Opdracht 'Minimum Spanning Tree'

Bij deze opdracht is het de bedoeling dat je een applicatie schrijft die de 'Minimum Spanning Tree' (MST) bepaalt van een gegeven (verbonden, ongerichte) graaf. Zoals je bij Wiskunde hebt gezien zijn hier 2 algoritmes voor: Prim algoritme en Kruskal algoritme. De definities van deze 2 algoritmes staan hieronder:

[Prim]

1. selecteer een willekeurige knoop;
2. kies de tak vanuit de geselecteerde knopen met het laagste gewicht, zonder een cycle te maken, en plaats de eindknoop van deze tak in de geselecteerde knopen;
3. herhaal stap 2 tot 'n-1' takken (alle knopen) geselecteerd zijn;

[Kruskal]

1. kies de tak/kant met het laagste gewicht;
2. kies een volgende tak met het laagste gewicht, zonder een cycle te maken;
3. herhaal stap 2 totdat 'n-1' takken gekozen zijn;

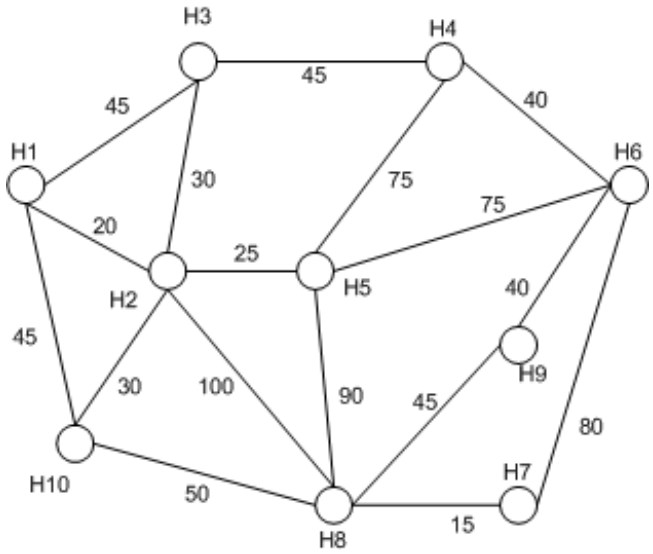
Het is de bedoeling dat je het Prim algoritme implementeert en wel op de 'tabel-manier'. Dit principe staat hieronder:

1. bouw een tabel (`System.Data.DataTable`) op met [aantal knopen] rijen en [aantal knopen] kolommen, met daarin opgenomen de onderlinge afstanden; toon deze tabel zodat je kunt controleren of de tabel correct gevuld is;
2. selecteer een willekeurige knoop en voeg deze toe aan de lijst 'geselecteerde knopen', en verwijder de betreffende rij uit de tabel;
3. selecteer vanuit de lijst 'geselecteerde knopen' (kolommen) de eindknoop met de laagste waarde (bij meerdere 'laagste waarden' kies er één willekeurig uit); voeg aan de lijst 'kanten' de kant (kolomknoop, rijknoop) toe;
4. voeg de eindknoop toe aan de lijst 'geselecteerde knopen' en verwijder de betreffende rij uit de tabel;
5. herhaal vanaf 3 totdat alle knopen in de lijst 'geselecteerde knopen' opgenomen zijn;
6. de lijst 'kanten' is nu de oplossing (print deze uit op het scherm);

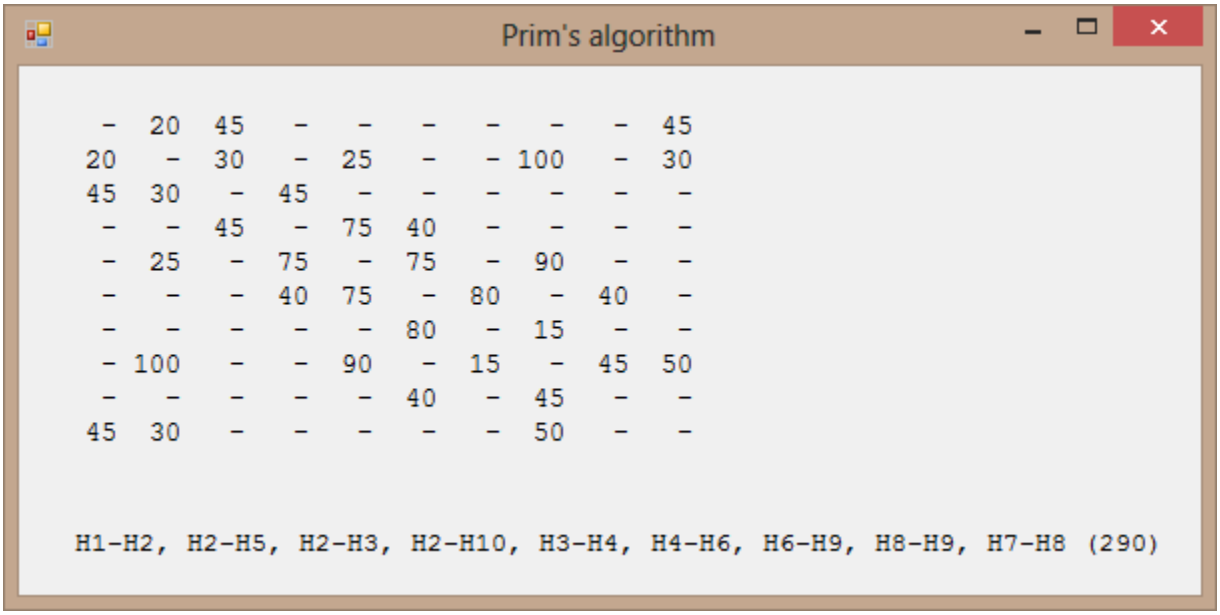
Je kunt gebruik maken van de code op de volgende bladzijden. Je hoeft een graaf niet uit een bestand te lezen; je mag deze hardcoded in je applicatie opnemen. Succes!

```
class Knoop
{
    private string identifier;

    public Knoop(string id)
    {
        identifier = id;
    }

    public string Identifier
    {
        get { return identifier; }
    }
}


class Kant
{
    private Knoop a, b;
    private int lengte;

    public Kant(Knoop a, Knoop b, int lengte)
    {
        this.a = a;
        this.b = b;
        this.lengte = lengte;
    }

    public Knoop KnoopA
    {
        get { return a; }
    }

    public Knoop KnoopB
    {
        get { return b; }
    }

    public int Lengte
    {
        get { return lengte; }
    }
}
```

```
public partial class Form1 : Form
{
    private List<Knoop> knopen = new List<Knoop>();
    private List<Kant> kanten = new List<Kant>();

    public Form1()
    {
        InitializeComponent();
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        // maak knopen aan (hardcoded)
        Knoop h1 = new Knoop("H1");
        Knoop h2 = new Knoop("H2");
        Knoop h3 = new Knoop("H3");
        Knoop h4 = new Knoop("H4");
        Knoop h5 = new Knoop("H5");
        Knoop h6 = new Knoop("H6");
        Knoop h7 = new Knoop("H7");
        Knoop h8 = new Knoop("H8");
        Knoop h9 = new Knoop("H9");
        Knoop h10 = new Knoop("H10");

        knopen.Add(h1);
        knopen.Add(h2);
        knopen.Add(h3);
        knopen.Add(h4);
        knopen.Add(h5);
        knopen.Add(h6);
        knopen.Add(h7);
        knopen.Add(h8);
        knopen.Add(h9);
        knopen.Add(h10);

        // maak kanten aan (hardcoded)
        kanten.Add(new Kant(h1, h2, 20));
        kanten.Add(new Kant(h1, h3, 45));
        kanten.Add(new Kant(h1, h10, 45));
        kanten.Add(new Kant(h2, h3, 30));
        kanten.Add(new Kant(h2, h5, 25));
        kanten.Add(new Kant(h2, h8, 100));
        kanten.Add(new Kant(h2, h10, 30));
        kanten.Add(new Kant(h3, h4, 45));
        kanten.Add(new Kant(h4, h5, 75));
        kanten.Add(new Kant(h4, h6, 40));
        kanten.Add(new Kant(h5, h6, 75));
        kanten.Add(new Kant(h5, h8, 90));
        kanten.Add(new Kant(h6, h7, 80));
        kanten.Add(new Kant(h6, h9, 40));
        kanten.Add(new Kant(h7, h8, 15));
        kanten.Add(new Kant(h8, h9, 45));
        kanten.Add(new Kant(h8, h10, 50));

        // bepaal 'Minimum Spanning Tree' via prim algoritme
        MinimumSpanningTree(knopen, kanten);
    }
```



```
private void MinimumSpanningTree(List<Knoop> knopen, List<Kant> kanten)
{
    // 1) bouw een tabel op met [aantal knopen] rijen en [aantal knopen] kolommen,
    // met daarin opgenomen de onderlinge afstanden
    DataTable table = new DataTable("PrimTable");
    // TODO...

    // 2) selecteer een willekeurige knoop en verwijder de betreffende rij
    // TODO...

    // 5) zolang niet alle knopen geselecteerd zijn, ga door...
    while (...)
    {
        // 3a) selecteer vanuit de geselecteerde knopen (kolommen) de eindknoop met
        // de laagste waarde (bij meerdere 'laagste waarden' kies er één uit)
        // TODO...

        // 3b) voeg de kant [kolomknoop, rijknoop] toe aan de lijst geselecteerde kanten
        // TODO...

        // 4) voeg de eindknoop toe aan de geselecteerde knopen en verwijder de
        // betreffende rij uit de tabel
        // TODO...
    }

    // 6) de lijst met geselecteerde kanten is de oplossing, toon deze lijst
    // TODO...
}
```



Figuur 1: Mogelijke uitvoer v/h programma