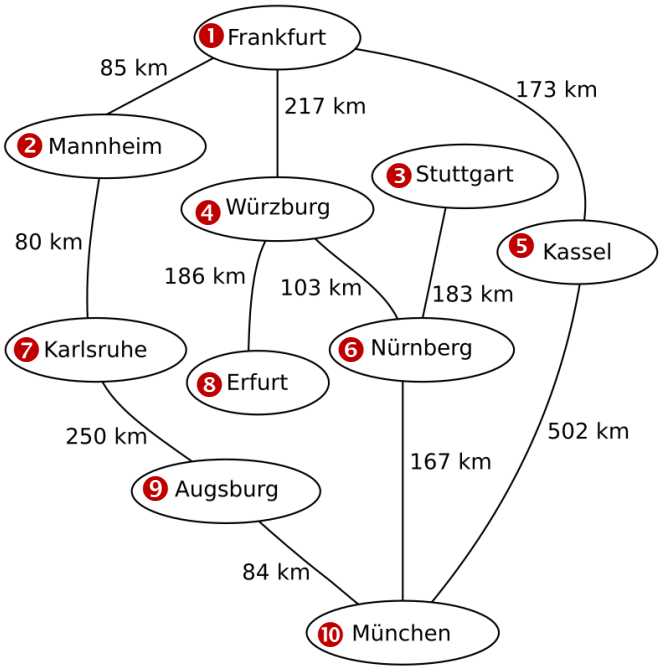
G03. Legrövidebb út I. Szélességi bejárás*

Legrövidebb út I.

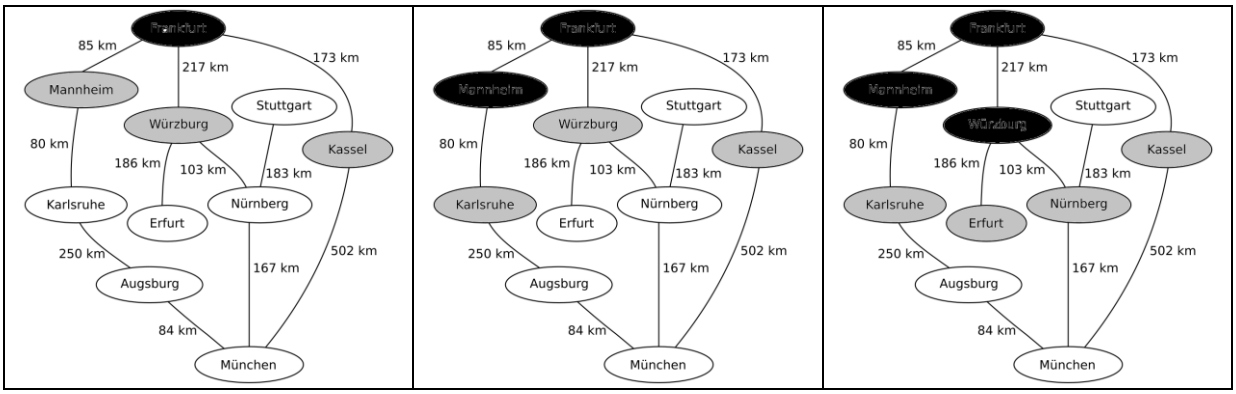
Az ábrán azt látjuk, hogy egyes német városok, hogyan érhetők el Frankfurtból indulva az őket összekötő utak mentén.
Első megközelítésben azt vizsgáljuk, hogy hány nagyvárost érintve jutunk a célhoz (a városok távolságát 1-nek vesszük.)
Az adatokat az ábrán látható gráffal szemléltetjük, matematikailag pedig a gráfot szomszédsági listákkal adjuk meg az alábbi táblázatban látható módon.
Például a 4-es számú városból (Würzburg) egy lépésben elérhető Frankfurt (1), Erfurt (8) és Nürnberg (6). A listát egy 0 zárja.



csúcs	élek			
1	2	4	5	0
2	1	7	0	
3	4	0		
4	1	8	6	0
5	1	10	0	
6	4	3	10	0
7	2	9	0	
8	4	0		
9	7	10	0	
10	9	6	5	0

Szélességi bejárás

A gráf bejárására ezúttal a szélességi bejárást használjuk. Ennek lényege, hogy elindulunk a gráf egy adott pontjából, ahonnan a távolságokat számítjuk (ezúttal Frankfurt) és megvizsgáljuk annak szomszédjait. Majd végig lépkedünk a szomszédokon, és megvizsgáljuk azok szomszédjait.
Az ábrán fekete színnel jelöljük a már megvizsgált várost, szürkével a vizsgálásra előjegyzett városokat (szomszédok) és fehérrel a többit. (Az ábra az első három lépést mutatja).



* *Források:* Képek: https://hu.wikipedia.org/wiki/Szélességi_bejárás, Utolsó letöltés: 2024.10 19.
Algoritmus: https://adatszerk1.elte.hu/?Előadás, Utolsó letöltés: 2024.10 19.

Az algoritmus

A vizsgálandó városok adatainak tárolása egy *sor* segítségével történik (mindig kivesszük a sor elején álló várost, amit feldolgozunk, és mindig betesszük a sor végére azt a város/városokat, amelyet majd fel kell dolgoznunk).

A fenti színkódokkal az algoritmus a következő:
Az elsőnek feldolgozandó város az 1., továbbá a cs-edik város i-edik közvetlen szomszédja a Szomszéd(cs, i).

```
Szélességibejárás(p):
      cs:=1; Szín(cs):=szürke; Sor.Be(cs);
      Honnan(cs):=cs; Táv(cs):=0
      Ciklus amíg Nem_üres(Sor)
            SorKi(cs); Szín(cs):=fekete
            Ciklus i:=1-től Szomszéd_pontok_száma(cs)-ig
                  sz:=Szomszéd(cs,i)
                  Ha Szín(sz) = fehér akkor
                        Sor.Be(sz); Szín(sz):=szürke;
                        Honnan(sz):=cs; Táv(sz)=Táv(cs)+1;
            Ciklus vége
      Ciklus vége
Eljárás vége.
```

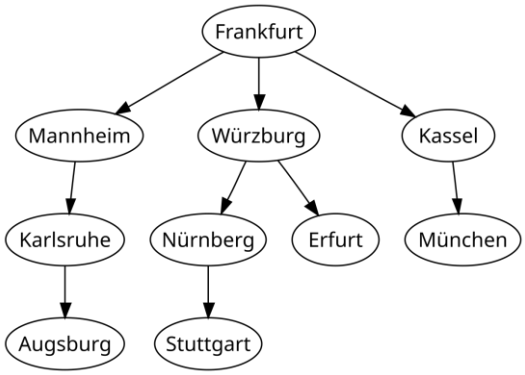
Minimális távolság, feszítőfa

A dőlt betűkkel szedett rész nem tartozik a bejáráshoz, ezek a sorok az adatok feldolgozását végzik. Bejárás után a

Honnan(cs) értéke azt adja meg, hogy a cs csúcsot melyik csúcsból értük el, míg a

Táv(cs) azt, hogy az adott csúcs hány lépésre van a kiindulóponttól.

Bizonyítható, hogy a *Táv(cs)* valóban a legrövidebb távolságot adja meg.
A *Honnan(cs)* segítségével pedig létrejön egy fa, amelynek élei a (cs, Honnan(cs)) élek. Ez a fa már nem tartalmazza az eredeti gráf összes élét, de tartalmazza valamennyi csúcsát (feszítőfa). Egy gráfhoz többféle feszítőfa tartozhat ezúttal az alábbit állítottuk elő:



A programkód C# nyelven

A programban kihasználjuk, hogy a C#-ban a sor kezelésére külön típus áll rendelkezésünkre.

```
namespace SzélességiBejárás
{
    internal class Program
    {
        static int n = 10;
        //Szomszédsági mátrix: az i-edik sorba az i-edik csúcsból iduló
        //élek végpontja szerepel. A számozás 1-től indul, a 0. helyen
        //a csúcsból induló élek száma van
        static int[,] g = new int[n + 1, n + 1];
        //Szinek kezelése
        static int fehér = 0;
        static int szürke = 1;
        static int fekete = 2;
        static int[] szin = new int[n + 1];
        //Sor a szürke pontoknak
        static Queue<int> Q = new Queue<int>();
        //Csúcs Őse, ahonann indul az él
        static int[] honnan = new int[n + 1];
        //Távolság az első elemtől
        static int[] tav = new int[n + 1];

        static void Main(string[] args)
        {
            Beolvasás();
            Bejárás();
            Kiírás();
        }

        static void Kiírás()
        {
            Console.WriteLine("csúcs\telőd\ttávolság");
            for (int i = 1; i <= n; i++)
                Console.WriteLine($"  {i}\t  {honnan[i]}\t   {tav[i]}");
        }

        static void Bejárás()
        {
            //Az első elem bekerül a sorba
            int cs = 1; Q.Enqueue(cs); szin[cs] = szürke;
            //Az első elem feldolgozása
            honnan[cs] = cs; tav[cs] = 0;
            while (Q.Count > 0)
            {
                //Kivesszük a sor következő elemét, őt már feldolgoztuk
                cs = Q.Dequeue(); szin[cs] = fekete;
                //Előkészítjk a szomszédait
                for (int j = 1; j <= g[cs, 0]; j++)
                {
                    //A vizsgált szomszéd
                    int sz = g[cs, j];
                    //Ha még nem vizsgáltuk, bekerül a sorba
                    if (szin[sz] == fehér)
                    {
                        szin[sz] = szürke;
                        Q.Enqueue(sz);
                        //Az adott elem feldolgozása
                        honnan[sz] = cs; tav[sz] = tav[cs] + 1;
                    }
                }
            }
        }
```

```
        static void Beolvasás()
        {
            StreamReader sr = new StreamReader("varosok.txt");
            int i = 0;
            while (sr.Peek() != -1)
            {
                string[] be = sr.ReadLine().Split(' ');
                int j = 0;
                while (int.Parse(be[j]) != 0)
                {
                    g[i + 1, j + 1] = int.Parse(be[j]);
                    j++;
                }
                g[i + 1, 0] = j;
                i++;
            }
            sr.Close();
        }
    }
}
```

Az adatokat tartalmazó fájl tartalma:

varosok.txt:

2 4 5 0
1 7 0
4 0
1 8 6 0
1 10 0
4 3 10 0
2 9 0
4 0
7 10 0
9 6 5 0