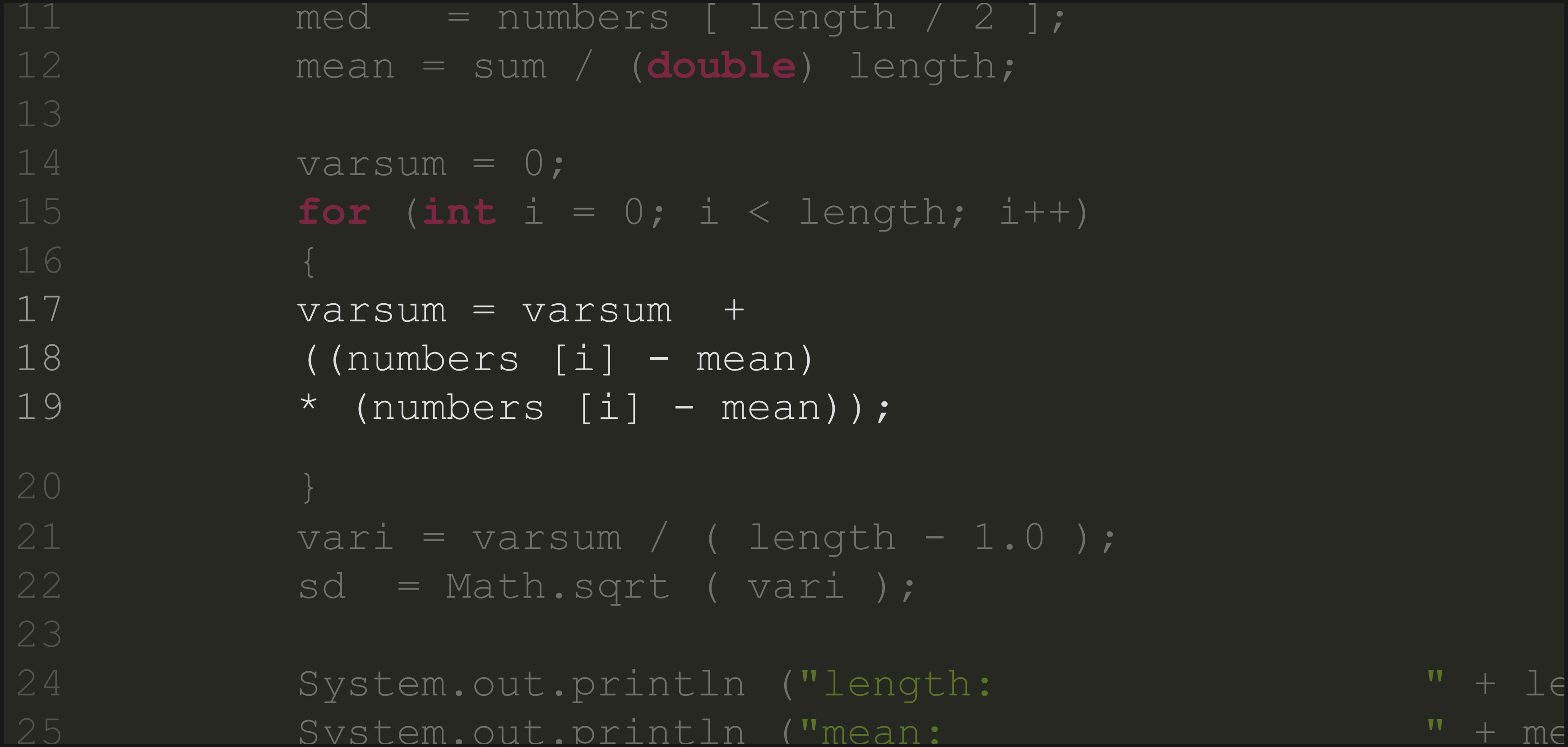


```
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
```

DEFS

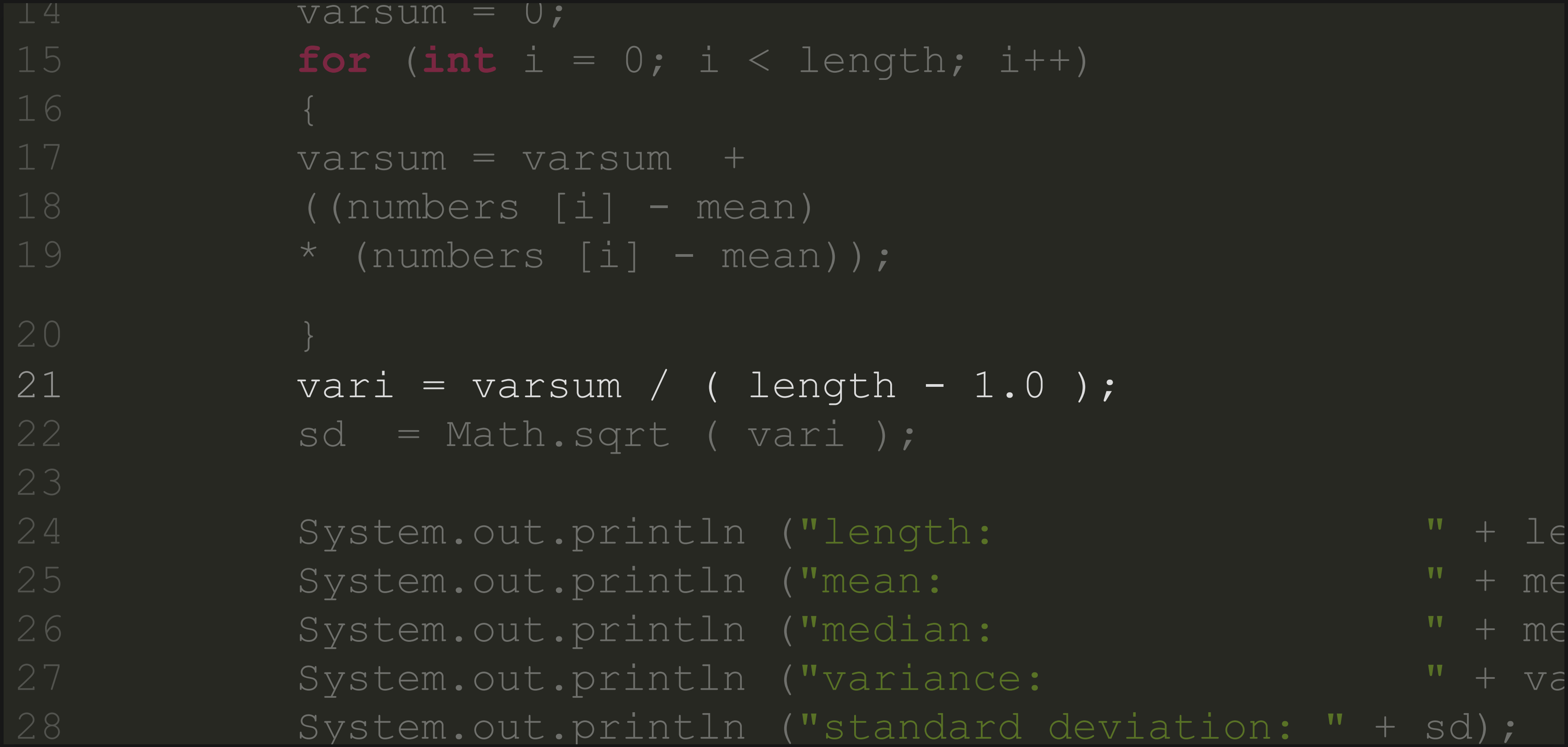
```
{
        sum += numbers[i];
}
med    = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
sd   = Math.sqrt ( vari );
```

DEFS



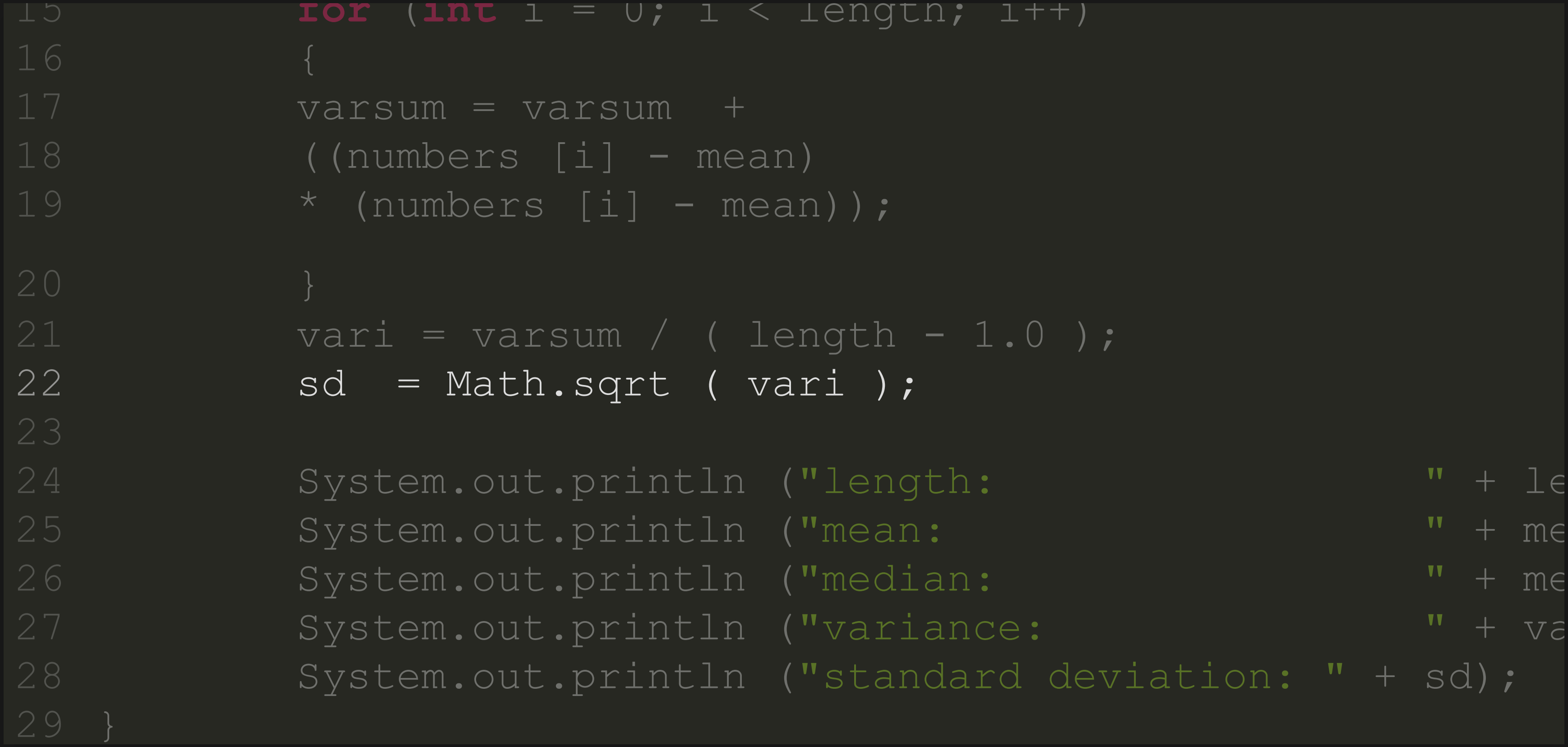
```
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
sd  = Math.sqrt ( vari );

System.out.println ("length:                 " + le
System.out.println ("mean:                   " + me
```

DEFS



```
varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
*  (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
sd  = Math.sqrt ( vari );

System.out.println ("length:                 " + le
System.out.println ("mean:                   " + me
System.out.println ("median:                 " + me
System.out.println ("variance:               " + va
System.out.println ("standard deviation: " + sd);
```

DEFS



```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        * (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```

USES

```
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

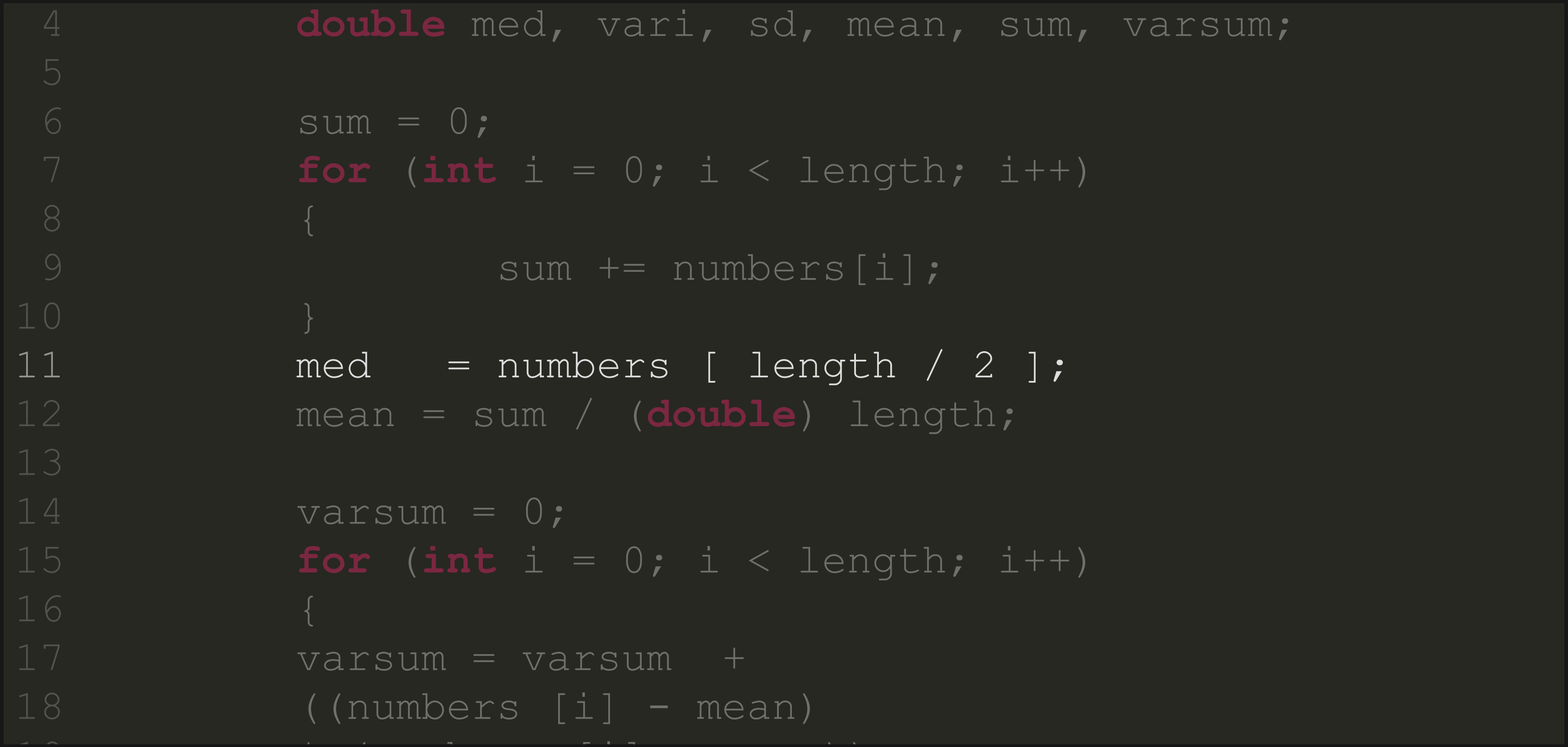
USES

```
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

USES

```
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
        {
```

USES



```
double med, vari, sd, mean, sum, varsum;

sum = 0;
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
```

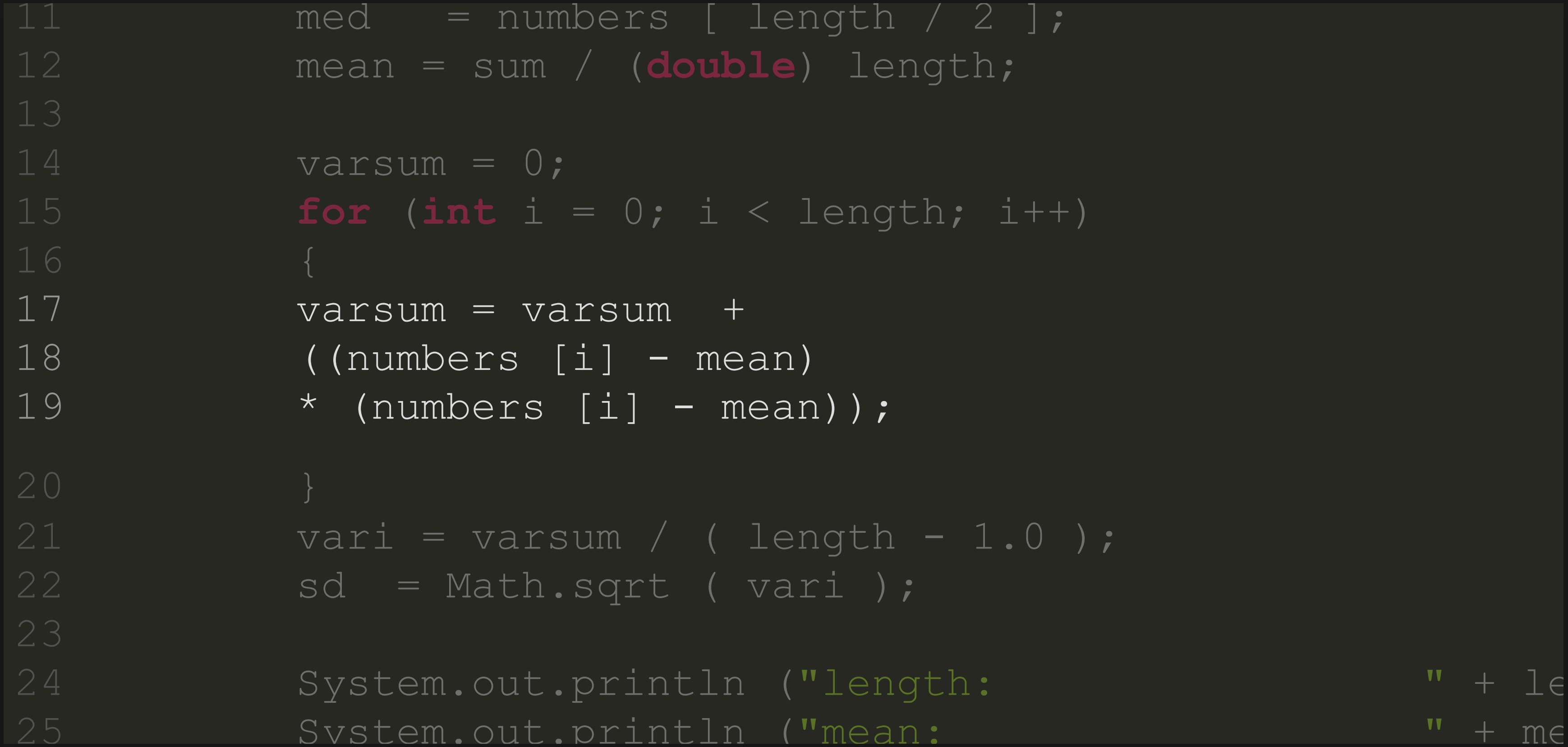
USES

```

sum = 0;
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));
```

USES

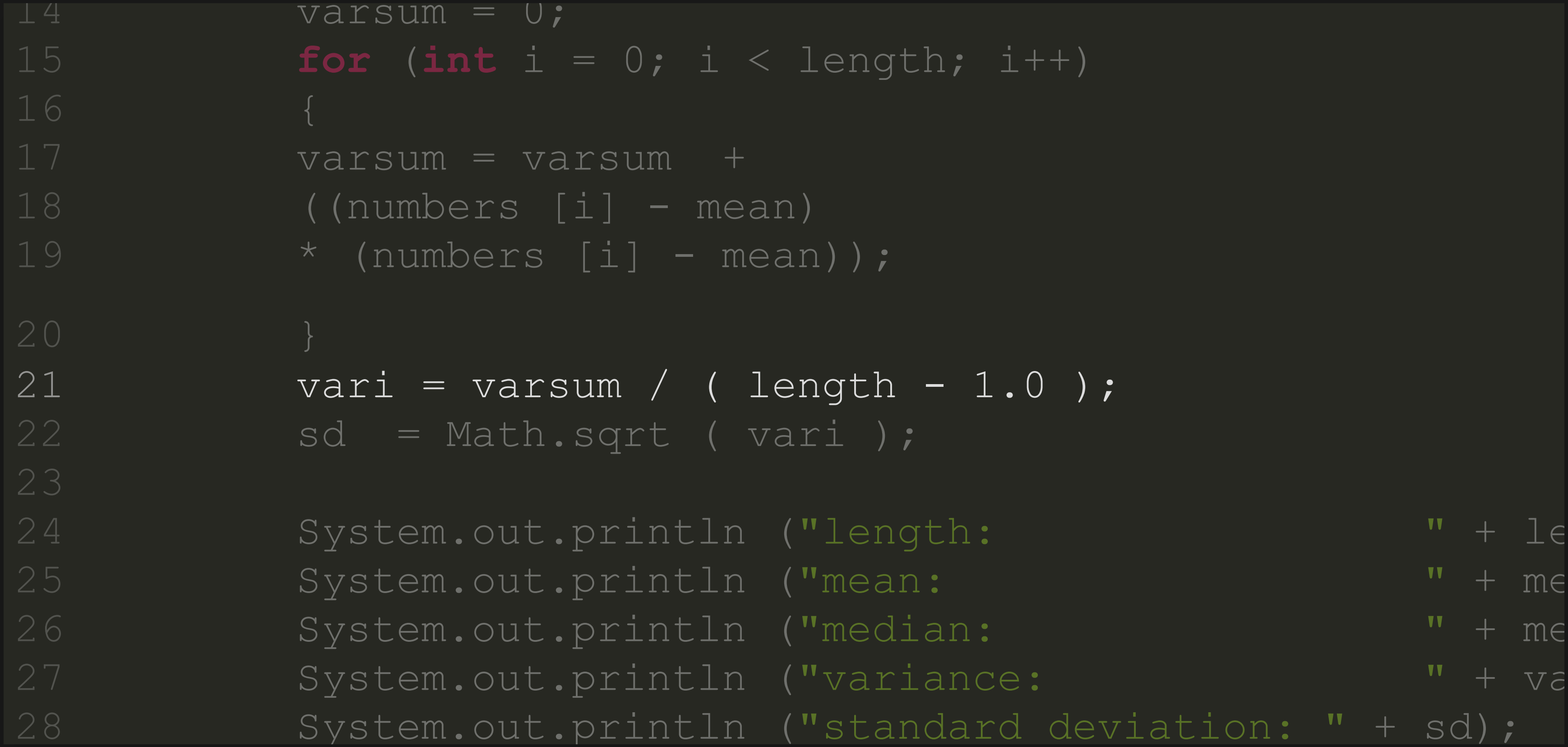
```
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
sd  = Math.sqrt ( vari );
```

USES



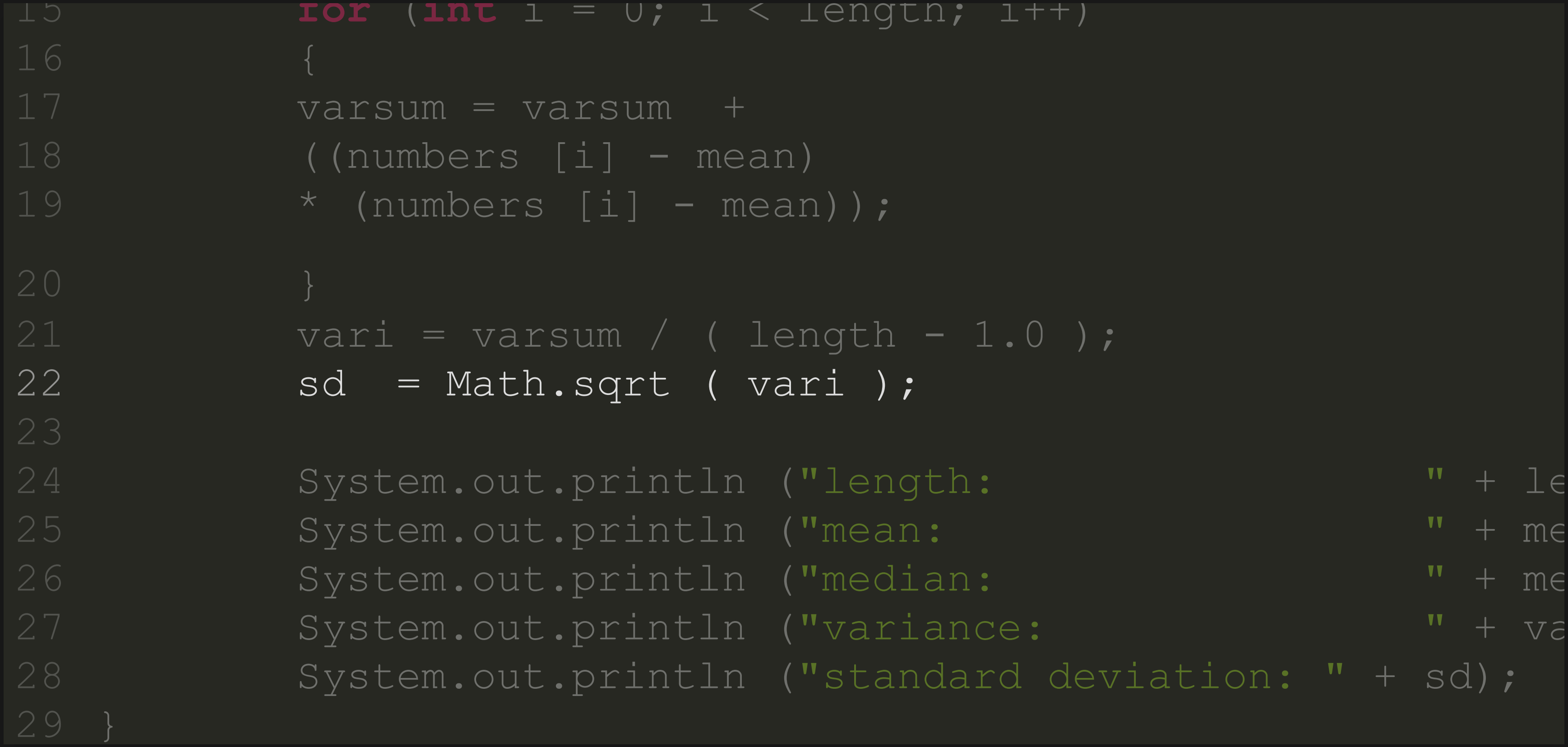
```
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
sd  = Math.sqrt ( vari );

System.out.println ("length:                    " + le
System.out.println ("mean:                      " + me
```

USES



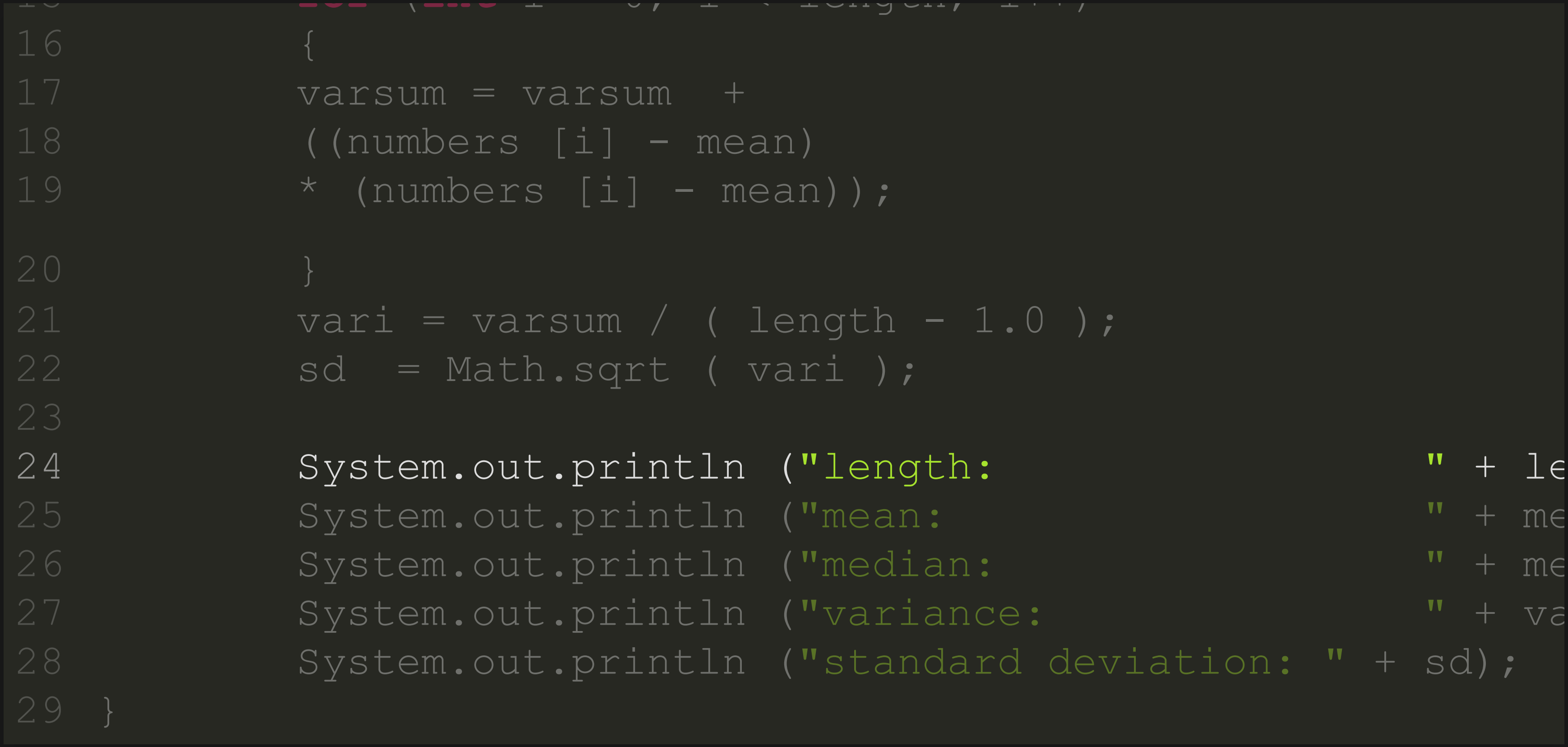
```
varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
* (numbers [i] - mean));

}
vari = varsum / ( length - 1.0 );
sd  = Math.sqrt ( vari );

System.out.println ("length:                  " + le
System.out.println ("mean:                    " + me
System.out.println ("median:                  " + me
System.out.println ("variance:                " + va
System.out.println ("standard deviation: " + sd);
```

USES



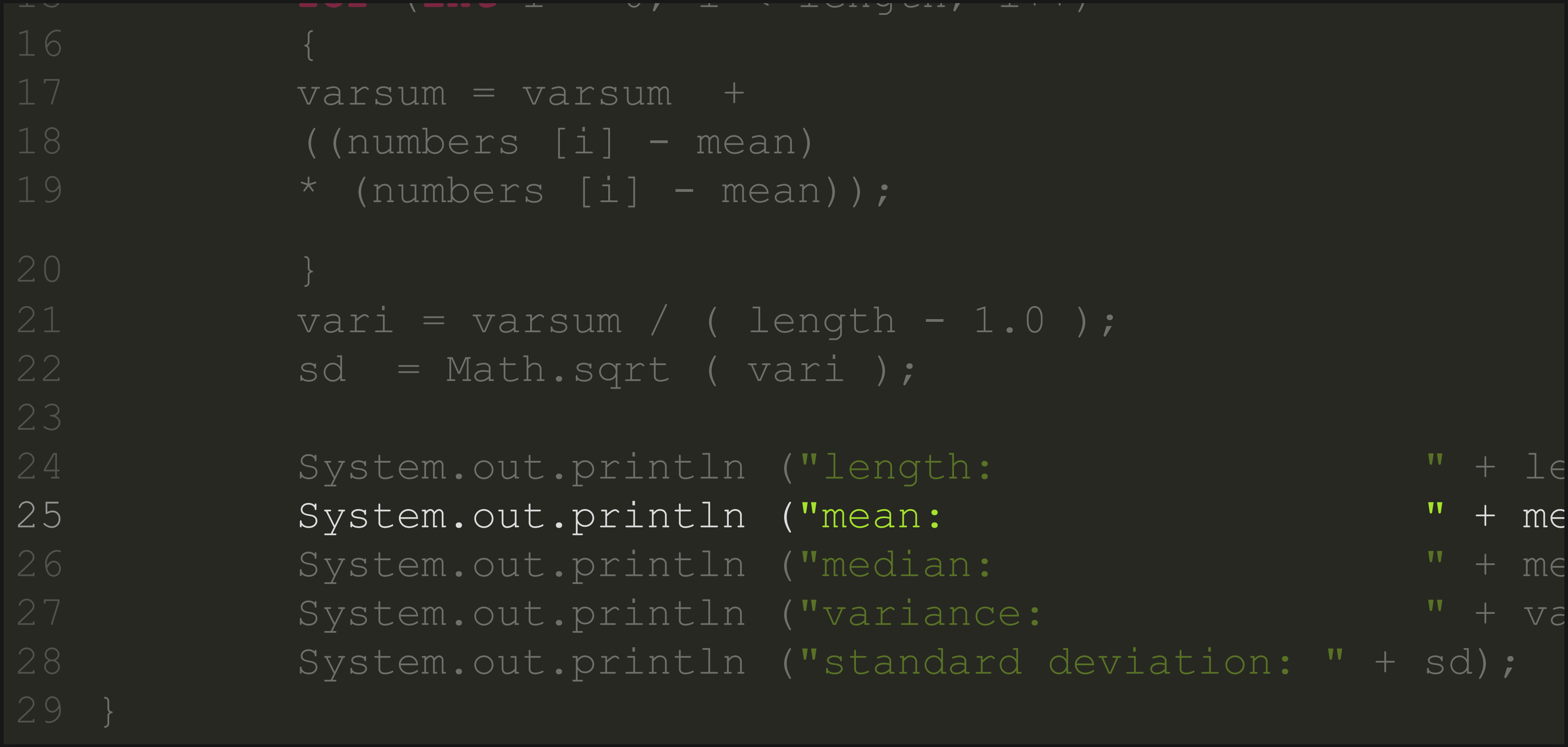
```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        * (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```

USES



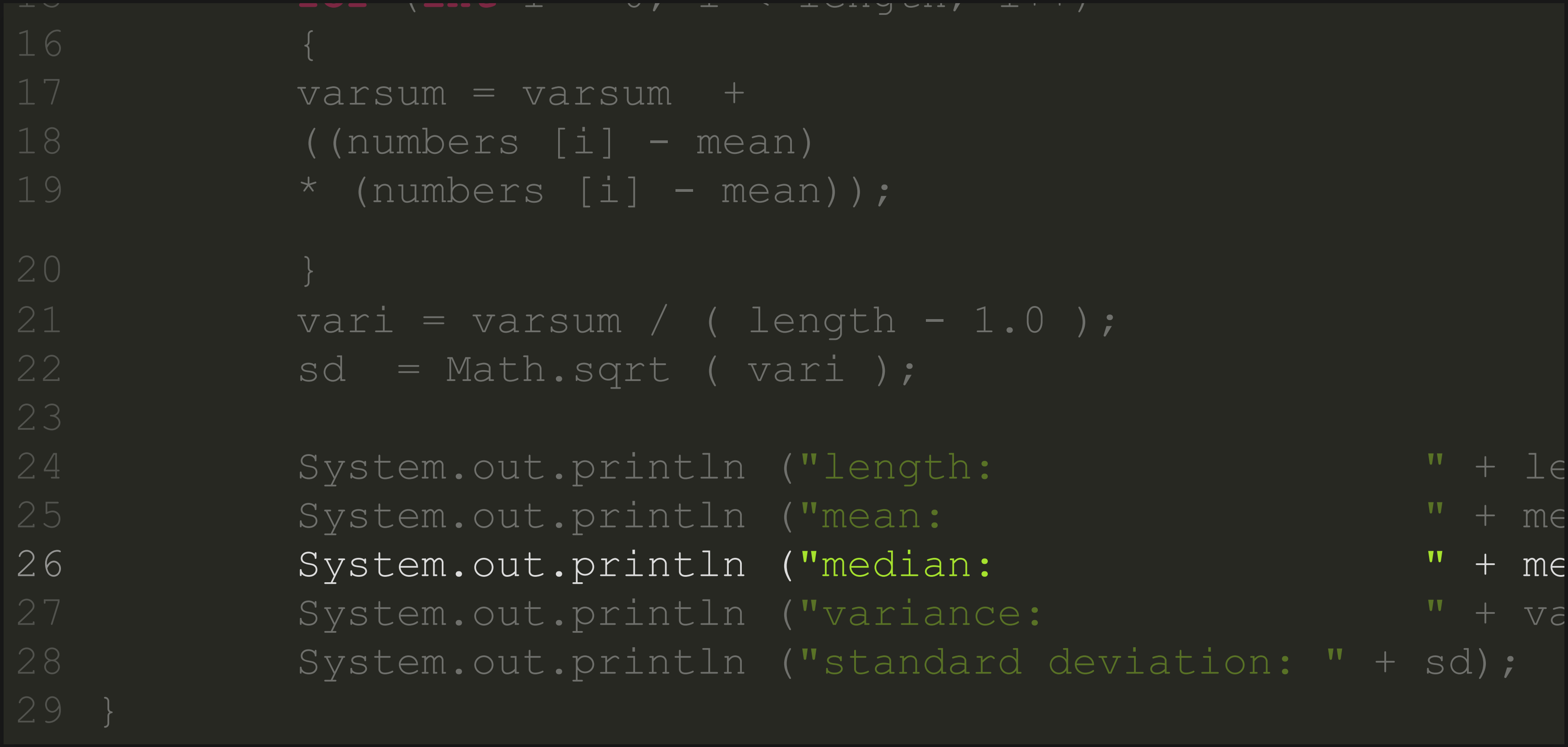
```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        * (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```

USES



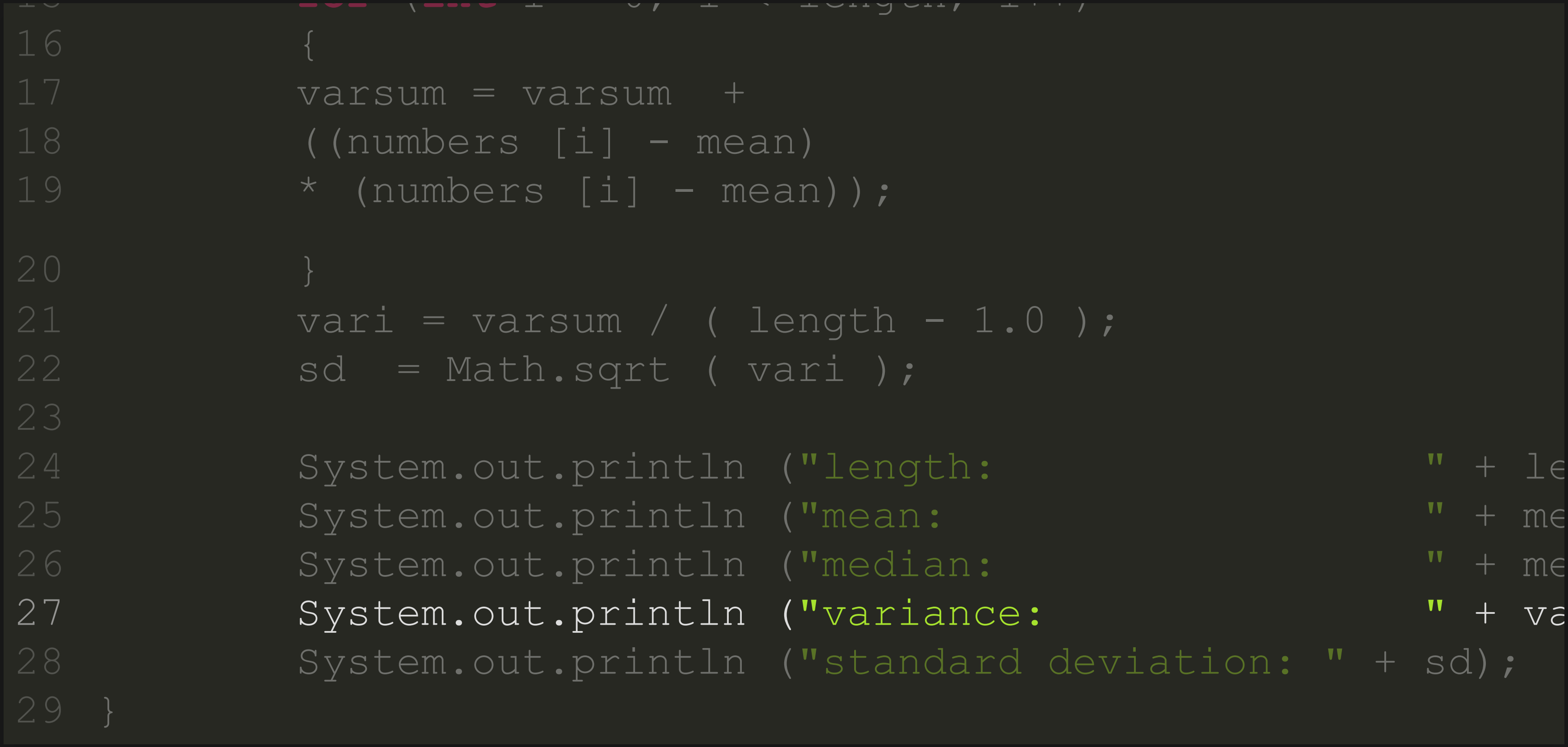
```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        * (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```

USES



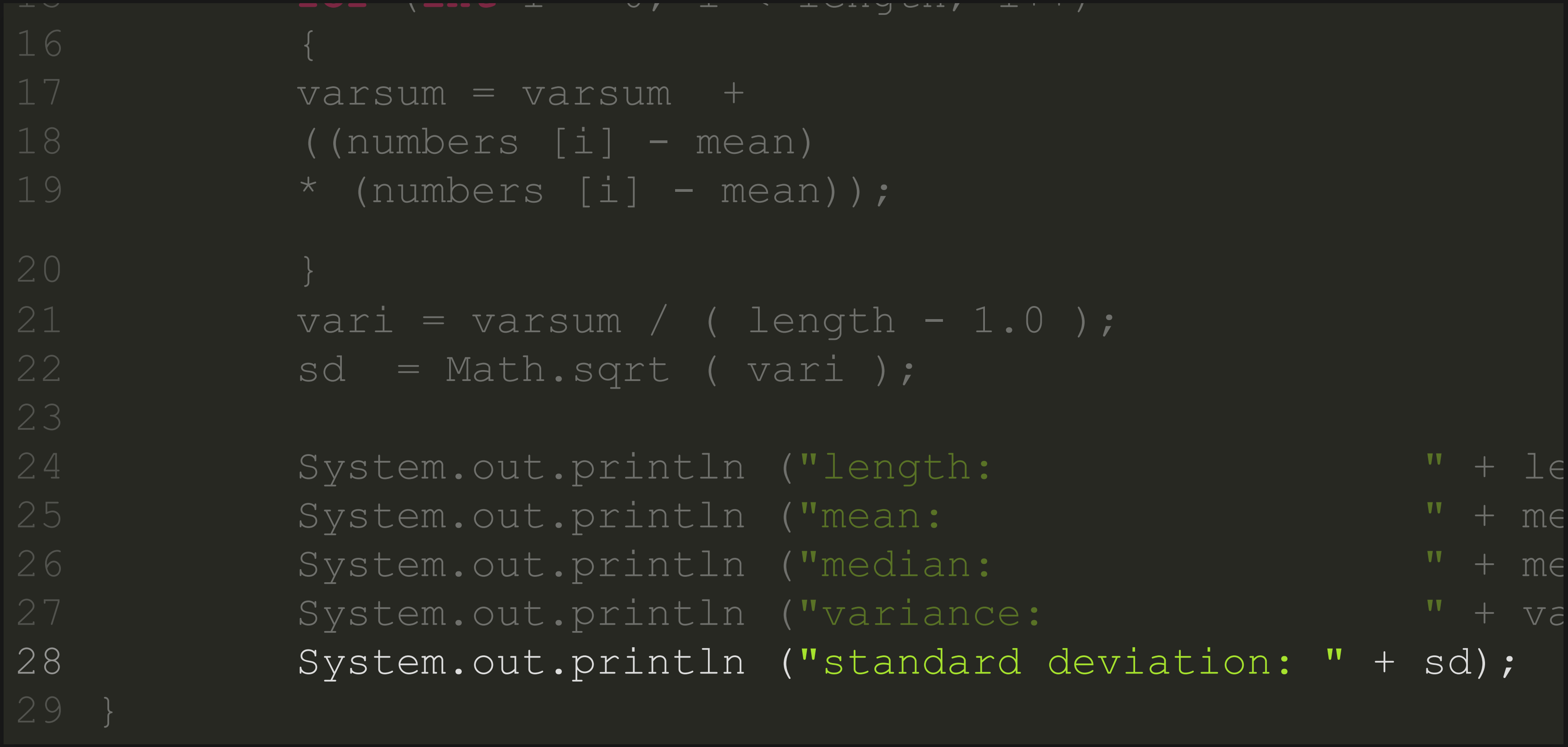
```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        *  (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```

USES

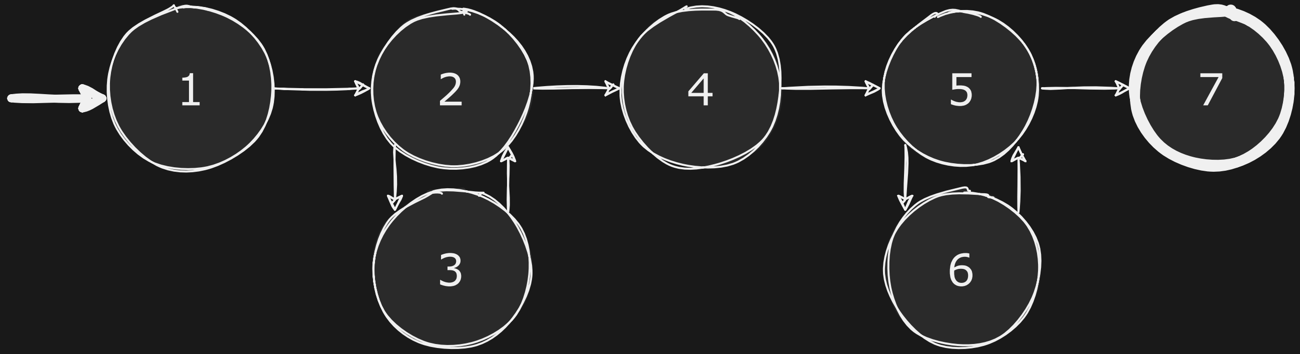


```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        *  (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```

USES



```
        for (int i = 0; i < length; i++)
        {
        varsum = varsum  +
        ((numbers [i] - mean)
        *  (numbers [i] - mean));

        }
        vari = varsum / ( length - 1.0 );
        sd  = Math.sqrt ( vari );

        System.out.println ("length:                 " + le
        System.out.println ("mean:                   " + me
        System.out.println ("median:                 " + me
        System.out.println ("variance:               " + va
        System.out.println ("standard deviation: " + sd);
}
```



def(1) = { numbers, sum, length, i }
use(2) = { i, length }
def(3) = { sum, i }
use(3) = { sum, numbers, i }
def(4) = { mean, med, varsum, i }
use(4) = { numbers, length, sum }
use(5) = { i, length }
def(6) = { varsum, i }
use(6) = { varsum, numbers, i, mean }
def(7) = {vari, sd }
use(7) = { varsum, length, mean, med, vari, sd }

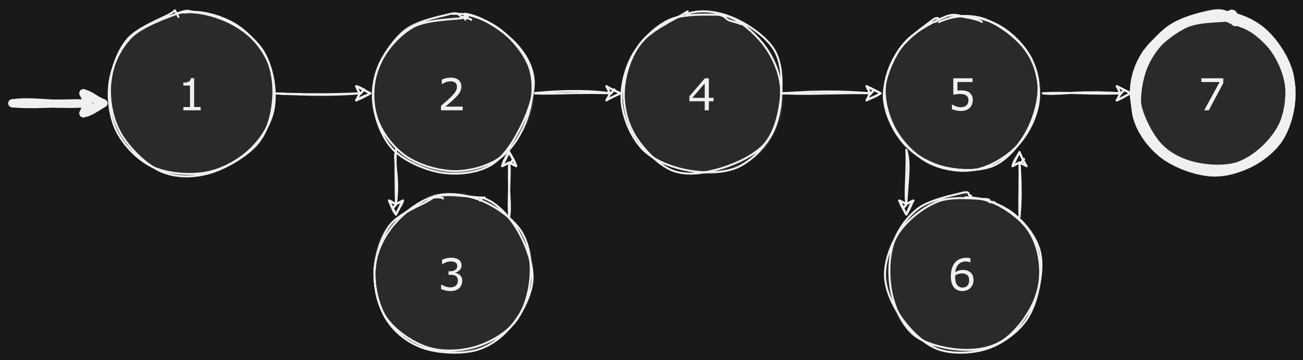
Variable	DU Pairs
numbers	(1 3), (1 4), (1 6)
length	(1 2), (1 4), (1 5)
med	(4 7)
vari	(7 7)
sd	(7 7)
mean	(4 6) (4 7)
sum	(1 3) (1 4) (3 3) (3 4)
varsum	(4 6) (4 7) (6 6) (6 7)
i	(1 2) (1 3) (1 4) (1 5) (1 6) (3 3) (3 2) (3 4) (3 5) (3 6) (6 6) (6 5)

Variable	DU Pairs
numbers	(1 3), (1 4), (1 6)
length	(1 2), (1 4), (1 5)
med	(4 7)
vari	(7 7)
sd	(7 7)
mean	(4 6) (4 7)
sum	(1 3) (1 4) (3 3) (3 4)
varsum	(4 6) (4 7) (6 6) (6 7)
i	(1 2) (1 3) (1 4) (1 5) (1 6) (3 3) (3 2) (3 4) (3 5) (3 6) (6 6) (6 5)

Variable	DU Pairs
numbers	(1 3), (1 4), (1 6)
length	(1 2), (1 4), (1 5)
med	(4 7)
vari	(7 7)
sd	(7 7)
mean	(4 6) (4 7)
sum	(1 3) (1 4) (3 3) (3 4)
varsum	(4 6) (4 7) (6 6) (6 7)
i	(1 2) (1 3) (1 4) (1 5) (1 6) (3 3) (3 2) (3 4) (3 5) (3 6) (6 6) (6 5)

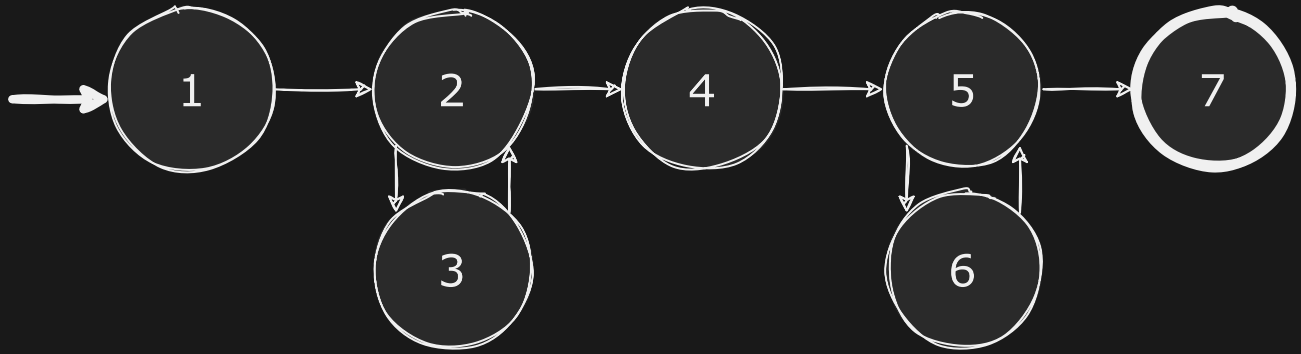
Watch out for scope.

DU PATHS FOR 'NUMBERS'



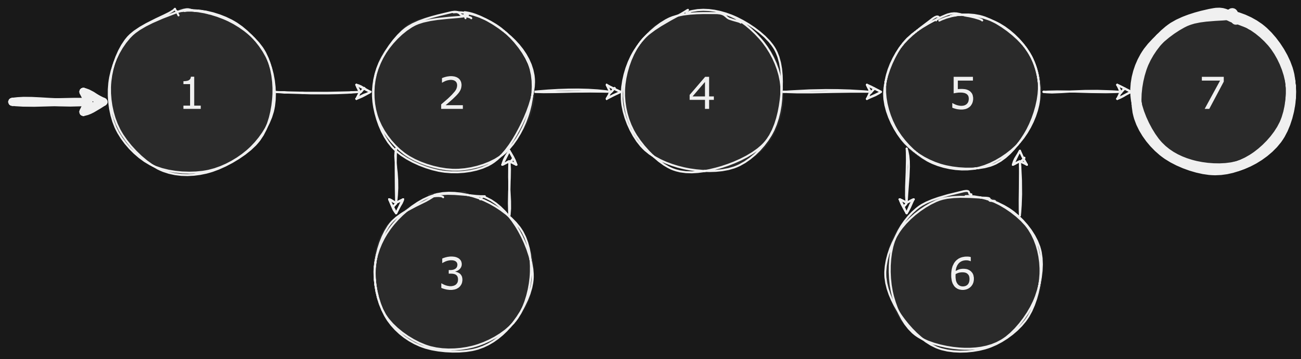
DU Pairs	DU Paths
(1 3)	[1 2 3]
(1 4)	[1 2 4]
(1 6)	[1 2 4 5 6]

DU PATHS FOR 'LENGTH'



DU Pairs	DU Paths
(1 2)	[1 2]
(1 4)	[1 2 4]
(1 5)	[1 2 4 5]

DU PATHS FOR 'LENGTH'



DU Pairs	DU Paths
(1 2)	[1 2]
(1 4)	[1 2 4]
(1 5)	[1 2 4 5]

To tour these paths, we must skip the for loops.

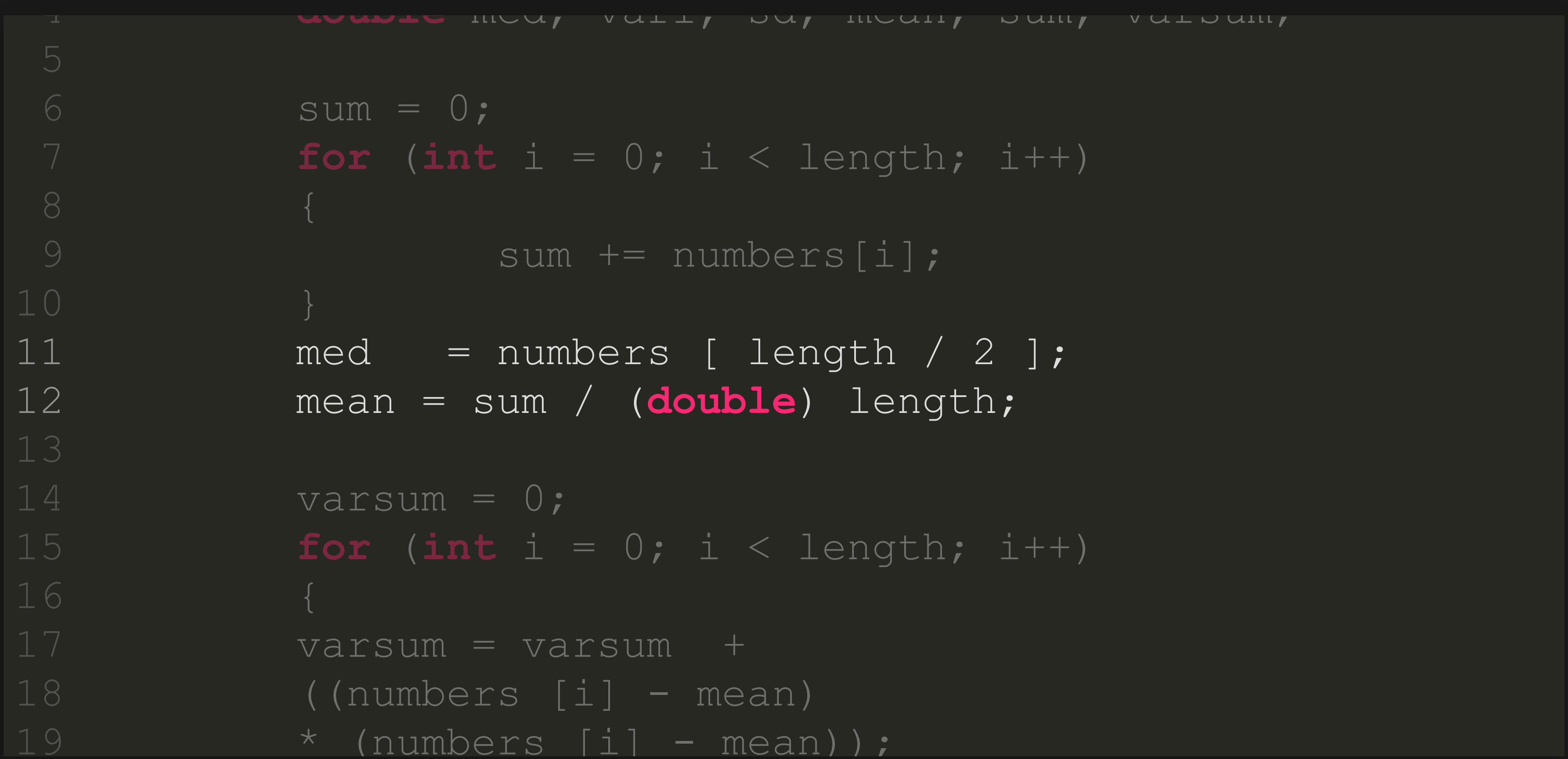
SKIPPING THE FOR LOOPS...

```java
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

SKIPPING THE FOR LOOPS...

```
public static void computeStats (int [ ] numbers)
{
        int length = numbers.length;
        double med, vari, sd, mean, sum, varsum;

        sum = 0;
        for (int i = 0; i < length; i++)
        {
                sum += numbers[i];
        }
        med   = numbers [ length / 2 ];
        mean = sum / (double) length;

        varsum = 0;
        for (int i = 0; i < length; i++)
```

SKIPPING THE FOR LOOPS...



```
double med, vari, sd, mean, sum, varsum;

sum = 0;
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med   = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
*  (numbers [i] - mean));
```

SKIPPING THE FOR LOOPS...

```
sum = 0;
for (int i = 0; i < length; i++)
{
        sum += numbers[i];
}
med    = numbers [ length / 2 ];
mean = sum / (double) length;

varsum = 0;
for (int i = 0; i < length; i++)
{
varsum = varsum  +
((numbers [i] - mean)
*  (numbers [i] - mean));
```

FINITE STATE MACHINES

That's almost it for today, but...

UP NEXT...

That concludes graph coverage! Next week:

- Logic
- Lots of logic.