FLOYD – UORSHELL ALGORITMI

Farmonov Sherzodbek Raxmonjonovich
Farg‘ona davlat universiteti amaliy matematika va informatika kafedrasi katta o’qituvchisi
farmonovsh@gmail.com
O`ktamjonova Maxfirat Ikromjon qizi
Farg‘ona davlat universiteti talabasi
mahfiratoyoktamjonova@gmail.com

Anotatsiya: *Ushbu maqolada Uorshell algoritimidan foydalanish mumkin bo’lgan sohalar, shu sohalardagi qo’llash mumkin bo’lgan o’rinlar keltirilgan bo’lib sun’iy intellekt va mashinalar bilan ishlashga doir misol va masalalarni Uorshell algoritimi yordamida samarali yechish va tahlil qilish ko’rib chiqilgan.*

Kalit so‘zlar: *Uorshell algoritimi, sun’iy intellekt, grafda harakat,kenglik bo’yicha qidiruv, eng qisqa yo’l topish.*

Annotation: *This article lists the areas in which the Uorshell algorithm can be used, the areas that can be applied in these areas, and the effective solution and analysis of examples and issues of working with artificial intelligence and machines using the Uorshell algorithm.*

Keywords: *Uorshell algorithms, artificial intelligence, action in the graph, search by latitude, find the shortest path.*

Аннотация: *В данной статье представлены области, в которых может использоваться алгоритм Uorshell, возможные места применения в этих областях, а также рассмотрены примеры и проблемы, связанные с работой с искусственным интеллектом и машинами, а также эффективное решение и анализ проблем с использованием рассмотрен алгоритм Uorshell.*

Ключевые слова: *алгоритм Uorshell, искусственный интеллект, обход графа, поиск по широте, поиск кратчайшего пути*

Muammo yoki mavzu haqida qisqacha tanishtirish: Bu algoritmning nomlanishi 1962 yilda bir vaqtning o'zida kashf etgan ikkita amerikalik tadqiqotchilar Robert Floyd va Stiven Uorshell sharafiga qo'yilgan. Nomlanishning boshqa variantlari kamroq tarqalgan: Roy - Uorshall algoritmi yoki Roy - Floyd algoritmi. Roy - bu algoritmni hamkasblaridan 3 yil oldin (1959 yilda) ishlab chiqqan professorning familiyasidir, ammo uning kashfiyoti nashr qilimay qolgan. Floyd-Uorshell algoritmi – grafning har bir tugunigacha bo'lgan qisqa masofalarni hisoblashning dinamik algoritmidir. Bu metodni musbat va manfiy vaznga ega bo'lgan graflarda qo'llash mumkin, faqat manfiy sikllardan tashqari. Shu sababli oldin ko'rib o'tilgan Deykstra algoritmiga qaraganda umumiyroq hisoblanadi.

Algoritmning natijalari va samaradorligi: Floyd-Uorshell algoritmini amalga oshirish uchun har bir tuguni 1 dan |V| gacha nomerlangan G=(V,E) grafning qo'shma matrisasi D[][] ni tashkil etamiz. Bu matrisa |V|x|V| o'lchamga ega bo'ladi va har bir D[i][j] elementga i dan j gacha bo'lgan qirralarning vazni o'zlashtiriladi. Algoritm bajarilishi mobaynida ushbu matrisa qayta yoziladi: uning har bir yacheykasiga i tugundan j tugungacha hisoblangan eng optimal, qisqa masofa yoziladi. Endi algoritmning asosiy qismini tuzishdan oldin, eng qisqa yo'llarning matritsasi tarkibini tushunish kerak. Uning har bir elementi D[i][j] mavjud bo'lgan marshrutlarning eng kichikini o'z ichiga olishi kerakligi sababli, biz darhol ayta olamizki, yagona tugundan uchun u nolga teng, hattoki pastadir (manfiy sikllar hisobga olinmaydi), shuning uchun asosiy diagonalning barcha elementlari (D[i][i]) ni 0 qilish kerak.

Diagonalda bo’lmagan 0 qiymatli elementlar (matrisaning i va j tugunlari o'rtasida bevosita qirra mavjud bo'lmagan joylarida nollar bo'lishi mumkin) ularning qiymatini iloji boricha o'zgartirish uchun biz ularni cheksizlik bilan tenglashtiramiz, bu dasturda bo'lishi mumkin, masalan, grafda mumkin bo'lgan maksimal yo'l, yoki shunchaki katta son. Algoritmning uchta - sikl, ifoda va shartli operatordan iborat asosiy qismi juda ixcham tarzda yoziladi.

k=1 dan |V| gacha bajariladi

i=1 dan |V| gacha bajariladi

j=1 dan |V| gacha bajariladi

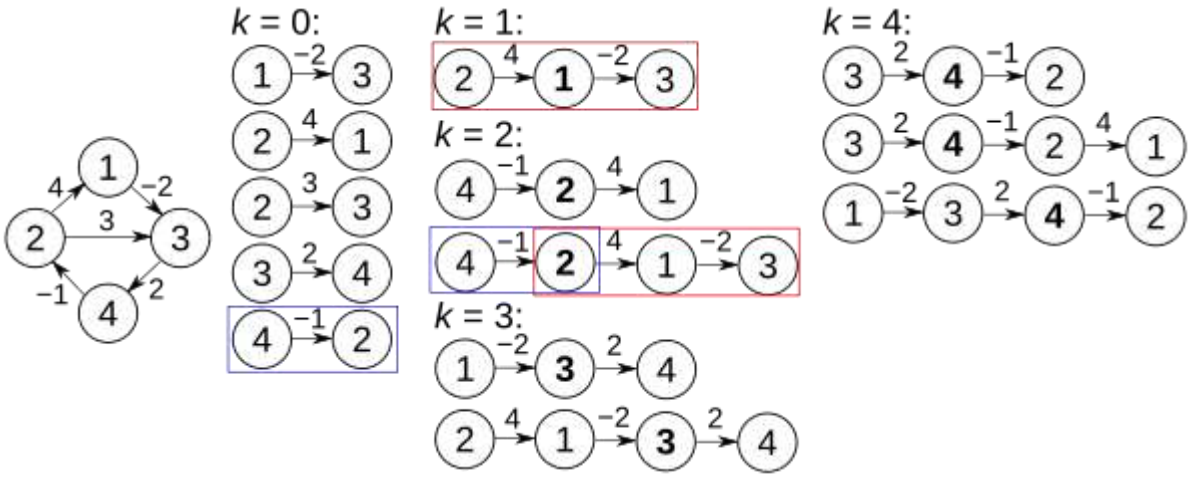
agar D[i][k]+D[k][j]<D[i][j] bo’lsa, u holda D[i][j]

←D[i][k]+D[k][j]

i tugunidan j tugunigacha eng qisqa yo'l ular orqali va boshqa tugunlar to'plamidan o'tishi mumkin k∈(1, ..., |V|). i dan j gacha bo'lgan yo'l k tugundan o’tishi yoki o'tmasligi ham mumkin. Agar boshqa yo'l mavjud bo’lsa, u i dan k ga, keyin k dan j gacha o'tishini anglatadi, shuning uchun u qisqa yo'lning qiymati D[i][j] ni D[i][k] + D[k][j] yig'indi bilan almashtirish kerak.

Floyd-Uorshall algoritmi har bir juft cho'qqi orasidagi grafik orqali barcha mumkin bo'lgan yo'llarni solishtiradi. Bunga ikki cho'qqi orasidagi eng qisqa yo'lni baholashni sekin-asta takomillashtirish orqali erishiladi, shunda taxmin optimal bo'ladi.

Algoritmni bajarilishi:



Yuqorida k = 0 deb belgilangan tashqi halqaning birinchi rekursiyasidan oldin faqat ma'lum bo'lgan yo'llar grafikning alohida qirralariga to'g'ri keladi. k = 1 bo'lganda, 1 cho'qqisidan o'tuvchi yo'llar topiladi: xususan, yo'l [2,1,3] topiladi, u yo'lni [2,3] o'rnini egallaydi, uning qirralari kamroq, lekin uzunroq (og'irlik bo'yicha). k = 2 bo'lganda 1,2 cho'qqilardan o'tuvchi yo'llar mavjud. Qizil va ko'k qutilar [4,2,1,3] yo'lning oldingi iteratsiyalarda uchragan ikkita ma'lum yo'llardan [4,2] va [2,1,3] qanday yig'ilganligini ko'rsatadi, chorrahada 2. [4,2,3] yo'li hisobga olinmaydi, chunki [2,1,3] 2 dan 3 gacha bo'lgan eng qisqa yo'ldir. k = 3 uchun 1,2,3 cho'qqilardan o'tuvchi yo'llar topiladi. Nihoyat, k = 4 bo'lganda, barcha eng qisqa yo'llar topiladi.

Har bir iteratsiyadagi masofa matritsasi k, yangilangan masofalar **qalin harf** bilan quyidagicha ko'rinadi:

$k = 0$

i \ J	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=1$		**1**	**2**	**3**	**4**
	1	0	∞	−2	∞
i	**2**	4	0	**2**	∞
	3	∞	∞	0	2
	4	∞	−1	∞	0
$k=2$		J			
		1	**2**	**3**	**4**
	1	0	∞	−2	∞
i	**2**	4	0	2	∞
	3	∞	∞	0	2
	4	**3**	−1	**1**	0
$k=3$		J			
		1	**2**	**3**	**4**
	1	0	∞	−2	**0**
i	**2**	4	0	2	**4**
	3	∞	∞	0	2
	4	3	−1	1	0
$k=4$		J			
		1	**2**	**3**	**4**
	1	0	**−1**	−2	0
i	**2**	4	0	2	4
	3	**5**	**1**	0	2
	4	3	−1	1	0

Floyd-Warshall algoritmining vaqt murakkabligi $O(n^3)$ bo`ladi. Bu, ya'ni, grafda n ta vertikal bo'lsa, algoritmning bajarilish vaqti n^3 bo'ladi.

Bu, kichik grafiklar uchun samarali bo'lsa-da, katta grafiklar uchun juda sekin ishlashi mumkin, chunki n^3 ko'tarilgan qiymatlar juda katta bo'lishi mumkin.

Biroq, Floyd-Warshall algoritmi juda qulay va sodda hisoblanadi.

Transport tizimlarida, masalan, shaharlarda yoki viloyatlararo yo‘llar tizimida, eng qisqa yo‘lni topish muhim masala hisoblanadi. Agar har bir shahar yoki nuqta o‘rtasida yo‘l bor bo‘lsa, Floyd-Warshall algoritmi barcha juft nuqtalar o‘rtasidagi eng qisqa yo‘lni hisoblash uchun ishlatiladi. Agar har bir nuqta (shahar yoki transport stansiyasi) o‘rtasida to‘g‘ridan-to‘g‘ri yo‘llar mavjud bo‘lsa, Floyd-Warshall algoritmi barcha juft shaharlarga o‘rtasidagi eng qisqa yo‘llarni topishda ishlatiladi. Bu nafaqat transport tarmog‘ini tahlil qilish, balki yo‘l tarmoqlarini optimallashtirish uchun ham muhim. Temir yo‘l va avtobus yo‘llarida ham yo‘llar orasida eng qisqa yo‘lni aniqlashda foydalaniladi. Shuningdek, ko‘p shaharlar va stansiyalarni o‘zaro bog‘laydigan tizimlarda, bu algoritm orqali barcha nuqtalar orasidagi eng qisqa vaqt yoki masofani hisoblash mumkin. Agar transport tarmog‘ida ko‘p nuqtalar mavjud bo‘lsa (masalan, yuklarni ko‘plab omborlardan mijozlarga etkazib berish), Floyd-Warshall algoritmi barcha omborlar va manzillar o‘rtasida eng qisqa yo‘llarni aniqlash uchun ishlatiladi. Bu yo‘lni tanlash orqali, yuklarni eng tez va samarali yo‘l bilan etkazib berish mumkin. Transport tizimining samaradorligini oshirish maqsadida yangi yo‘llar yoki yo‘nalishlar qo‘shilsa, yo‘llarni optimallashtirish va tarmoqni kengaytirish uchun Floyd-Warshall algoritmi ishlatiladi. Masalan, transport tarmog‘ida mavjud bo‘lgan yo‘llarning o‘zgarishi yoki yangi yo‘llarning qo‘shilishi bilan, bu algoritmning yangi yo‘llarni hisoblash va tizimni qayta baholash imkoniyatlari mavjud.

Floyd-Uorshall algoritmi odatda faqat barcha cho'qqilar juftligi orasidagi yo'l uzunligini ta'minlaydi. Oddiy o'zgartirishlar yordamida har qanday ikkita oxirgi nuqta cho'qqilari orasidagi haqiqiy yo'lni qayta qurish usuli yaratilishi mumkin. Har bir cho'qqidan boshqa cho'qqigacha

bo'lgan 3 ta haqiqiy yo'lni saqlashga moyil bo'lishi mumkin bo'lsa-da, bu shart emas. Buning o'rniga, har bir tugun uchun eng qisqa yo'l daraxtini hisoblash mumkin xotiradan foydalanish vaqtihar bir daraxtni saqlash, bu bizga har qanday ikkita bog'langan cho'qqidan yo'lni samarali tarzda qayta qurish imkonini beradi.

Tadqiqotning qisqacha natijasi:

Tadqiqotda Floyd-Warshallning barcha juft nuqtalar bo'yicha eng qisqa yo'lni topishdagi tahlil qilingan. UO(n^3) murakkablikka ega bo'lib, grafning har bir tuguni uchun barcha juft nuqtalar masofani ajratib beradi. Bu o'ziga xos afzalliklarni taqdim etsa-da, katta va siyrak grafalarda samaradorlik masalalari kelishish mumkin.

Floyd-Warshall algoritmi kichik grafalarda (nuqtalar soni kichik bo‘lsa) juda samarali ishlaydi, chunki uning vaqt murakkabligi nisbatan past bo‘ladi va xotira iste'moli ham cheklangan. Agar grafda nuqtalar soni katta bo‘lsa, O(n^3) vaqt murakkabligi katta bo‘lishi mumkin. Masalan, 1000 ta nuqtali graf uchun $n^3=1000^3=1,000,000,000$ operatsiya talab etiladi, bu esa ko‘p vaqt talab qilishi mumkin. Shuning uchun katta grafalarda bu algoritmni ishlatish sekinlashadi. Grafning kattaligi bilan birga xotira iste'moli ham ortadi. O(n^2) xotira talab etilishi, graf katta bo‘lsa, tizimda xotira cheklovlari bo‘lishi mumkin. Dasturlashni o‘rganishda, ayniqsa dinamik dasturlashga oid masalalarni yechishda, Floyd-Warshall algoritmi o‘rgatishda juda foydalidir.Agar maqsad barcha juft nuqtalar orasidagi eng qisqa yo‘llarni topish bo‘lsa va graf o‘rtacha kattalikda bo‘lsa, Floyd-Warshall algoritmi yaxshi yechim bo‘lishi mumkin.

Umuman olganda, tadqiqot Floyd-Warshall algoritmining jarayonini yanada yaxshilash va uni yaxshilashni nazorat qilish uchun yangi yondoshuv va texnologiyani qo'llashni ko'rib chiqish.

Floyd-Uorshall algoritmi odatda faqat barcha cho'qqilar juftligi orasidagi yo'l uzunligini ta'minlaydi. Oddiy o'zgartirishlar yordamida har qanday ikkita oxirgi nuqta cho'qqilari orasidagi haqiqiy yo'lni qayta qurish usuli yaratilishi mumkin. Har bir cho'qqidan boshqa cho'qqigacha bo'lgan 3 ta haqiqiy yo'lni saqlashga moyil bo'lishi mumkin bo'lsa-da, bu shart emas. Buning o'rniga, har bir tugun uchun eng qisqa yo'l daraxtini hisoblash mumkin xotiradan foydalanish vaqtihar bir daraxtni saqlash, bu bizga har qanday ikkita bog'langan cho'qqidan yo'lni samarali tarzda qayta qurish imkonini beradi.

Floyd Uorshall Algoritmi C# Kodida

```
using System;
class Program
{
   const int INF = int.MaxValue; // Yoki cheksiz kattalik uchun qiymat

   static void FloydWarshall(int[,] graph, int n)
   {
      // graph - boshlang'ich graf
      // n - vertikal soni

      // graphni yangilab, eng qisqa yo'llarni hisoblash
      for (int k = 0; k < n; k++) // oraliq nuqtani tanlash
      {
         for (int i = 0; i < n; i++) // boshlang'ich nuqtani tanlash
         {
            for (int j = 0; j < n; j++) // oxirgi nuqtani tanlash
            {
               // eng qisqa yo'lni yangilash
               if (graph[i, k] != INF && graph[k, j] != INF && graph[i, k] +
graph[k, j] < graph[i, j])
                  graph[i, j] = graph[i, k] + graph[k, j];
               }
            }
         }
```

```
        }
    }
    static void PrintSolution(int[,] dist, int n)
    {
        Console.WriteLine("Eng qisqa yo'llar (dist[vertikal1, vertikal2]):");
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                if (dist[i, j] == INF)
                    Console.Write("INF ".PadRight(5));
                else
                    Console.Write(dist[i, j].ToString().PadRight(5));
            }
            Console.WriteLine();
        }
    }
     static void Main()
    {
        int n = 4; // Vertikal soni
        int[,] graph = {
            { 0, 3, INF, INF },
            { INF, 0, 1, INF },
            { INF, INF, 0, 7 },
            { 2, INF, INF, 0 }
        };
        Console.WriteLine("Boshlang'ich graf (og'irliklar):");
        PrintSolution(graph, n);

        FloydWarshall(graph, n); // Floyd-Warshall algoritmini chaqirish

        Console.WriteLine("\nEng qisqa yo'llar hisoblandi:");
        PrintSolution(graph, n);
        Console.ReadKey();
    }
}
```

Natijasi:

Xulosa

Floyd-Warshall algoritmi — bu barcha juft vertikal o'rtasidagi eng qisqa yo'llarni topish uchun juda qulay va sodda usuldir. Kichik graflar uchun juda samarali bo'lsa-da, uning $O(n^3)$ vaqt murakkabligi katta graflar uchun unchalik samarali emas. Yana bir kamchilik — bu algoritm xotira talablarini ham ancha oshiradi, chunki har bir vertikal uchun juft vertikalni saqlash kerak bo'ladi. Shunga qaramay, agar sizga butun graf uchun eng qisqa yo'llar kerak bo'lsa, Floyd-Warshall algoritmi juda foydali va samaralidir. Xotiradan samarali foydalanish imkoniyatini oshirish orqali grafikdagi barcha ma’lumotlarni qamrab olish va natijalarni tezroq qaytarish imkonini beradi. Kelajakda Floyd Uorshall algoritmini yanada takomillashtirish yo‘nalishlari bo‘yicha ilmiy izlanishlar davom etmoqda, va bu sohada qator istiqbolli yo‘nalishlar mavjud. Ammo, katta grafalarda samaradorligi sezilarli darajada pasayadi va boshqa optimallashtirilgan algoritmlar ko‘proq afzalliklarga ega bo‘lishi mumkin.

Foydalanilgan adabiyotlar:

1. Knuth, D.E. (1997). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.
2. Aho, A.V., Hopcroft, J.E., & Ullman, J.D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.
3. Mehlhorn, K., & Sanders, P. (2008). *Algorithms and Data Structures: The Basic Toolbox*. Springer.

4. Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Addison-Wesley Professional.
5. Even, S. (2011). *Graph Algorithms*. Cambridge University Press.
6. Kleinberg, J., & Tardos, E. (2006). *Algorithm Design*. Addison-Wesley.
7. Tarjan, R.E. (1983). *Data Structures and Network Algorithms*. Society for Industrial and Applied Mathematics.
8. Eppstein, D., Galil, Z., Italiano, G.F., & Spencer, T. (1997). "Separator-Based Parallel Algorithm for the Single-Source Shortest Path Problem," *Journal of Algorithms*, 23(1), 122-145.
9. Valiant, L.G. (1990). "A Bridging Model for Parallel Computation," *Communications of the ACM*, 33(8), 103-111.
10. Papadimitriou, C.H. (1994). *Computational Complexity*. Addison-Wesley.
11. Alon, N., & Spencer, J.H. (2008). *The Probabilistic Method*. Wiley.