```
public static class ExtraMath
{
    public static IEnumerable<IEnumerable<T>> GeneralizedTartaglia<T>(T seed,
         Func<T,T,T> generator)
    {
        var dict = new Dictionary<int, List<T>>();
        var exceptions = new List<Exception>();
        var line = 0;
        while (true)
        {
            dict.Add(line, new List<T>());
            for (var i = 0; i <= line; i++)
            {
                if (i == line || i == 0) dict[line].Add(seed);
                else
                {
                    var aux = dict[line - 1].ToArray();
                    try { dict[line].Add(generator(aux[i - 1], aux[i])); }
                    catch (Exception ex) { exceptions.Add(ex); }
                }
            }
            if (exceptions.Count > 0)
               throw new AggregateException(
                    "Errori multipli nella creazione di una riga",
                    exceptions);
            line++;
            yield return dict[line-1];
        }

    }
}

public class ExtraMathTest
{
    public class MyException : Exception
    {
        private static int _count;
        public int Index { get; } = ++_count;
        public MyException() { }
        public MyException(string message) : base(message) { }
        public MyException(string message, Exception innerException) :
         base(message, innerException) { }
    }

    [Test]
    public void NormalBehaviour()
    {
        var add = (int a, int b) => a + b;
        var ris = new Dictionary<int, IEnumerable<int>>();
        int i = 0;
        foreach(var line in ExtraMath.GeneralizedTartaglia(1, add).Take(5))
        {
            ris.Add(i, line);
            i++;
        }
        var expectedRis = new Dictionary<int, IEnumerable<int>>()
        {
            { 0, new[] { 1 } },
            { 1, new[] { 1,1 } },
            { 2, new[] { 1,2,1 } },
            { 3, new[] { 1,3,3,1 } },
            { 4, new[] { 1,4,6,4,1 } },
        };
        Assert.That(ris, Is.EqualTo(expectedRis));
```

```
    }

    [Test]
    public void TestException()
    {
        var concat = (string a, string b) =>
         a.Length != 4 && b.Length != 4 ? a + b : throw new MyException("ciao");
        try
        {
            ExtraMath.GeneralizedTartaglia("x", concat);
        }
        catch(AggregateException ex)
        {
            var index = new List<int>();
            foreach(MyException exp in ex.InnerExceptions)
            {
                index.Add(exp.Index);
            }
            Assert.That(index, Is.EqualTo(new[] {1,2,3,4}));
        }
    }
    [TestCase(5)]
    [TestCase(2)]
    [TestCase(0)]
    public void TestCount(int lineNumber)
    {
        if (lineNumber <= 0) Assert.Inconclusive();

        int isCalled = 0;
        var add = (int a, int b) => a + b;
        var triangle = ExtraMath.GeneralizedTartaglia(1, add).Take(lineNumber);
        foreach(var line in triangle)
        {
            var lenght = line.ToArray().Length;
            //nella prima e nella seconda riga non ci sono somme ma la seconda si
               annulla con il -2
            if (!(lenght == 1) ) isCalled += lenght - 2;
        }

        Assert.That(isCalled, Is.EqualTo(((lineNumber-1)*(lineNumber-2))/2));
    }
}
```