MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI

INSPECTORATUL ȘCOLAR JUDEȚEAN BIHOR

LICEUL TEORETIC ONISIFOR GHIBU ORADEA

Nr. Înreg. _____/ _____

PROIECT PENTRU OBȚINEREA CERTIFICATULUI DE COMPETENȚE PROFESIONALE

– SPECIALIZAREA INFORMATICĂ –

PROFESOR COORDONATOR,

Lukács Sándor, Luncan Alina

ABSOLVENT,

Foica Rareș Ioan

ORADEA, 2018

MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI

INSPECTORATUL ȘCOLAR JUDEȚEAN BIHOR

LICEUL TEORETIC ONISIFOR GHIBU ORADEA

Nr. Înreg. _____/ _____

SOFT EDUCAȚIONAL – ARBORE PARȚIAL DE COST MINIM

PROFESOR COORDONATOR,

Lukács Sándor, Luncan Alina

ABSOLVENT,

Foica Rareș Ioan

ORADEA, 2018

CONȚINUTUL LUCRĂRII

I. MOTIVUL ALEGERII TEMEI

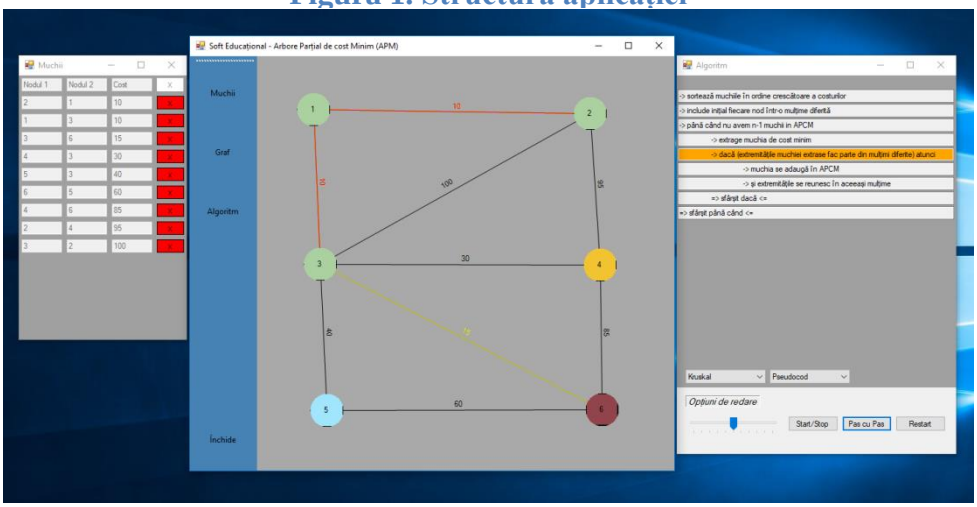
Motivația principală a proiectului constă în identificarea rolului major pe care reprezentarea vizuală o are în procesul de înțelegere și însușire a algoritmilor clasici din teoria grafurilor. Astfel, aflați în posesia unui suport grafic care să permită vizualizarea grafurilor și funcționarea algoritmilor responsabili pentru determinarea arborelui parțial de cost minim (precum algoritmul lui Kruskal sau algoritmul lui Prim) , atât elevii, cât și pasionații de algoritmică își pot însuși cu ușurință bazele acestora.

Pornind de la această premisă și considerând fascinante provocările prezente în dezvoltarea unei astfel de interfețe grafice, se poate explica motivul alegerii „Soft-ului educațional – Arbore parțial de cost minim” ca temă pentru lucrarea de atestat.

II. STRUCTURA GENERALĂ A APLICAȚIEI

„Soft educațional – Arbore parțial de cost minim” este o aplicație Windows Forms, ce permite desenarea propriului graf ponderat neorientat și vizualizarea pas-cu-pas a procesului prin care se determină arborele parțial de cost minim corespunzător grafului, conform unuia din algoritmii Prim sau Kruskal. Aplicația constă în trei form-uri (ferestre): „Soft educațional”, „Muchii” și „Algoritm” (ca în Figura 1).

Figura 1. Structura aplicației



Form-ul „Soft educațional” este form-ul principal al aplicației și include suprafața pentru desenarea grafului cu ajutorul butoanelor mouse-ului, precum și meniul principal al aplicației care permite: accesarea form-ului „Muchii” (prin opțiunea „Muchii), accesarea form-ului „Algoritm” (prin opțiunea „Algoritm”), activarea suprafeței de lucru pentru desenarea grafului (prin opțiunea „Graf > Desenează”), resetarea suprafeței de lucru și, implicit, ștergerea oricărui graf desenat anterior (prin opțiunea „Graf >Resetează”) precum și închiderea aplicației (prin opțiunea „Închide”).

Form-ul „Muchii” permite vizualizarea tuturor muchiilor din graf sub formă tabelară (tabel cu 3 coloane pentru prima extremitate a muchiei, Nodul1, a doua extremitate a muchiei, Nodul 2, și costul muchiei, Cost), precum și ștergerea individuală a fiecărei muchii în parte (prin apăsarea butonului roșu din dreptul muchiei).

Form-ul „Algoritm” permite selectarea algoritmului dorit (Prim sau Kruskal) și vizualizarea acestuia în pseudocod. Tot acest form prezintă și o serie de opțiuni de redare (precum „Start/Stop”, „Pas cu pas” și „Restart”) pentru „rularea” algoritmului selectat pe graful desenat anterior în suprafața de lucru.

III. DESCRIEREA ALGORITMILOR, STRUCTURI DE DATE FOLOSITE

Programul folosește o multitudine de algoritmi și structuri de date, însă excluzându-le pe cele responsabile de administrarea form-urilor și de aspectele lor vizuale, cele mai semnificative elemente sunt: structurile de date utilizate pentru noduri și muchii, precum și algoritmii în care acestea sunt folosite.

Nodurile grafului desenat sunt reținute într-un vector sub formă de butoane rotunde (RoundButton), iar muchiile sunt reținute în memorie într-o listă alocată dinamic de tip structură cu trei câmpuri: prima extremitate a muchiei (firstNode), a doua extremitate a muchiei (secondNode) și costul/ponderea asociată muchiei (weight), ca în Figura 2.

Figura 2. Structurile de date pentru graf

```
//The nodes
public static RoundButton[] nodes = new RoundButton[200];
public static int nrNodes = 0;

//The edges
public struct edge {
    public RoundButton firstNode;
    public RoundButton secondNode;
    public int weight;

    public edge(RoundButton node1, RoundButton node2, int newWeight) : this()
    {
        this.firstNode = node1;
        this.secondNode = node2;
        this.weight = newWeight;
    }
}
public static List<edge> edges = new List<edge> { };
```

În ceea ce privește simularea algoritmilor Prim și Kruskal pe graful desenat, din moment ce timpii eficienți de execuție nu au reprezentat o necesitate, s-au folosit metode „naive” de inserare, sortare (sortarea prin selecție) și extragere a valorii minime, rezultând în complexități finale de O(n), $O(n^2)$ și, respectiv, O(n). Întreaga simulare este gestionată în form-ul „Algoritm” de două Timer-e (unul pentru fiecare algoritm) cu interval ajustabil prin TrackBar-ul opțiunilor de redare. Figurile 3 și 4 reprezintă exact un „Tick”(o iterație) al Timer-elor pentru algoritmul Kruskal și, respectiv, algoritmul Prim.

Figura 3. Algoritmul lui Kruskal

```
private void timerKruskalPseudo_Tick(object sender, EventArgs e) {
    step++; //increase the step

    if (step <= nrLines) //if code has not finished, proceed and run the algorithm
    {
        if (step >= 1) //remove highlight from current line of code
            lineCode[step].BackColor = this.BackColor;

             //and execute the current line of code
        if (step == 1)
        {
            //sorteza muchiile in ordine crescatoare a costurilor
            for (int i = 0; i < Form1.edges.Count() - 1; i++) {
                for (int j = i + 1; j < Form1.edges.Count(); j++)
                    if (Form1.edges[i].weight > Form1.edges[j].weight) {
                        Form1.edge aux = new Form1.edge(Form1.edges[i].firstNode,
Form1.edges[i].secondNode, Form1.edges[i].weight);
                        Form1.edges[i] = Form1.edges[j];
                        Form1.edges[j] = aux;
                    }
            }
            //si actualizeaza Form Edges
            Form1.frmEdges.FormEdges_VisibleChanged(this, e);
        }
        else if (step == 2)
        {
            //include fiecare nod intr-o multime diferita
            for (int i = 1; i <= Form1.nrNodes; i++) {
                M[Form1.nodes[i]] = i;
                //da nodului o culoare random
                Color randomColor = Color.FromArgb(50+rand.Next(206), 50+rand.Next(206),
50+rand.Next(206));
                Form1.nodes[i].BackColor = randomColor;
            }
        }
        else if (step == 3)
        {
            //daca avem deja n-1 muchii in APM, ne oprim
            if (inAPM.Count() == Form1.nrNodes - 1)
                step = 8;
            //altfel, intram in 'pana cand'
        }
        else if (step == 4)
        {
            //extrage muchia de cost minim (exact urmatoarea din lista)
            currEdge++;
            Form1.DrawEdge(Form1.edges[currEdge], Color.Yellow);
        }
        else if (step == 5)
        {
       //daca extremitatile muchiei extrase apartin aceleasi multimi, trecem peste 'daca'
            if (M[Form1.edges[currEdge].firstNode] ==
M[Form1.edges[currEdge].secondNode]) {
                Form1.DrawEdge(Form1.edges[currEdge], Color.Black);
                step = 7;
            }

       //altfel, intram in 'daca'
        }
        else if (step == 6)
        {
            //adauga muchia extrasa in APM
            Form1.DrawEdge(Form1.edges[currEdge], Color.Red);
```

```
            inAPM.Add(Form1.edges[currEdge]);
        }
        else if (step == 7)
        {
        //multimea A (a unei extremitati) si multimea B (a celeilalte extremitati) cu A<B
            int A, B;
            Color myColor = new Color(); //color to change the nodes of one set to
            if (M[Form1.edges[currEdge].firstNode] < M[Form1.edges[currEdge].secondNode])
            {
                A = M[Form1.edges[currEdge].firstNode];
                B = M[Form1.edges[currEdge].secondNode];
                myColor = Form1.edges[currEdge].firstNode.BackColor;
            } else {
                A = M[Form1.edges[currEdge].secondNode];
                B = M[Form1.edges[currEdge].firstNode];
                myColor = Form1.edges[currEdge].secondNode.BackColor;
            }
           //reuneste multimile celor doua extremitati (schimba multimea B cu multimea A)
            for (int i = 1; i <= Form1.nrNodes; i++) {
                if (M[Form1.nodes[i]] == B) {
                    M[Form1.nodes[i]] = A;
                    Form1.nodes[i].BackColor = myColor;
                }
            }
        }
        else if (step == 8)
        {
            //revino la 'pana cand'
            step = 2;
        }

        if (step < nrLines) //highlight next line of code, if exists
            lineCode[step + 1].BackColor = Color.Orange;
    }
       else if (step > nrLines) //else, if the code has finished, stop the timer and
reset the algorithm
       {
        timerKruskalPseudo.Enabled = false;

        //disable playback buttons
        startStop.Enabled = false;
        stepByStep.Enabled = false;
    }
}
```

Figura 4. Algoritmul lui Prim

```
private void timerPrimPseudo_Tick(object sender, EventArgs e){
       step++; //increase the step

       if (step <= nrLines) //if code has not finished, proceed and run the algorithm
       {
              if (step >= 1) //remove highlight from current line of code
                     lineCode[step].BackColor = this.BackColor;

              //and execute the current line of code
              if (step == 1) {
                     //extrage nodul de start
                     nodStart = Int32.Parse(textBox2.Text);
            Vmin = nodStart;
            //marcheaza nodul de start ca vizitat
            viz[nodStart] = true;
            addedNodes++;
            //highlight the node
```

```
            Form1.nodes[nodStart].BackColor = Color.Orange;

              } else if (step == 2) {
            //daca am vizitat toate nodurile, termina algoritmul
            if (addedNodes == Form1.nrNodes)
                step = 6;
            //altfel, intra in structura repetitiva

              } else if (step == 3) {
            //se consideră nodul nevizitat (NVmin) aflat la distanță minimă față de un
nod deja vizitat (Vmin)
            int min = int.MaxValue;

            for(int i=0; i<Form1.edges.Count(); i++) {
                int n1 = Int32.Parse(Form1.edges[i].firstNode.Text);
                int n2 = Int32.Parse(Form1.edges[i].secondNode.Text);
                if (viz[n1] == true && viz[n2] == false)
                {
                    if (Form1.edges[i].weight < min)
                    {
                        min = Form1.edges[i].weight;
                        mch = Form1.edges[i];
                        NVmin = n2;
                    }
                }
                else if (viz[n1] == false && viz[n2] == true)
                {
                    if(Form1.edges[i].weight < min)
                    {
                        min = Form1.edges[i].weight;
                        mch = Form1.edges[i];
                        NVmin = n1;
                    }
                }
            }
            //highlight edge and NVmin
            Form1.DrawEdge(mch, Color.Yellow);
            Form1.nodes[NVmin].BackColor = Color.Yellow;

        } else if (step == 4) {
            //muchia NVmin-Vmin se adaugă la arbore
            Form1.DrawEdge(mch, Color.Red);

        } else if (step == 5) {
            //NVmin se consideră vizitat
            addedNodes++;
            viz[NVmin] = true;
            Form1.nodes[NVmin].BackColor = Color.Orange;

            //revino la inceputul structurii repetitive
            step = 1;
        }

              if (step < nrLines) //highlight next line of code, if exists
                    lineCode[step + 1].BackColor = Color.Orange;
      }
      else if (step > nrLines) //else, if the code has finished, stop the timer
      {
       timerPrimPseudo.Enabled = false;
            startStop.Enabled = false;
            stepByStep.Enabled = false;
      }
}
```

IV. DETALII TEHNICE DE IMPLEMENTARE

Programul constă într-o aplicație de tipul Windows Forms, realizată în limbajul de programare C#, folosind framework-ul Microsoft .NET și biblioteca WinForms în mediul de dezvoltare Visual Studio 2015, versiunea Community. Concret, pentru crearea suprafeței grafice de lucru au fost folosite funcționalitățile grafice GDI+ de bază disponibile în biblioteca System.Drawing.

V. CERINȚE SOFT ȘI HARD

Din punct de vedere al software-ului, programul ar trebui să ruleze pe orice versiune a sistemului de operare windows mai nouă ca Windows XP, atâta timp cât este instalată ultima versiune a framework-ului Microsoft .NET.

Din punct de vedere al hardware-ului, din moment ce aplicația nu necesită nici prea multă memorie (niciodată mai mult de 30MB RAM), nici prea multă putere de procesare, cerințele minime de sistem coincid cu cerințele minime ale sistemului de operare, cu câteva precizări. Pentru referință, cerințele minime de sistem ale Windows 7 sunt:

- Procesor de 1 gigahertz (GHz) sau mai rapid, pe 32 de biți (x86) sau pe 64 de biți (x64)
- 1 gigabait (GB) de RAM (pentru 32 de biți) sau 2 GB de RAM (pentru 64 de biți)
- 16 GB de spațiu disponibil pe hard disk (pentru 32 de biți) sau 20 GB (pentru 64 de biți) + 2.5 MB disponibili pe hard disk pentru stocarea programul în sine
- Dispozitiv cu placă video DirectX 9 cu driver WDDM 1.0 sau mai recent

VI. POSIBILITĂȚI DE DEZVOLTARE

În momentul de față, programul permite numai operarea cu grafuri ponderate neorientate și modificarea muchiilor acestora. Dacă utilizatorul închide aplicația sau resetează suprafața de lucru, se va pierde orice graf desenat anterior. O primă posibilitate de dezvoltare a programului ar fi introducerea opțiunii de a lucra cu grafuri orientate și implementarea unei metode de a salva grafurile desenate sub o anumită formă (spre exemplu, sub formă de fișier text), pentru a fi păstrate și după închiderea aplicației sau după resetarea suprafeței de lucru.

O altă posibilitate de dezvoltare ar fi traducerea din pseudocod a algoritmilor Prim și Kruskal pentru determinarea arborelui parțial de cost minim și în alte limbaje de programare, precum C, C++, Java, JavaScript, Python, C#, Visual Basic etc.

VII. CONCLUZII

Dezvoltarea unei aplicații precum „Soft educațional – Arbore parțial de cost minim” prezintă o multitudine de foloase, atât pentru partea care dezvoltă programul, cât și pentru partea care beneficiază de acesta.

Pe de-o parte, dezvoltator(ul)/(ii) programului are ocazia de a-și completa, solidifica și exersa cunoștințele de programare într-o manieră practică, palpabilă și propice învățării, lovindu-se de diverse provocări ridicate la implementare și încercând să le găsească o rezolvare.

Pe de altă parte, utilizatorilor/beneficiarilor programului li se pune la dispoziție un material interactiv ce înglobează și demonstrează cunoștințe de bază în algoritmică cu scopul facilitării procesului de învățare și însușire a metodelor prezentate.