Kirándulási sorrend

Innen: Algowiki

Tartalomjegyzék

Feladat

A feladat (https://www.oktatas.hu/pub_bin/dload/kozoktatas/tanulmanyi_versenyek/oktv/oktv2019_2020_donto/info2_flap_d_oktv_1920.pdf#page=6) szerint adott N darab város, illetve az ezeket összekötő M darab útszakasz. Ismerjük továbbá az egyes útszakaszok megtételéhez szükséges időtartamot (Idő$_i$). Tudjuk még, hogy bármely városból el lehet jutni bármely másik városba, valamint hogy egy várost sem köt össze saját magával közvetlenül útszakasz. A feladat szerint az L városból indulva meg akarjuk látogatni az összes többi várost, majd visszatérni az L városba. Meg kell keresni a legrövidebb ilyen útvonalat.

Korlátok

- $1 \leq N \leq 10$
- $1 \leq M \leq 45$
- $1 \leq L \leq 10$
- $1 \leq$ Idő$_i \leq 1000$
- Időlimit: 0.4 mp.
- Memórialimit: 32 MB

Megoldás

1. ötlet

A korlátokat megnézve észrevehetjük, hogy a gráf maximális méretei meglehetősen kicsik. Ebből arra következhetünk, hogy lehet elég minden lehetőséget végigpróbálni, majd kiválasztani a legrövidebbet.

2. ötlet

Vegyük a csúcsok összes lehetséges meglátogatási sorrendjét (permutációját), majd keressük meg a legrövidebb utat az egymást követő csúcsok között. Mivel utóbbit többször is meg kell keresni minden egyes csúcs párra, célszerű a legrövidebb utakat előre meghatározni minden egyes csúcs párra. Ehhez futtathatjuk minden egyes csúcsra a Dijkstra algoritmust, mivel az élek súlyai biztosan nem negatívak. Alternatív lehetőségként használhatjuk a Floyd-Warshall algoritmust is. Utóbbi lassabb lesz, de egyszerűbb implementálni és még ez is belefér az időlimitbe.

Miután meghatároztuk páronként a legrövidebb utakat, megvizsgáljuk a csúcsok összes permutációját és permutációnként kiszámoljuk az útvonal teljes hosszát. A hosszt itt az egymást követő csúcsok közt már kiszámolt legrövidebb távolságok összegeinek, illetve az utolsó és az első csúcs közötti legrövidebb távolság összegeként kapjuk meg.

($\sum_{i=2}^{N} d(p[i-1], p[i]) + d(p[1], p[N])$, ahol $p[i]$ az aktuálisan vizsgált permutációban i. helyen lévő csúcs)

A megoldás az a permutáció, amelyre az előbbiek szerint kiszámolt összhossz a legrövidebb.

3. ötlet

Vegyük észre, hogy a megoldás minden esetben kör, mivel vissza kell térni L-be. Ez azt jelenti, hogy a permutációt ciklikusan eltolva ugyan azt az útvonalat kapjuk. Mivel N elemből áll a permutáció, N-féle képpen kaphatjuk ugyan azt az eredményt ciklikusan eltolva. Ennek kizárásával jelentős mértékben csökkenthetnénk a programunk futásidejét. Rögzítsük a kiindulási várost a permutáció első helyén! Így biztosan minden különböző útvonalat csak egyszer fogunk meghatározni. Mivel N maximum 10, az összes lehetséges permutáció $9! = 362880$. Ennyi lépés már várhatóan belefér a 0.4 másodperces időkeretbe.

Futásidő

A Floyd-Warshall algoritmus futtatása: $O(N^3)$ (Dijkstra: $O((N+M) \cdot \log_2 N)$)

A permutációk vizsgálata: $O((N-1)! \cdot N) = O(N!)$ ($N-1$ permutációt vizsgálunk, mindegyiknél N számot adunk össze)

Összesen: $O(N^3 + N!) = O(N!)$

Memóriaigény

Tárolnunk kell a pontok távolságát páronként: $O(N^2)$

Tárolnunk kell még az aktuális permutációt, az eddigi legjobb megoldást és konstans számú segédváltozót: $O(N)$

Összesen: $O(N^2)$

Implementáció

A 2019/2020-as OKTV döntőjének mintamegoldásai (https://www.oktatas.hu/pub_bin/dload/kozoktatas/tanulmanyi_versenyek/oktv/oktv2019_2020_donto/info2_javutmegoldas_d_oktv_1920.zip) [becsuk]

Megoldás Flyd-Warshall algoritmussal C# nyelven:

```
1 using System;
2 using System.Text;
3 using System.IO;
4
5 class Program{
6     public static void Main(){
7         int n, m, l;
8
9         string[] str = Console.ReadLine().Split();
10
```

```
        n = int.Parse(str[0]);
        m = int.Parse(str[1]);
        l = int.Parse(str[2]) - 1;

        int[,] tav = new int[n,n];
        for(int i = 0; i < n; ++i)
            for(int j = 0; j < n; ++j)
                tav[i,j] = int.MaxValue/2;

        for(int i = 0; i < m; ++i){
            int a, b, ido;
            str = Console.ReadLine().Split();

            a = int.Parse(str[0]) - 1;


            // A lakhely és az első város megcserélése
            if(a == l)
                a = 0;
            else if(a == 0)
                a = l;
            b = int.Parse(str[1]) - 1;
            if(b == l)
                b = 0;
            else if(b == 0)
                b = l;
            ido = int.Parse(str[2]);

            tav[a,b] = ido;
            tav[b,a] = ido;
        }


        // Floyd-Warshall algoritmus
        for(int d = 0; d < n; ++d)
            for(int i = 0; i < n; ++i)
                for(int j = 0; j < n; ++j)
                    tav[i,j] = Math.Min(tav[i,j], tav[i,d]+tav[d,j]);


        int[] sor = new int[n-1];
        int[] megoldas = new int[n-1];
        int minTav = int.MaxValue;

        for(int i = 0; i < n-1; ++i)
            sor[i] = i+1;


        do{
            int osszTav = tav[sor[n-2],0];
            for(int i = 1; i < n-1; ++i)
                osszTav += tav[sor[i-1],sor[i]];
            osszTav += tav[0, sor[0]];

            if(osszTav < minTav){
                minTav = osszTav;
                sor.CopyTo(megoldas,0);
            }
        } while(KovPermutacio(sor));


        StringBuilder sb = new StringBuilder();
        sb.AppendLine(minTav.ToString());

        for(int i = 0; i < n-1; ++i){
            // Lakhely és első város visszacserélése
            if(megoldas[i] == l)
                megoldas[i] = 0;
            sb.Append(megoldas[i]+1);
            sb.Append(' ');
        }

        sb.Length --;   // utolso space torlese

        Console.WriteLine(sb.ToString());

        return;
    }

    static bool KovPermutacio(int[] tomb){
```

```
91          int ind = tomb.Length-2;
92          while(ind >= 0 && tomb[ind] > tomb[ind+1])
93              --ind;
94
95          if(ind == -1)
96              return false;
97
98
99          int bal = ind+1, jobb = tomb.Length-1, kozep;
100         while(bal < jobb){
101             kozep = (bal+jobb+1)/2;
102             if(tomb[kozep] < tomb[ind]){
103                 jobb = kozep-1;
104             } else {
105                 bal = kozep;
106             }
107         }
108
109         int x = tomb[ind];
110         tomb[ind] = tomb[bal];
111         tomb[bal] = x;
112
113         bal = ind+1; jobb = tomb.Length-1;
114         while(bal < jobb){
115             x = tomb[bal];
116             tomb[bal] = tomb[jobb];
117             tomb[jobb] = x;
118             ++bal; --jobb;
119         }
120         return true;
121     }
122
123 }
```

A lap eredeti címe: „https://algowiki.miraheze.org/w/index.php?title=Kirándulási_sorrend&oldid=1354”