Készítette: Lehoczki Gergő Péter ZH5ED7

A feladat felvezetése

Amikor elmegyünk egy céghez dolgozni, akkor szembesülünk azzal, hogy hogyan is néz ki egy vállalat hierarchiája a dolgozók szempontjából.
A felsővezértől kezdve, egészen a fizikai munkásokig láthatjuk, hogy különféle területeken, különféle vezetők és dolgozók vannak.

Ezen vállalatok többsége mára már modern, kidolgozott rendszereken keresztül tudja nyomon követni a dolgozókat és az ő vagy akár saját feladataikat, viszont a probléma alapvetően az, hogy a jelenlegi vállalat még nem rendelkezik ezzel a megoldással, hanem a régi, hagyományos .ppt ábrákkal valósítják meg. Bár maga a ppt is egyfajta fa struktúra ként képes ábrázolni jelen problémát, viszont gondoljunk bele abba, hogy mennyivel egyszerűbb, ha egy program segítségével tudunk felvenni új dolgozót, le tudjuk írni az ő munkakörét, később pedig hozzá feladatot tudunk felvenni, vagy ellenkező esetben eltávolítani dolgozót, és az ő eddigi feladatait odaadni egy másik dolgozónak.

Ezen rendszer kiépítése külső feladat számomra, amelyet megkaptam az én főnökömtől, aki elvárja ennek a rendszernek a stabil és skálázható működését.

A munkafolyamat

Miután át tárgyaltuk a problémát és az arra szolgáló megoldást, felmerülhet bennünk a kérdés, hogy milyen rekurzív illetve iterációs elemeket tartalmazhat, cső vagy veremszerű adatszerkezet folyamatait hogyan valósíthatjuk meg, illetve a fa struktúrát hogyan tudjuk ábrázolni rajta.

Ha a rekurzió vagy iteráció oldaláról közelítjük meg a feladatot, akkor gondoljunk bele abba, hogy van egy programunk, amely képes arra, hogy végig iterál a dolgozók feladatai során és képes eldönteni egy-egy feladatról, hogy az határidő közeli vagy sem, illetve, hogy ha kész állapotban van a feladat, akkor képes arra, hogy eltávolítsa a dolgozó feladatai közül

Ha a fa struktúrát a dolgozók hierarchiáján nézzük, akkor létezik egy algoritmus, amely képes arra, hogy új dolgozót hozzá adjon munkaköri leírással, illetve képes dolgozót törölni a listából, ha az a dolgozó felmondott.

Könnyen láthatjuk azt, hogy ezen esetben, akár a dolgozókat, akár feladataikat nézzük, megvalósul kettő vezérlési szerkezet, a szelekció és az iteráció. Ezen összetevők fognak alkotni egy algoritmust. Dolgozók esetében az eldöntés tételét, feladataik esetében pedig a listába való beszúrást, vagy a listából való törlést.

Most már nem maradt más hátra, mint hogy megtárgyaljuk azt, hogy hogyan is tudjuk alkalmazni a fa struktúrát jelen problémán.
Ahhoz, hogy ehhez hozzá kezdjünk, meg kell vizsgálnunk a fa struktúra 3 legfőbb elemét.

- Gyökér
- Csomópont
- Levél

A gyökér az, kinek nincs őse, viszont 1 vagy több utódja van.
Csomópontnak hívjuk azt, akinek van legalább 1 őse és legalább 1 utódja.
Levélnek hívjuk azt, aminek már nincs több utódja, viszont van legalább 1 őse.

Miután megvizsgáltuk a fa struktúrát, könnyen láthatjuk azt, hogy a gyökér elemet a cégen belül a vezérigazgató alkotja, akinek a cégen belül már nincsen főnöke, viszont alá tartoznak igazgatók.
Ebből következik tehát, hogy az igazgatók lehetnek csomópontok, hiszen legalább 1 őse van és legalább 1 utódja biztosan lesz.
Ha a levél adatot szeretnénk vizsgálni, akkor egészen mélyre, pontosan a hierarchia legaljára kell mennünk, ahol a fizikai dolgozók találhatóak, (persze egy másik ágán a fának elmehetünk másik területre) akik már levélként viselkednek, hiszen legalább 1 ős tartozik hozzájuk, azonban nem tartozik hozzájuk egyetlen utód sem.Ugyanezt természetesen ábrázolhatjuk a dolgozók feladatai szempontjából is.

Összetevők

Most, hogy megtárgyaltuk azt, hogy milyen iteratív algoritmusokat, illetve milyen adatszerkezetet fog tartalmazni a probléma megoldására szolgáló programunk, elkezdhetjük a táblázatos, folyamatábrás illetve a fa struktúrás ábrázolás vázolását.

Ahhoz, hogy ebbe bele kezdjünk, ismételjük meg azokat az elemeket amelyeket táblázatban szeretnénk ábrázolni.Ilyenek lesznek a **beosztások,** amely tartalmazni fogja a beosztás azonosítóját, a beosztás megnevezését, a beosztás felé tartozó beosztást(okat) ,illetve az alá tartozó beosztás(okat) is, és végül az öröklődés sorrendjét.

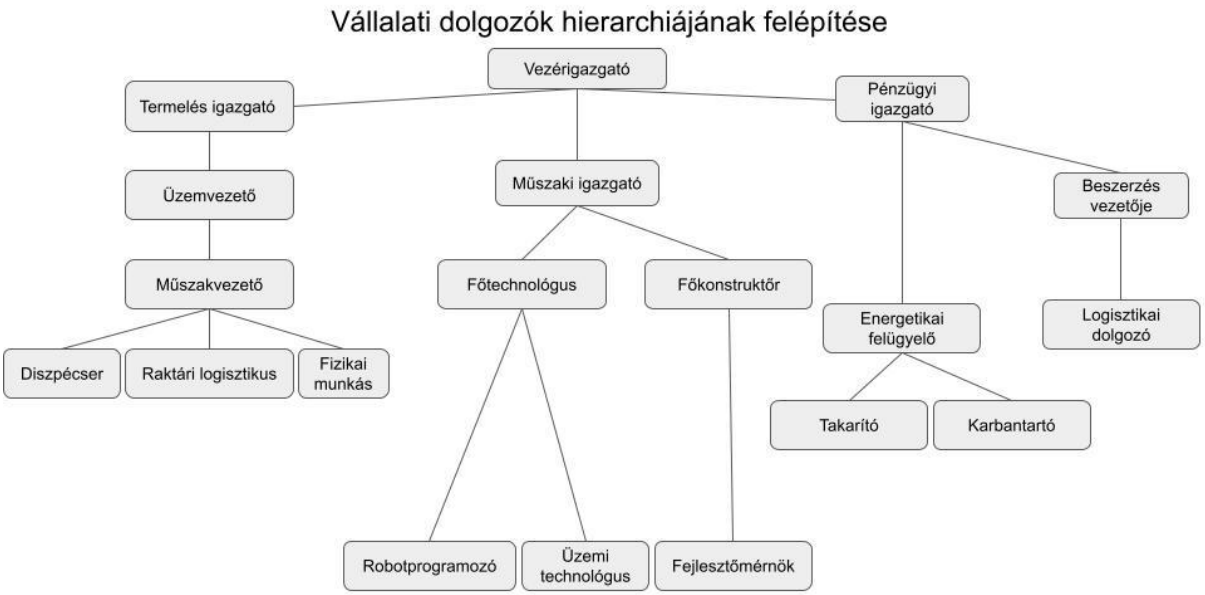
Ezt az *1.ábra* jól szemlélteti tábla formájában, hiszen leltárba vettük mind azt amire szükségünk lesz ezen adattábla elkészítéséhez.Vegyük tehát szemügyre az 1.ábrát.

ID	Beosztás	Származás	Származási Sorren
18	beszerzés vezetője	14	2.
5	diszpécser	4	1.
15	energetikai felügyelő	14	1.
13	fejlesztőmérnök	10	1.
7	fizikai munkás	4	3.
10	főkonstruktőr	8	2.
9	főtechnológus	8	1.
17	karbantartó	15	2.
19	logisztikai dolgozó	18	1.
8	műszaki igazgató	1	2.
4	műszakvezető	3	1.
14	pénzügyi igazgató	1	3.
6	raktári logisztikus	4	2.
11	robotprogramozó	9	1.
16	takarító	15	1.
2	termelés igazgató	1	1.
1	vezérigazgató	0	1.
12	üzemi technológus	9	2.
3	üzemvezető	2	1.

1.ábra

Láthatjuk azt, hogy minden beosztáshoz tartozik egy egyedi azonosító, amely alapján megmondható az, hogy ő csomópont, levél vagy gyökér, ha szemügyre vesszük a származás oszlopot, ahol láthatjuk azt, hogy mely beosztás felett vannak még beosztások, illetve alatta is szerepelnek még beosztások, vagy esetleg nincs vagy őse vagy utódja.

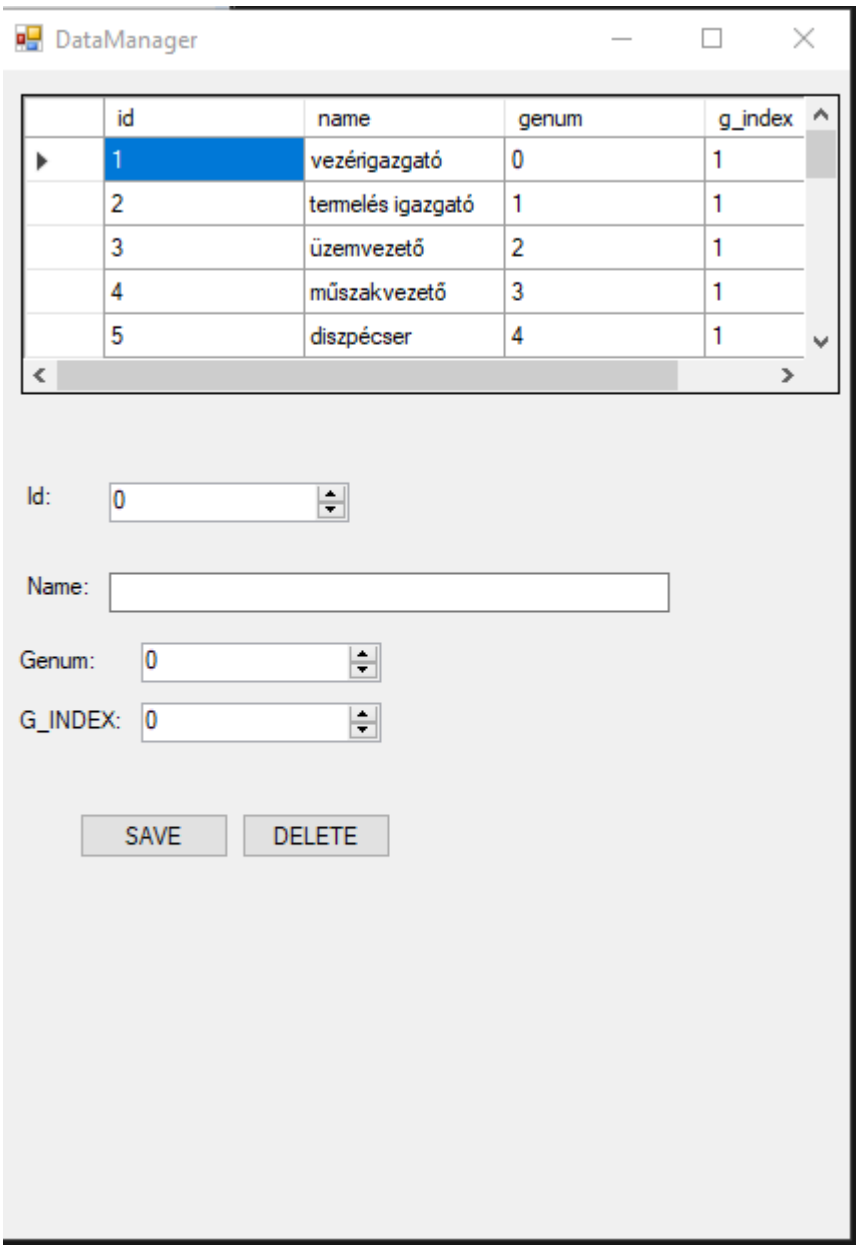
Következő lépésben nézzük meg azt, hogy milyen kiinduló fa struktúrás ábrázolásból voltunk képesek leltárba venni a beosztásokat, ezt szemlélteti a 2.ábra.



2.ábra

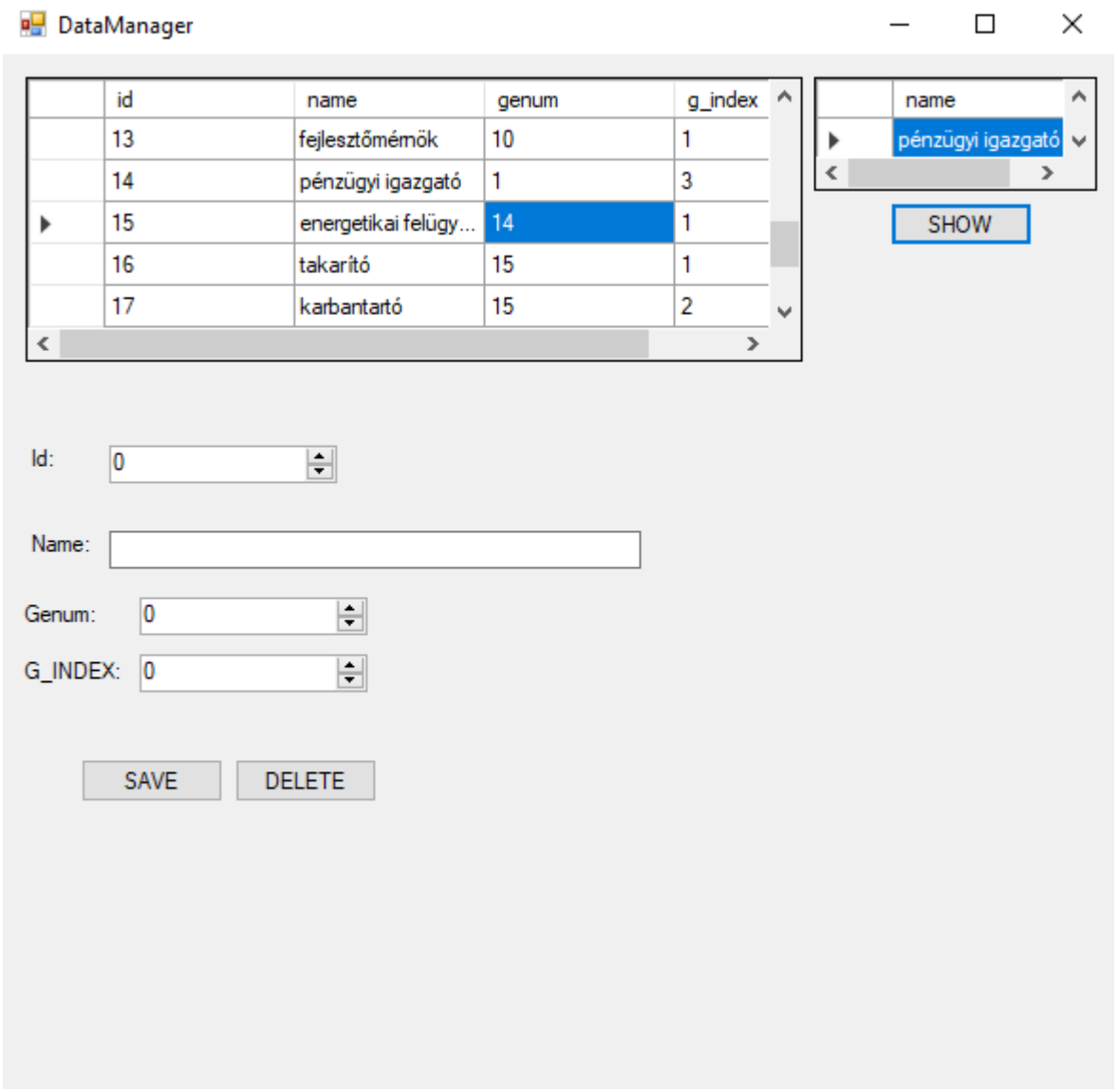
Könnyen láthatjuk azt, hogy a két ábra között lényeges információ beli különbségek vannak.Természetesen ez nem azt jelenti, hogy a 2.ábra szemügyre vétele után, nem tudjuk megmondani azt, hogy például a robotprogramozó egy levél elem, és a származási útvonala a főtechnológus->műszaki igazgató->vezérigazgató. Mindazonáltal lássuk be, hogy az 1.ábrával összehasonlítva, sokkal egyszerűbb dolgunk van, főleg ha ezt egy algoritmus segítségével tesszük meg. Gondoljunk csak bele abba, hogy milyen kényelmes lenne, ha egy program segítségével, és az eldöntés algoritmusát használva, ki tudnánk listázni egy adott pozíció származási útvonalát. Pontosan erre fognak szolgálni az azonosítók, melyek a beosztásokat azonosítják.

Erre fog szolgálni nekünk egy C#-nyelven megírt program, amely képes MySQL adatbázishoz kapcsolódni, és onnan lekérdezni adatokat. A 3.ábrán láthatjuk a programunkat, amely a datamanager névre hallgat, mely képes adatokat felvinni egy táblába, illetve képes onnan törölni adatokat.Ezzel megvalósítjuk a listába való beszúrást, és a listából való eltávolítást is.



3.ábra

Na de térjünk vissza az eredeti problémára, amelyet tárgyalni szeretnénk.Hogyan tudjuk elérni azt, hogy kiválasztva egy bizonyos beosztást, képesek legyünk annak teljes származási útvonalát megjeleníteni. Erre szolgál nekünk a kiválogatás tétele. Egy algoritmus segítségével végig megyünk az adatainkon, és amennyiben a keresett feltétellel egyezőt találunk, azt megjelenítjük a programunkban. A 4.ábra szemlélteti a továbbfejlesztett programunkat, amely egy SQL parancs segítségével képes arra, hogy megjelenítse egy kiválasztott elem ősét.



4.ábra

Ebben a példában azt láthatjuk, hogy kiválasztottuk az energetikai felügyelőt, akinek az őse a pénzügyi igazgató. Aztán kiválaszthatjuk a pénzügyi igazgatót, akinek az őse pediglen a vezérigazgató lesz és így tovább. Ezáltal vissza is utalhatunk arra, hogy mennyivel jobb egy egér kattintással lekérdezni a származását egy elemnek, mintsem végig keresni egy egyszerű ppt állományban.

Továbbá könnyen beláthatjuk, hogy tulajdonképpen egy kiválogatás algoritmust milyen egyszerű módon sikerült megvalósítani.Ezt természetesen egy folyamatábra segítségével, egy háttérbeli működést megvalósítva tudjuk szemléltetni.

Részproblémák

Ha tovább szeretnénk gondolni a programunkat, és ki szeretnénk térni egyéb rész problémákra, olyan téren, hogy mindenkihez tartozik egy feladatlista, amelyben tároljuk azt, hogy mi a feladat leírása, a határideje, és mi a feladat jelenlegi állapota, akkor szükségünk lesz még egy táblára, mely össze lesz kötve a már meglévő hierarchia táblával.

Ismételten szeretnék visszacsatolni arra, hogy egyszerűen, egy beosztásra kattintva, ki tudom listázni annak a dolgozónak a nevét, és a hozzá kapcsolódó feladatokat. Ezt szintén meg oldhatom egy SQL lekérdezéssel a C# kódon belül, de ez a téma nem erre a rész problémára koncentrál jelen pillanatban.

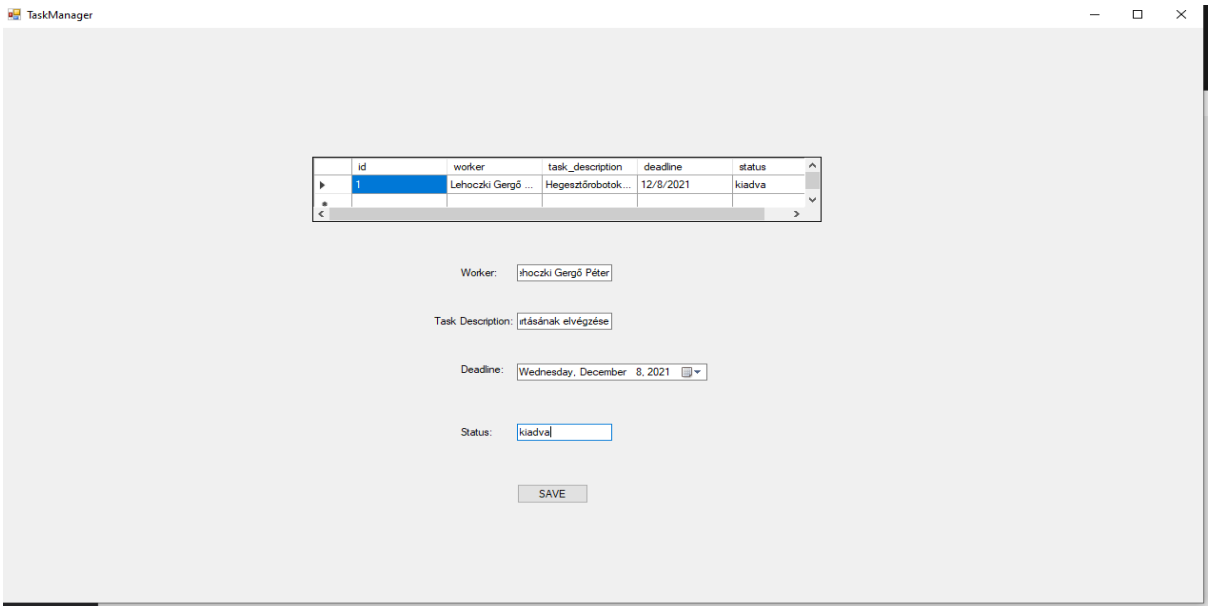
Legelőször viszont, szükségünk lesz arra, hogy készítsünk egy adattáblát, amely tartalmazni fogja a szükséges elemeket. Ezt szintén megtehetjük először egy leltárba vétellel, aztán egy táblázatos forma kialakításával, végezetül pedig egy SQL paranccsal létrehozhatjuk ezt a táblát. Az 5.ábra ezen leltárba vételt szolgál mutatni.

ID	Dolgozó	Feladat Leírása	Határidő	Állapot
1	Lehoczki Gergő Péter	Hegesztőrobot éves karbantartásának elvégzése	2021.12.10	kiadva

5.ábra

Láthatjuk, hogy bekerült egy dolgozó, kihez tartozik egy feladat és ahhoz leírás, határidő illetve egy feladat állapota mező.

Mi lenne ha azt mondanánk, hogy ezeket a rekordokat el tudnánk tárolni a programunkban, és azt mondhatnánk, hogy ott különböző műveleteket tudunk elvégezni, különféle algoritmusok segítségével.Ehhez először is nézzük meg azt a programot, amely képes felvenni új feladatot dolgozói név megadásával. Ezt a 6.ábra szolgáltat bemutatni. Láthatjuk, hogy milyen könnyen, az adattáblába felvettünk egy új rekordot, szemben azzal a hagyományos megoldással, hogy ppt vagy akár excel állományban kellett volna mókolni.



6.ábra

Ahhoz, hogy kezelni tudjuk az adatainkat, létre kell hozzunk egy az adattáblának megfelelő rekordot, melyben pontosan ugyanolyan típusú mezőket tárolunk el, amely a táblánkban is szerepel.Nézzük tehát a 7.ábra által szemléltetett forráskód részletet. (Jelenleg tekintsünk el attól, hogy minden publikus).

```
public class Tasks
{



    public int id;
    public string worker;
    public string taskDescription;
    public DateTime deadline;
    public string status;
    1 reference
    public Tasks(int id, string worker, string taskDescription, DateTime deadline, string status)
    {
        this.id = id;
        this.worker = worker;
        this.taskDescription = taskDescription;
        this.deadline = deadline;
        this.status = status;
    }



}
```

7.ábra

Láthatjuk, hogy létrehozzuk a Tasks nevű rekordot, mely képes tárolni minden mezőt, amelyek léteznek az adattáblában is. Ezt követően létrehozunk egy publikus konstruktort amely paraméterben megkapja a formális paramétereit, és ezeket egyenlővé teszi a fent létrehozott mezőkkel.Ezt követően meg kell vizsgálnunk ezen értékek további életútját. Ehhez szükségünk lesz két metódusra, abból az egyik a

fájlba való kiírás, a másik pedig onnan való visszaolvasás. A fájlba való kiírást a 8.ábra szolgáltat szemléltetni.Láthatjuk azt, hogy egy “tasks.csv” fájlba addig írunk ki, a táblázatos nézetünk adataiból, amíg azok el nem fogynak. Mindazonáltal, figyelembe kell vennünk azt, hogy egy fajta adatot nem tudnánk eltárolni, ezért volt szükségünk létrehozni a Tasks nevű rekordot, amely képes arra, hogy saját típust építsünk fel, és képes ezáltal arra, hogy egy listában el tudjuk ezeket az adatokat később tárolni egy be olvasást követően.Vegyük szemügyre a 8.ábra segítségével, hogy pontosan mi is történik. Le kérjük az adattábla megjelenítésére szolgáló nézetből az adatokat, amelyeket két ciklus segítségével egy csv fájlba kiíratunk, mely később arra szolgál, hogy onnan ezeket az adatokat vissza tudjuk majd olvasni. Könnyen láthatjuk azt, hogy erre azért lesz szükség, mert nem szeretnénk az adatbázisban mókolni, hanem szeretnénk egy előzetes vizsgálatot végezni majd a mezőkkel, a műveletnek megfelelően. Ebben a témában jelenleg arra lesz szükségünk, hogy egy feladatról meg kell tudjuk mondani azt, hogy az határidő közeli vagy sem. Jelen esetben határidő közelinek azokat a feladatokat tekinthetjük, amelyek elvégzésére már csak kevesebb mint egy hét áll rendelkezésére a dolgozónak.

Továbbá rész problémának tekinthetjük azt is ebben a témában, hogy miután megoldottuk azt, hogy el tudjuk dönteni egy feladatról a fenti kritériumot, akkor azt valamilyen módon jelezni kell.Illetve, gondoljunk bele abba, hogy sok feladat bent marad egy adatbázisban amit már elvégeztünk akár kettő évvel ezelőtt és még mindig a mi tárhelyünket foglalja feleslegesen.

```
public void StreamWriter()
{

    StreamWriter file = new StreamWriter("tasks.csv");
    string sLine = "";


    for (int r = 0; r <= dataGridView1.Rows.Count - 2; r++)
    {

        for (int c = 0; c <= dataGridView1.Columns.Count - 2; c++)
        {
            sLine = sLine + dataGridView1.Rows[r].Cells[c].Value;
            if (c != dataGridView1.Columns.Count - 1)
            {

                sLine = sLine + ";";
            }
        }

        file.WriteLine(sLine);
        sLine = "";
    }

    file.Close();

}
```

8.ábra

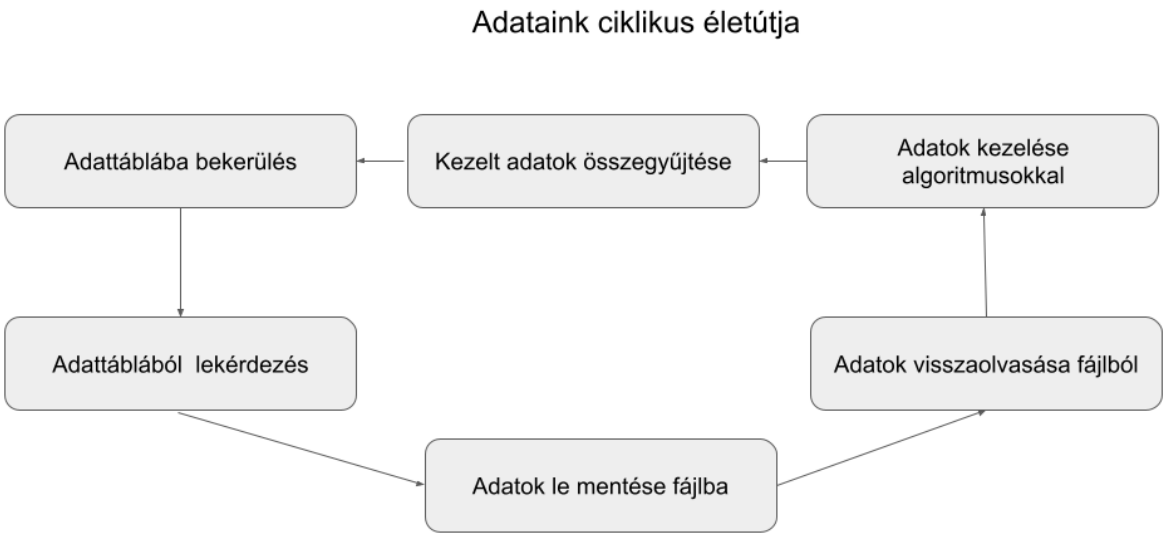
A probléma megoldására tehát egy automatizált algoritmust kell írnunk,amely képes egy gombnyomással törölni az adattáblából azokat a feladatokat amelyek el lettek végezve és már nem is aktuálisak mert mondjuk 2 évvel ezelőtt lettek kiadva.

Ahhoz viszont, hogy ehhez a feladathoz hozzá kezdjünk egy megelőző lépésre lesz szükségünk -amit már korábban tárgyaltunk- egy fájlból való beolvasásra.Méghozzá abból a csv fájlból fogunk be olvasni, amelybe kimentettük az adatainkat. Ezt szemlélteti a 9.ábra

```
public void StreamReader()
{
    List<Tasks> list = new List<Tasks>();
    StreamReader reader = new StreamReader("tasks.csv");
    while (!reader.EndOfStream)
    {


        string read = reader.ReadLine();
        string[] datas = read.Split(';');
        Tasks t = new Tasks(Convert.ToInt32(datas[0]), datas[1], datas[2], Convert.ToDateTime(datas[3]), datas[4]);
        list.Add(t);


    }

    foreach (var item in list)
    {
        Console.WriteLine(item.id + ";" + item.worker + ";" + item.taskDescription + ";" + item.deadline + ";" + item.status);
    }
    reader.Close();
}
```

9.ábra

Láthatjuk, hogy egy Tasks típusú listába képesek vagyunk elmenteni a beolvasott adatokat, ezáltal képesek vagyunk algoritmusokat írni erre az adatszerkezetre.(A konzolra való kiíratást most ne vegyük figyelembe, az csak teszteléshez szükséges).Láthatjuk azt, hogy pontosan az adattáblához megfeleltetjük az adattípusokat, amelyek vissza olvasásra kerültek, és így már teljes az adataink ciklikus életútja, melyet a 10.ábra segítségével ábrázolunk.



10.ábra

Miután szemügyre vettük az ábrát, láthatjuk azt, hogy az adattáblába bekerülést, onnan való lekérdezést, a lekérdezés eredményének a le mentését illetve az adatok

visszaolvasását már megoldottuk, most már elkészíthetjük az automatizált algoritmusokat.

Algoritmusok szervezése

Ahhoz, hogy algoritmust hozzunk létre, két fő dolgot kell ismernünk.

- Algoritmust alkotó elemeket
- Milyen problémára szeretnénk irányítani az algoritmust

Természetesen ezeket is be tudjuk szervezni egy táblázatba, hogy pontosan milyen elemek fogják alkotni az algoritmust és azt később,ennek segítségével, le tudjuk írni egy folyamatábrán.

Tehát először is vegyük leltárba az algoritmust alkotó elemeket.
Ezt a 11.ábra fogja mutatni nekünk.

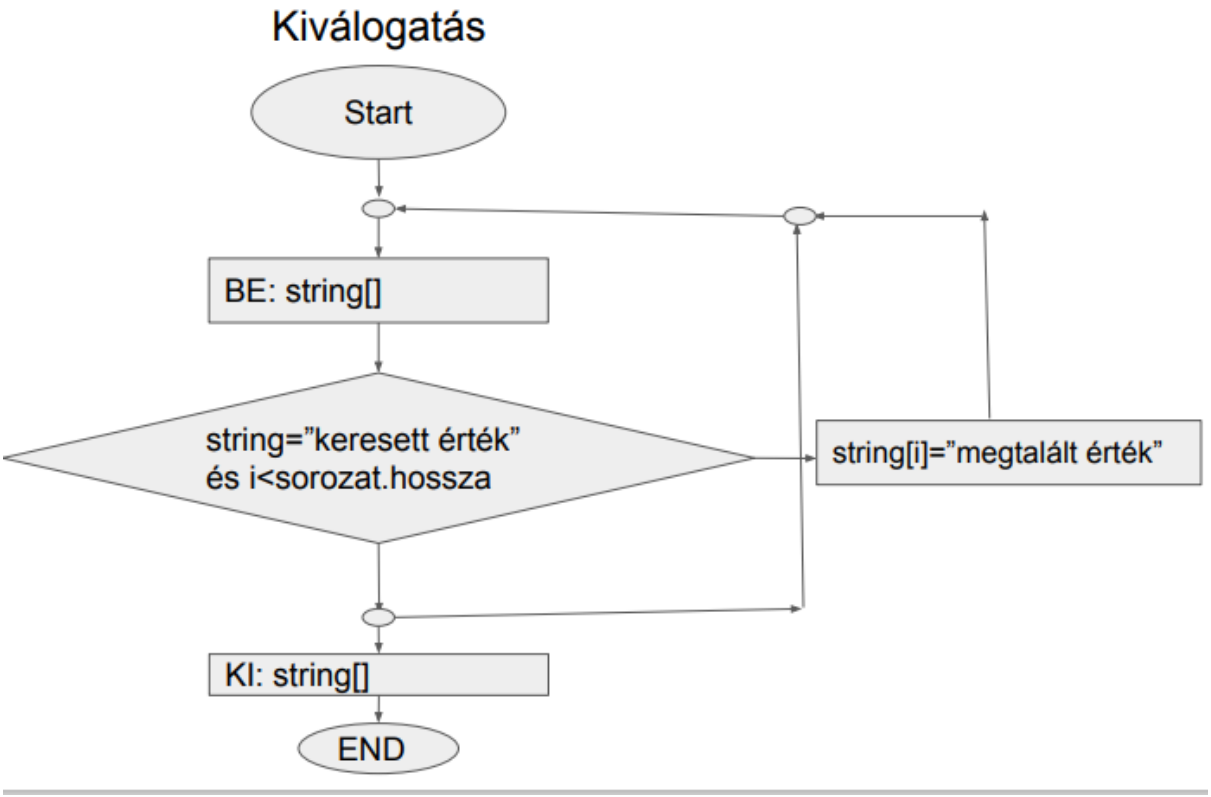
Függvény Neve	Milyen Problémát Old Meg	Milyen Elemek Alkotják
Kiválogatás	Kiválogatja azokat a rekordokat, amelyeknél a feladat határideje 2 évnél régebbi és teljesítve lettek	Iteráció, szelekció,szekvencia, lista
Megszámlálás	Összegezhetjük vele, hogy egy bizonyos dolgozó mennyi feladatot kapott	Iteráció,szelekció,szekvencia,lista

11.ábra

. Miután leltárba lett véve egy algoritmust alkotó elemek összessége, elkészíthetjük a hozzá tartozó folyamatábrát.Ezt a folyamatábrát a 12.ábra fogja szemléltetni nekünk.

Láthatjuk azt, hogy bemenetként kap az algoritmus egy adatszerkezetet, jelen esetben egy tömböt, amelyet fel kell hogy töltsön az általunk meghatározott, keresett elemekkel.Vegyük észre azt, hogy egy iterációt valósítunk meg, amely tartalmaz egy feltételt, és ezen feltétel alapján végig tud menni az adatokon, és ebből ki tudja válogatni azokat a mezőket, amelyek megegyeznek a keresett elemmel.

Jelen esetben a keresett érték egy olyan feladat amely státusza a “befejezett” státuszba került.Ezt bővíthetjük azzal, hogy ez a feladat akár 2 akár 3 évvel ezelőtti volt vagy sem, ez direkt nem került bele a folyamatábrába, hiszen ezt már kedvünk szerint bővíthetjük.Kezdetben tehát deklarálni kell egy tömb vagy lista adatszerkezetet, amelyben eltárolhatjuk majd a kapott elemeket.



12.ábra

C# kódban a következőképpen néz ki a történet (13.ábra).
Láthatjuk tehát, hogy egy listában eltárolunk elemeket, onnan később kibonthatunk elemeket, majd ezeket megfeleltethetjük egy SQL lekérdezés kritériuma ként.

```
List<Tasks> selectedItems = new List<Tasks>();
1 reference
public void Selection()
{

    for (int i = 0; i < list.Count; i++)
    {
        if (list[i].status=="befejezve")
        {
            selectedItems.Add(list[i]);
        }

    }


}
```

13.ábra

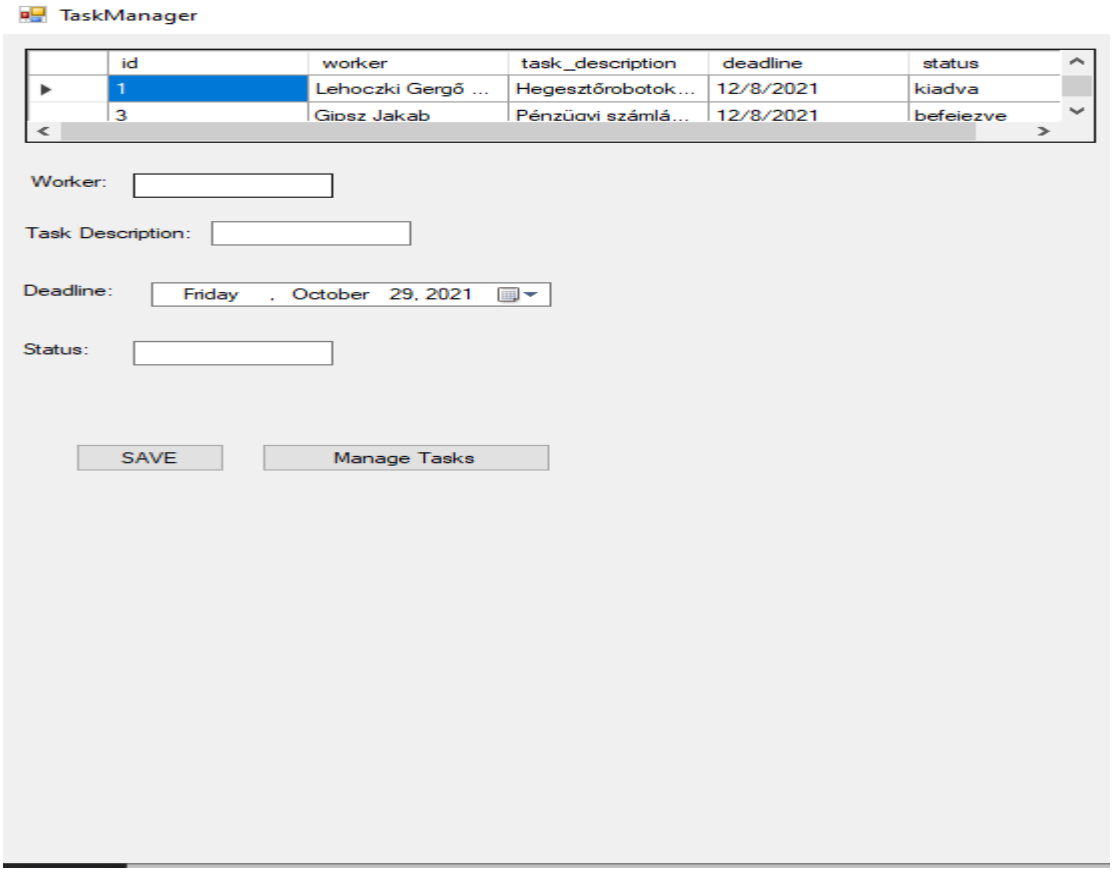
Gondoljunk bele tehát, hogy a fenti algoritmus képes listába tenni azokat az adatokat, amelyek egyeznek a kritériummal, és ezáltal képesek vagyunk arra, hogy egy SQL DELETE parancsba ez által az algoritmus által adott eredményt be helyettesítsük a WHERE feltételhez.Ezt a következő forráskód szemlélteti(14.ábra)

```
private void manageBTN_Click(object sender, EventArgs e)
{
    Selection();
    conn = new MySqlConnection(connString);
    for (int i = 0; i < selectedItems.Count; i++)
    {
        string query = "DELETE FROM company.tasks WHERE status = '" + selectedItems[i].status + "';";
        cmd2 = new MySqlCommand(query, conn);
        conn.Open();
        reader2 = cmd2.ExecuteReader();
    }
    dataGridView1.Refresh();

}
```

14.ábra

Láthatjuk tehát azt, hogy egy iteráció segítségével, végigmegyünk a listában található elemeken, és töröljük azokat a rekordokat az adattáblából amelyeknél be lettek fejezve a feladatok, azaz a státusz már “befejezve” állapotba került.

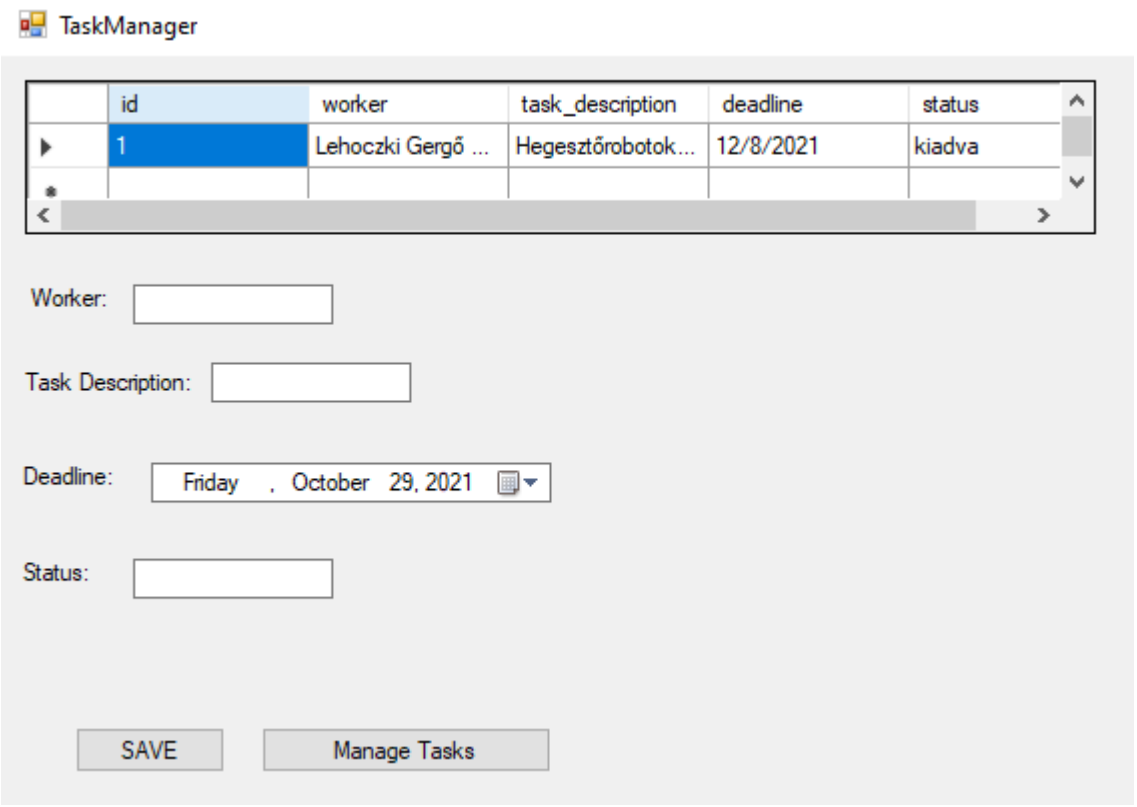
Szemléltessük tehát két ábra segítségével (15,16.ábrák).



15.ábra

Láthatjuk tehát, hogy Gipsz Jakab már elvégezte a feladatát, viszont még mindig megtalálható az adattáblában az ő feladata, amely már “befejezve” státuszban van.

Ez azért sem megfelelő, hiszen feleslegesen foglalja a mi tárhelyünket. Milyen könnyű dolga lesz tehát a főnökének, vagy annak aki kezeli a feladatokat, hiszen egy gombnyomással le tudja frissíteni az aktualitásokat.
A 16.ábra fogja szemléltetni a problémát megoldva.



16.ábra

Vegyük észre tehát, hogy bizony Gipsz Jakab feladata, amely “befejezve” státuszban volt eltűnt miután a Manage Tasks gombra kattintottunk, viszont Lehoczki Gergő Péter feladata, amely “kiadva” státuszban van, maradt, hiszen ezen rekord törlésére nem volt szükségünk.

Hasonló módon természetesen a többi algoritmust is le tudjuk fejleszteni, szintén leltárba vétel, szintén folyamatábra és végül C# kód megvalósításával, viszont ezeket ebben a témában már nem tárgyaljuk.

Ezzel szemben viszont, beszélnünk kell arról, hogy a beosztások fa szerkezetében mikor következhet be változás.
Változásról beszélhetünk akkor, ha egy pozíció megszűnik, ha egy új pozíciót akarunk hozzáadni. Ilyenkor egy új levelet, vagy csomópontot tudunk beilleszteni.

Összehasonlítás

A témánk végéhez érve, hasonlítsuk össze az itt tárgyalt megoldás egy egyszerű ppt és excel kombinációval.Amíg ppt-ben tudjuk ábrázolni a fa adatszerkezetet, addig -bár a ppt itt is jelen van- tovább gondolva a dolgot, egy adattábla segítségével fel is tudtuk rögzíteni a megfelelő beosztásokat, megfelelő származási sorrenddel.Ezt követően könnyen megírtuk a C# programunkat, amely képes törölni egy kiválasztott rekordot, illetve képes azokat megjeleníteni egy adattáblában. Továbbá, egy kiválasztott rekordról képesek voltunk megmondani azt, hogy kitől származik, illetve a rá következőről szintén meg tudjuk ezt mondani.Lássuk be azt a tényt, hogy bizony, amit pusztán, a ppt hordozott magában információkat, azt tudtuk automatizálni.

Továbbá, gondoljunk bele abba, hogy C# program segítségével,képesek vagyunk már új feladatokat a dolgozókhoz rendelni, és ott is képesek vagyunk szűrni bizonyos feladatokra. Ezt egy sima excel táblázatban, bár megtehetjük, de nincs automatizálva a folyamat, tehát a problémát megoldottuk.

Felhasznált fejlesztői eszközök és letölthető állományok

- Visual Studio 2019
- Apex Oracle
- Google Drive PPT
- Google Drive Excel
- Google Drive Word
- Spinning tool
- Trello
- XAMPP
- PhPMyAdmin

Letölthető a forráskód, és a hozzá tartozó adatbázis az alábbi linkről
https://1drv.ms/u/s!AmTMI0Ov6QhVo012DY2w4KTG1yZU?e=OGk9EI

FIGYELEM!!!

A fenti linken letölthető .rar állomány tartalmazza a company nevű mappát és néhány inno-DB file-t, melyeket be kell másolni az XAMPP->mysql->data mappába mert csak ez után lesz meg az adatbázisunk. Természetesen ebből következik, hogy használnunk kell majd az XAMPP-ot.

A másolás során a számítógép többször is megkérdezi majd, hogy biztosan szeretnénk-e felülírni az adatokat, nyugodtan menjünk minden esetben az igen-re.