Input space partition

Each choice coverage:

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(-1,-1,-1));

*assertEquals*(4,trityp.*Triang*(0,0,0));

*assertEquals*(4,trityp.*Triang*(1,1,1));

}

Pair-wise coverage

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(-1,-1,-1));

*assertEquals*(4,trityp.*Triang*(-1,0,0));

*assertEquals*(4,trityp.*Triang*(-1,1,1));

*assertEquals*(4,trityp.*Triang*(0,0,-1));

*assertEquals*(4,trityp.*Triang*(0,1,0));

*assertEquals*(4,trityp.*Triang*(0,-1,1));

*assertEquals*(4,trityp.*Triang*(1,1,-1));

*assertEquals*(4,trityp.*Triang*(1,0,1));

*assertEquals*(4,trityp.*Triang*(1,-1,0));

}

Graph coverage

Node coverage

@Test

**public** **void** testTriang\_1(){

*assertEquals*(1,trityp.*Triang*(2,3,4));

}

@Test

**public** **void** testTriang\_2(){

*assertEquals*(2,trityp.*Triang*(2,2,3));

*assertEquals*(2,trityp.*Triang*(2,3,2));

*assertEquals*(2,trityp.*Triang*(3,2,2));

}

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(0,0,0));

*assertEquals*(4,trityp.*Triang*(1,2,3));

*assertEquals*(4,trityp.*Triang*(2,2,5));

}

Edge coverage

@Test

**public** **void** testTriang\_1(){

*assertEquals*(1,trityp.*Triang*(2,3,4));

}

@Test

**public** **void** testTriang\_2(){

*assertEquals*(2,trityp.*Triang*(2,2,3));

*assertEquals*(2,trityp.*Triang*(2,3,2));

*assertEquals*(2,trityp.*Triang*(3,2,2));

}

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(0,0,0));

*assertEquals*(4,trityp.*Triang*(1,2,3));

*assertEquals*(4,trityp.*Triang*(2,2,5));

}

Prime path coverage

@Test

**public** **void** testTriang\_1(){

*assertEquals*(1,trityp.*Triang*(2,3,4));

}

@Test

**public** **void** testTriang\_2(){

*assertEquals*(2,trityp.*Triang*(2,2,3));

*assertEquals*(2,trityp.*Triang*(2,3,2));

*assertEquals*(2,trityp.*Triang*(3,2,2));

}

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(0,0,0));

*assertEquals*(4,trityp.*Triang*(1,2,3));

*assertEquals*(4,trityp.*Triang*(2,2,5));

*assertEquals*(4,trityp.*Triang*(2,5,2));

*assertEquals*(4,trityp.*Triang*(5,2,2));

}

All-use coverage

@Test

**public** **void** testTriang\_1(){

*assertEquals*(1,trityp.*Triang*(2,3,4));

}

@Test

**public** **void** testTriang\_2(){

*assertEquals*(2,trityp.*Triang*(2,2,3));

*assertEquals*(2,trityp.*Triang*(2,3,2));

*assertEquals*(2,trityp.*Triang*(3,2,2));

}

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(0,0,0));

*assertEquals*(4,trityp.*Triang*(1,2,3));

*assertEquals*(4,trityp.*Triang*(2,2,5));

*assertEquals*(4,trityp.*Triang*(2,5,2));

*assertEquals*(4,trityp.*Triang*(5,2,2));

}

Logic expression

**if** (Side1 <= 0 || Side2 <= 0 || Side3 <= 0)

{

triOut = 4;

**return** (triOut);

}

Predicate coverage

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(0,1,1));

}

Clause coverage

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(0,0,0));

}

Correlated active clause coverage

@Test

**public** **void** testTriang\_3(){

*assertEquals*(3,trityp.*Triang*(1,1,1));

}

@Test

**public** **void** testTriang\_4(){

*assertEquals*(4,trityp.*Triang*(1,0,1));

*assertEquals*(4,trityp.*Triang*(0,1,1));

*assertEquals*(4,trityp.*Triang*(1,1,0));

}

Mutation testing

*assertEquals*(4,TritypMutantOne.*Triang*(1,1,2));

*assertEquals*(4,TritypMutantTwo.*Triang*(1,1,2));