```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Text;
using System.Diagnostics;
using System.IO;

/* The problem is to move through a matrix where each cell is weighted.
 * Movement starts at any row in the first column of the matrix
 * Movemment progresses by moving one column forward an either
 * moveing to the same row or up and down a row
 * Weights are accumulated form each cell traversed
 * must exit matri on the right most column with a weight of 50 or less.
 * Marix may vary from 1x5 to 10 x 100
 **/


namespace JerryWebMVC.Classes
{
    public class WaterTransit
    {
        private int WeightLimit = 50;
        private int maxRow;
        private int maxCol;
        private int maxRowPerm;
        private int curRow;
        private int curCol;
        private int CulWeight = 0;
        public bool FoundRoute = false;
        public int[,] Matrix;
        public int SampleCount = 0;
        private WaterStep[] WaterSteps; // use an array because it is an ordered list
        private List<WaterRoute> WaterRoutes = new List<WaterRoute>();


        public WaterTransit()
        { }



        public WaterRoute LeastResitant()
        {
            int wt = 1000000;
            WaterRoute _route = null;
            foreach (var r in WaterRoutes)
            {
                if (r.TotalWeight < wt)
                {
                    wt = r.TotalWeight;
                    _route = r;
                }
            }
            return _route;
        }

        private void Setup(int numrows, int numcols)
        {
            Matrix = new int[numrows, numcols];
            maxRow = numrows - 1;
            maxCol = numcols - 1;
            maxRowPerm = Math.Min(numrows, 3) * (numcols - 1);

        }

        public void AutoMatrix(int numrows, int numcols, bool allintgers)
        {
```

```
            Setup(numrows, numcols);
            WaterSteps = new WaterStep[numcols];


            Stopwatch sw = new Stopwatch();
            sw.Start();

            for (int j = 0; j < numcols; j++)
            {
                for (int i = 0; i < numrows; i++)
                {
                    Random rand = new Random((int)(sw.ElapsedTicks/1000));
                    // i need negative and positive numbers
                    ///TODO change 50/25 to a number based on teh size of the matrix
                    ///with an input of variacne for success
                    if (allintgers)
                    {
                        Matrix[i, j] = (int)((rand.NextDouble() * 50) - 25);
                    }
                    else
                    {
                        Matrix[i, j] = (int)((rand.NextDouble() * 50));

                    }

                    for (int p = 0; p < 50000; p++)
                    { }
                }

            }
            sw.Stop();
        }

        private void ProcessLine(string line, int row)
        {
            int i = 0;
            int pos;
            int val;
            char[] ch = new char[2] {',','\n'};
            while (!string.IsNullOrEmpty(line))
            {
                pos = line.IndexOfAny(ch);
                string tst = pos==-1 ? line : line.Substring(0, pos);
                int.TryParse(tst, out val);
                Matrix[row,i ] = val;
                i++;
                line = pos == -1? string.Empty : line.Substring(pos + 1);

            }

        }

        public void LoadMatrix(string file, int rownum, int colnum)
        {
            Setup(rownum, colnum);
            FileStream stream = new FileStream(file, FileMode.Open);
            StreamReader reader = new StreamReader(stream);
            try
            {
                string line;
                int row = 0;
                while (!reader.EndOfStream)
                {
                    line = reader.ReadLine();
                    ProcessLine(line, row);
                    row++;
                }
```

```
            }
            finally
            {
                reader.Close();
            }
        }


        public WaterStep[] LastRoute()
        {
            return WaterRoutes[WaterRoutes.Count() - 1].Steps;
        }

        public string RowRoute()
        {

            StringBuilder rowRoute = new StringBuilder();
            WaterStep[] step;


            step = LeastResitant().Steps;


            for (int i = 0; i < step.Count(); i++)
            {
                rowRoute.Append((step[i].row+1).ToString() + ",");
            }

            return rowRoute.ToString().TrimEnd(',');

        }

        public string Weight()
        {

            return LeastResitant().TotalWeight.ToString();

        }

        private int PrevRow(int row)
        {
            return row == 0 ? maxRow : row - 1;
        }

        private int NextRow(int row)
        {
            return row == maxRow ? 0 : row + 1;
        }




        // as we are requiredd to find only one route which succeeds
        // we can stop when we find it and the last route in WaterROutes is teh
        // successful route.

        public bool ProcessRoutes()
        {
            FoundRoute = false;
            for (int i = 0; i <= maxRow; i++)
            {
                RouteThruMatrix(i);
            }
            return FoundRoute;
        }

        public void RouteThruMatrix(int row)
```

```
        {
            // try the number of maximum routes
            CulWeight = Matrix[row, 0]; // weight in starting cell
            curRow = row;

            for (int j = 0; j <= maxRowPerm; j++)
            {
                WaterSteps = new WaterStep[maxCol+1];
                WaterSteps[0] = new WaterStep();
                WaterSteps[0].row = curRow;
                WaterSteps[0].col = 0;
                curCol = 0;
                CulWeight = Matrix[row,0];
                for (int i = 1; i <= maxCol; i++)
                {
                    SampleCount++;

                    Move();

                }
                if (CulWeight <= WeightLimit)
                {
                    FoundRoute = true;
                }
                WaterRoutes.Add(new WaterRoute(WaterSteps,CulWeight));
            }

        }

        public void Move()
        {
            int _wt = WeightLimit;
            int prow = PrevRow(curRow); // done for efficiency
            int nrow = NextRow(curRow);

            WaterStep step = new WaterStep();
            step.col = curCol + 1;

            if ((Matrix[prow, curCol + 1] < _wt) && ValidStep(WaterSteps,prow,curCol+1))
            {
                _wt = Matrix[prow, curCol + 1];
                step.row = prow;
            };
            if ((Matrix[curRow, curCol + 1] < _wt) && ValidStep(WaterSteps, curRow, curCol↙
    + 1))
            {
                _wt = Matrix[curRow, curCol + 1];
                step.row = curRow;
            };
            if ((Matrix[nrow, curCol + 1] < _wt) && ValidStep(WaterSteps,nrow,curCol+1))
            {
                _wt = Matrix[nrow, curCol + 1];
                step.row = nrow;
            };

            CulWeight = CulWeight + _wt;
            WaterSteps[curCol + 1] = step;
            curCol = step.col;
            curRow = step.row;

        }

        private bool CheckNextStep(WaterStep[] steps, List<WaterRoute> routes,int nextcol,↙
    int currow, int curcol)
        {
            List<WaterRoute> _route = new List<WaterRoute>();
            if (nextcol < curcol)
```

```
            {
                foreach (var route in routes)
                {
                   if (route.Steps[nextcol].row == steps[nextcol].row)
                    {
                        _route.Add(route);
                    }

                }
               return CheckNextStep(steps,_route, nextcol + 1, currow, curcol);
            }else
            {
                foreach (var route in routes)
                {
                    if (route.Steps[curcol].row == currow)
                    {
                        _route.Add(route);
                    }
                }
                return (routes.Count == 0);

            }
        }

        // checks to see if this step has been taken in a previous route
        private bool ValidStep(WaterStep[] steps,int currow, int curcol)
        {
            // booundary condition - starting anther tries are ordered so would not come ↙
    here
            List<WaterRoute> routes = new List<WaterRoute>();
            foreach (var route in WaterRoutes)
            {
                if (steps[0].row == route.Steps[0].row)
                {
                    routes.Add(route);
                }
                if (routes.Count() > 0)
                {
                    return CheckNextStep(steps, routes,1, currow, curcol);
                }

            }
            return true;

        }

    }

    public class WaterRoute
    {
        public WaterStep[] Steps;
        public int TotalWeight;

        public WaterRoute(WaterStep[] steps, int weight)
        {
            Steps = steps;
            TotalWeight = weight;

        }

    }


    public class WaterStep
    {
        int _row;
        int _col;
```

```
        public WaterStep()
        {
        }

        public int row { get; set; }
        public int col { get; set; }

    }
}
```