16-10-2020

Rapport

Obligatorisk opgave C-verden

Louise Helene Hamle

Indholdsfortegnelse

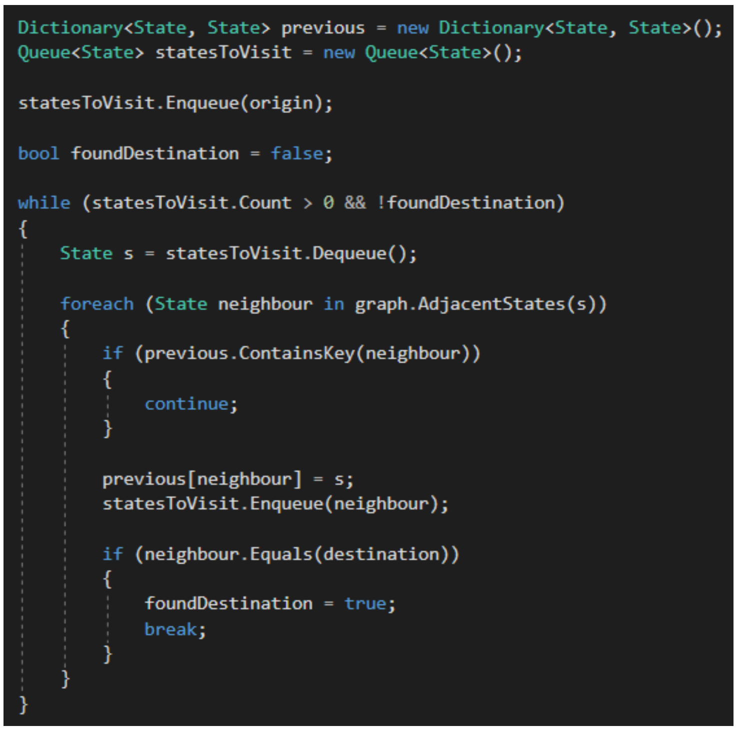
Introduktion

Koden til opgaverne findes i filen LHH_ObligatoriskOpgave.sln, hvor der er oprettet et konsol projekt for hver opgave, så det burde være nemt og overskueligt at finde. Under afsnittet Bilag, er der desuden rigtig mange billeder af kodestykker fra alle opgaverne, både klasser og metoder.

Opgave 1 – Grafer og grafgennemløb

Løsningen er bygget op fra bunden, hvor der er implementeret en graf baseret på adjacency list strukturen, da denne struktur er mere effektiv, end edge list strukturen, og mere simpel at implementere, end adjacency map og adjacency matrix strukturerne. Dertil er Breadth-First-Search-algoritmen implementeret til at finde én sti fra en stat til en anden, fordi den algoritme altid finder den korteste sti. Opgaven kunne også løses med en implementation af Depth-First-Search-algoritmen, men den finder ikke nødvendigvis den korteste sti. Denne løsning undersøger, om der er en sti mellem de to stater, og hvis der er, skrives stien ud.

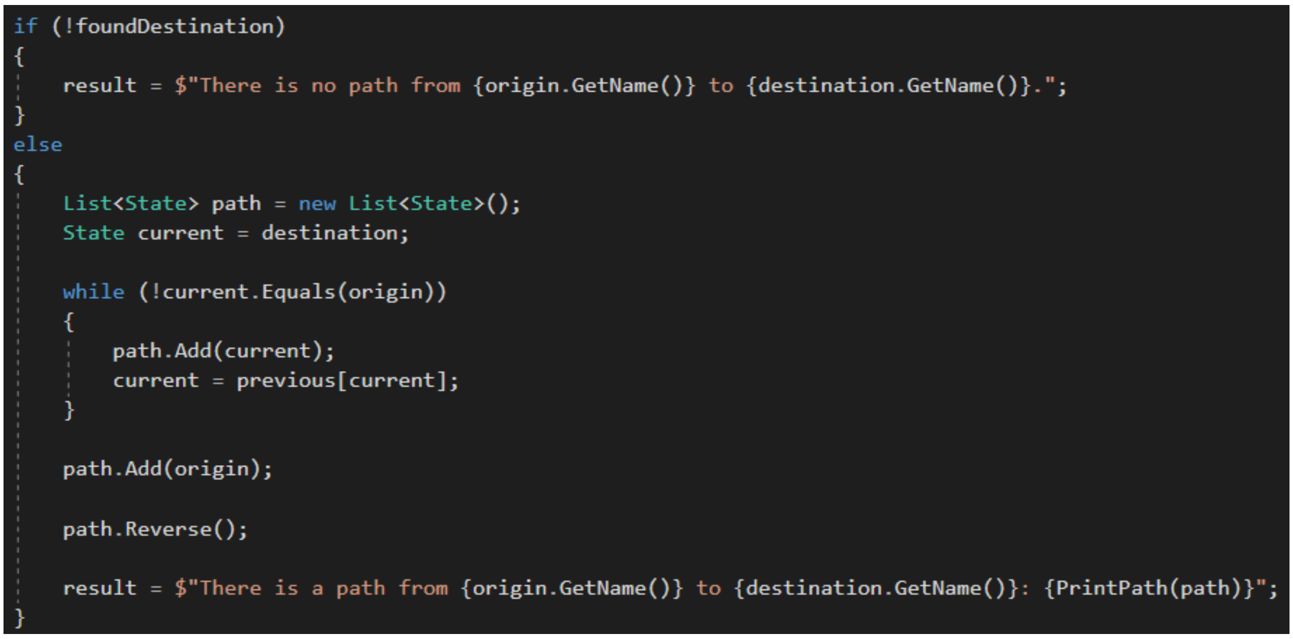
Figur 1.1



```
Dictionary<State, State> previous = new Dictionary<State, State>();
Queue<State> statesToVisit = new Queue<State>();

statesToVisit.Enqueue(origin);

bool foundDestination = false;

while (statesToVisit.Count > 0 && !foundDestination)
{
    State s = statesToVisit.Dequeue();

    foreach (State neighbour in graph.AdjacentStates(s))
    {
        if (previous.ContainsKey(neighbour))
        {
            continue;
        }

        previous[neighbour] = s;
        statesToVisit.Enqueue(neighbour);

        if (neighbour.Equals(destination))
        {
            foundDestination = true;
            break;
        }
    }
}
```

Figur 1.1 viser undersøgelsen om hvorvidt der er en sti mellem to stater. Her er en Dictionary, previous, benyttet til at holde styr på, hvilke stater algoritmen følger, fra start til slut, ved at ”mappe” hver stat på stien til den forrige stat på stien.

Figur 1.2

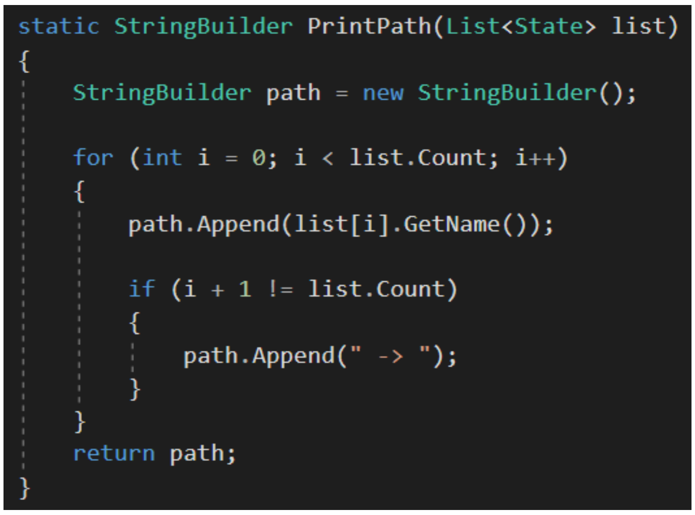


```
if (!foundDestination)
{
    result = $"There is no path from {origin.GetName()} to {destination.GetName()}.";
}
else
{
    List<State> path = new List<State>();
    State current = destination;

    while (!current.Equals(origin))
    {
        path.Add(current);
        current = previous[current];
    }

    path.Add(origin);

    path.Reverse();

    result = $"There is a path from {origin.GetName()} to {destination.GetName()}: {PrintPath(path)}";
}
```

Figur 1.2 viser, hvordan previous kan bruges til at printe stien ud. Først tilbagespores stien i previous fra slut til start, hvor staterne kopieres over i en liste en for en. Derefter bliver listen vendt ved at kalde metoden Reverse() på listen, så stien bliver vendt rigtigt. Til sidst printes stien ud til konsollen ved hjælp af hjælpemetoden PrintPath().

Hjælpemetoden, som kan ses af figur 1.3, startede med at være implementeret med string concatenation til at lave stien, men blev senere ændret til at implementere en StringBuilder i stedet, da den er mere effektiv. Det kan dog diskuteres, om det er nødvendigt i lige præcis dette program, da det ikke er særligt mange string concatenations der bliver udført i programmets nuværende form.

Figur 1.3



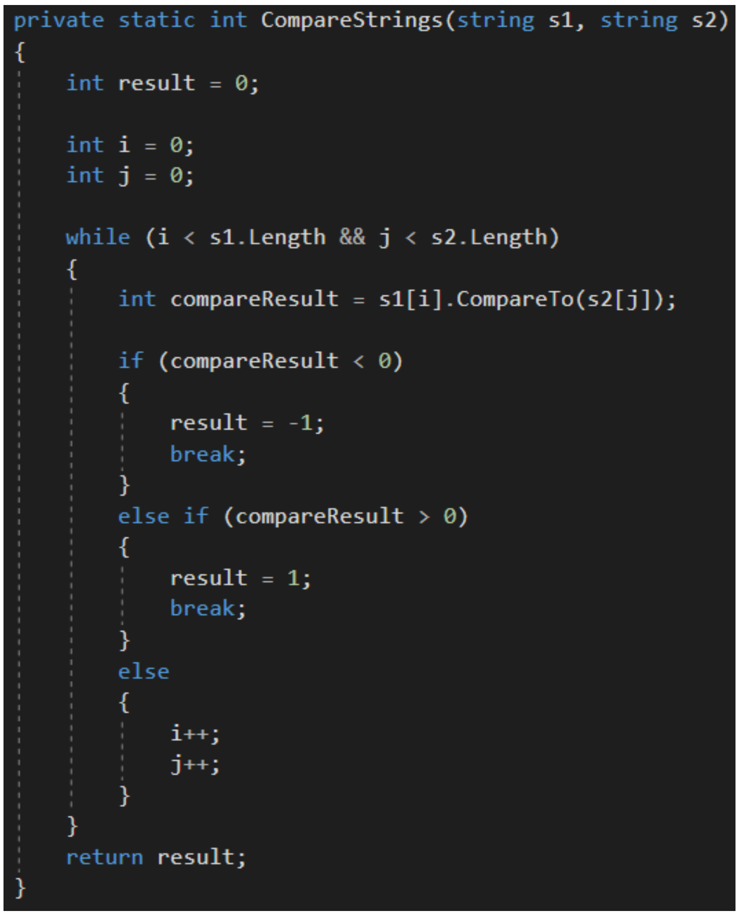
```
static StringBuilder PrintPath(List<State> list)
{
    StringBuilder path = new StringBuilder();

    for (int i = 0; i < list.Count; i++)
    {
        path.Append(list[i].GetName());

        if (i + 1 != list.Count)
        {
            path.Append(" -> ");
        }
    }
    return path;
}
```

Opgave 2 – Sorteringsalgoritmer og datastrukturer med strenge

Denne opgave er løst med tre forskellige datastrukturer, hhv. List, StringCollection og Array. Dertil er der implementeret to sorteringsalgoritmer, BubbleSort og en meget hurtigere sorteringsalgoritme, MergeSort.
Figurerne 2.5 – 2.23 under Bilag, viser kode for diverse metoder og algoritmer samt testdata og output af tests. Der kan lige knyttes et par kommentarer til et par af de mere interessante metoder og algoritmer.

Bemærk figur 2.6, som viser compare metoden, der sammenligner to strenge.

Figur 2.6

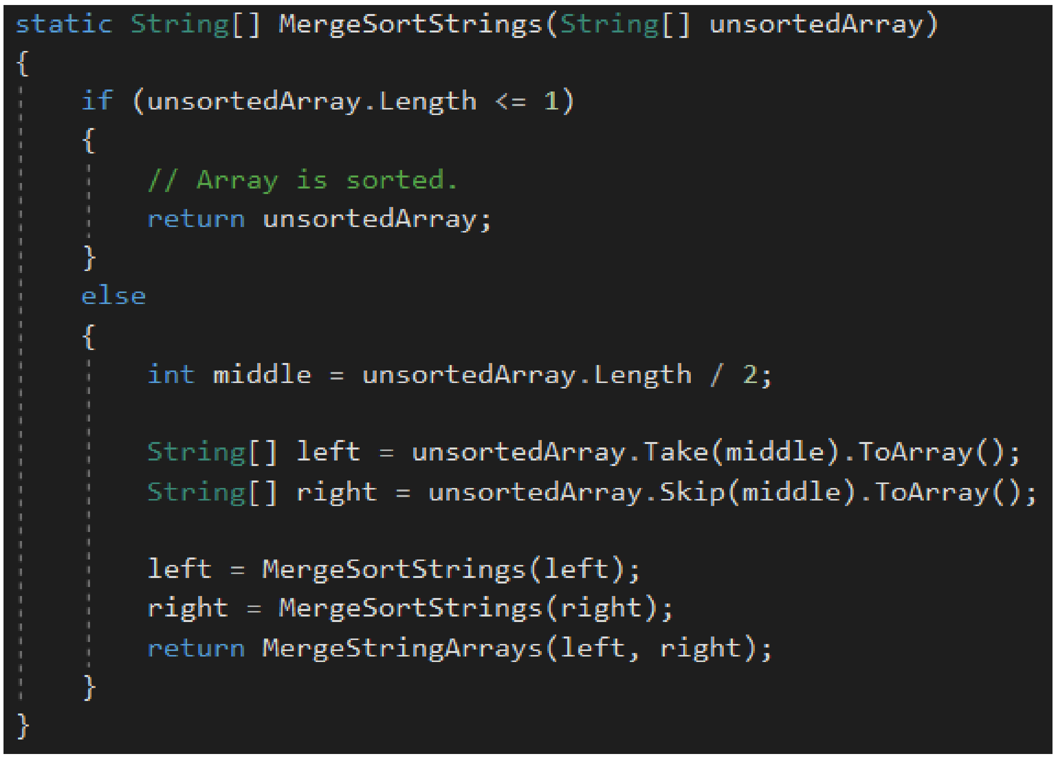


```
private static int CompareStrings(string s1, string s2)
{
    int result = 0;

    int i = 0;
    int j = 0;

    while (i < s1.Length && j < s2.Length)
    {
        int compareResult = s1[i].CompareTo(s2[j]);

        if (compareResult < 0)
        {
            result = -1;
            break;
        }
        else if (compareResult > 0)
        {
            result = 1;
            break;
        }
        else
        {
            i++;
            j++;
        }
    }
    return result;
}
```

Her er sammenligningen baseret på de enkelte chars i strengen, som sammenlignes efter den indbyggede CompareTo metode for chars. Dertil skal det også noteres, at resultatet af char sammenligningen gemmes i en variabel, hvorefter compare metoden fortsætter sammenligningen baseret på den variabel. Det er implementeret sådan for at optimere koden, da sammenligningen mellem de to chars ellers kan risikere at skulle køres to gange for den samme sammenligning.

Bemærk også figurerne 2.9 og 2.10, som viser implementationen af MergeSort for Array datastrukturen med streng elementer.

Figur 2.9



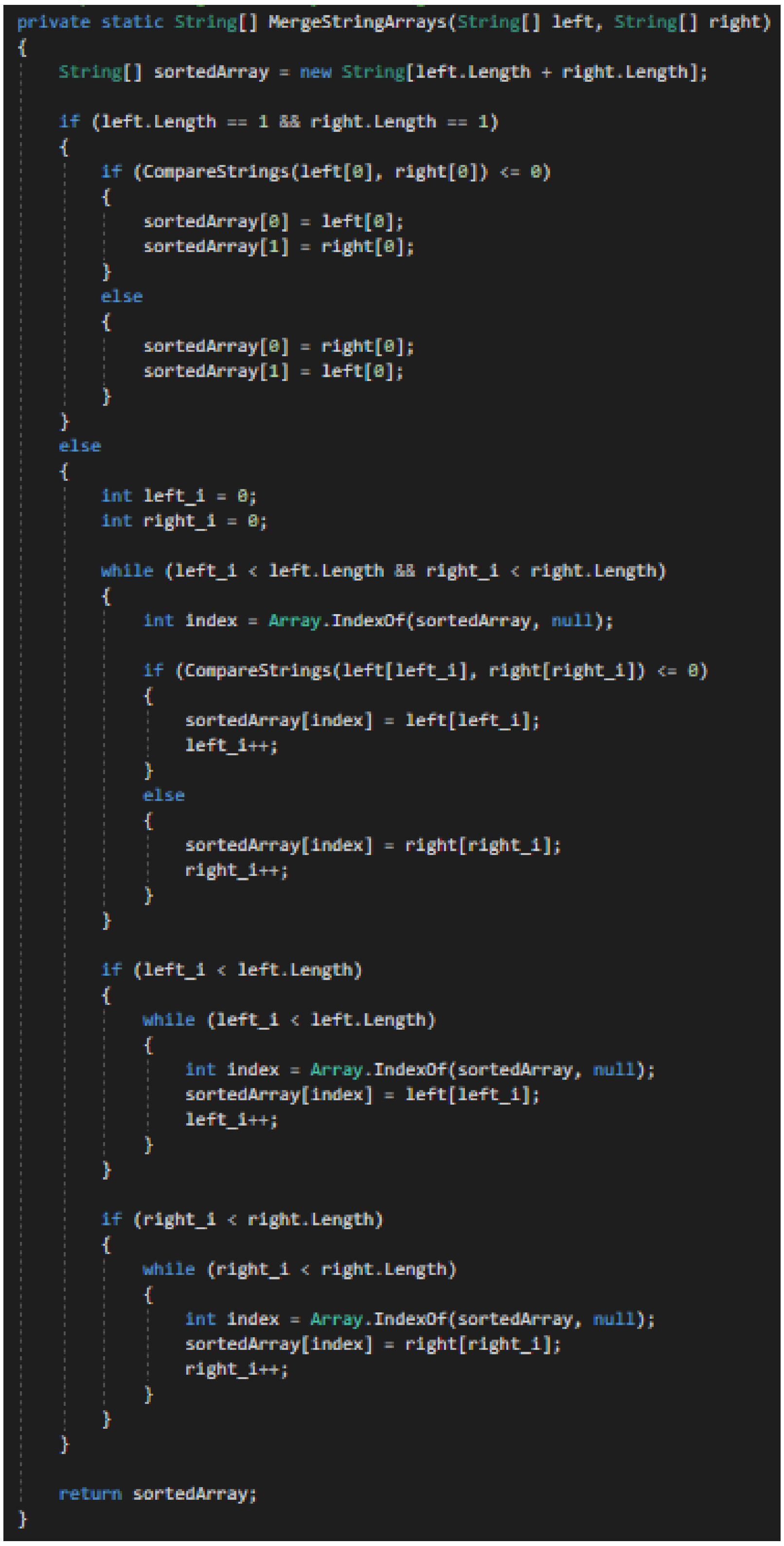
```
static String[] MergeSortStrings(String[] unsortedArray)
{
    if (unsortedArray.Length <= 1)
    {
        // Array is sorted.
        return unsortedArray;
    }
    else
    {
        int middle = unsortedArray.Length / 2;

        String[] left = unsortedArray.Take(middle).ToArray();
        String[] right = unsortedArray.Skip(middle).ToArray();

        left = MergeSortStrings(left);
        right = MergeSortStrings(right);
        return MergeStringArrays(left, right);
    }
}
```

Metoden er rekursiv, og termineringsreglen er her, at hvis et array er tomt eller kun indeholder et element, så er det sorteret.
Er der flere, end et element i arrayet, beregnes først det midterste index i arrayet. Derefter benyttes et par smarte metodebaserede linq queries til at putte hhv. den første halvdel og den sidste halvdel af elementerne i hvert deres array, således, at det originale array bliver delt i to.

Metoden Take returnerer elementerne fra starten af arrayet og frem til det antal elementer der bliver angivet i argumentet til metoden. Metoden Skip springer det antal elementer over, som bliver givet som argument til metoden, og returnerer resten af elementerne i arrayet. Metoderne kan kaldes på alle datastrukturer, som implementerer IEnumerable<T> interfacet, som f.eks. Array klassen.

Derefter kaldes metoden rekursivt på hh. det ene og det andet array, så de bliver sorterede, og til sidst returneres resultatet af et kald til hjælpemetoden med de to arrays som argumenter, som fletter de to sorterede arrays sammen og returnerer det sorterede array.

Figur 2.10



```
private static String[] MergeStringArrays(String[] left, String[] right)
{
    String[] sortedArray = new String[left.Length + right.Length];

    if (left.Length == 1 && right.Length == 1)
    {
        if (CompareStrings(left[0], right[0]) <= 0)
        {
            sortedArray[0] = left[0];
            sortedArray[1] = right[0];
        }
        else
        {
            sortedArray[0] = right[0];
            sortedArray[1] = left[0];
        }
    }
    else
    {
        int left_i = 0;
        int right_i = 0;

        while (left_i < left.Length && right_i < right.Length)
        {
            int index = Array.IndexOf(sortedArray, null);

            if (CompareStrings(left[left_i], right[right_i]) <= 0)
            {
                sortedArray[index] = left[left_i];
                left_i++;
            }
            else
            {
                sortedArray[index] = right[right_i];
                right_i++;
            }
        }

        if (left_i < left.Length)
        {
            while (left_i < left.Length)
            {
                int index = Array.IndexOf(sortedArray, null);
                sortedArray[index] = left[left_i];
                left_i++;
            }
        }

        if (right_i < right.Length)
        {
            while (right_i < right.Length)
            {
                int index = Array.IndexOf(sortedArray, null);
                sortedArray[index] = right[right_i];
                right_i++;
            }
        }
    }

    return sortedArray;
}
```

Denne metode tager to argumenter af typen array. Her starter metoden med at oprette et nyt array med længden af de to argument arrays’ summerede længder. Derefter undersøges det, om begge

argument arraysene kun har ét element, og hvis de har, sammenlignes de to elementer og bliver kopieret over i det nye array i den korrekte rækkefølge, hvorefter metoden afsluttes og det sorterede array returneres.
Er der flere, end ét element i bare ét af de to arrays, bliver der oprettet to index variable, en til hver af de to arrays. Derefter køres en while løkke, så længe begge index variable er mindre end længden på hver af arraysene. I while løkken oprettes først en ny index variabel, som holder den næste tomme position i det nye array. Derefter sammenlignes de to strenge som er på de angivne index positioner i hver af arraysene. Den streng, der har højest prioritet bliver så kopieret over i det nye array, og index variablen til det array, som strengen kom fra inkrementeres med en.
Når en af index variablene får en værdi der ikke længere er mindre, end længden på det tilhørende array, går metoden videre og samler en eventuel ’hale’ op, ved at undersøge, om der stadig skulle være elementer i et af arraysene, og hvis der er, bliver der i en ny while løkke tilføjet en kopi af hver af de manglende elementer til det nye array. Til sidst returneres det sorterede array.

En lille kommentar til Main metodens tests, som kan ses af figurerne 2.11 – 2.21, skal da også noteres; de mange test variable sikkert godt kunne få en bedre navngivning.

Derudover er der kørt lidt statistik, som ses i de fire statistik skemaer, vist i figurerne 2.1 – 2.4, som viser statistik over køretider for sorteringsalgoritmerne til hver af de tre datastrukturer, samt køretider for søgning i de tre datastrukturer. OBS! Disse skemaer nåede desværre ikke at blive færdige, og de tests som ikke blev færdige, er i skemaerne markeret med rød baggrund. Den værdi der står i felterne med rød baggrund, er enten en initial værdi(der er ikke kørt en eneste test endnu) eller resultatet af få test kørsler.

Figur 2.1: Sortering af strenge i en List<String> datastruktur med forskellige antal strenge.

Antal strenge i en List<String>	**BubbleSort**	**MergeSort**	**Sort**
10.000	00 min. 04,35 – 04,68 sec.	00 min. 00,03 – 00,05 sec.	00 min. 00,10 – 00,11 sec.
20.000	00 min. 14,18 – 18,88 sec.	00 min. 00,03 – 00,05 sec.	00 min. 00,10 – 00,16 sec.
30.000	00 min. 34,29 – 35,66 sec.	00 min. 00,04 – 00,05 sec.	00 min. 00,15 – 00,17 sec.
40.000	01 min. 09,13 – 11,05 sec.	00 min. 00,07 – 00,10 sec.	00 min. 00,22 – 00,29 sec.
50.000	01 min. 54,83 – 59,90 sec.	00 min. 00,09 – 00,14 sec.	00 min. 00,28 – 0,39 sec.
100.000	13 min. 30,16 sec.	01 min. 22,00 – 59,00 sec.	02 min. 00,00 – 59,00 sec.

Figur 2.2: Sortering af strenge i en StringCollection datastruktur med forskellige antal strenge.

Antal strenge i en StringCollection	**BubbleSort**	**MergeSort**
10.000	00 min. 09,47 – 09,81 sec.	00 min. 00,08 – 00,09 sec.
20.000	00 min. 16,30 – 21,78 sec.	00 min. 00,03 – 00,04 sec.
30.000	00 min.37.96 – 40,21 sec.	00 min. 00,05 – 00,06 sec.
40.000	01 min. 17,47 – 90,80 sec.	00 min. 00,09 – 00,14 sec.
50.000	02 min. 07,15 – 20,25 sec.	00 min. 00,11 – 00,17 sec.
100.000	28 min. 28,19 sec.	00 min. 00,90 – 00,99 sec.

Figur 2.3: Sortering af strenge i en Array datastruktur med forskellige antal strenge.

Antal strenge i et array	**BubbleSort**	**MergeSort**	**Array.Sort**
10.000	00 min. 03,56 – 04,51 sec.	00 min. 00,90 – 00,99 sec.	00 min. 00,03 – 00,06 sec.
20.000	00 min.12,24 – 15,82 sec.	00 min. 00,38 – 00,46 sec.	00 min. 00,06 – 00,09 sec.
30.000	00 min. 29,02 – 30,21 sec.	00 min. 00,83 – 00,86 sec.	00 min. 00,10 – 00,12 sec.
40.000	00 min. 58,70 – 95,80 sec.	00 min. 01,52 – 01,76 sec.	00 min. 00,14 – 00,24 sec.
50.000	01 min. 39,11 – 95,65 sec.	00 min. 02,38 – 03,52 sec.	00 min. 00,19 – 00,29 sec.
100.000	12 min. 18,04 sec.	00 min. 20,00 – 40,00 sec.	00 min. 00,58 sec.

Skemaerne med statistik over sortering i hhv. et array og en list indeholder også køretider for

sortering af array og list med klassernes indbyggede sorteringsalgoritmer. Det vakte interesse at undersøge, om disse indbyggede sorteringsalgoritmer var de hurtigste til at sortere deres respektive datastruktur. Her viste det sig, at det ikke helt var tilfældet, da MergeSort ser ud til at være hurtigere til at sortere en liste, end list.Sort metoden. For array strukturen, var resultatet dog som forventet; Bubblesort er klart den langsomste, mens Array.Sort er den hurtigste af disse tre algoritmer til at sortere et array. Overordnet set, har det vist sig at være MergeSort, der oftest er den hurtigste af disse algoritmer, mens BubbleSort i alle tilfælde er den langsomste. Datastrukturen StringCollection, som er en specialiseret kollektion lavet specielt til at vedligeholde strenge, kunne tænkes at levere høje ydelser, men det viste sig ikke at være tilfældet.
De tre datastrukturer fordeler sig relativt klart med hensyn til datastrukturernes effektivitet, hvor det viser sig at være Array strukturen, som er den, der er hurtigst at sortere, mens StringCollection er den langsomste.

Figur 2.4: Søgning i forskellige datastrukturer med forskelligt antal strenge.

Antal strenge i datastrukturen	List<String>	StringCollection	Array
100.000	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.
200.000	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.
300.000	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.
400.000	00,0051502 – 00,0622248 sec.	00,0030141 – 00,0560306 sec.	00,0012304 – 00,0327321 sec.
600.000	00,0080424 – 00,0798676 sec.	00,0209383 – 00,0897238 sec.	00,0092858 – 00,0476181 sec.
800.000	00,0062061 – 00,0795540 sec.	00,0027690 – 00,0910171 sec.	00,0018783 – 00,0659603 sec.
1.000.000	00,0112372 – 00,0721235 sec.	00,0569319 – 00,1366675 sec.	00,0263200 – 00,0690043 sec.
2.000.000	00,001330342 – 00,1565724 sec.	00,0172298 – 00,1612037 sec.	00,0015577 – 00,1130876 sec.

Det sidste skema indeholder statistik over køretider for søgning i de tre datastrukturer ved brug af deres indbyggede metoder. Her kunne det også have været forventningen, at StringCollection ville brilliere med nogle af de bedste køretider for søgning efter strenge. Resultaterne viser dog endnu engang et andet billede; StringCollection er den af disse datastrukturer, der er langsomst at søge i, mens Array strukturen statistisk set er den hurtigste at søge i.

Baseret på disse testresultater, kan man konkludere, at Array strukturen er den mest effektive af disse datastrukturer, når det kommer til sortering og søgning, mens StringCollection er den mindst effektive. Når der er så meget godt at sige om Array strukturen, skal det også lige bemærkes, at Array strukturen til gengæld er enormt langsom at indsætte elementer i, så opsætningen til disse tests har taget længst tid, når programmet har skullet oprette arrays med store mængder data i.

Opgave 3 – Runtime, Compile-time og garbage collection

Der er lavet to metoder til denne opgave; en, som gemmer alle cirklerne i en liste, samt en, som kun gemmer referencen til det senest oprettede objekt. Disse metoder kan ses af figurerne 3.2 og 3.3 under bilag. Derudover er der et kodeafsnit sidst i programmet til at køre stort set de samme kodelinjer, som i metoderne, direkte i Main metoden. Dette kode kan også ses under bilag, af figurerne 3.4. og 3.5. Cirkel klassen kan ses af figur 3.6 i bilag.

Figur 3.1

Antal cirkler der bliver oprettet.	**Metodekald: MakeListOfCirclesTimed()**	**Loop i Main() med liste.**	**Metodekald: MakeCirclesTimed()**	**Loop i Main() uden liste.**
1 million (10^6)	00,06 – 00,14 sec.	00,06 – 00,12 sec.	00,01 – 00,03 sec.	00,05 – 00,19 sec.
10 million (10^7)	01,10 – 01,29 sec.	01,88 – 02,33 sec.	00,19 – 00,25 sec.	00,22 – 00,44 sec.
100 million (10^8)	11,10 – 13,01 sec.	12,11 – 13,40 sec.	01,94 – 02,25 sec.	02,15 – 02,78 sec.
134,217 million (10^8) 134.217.72**8**	15,64 – 19,69 sec.	16,29 – 18,17 sec.	02,44 – 03,07 sec.	03,00 – 03,06 sec.
134,217 million (10^8) 134.217.72**9**	OutOfMemoryException 18,32 – 21,29 sec.	OutOfMemoryException 19,60 – 21,42 sec.	02,43 – 03,24 sec.	03,00 – 04,74 sec.
1 billion (10^9)	OutOfMemoryException 18,24 – 21,18 sec.	OutOfMemoryException 18,17 - 18,46 sec.	18,21 – 21,60 sec.	24,07 – 25,67 sec.
2,147 billion (10^9) 2.147.483.64**7**	OutOfMemoryException 18,08 – 21,44 sec.	OutOfMemoryException 18,31 – 18,79 sec.	39,06 – 41,85 sec.	52,35 – 59,09 sec.
2,147 billion (10^9) 2.147.483.64**8**	Compiler Error - CS1503: Cannot convert from uint to int.	Compiler Warning - CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.	Compiler Error - CS1503: Cannot convert from uint to int.	Compiler Warning - CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.
4,294 billion (10^9) 4.294.967.296	Compiler Error - CS1503: Cannot convert from long to int.	Compiler Warning- CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.	Compiler Error - CS1503: Cannot convert from long to int.	Compiler Warning - CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.
9,223 quintillion (10^{18}) 9.223.372.036.854.775.808	Compiler Error - CS1503: Cannot convert from ulong to int.	Compiler Error - CS0034: Operator < is ambiguous on operands of type int and ulong.	Compiler Error - CS1503: Cannot convert from ulong to int.	Compiler Error - CS0034: Operator < is ambiguous on operands of type int and ulong.
18,446 quintillion (10^{19}) 18.446.744.073.709.551.616	Compiler Error - CS1021: Integral constant is too large.	Compiler Error - CS1021: Integral constant is too large.	Compiler Error - CS1021: Integral constant is too large.	Compiler Error - CS1021: Integral constant is too large.

Figur 3.1 viser et skema med statistik over kørselstider for metoderne og koden direkte i Main metoden samt de fejl og exceptions der opstår ved de forskellige grænseværdier, der undervejs er blevet synliggjort. Vær opmærksom på, at antallet af cirkler, der er angivet i skemaet, er navngivet efter de amerikanske navne for talværdierne, derfor er talværdierne også angivet i potenstal, så det er nemmere at overskue talværdierne. Derudover er det værd at bemærke, at der er forskel på kørselstiderne alt efter hvor mange cirkler man laver med det kode der er direkte i Main metoden, hvor man kun kan teste ét antal cirkler ad gangen. Af samme grund er køretiderne i skemaet angivet med omtrentlige tidsrum.

Det der er interessant ved også at skrive kode direkte i Main metoden, og sammenligne det med kode, der er indkapslet i en metode, som kaldes fra Main metoden, er de compiler fejl der opstår. Det er blevet afdækket, at der for kode direkte i Main metoden opstår en compiler advarsel, som ikke opstår, når der laves metodekald til en anden metode, som indeholder stort set samme kode. Derudover er der også forskel på de compiler fejl, der opstår med det stigende antal objekter, det forsøges at oprette. Når metoderne kaldes fra Main, kan der fremkaldes tre varianter af den samme compiler fejl(CS1503), mens koden direkte i Main fremkalder compiler advarsel(CS0652) og en anden compiler fejl(CS0034). Endelig er der også en tredje compiler fejl, og det er uanset hvilken metode der kaldes, om koden bliver kørt direkte fra Main eller i en metode kaldt fra Main metoden. Det er den compiler fejl der angiver, at den maksimale int værdi er overskredet, det er fejl CS1021, som opstår, når det forsøges at lave 18.446.744.073.709.551.616 (10^{19}) eller flere cirkler.

For både metoden, der ikke gemmer cirklerne i en liste, og for den del af koden, som ligger direkte i Main metoden, der heller ikke bruger en liste, kan der ikke fremkaldes en OutOfMemoryException, da en compiler fejl opstår, før det når så vidt, at garbage collectoren ikke kan følge med mere. For metoden og det kode, der ligger i Main, som begge bruger en liste, fremkaldes en OutOfMemoryException, når det forsøges at oprette 134.217.729 (10^{8}) eller flere cirkler. Compiler fejlene opstår, når det forsøges at lave 2.147.483.648 (10^{9}) eller flere cirkler.

Der er try-catch strukturer implementeret både i den metode, som bruger en liste og omkring det kode der kompileres direkte fra Main metoden, som også bruger en liste.

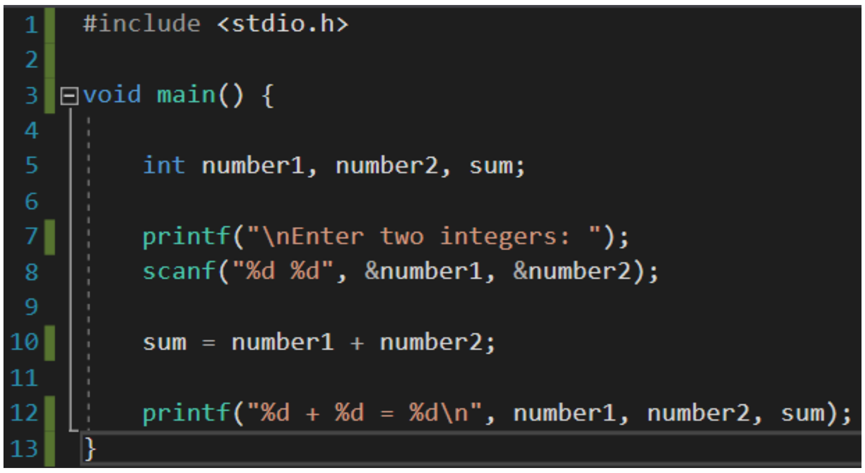
Opgave 4 – Delegates og events

Denne løsning tager udgangspunkt i eksemplet med klassen Stock, som indeholder en brugerdefineret eventhandler på prisændringer, kaldet PriceChangedHandler. Denne løsning er implementeret med to events til prisændringer, en som giver besked om stigende prisændringer, og en der giver besked om faldende prisændringer. Stock klassen kan ses af figur 4.1 i bilag. Der er også implementeret to subscriber metoder, en som abonnerer på prisstigninger og en, som abonnerer på prisfald, de er vist i figurerne 4.2 og 4.3. De bliver tilmeldt hver sin event og senere i programmet bliver de afmeldt eventsene igen. I klassen Stock er der implementeret en if statement, som vurderer, hvilken af de to events, der skal aktiveres, når der sker en prisændring således, at det f.eks. i tilfældet med eventet for prisstigninger kun er de abonnenter, som er interesserede i prisstigninger, der får besked om prisændringen. Main metoden med test data kan ses af figur 4.4, mens output data fra programmet kan ses af figur 4.5 under bilag.

Opgave 5 – C og Assembler kode

Denne løsning laver et simpelt C program, som prompter brugeren til at indtaste to heltal i konsollen, hvorefter funktionen main i programmet lægger tallene sammen og udskriver resultatet til brugeren. I denne løsning er det besluttet at arbejde videre med assembly koden for C programmet til egen maskine, som har en 64-bit arkitektur. Derefter er en assembly fil for C programmet blevet genereret med GCC, hvor option O2 er benyttet. Disse filer ligger i kodeprojektet under LHH_Opgave5 mappen.

Figur 5.1



```
1  #include <stdio.h>
2
3  void main() {
4
5      int number1, number2, sum;
6
7      printf("\nEnter two integers: ");
8      scanf("%d %d", &number1, &number2);
9
10     sum = number1 + number2;
11
12     printf("%d + %d = %d\n", number1, number2, sum);
13 }
```

Figurerne 5.1 og 5.2 viser hhv. C koden og assembly koden for programmet.

I assembly koden starter linje 1 med at angive kildefilens navn.

I linjerne 5 – 14, ser det ud til, at strengene der bruges i programmet, bliver cachet til LC0, LC1 og LC2, hvor LC0 indeholder strengen ”Enter two integers: ”, LC1 indeholder brugerens input, mens LC2 ser ud til at indeholde definitionen for main funktionen, funktionens retur værdi samt formatering og opsætning af teksten.

Linje 16 ser ud til at begynde en blok, som indeholder alt koden til main funktionen.

Linje 17 ser ud til at aktivere proceduren for funktionen.

Det ser ud til at linjerne 18 - 24 godt kunne være main funktionens prologue, som laver opsætning af funktionens StackFrame med en base pointer(esb) og en stack pointer(esp).

Linje 25 kalder main funktionen, som herefter eksekveres.

Det ser ud til at linjerne 26-27 i assembly filen svarer til linje 7 i C koden.

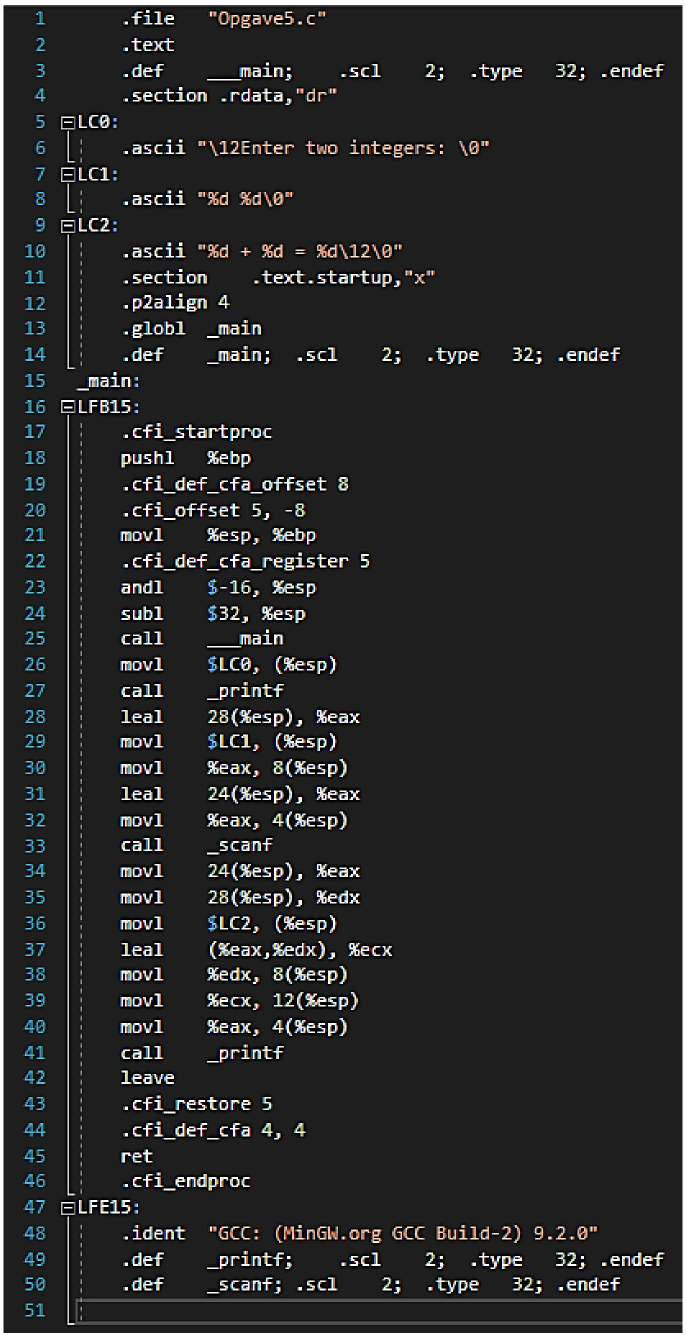
I linje 26 flyttes værdien for LC0 over i lokationen for stackpointeren.

Linje 27 kalder printf, som skriver prompt strengen ud.

Det ser også ud til at linjerne 28 - 33 svarer til linje 8 i C koden, samt at linjerne 34 – 41 i assembly koden svarer til linje 10-12 i C koden.

Linje 47 ser ud til at markere slutningen på den blok som indeholder assembly koden for main metoden.

Figur 5.2



```
 1      .file   "Opgave5.c"
 2      .text
 3      .def    ___main;    .scl    2;  .type   32; .endef
 4      .section .rdata,"dr"
 5  LC0:
 6      .ascii "\12Enter two integers: \0"
 7  LC1:
 8      .ascii "%d %d\0"
 9  LC2:
10      .ascii "%d + %d = %d\12\0"
11      .section    .text.startup,"x"
12      .p2align 4
13      .globl  _main
14      .def    _main;  .scl    2;  .type   32; .endef
15  _main:
16  LFB15:
17      .cfi_startproc
18      pushl   %ebp
19      .cfi_def_cfa_offset 8
20      .cfi_offset 5, -8
21      movl    %esp, %ebp
22      .cfi_def_cfa_register 5
23      andl    $-16, %esp
24      subl    $32, %esp
25      call    ___main
26      movl    $LC0, (%esp)
27      call    _printf
28      leal    28(%esp), %eax
29      movl    $LC1, (%esp)
30      movl    %eax, 8(%esp)
31      leal    24(%esp), %eax
32      movl    %eax, 4(%esp)
33      call    _scanf
34      movl    24(%esp), %eax
35      movl    28(%esp), %edx
36      movl    $LC2, (%esp)
37      leal    (%eax,%edx), %ecx
38      movl    %edx, 8(%esp)
39      movl    %ecx, 12(%esp)
40      movl    %eax, 4(%esp)
41      call    _printf
42      leave
43      .cfi_restore 5
44      .cfi_def_cfa 4, 4
45      ret
46      .cfi_endproc
47  LFE15:
48      .ident  "GCC: (MinGW.org GCC Build-2) 9.2.0"
49      .def    _printf;    .scl    2;  .type   32; .endef
50      .def    _scanf; .scl    2;  .type   32; .endef
51
```

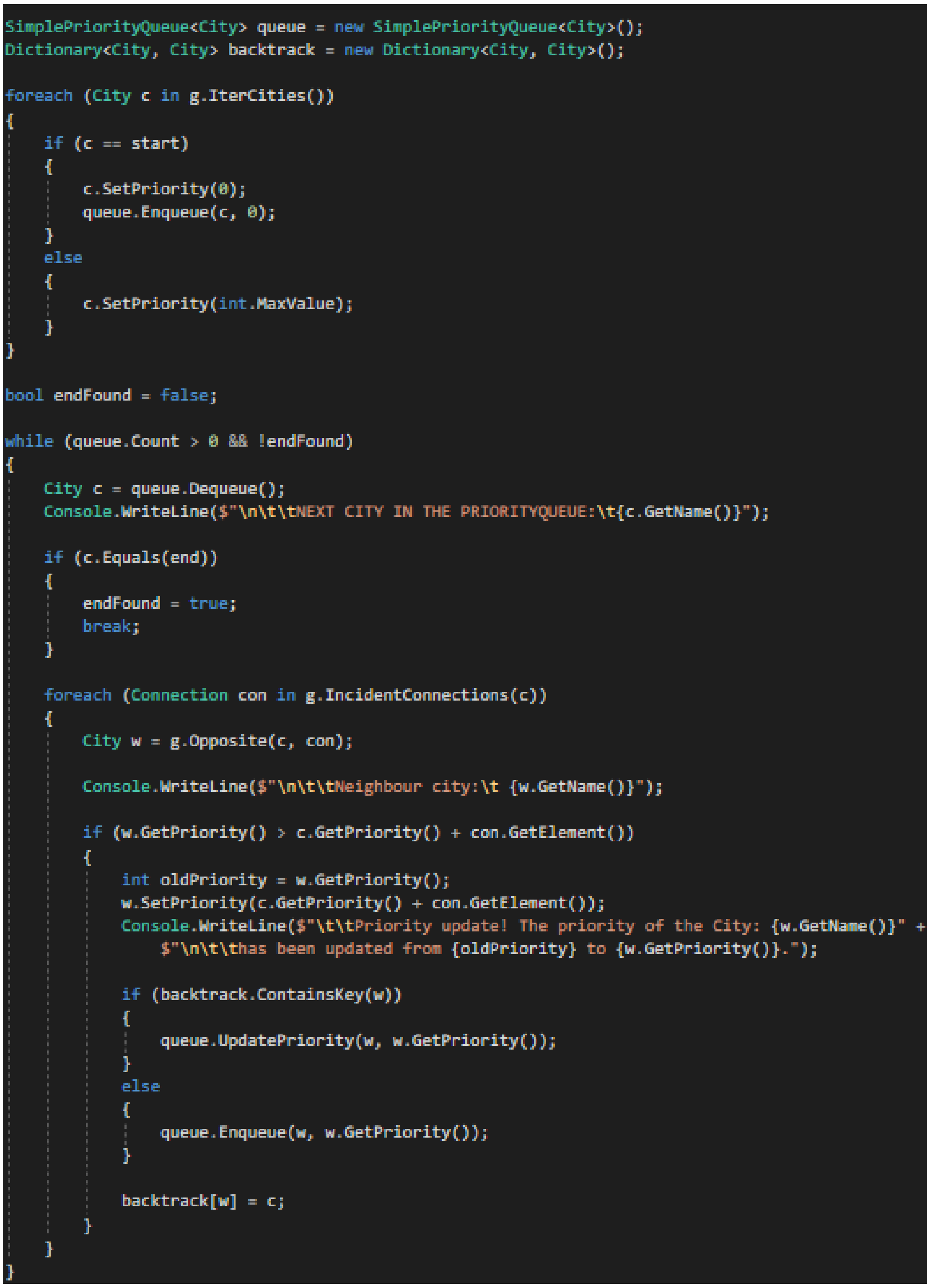
Opgave 6 – Dijkstra's algoritme

Denne løsning vil finde den korteste sti mellem to knuder i en vægtet graf, ved hjælp af dijkstras algoritme, og den returnerer både den samlede omkostning for stien og de knuder stien består af. Løsningen bliver testet på en vægtet graf, som er repræsenteret med en adjacency list struktur, hvilket betyder, at der i knude klassen er implementeret en liste af kanter, som er forbundet til knuden. Det vil sige at hvert knude objekt kender alle sine tilstødende kanter og derfor bliver det også nemmere at tilgå de tilstødende knuder, hvilket effektiviserer visse af grafens metoder sammenlignet med en graf der er repræsenteret med en edge list struktur. Implementationen af grafstrukturen er vist i figurerne 6.4 – 6.11.
Dijkstras algoritme er her implementeret med inspiration og ressourcer fra flere kilder, f.eks. er priority queue datastrukturen, som benyttes i denne løsning, hentet fra NuGet Package Manager, hvor der i projektet for denne opgave er installeret en NuGet package der hedder OptimizedPriorityQueue. Dertil er der et using directive i filen Program.cs, da det er den fil der indeholder algoritmen, som benytter datastrukturen.
Knude klassen er blevet modificeret til at have en variabel, som holder styr på knudens omkostning fra start knuden til denne knude, denne omkostning bruges også som prioritet, således, at denne er let at tilgå. Den sidste funktionalitet er vigtigst, derfor er variablen simpelt nok navngivet priority. Under bilag er det figurerne 6.4 og 6.5 der viser denne modifikation til knudeklassen City.

Figur 6.1 viser en del af algoritmen, hvor der udover prioritetskøen benyttes en dictionary til at holde styr på, hvilke byer algoritmen følger. Undervejs i algoritmen bliver der udskrevet kommentarer til konsollen, for at man kan følge og se, at algoritmen fungerer korrekt.
Til at starte med initialiseres alle byers prioritet med den største int værdi, som C# programmet kan præstere, med undtagelse af den by algoritmen starter fra, som initialiseres til at have en prioritet på 0. Startbyen puttes i prioritetskøen med det samme, mens de andre knuder først kommer i kø senere i algoritmen. For at stoppe algoritmen, når den har fundet slutbyen, er der lavet en boolsk variabel, som indikerer om denne by er fundet. En while løkke køres på to betingelser, 1) så længe at prioritetskøen ikke er tom og 2) så længe at den boolske variabel er falsey. I while løkken hives det element, som har den højeste prioritet ud af prioritetskøen. Derefter undersøges, om elementet er slutbyen, og hvis den er, sættes den boolske værdi til at være truthy, og et break statement sender compileren ud af while løkken. Er elementet fra prioritetskøen ikke slutbyen, startes en foreach løkke for at undersøge knudens naboer. For hver nabo inspiceres naboens prioritet. Er en naboknude allerede i køen, men har opdateret sin prioritet, bliver dens plads i køen blot opdateret. En nyfundet

knude puttes ind i køen med sin prioritet. Til sidst puttes naboknuden i dicitionariet, hvor den bliver mappet til den knude som sidst blev taget ud af køen. Dette fortsætter, indtil en af de to betingelser ikke længere er opfyldt.

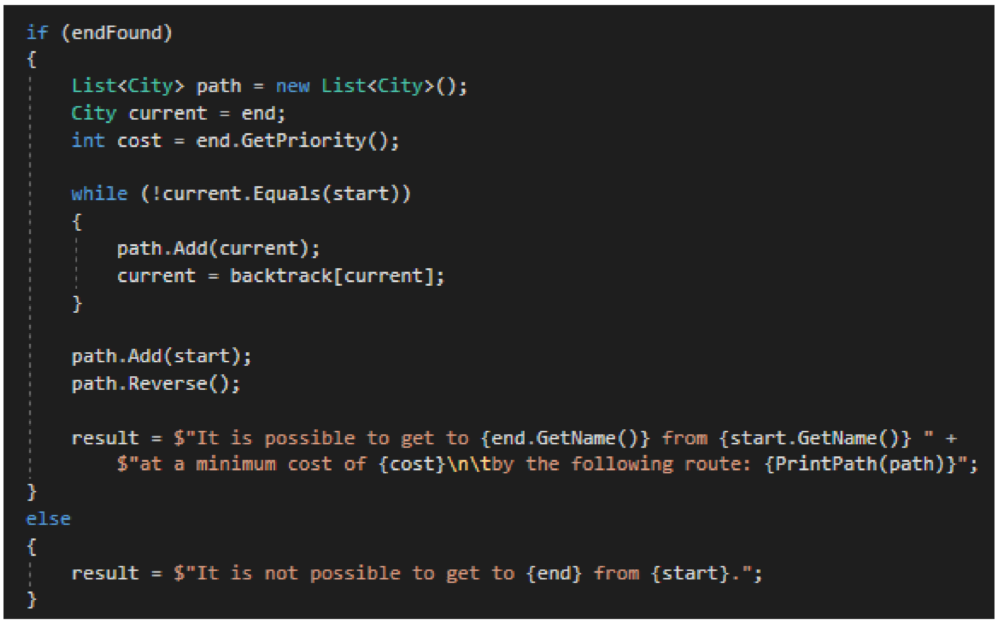
Figur 6.1



```
SimplePriorityQueue<City> queue = new SimplePriorityQueue<City>();
Dictionary<City, City> backtrack = new Dictionary<City, City>();

foreach (City c in g.IterCities())
{
    if (c == start)
    {
        c.SetPriority(0);
        queue.Enqueue(c, 0);
    }
    else
    {
        c.SetPriority(int.MaxValue);
    }
}

bool endFound = false;

while (queue.Count > 0 && !endFound)
{
    City c = queue.Dequeue();
    Console.WriteLine($"\n\t\tNEXT CITY IN THE PRIORITYQUEUE:\t{c.GetName()}");

    if (c.Equals(end))
    {
        endFound = true;
        break;
    }

    foreach (Connection con in g.IncidentConnections(c))
    {
        City w = g.Opposite(c, con);

        Console.WriteLine($"\n\t\tNeighbour city:\t {w.GetName()}");

        if (w.GetPriority() > c.GetPriority() + con.GetElement())
        {
            int oldPriority = w.GetPriority();
            w.SetPriority(c.GetPriority() + con.GetElement());
            Console.WriteLine($"\t\tPriority update! The priority of the City: {w.GetName()}" +
                $"\n\t\thas been updated from {oldPriority} to {w.GetPriority()}.");

            if (backtrack.ContainsKey(w))
            {
                queue.UpdatePriority(w, w.GetPriority());
            }
            else
            {
                queue.Enqueue(w, w.GetPriority());
            }

            backtrack[w] = c;
        }
    }
}
```

Efter while løkken kan man af figur 6.2 se, hvordan resultatet evalueres og konstrueres.

Figur 6.2



```
if (endFound)
{
    List<City> path = new List<City>();
    City current = end;
    int cost = end.GetPriority();

    while (!current.Equals(start))
    {
        path.Add(current);
        current = backtrack[current];
    }

    path.Add(start);
    path.Reverse();

    result = $"It is possible to get to {end.GetName()} from {start.GetName()} " +
        $"at a minimum cost of {cost}\n\tby the following route: {PrintPath(path)}";
}
else
{
    result = $"It is not possible to get to {end} from {start}.";
}
```

Der bliver først evalueret på, om slutbyen blev fundet, hvorefter der i det tilfælde bliver oprettet en liste, en midlertidig knude som initialiseres til at være lig med slutknuden samt en variabel der holder værdien for den samlede omkostning fra start til slut. Derefter følger en while løkke, som løber hele dictionariet igennem, hvor den midlertidige knude bliver tildelt den ene by efter den anden, som kopieres over i listen, indtil den midlertidige knude bliver tildelt startbyen, derved afsluttes while løkken. Startbyen bliver kopieret over i listen efter while løkken og listen som indeholder stien i omvendt rækkefølge, bliver vendt og resultatet kan derefter konstrueres med hjælp fra en Print metode til at printe listen pænt ud.

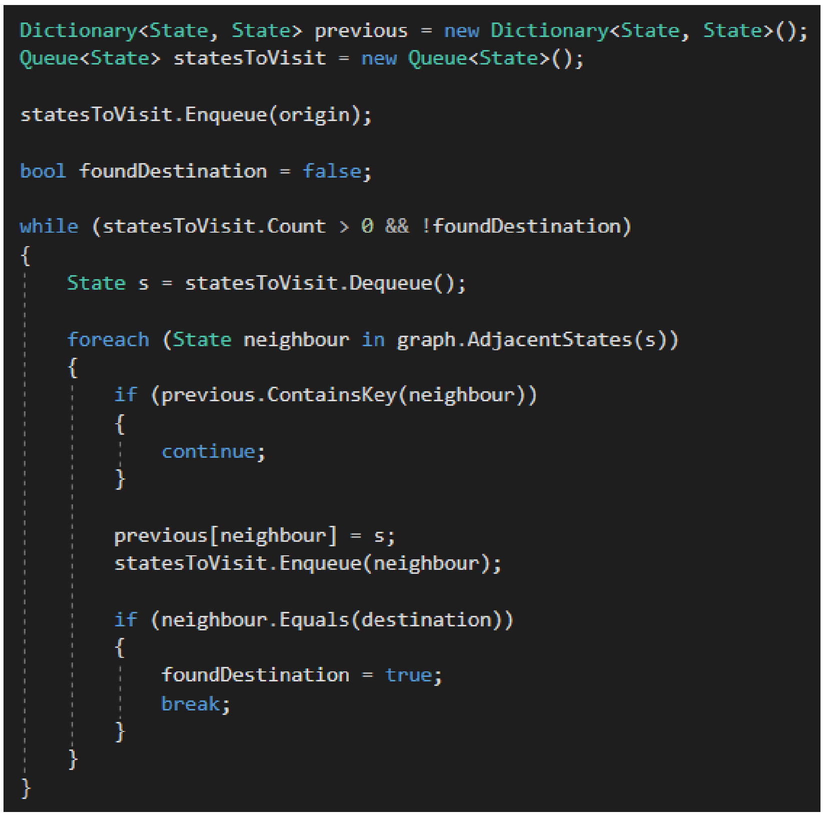
Der er desuden dette kode lavet try-catch strukturer i algoritmen, for at sikre, at der ikke sker nogle fejl, såsom at algoritmen får et ugyldigt argument som f.eks. en by, som slet ikke er en del af grafen.

Løsningen er testet i Main metoden, hvor test data og eksempler på output af test data fra Main metoden kan ses af figurerne 6.12 – 6.21 under bilag.

Bilag

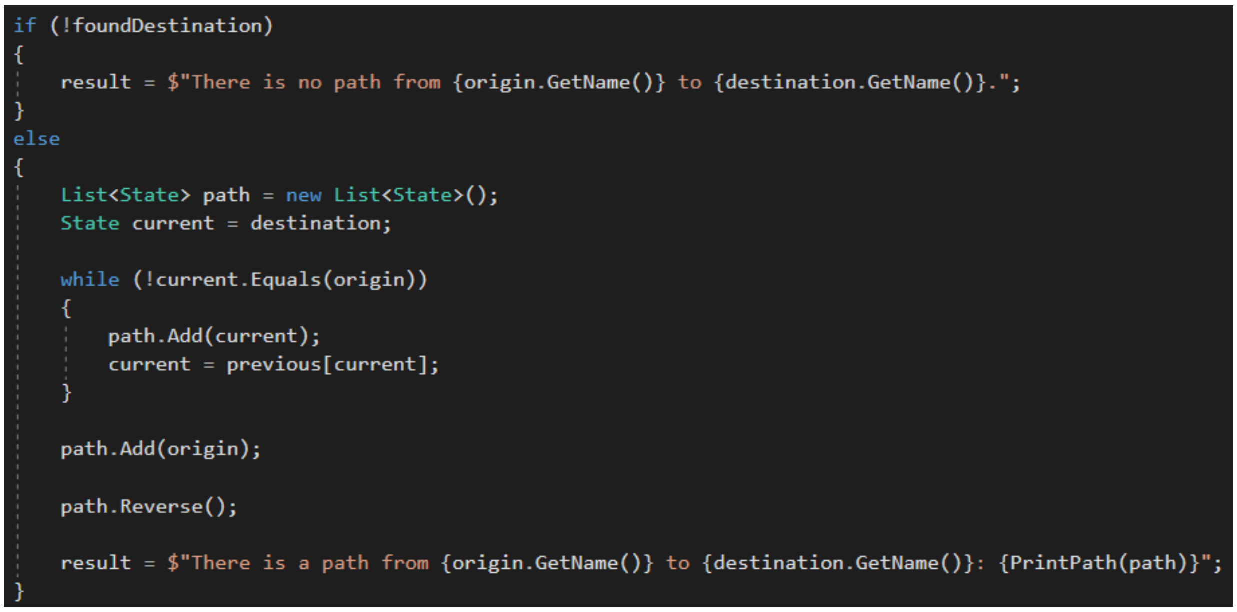
Opgave 1

Figur 1.1: Metoden FindPathBFS - Find en sti, hvis der er en



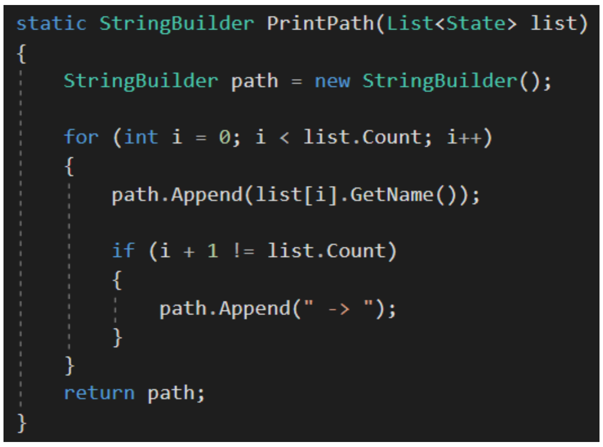
```
Dictionary<State, State> previous = new Dictionary<State, State>();
Queue<State> statesToVisit = new Queue<State>();

statesToVisit.Enqueue(origin);

bool foundDestination = false;

while (statesToVisit.Count > 0 && !foundDestination)
{
    State s = statesToVisit.Dequeue();

    foreach (State neighbour in graph.AdjacentStates(s))
    {
        if (previous.ContainsKey(neighbour))
        {
            continue;
        }

        previous[neighbour] = s;
        statesToVisit.Enqueue(neighbour);

        if (neighbour.Equals(destination))
        {
            foundDestination = true;
            break;
        }
    }
}
```

Figur 1.2: Metoden FindPathBFS - Resultatet udskrives



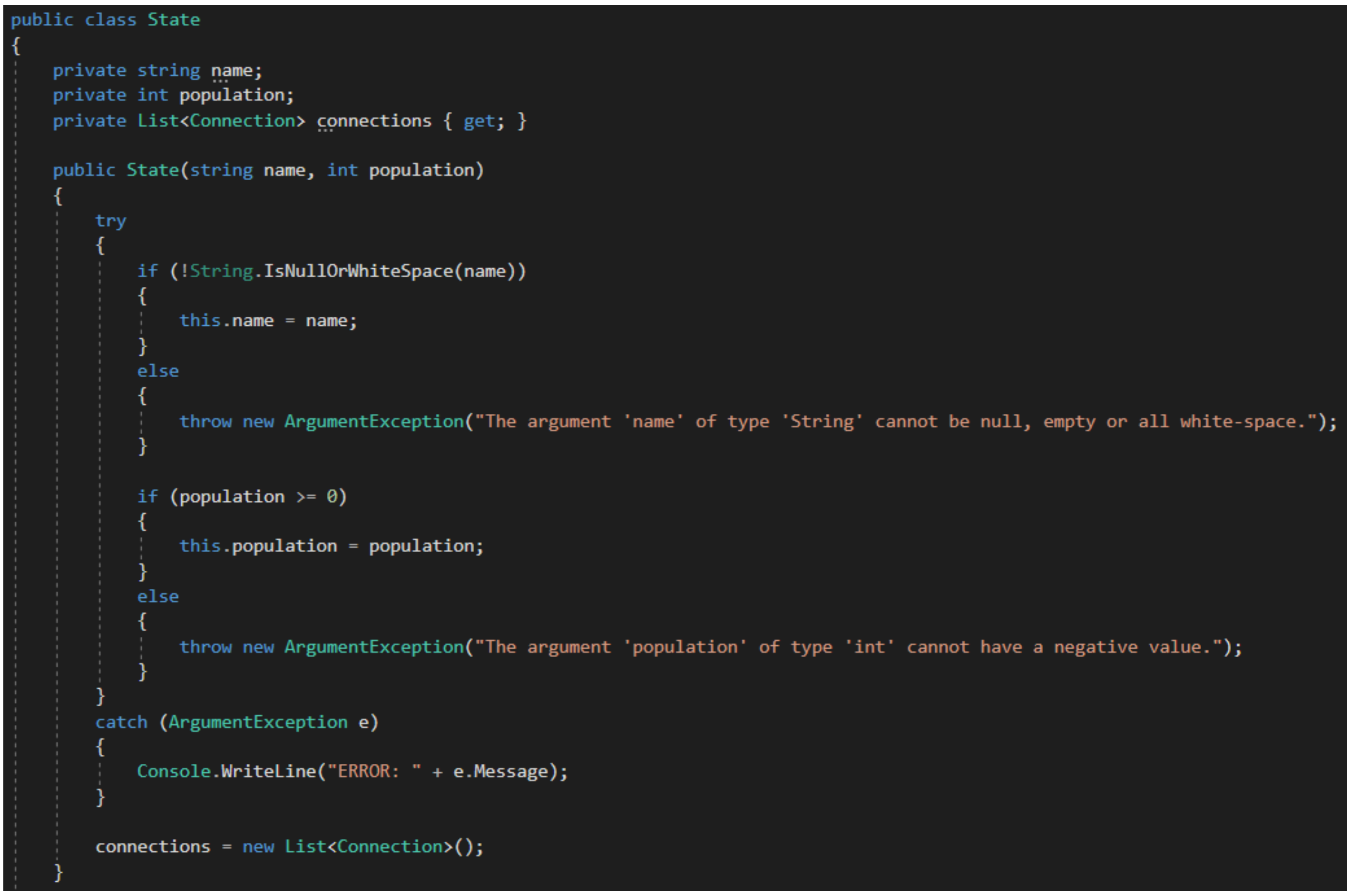
```
if (!foundDestination)
{
    result = $"There is no path from {origin.GetName()} to {destination.GetName()}.";
}
else
{
    List<State> path = new List<State>();
    State current = destination;

    while (!current.Equals(origin))
    {
        path.Add(current);
        current = previous[current];
    }

    path.Add(origin);

    path.Reverse();

    result = $"There is a path from {origin.GetName()} to {destination.GetName()}: {PrintPath(path)}";
}
```

Figur 1.3: Hjælpemetode PrintPath - Print en sti



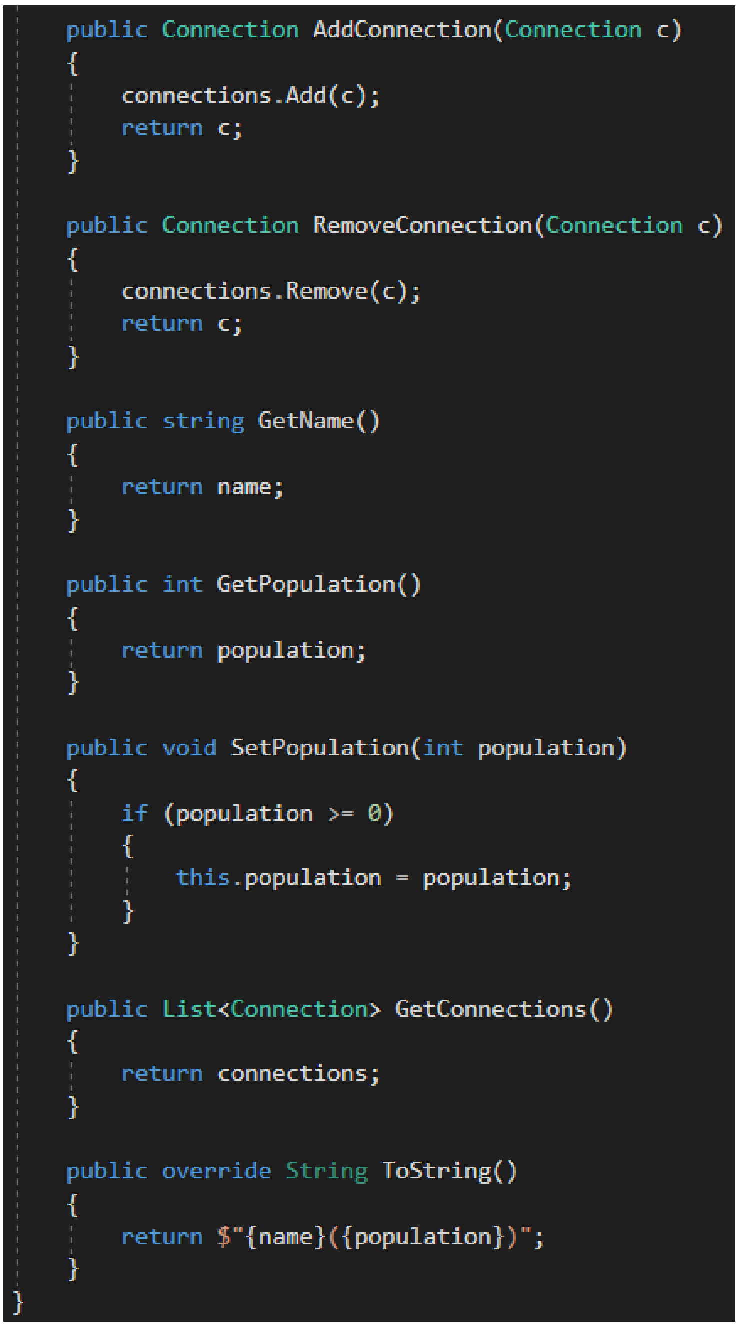
```
static StringBuilder PrintPath(List<State> list)
{
    StringBuilder path = new StringBuilder();

    for (int i = 0; i < list.Count; i++)
    {
        path.Append(list[i].GetName());

        if (i + 1 != list.Count)
        {
            path.Append(" -> ");
        }
    }
    return path;
}
```

Figur 1.4: Knudeklassen State, del 1 af 2



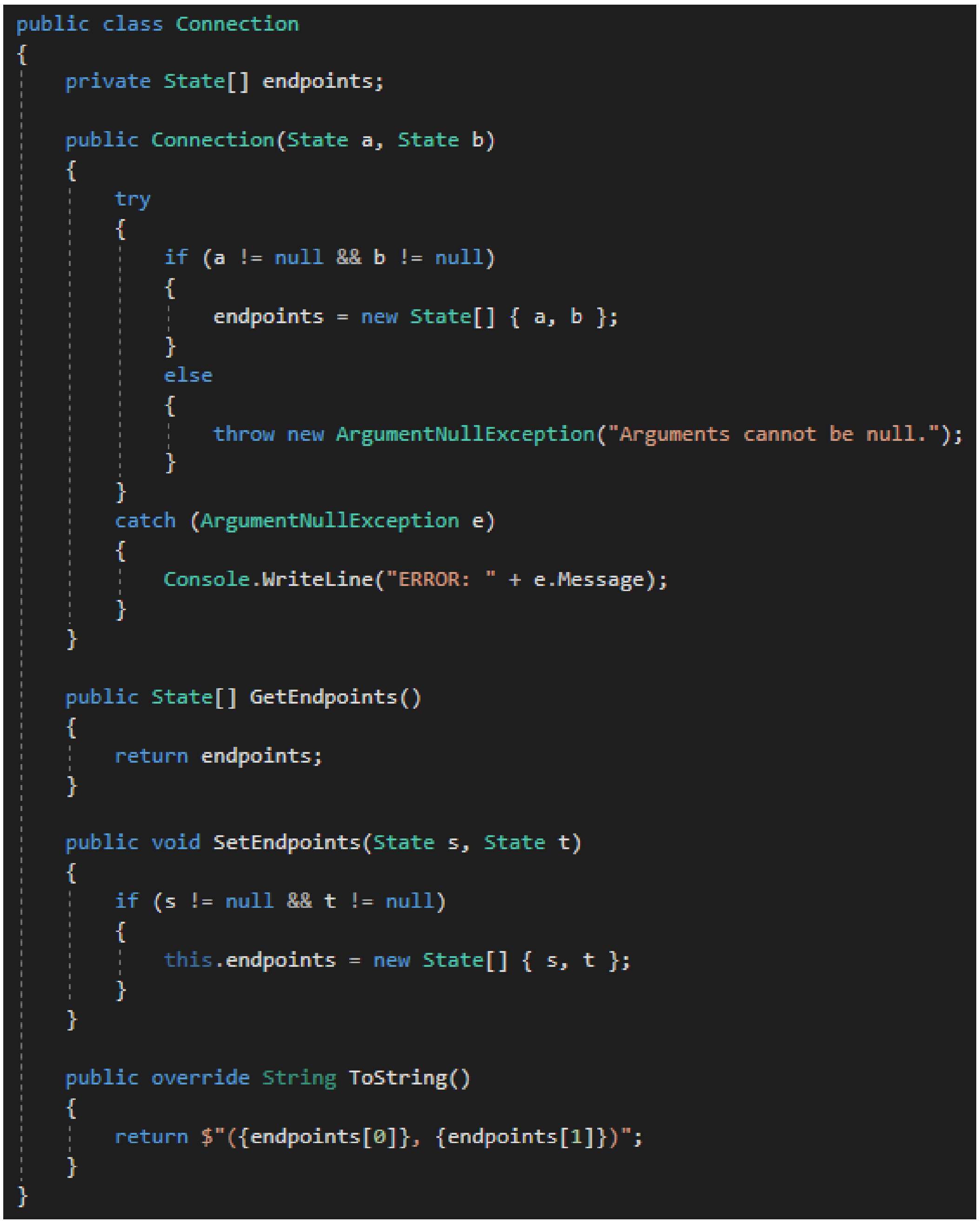
```
public class State
{
    private string name;
    private int population;
    private List<Connection> connections { get; }

    public State(string name, int population)
    {
        try
        {
            if (!String.IsNullOrWhiteSpace(name))
            {
                this.name = name;
            }
            else
            {
                throw new ArgumentException("The argument 'name' of type 'String' cannot be null, empty or all white-space.");
            }

            if (population >= 0)
            {
                this.population = population;
            }
            else
            {
                throw new ArgumentException("The argument 'population' of type 'int' cannot have a negative value.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        connections = new List<Connection>();
    }
```

Figur 1.5: Knudeklassen State, del 2 af 2



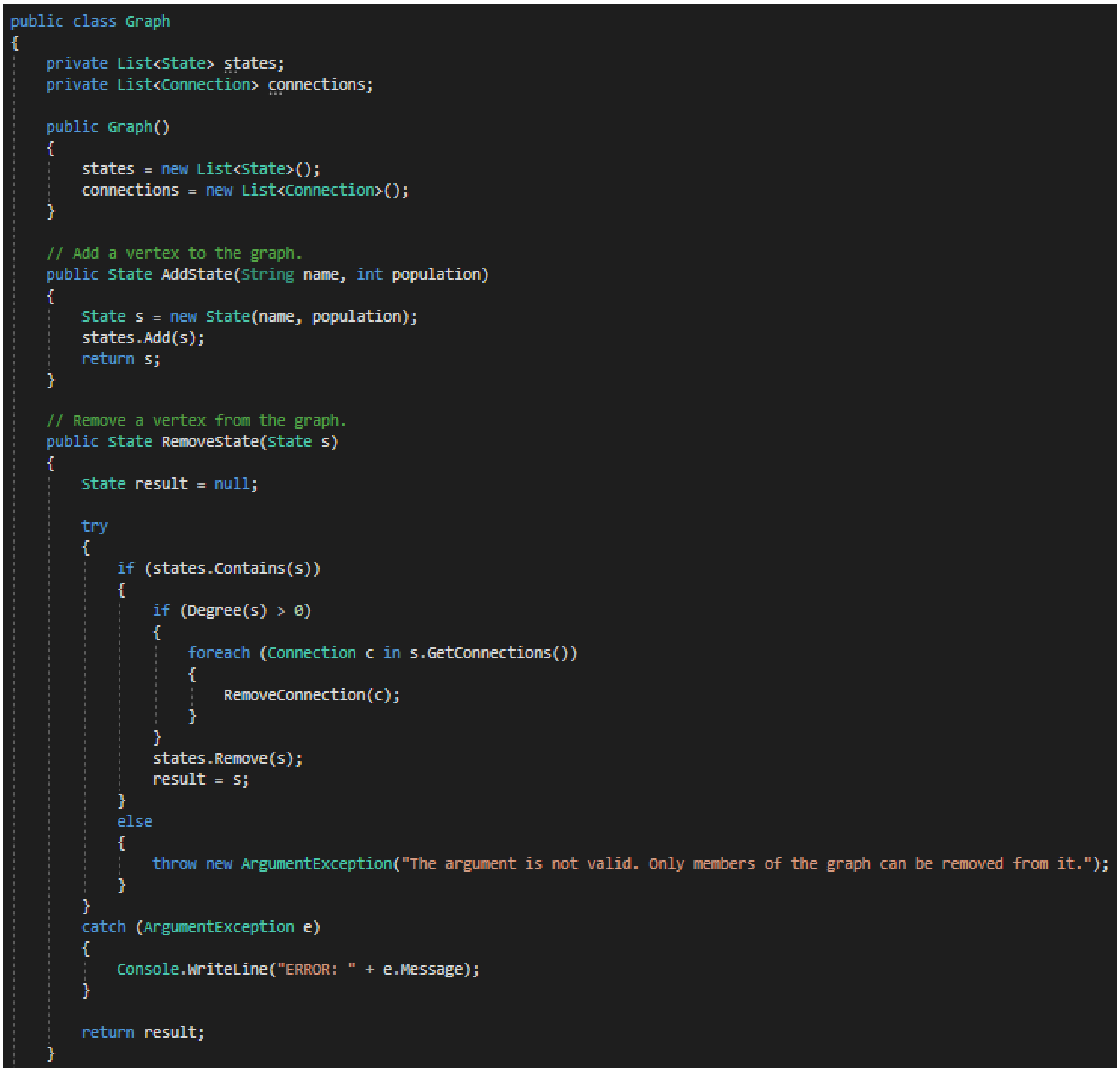
```
    public Connection AddConnection(Connection c)
    {
        connections.Add(c);
        return c;
    }

    public Connection RemoveConnection(Connection c)
    {
        connections.Remove(c);
        return c;
    }

    public string GetName()
    {
        return name;
    }

    public int GetPopulation()
    {
        return population;
    }

    public void SetPopulation(int population)
    {
        if (population >= 0)
        {
            this.population = population;
        }
    }

    public List<Connection> GetConnections()
    {
        return connections;
    }

    public override String ToString()
    {
        return $"{name}({population})";
    }
}
```

Figur 1.6: Kantklassen Connection



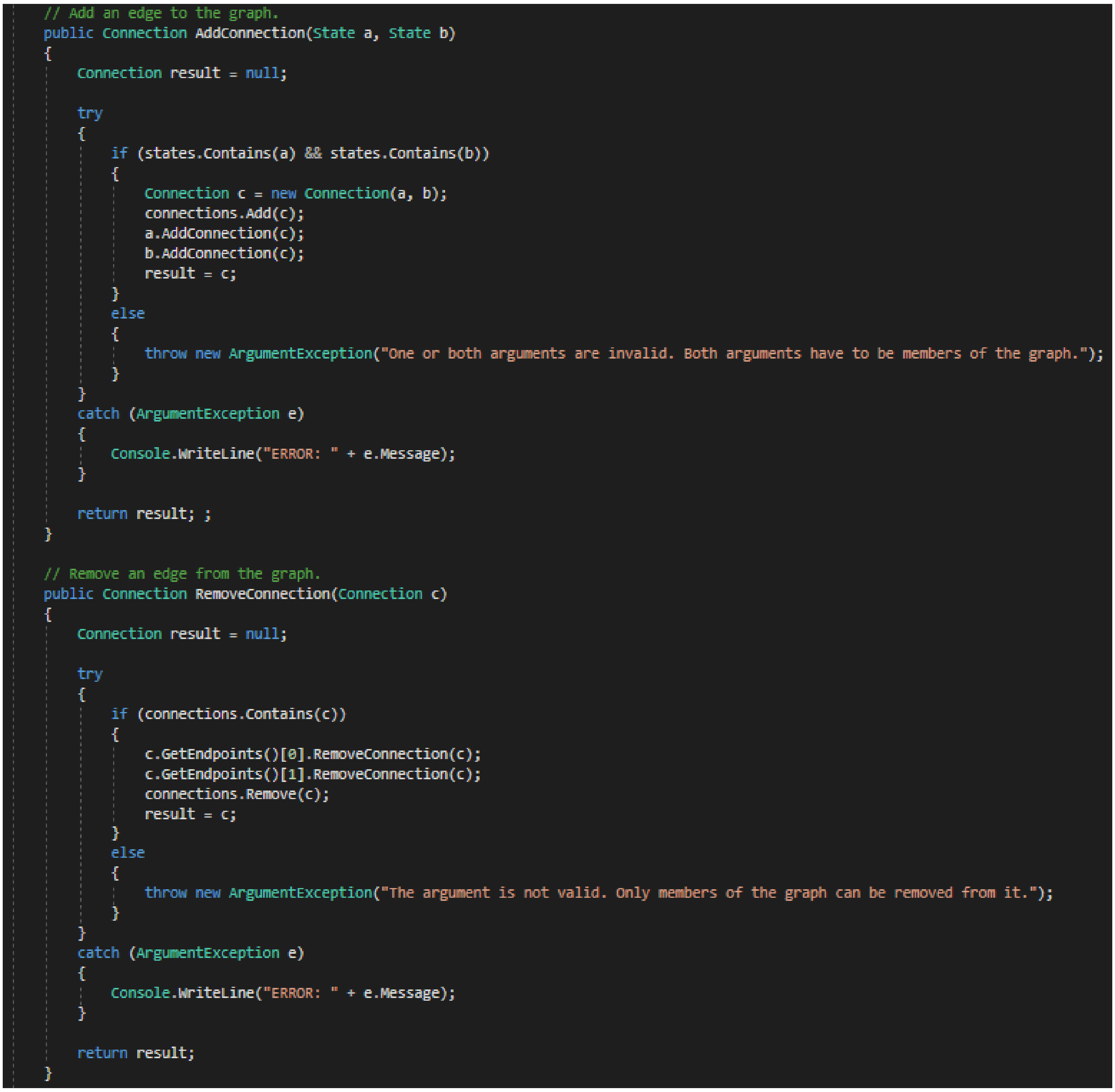
```
public class Connection
{
    private State[] endpoints;

    public Connection(State a, State b)
    {
        try
        {
            if (a != null && b != null)
            {
                endpoints = new State[] { a, b };
            }
            else
            {
                throw new ArgumentNullException("Arguments cannot be null.");
            }
        }
        catch (ArgumentNullException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }
    }

    public State[] GetEndpoints()
    {
        return endpoints;
    }

    public void SetEndpoints(State s, State t)
    {
        if (s != null && t != null)
        {
            this.endpoints = new State[] { s, t };
        }
    }

    public override String ToString()
    {
        return $"({endpoints[0]}, {endpoints[1]})";
    }
}
```

Figur 1.7: Grafklassen Graph, del 1 af 5



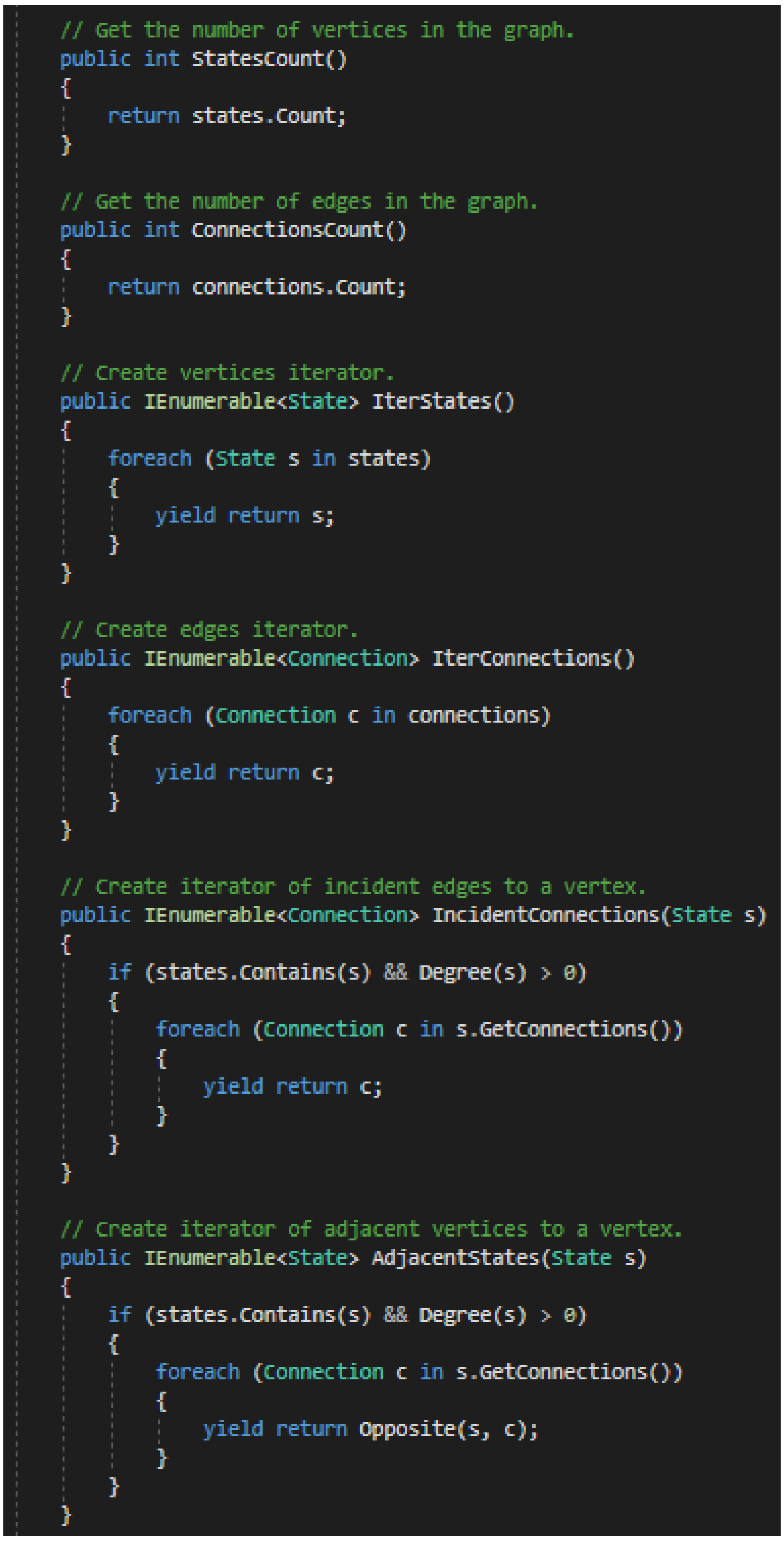
```
public class Graph
{
    private List<State> states;
    private List<Connection> connections;

    public Graph()
    {
        states = new List<State>();
        connections = new List<Connection>();
    }

    // Add a vertex to the graph.
    public State AddState(String name, int population)
    {
        State s = new State(name, population);
        states.Add(s);
        return s;
    }

    // Remove a vertex from the graph.
    public State RemoveState(State s)
    {
        State result = null;

        try
        {
            if (states.Contains(s))
            {
                if (Degree(s) > 0)
                {
                    foreach (Connection c in s.GetConnections())
                    {
                        RemoveConnection(c);
                    }
                }
                states.Remove(s);
                result = s;
            }
            else
            {
                throw new ArgumentException("The argument is not valid. Only members of the graph can be removed from it.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        return result;
    }
```

Figur 1.8: Grafklassen Graph, del 2 af 5



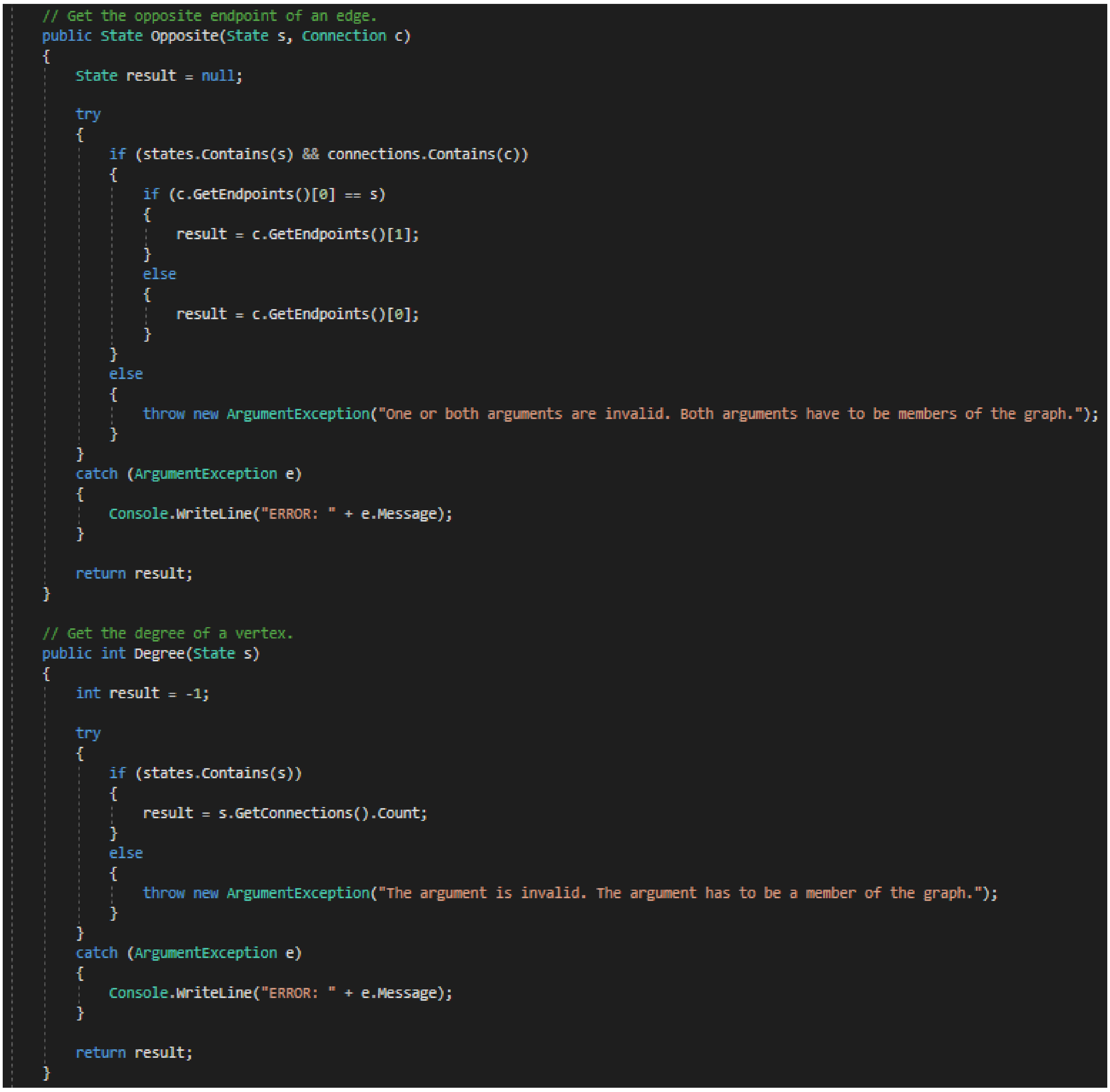
```
// Add an edge to the graph.
public Connection AddConnection(State a, State b)
{
    Connection result = null;

    try
    {
        if (states.Contains(a) && states.Contains(b))
        {
            Connection c = new Connection(a, b);
            connections.Add(c);
            a.AddConnection(c);
            b.AddConnection(c);
            result = c;
        }
        else
        {
            throw new ArgumentException("One or both arguments are invalid. Both arguments have to be members of the graph.");
        }
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("ERROR: " + e.Message);
    }

    return result; ;
}

// Remove an edge from the graph.
public Connection RemoveConnection(Connection c)
{
    Connection result = null;

    try
    {
        if (connections.Contains(c))
        {
            c.GetEndpoints()[0].RemoveConnection(c);
            c.GetEndpoints()[1].RemoveConnection(c);
            connections.Remove(c);
            result = c;
        }
        else
        {
            throw new ArgumentException("The argument is not valid. Only members of the graph can be removed from it.");
        }
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("ERROR: " + e.Message);
    }

    return result;
}
```

Figur 1.9: Grafklassen Graph, del 3 af 5



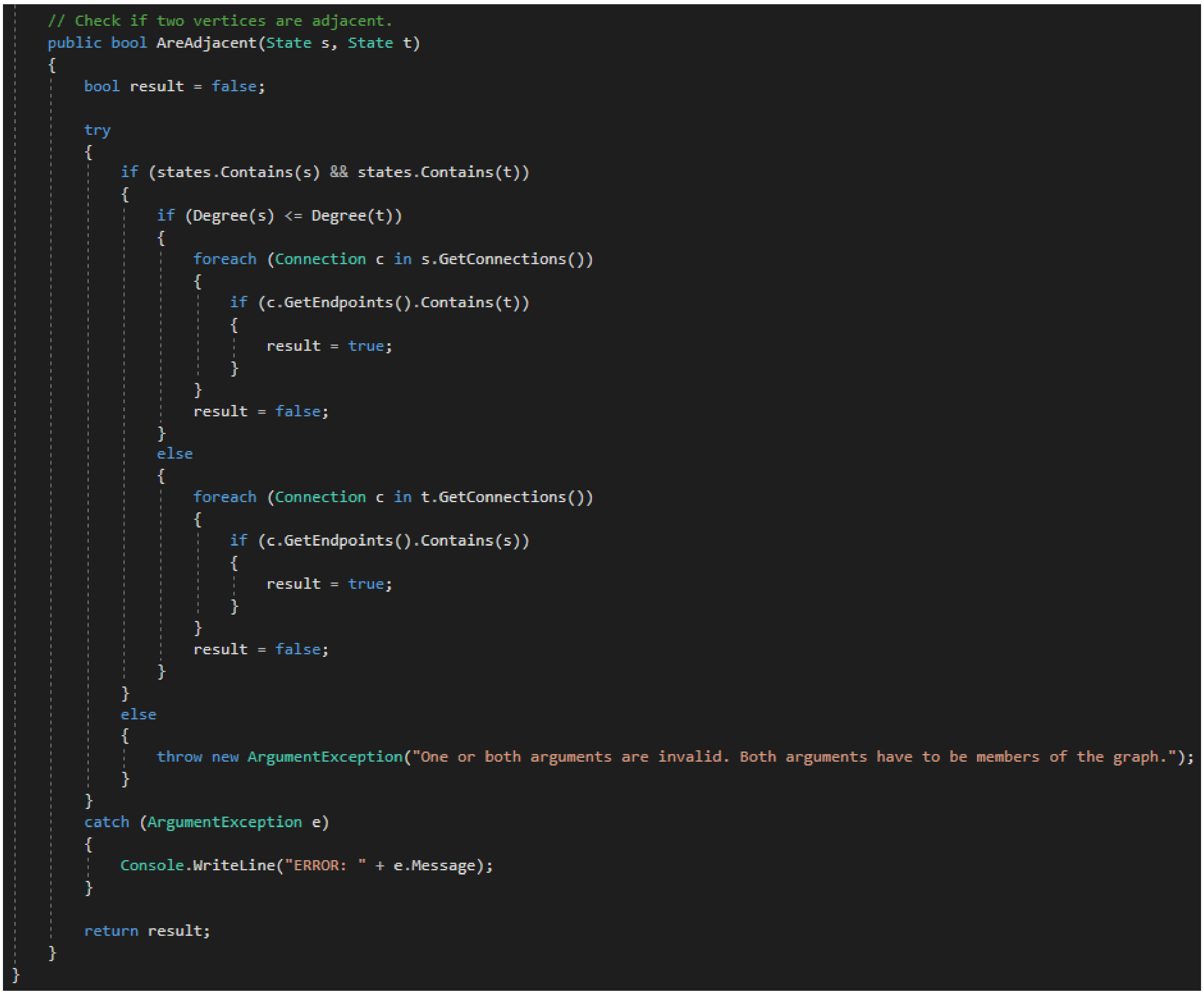
```
// Get the number of vertices in the graph.
public int StatesCount()
{
    return states.Count;
}

// Get the number of edges in the graph.
public int ConnectionsCount()
{
    return connections.Count;
}

// Create vertices iterator.
public IEnumerable<State> IterStates()
{
    foreach (State s in states)
    {
        yield return s;
    }
}

// Create edges iterator.
public IEnumerable<Connection> IterConnections()
{
    foreach (Connection c in connections)
    {
        yield return c;
    }
}

// Create iterator of incident edges to a vertex.
public IEnumerable<Connection> IncidentConnections(State s)
{
    if (states.Contains(s) && Degree(s) > 0)
    {
        foreach (Connection c in s.GetConnections())
        {
            yield return c;
        }
    }
}

// Create iterator of adjacent vertices to a vertex.
public IEnumerable<State> AdjacentStates(State s)
{
    if (states.Contains(s) && Degree(s) > 0)
    {
        foreach (Connection c in s.GetConnections())
        {
            yield return Opposite(s, c);
        }
    }
}
```

Figur 1.10: Grafklassen Graph, del 4 af 5



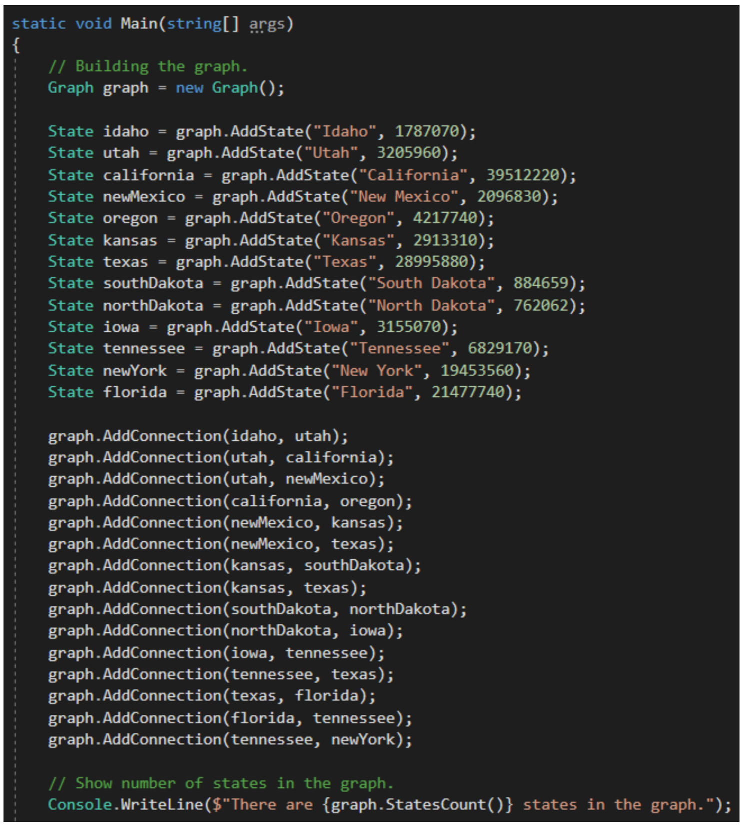
```
// Get the opposite endpoint of an edge.
public State Opposite(State s, Connection c)
{
    State result = null;

    try
    {
        if (states.Contains(s) && connections.Contains(c))
        {
            if (c.GetEndpoints()[0] == s)
            {
                result = c.GetEndpoints()[1];
            }
            else
            {
                result = c.GetEndpoints()[0];
            }
        }
        else
        {
            throw new ArgumentException("One or both arguments are invalid. Both arguments have to be members of the graph.");
        }
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("ERROR: " + e.Message);
    }

    return result;
}

// Get the degree of a vertex.
public int Degree(State s)
{
    int result = -1;

    try
    {
        if (states.Contains(s))
        {
            result = s.GetConnections().Count;
        }
        else
        {
            throw new ArgumentException("The argument is invalid. The argument has to be a member of the graph.");
        }
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("ERROR: " + e.Message);
    }

    return result;
}
```

Figur 1.11: Grafklassen Graph, del 5 af 5



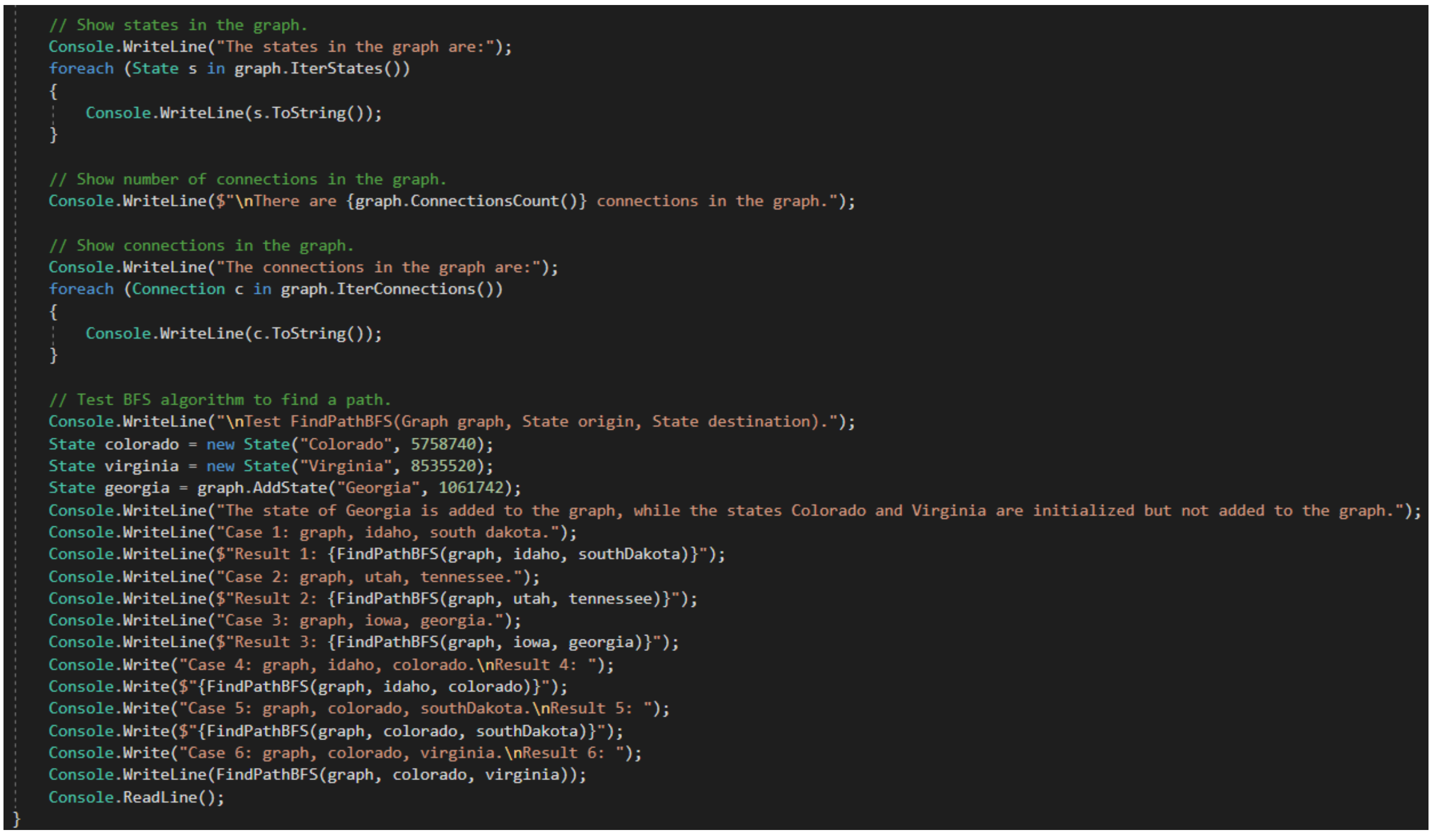
```
    // Check if two vertices are adjacent.
    public bool AreAdjacent(State s, State t)
    {
        bool result = false;

        try
        {
            if (states.Contains(s) && states.Contains(t))
            {
                if (Degree(s) <= Degree(t))
                {
                    foreach (Connection c in s.GetConnections())
                    {
                        if (c.GetEndpoints().Contains(t))
                        {
                            result = true;
                        }
                    }
                    result = false;
                }
                else
                {
                    foreach (Connection c in t.GetConnections())
                    {
                        if (c.GetEndpoints().Contains(s))
                        {
                            result = true;
                        }
                    }
                    result = false;
                }
            }
            else
            {
                throw new ArgumentException("One or both arguments are invalid. Both arguments have to be members of the graph.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        return result;
    }
}
```

Figur 1.12: Main metode, del 1 af 2



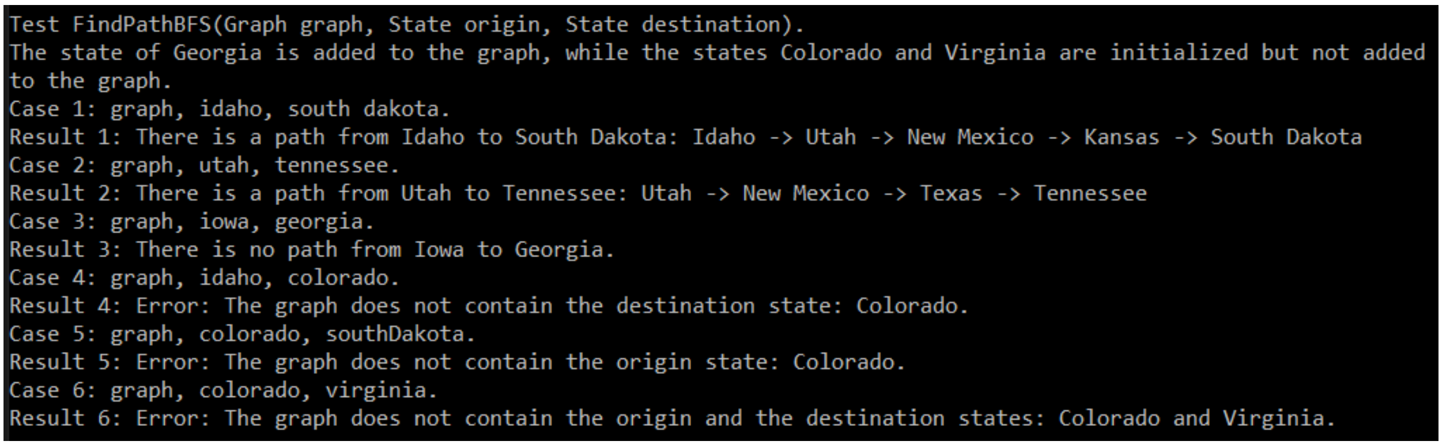
```
static void Main(string[] args)
{
    // Building the graph.
    Graph graph = new Graph();

    State idaho = graph.AddState("Idaho", 1787070);
    State utah = graph.AddState("Utah", 3205960);
    State california = graph.AddState("California", 39512220);
    State newMexico = graph.AddState("New Mexico", 2096830);
    State oregon = graph.AddState("Oregon", 4217740);
    State kansas = graph.AddState("Kansas", 2913310);
    State texas = graph.AddState("Texas", 28995880);
    State southDakota = graph.AddState("South Dakota", 884659);
    State northDakota = graph.AddState("North Dakota", 762062);
    State iowa = graph.AddState("Iowa", 3155070);
    State tennessee = graph.AddState("Tennessee", 6829170);
    State newYork = graph.AddState("New York", 19453560);
    State florida = graph.AddState("Florida", 21477740);

    graph.AddConnection(idaho, utah);
    graph.AddConnection(utah, california);
    graph.AddConnection(utah, newMexico);
    graph.AddConnection(california, oregon);
    graph.AddConnection(newMexico, kansas);
    graph.AddConnection(newMexico, texas);
    graph.AddConnection(kansas, southDakota);
    graph.AddConnection(kansas, texas);
    graph.AddConnection(southDakota, northDakota);
    graph.AddConnection(northDakota, iowa);
    graph.AddConnection(iowa, tennessee);
    graph.AddConnection(tennessee, texas);
    graph.AddConnection(texas, florida);
    graph.AddConnection(florida, tennessee);
    graph.AddConnection(tennessee, newYork);

    // Show number of states in the graph.
    Console.WriteLine($"There are {graph.StatesCount()} states in the graph.");
```

Figur 1.13: Main metode, del 2 af 2



```
    // Show states in the graph.
    Console.WriteLine("The states in the graph are:");
    foreach (State s in graph.IterStates())
    {
        Console.WriteLine(s.ToString());
    }

    // Show number of connections in the graph.
    Console.WriteLine($"\nThere are {graph.ConnectionsCount()} connections in the graph.");

    // Show connections in the graph.
    Console.WriteLine("The connections in the graph are:");
    foreach (Connection c in graph.IterConnections())
    {
        Console.WriteLine(c.ToString());
    }

    // Test BFS algorithm to find a path.
    Console.WriteLine("\nTest FindPathBFS(Graph graph, State origin, State destination).");
    State colorado = new State("Colorado", 5758740);
    State virginia = new State("Virginia", 8535520);
    State georgia = graph.AddState("Georgia", 1061742);
    Console.WriteLine("The state of Georgia is added to the graph, while the states Colorado and Virginia are initialized but not added to the graph.");
    Console.WriteLine("Case 1: graph, idaho, south dakota.");
    Console.WriteLine($"Result 1: {FindPathBFS(graph, idaho, southDakota)}");
    Console.WriteLine("Case 2: graph, utah, tennessee.");
    Console.WriteLine($"Result 2: {FindPathBFS(graph, utah, tennessee)}");
    Console.WriteLine("Case 3: graph, iowa, georgia.");
    Console.WriteLine($"Result 3: {FindPathBFS(graph, iowa, georgia)}");
    Console.Write("Case 4: graph, idaho, colorado.\nResult 4: ");
    Console.Write($"{FindPathBFS(graph, idaho, colorado)}");
    Console.Write("Case 5: graph, colorado, southDakota.\nResult 5: ");
    Console.Write($"{FindPathBFS(graph, colorado, southDakota)}");
    Console.Write("Case 6: graph, colorado, virginia.\nResult 6: ");
    Console.WriteLine(FindPathBFS(graph, colorado, virginia));
    Console.ReadLine();
}
```

Figur 1.14: Output test data



```
Test FindPathBFS(Graph graph, State origin, State destination).
The state of Georgia is added to the graph, while the states Colorado and Virginia are initialized but not added
to the graph.
Case 1: graph, idaho, south dakota.
Result 1: There is a path from Idaho to South Dakota: Idaho -> Utah -> New Mexico -> Kansas -> South Dakota
Case 2: graph, utah, tennessee.
Result 2: There is a path from Utah to Tennessee: Utah -> New Mexico -> Texas -> Tennessee
Case 3: graph, iowa, georgia.
Result 3: There is no path from Iowa to Georgia.
Case 4: graph, idaho, colorado.
Result 4: Error: The graph does not contain the destination state: Colorado.
Case 5: graph, colorado, southDakota.
Result 5: Error: The graph does not contain the origin state: Colorado.
Case 6: graph, colorado, virginia.
Result 6: Error: The graph does not contain the origin and the destination states: Colorado and Virginia.
```

Opgave 2

Figur 2.1: Sortering af strenge i en List<String> datastruktur med forskellige antal strenge

Antal strenge i en List<String>	BubbleSort	MergeSort	Sort
10.000	00 min. 04,35 – 04,68 sec.	00 min. 00,03 – 00,05 sec.	00 min. 00,10 – 00,11 sec.
20.000	00 min. 14,18 – 18,88 sec.	00 min. 00,03 – 00,05 sec.	00 min. 00,10 – 00,16 sec.
30.000	00 min. 34,29 – 35,66 sec.	00 min. 00,04 – 00,05 sec.	00 min. 00,15 – 00,17 sec.
40.000	01 min. 09,13 – 11,05 sec.	00 min. 00,07 – 00,10 sec.	00 min. 00,22 – 00,29 sec.
50.000	01 min. 54,83 – 59,90 sec.	00 min. 00,09 – 00,14 sec.	00 min. 00,28 – 0,39 sec.
100.000	13 min. 30,16 sec.	01 min. 22,00 – 59,00 sec.	02 min. 00,00 – 59,00 sec.

Figur 2.2: Sortering af strenge i en StringCollection datastruktur med forskellige antal strenge

Antal strenge i en StringCollection	BubbleSort	MergeSort
10.000	00 min. 09,47 – 09,81 sec.	00 min. 00,08 – 00,09 sec.
20.000	00 min. 16,30 – 21,78 sec.	00 min. 00,03 – 00,04 sec.
30.000	00 min.37.96 – 40,21 sec.	00 min. 00,05 – 00,06 sec.
40.000	01 min. 17,47 – 90,80 sec.	00 min. 00,09 – 00,14 sec.
50.000	02 min. 07,15 – 20,25 sec.	00 min. 00,11 – 00,17 sec.
100.000	28 min. 28,19 sec.	00 min. 00,90 – 00,99 sec.

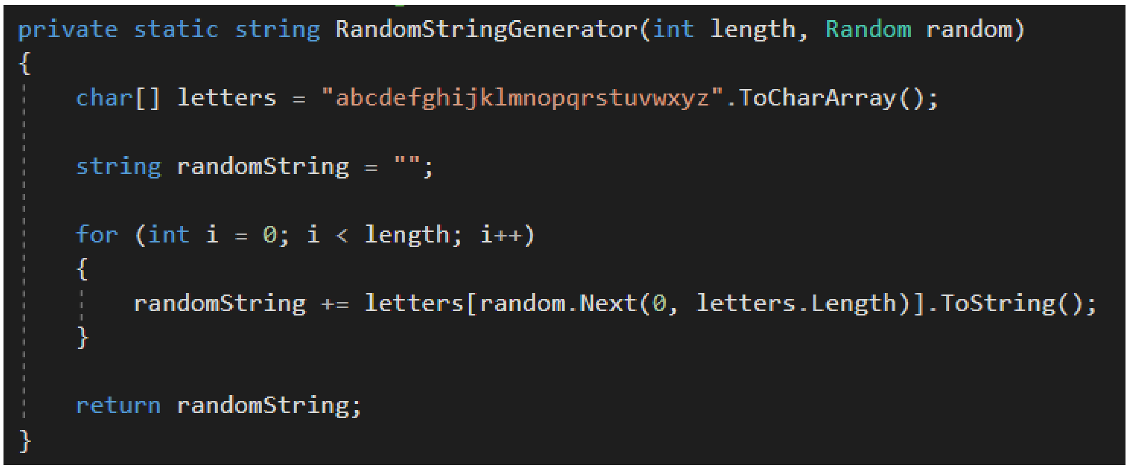
Figur 2.3: Sortering af strenge i en Array datastruktur med forskellige antal strenge

Antal strenge i et array	BubbleSort	MergeSort	Array.Sort
10.000	00 min. 03,56 – 04,51 sec.	00 min. 00,90 – 00,99 sec.	00 min. 00,03 – 00,06 sec.
20.000	00 min.12,24 – 15,82 sec.	00 min. 00,38 – 00,46 sec.	00 min. 00,06 – 00,09 sec.
30.000	00 min. 29,02 – 30,21 sec.	00 min. 00,83 – 00,86 sec.	00 min. 00,10 – 00,12 sec.
40.000	00 min. 58,70 – 95,80 sec.	00 min. 01,52 – 01,76 sec.	00 min. 00,14 – 00,24 sec.
50.000	01 min. 39,11 – 95,65 sec.	00 min. 02,38 – 03,52 sec.	00 min. 00,19 – 00,29 sec.
100.000	12 min. 18,04 sec.	00 min. 20,00 – 40,00 sec.	00 min. 00,58 sec.

Figur 2.4: Søgning i forskellige datastrukturer med forskelligt antal strenge

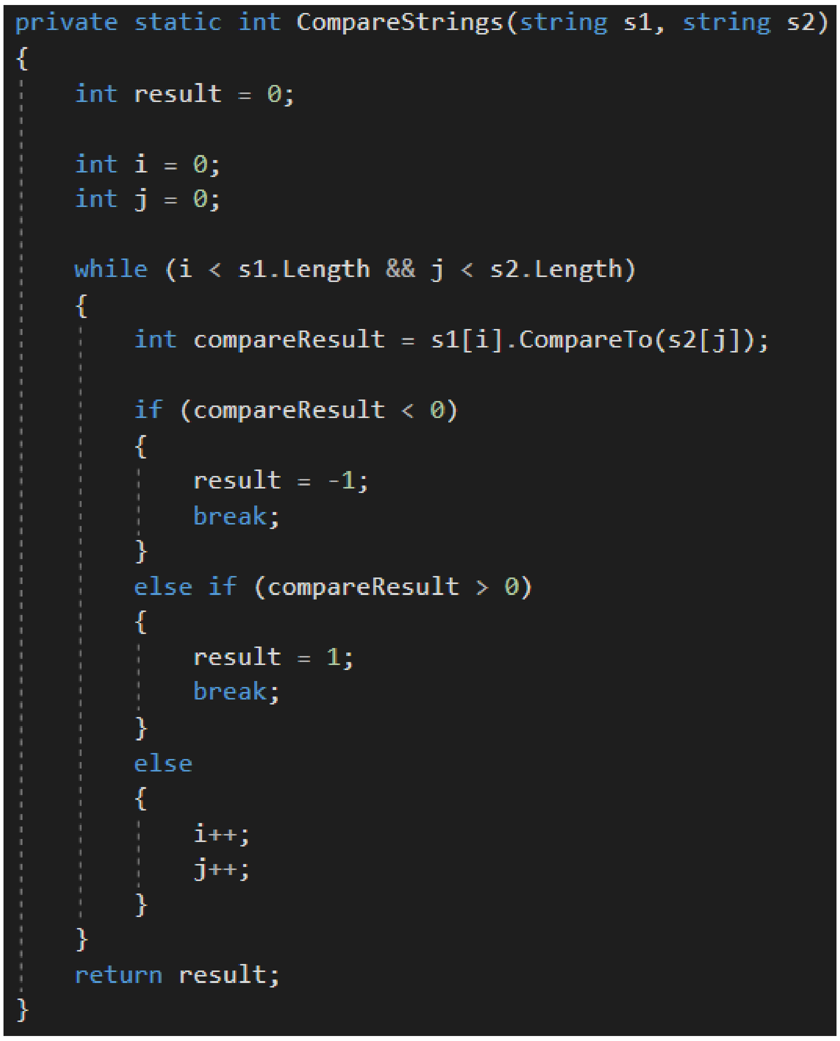
Antal strenge i datastrukturen	List<String>	StringCollection	Array
100.000	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.
200.000	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.
300.000	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.	00,1000000 – 00,0000000 sec.
400.000	00,0051502 – 00,0622248 sec.	00,0030141 – 00,0560306 sec.	00,0012304 – 00,0327321 sec.
600.000	00,0080424 – 00,0798676 sec.	00,0209383 – 00,0897238 sec.	00,0092858 – 00,0476181 sec.
800.000	00,0062061 – 00,0795540 sec.	00,0027690 – 00,0910171 sec.	00,0018783 – 00,0659603 sec.
1.000.000	00,0112372 – 00,0721235 sec.	00,0569319 – 00,1366675 sec.	00,0263200 – 00,0690043 sec.
2.000.000	00,001330342 – 00,1565724 sec.	00,0172298 – 00,1612037 sec.	00,0015577 – 00,1130876 sec.

Figur 2.5: Metoden som genererer en tilfældig streng



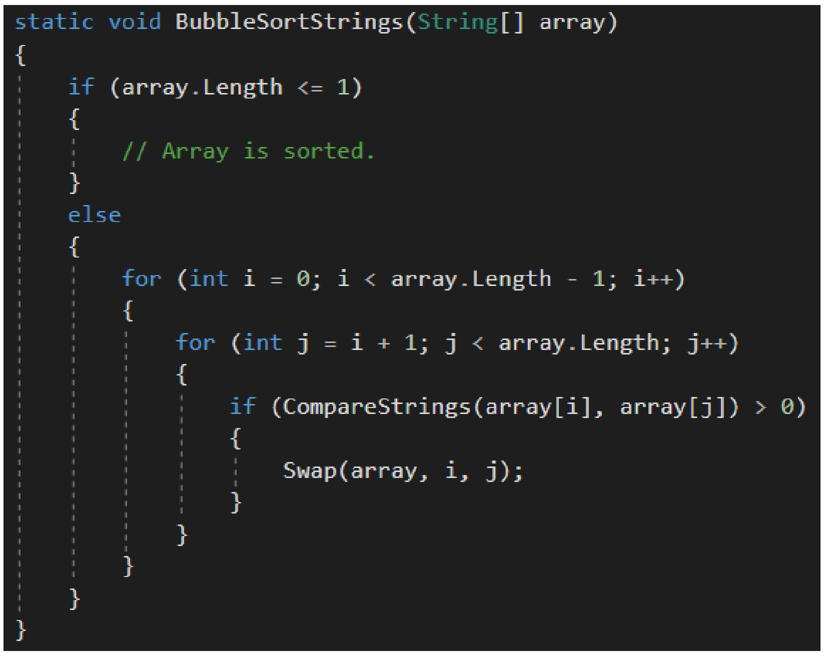
```
private static string RandomStringGenerator(int length, Random random)
{
    char[] letters = "abcdefghijklmnopqrstuvwxyz".ToCharArray();

    string randomString = "";

    for (int i = 0; i < length; i++)
    {
        randomString += letters[random.Next(0, letters.Length)].ToString();
    }

    return randomString;
}
```

Figur 2.6: Compare metoden som sammenligner to strenge



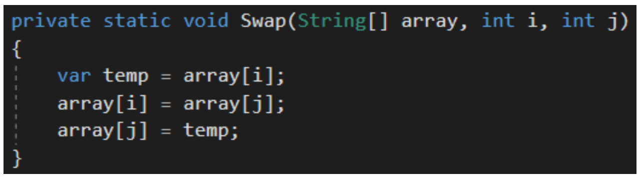
```
private static int CompareStrings(string s1, string s2)
{
    int result = 0;

    int i = 0;
    int j = 0;

    while (i < s1.Length && j < s2.Length)
    {
        int compareResult = s1[i].CompareTo(s2[j]);

        if (compareResult < 0)
        {
            result = -1;
            break;
        }
        else if (compareResult > 0)
        {
            result = 1;
            break;
        }
        else
        {
            i++;
            j++;
        }
    }
    return result;
}
```

Figur 2.7: BubbleSort som sorterer et array af strenge



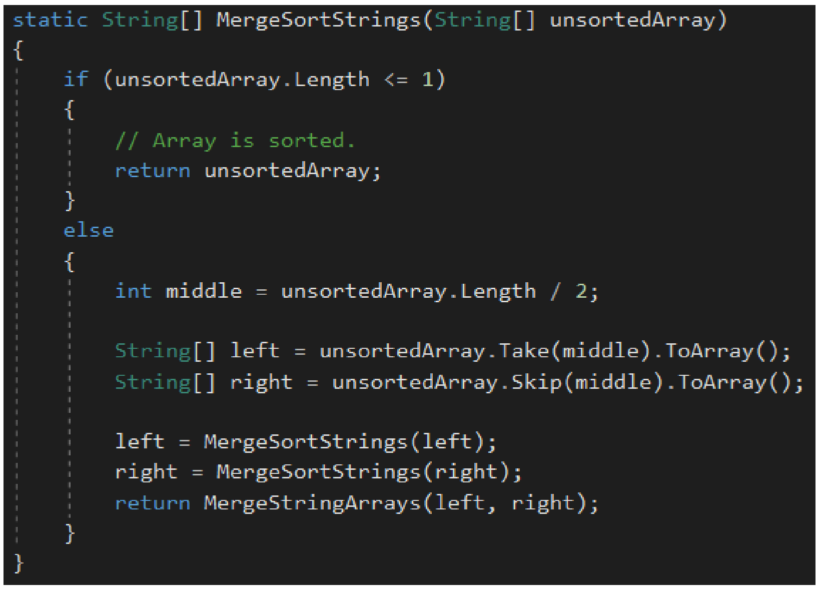
```
static void BubbleSortStrings(String[] array)
{
    if (array.Length <= 1)
    {
        // Array is sorted.
    }
    else
    {
        for (int i = 0; i < array.Length - 1; i++)
        {
            for (int j = i + 1; j < array.Length; j++)
            {
                if (CompareStrings(array[i], array[j]) > 0)
                {
                    Swap(array, i, j);
                }
            }
        }
    }
}
```

Figur 2.8: Hjælpemetoden Swap til BubbleSort metoden som sorterer et array af strenge



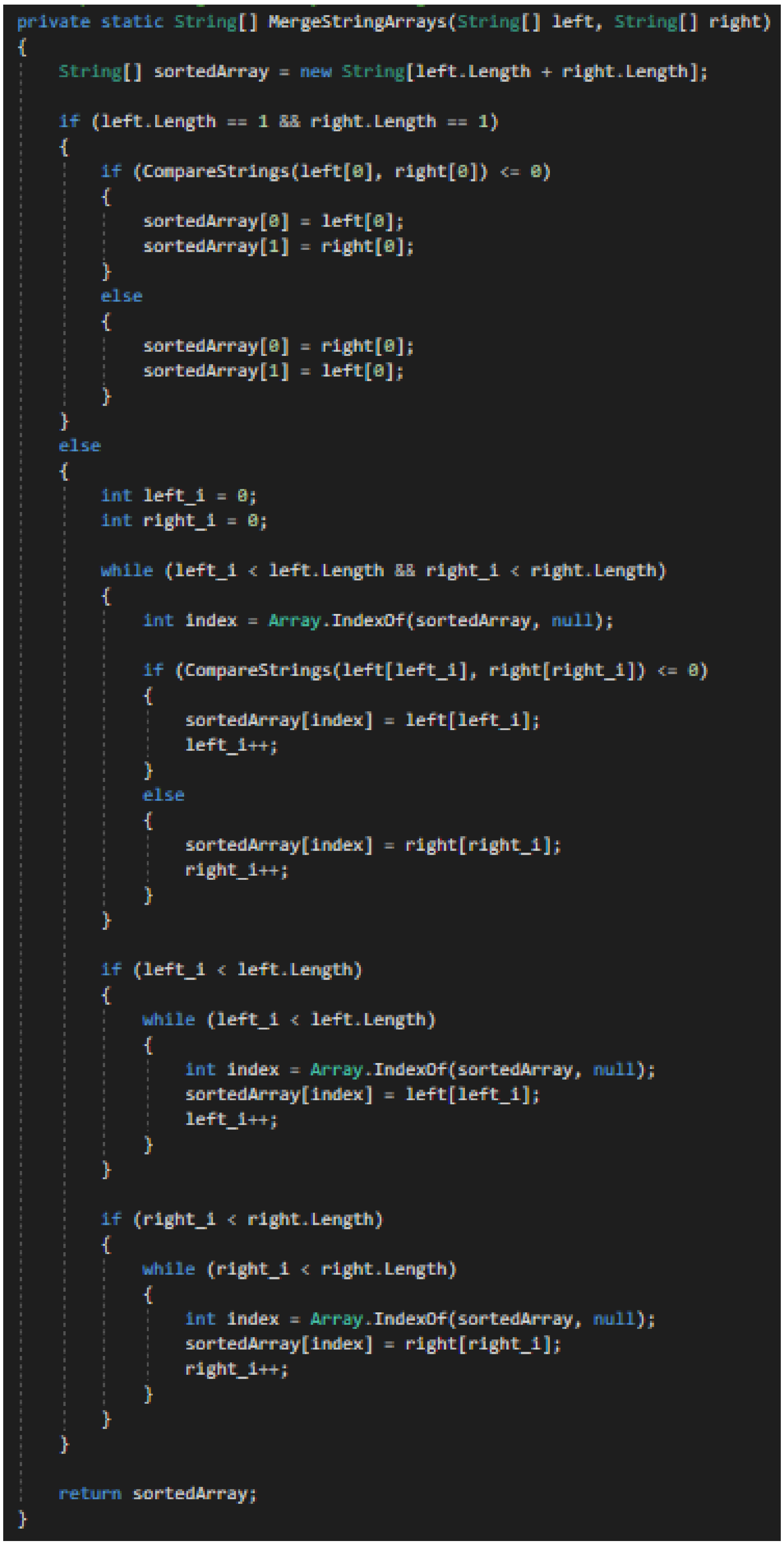
```
private static void Swap(String[] array, int i, int j)
{
    var temp = array[i];
    array[i] = array[j];
    array[j] = temp;
}
```

Figur 2.9: MergeSort som sorterer et array af strenge



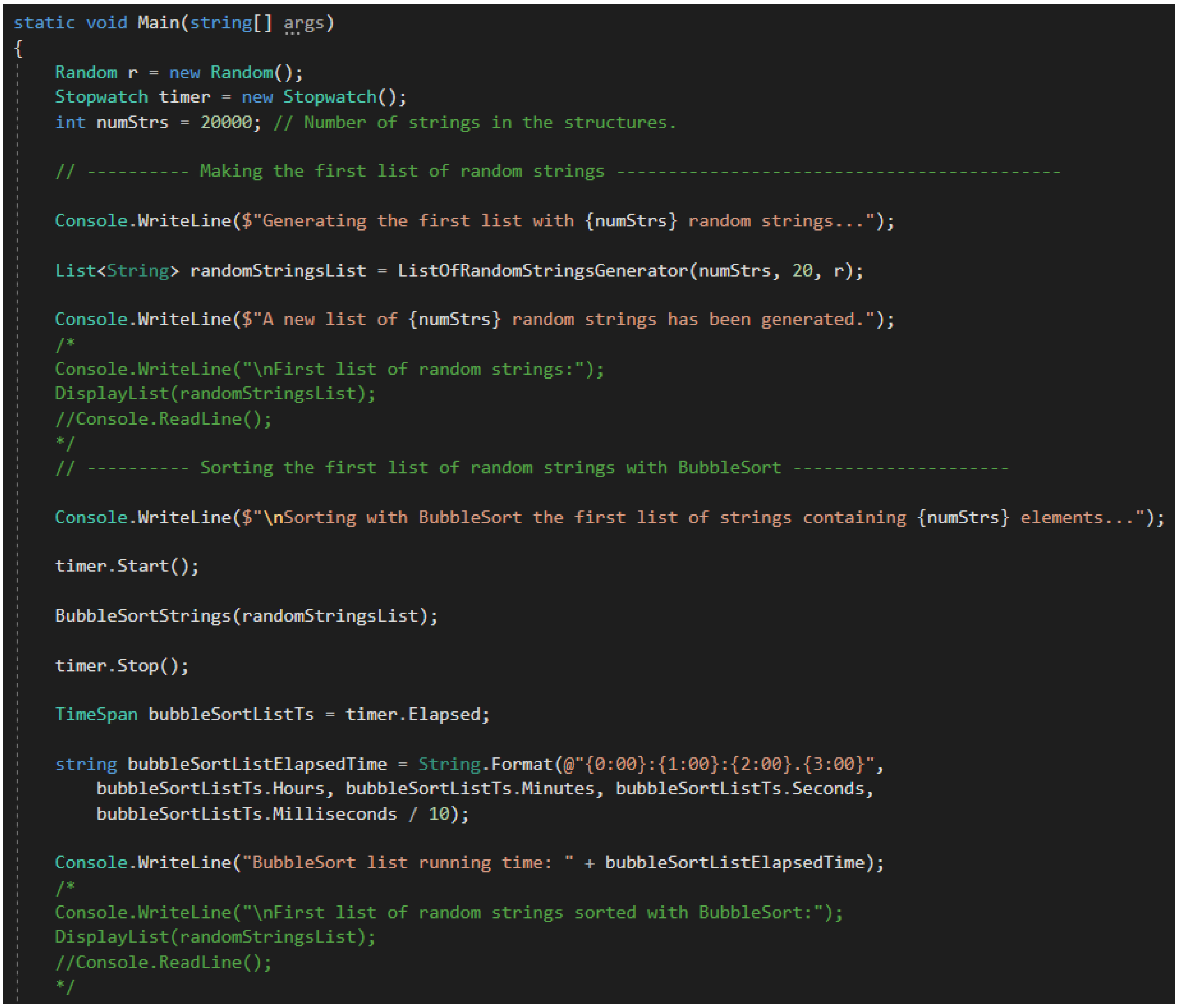
```
static String[] MergeSortStrings(String[] unsortedArray)
{
    if (unsortedArray.Length <= 1)
    {
        // Array is sorted.
        return unsortedArray;
    }
    else
    {
        int middle = unsortedArray.Length / 2;

        String[] left = unsortedArray.Take(middle).ToArray();
        String[] right = unsortedArray.Skip(middle).ToArray();

        left = MergeSortStrings(left);
        right = MergeSortStrings(right);
        return MergeStringArrays(left, right);
    }
}
```

Figur 2.10: Hjælpemetoden MergeSortStrings til MergeSort metoden som sorterer et array af strenge



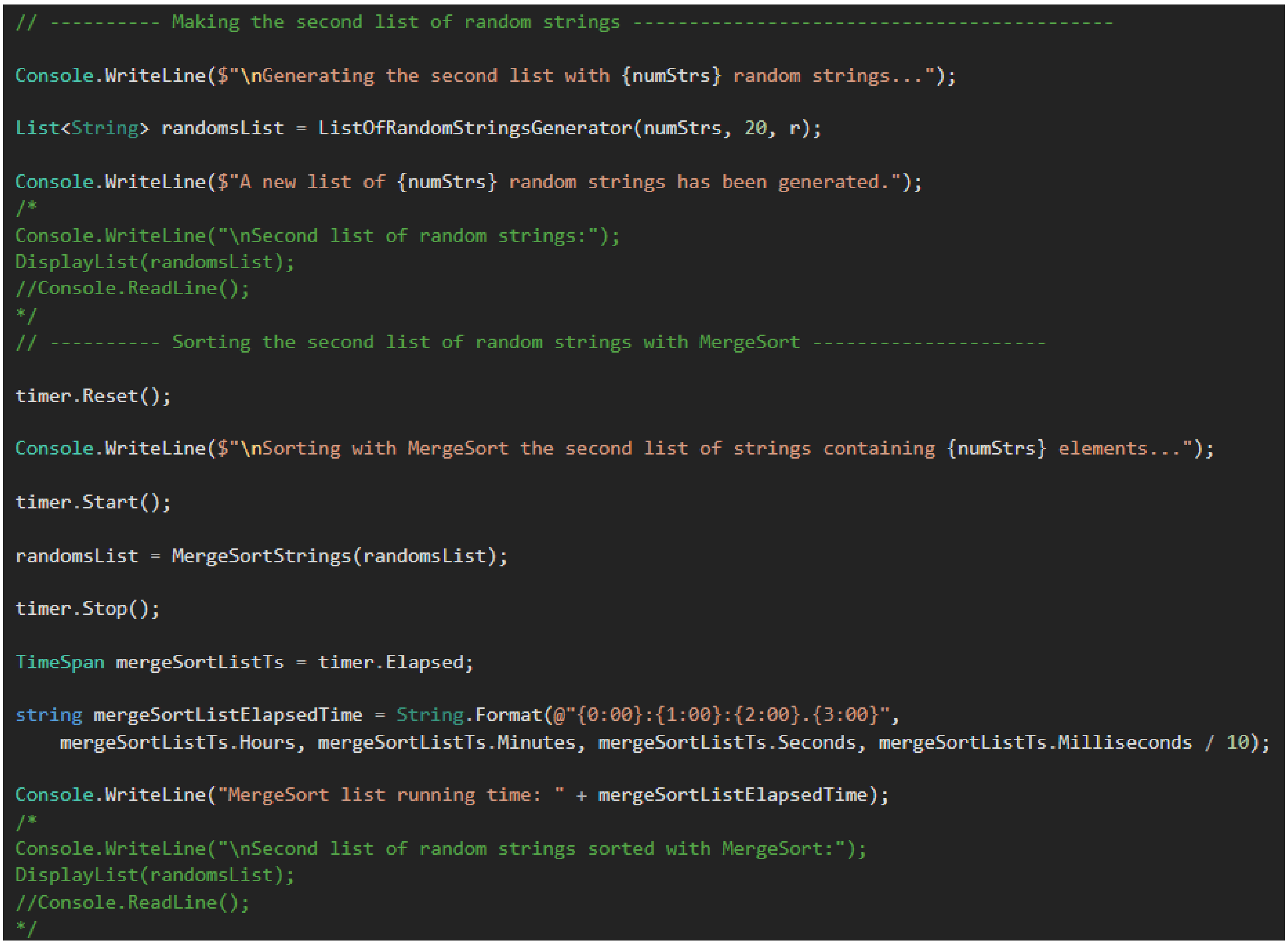
```
private static String[] MergeStringArrays(String[] left, String[] right)
{
    String[] sortedArray = new String[left.Length + right.Length];

    if (left.Length == 1 && right.Length == 1)
    {
        if (CompareStrings(left[0], right[0]) <= 0)
        {
            sortedArray[0] = left[0];
            sortedArray[1] = right[0];
        }
        else
        {
            sortedArray[0] = right[0];
            sortedArray[1] = left[0];
        }
    }
    else
    {
        int left_i = 0;
        int right_i = 0;

        while (left_i < left.Length && right_i < right.Length)
        {
            int index = Array.IndexOf(sortedArray, null);

            if (CompareStrings(left[left_i], right[right_i]) <= 0)
            {
                sortedArray[index] = left[left_i];
                left_i++;
            }
            else
            {
                sortedArray[index] = right[right_i];
                right_i++;
            }
        }

        if (left_i < left.Length)
        {
            while (left_i < left.Length)
            {
                int index = Array.IndexOf(sortedArray, null);
                sortedArray[index] = left[left_i];
                left_i++;
            }
        }

        if (right_i < right.Length)
        {
            while (right_i < right.Length)
            {
                int index = Array.IndexOf(sortedArray, null);
                sortedArray[index] = right[right_i];
                right_i++;
            }
        }
    }

    return sortedArray;
}
```

Figur 2.11: Main metoden med test data til List datastrukturen, del 1 af 3



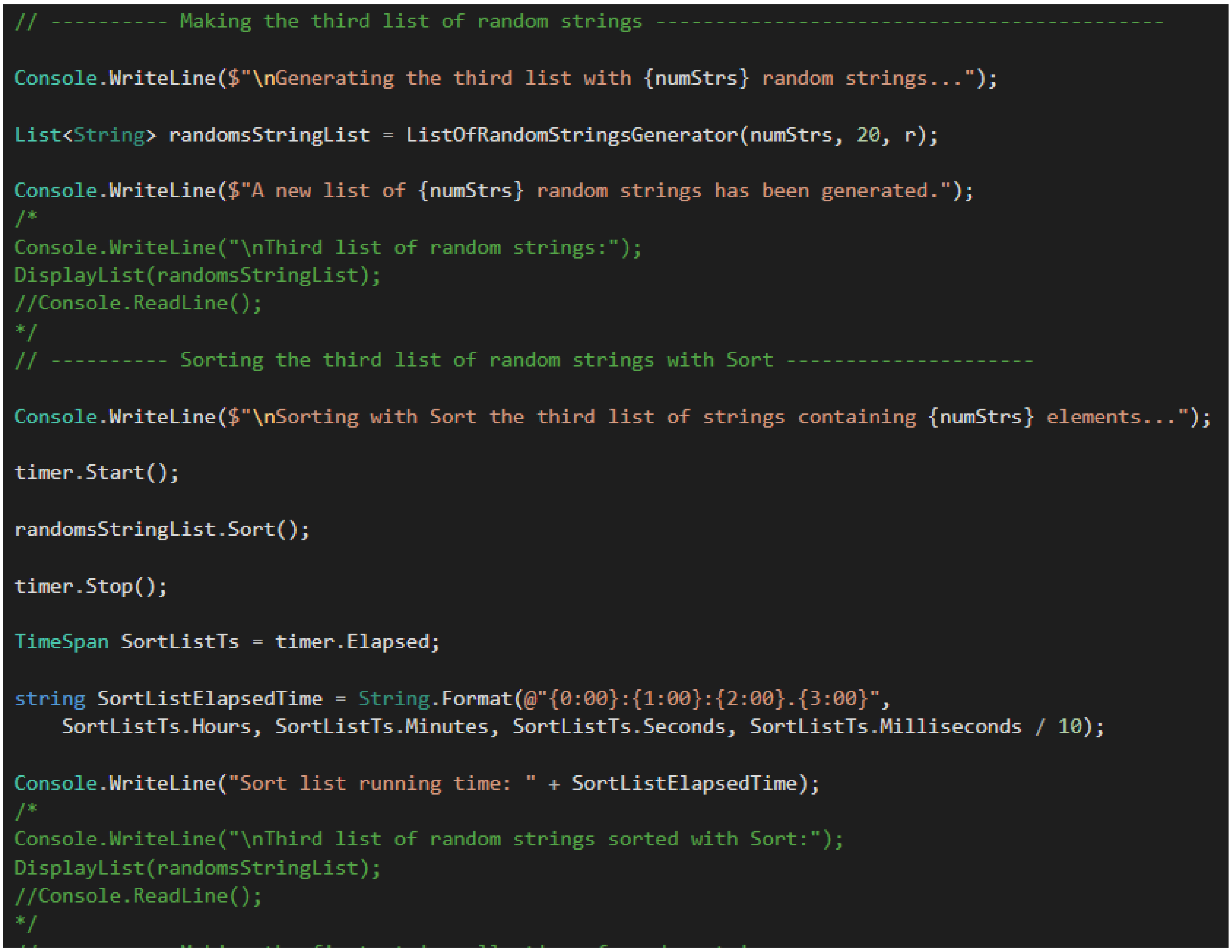
```
static void Main(string[] args)
{
    Random r = new Random();
    Stopwatch timer = new Stopwatch();
    int numStrs = 20000; // Number of strings in the structures.

    // ---------- Making the first list of random strings -------------------------------------------

    Console.WriteLine($"Generating the first list with {numStrs} random strings...");

    List<String> randomStringsList = ListOfRandomStringsGenerator(numStrs, 20, r);

    Console.WriteLine($"A new list of {numStrs} random strings has been generated.");
    /*
    Console.WriteLine("\nFirst list of random strings:");
    DisplayList(randomStringsList);
    //Console.ReadLine();
    */
    // ---------- Sorting the first list of random strings with BubbleSort ---------------------

    Console.WriteLine($"\nSorting with BubbleSort the first list of strings containing {numStrs} elements...");

    timer.Start();

    BubbleSortStrings(randomStringsList);

    timer.Stop();

    TimeSpan bubbleSortListTs = timer.Elapsed;

    string bubbleSortListElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
        bubbleSortListTs.Hours, bubbleSortListTs.Minutes, bubbleSortListTs.Seconds,
        bubbleSortListTs.Milliseconds / 10);

    Console.WriteLine("BubbleSort list running time: " + bubbleSortListElapsedTime);
    /*
    Console.WriteLine("\nFirst list of random strings sorted with BubbleSort:");
    DisplayList(randomStringsList);
    //Console.ReadLine();
    */
```

Figur 2.12: Main metoden med test data til List datastrukturen, del 2 af 3



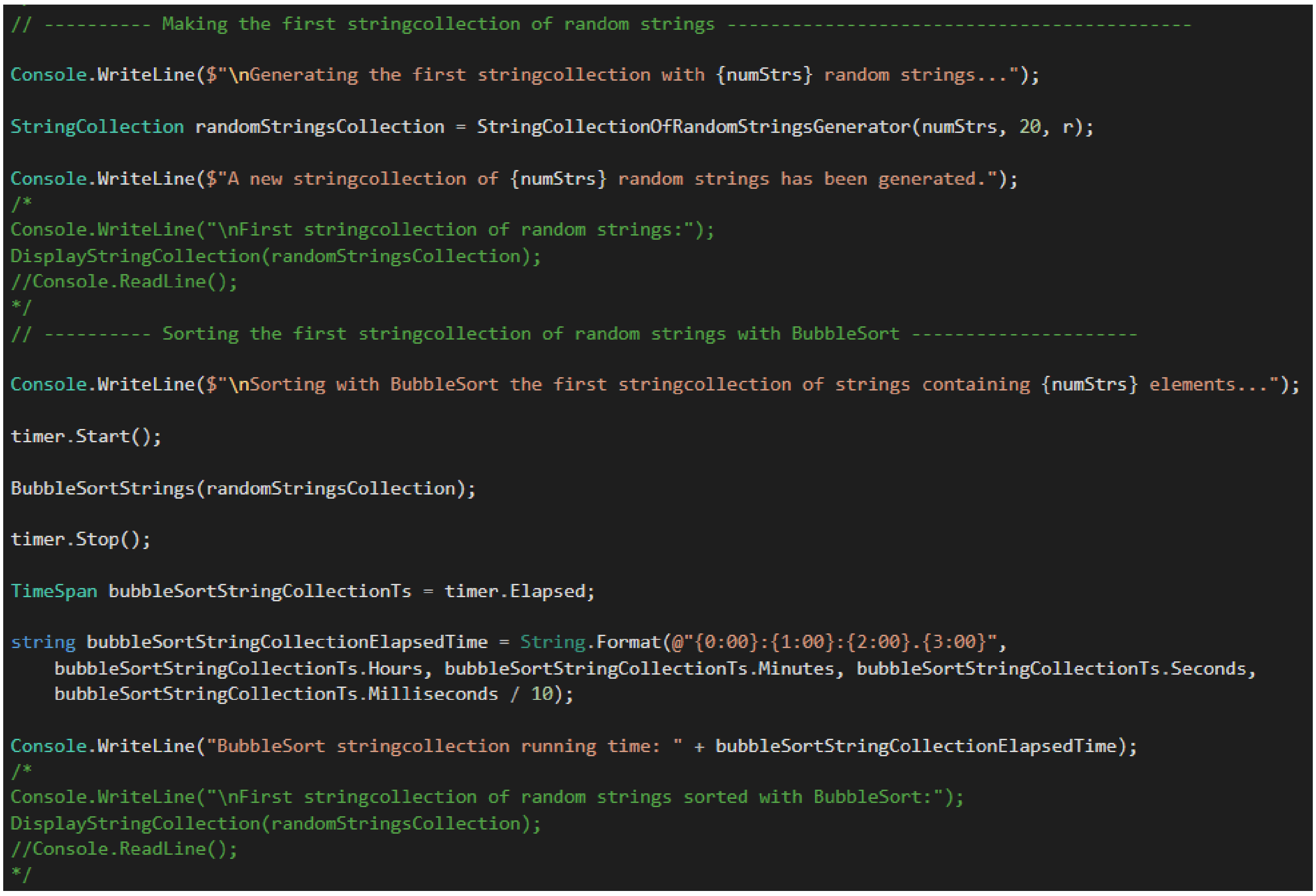
```
// ---------- Making the second list of random strings -------------------------------------------

Console.WriteLine($"\nGenerating the second list with {numStrs} random strings...");

List<String> randomsList = ListOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new list of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nSecond list of random strings:");
DisplayList(randomsList);
//Console.ReadLine();
*/
// ---------- Sorting the second list of random strings with MergeSort ---------------------

timer.Reset();

Console.WriteLine($"\nSorting with MergeSort the second list of strings containing {numStrs} elements...");

timer.Start();

randomsList = MergeSortStrings(randomsList);

timer.Stop();

TimeSpan mergeSortListTs = timer.Elapsed;

string mergeSortListElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    mergeSortListTs.Hours, mergeSortListTs.Minutes, mergeSortListTs.Seconds, mergeSortListTs.Milliseconds / 10);

Console.WriteLine("MergeSort list running time: " + mergeSortListElapsedTime);
/*
Console.WriteLine("\nSecond list of random strings sorted with MergeSort:");
DisplayList(randomsList);
//Console.ReadLine();
*/
```

Figur 2.13: Main metoden med test data til List datastrukturen, del 3 af 3



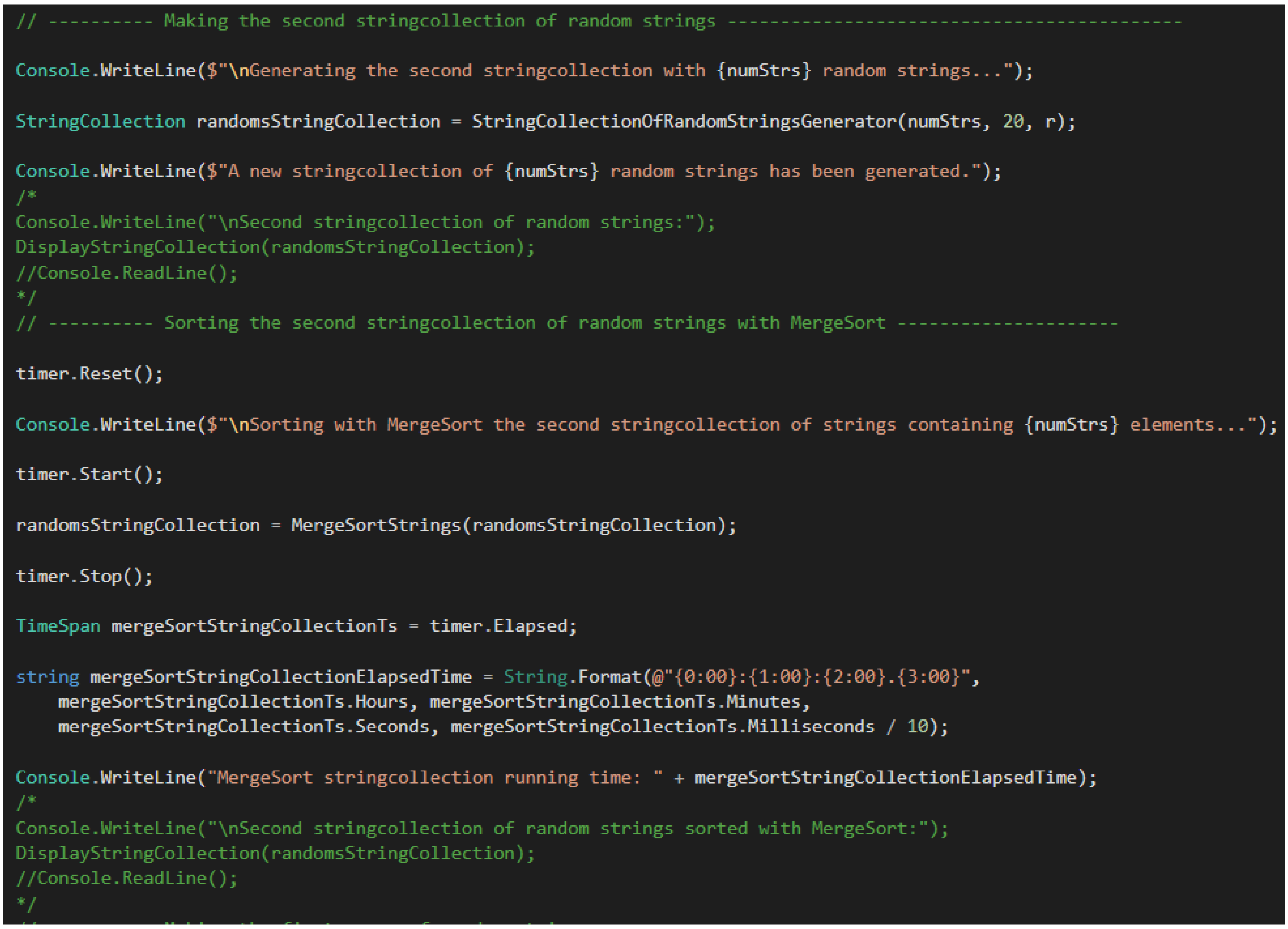
```
// ---------- Making the third list of random strings ------------------------------------------

Console.WriteLine($"\nGenerating the third list with {numStrs} random strings...");

List<String> randomsStringList = ListOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new list of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nThird list of random strings:");
DisplayList(randomsStringList);
//Console.ReadLine();
*/
// ---------- Sorting the third list of random strings with Sort ---------------------

Console.WriteLine($"\nSorting with Sort the third list of strings containing {numStrs} elements...");

timer.Start();

randomsStringList.Sort();

timer.Stop();

TimeSpan SortListTs = timer.Elapsed;

string SortListElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    SortListTs.Hours, SortListTs.Minutes, SortListTs.Seconds, SortListTs.Milliseconds / 10);

Console.WriteLine("Sort list running time:  " + SortListElapsedTime);
/*
Console.WriteLine("\nThird list of random strings sorted with Sort:");
DisplayList(randomsStringList);
//Console.ReadLine();
*/
```

Figur 2.14: Main metoden med test data til StringCollection datastrukturen, del 1 af 2



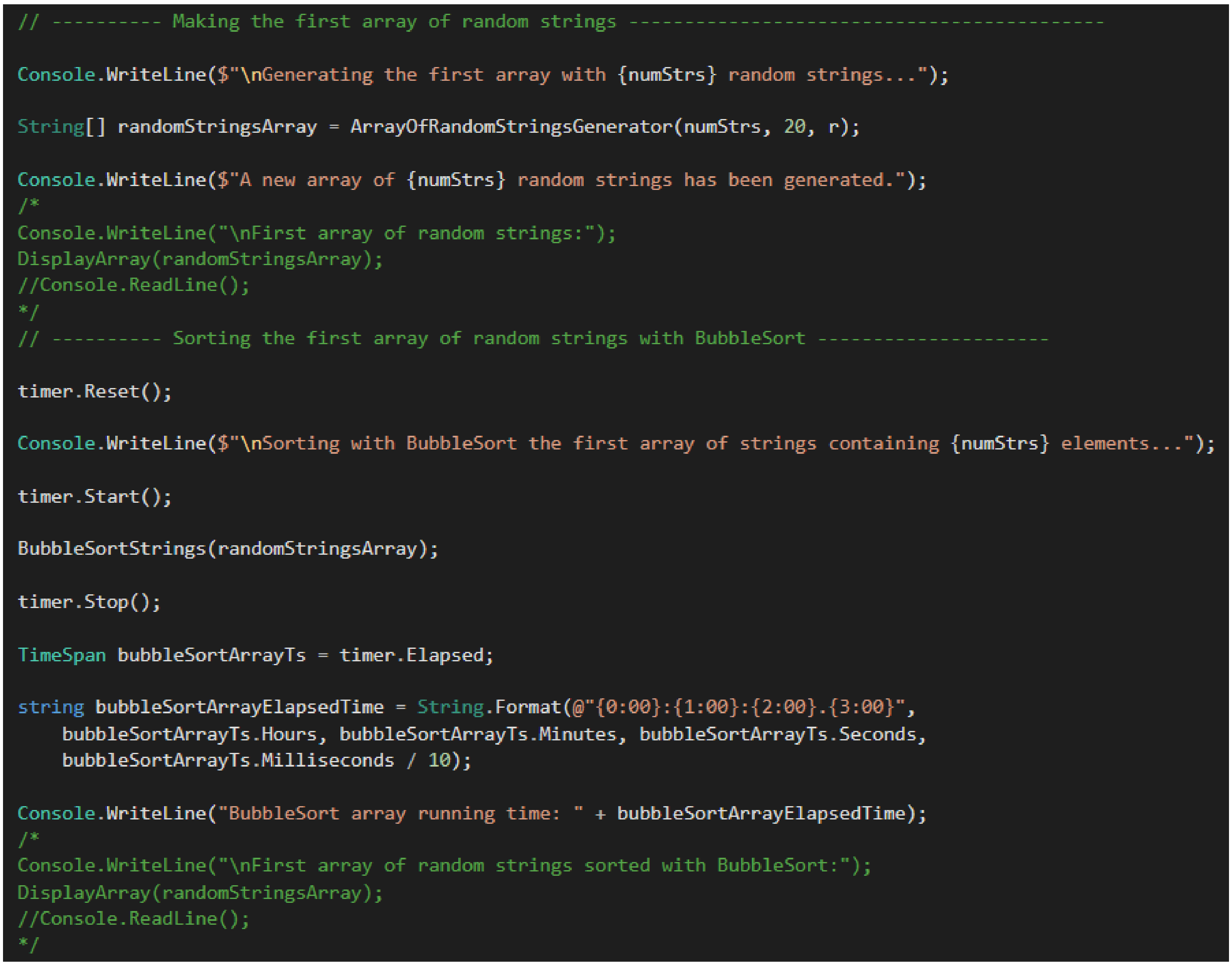
```
// ---------- Making the first stringcollection of random strings ------------------------------------------

Console.WriteLine($"\nGenerating the first stringcollection with {numStrs} random strings...");

StringCollection randomStringsCollection = StringCollectionOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new stringcollection of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nFirst stringcollection of random strings:");
DisplayStringCollection(randomStringsCollection);
//Console.ReadLine();
*/
// ---------- Sorting the first stringcollection of random strings with BubbleSort ---------------------

Console.WriteLine($"\nSorting with BubbleSort the first stringcollection of strings containing {numStrs} elements...");

timer.Start();

BubbleSortStrings(randomStringsCollection);

timer.Stop();

TimeSpan bubbleSortStringCollectionTs = timer.Elapsed;

string bubbleSortStringCollectionElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    bubbleSortStringCollectionTs.Hours, bubbleSortStringCollectionTs.Minutes, bubbleSortStringCollectionTs.Seconds,
    bubbleSortStringCollectionTs.Milliseconds / 10);

Console.WriteLine("BubbleSort stringcollection running time: " + bubbleSortStringCollectionElapsedTime);
/*
Console.WriteLine("\nFirst stringcollection of random strings sorted with BubbleSort:");
DisplayStringCollection(randomStringsCollection);
//Console.ReadLine();
*/
```

Figur 2.15: Main metoden med test data til StringCollection datastrukturen, del 2 af 2



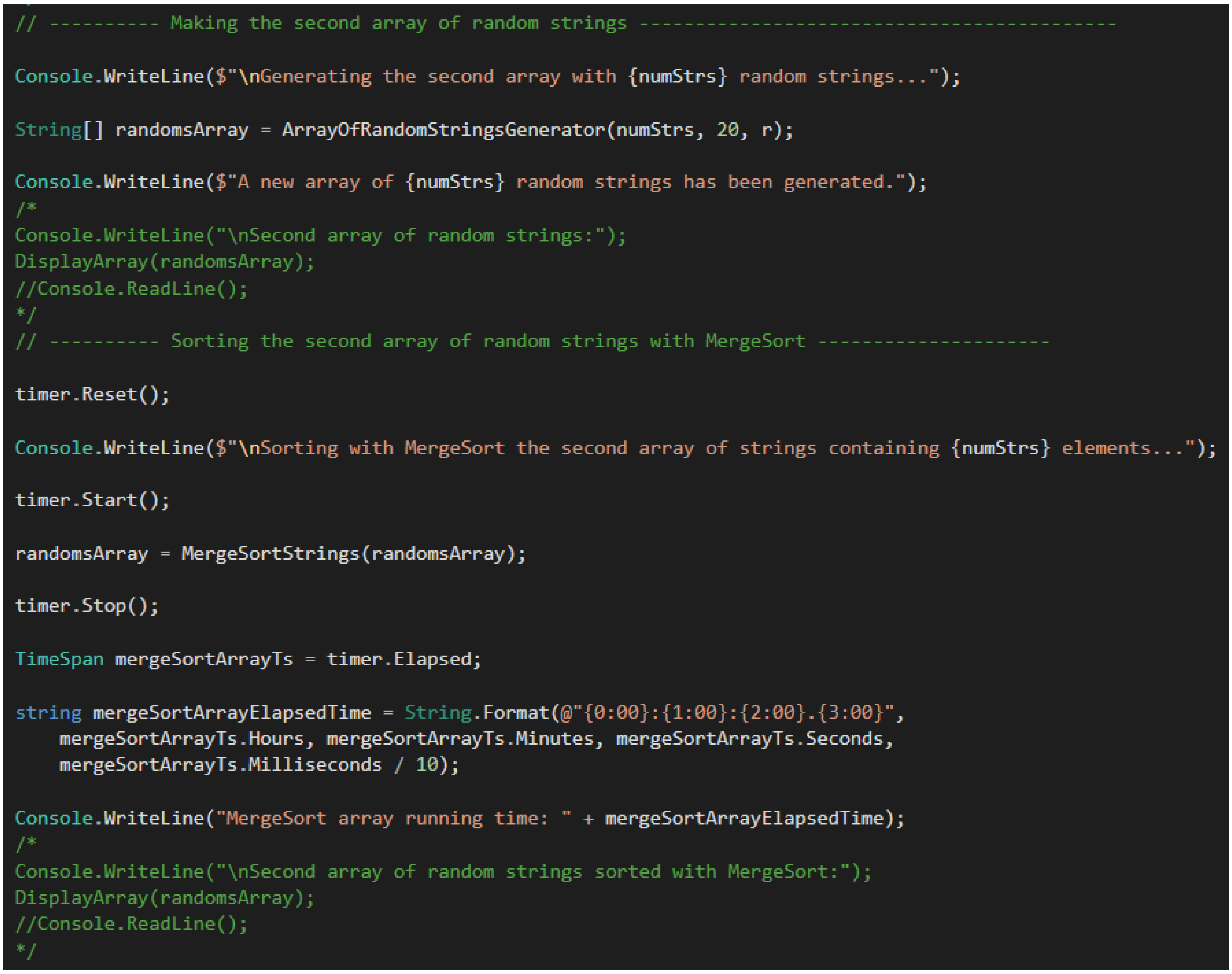
```
// ---------- Making the second stringcollection of random strings ------------------------------------------

Console.WriteLine($"\nGenerating the second stringcollection with {numStrs} random strings...");

StringCollection randomsStringCollection = StringCollectionOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new stringcollection of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nSecond stringcollection of random strings:");
DisplayStringCollection(randomsStringCollection);
//Console.ReadLine();
*/
// ---------- Sorting the second stringcollection of random strings with MergeSort ---------------------

timer.Reset();

Console.WriteLine($"\nSorting with MergeSort the second stringcollection of strings containing {numStrs} elements...");

timer.Start();

randomsStringCollection = MergeSortStrings(randomsStringCollection);

timer.Stop();

TimeSpan mergeSortStringCollectionTs = timer.Elapsed;

string mergeSortStringCollectionElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    mergeSortStringCollectionTs.Hours, mergeSortStringCollectionTs.Minutes,
    mergeSortStringCollectionTs.Seconds, mergeSortStringCollectionTs.Milliseconds / 10);

Console.WriteLine("MergeSort stringcollection running time: " + mergeSortStringCollectionElapsedTime);
/*
Console.WriteLine("\nSecond stringcollection of random strings sorted with MergeSort:");
DisplayStringCollection(randomsStringCollection);
//Console.ReadLine();
*/
```

Figur 2.16: Main metoden med test data til Array datastrukturen, del 1 af 3



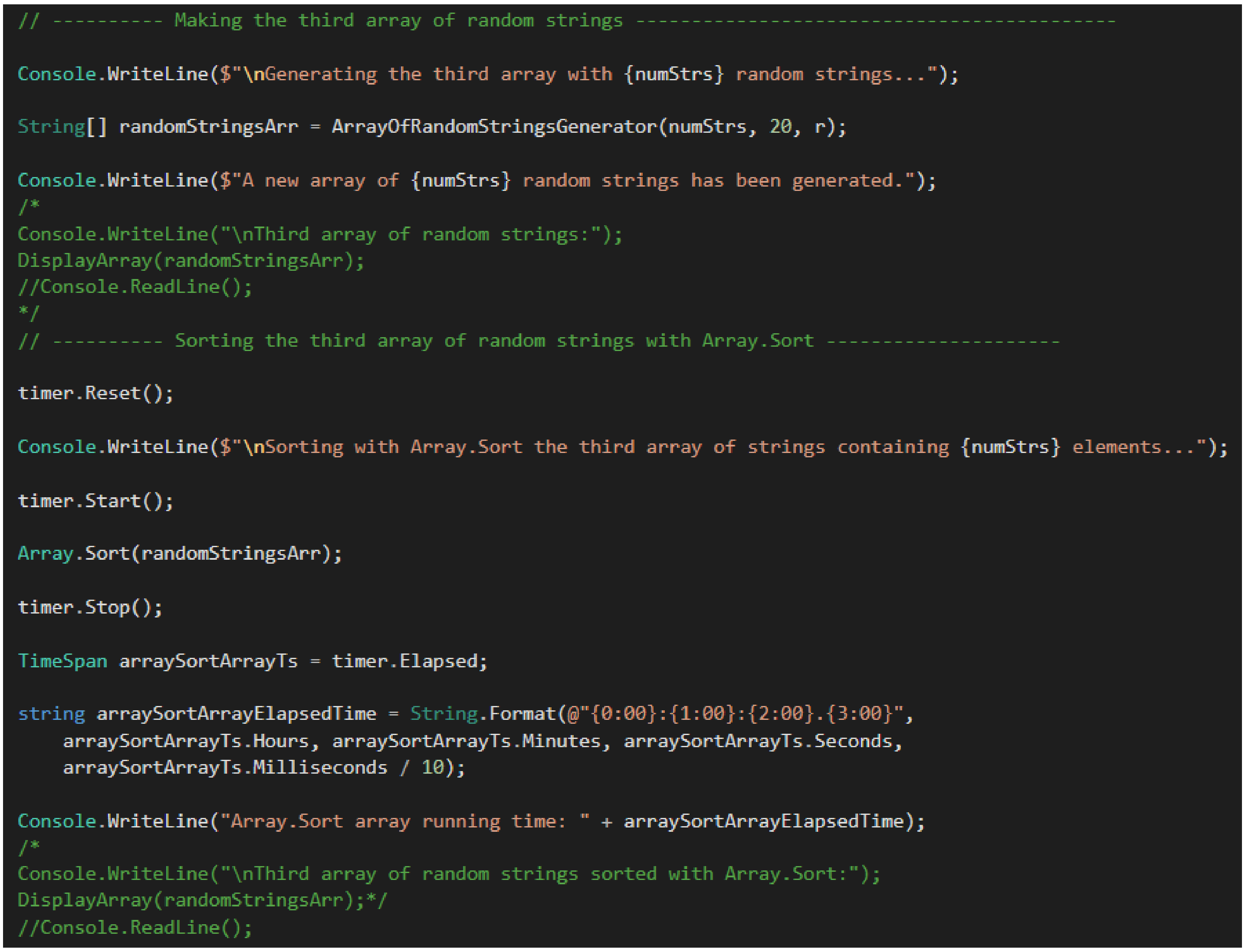
```
// ---------- Making the first array of random strings -------------------------------------------

Console.WriteLine($"\nGenerating the first array with {numStrs} random strings...");

String[] randomStringsArray = ArrayOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new array of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nFirst array of random strings:");
DisplayArray(randomStringsArray);
//Console.ReadLine();
*/
// ---------- Sorting the first array of random strings with BubbleSort ---------------------

timer.Reset();

Console.WriteLine($"\nSorting with BubbleSort the first array of strings containing {numStrs} elements...");

timer.Start();

BubbleSortStrings(randomStringsArray);

timer.Stop();

TimeSpan bubbleSortArrayTs = timer.Elapsed;

string bubbleSortArrayElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    bubbleSortArrayTs.Hours, bubbleSortArrayTs.Minutes, bubbleSortArrayTs.Seconds,
    bubbleSortArrayTs.Milliseconds / 10);

Console.WriteLine("BubbleSort array running time: " + bubbleSortArrayElapsedTime);
/*
Console.WriteLine("\nFirst array of random strings sorted with BubbleSort:");
DisplayArray(randomStringsArray);
//Console.ReadLine();
*/
```

Figur 2.17: Main metoden med test data til Array datastrukturen, del 2 af 3



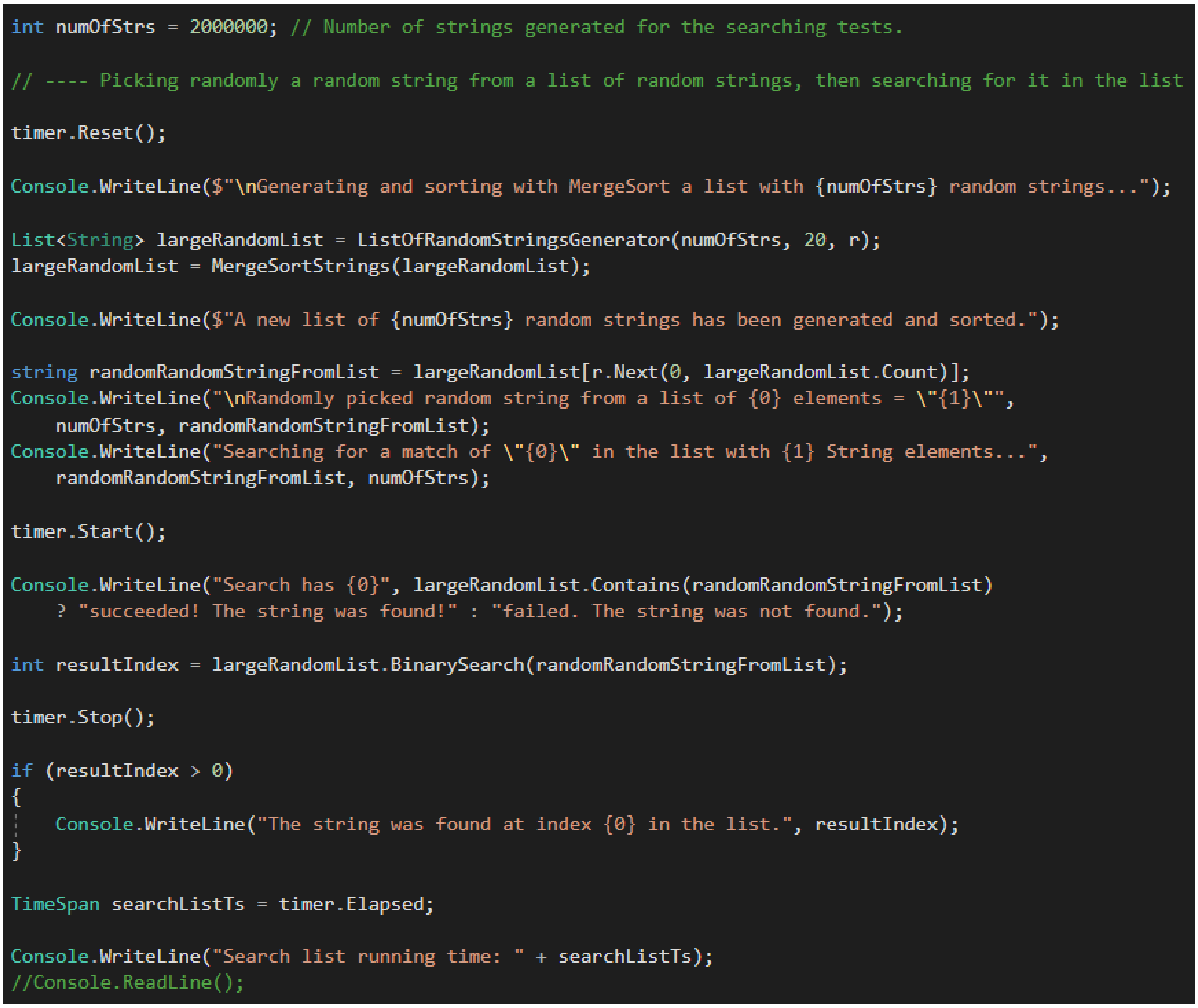
```
// ---------- Making the second array of random strings -------------------------------------------

Console.WriteLine($"\nGenerating the second array with {numStrs} random strings...");

String[] randomsArray = ArrayOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new array of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nSecond array of random strings:");
DisplayArray(randomsArray);
//Console.ReadLine();
*/
// ---------- Sorting the second array of random strings with MergeSort ---------------------

timer.Reset();

Console.WriteLine($"\nSorting with MergeSort the second array of strings containing {numStrs} elements...");

timer.Start();

randomsArray = MergeSortStrings(randomsArray);

timer.Stop();

TimeSpan mergeSortArrayTs = timer.Elapsed;

string mergeSortArrayElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    mergeSortArrayTs.Hours, mergeSortArrayTs.Minutes, mergeSortArrayTs.Seconds,
    mergeSortArrayTs.Milliseconds / 10);

Console.WriteLine("MergeSort array running time: " + mergeSortArrayElapsedTime);
/*
Console.WriteLine("\nSecond array of random strings sorted with MergeSort:");
DisplayArray(randomsArray);
//Console.ReadLine();
*/
```

Figur 2.18: Main metoden med test data til Array datastrukturen, del 3 af 3



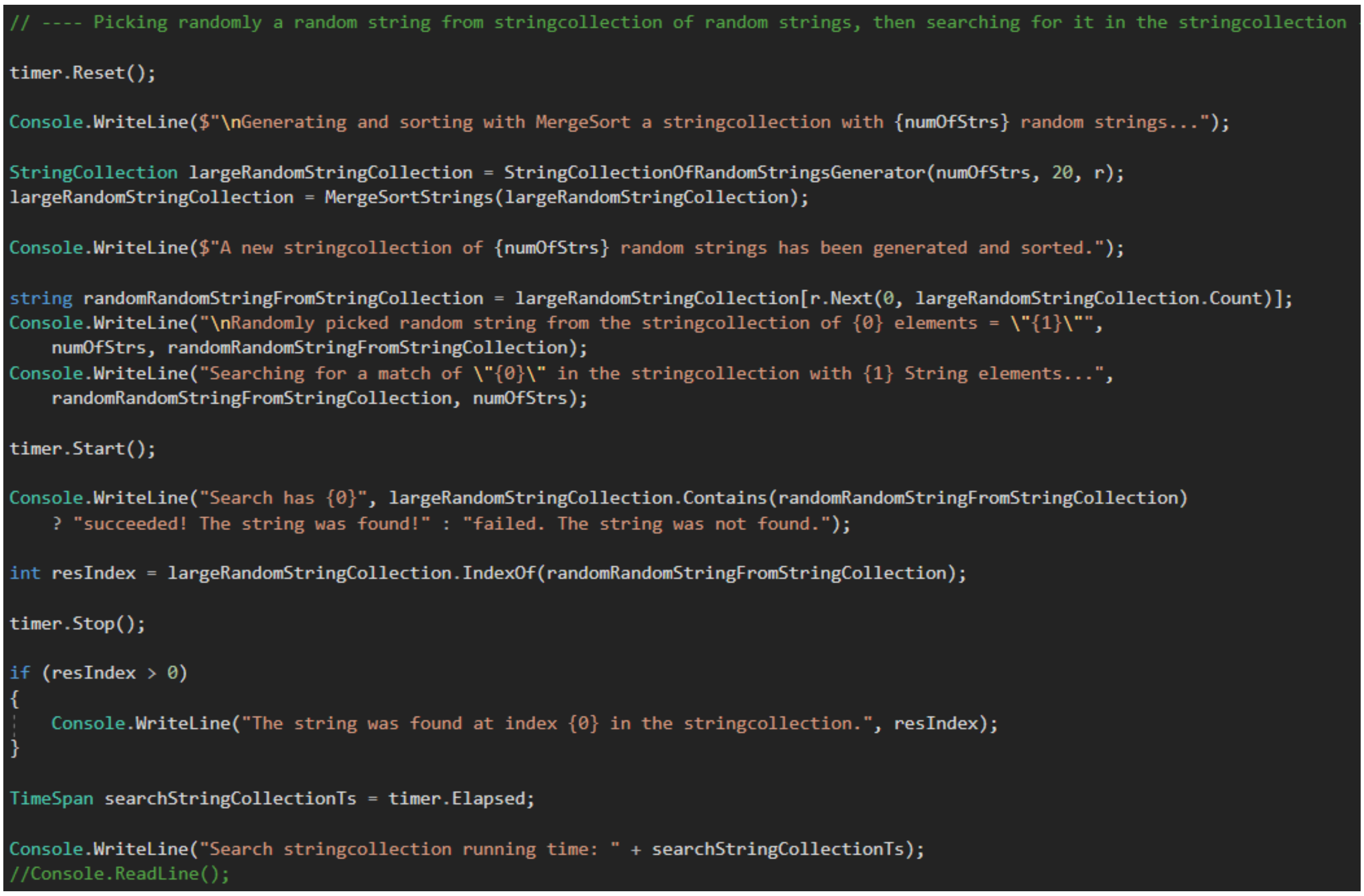
```
// ---------- Making the third array of random strings -------------------------------------------

Console.WriteLine($"\nGenerating the third array with {numStrs} random strings...");

String[] randomStringsArr = ArrayOfRandomStringsGenerator(numStrs, 20, r);

Console.WriteLine($"A new array of {numStrs} random strings has been generated.");
/*
Console.WriteLine("\nThird array of random strings:");
DisplayArray(randomStringsArr);
//Console.ReadLine();
*/
// ---------- Sorting the third array of random strings with Array.Sort ---------------------

timer.Reset();

Console.WriteLine($"\nSorting with Array.Sort the third array of strings containing {numStrs} elements...");

timer.Start();

Array.Sort(randomStringsArr);

timer.Stop();

TimeSpan arraySortArrayTs = timer.Elapsed;

string arraySortArrayElapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    arraySortArrayTs.Hours, arraySortArrayTs.Minutes, arraySortArrayTs.Seconds,
    arraySortArrayTs.Milliseconds / 10);

Console.WriteLine("Array.Sort array running time: " + arraySortArrayElapsedTime);
/*
Console.WriteLine("\nThird array of random strings sorted with Array.Sort:");
DisplayArray(randomStringsArr);*/
//Console.ReadLine();
```

Figur 2.19: Main metoden med test data til søgning i datastrukturerne, del 1 af 3



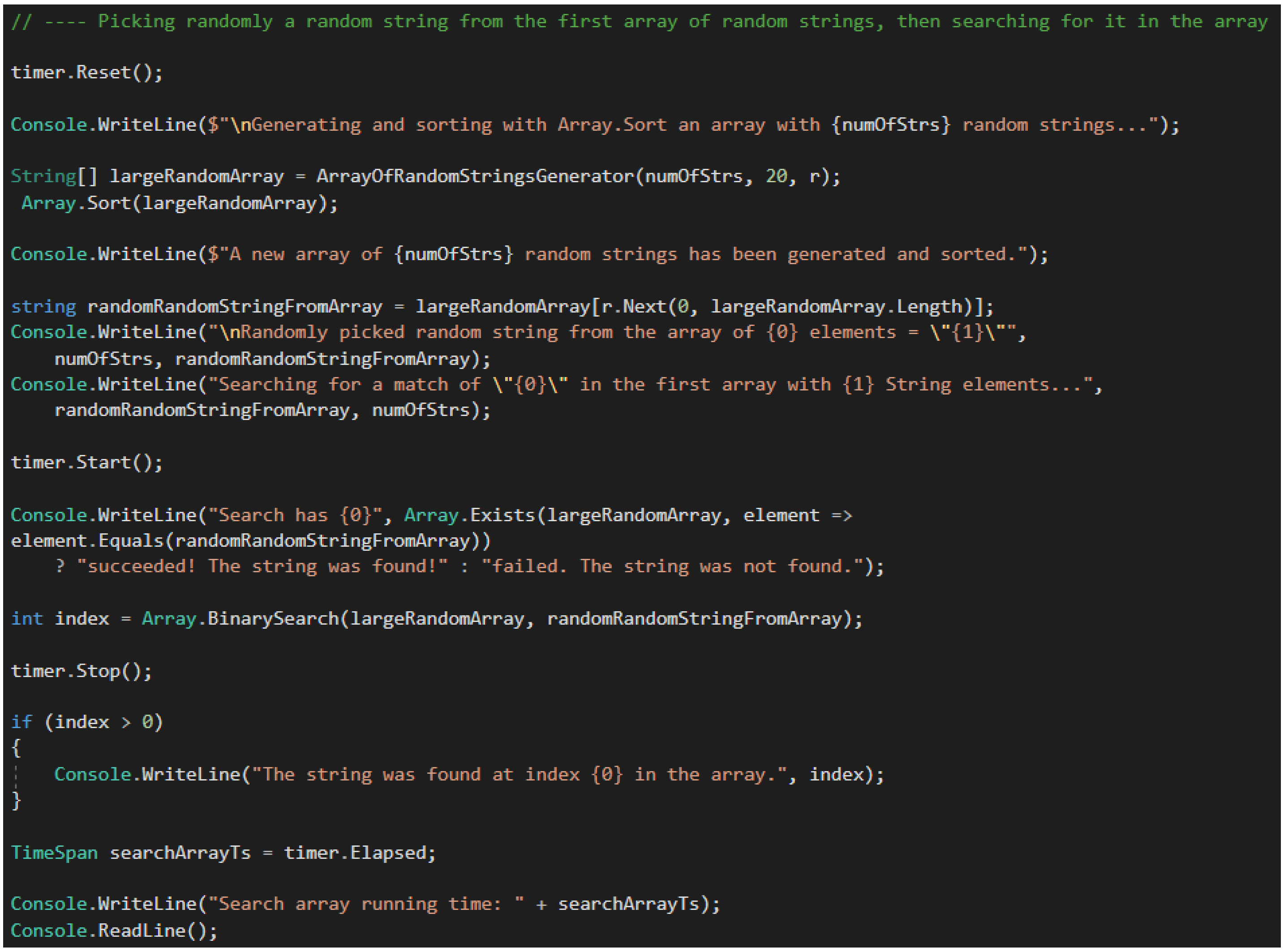
```
int numOfStrs = 2000000; // Number of strings generated for the searching tests.

// ---- Picking randomly a random string from a list of random strings, then searching for it in the list

timer.Reset();

Console.WriteLine($"\nGenerating and sorting with MergeSort a list with {numOfStrs} random strings...");

List<String> largeRandomList = ListOfRandomStringsGenerator(numOfStrs, 20, r);
largeRandomList = MergeSortStrings(largeRandomList);

Console.WriteLine($"A new list of {numOfStrs} random strings has been generated and sorted.");

string randomRandomStringFromList = largeRandomList[r.Next(0, largeRandomList.Count)];
Console.WriteLine("\nRandomly picked random string from a list of {0} elements = \"{1}\"",
    numOfStrs, randomRandomStringFromList);
Console.WriteLine("Searching for a match of \"{0}\" in the list with {1} String elements...",
    randomRandomStringFromList, numOfStrs);

timer.Start();

Console.WriteLine("Search has {0}", largeRandomList.Contains(randomRandomStringFromList)
    ? "succeeded! The string was found!" : "failed. The string was not found.");

int resultIndex = largeRandomList.BinarySearch(randomRandomStringFromList);

timer.Stop();

if (resultIndex > 0)
{
    Console.WriteLine("The string was found at index {0} in the list.", resultIndex);
}

TimeSpan searchListTs = timer.Elapsed;

Console.WriteLine("Search list running time: " + searchListTs);
//Console.ReadLine();
```

Figur 2.20: Main metoden med test data til søgning i datastrukturerne, del 2 af 3



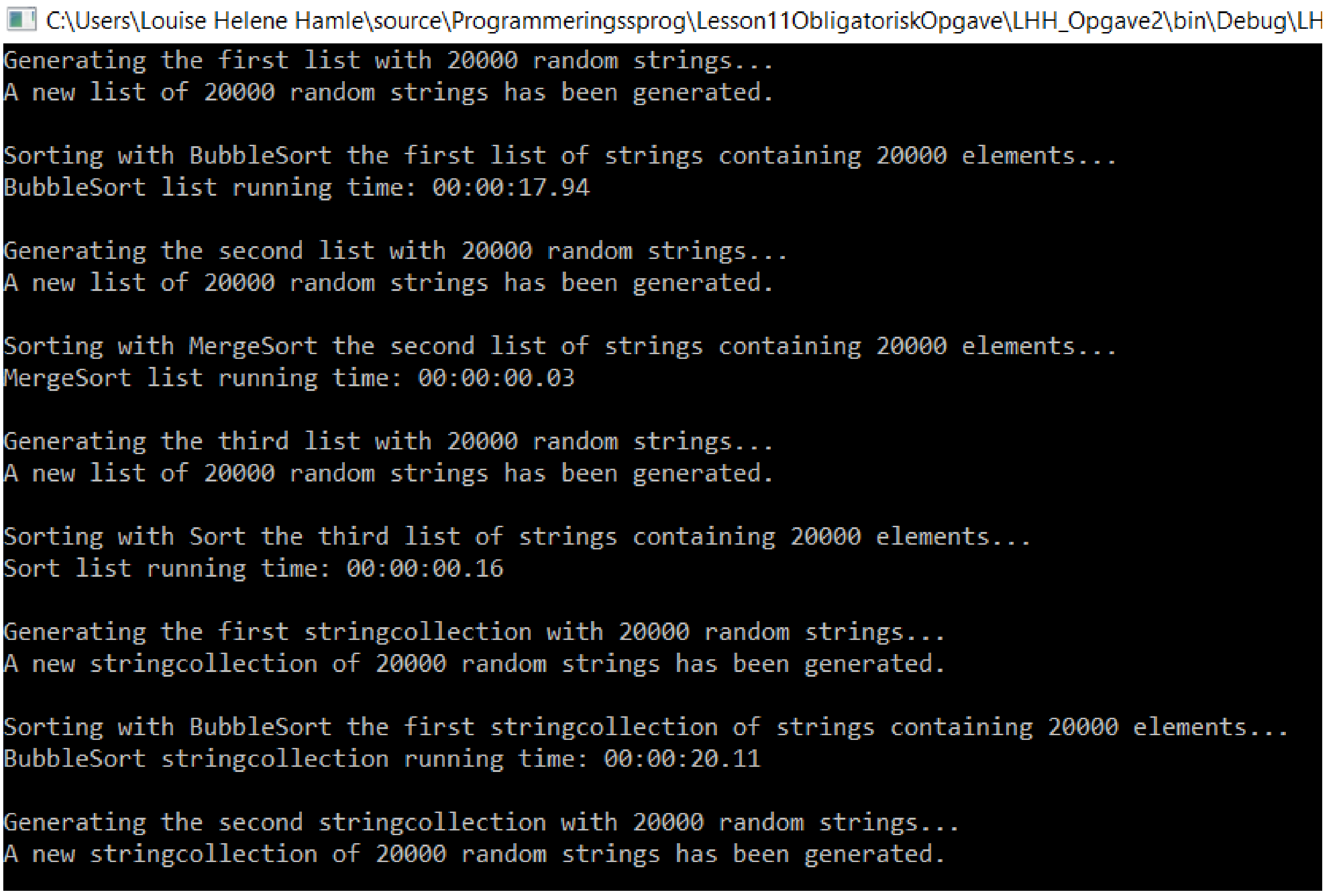
```
// ---- Picking randomly a random string from stringcollection of random strings, then searching for it in the stringcollection

timer.Reset();

Console.WriteLine($"\nGenerating and sorting with MergeSort a stringcollection with {numOfStrs} random strings...");

StringCollection largeRandomStringCollection = StringCollectionOfRandomStringsGenerator(numOfStrs, 20, r);
largeRandomStringCollection = MergeSortStrings(largeRandomStringCollection);

Console.WriteLine($"A new stringcollection of {numOfStrs} random strings has been generated and sorted.");

string randomRandomStringFromStringCollection = largeRandomStringCollection[r.Next(0, largeRandomStringCollection.Count)];
Console.WriteLine("\nRandomly picked random string from the stringcollection of {0} elements = \"{1}\"",
    numOfStrs, randomRandomStringFromStringCollection);
Console.WriteLine("Searching for a match of \"{0}\" in the stringcollection with {1} String elements...",
    randomRandomStringFromStringCollection, numOfStrs);

timer.Start();

Console.WriteLine("Search has {0}", largeRandomStringCollection.Contains(randomRandomStringFromStringCollection)
    ? "succeeded! The string was found!" : "failed. The string was not found.");

int resIndex = largeRandomStringCollection.IndexOf(randomRandomStringFromStringCollection);

timer.Stop();

if (resIndex > 0)
{
    Console.WriteLine("The string was found at index {0} in the stringcollection.", resIndex);
}

TimeSpan searchStringCollectionTs = timer.Elapsed;

Console.WriteLine("Search stringcollection running time: " + searchStringCollectionTs);
//Console.ReadLine();
```

Figur 2.21: Main metoden med test data til søgning i datastrukturerne, del 3 af 3



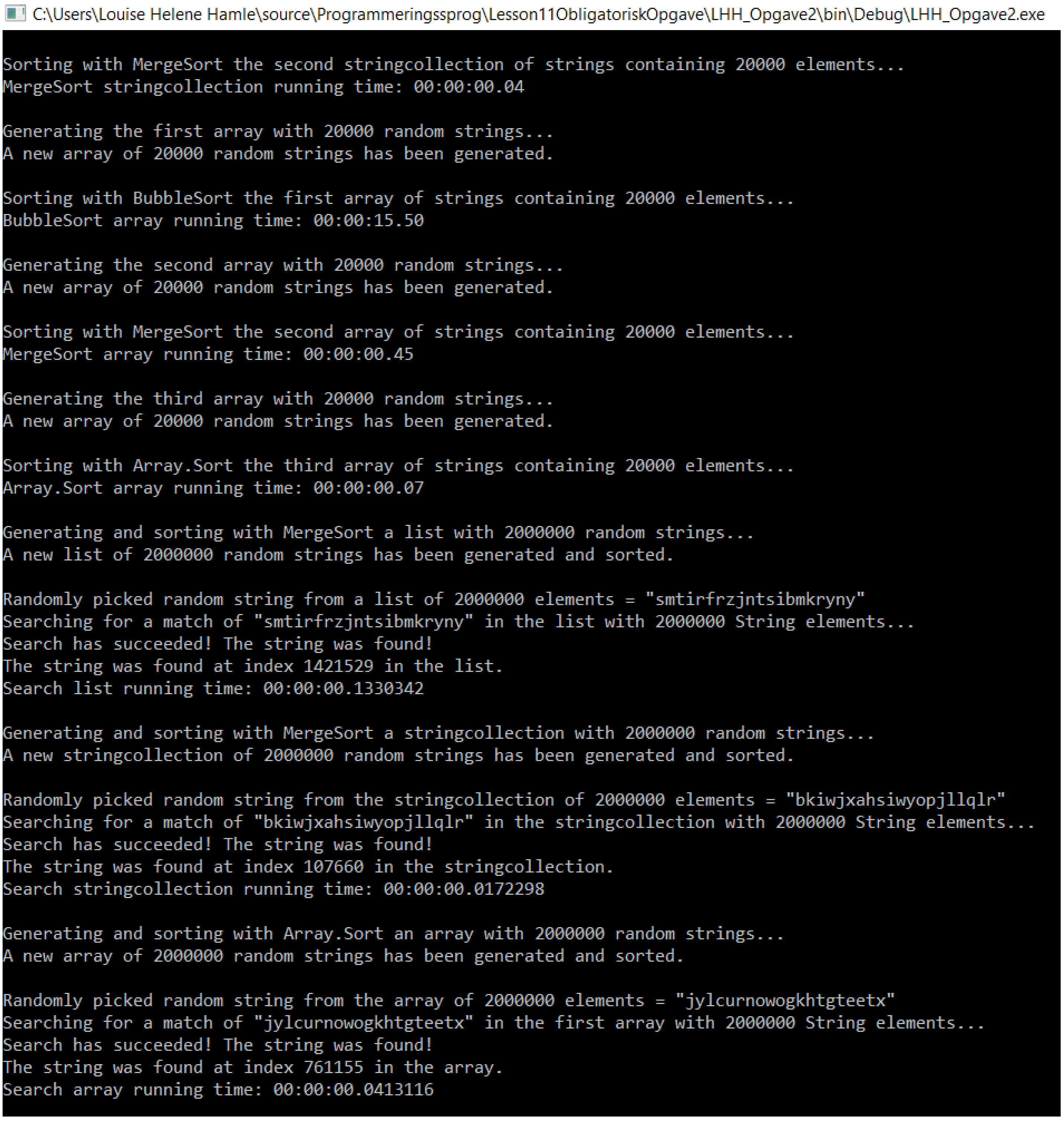
```
// ---- Picking randomly a random string from the first array of random strings, then searching for it in the array

timer.Reset();

Console.WriteLine($"\nGenerating and sorting with Array.Sort an array with {numOfStrs} random strings...");

String[] largeRandomArray = ArrayOfRandomStringsGenerator(numOfStrs, 20, r);
 Array.Sort(largeRandomArray);

Console.WriteLine($"A new array of {numOfStrs} random strings has been generated and sorted.");

string randomRandomStringFromArray = largeRandomArray[r.Next(0, largeRandomArray.Length)];
Console.WriteLine("\nRandomly picked random string from the array of {0} elements = \"{1}\"",
    numOfStrs, randomRandomStringFromArray);
Console.WriteLine("Searching for a match of \"{0}\" in the first array with {1} String elements...",
    randomRandomStringFromArray, numOfStrs);

timer.Start();

Console.WriteLine("Search has {0}", Array.Exists(largeRandomArray, element =>
element.Equals(randomRandomStringFromArray))
    ? "succeeded! The string was found!" : "failed. The string was not found.");

int index = Array.BinarySearch(largeRandomArray, randomRandomStringFromArray);

timer.Stop();

if (index > 0)
{
    Console.WriteLine("The string was found at index {0} in the array.", index);
}

TimeSpan searchArrayTs = timer.Elapsed;

Console.WriteLine("Search array running time: " + searchArrayTs);
Console.ReadLine();
```

Figur 2.22: Output data fra Main metoden, del 1 af 2


C:\Users\Louise Helene Hamle\source\Programmeringssprog\Lesson11ObligatoriskOpgave\LHH_Opgave2\bin\Debug\LH

```
Generating the first list with 20000 random strings...
A new list of 20000 random strings has been generated.

Sorting with BubbleSort the first list of strings containing 20000 elements...
BubbleSort list running time: 00:00:17.94

Generating the second list with 20000 random strings...
A new list of 20000 random strings has been generated.

Sorting with MergeSort the second list of strings containing 20000 elements...
MergeSort list running time: 00:00:00.03

Generating the third list with 20000 random strings...
A new list of 20000 random strings has been generated.

Sorting with Sort the third list of strings containing 20000 elements...
Sort list running time: 00:00:00.16

Generating the first stringcollection with 20000 random strings...
A new stringcollection of 20000 random strings has been generated.

Sorting with BubbleSort the first stringcollection of strings containing 20000 elements...
BubbleSort stringcollection running time: 00:00:20.11

Generating the second stringcollection with 20000 random strings...
A new stringcollection of 20000 random strings has been generated.
```

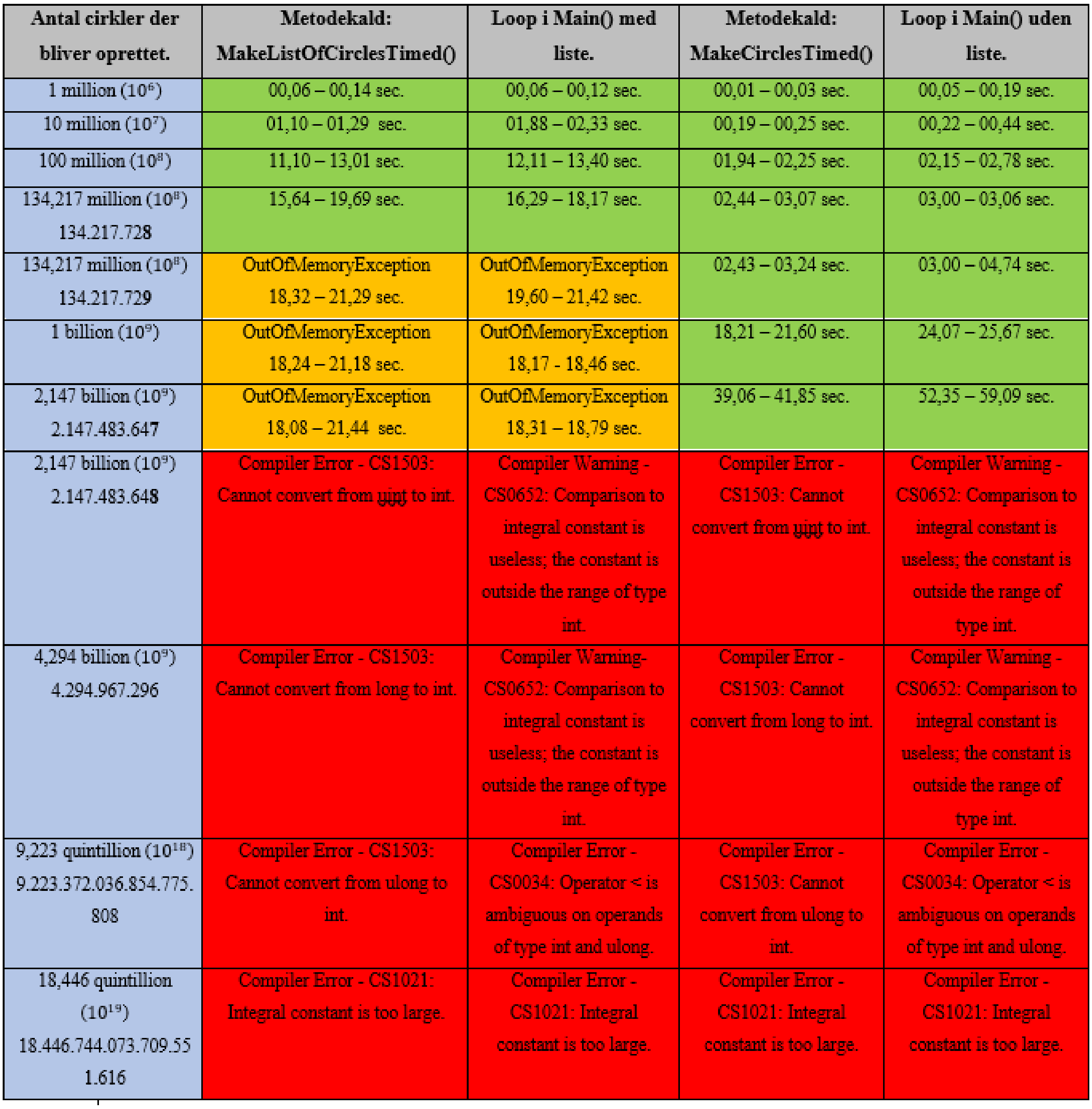
Figur 2.23: Output data fra Main metoden, del 2 af 2


C:\Users\Louise Helene Hamle\source\Programmeringssprog\Lesson11ObligatoriskOpgave\LHH_Opgave2\bin\Debug\LHH_Opgave2.exe

```
Sorting with MergeSort the second stringcollection of strings containing 20000 elements...
MergeSort stringcollection running time: 00:00:00.04

Generating the first array with 20000 random strings...
A new array of 20000 random strings has been generated.

Sorting with BubbleSort the first array of strings containing 20000 elements...
BubbleSort array running time: 00:00:15.50

Generating the second array with 20000 random strings...
A new array of 20000 random strings has been generated.

Sorting with MergeSort the second array of strings containing 20000 elements...
MergeSort array running time: 00:00:00.45

Generating the third array with 20000 random strings...
A new array of 20000 random strings has been generated.

Sorting with Array.Sort the third array of strings containing 20000 elements...
Array.Sort array running time: 00:00:00.07

Generating and sorting with MergeSort a list with 2000000 random strings...
A new list of 2000000 random strings has been generated and sorted.

Randomly picked random string from a list of 2000000 elements = "smtirfrzjntsibmkryny"
Searching for a match of "smtirfrzjntsibmkryny" in the list with 2000000 String elements...
Search has succeeded! The string was found!
The string was found at index 1421529 in the list.
Search list running time: 00:00:00.1330342

Generating and sorting with MergeSort a stringcollection with 2000000 random strings...
A new stringcollection of 2000000 random strings has been generated and sorted.

Randomly picked random string from the stringcollection of 2000000 elements = "bkiwjxahsiwyopjllqlr"
Searching for a match of "bkiwjxahsiwyopjllqlr" in the stringcollection with 2000000 String elements...
Search has succeeded! The string was found!
The string was found at index 107660 in the stringcollection.
Search stringcollection running time: 00:00:00.0172298

Generating and sorting with Array.Sort an array with 2000000 random strings...
A new array of 2000000 random strings has been generated and sorted.

Randomly picked random string from the array of 2000000 elements = "jylcurnowogkhtgteetx"
Searching for a match of "jylcurnowogkhtgteetx" in the first array with 2000000 String elements...
Search has succeeded! The string was found!
The string was found at index 761155 in the array.
Search array running time: 00:00:00.0413116
```

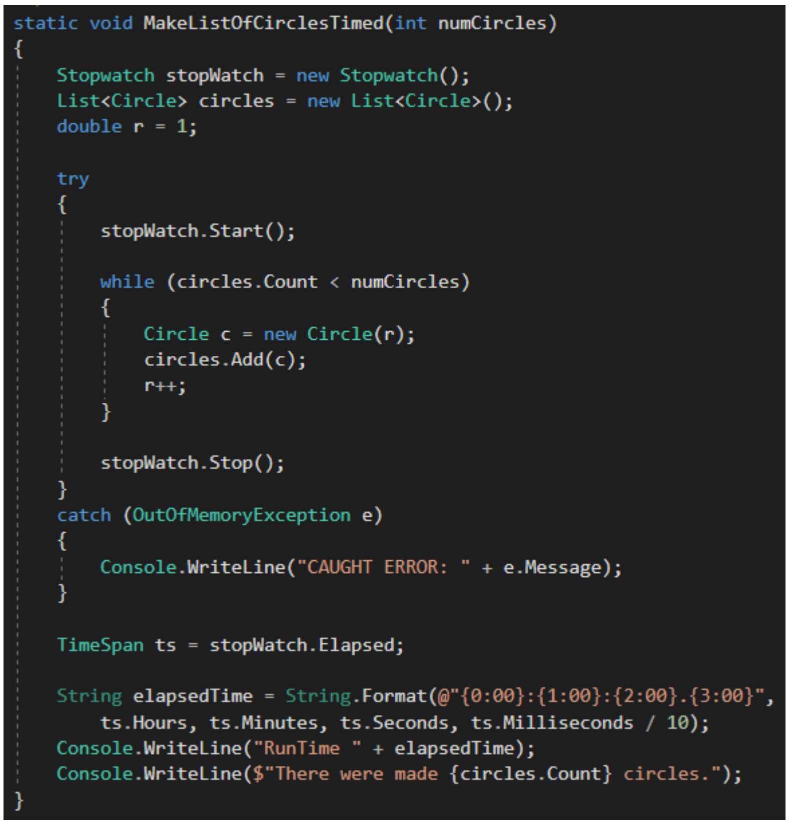
Opgave 3

Figur 3.1: Skema med statistik over køretider og fejlmeddelelser



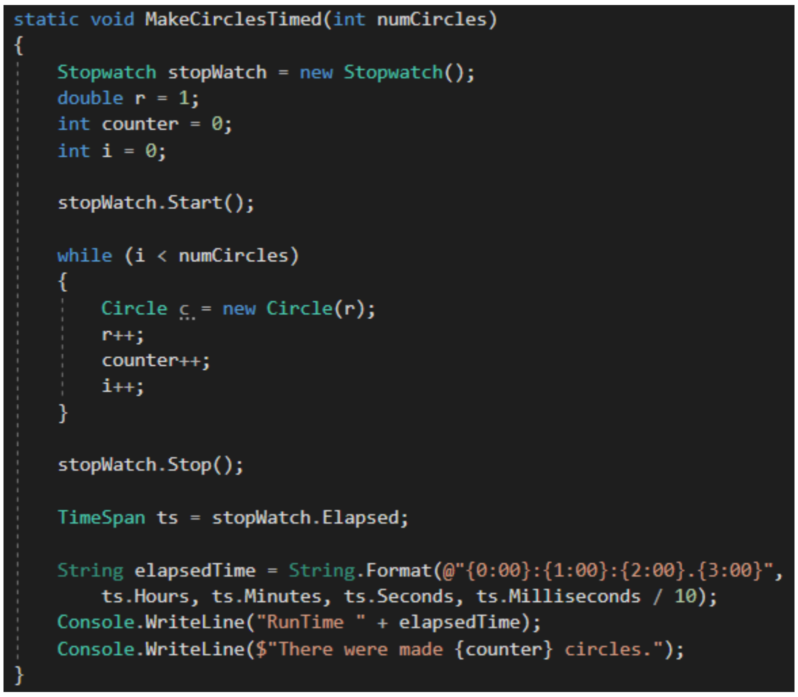
Antal cirkler der bliver oprettet.	Metodekald: MakeListOfCirclesTimed()	Loop i Main() med liste.	Metodekald: MakeCirclesTimed()	Loop i Main() uden liste.
1 million (10^6)	00,06 – 00,14 sec.	00,06 – 00,12 sec.	00,01 – 00,03 sec.	00,05 – 00,19 sec.
10 million (10^7)	01,10 – 01,29 sec.	01,88 – 02,33 sec.	00,19 – 00,25 sec.	00,22 – 00,44 sec.
100 million (10^8)	11,10 – 13,01 sec.	12,11 – 13,40 sec.	01,94 – 02,25 sec.	02,15 – 02,78 sec.
134,217 million (10^8) 134.217.728	15,64 – 19,69 sec.	16,29 – 18,17 sec.	02,44 – 03,07 sec.	03,00 – 03,06 sec.
134,217 million (10^8) 134.217.729	OutOfMemoryException 18,32 – 21,29 sec.	OutOfMemoryException 19,60 – 21,42 sec.	02,43 – 03,24 sec.	03,00 – 04,74 sec.
1 billion (10^9)	OutOfMemoryException 18,24 – 21,18 sec.	OutOfMemoryException 18,17 - 18,46 sec.	18,21 – 21,60 sec.	24,07 – 25,67 sec.
2,147 billion (10^9) 2.147.483.647	OutOfMemoryException 18,08 – 21,44 sec.	OutOfMemoryException 18,31 – 18,79 sec.	39,06 – 41,85 sec.	52,35 – 59,09 sec.
2,147 billion (10^9) 2.147.483.648	Compiler Error - CS1503: Cannot convert from uint to int.	Compiler Warning - CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.	Compiler Error - CS1503: Cannot convert from uint to int.	Compiler Warning - CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.
4,294 billion (10^9) 4.294.967.296	Compiler Error - CS1503: Cannot convert from long to int.	Compiler Warning- CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.	Compiler Error - CS1503: Cannot convert from long to int.	Compiler Warning - CS0652: Comparison to integral constant is useless; the constant is outside the range of type int.
9,223 quintillion (10^{18}) 9.223.372.036.854.775.808	Compiler Error - CS1503: Cannot convert from ulong to int.	Compiler Error - CS0034: Operator < is ambiguous on operands of type int and ulong.	Compiler Error - CS1503: Cannot convert from ulong to int.	Compiler Error - CS0034: Operator < is ambiguous on operands of type int and ulong.
18,446 quintillion (10^{19}) 18.446.744.073.709.551.616	Compiler Error - CS1021: Integral constant is too large.	Compiler Error - CS1021: Integral constant is too large.	Compiler Error - CS1021: Integral constant is too large.	Compiler Error - CS1021: Integral constant is too large.

Figur 3.2: Metoden MakeListOfCirclesTimed, som laver en liste af cirkler



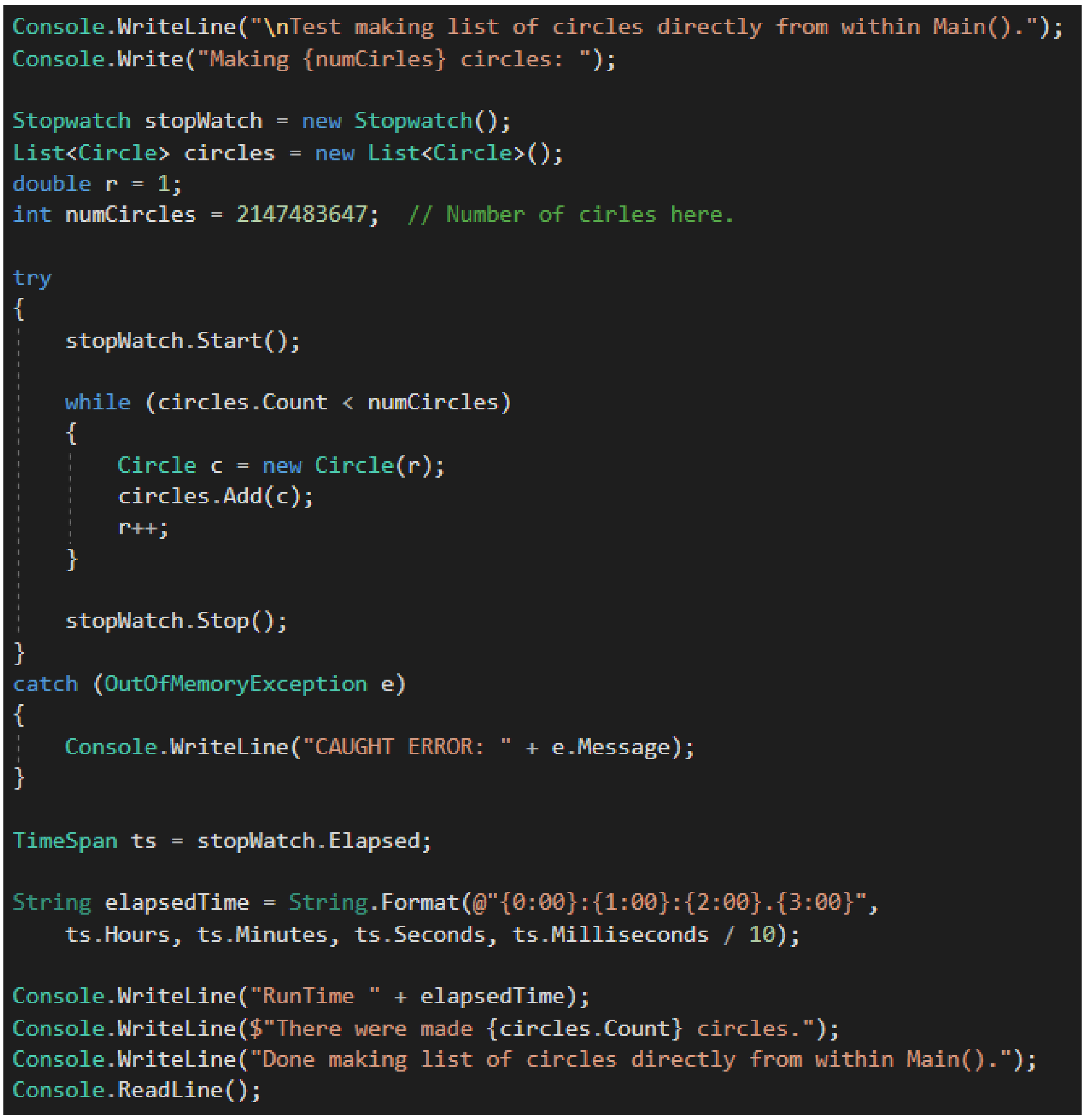
```
static void MakeListOfCirclesTimed(int numCircles)
{
    Stopwatch stopWatch = new Stopwatch();
    List<Circle> circles = new List<Circle>();
    double r = 1;

    try
    {
        stopWatch.Start();

        while (circles.Count < numCircles)
        {
            Circle c = new Circle(r);
            circles.Add(c);
            r++;
        }

        stopWatch.Stop();
    }
    catch (OutOfMemoryException e)
    {
        Console.WriteLine("CAUGHT ERROR: " + e.Message);
    }

    TimeSpan ts = stopWatch.Elapsed;

    String elapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
        ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
    Console.WriteLine("RunTime " + elapsedTime);
    Console.WriteLine($"There were made {circles.Count} circles.");
}
```

Figur 3.3: Metoden MakeCirclesTimed, som laver cirkler og kun gemmer/husker den sidst oprettede



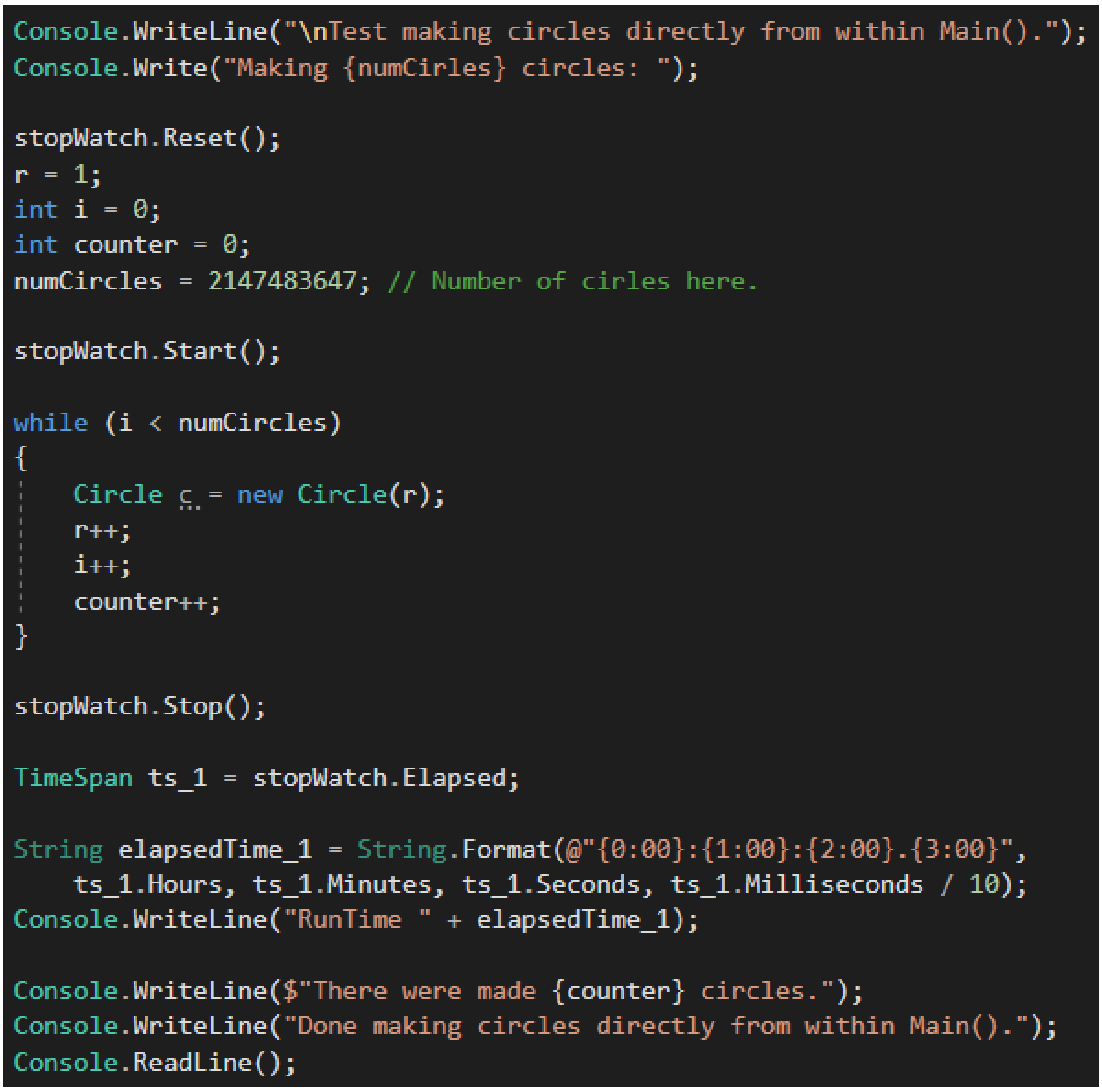
```
static void MakeCirclesTimed(int numCircles)
{
    Stopwatch stopWatch = new Stopwatch();
    double r = 1;
    int counter = 0;
    int i = 0;

    stopWatch.Start();

    while (i < numCircles)
    {
        Circle c = new Circle(r);
        r++;
        counter++;
        i++;
    }

    stopWatch.Stop();

    TimeSpan ts = stopWatch.Elapsed;

    String elapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
        ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
    Console.WriteLine("RunTime " + elapsedTime);
    Console.WriteLine($"There were made {counter} circles.");
}
```

Figur 3.4: Kode direkte i Main metoden, som gemmer cirkler i en liste



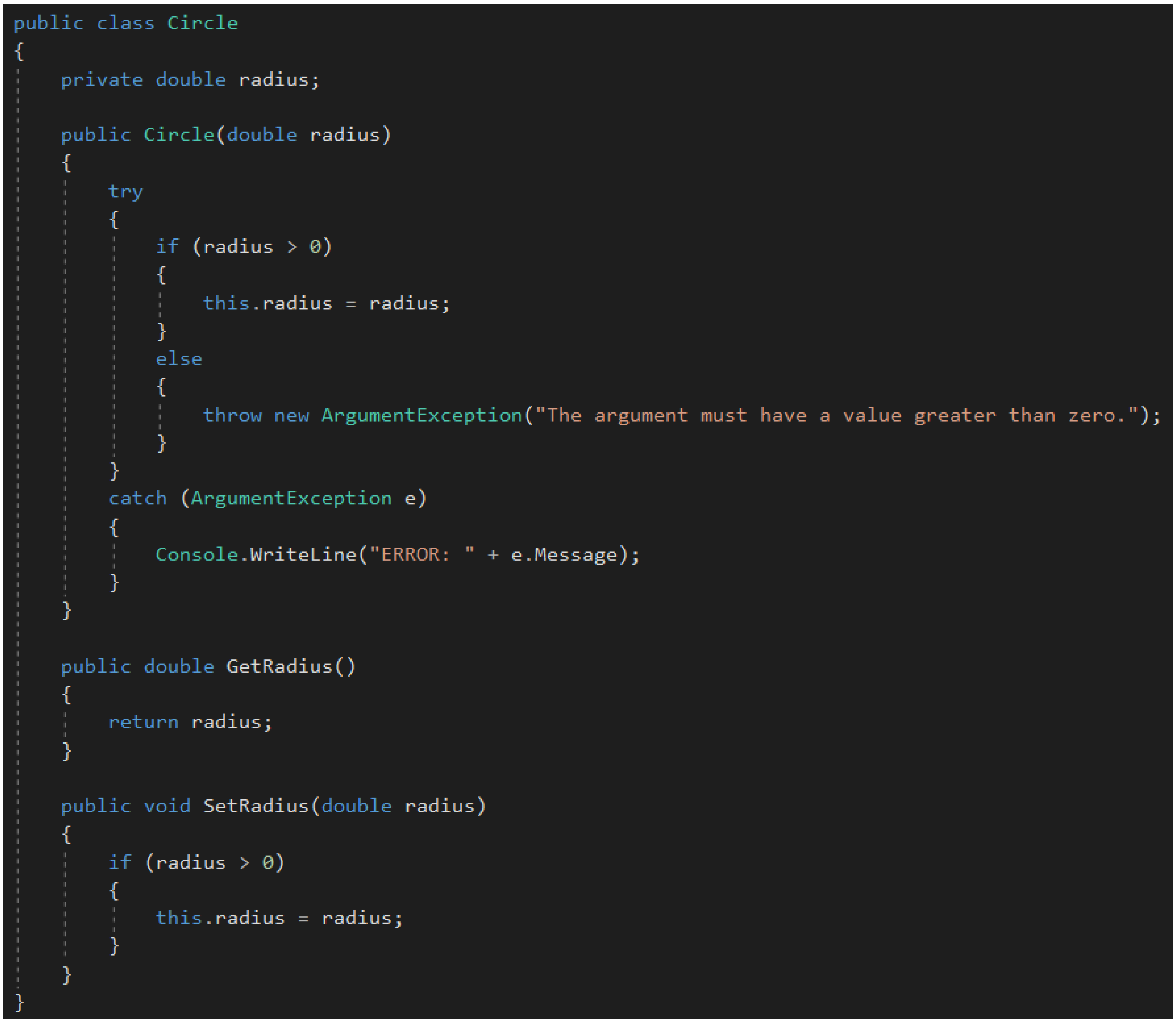
```
Console.WriteLine("\nTest making list of circles directly from within Main().");
Console.Write("Making {numCirles} circles: ");

Stopwatch stopWatch = new Stopwatch();
List<Circle> circles = new List<Circle>();
double r = 1;
int numCircles = 2147483647;  // Number of cirles here.

try
{
    stopWatch.Start();

    while (circles.Count < numCircles)
    {
        Circle c = new Circle(r);
        circles.Add(c);
        r++;
    }

    stopWatch.Stop();
}
catch (OutOfMemoryException e)
{
    Console.WriteLine("CAUGHT ERROR: " + e.Message);
}

TimeSpan ts = stopWatch.Elapsed;

String elapsedTime = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);

Console.WriteLine("RunTime " + elapsedTime);
Console.WriteLine($"There were made {circles.Count} circles.");
Console.WriteLine("Done making list of circles directly from within Main().");
Console.ReadLine();
```

Figur 3.5: Kode direkte i Main metoden, som kun gemmer den sidste cirkel i en variabel



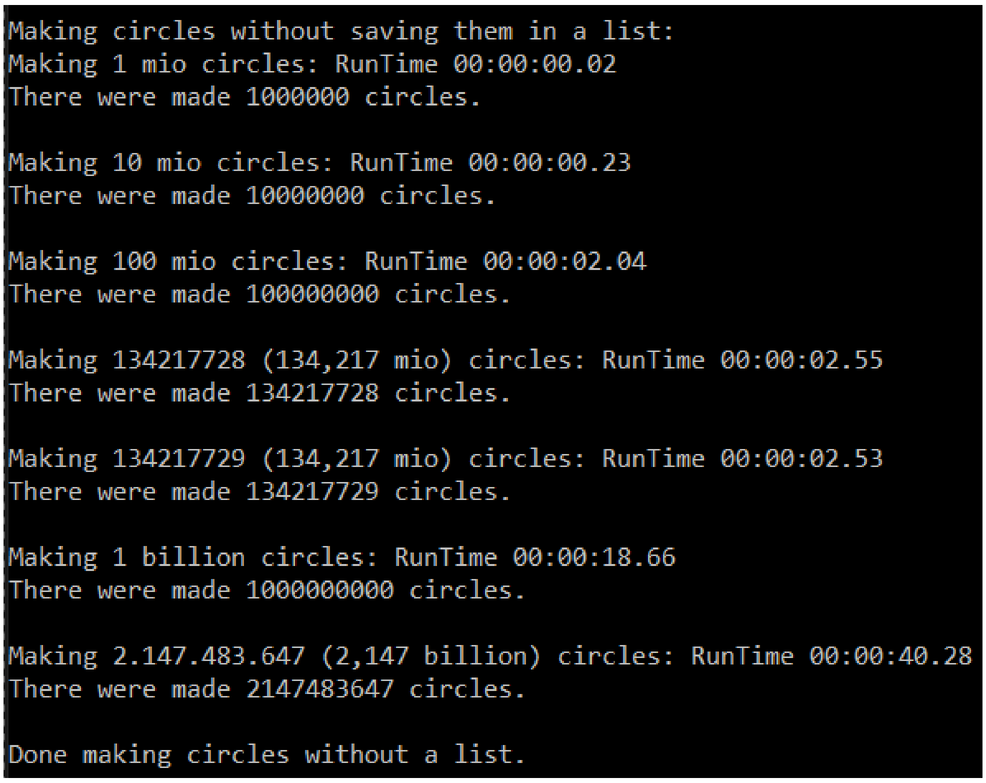
```
Console.WriteLine("\nTest making circles directly from within Main().");
Console.Write("Making {numCirles} circles: ");

stopWatch.Reset();
r = 1;
int i = 0;
int counter = 0;
numCircles = 2147483647; // Number of cirles here.

stopWatch.Start();

while (i < numCircles)
{
    Circle c = new Circle(r);
    r++;
    i++;
    counter++;
}

stopWatch.Stop();

TimeSpan ts_1 = stopWatch.Elapsed;

String elapsedTime_1 = String.Format(@"{0:00}:{1:00}:{2:00}.{3:00}",
    ts_1.Hours, ts_1.Minutes, ts_1.Seconds, ts_1.Milliseconds / 10);
Console.WriteLine("RunTime " + elapsedTime_1);

Console.WriteLine($"There were made {counter} circles.");
Console.WriteLine("Done making circles directly from within Main().");
Console.ReadLine();
```

Figur 3.6: Klassen Circle



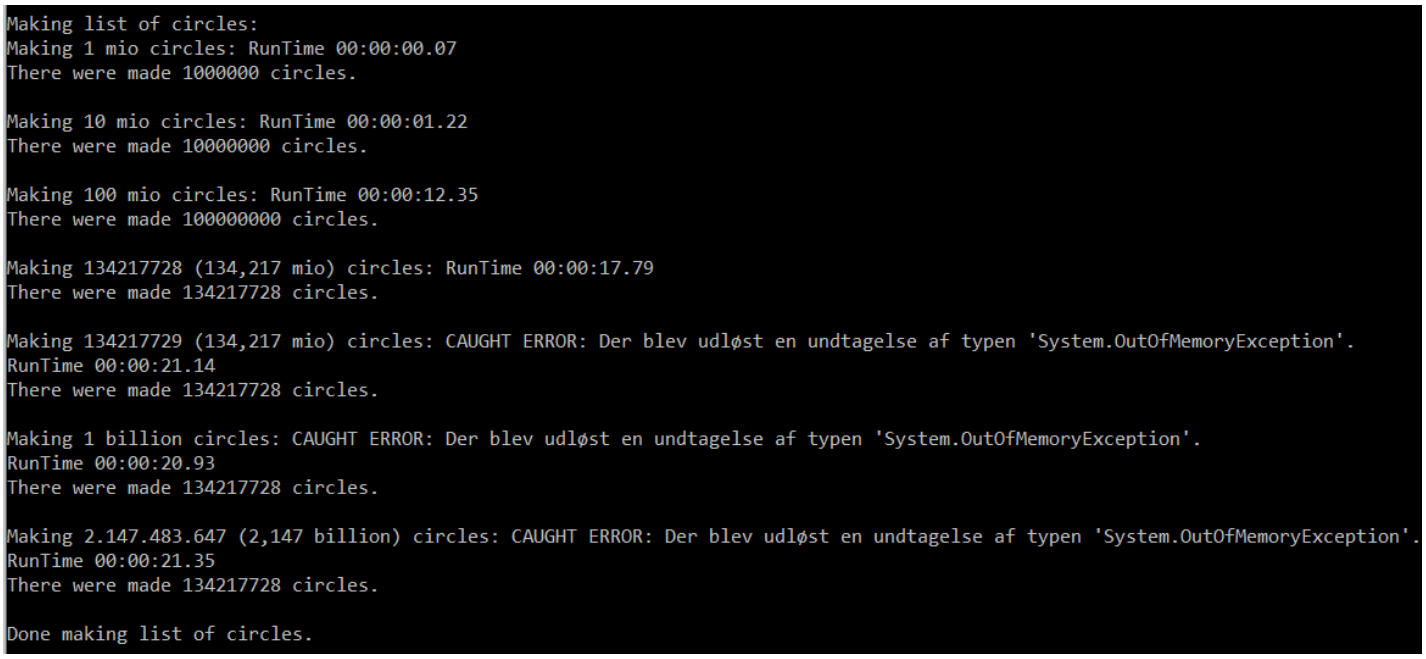
```
public class Circle
{
    private double radius;

    public Circle(double radius)
    {
        try
        {
            if (radius > 0)
            {
                this.radius = radius;
            }
            else
            {
                throw new ArgumentException("The argument must have a value greater than zero.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }
    }

    public double GetRadius()
    {
        return radius;
    }

    public void SetRadius(double radius)
    {
        if (radius > 0)
        {
            this.radius = radius;
        }
    }
}
```

Figur 3.7: Output testdata - metoden MakeCirclesTimed



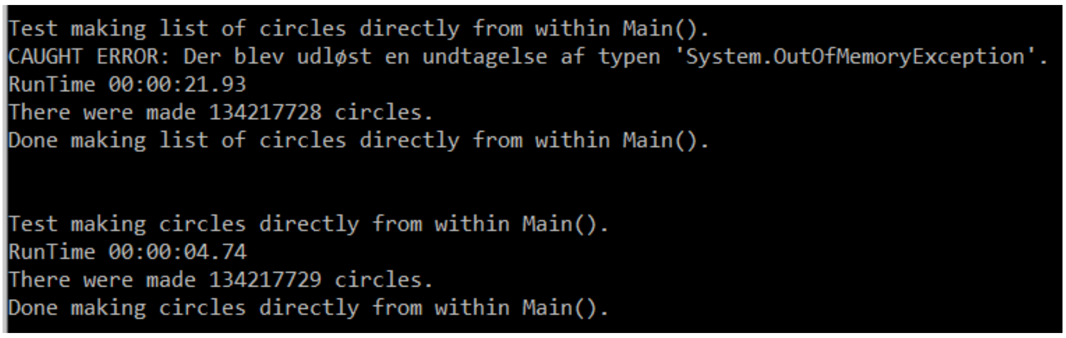
```
Making circles without saving them in a list:
Making 1 mio circles: RunTime 00:00:00.02
There were made 1000000 circles.

Making 10 mio circles: RunTime 00:00:00.23
There were made 10000000 circles.

Making 100 mio circles: RunTime 00:00:02.04
There were made 100000000 circles.

Making 134217728 (134,217 mio) circles: RunTime 00:00:02.55
There were made 134217728 circles.

Making 134217729 (134,217 mio) circles: RunTime 00:00:02.53
There were made 134217729 circles.

Making 1 billion circles: RunTime 00:00:18.66
There were made 1000000000 circles.

Making 2.147.483.647 (2,147 billion) circles: RunTime 00:00:40.28
There were made 2147483647 circles.

Done making circles without a list.
```

Figur 3.8: Output testdata – metoden MakeListOfCirlesTimed



```
Making list of circles:
Making 1 mio circles: RunTime 00:00:00.07
There were made 1000000 circles.

Making 10 mio circles: RunTime 00:00:01.22
There were made 10000000 circles.

Making 100 mio circles: RunTime 00:00:12.35
There were made 100000000 circles.

Making 134217728 (134,217 mio) circles: RunTime 00:00:17.79
There were made 134217728 circles.

Making 134217729 (134,217 mio) circles: CAUGHT ERROR: Der blev udløst en undtagelse af typen 'System.OutOfMemoryException'.
RunTime 00:00:21.14
There were made 134217728 circles.

Making 1 billion circles: CAUGHT ERROR: Der blev udløst en undtagelse af typen 'System.OutOfMemoryException'.
RunTime 00:00:20.93
There were made 134217728 circles.

Making 2.147.483.647 (2,147 billion) circles: CAUGHT ERROR: Der blev udløst en undtagelse af typen 'System.OutOfMemoryException'.
RunTime 00:00:21.35
There were made 134217728 circles.

Done making list of circles.
```

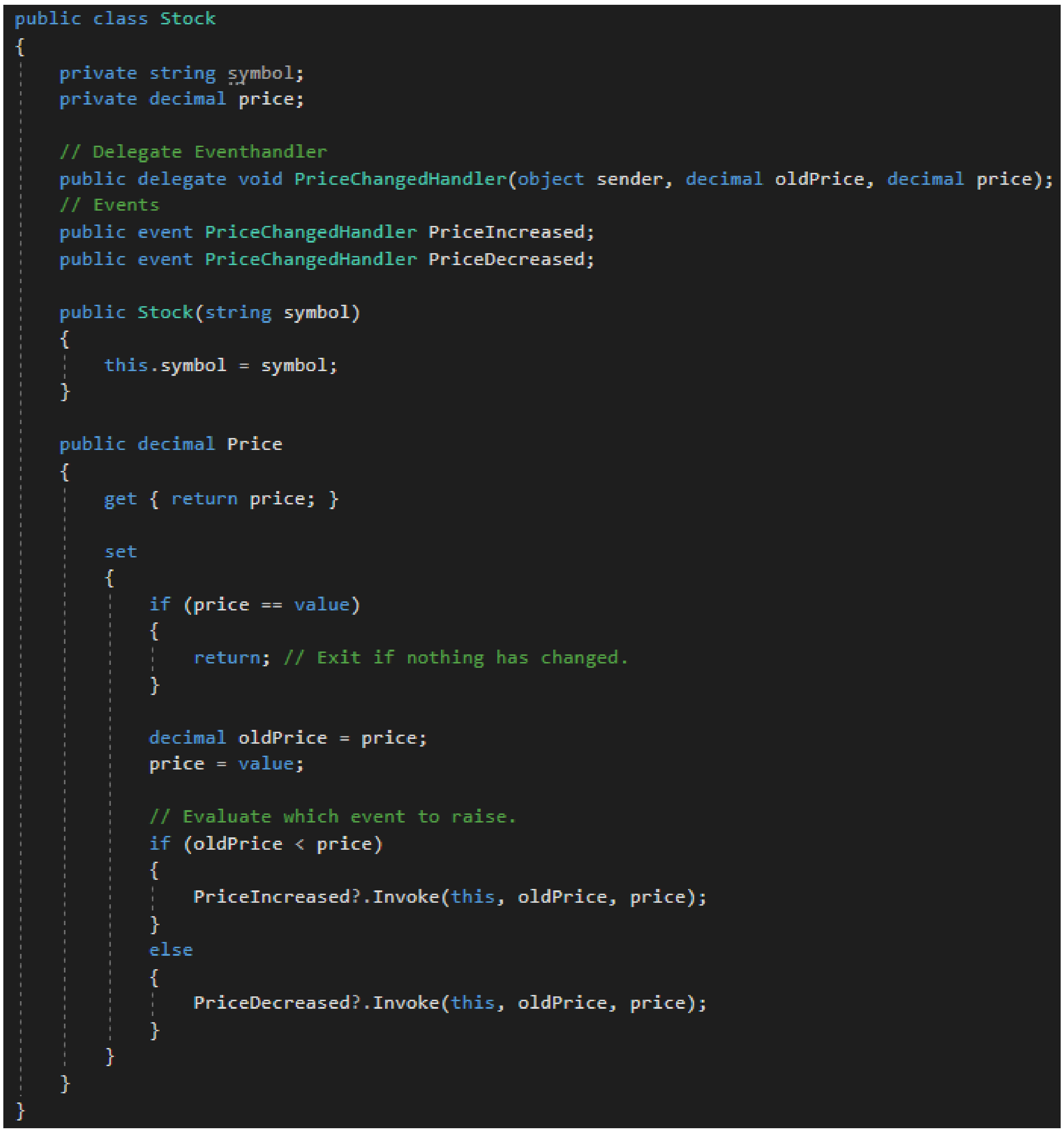
Figur 3.9: Output testdata – kode i Main metoden



```
Test making list of circles directly from within Main().
CAUGHT ERROR: Der blev udløst en undtagelse af typen 'System.OutOfMemoryException'.
RunTime 00:00:21.93
There were made 134217728 circles.
Done making list of circles directly from within Main().


Test making circles directly from within Main().
RunTime 00:00:04.74
There were made 134217729 circles.
Done making circles directly from within Main().
```

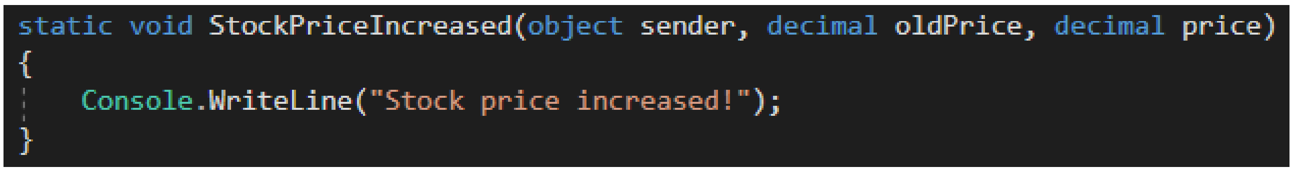
Opgave 4

Figur 4.1: Klassen Stock



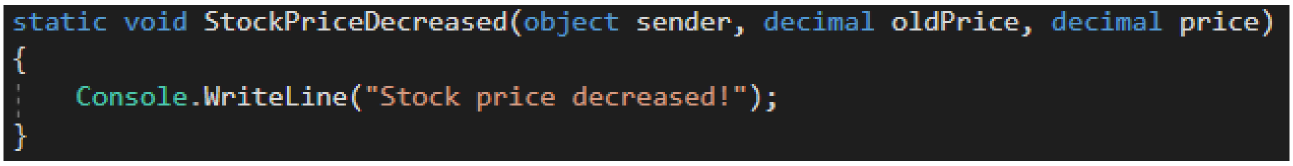
```
public class Stock
{
    private string symbol;
    private decimal price;

    // Delegate Eventhandler
    public delegate void PriceChangedHandler(object sender, decimal oldPrice, decimal price);
    // Events
    public event PriceChangedHandler PriceIncreased;
    public event PriceChangedHandler PriceDecreased;

    public Stock(string symbol)
    {
        this.symbol = symbol;
    }

    public decimal Price
    {
        get { return price; }

        set
        {
            if (price == value)
            {
                return; // Exit if nothing has changed.
            }

            decimal oldPrice = price;
            price = value;

            // Evaluate which event to raise.
            if (oldPrice < price)
            {
                PriceIncreased?.Invoke(this, oldPrice, price);
            }
            else
            {
                PriceDecreased?.Invoke(this, oldPrice, price);
            }
        }
    }
}
```

Figur 4.2: Subscriber metoden StockPriceIncreased til stigende prisændringer



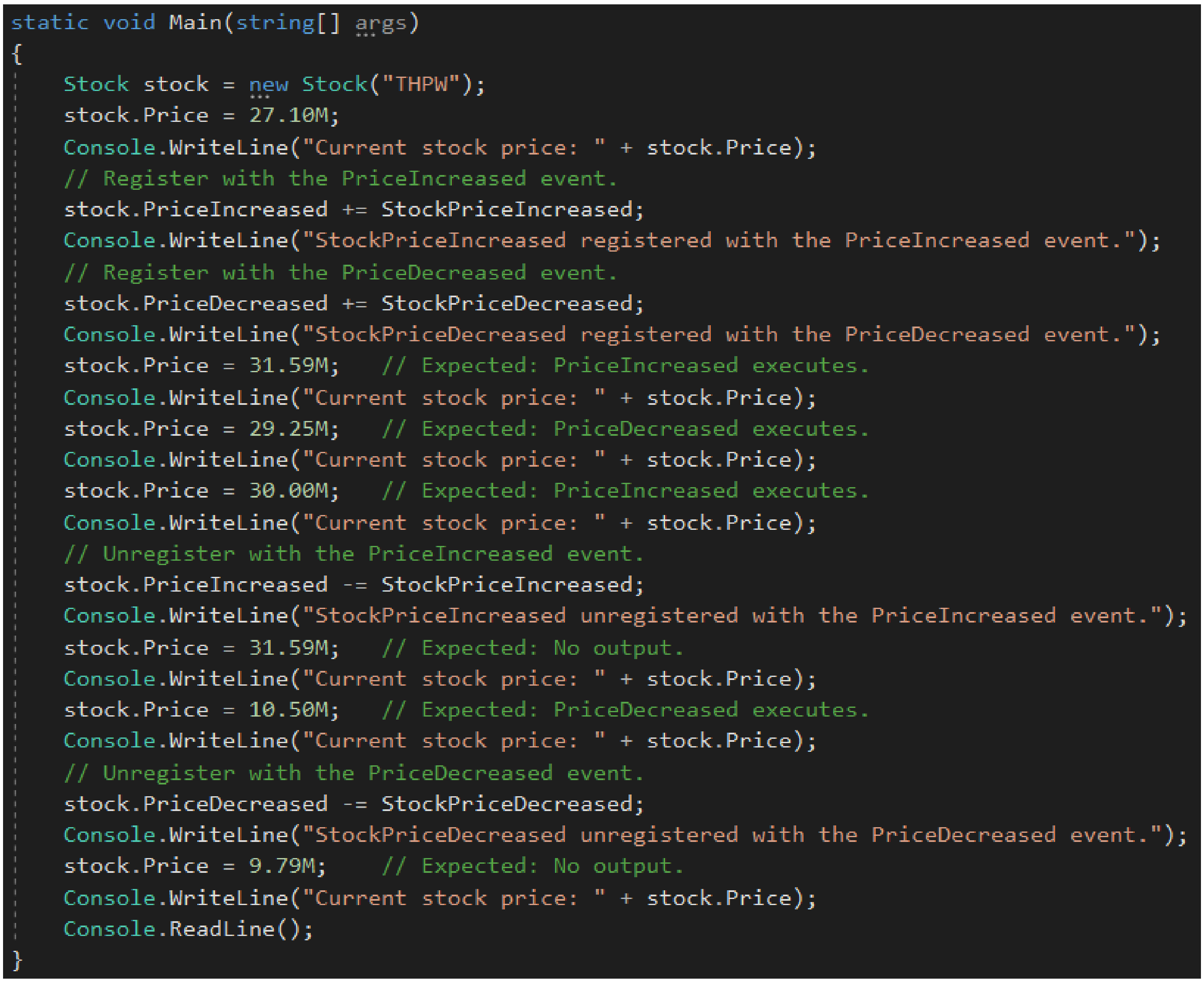
```
static void StockPriceIncreased(object sender, decimal oldPrice, decimal price)
{
    Console.WriteLine("Stock price increased!");
}
```

Figur 4.3: Subscriber metoden StockPriceDecreased til faldende prisændringer



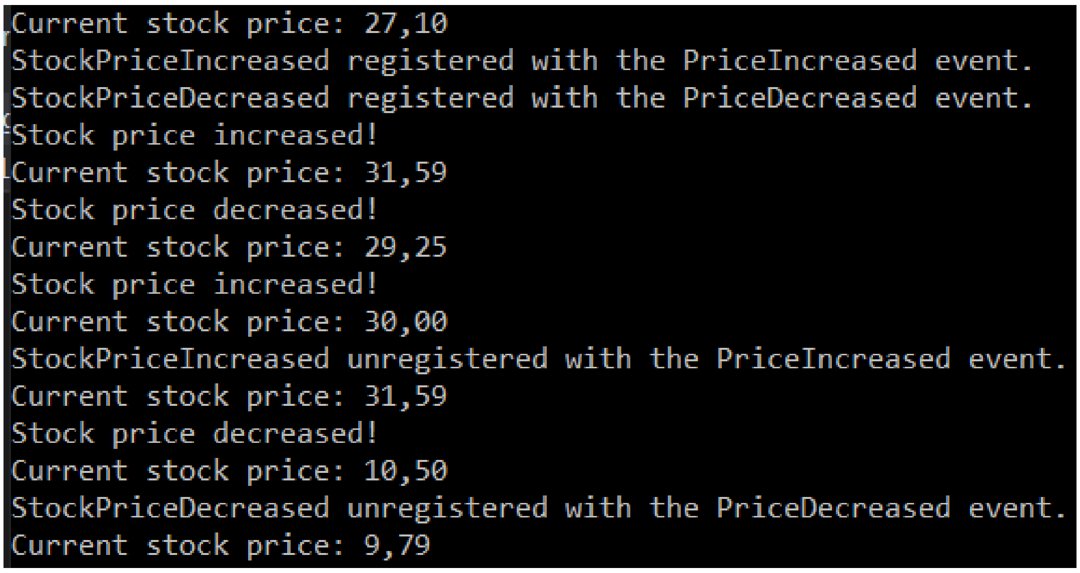
```
static void StockPriceDecreased(object sender, decimal oldPrice, decimal price)
{
    Console.WriteLine("Stock price decreased!");
}
```

Figur 4.4: Main metoden med test data



```
static void Main(string[] args)
{
    Stock stock = new Stock("THPW");
    stock.Price = 27.10M;
    Console.WriteLine("Current stock price: " + stock.Price);
    // Register with the PriceIncreased event.
    stock.PriceIncreased += StockPriceIncreased;
    Console.WriteLine("StockPriceIncreased registered with the PriceIncreased event.");
    // Register with the PriceDecreased event.
    stock.PriceDecreased += StockPriceDecreased;
    Console.WriteLine("StockPriceDecreased registered with the PriceDecreased event.");
    stock.Price = 31.59M;   // Expected: PriceIncreased executes.
    Console.WriteLine("Current stock price: " + stock.Price);
    stock.Price = 29.25M;   // Expected: PriceDecreased executes.
    Console.WriteLine("Current stock price: " + stock.Price);
    stock.Price = 30.00M;   // Expected: PriceIncreased executes.
    Console.WriteLine("Current stock price: " + stock.Price);
    // Unregister with the PriceIncreased event.
    stock.PriceIncreased -= StockPriceIncreased;
    Console.WriteLine("StockPriceIncreased unregistered with the PriceIncreased event.");
    stock.Price = 31.59M;   // Expected: No output.
    Console.WriteLine("Current stock price: " + stock.Price);
    stock.Price = 10.50M;   // Expected: PriceDecreased executes.
    Console.WriteLine("Current stock price: " + stock.Price);
    // Unregister with the PriceDecreased event.
    stock.PriceDecreased -= StockPriceDecreased;
    Console.WriteLine("StockPriceDecreased unregistered with the PriceDecreased event.");
    stock.Price = 9.79M;    // Expected: No output.
    Console.WriteLine("Current stock price: " + stock.Price);
    Console.ReadLine();
}
```

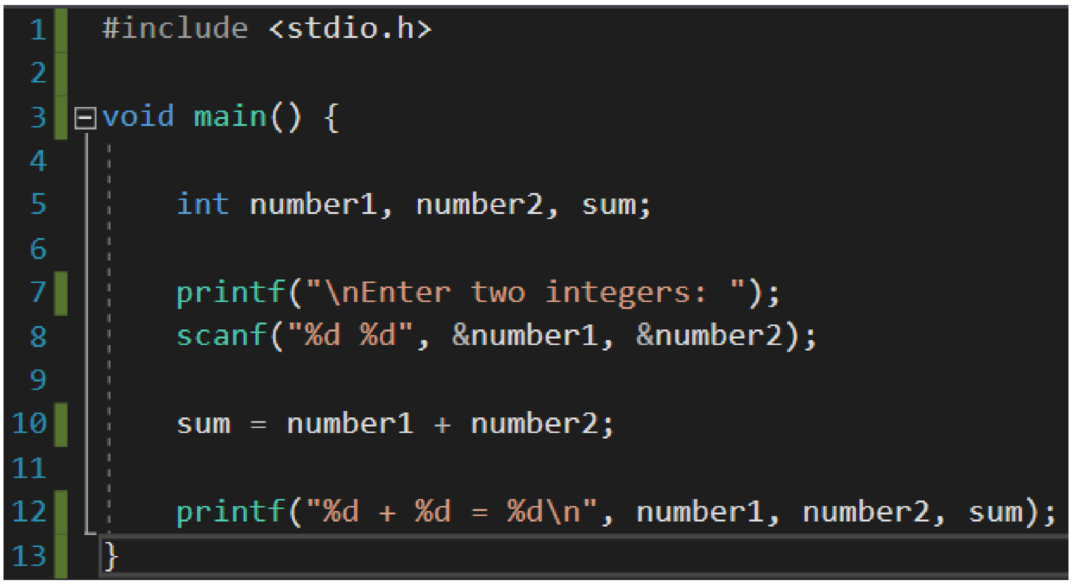
Figur 4.5: Output data fra Main metoden



```
Current stock price: 27,10
StockPriceIncreased registered with the PriceIncreased event.
StockPriceDecreased registered with the PriceDecreased event.
Stock price increased!
Current stock price: 31,59
Stock price decreased!
Current stock price: 29,25
Stock price increased!
Current stock price: 30,00
StockPriceIncreased unregistered with the PriceIncreased event.
Current stock price: 31,59
Stock price decreased!
Current stock price: 10,50
StockPriceDecreased unregistered with the PriceDecreased event.
Current stock price: 9,79
```

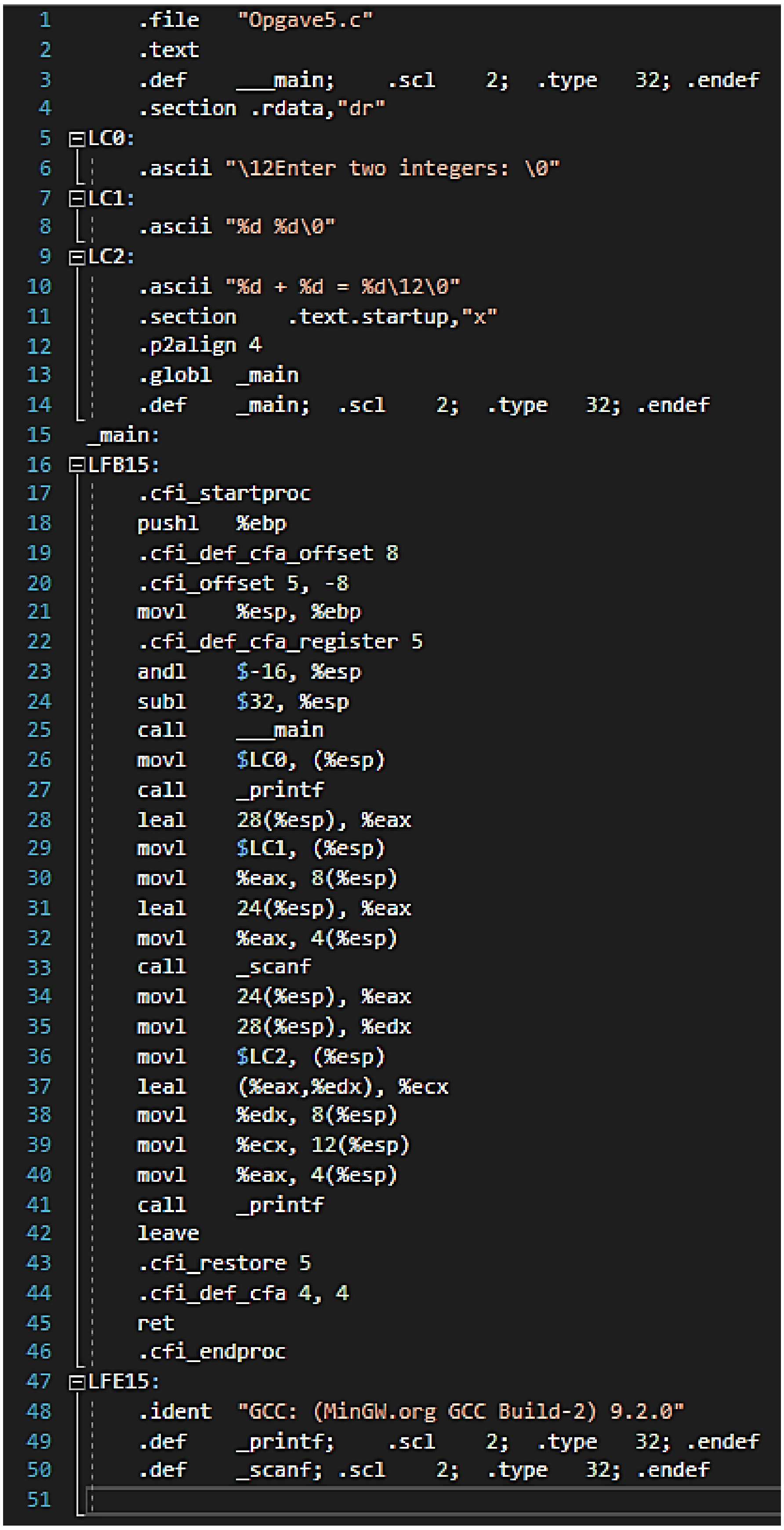
Opgave 5

Figur 5.1: C koden for et simpelt program



```c
#include <stdio.h>

void main() {

    int number1, number2, sum;

    printf("\nEnter two integers: ");
    scanf("%d %d", &number1, &number2);

    sum = number1 + number2;

    printf("%d + %d = %d\n", number1, number2, sum);
}
```

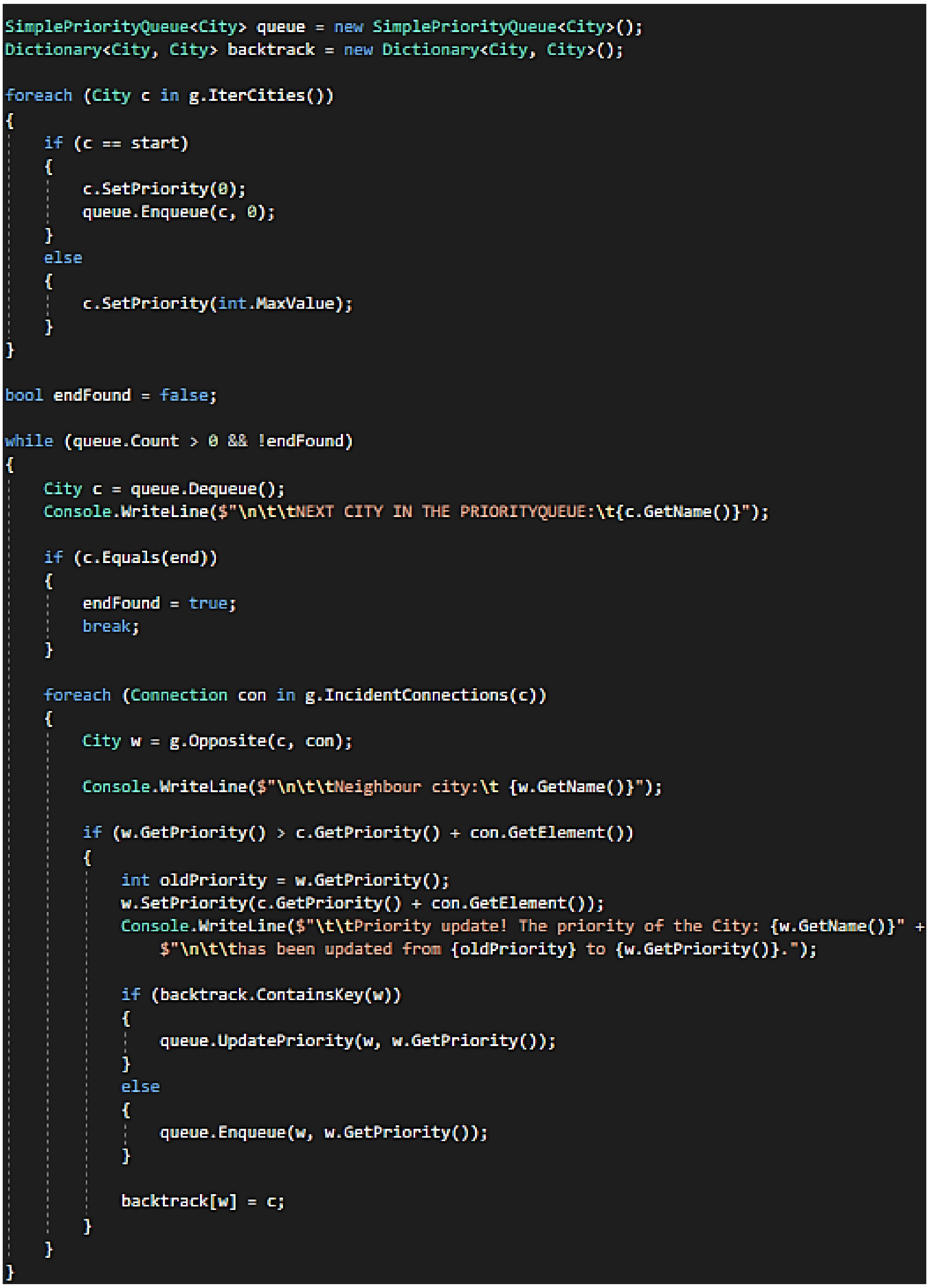
Figur 5.2: Assembler koden for det simple C program



```
	.file	"Opgave5.c"
	.text
	.def	___main;	.scl	2;	.type	32;	.endef
	.section .rdata,"dr"
LC0:
	.ascii "\12Enter two integers: \0"
LC1:
	.ascii "%d %d\0"
LC2:
	.ascii "%d + %d = %d\12\0"
	.section	.text.startup,"x"
	.p2align 4
	.globl	_main
	.def	_main;	.scl	2;	.type	32;	.endef
_main:
LFB15:
	.cfi_startproc
	pushl	%ebp
	.cfi_def_cfa_offset 8
	.cfi_offset 5, -8
	movl	%esp, %ebp
	.cfi_def_cfa_register 5
	andl	$-16, %esp
	subl	$32, %esp
	call	___main
	movl	$LC0, (%esp)
	call	_printf
	leal	28(%esp), %eax
	movl	$LC1, (%esp)
	movl	%eax, 8(%esp)
	leal	24(%esp), %eax
	movl	%eax, 4(%esp)
	call	_scanf
	movl	24(%esp), %eax
	movl	28(%esp), %edx
	movl	$LC2, (%esp)
	leal	(%eax,%edx), %ecx
	movl	%edx, 8(%esp)
	movl	%ecx, 12(%esp)
	movl	%eax, 4(%esp)
	call	_printf
	leave
	.cfi_restore 5
	.cfi_def_cfa 4, 4
	ret
	.cfi_endproc
LFE15:
	.ident	"GCC: (MinGW.org GCC Build-2) 9.2.0"
	.def	_printf;	.scl	2;	.type	32;	.endef
	.def	_scanf;	.scl	2;	.type	32;	.endef
```

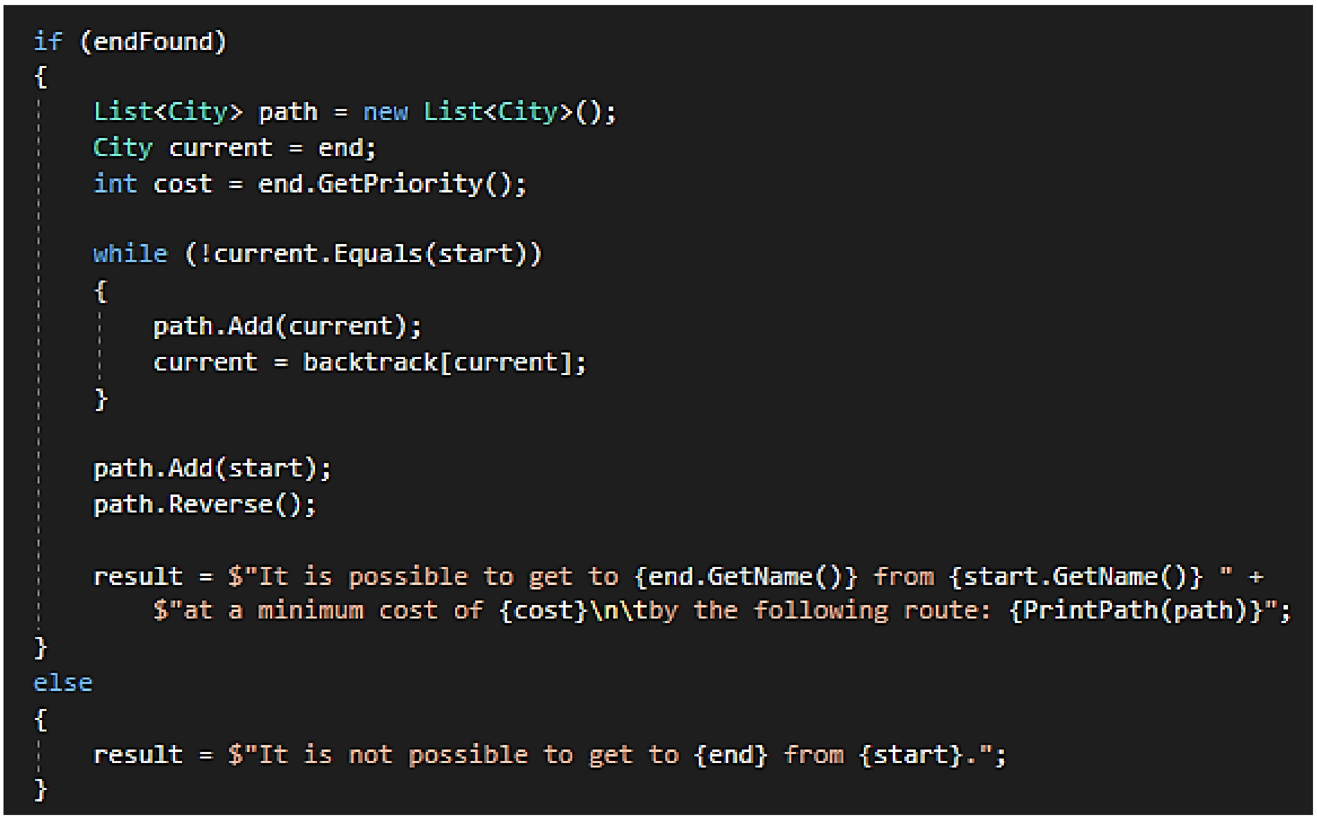
Opgave 6

Figur 6.1: Metoden Dijkstras - Find en sti, hvis der er en



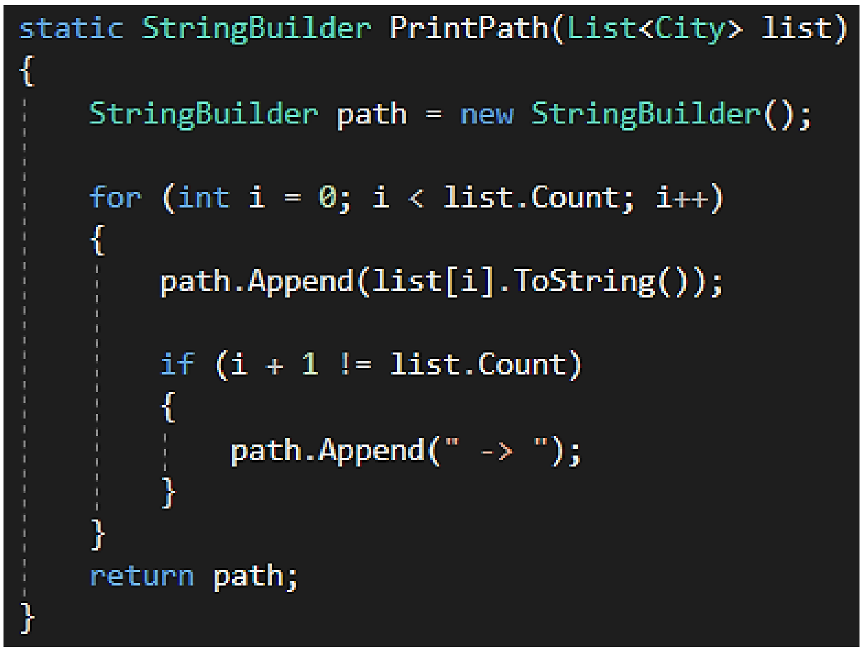
```
SimplePriorityQueue<City> queue = new SimplePriorityQueue<City>();
Dictionary<City, City> backtrack = new Dictionary<City, City>();

foreach (City c in g.IterCities())
{
    if (c == start)
    {
        c.SetPriority(0);
        queue.Enqueue(c, 0);
    }
    else
    {
        c.SetPriority(int.MaxValue);
    }
}

bool endFound = false;

while (queue.Count > 0 && !endFound)
{
    City c = queue.Dequeue();
    Console.WriteLine($"\n\t\tNEXT CITY IN THE PRIORITYQUEUE:\t{c.GetName()}");

    if (c.Equals(end))
    {
        endFound = true;
        break;
    }

    foreach (Connection con in g.IncidentConnections(c))
    {
        City w = g.Opposite(c, con);

        Console.WriteLine($"\n\t\tNeighbour city:\t {w.GetName()}");

        if (w.GetPriority() > c.GetPriority() + con.GetElement())
        {
            int oldPriority = w.GetPriority();
            w.SetPriority(c.GetPriority() + con.GetElement());
            Console.WriteLine($"\t\tPriority update! The priority of the City: {w.GetName()}" +
                $"\n\t\thas been updated from {oldPriority} to {w.GetPriority()}.");

            if (backtrack.ContainsKey(w))
            {
                queue.UpdatePriority(w, w.GetPriority());
            }
            else
            {
                queue.Enqueue(w, w.GetPriority());
            }

            backtrack[w] = c;
        }
    }
}
```

Figur 6.2: Metoden Dijkstras - Resultatet udskrives



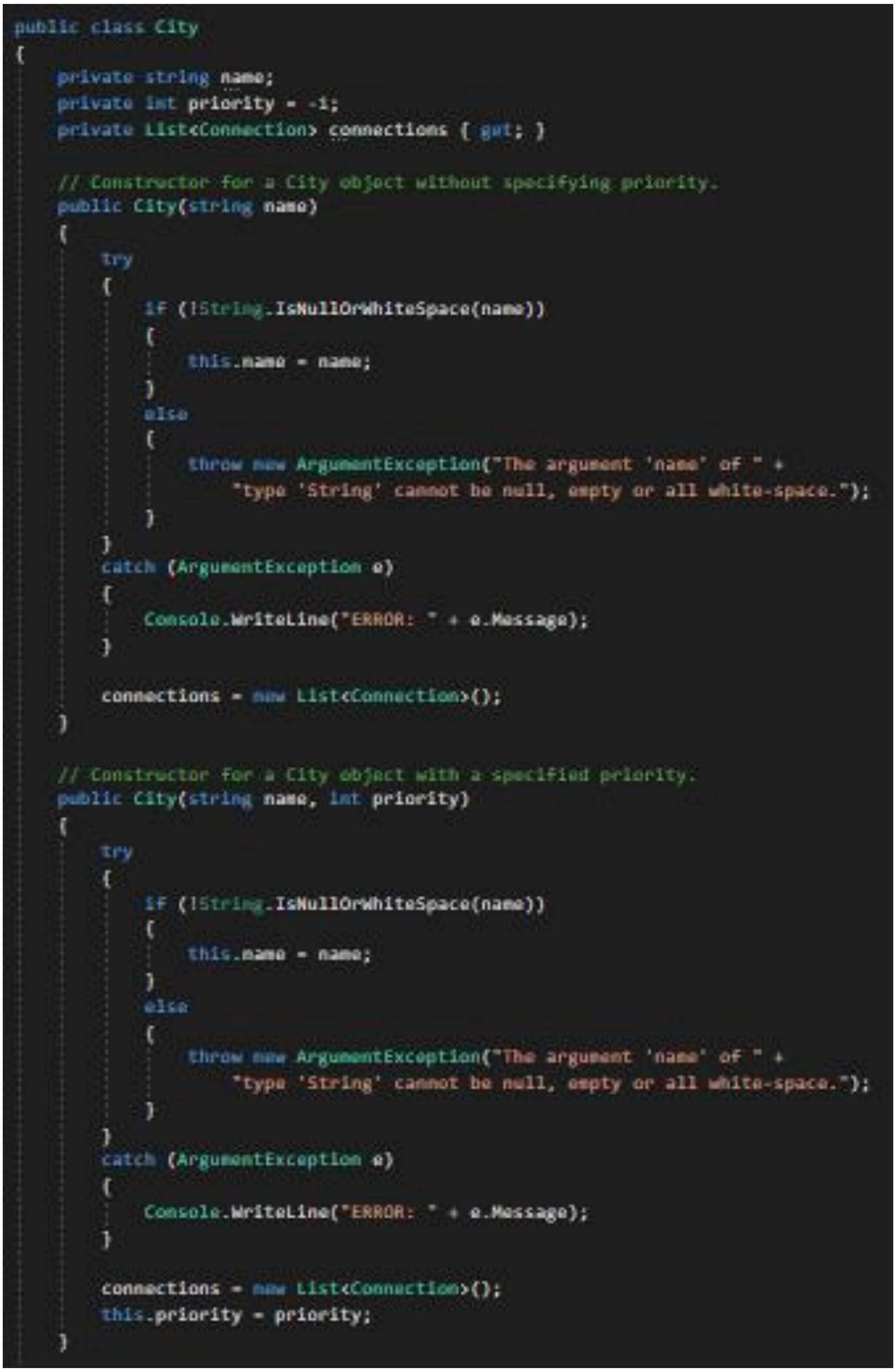
```
if (endFound)
{
    List<City> path = new List<City>();
    City current = end;
    int cost = end.GetPriority();

    while (!current.Equals(start))
    {
        path.Add(current);
        current = backtrack[current];
    }

    path.Add(start);
    path.Reverse();

    result = $"It is possible to get to {end.GetName()} from {start.GetName()} " +
        $"at a minimum cost of {cost}\n\tby the following route: {PrintPath(path)}";
}
else
{
    result = $"It is not possible to get to {end} from {start}.";
}
```

Figur 6.3: Hjælpemetode PrintPath - Print en sti



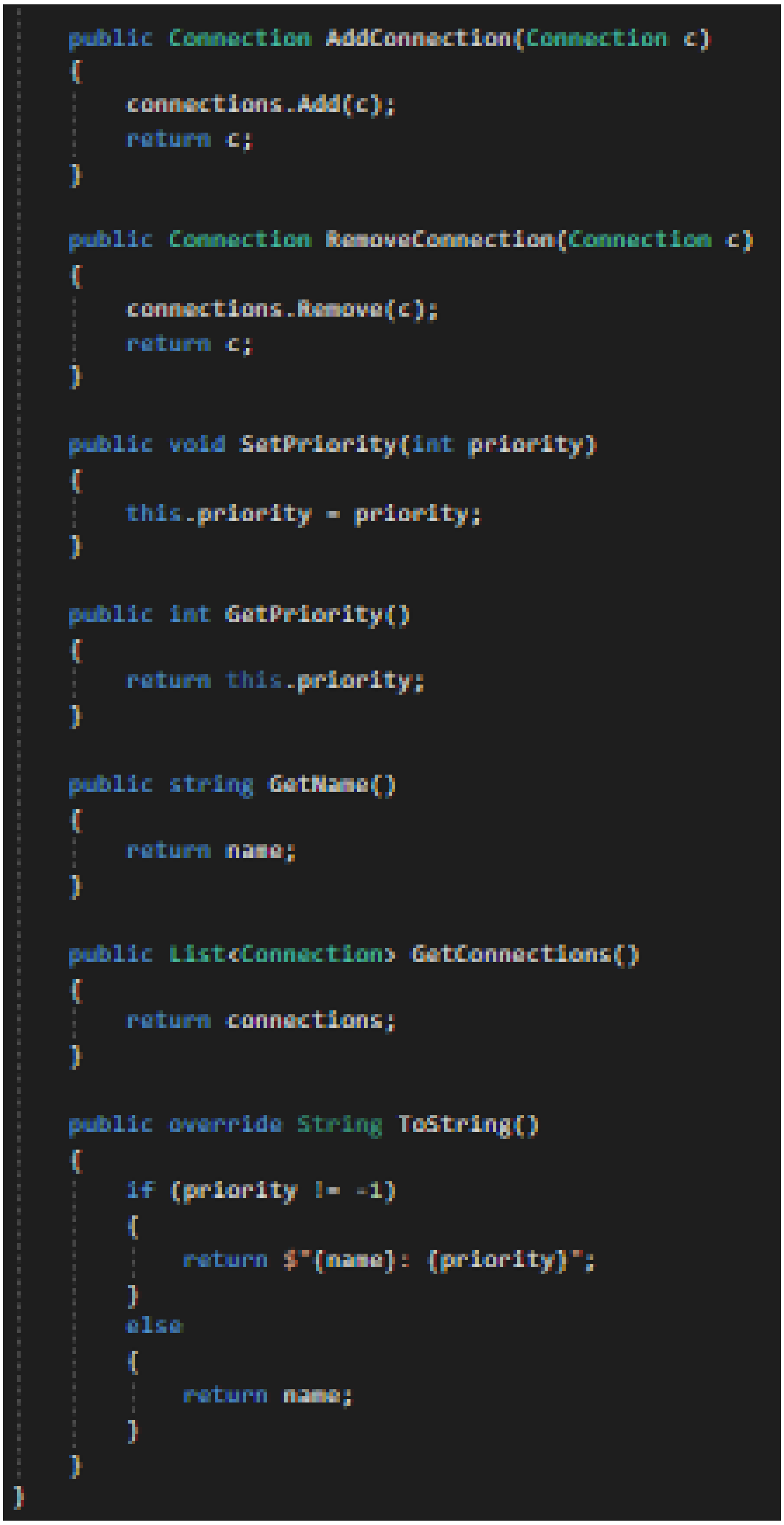
```
static StringBuilder PrintPath(List<City> list)
{
    StringBuilder path = new StringBuilder();

    for (int i = 0; i < list.Count; i++)
    {
        path.Append(list[i].ToString());

        if (i + 1 != list.Count)
        {
            path.Append(" -> ");
        }
    }
    return path;
}
```

Figur 6.4: Knudeklassen City, del 1 af 2



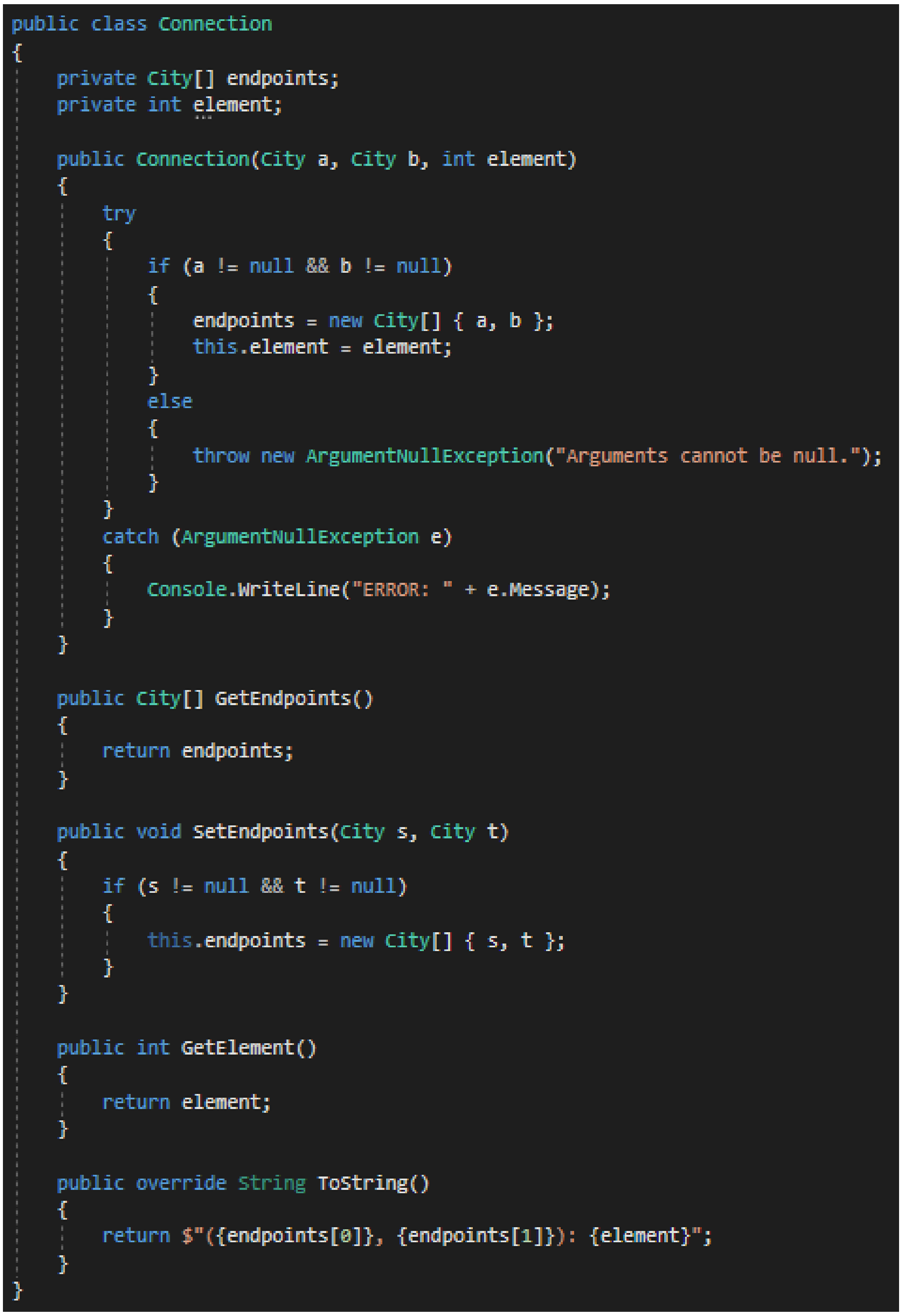
```
public class City
{
    private string name;
    private int priority = -1;
    private List<Connection> connections { get; }

    // Constructor for a City object without specifying priority.
    public City(string name)
    {
        try
        {
            if (!String.IsNullOrWhiteSpace(name))
            {
                this.name = name;
            }
            else
            {
                throw new ArgumentException("The argument 'name' of " +
                    "type 'String' cannot be null, empty or all white-space.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        connections = new List<Connection>();
    }

    // Constructor for a City object with a specified priority.
    public City(string name, int priority)
    {
        try
        {
            if (!String.IsNullOrWhiteSpace(name))
            {
                this.name = name;
            }
            else
            {
                throw new ArgumentException("The argument 'name' of " +
                    "type 'String' cannot be null, empty or all white-space.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        connections = new List<Connection>();
        this.priority = priority;
    }
```

Figur 6.5: Knudeklassen City, del 2 af 2



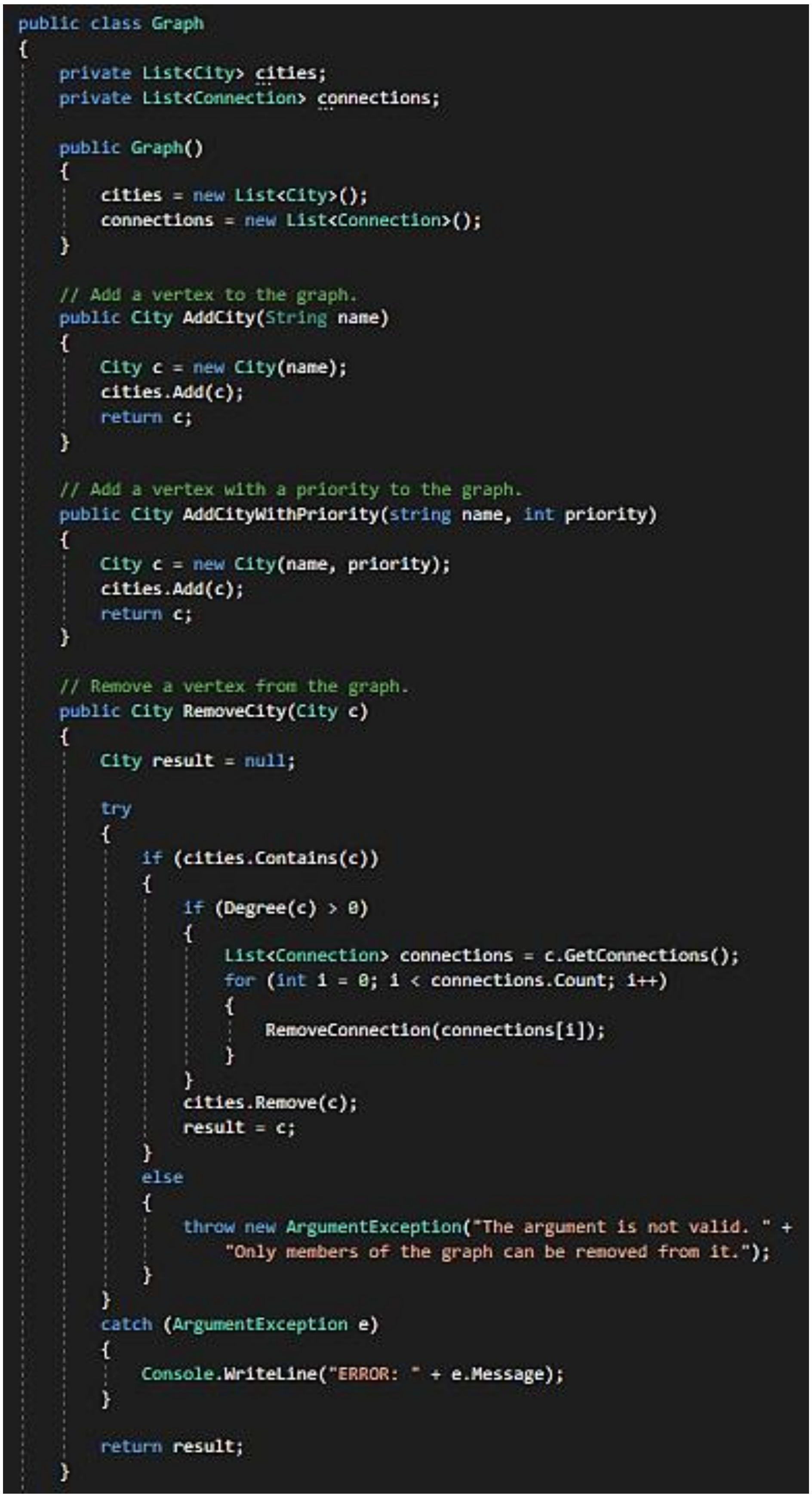
```
    public Connection AddConnection(Connection c)
    {
        connections.Add(c);
        return c;
    }

    public Connection RemoveConnection(Connection c)
    {
        connections.Remove(c);
        return c;
    }

    public void SetPriority(int priority)
    {
        this.priority = priority;
    }

    public int GetPriority()
    {
        return this.priority;
    }

    public string GetName()
    {
        return name;
    }

    public List<Connection> GetConnections()
    {
        return connections;
    }

    public override String ToString()
    {
        if (priority != -1)
        {
            return $"{name}: {priority}";
        }
        else
        {
            return name;
        }
    }
}
```

Figur 6.6: Kantklassen Connection



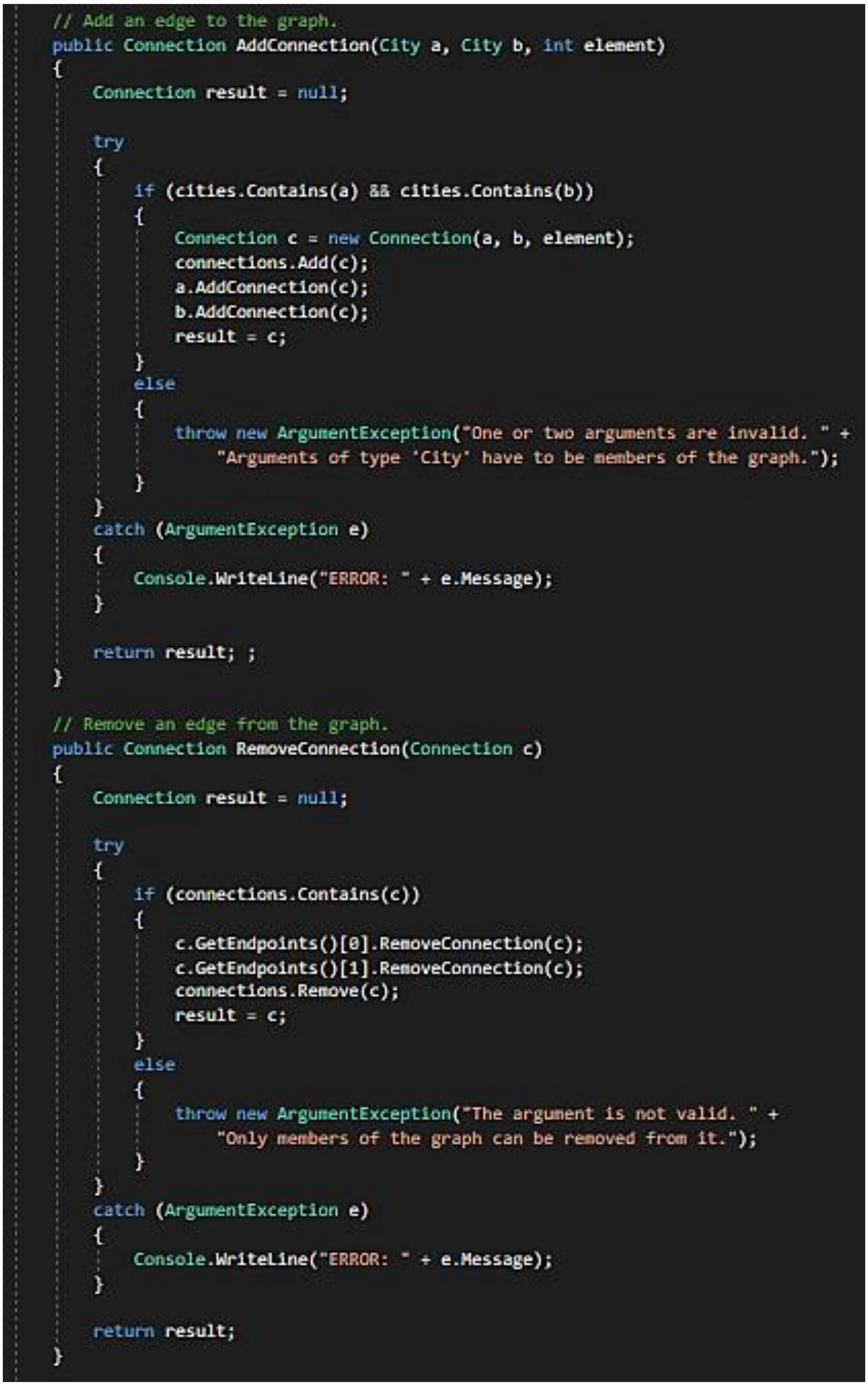
```
public class Connection
{
    private City[] endpoints;
    private int element;

    public Connection(City a, City b, int element)
    {
        try
        {
            if (a != null && b != null)
            {
                endpoints = new City[] { a, b };
                this.element = element;
            }
            else
            {
                throw new ArgumentNullException("Arguments cannot be null.");
            }
        }
        catch (ArgumentNullException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }
    }

    public City[] GetEndpoints()
    {
        return endpoints;
    }

    public void SetEndpoints(City s, City t)
    {
        if (s != null && t != null)
        {
            this.endpoints = new City[] { s, t };
        }
    }

    public int GetElement()
    {
        return element;
    }

    public override String ToString()
    {
        return $"({endpoints[0]}, {endpoints[1]}): {element}";
    }
}
```

Figur 6.7: Grafklassen Graph, del 1 af 5



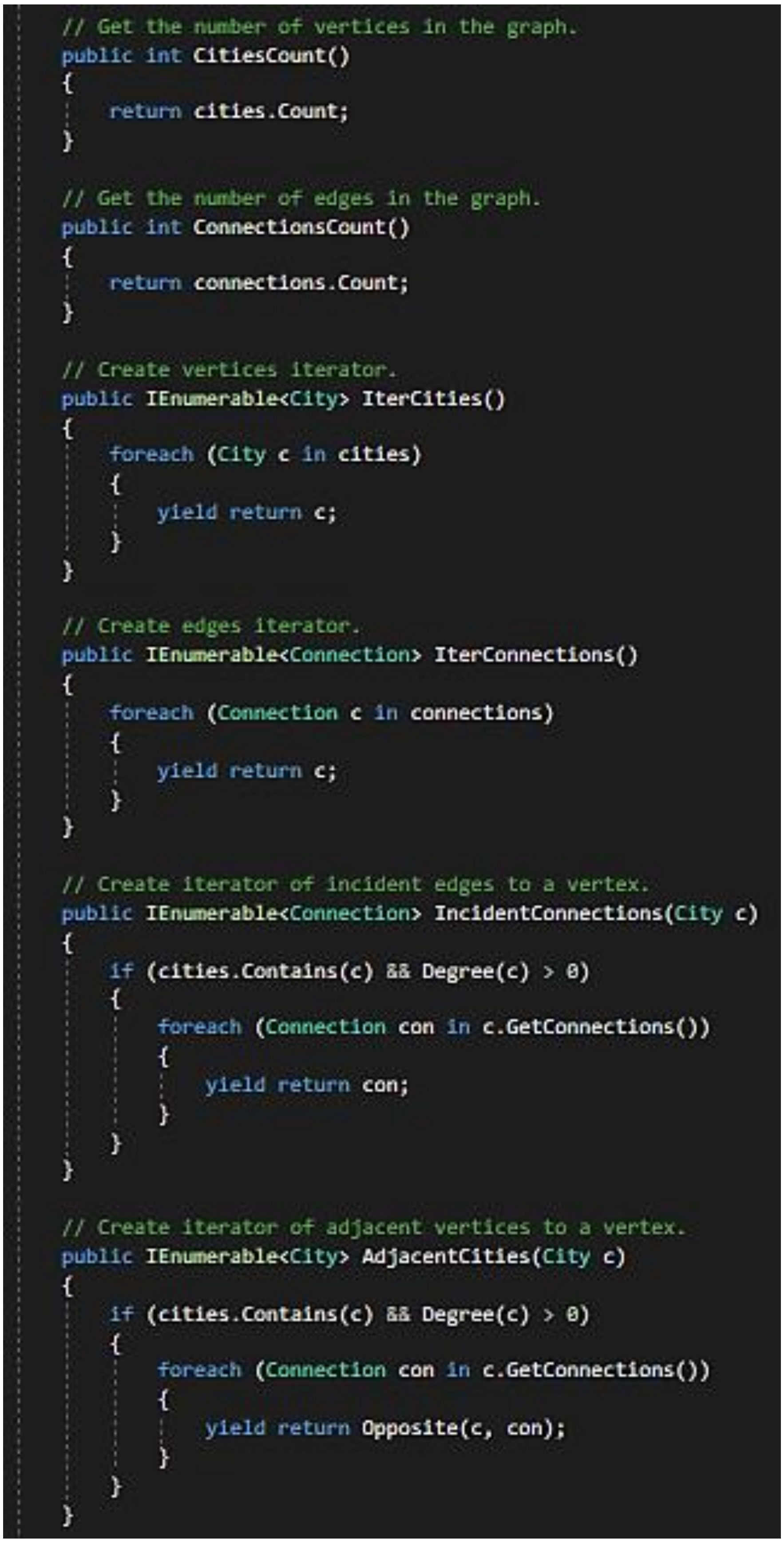
```
public class Graph
{
    private List<City> cities;
    private List<Connection> connections;

    public Graph()
    {
        cities = new List<City>();
        connections = new List<Connection>();
    }

    // Add a vertex to the graph.
    public City AddCity(String name)
    {
        City c = new City(name);
        cities.Add(c);
        return c;
    }

    // Add a vertex with a priority to the graph.
    public City AddCityWithPriority(string name, int priority)
    {
        City c = new City(name, priority);
        cities.Add(c);
        return c;
    }

    // Remove a vertex from the graph.
    public City RemoveCity(City c)
    {
        City result = null;

        try
        {
            if (cities.Contains(c))
            {
                if (Degree(c) > 0)
                {
                    List<Connection> connections = c.GetConnections();
                    for (int i = 0; i < connections.Count; i++)
                    {
                        RemoveConnection(connections[i]);
                    }
                }
                cities.Remove(c);
                result = c;
            }
            else
            {
                throw new ArgumentException("The argument is not valid. " +
                    "Only members of the graph can be removed from it.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        return result;
    }
```

Figur 6.8: Grafklassen Graph, del 2 af 5



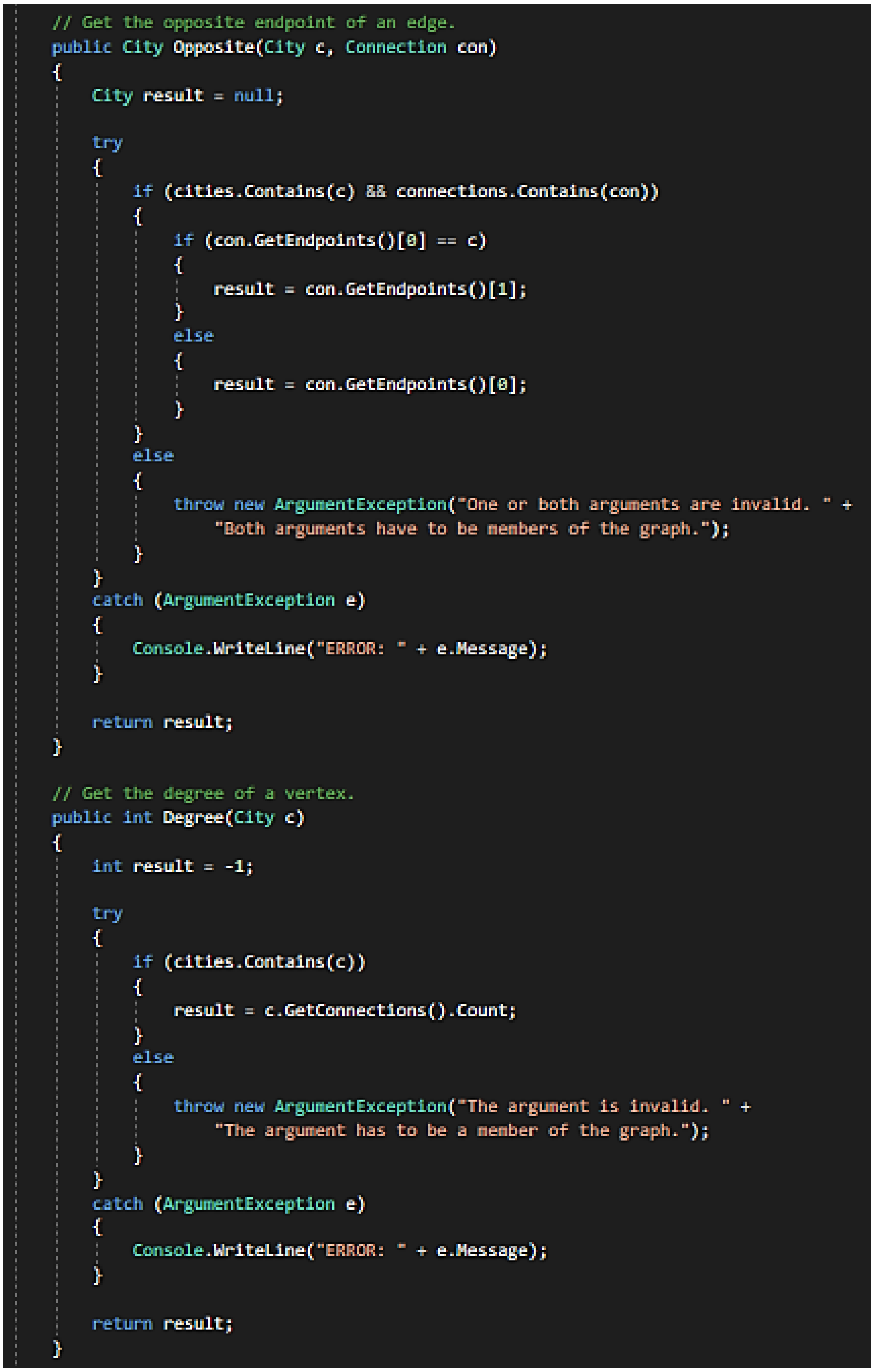
```
// Add an edge to the graph.
public Connection AddConnection(City a, City b, int element)
{
    Connection result = null;

    try
    {
        if (cities.Contains(a) && cities.Contains(b))
        {
            Connection c = new Connection(a, b, element);
            connections.Add(c);
            a.AddConnection(c);
            b.AddConnection(c);
            result = c;
        }
        else
        {
            throw new ArgumentException("One or two arguments are invalid. " +
                "Arguments of type 'City' have to members of the graph.");
        }
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("ERROR: " + e.Message);
    }

    return result; ;
}

// Remove an edge from the graph.
public Connection RemoveConnection(Connection c)
{
    Connection result = null;

    try
    {
        if (connections.Contains(c))
        {
            c.GetEndpoints()[0].RemoveConnection(c);
            c.GetEndpoints()[1].RemoveConnection(c);
            connections.Remove(c);
            result = c;
        }
        else
        {
            throw new ArgumentException("The argument is not valid. " +
                "Only members of the graph can be removed from it.");
        }
    }
    catch (ArgumentException e)
    {
        Console.WriteLine("ERROR: " + e.Message);
    }

    return result;
}
```

Figur 6.9: Grafklassen Graph, del 3 af 5



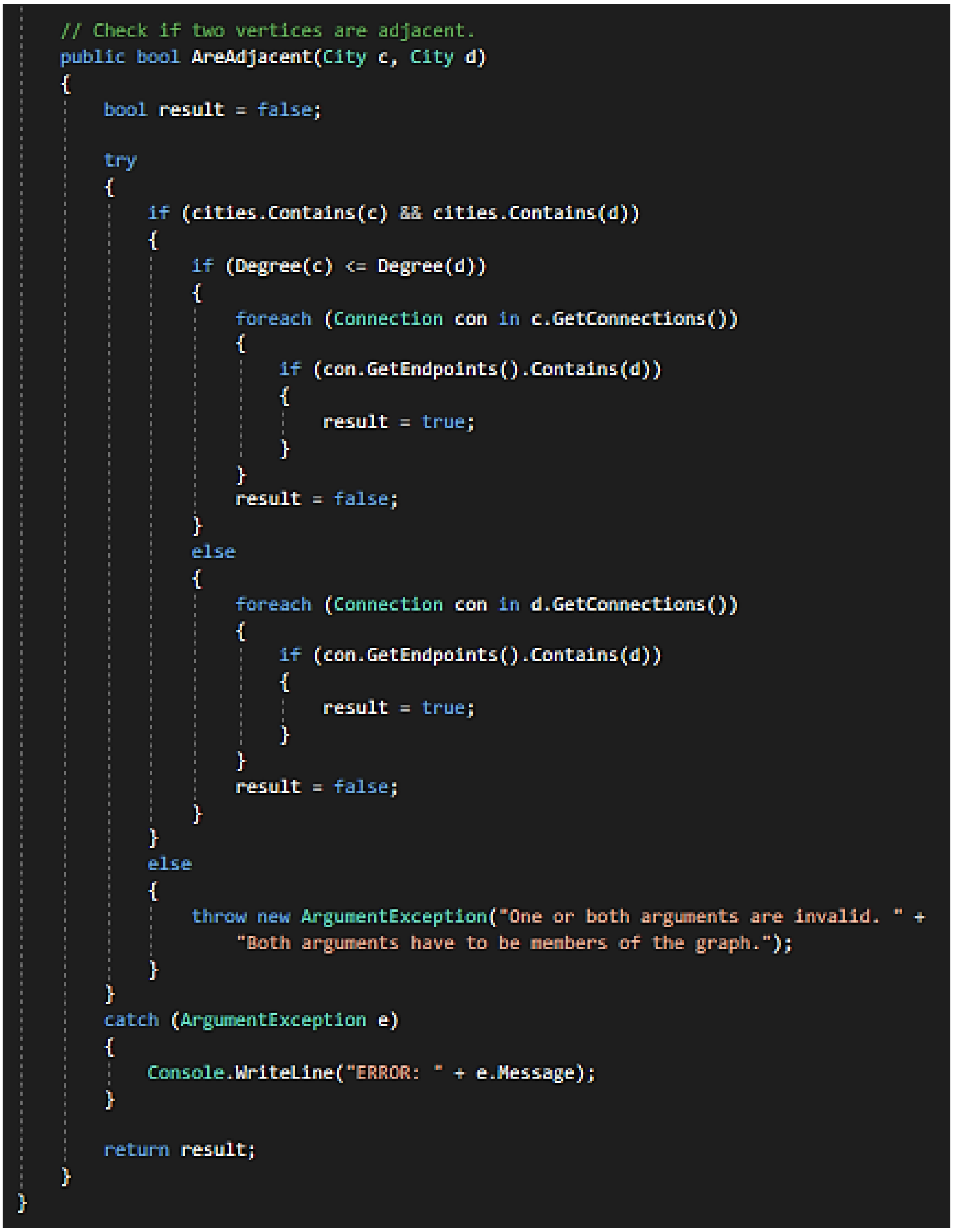
```
    // Get the number of vertices in the graph.
    public int CitiesCount()
    {
        return cities.Count;
    }

    // Get the number of edges in the graph.
    public int ConnectionsCount()
    {
        return connections.Count;
    }

    // Create vertices iterator.
    public IEnumerable<City> IterCities()
    {
        foreach (City c in cities)
        {
            yield return c;
        }
    }

    // Create edges iterator.
    public IEnumerable<Connection> IterConnections()
    {
        foreach (Connection c in connections)
        {
            yield return c;
        }
    }

    // Create iterator of incident edges to a vertex.
    public IEnumerable<Connection> IncidentConnections(City c)
    {
        if (cities.Contains(c) && Degree(c) > 0)
        {
            foreach (Connection con in c.GetConnections())
            {
                yield return con;
            }
        }
    }

    // Create iterator of adjacent vertices to a vertex.
    public IEnumerable<City> AdjacentCities(City c)
    {
        if (cities.Contains(c) && Degree(c) > 0)
        {
            foreach (Connection con in c.GetConnections())
            {
                yield return Opposite(c, con);
            }
        }
    }
```

Figur 6.10: Grafklassen Graph, del 4 af 5



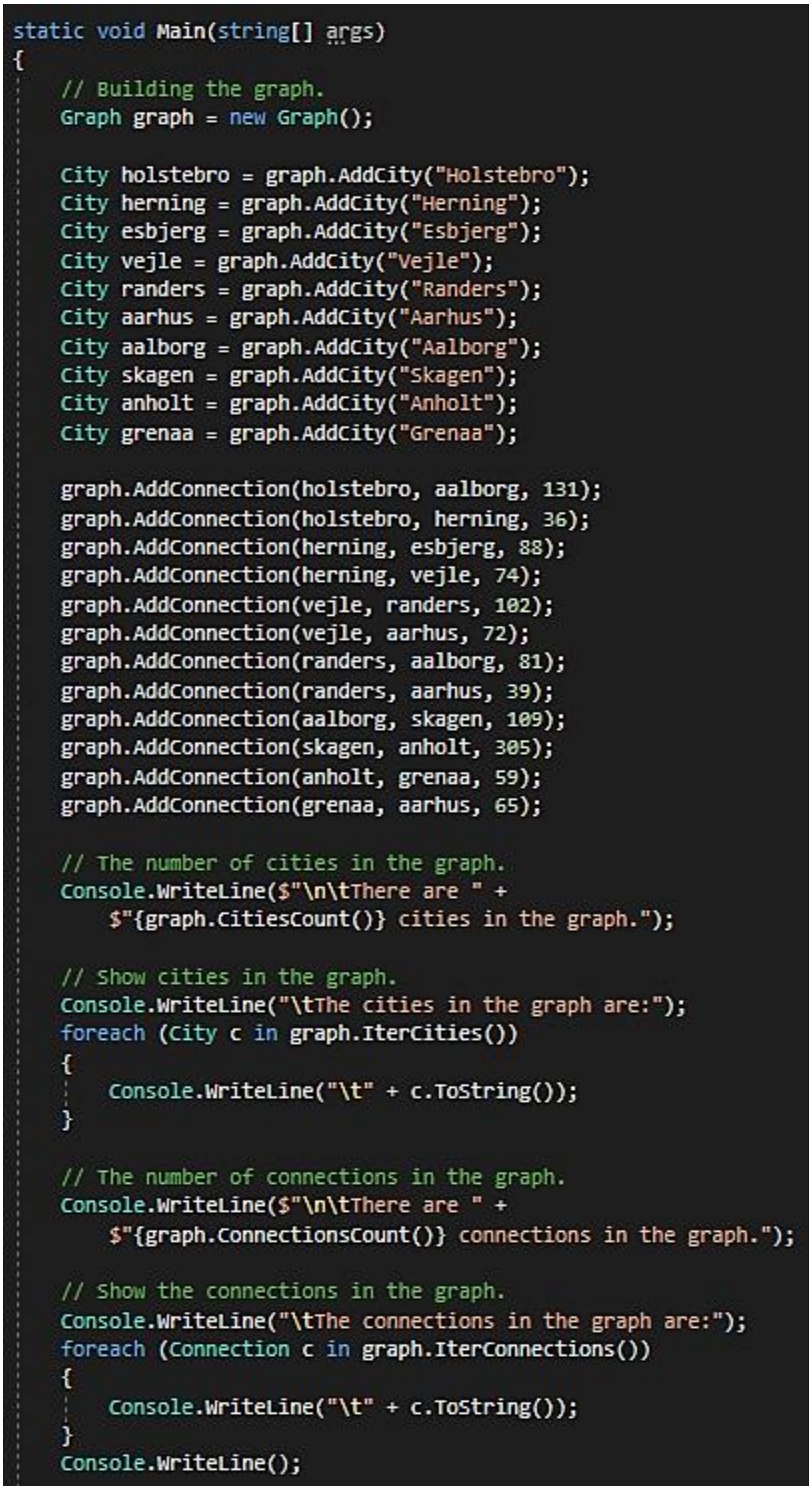
```
    // Get the opposite endpoint of an edge.
    public City Opposite(City c, Connection con)
    {
        City result = null;

        try
        {
            if (cities.Contains(c) && connections.Contains(con))
            {
                if (con.GetEndpoints()[0] == c)
                {
                    result = con.GetEndpoints()[1];
                }
                else
                {
                    result = con.GetEndpoints()[0];
                }
            }
            else
            {
                throw new ArgumentException("One or both arguments are invalid. " +
                    "Both arguments have to be members of the graph.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        return result;
    }

    // Get the degree of a vertex.
    public int Degree(City c)
    {
        int result = -1;

        try
        {
            if (cities.Contains(c))
            {
                result = c.GetConnections().Count;
            }
            else
            {
                throw new ArgumentException("The argument is invalid. " +
                    "The argument has to be a member of the graph.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        return result;
    }
```

Figur 6.11: Grafklassen Graph, del 5 af 5



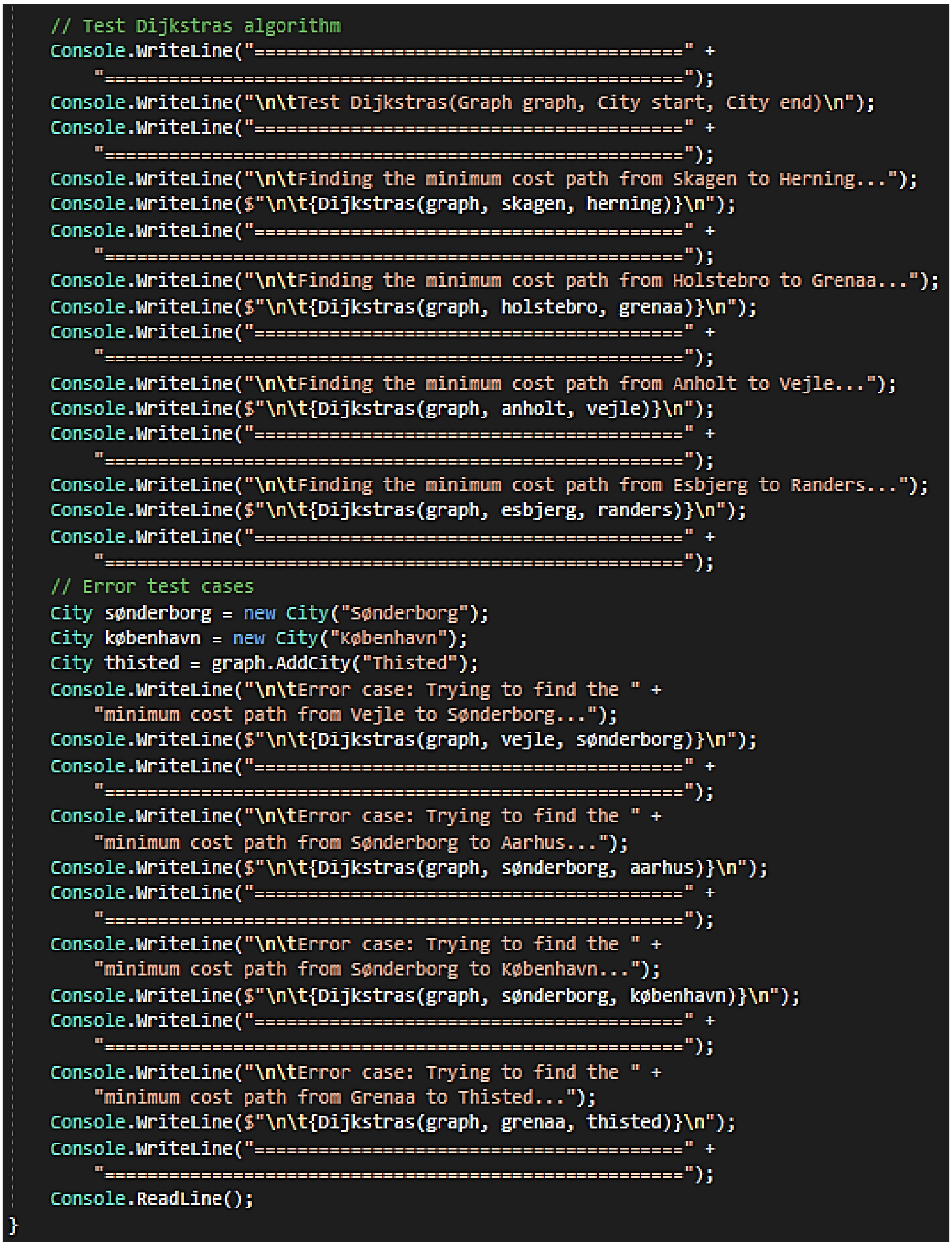
```
    // Check if two vertices are adjacent.
    public bool AreAdjacent(City c, City d)
    {
        bool result = false;

        try
        {
            if (cities.Contains(c) && cities.Contains(d))
            {
                if (Degree(c) <= Degree(d))
                {
                    foreach (Connection con in c.GetConnections())
                    {
                        if (con.GetEndpoints().Contains(d))
                        {
                            result = true;
                        }
                    }
                    result = false;
                }
                else
                {
                    foreach (Connection con in d.GetConnections())
                    {
                        if (con.GetEndpoints().Contains(d))
                        {
                            result = true;
                        }
                    }
                    result = false;
                }
            }
            else
            {
                throw new ArgumentException("One or both arguments are invalid. " +
                    "Both arguments have to be members of the graph.");
            }
        }
        catch (ArgumentException e)
        {
            Console.WriteLine("ERROR: " + e.Message);
        }

        return result;
    }
}
```

Figur 6.12: Main metoden – Opsætning til test data



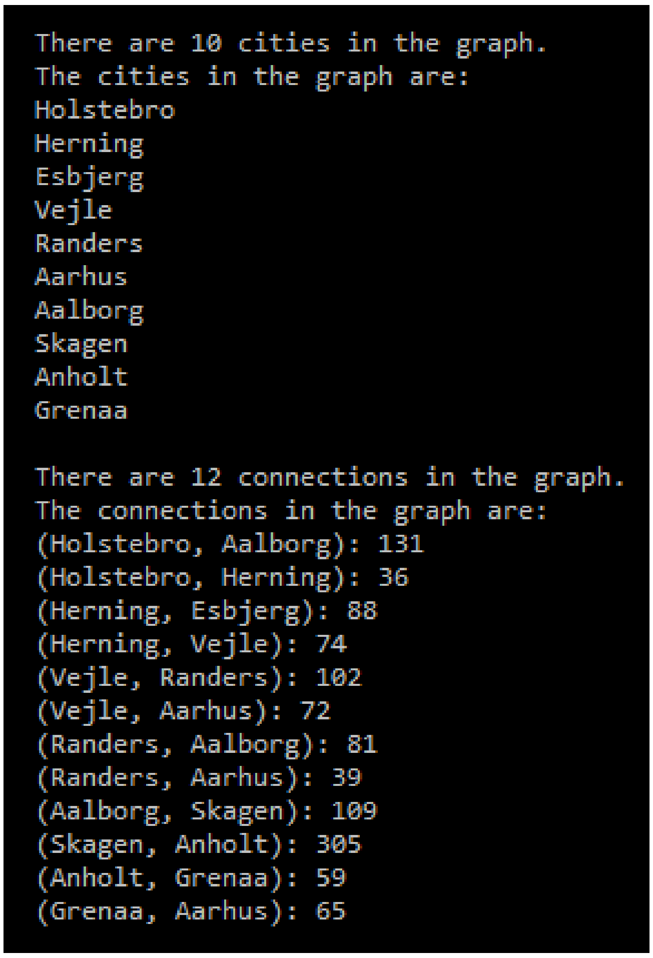
```
static void Main(string[] args)
{
    // Building the graph.
    Graph graph = new Graph();

    City holstebro = graph.AddCity("Holstebro");
    City herning = graph.AddCity("Herning");
    City esbjerg = graph.AddCity("Esbjerg");
    City vejle = graph.AddCity("Vejle");
    City randers = graph.AddCity("Randers");
    City aarhus = graph.AddCity("Aarhus");
    City aalborg = graph.AddCity("Aalborg");
    City skagen = graph.AddCity("Skagen");
    City anholt = graph.AddCity("Anholt");
    City grenaa = graph.AddCity("Grenaa");

    graph.AddConnection(holstebro, aalborg, 131);
    graph.AddConnection(holstebro, herning, 36);
    graph.AddConnection(herning, esbjerg, 88);
    graph.AddConnection(herning, vejle, 74);
    graph.AddConnection(vejle, randers, 102);
    graph.AddConnection(vejle, aarhus, 72);
    graph.AddConnection(randers, aalborg, 81);
    graph.AddConnection(randers, aarhus, 39);
    graph.AddConnection(aalborg, skagen, 109);
    graph.AddConnection(skagen, anholt, 305);
    graph.AddConnection(anholt, grenaa, 59);
    graph.AddConnection(grenaa, aarhus, 65);

    // The number of cities in the graph.
    Console.WriteLine($"\n\tThere are " +
        $"{graph.CitiesCount()} cities in the graph.");

    // Show cities in the graph.
    Console.WriteLine("\tThe cities in the graph are:");
    foreach (City c in graph.IterCities())
    {
        Console.WriteLine("\t" + c.ToString());
    }

    // The number of connections in the graph.
    Console.WriteLine($"\n\tThere are " +
        $"{graph.ConnectionsCount()} connections in the graph.");

    // Show the connections in the graph.
    Console.WriteLine("\tThe connections in the graph are:");
    foreach (Connection c in graph.IterConnections())
    {
        Console.WriteLine("\t" + c.ToString());
    }
    Console.WriteLine();
```

Figur 6.13: Main metoden - Test data

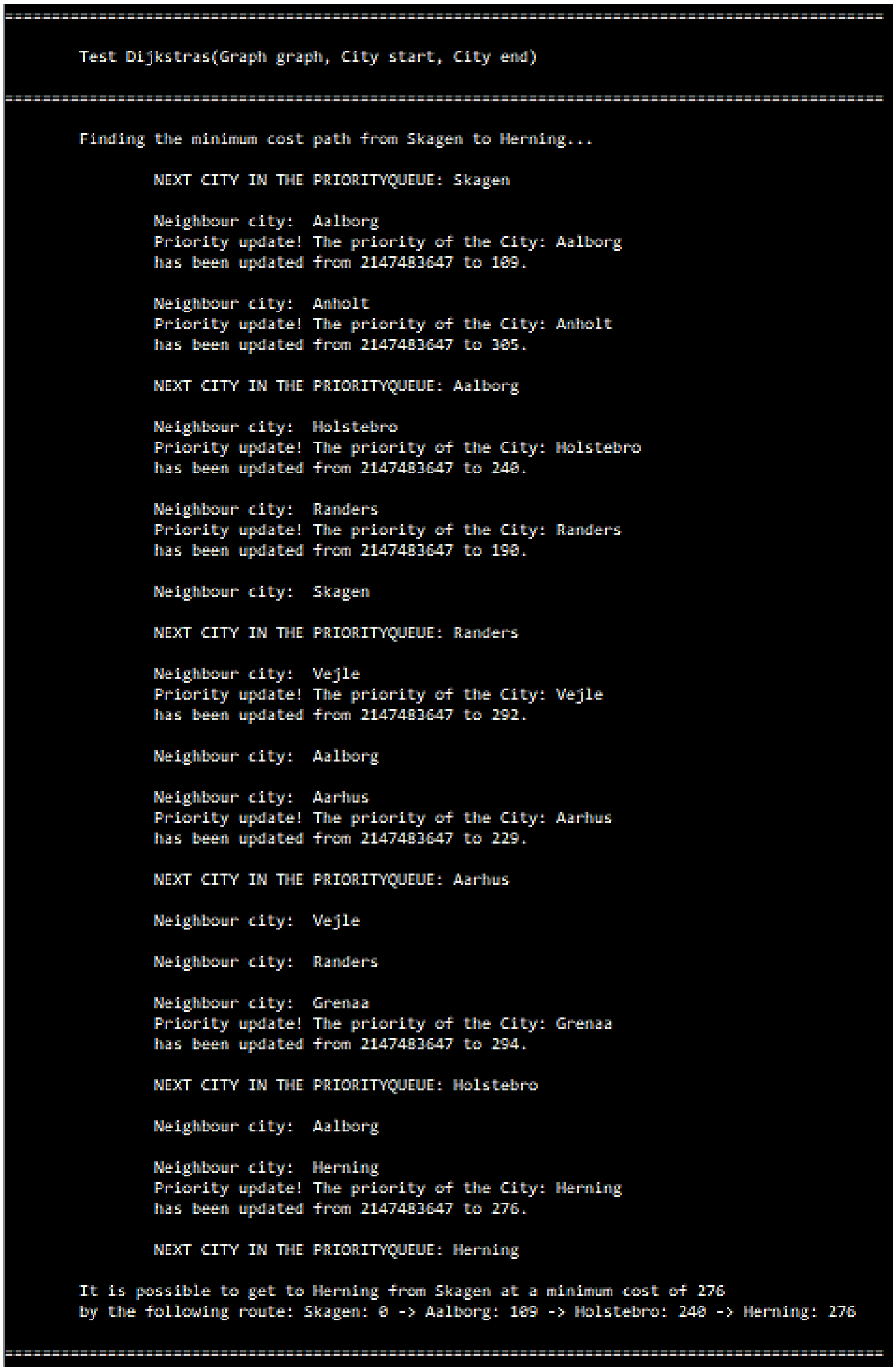


```
    // Test Dijkstras algorithm
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tTest Dijkstras(Graph graph, City start, City end)\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tFinding the minimum cost path from Skagen to Herning...");
    Console.WriteLine($"\n\t{Dijkstras(graph, skagen, herning)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tFinding the minimum cost path from Holstebro to Grenaa...");
    Console.WriteLine($"\n\t{Dijkstras(graph, holstebro, grenaa)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tFinding the minimum cost path from Anholt to Vejle...");
    Console.WriteLine($"\n\t{Dijkstras(graph, anholt, vejle)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tFinding the minimum cost path from Esbjerg to Randers...");
    Console.WriteLine($"\n\t{Dijkstras(graph, esbjerg, randers)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    // Error test cases
    City sønderborg = new City("Sønderborg");
    City københavn = new City("København");
    City thisted = graph.AddCity("Thisted");
    Console.WriteLine("\n\tError case: Trying to find the " +
        "minimum cost path from Vejle to Sønderborg...");
    Console.WriteLine($"\n\t{Dijkstras(graph, vejle, sønderborg)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tError case: Trying to find the " +
        "minimum cost path from Sønderborg to Aarhus...");
    Console.WriteLine($"\n\t{Dijkstras(graph, sønderborg, aarhus)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tError case: Trying to find the " +
        "minimum cost path from Sønderborg to København...");
    Console.WriteLine($"\n\t{Dijkstras(graph, sønderborg, københavn)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.WriteLine("\n\tError case: Trying to find the " +
        "minimum cost path from Grenaa to Thisted...");
    Console.WriteLine($"\n\t{Dijkstras(graph, grenaa, thisted)}\n");
    Console.WriteLine("========================================" +
        "======================================================");
    Console.ReadLine();
}
```

Figur 6.14: Output data fra Main metoden – Opsætningen til test data

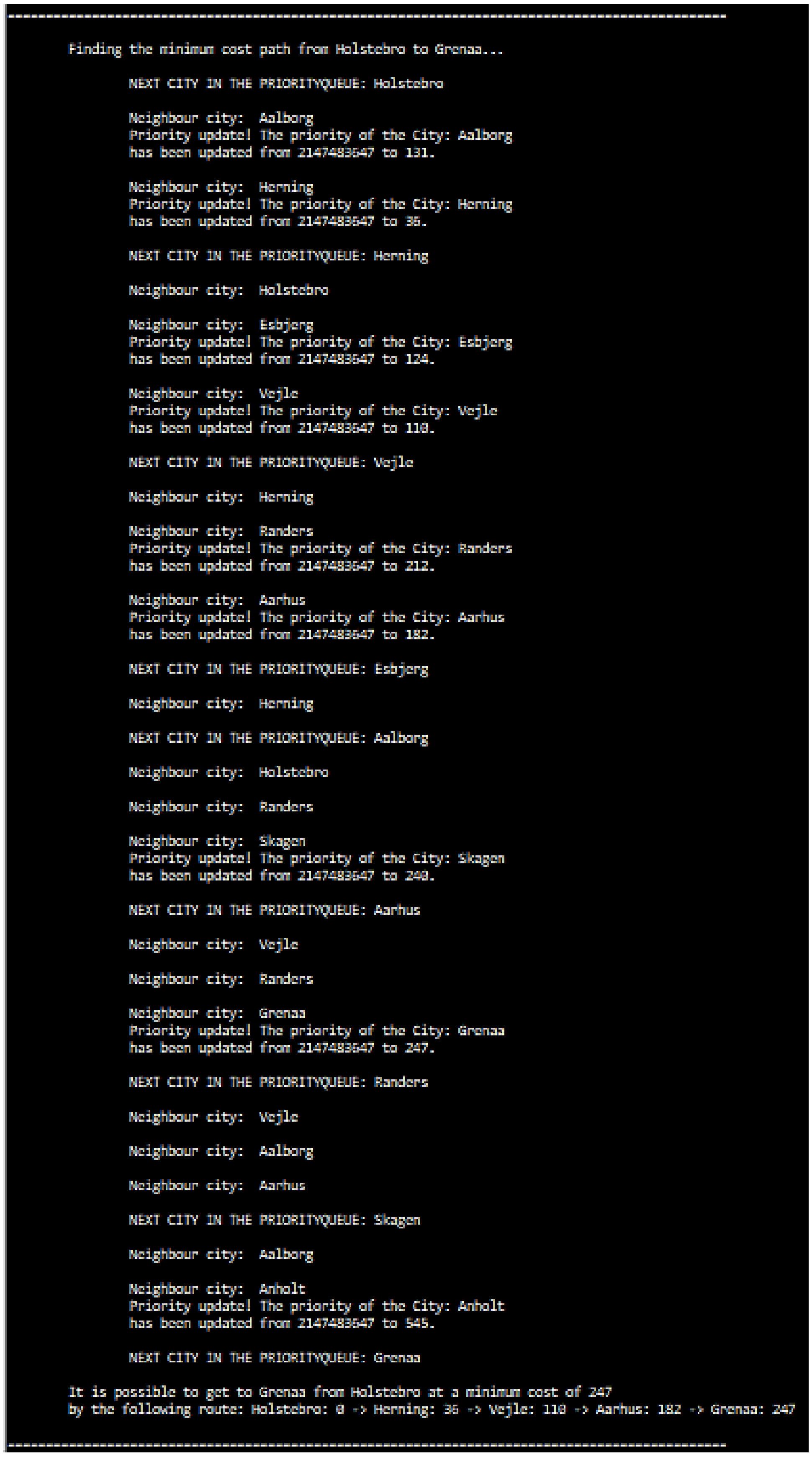

```
There are 10 cities in the graph.
The cities in the graph are:
Holstebro
Herning
Esbjerg
Vejle
Randers
Aarhus
Aalborg
Skagen
Anholt
Grenaa

There are 12 connections in the graph.
The connections in the graph are:
(Holstebro, Aalborg): 131
(Holstebro, Herning): 36
(Herning, Esbjerg): 88
(Herning, Vejle): 74
(Vejle, Randers): 102
(Vejle, Aarhus): 72
(Randers, Aalborg): 81
(Randers, Aarhus): 39
(Aalborg, Skagen): 109
(Skagen, Anholt): 305
(Anholt, Grenaa): 59
(Grenaa, Aarhus): 65
```

Figur 6.15: Output data fra Main metoden – Test 1



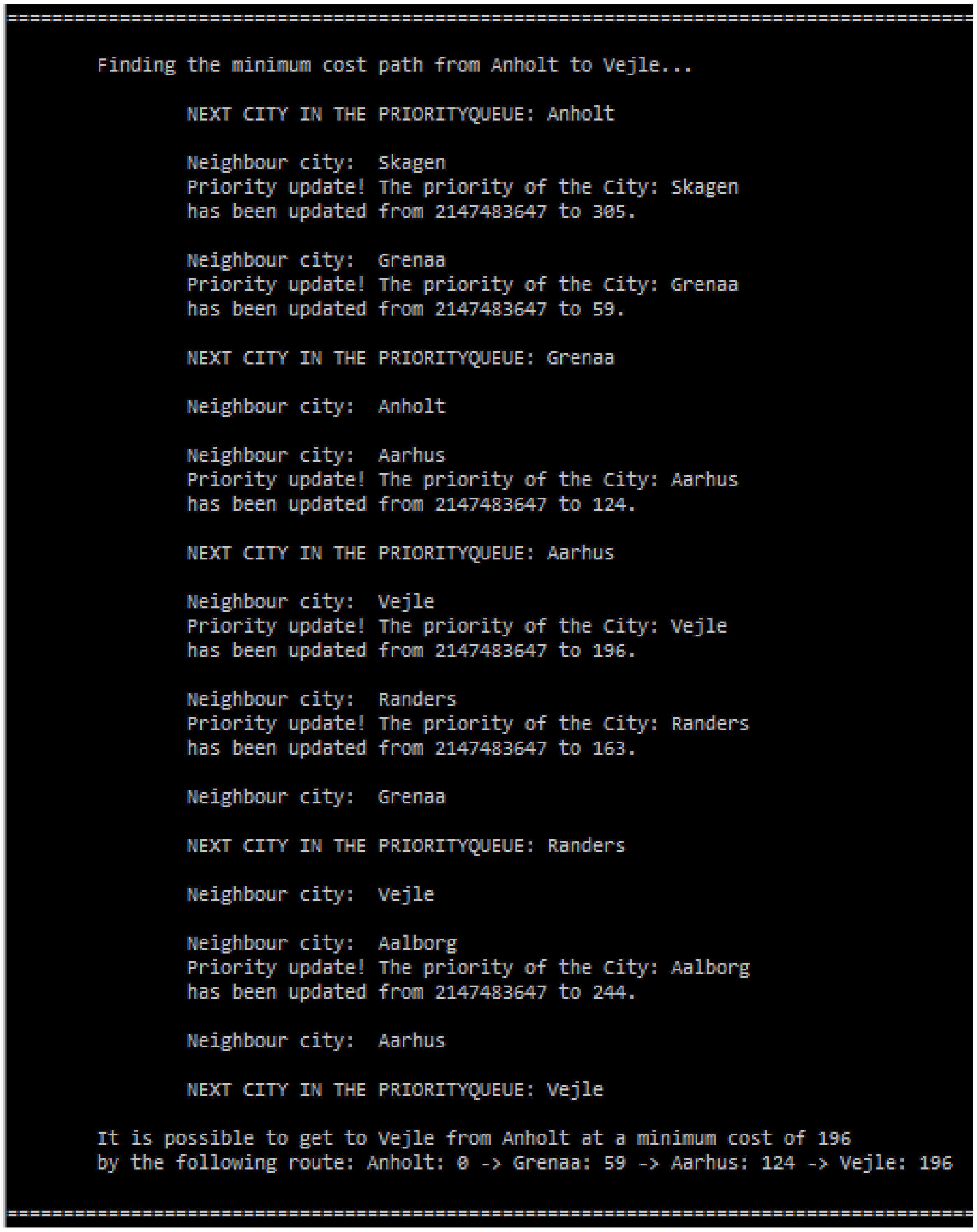
```
===========================================================================================

       Test Dijkstras(Graph graph, City start, City end)

===========================================================================================

       Finding the minimum cost path from Skagen to Herning...

               NEXT CITY IN THE PRIORITYQUEUE: Skagen

               Neighbour city:  Aalborg
               Priority update! The priority of the City: Aalborg
               has been updated from 2147483647 to 109.

               Neighbour city:  Anholt
               Priority update! The priority of the City: Anholt
               has been updated from 2147483647 to 305.

               NEXT CITY IN THE PRIORITYQUEUE: Aalborg

               Neighbour city:  Holstebro
               Priority update! The priority of the City: Holstebro
               has been updated from 2147483647 to 240.

               Neighbour city:  Randers
               Priority update! The priority of the City: Randers
               has been updated from 2147483647 to 190.

               Neighbour city:  Skagen

               NEXT CITY IN THE PRIORITYQUEUE: Randers

               Neighbour city:  Vejle
               Priority update! The priority of the City: Vejle
               has been updated from 2147483647 to 292.

               Neighbour city:  Aalborg

               Neighbour city:  Aarhus
               Priority update! The priority of the City: Aarhus
               has been updated from 2147483647 to 229.

               NEXT CITY IN THE PRIORITYQUEUE: Aarhus

               Neighbour city:  Vejle

               Neighbour city:  Randers

               Neighbour city:  Grenaa
               Priority update! The priority of the City: Grenaa
               has been updated from 2147483647 to 294.

               NEXT CITY IN THE PRIORITYQUEUE: Holstebro

               Neighbour city:  Aalborg

               Neighbour city:  Herning
               Priority update! The priority of the City: Herning
               has been updated from 2147483647 to 276.

               NEXT CITY IN THE PRIORITYQUEUE: Herning

       It is possible to get to Herning from Skagen at a minimum cost of 276
       by the following route: Skagen: 0 -> Aalborg: 109 -> Holstebro: 240 -> Herning: 276

===========================================================================================
```

Figur 6.16: Output data fra Main metoden – Test 2



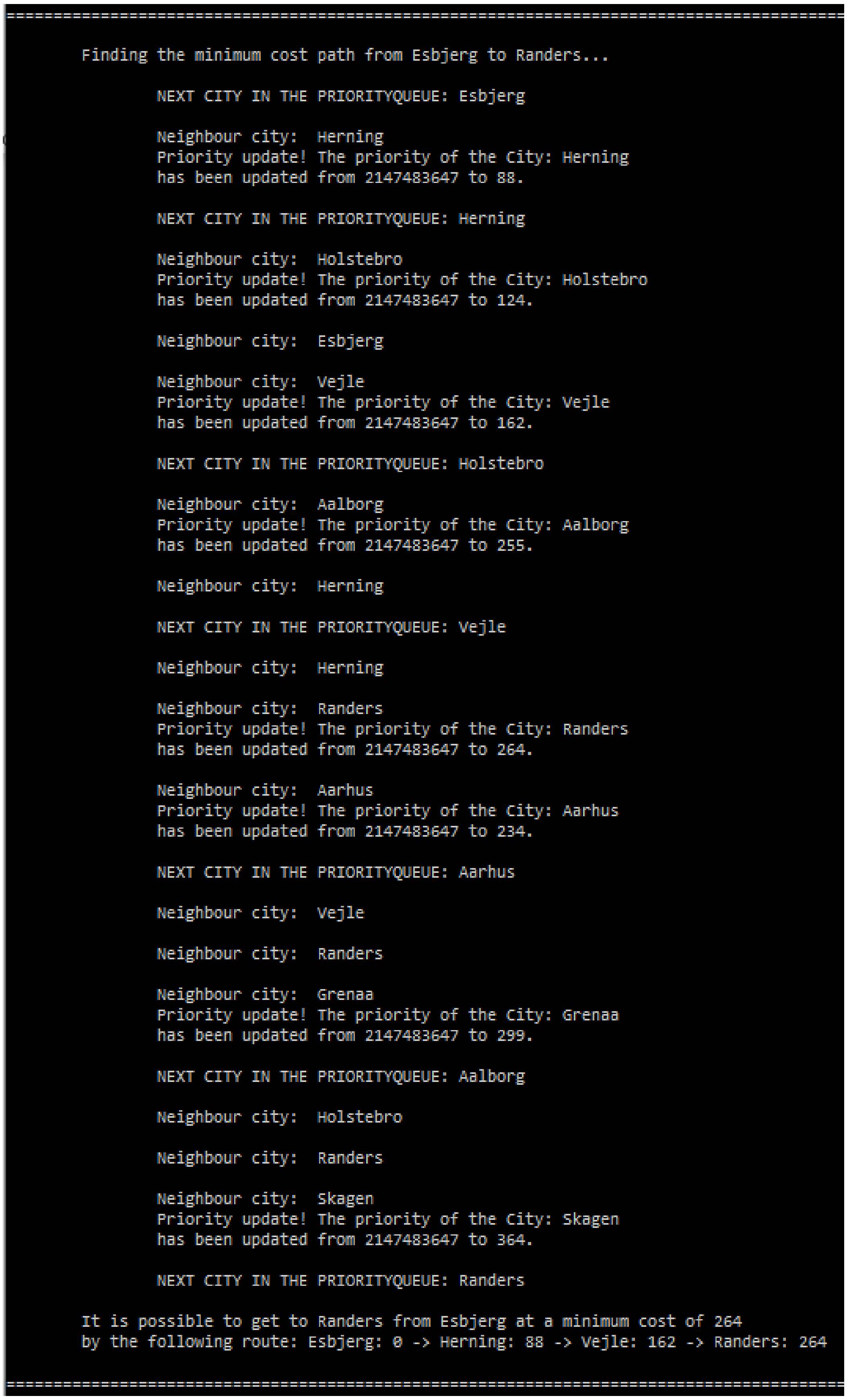
```
----------------------------------------------------------------------------------------------

        Finding the minimum cost path from Holstebro to Grenaa...

                NEXT CITY IN THE PRIORITYQUEUE: Holstebro

                Neighbour city:  Aalborg
                Priority update! The priority of the City: Aalborg
                has been updated from 2147483647 to 131.

                Neighbour city:  Herning
                Priority update! The priority of the City: Herning
                has been updated from 2147483647 to 36.

                NEXT CITY IN THE PRIORITYQUEUE: Herning

                Neighbour city:  Holstebro

                Neighbour city:  Esbjerg
                Priority update! The priority of the City: Esbjerg
                has been updated from 2147483647 to 124.

                Neighbour city:  Vejle
                Priority update! The priority of the City: Vejle
                has been updated from 2147483647 to 110.

                NEXT CITY IN THE PRIORITYQUEUE: Vejle

                Neighbour city:  Herning

                Neighbour city:  Randers
                Priority update! The priority of the City: Randers
                has been updated from 2147483647 to 212.

                Neighbour city:  Aarhus
                Priority update! The priority of the City: Aarhus
                has been updated from 2147483647 to 182.

                NEXT CITY IN THE PRIORITYQUEUE: Esbjerg

                Neighbour city:  Herning

                NEXT CITY IN THE PRIORITYQUEUE: Aalborg

                Neighbour city:  Holstebro

                Neighbour city:  Randers

                Neighbour city:  Skagen
                Priority update! The priority of the City: Skagen
                has been updated from 2147483647 to 240.

                NEXT CITY IN THE PRIORITYQUEUE: Aarhus

                Neighbour city:  Vejle

                Neighbour city:  Randers

                Neighbour city:  Grenaa
                Priority update! The priority of the City: Grenaa
                has been updated from 2147483647 to 247.

                NEXT CITY IN THE PRIORITYQUEUE: Randers

                Neighbour city:  Vejle

                Neighbour city:  Aalborg

                Neighbour city:  Aarhus

                NEXT CITY IN THE PRIORITYQUEUE: Skagen

                Neighbour city:  Aalborg

                Neighbour city:  Anholt
                Priority update! The priority of the City: Anholt
                has been updated from 2147483647 to 545.

                NEXT CITY IN THE PRIORITYQUEUE: Grenaa

        It is possible to get to Grenaa from Holstebro at a minimum cost of 247
        by the following route: Holstebro: 0 -> Herning: 36 -> Vejle: 110 -> Aarhus: 182 -> Grenaa: 247

----------------------------------------------------------------------------------------------
```

Figur 6.17: Output data fra Main metoden – Test 3



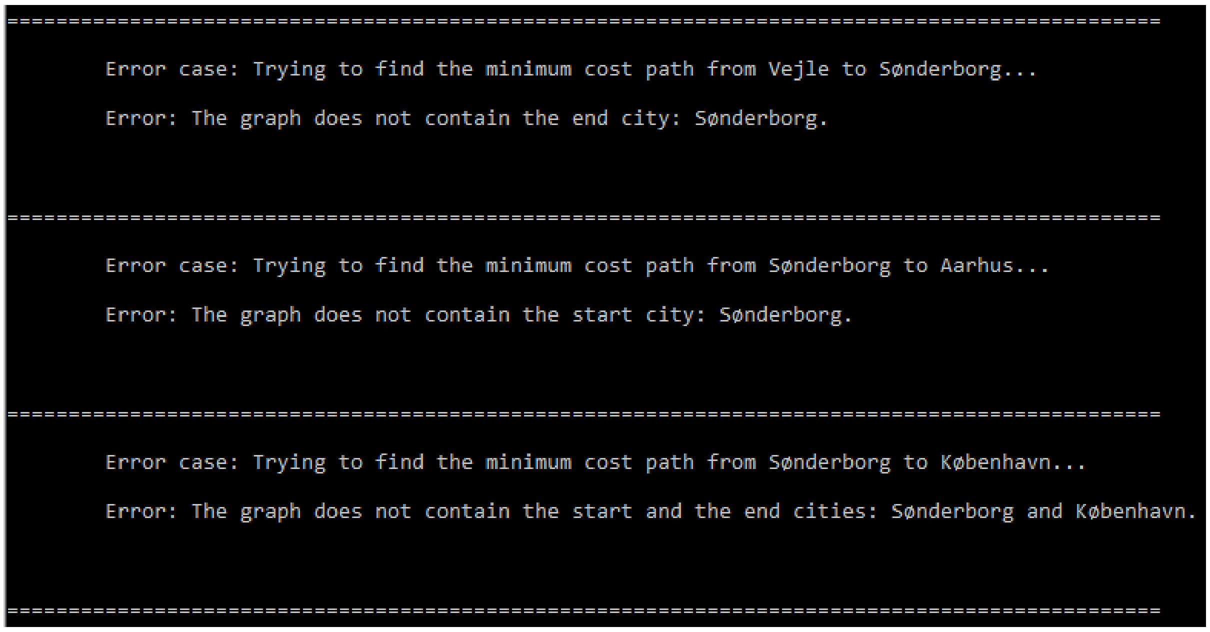
```
================================================================================

       Finding the minimum cost path from Anholt to Vejle...

               NEXT CITY IN THE PRIORITYQUEUE: Anholt

               Neighbour city:  Skagen
               Priority update! The priority of the City: Skagen
               has been updated from 2147483647 to 305.

               Neighbour city:  Grenaa
               Priority update! The priority of the City: Grenaa
               has been updated from 2147483647 to 59.

               NEXT CITY IN THE PRIORITYQUEUE: Grenaa

               Neighbour city:  Anholt

               Neighbour city:  Aarhus
               Priority update! The priority of the City: Aarhus
               has been updated from 2147483647 to 124.

               NEXT CITY IN THE PRIORITYQUEUE: Aarhus

               Neighbour city:  Vejle
               Priority update! The priority of the City: Vejle
               has been updated from 2147483647 to 196.

               Neighbour city:  Randers
               Priority update! The priority of the City: Randers
               has been updated from 2147483647 to 163.

               Neighbour city:  Grenaa

               NEXT CITY IN THE PRIORITYQUEUE: Randers

               Neighbour city:  Vejle

               Neighbour city:  Aalborg
               Priority update! The priority of the City: Aalborg
               has been updated from 2147483647 to 244.

               Neighbour city:  Aarhus

               NEXT CITY IN THE PRIORITYQUEUE: Vejle

       It is possible to get to Vejle from Anholt at a minimum cost of 196
       by the following route: Anholt: 0 -> Grenaa: 59 -> Aarhus: 124 -> Vejle: 196

================================================================================
```

Figur 6.18: Output data fra Main metoden – Test 4



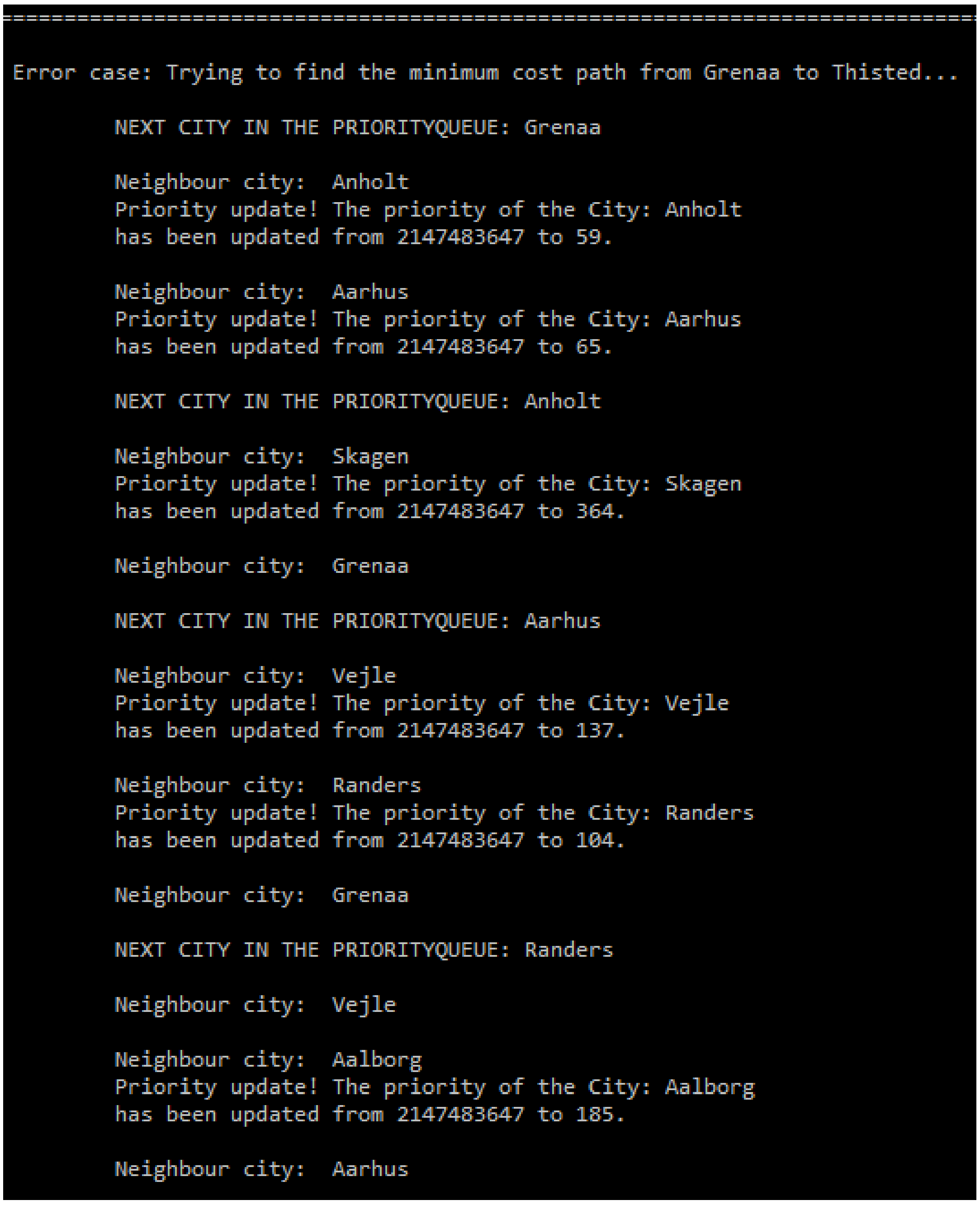
```
================================================================================

        Finding the minimum cost path from Esbjerg to Randers...

                NEXT CITY IN THE PRIORITYQUEUE: Esbjerg

                Neighbour city:  Herning
                Priority update! The priority of the City: Herning
                has been updated from 2147483647 to 88.

                NEXT CITY IN THE PRIORITYQUEUE: Herning

                Neighbour city:  Holstebro
                Priority update! The priority of the City: Holstebro
                has been updated from 2147483647 to 124.

                Neighbour city:  Esbjerg

                Neighbour city:  Vejle
                Priority update! The priority of the City: Vejle
                has been updated from 2147483647 to 162.

                NEXT CITY IN THE PRIORITYQUEUE: Holstebro

                Neighbour city:  Aalborg
                Priority update! The priority of the City: Aalborg
                has been updated from 2147483647 to 255.

                Neighbour city:  Herning

                NEXT CITY IN THE PRIORITYQUEUE: Vejle

                Neighbour city:  Herning

                Neighbour city:  Randers
                Priority update! The priority of the City: Randers
                has been updated from 2147483647 to 264.

                Neighbour city:  Aarhus
                Priority update! The priority of the City: Aarhus
                has been updated from 2147483647 to 234.

                NEXT CITY IN THE PRIORITYQUEUE: Aarhus

                Neighbour city:  Vejle

                Neighbour city:  Randers

                Neighbour city:  Grenaa
                Priority update! The priority of the City: Grenaa
                has been updated from 2147483647 to 299.

                NEXT CITY IN THE PRIORITYQUEUE: Aalborg

                Neighbour city:  Holstebro

                Neighbour city:  Randers

                Neighbour city:  Skagen
                Priority update! The priority of the City: Skagen
                has been updated from 2147483647 to 364.

                NEXT CITY IN THE PRIORITYQUEUE: Randers

        It is possible to get to Randers from Esbjerg at a minimum cost of 264
        by the following route: Esbjerg: 0 -> Herning: 88 -> Vejle: 162 -> Randers: 264

================================================================================
```

Figur 6.19: Output data fra Main metoden – Error test cases 1, 2 og 3



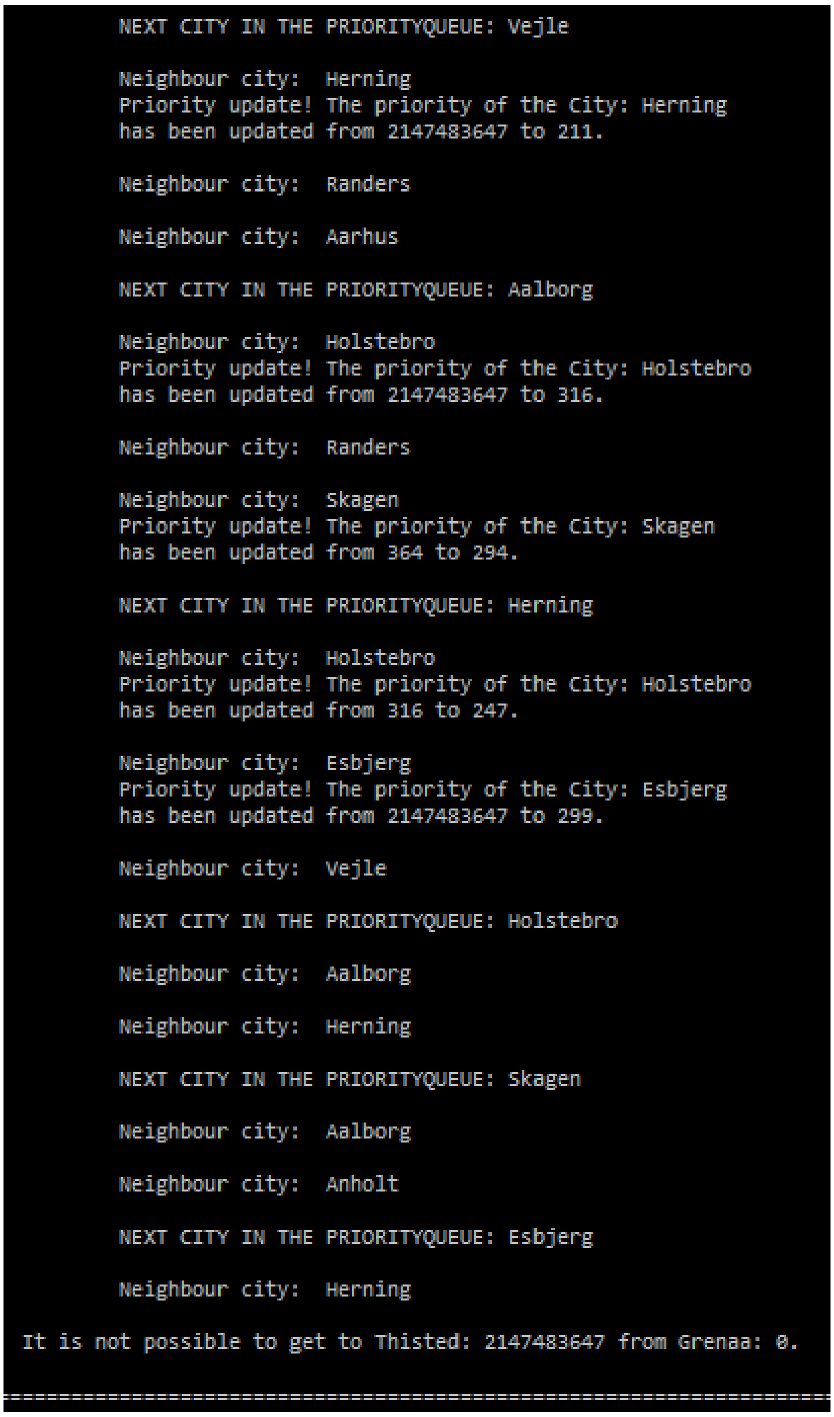
```
=============================================================================================

        Error case: Trying to find the minimum cost path from Vejle to Sønderborg...

        Error: The graph does not contain the end city: Sønderborg.


=============================================================================================

        Error case: Trying to find the minimum cost path from Sønderborg to Aarhus...

        Error: The graph does not contain the start city: Sønderborg.


=============================================================================================

        Error case: Trying to find the minimum cost path from Sønderborg to København...

        Error: The graph does not contain the start and the end cities: Sønderborg and København.


=============================================================================================
```

Figur 6.20: Output data fra Main metoden – Error test case 4, del 1 af 2



```
================================================================================

Error case: Trying to find the minimum cost path from Grenaa to Thisted...

        NEXT CITY IN THE PRIORITYQUEUE: Grenaa

        Neighbour city:  Anholt
        Priority update! The priority of the City: Anholt
        has been updated from 2147483647 to 59.

        Neighbour city:  Aarhus
        Priority update! The priority of the City: Aarhus
        has been updated from 2147483647 to 65.

        NEXT CITY IN THE PRIORITYQUEUE: Anholt

        Neighbour city:  Skagen
        Priority update! The priority of the City: Skagen
        has been updated from 2147483647 to 364.

        Neighbour city:  Grenaa

        NEXT CITY IN THE PRIORITYQUEUE: Aarhus

        Neighbour city:  Vejle
        Priority update! The priority of the City: Vejle
        has been updated from 2147483647 to 137.

        Neighbour city:  Randers
        Priority update! The priority of the City: Randers
        has been updated from 2147483647 to 104.

        Neighbour city:  Grenaa

        NEXT CITY IN THE PRIORITYQUEUE: Randers

        Neighbour city:  Vejle

        Neighbour city:  Aalborg
        Priority update! The priority of the City: Aalborg
        has been updated from 2147483647 to 185.

        Neighbour city:  Aarhus
```

Figur 6.21: Output data fra Main metoden – Error test case 4, del 2 af 2



```
        NEXT CITY IN THE PRIORITYQUEUE: Vejle

        Neighbour city:  Herning
        Priority update! The priority of the City: Herning
        has been updated from 2147483647 to 211.

        Neighbour city:  Randers

        Neighbour city:  Aarhus

        NEXT CITY IN THE PRIORITYQUEUE: Aalborg

        Neighbour city:  Holstebro
        Priority update! The priority of the City: Holstebro
        has been updated from 2147483647 to 316.

        Neighbour city:  Randers

        Neighbour city:  Skagen
        Priority update! The priority of the City: Skagen
        has been updated from 364 to 294.

        NEXT CITY IN THE PRIORITYQUEUE: Herning

        Neighbour city:  Holstebro
        Priority update! The priority of the City: Holstebro
        has been updated from 316 to 247.

        Neighbour city:  Esbjerg
        Priority update! The priority of the City: Esbjerg
        has been updated from 2147483647 to 299.

        Neighbour city:  Vejle

        NEXT CITY IN THE PRIORITYQUEUE: Holstebro

        Neighbour city:  Aalborg

        Neighbour city:  Herning

        NEXT CITY IN THE PRIORITYQUEUE: Skagen

        Neighbour city:  Aalborg

        Neighbour city:  Anholt

        NEXT CITY IN THE PRIORITYQUEUE: Esbjerg

        Neighbour city:  Herning

 It is not possible to get to Thisted: 2147483647 from Grenaa: 0.

====================================================================
```