GRAFDAGI YO'NALTIRILGAN VA YO'NALTIRILMAGAN SIKLLARNI DFS YORDAMIDA ANIQLASH

Sh.R.Farmonov
Farg'ona davlat universiteti amaliy matematika va informatika kafedrasi katta o'qituvchisi
farmonovsh@gmail.com
A.V.Saxobiddinov
Farg'ona davlat unversiteti talabasi
sahobiddinovadhamjon@gmail.com

Anotatsiya: *DFS (Depth First Search) algoritmi graf va daraxt tuzilmalarida chuqur qidiruvni amalga oshiruvchi asosiy algoritmlaridan biridir. Ushbu algoritm graf yoki daraxtdagi barcha tugunlarni chuqurroq tekshirib, har bir tugunni faqat bir marta o'rganishga asoslanadi. DFS algoritmining ishlash prinsipi boshlang'ich tugundan boshlab, uning qo'shni tugunlariga o'tib, har bir tugunni qayta-qayta tekshirib chiqishni o'z ichiga oladi.*

Kalit so'zlar: *DFS algoritmi, Depth First Search, graf, daraxt, qidiruv algoritmi, rekursiv yondashuv, iterativ yondashuv, bog'lanish, topologik tartib, tsikl aniqlash, graflarda chuqur qidiruv, daraxtda qidiruv, ma'lumotlar tuzilmalari, optimizatsiya, algoritm samaradorligi.*

Abstract: *The DFS (Depth First Search) algorithm is one of the main algorithms used for deep search in graph and tree structures. This algorithm works by deeply examining all the nodes in a graph or tree, ensuring that each node is explored only once. The working principle of the DFS algorithm involves starting from an initial node and proceeding to its neighboring nodes, exploring each node without revisiting it.*

Keywords: *DFS algorithm, Depth First Search, graph, tree, search algorithm, recursive approach, iterative approach, connectivity, topological order, cycle detection, deep search in graphs, search in trees, data structures, optimization, algorithm efficiency.*

Аннотация: *Алгоритм DFS (поиск в глубину) является одним из основных алгоритмов для выполнения глубинного поиска в графах и деревьях. Этот алгоритм работает, глубоко исследуя все узлы в графе или дереве, гарантируя, что каждый узел исследуется только один раз. Принцип работы алгоритма DFS заключается в том, чтобы начать с начального узла и переходить к его соседним узлам, исследуя каждый узел без повторного посещения.*

Ключевые слова: *алгоритм DFS, поиск в глубину, граф, дерево, алгоритм поиска, рекурсивный подход, итеративный подход, связность, топологический порядок, обнаружение циклов, глубинный поиск в графах, поиск в деревьях, структуры данных, оптимизация, эффективность алгоритма.*

DFS (Depth-First Search) algoritmi, graflar va daraxtlar kabi ma'lumotlar tuzilmalarini o'rganishda keng qo'llaniladi. Bu algoritmning ko'plab afzalliklari bor, ammo u ba'zi kamchiliklarga ham ega. DFSning afzalliklarini va kamchiliklarini batafsil tahlil qilish orqali uning turli vazifalar va muammolar uchun mosligini yaxshiroq tushunish mumkin. DFS algoritmining eng katta afzalliklaridan biri uning oddiyligi va samaradorligidir. Rekursiya yordamida yozilishi oson bo'lgan bu algoritm, graf yoki daraxtning har bir tugunini faqat bir marta tekshiradi, shuningdek, uni o'rganish uchun juda oddiy strukturaga ega. DFS algoritmi orqali ma'lumotlar tuzilmasi chuqurdan o'rganiladi, bu esa ba'zi vazifalar uchun juda qulay. Masalan, ba'zi muammolar, masalan, labirintlarni echish yoki grafdagi bog'lanish komponentlarini aniqlash, DFS yordamida samarali hal qilinadi. DFS algoritmi har bir tugunni chuqurdan o'rganishga qaratilgan bo'lib, bu ba'zi vazifalarda natijalarni tezda topishga yordam beradi. DFSni ishlatishda xotira sarfi odatda kichik bo'ladi, chunki algoritm har safar faqat eng chuqur tugunga qadar borib, o'rganilgan tugunlar ro'yxatiga qo'shadi. Bu algoritmning o'ziga xos jihati, ya'ni graflarda yoki daraxtlarda mavjud bo'lgan barcha bog'lanishlarni tahlil qilishda uning samaradorligidir. DFS algoritmida ba'zi holatlarda tugunlarni qayta ko'rish ehtimoli mavjud, bu esa graflarda mavjud bo'lgan yo'llarni to'liq o'rganishda muammolar yaratishi mumkin. Rekursiya bilan ishlashda xatolik yuzaga kelishi ehtimoli ortadi. Agar rekursiya chuqurligi juda katta bo'lsa, bu tizimda xotira chegaralarini oshirib, stack overflow (stakning to'lib ketishi) muammosini keltirib chiqarishi mumkin. Bu ayniqsa, juda katta graflarda va chuqur daraxtlar bilan ishlaganda ko'zga tashlanadi. DFSning yana bir kamchiligi shundaki, u har doim bir xil tartibda tekshiradi va har bir tugunni to'liq o'rganishga urinadi. Bu algoritmning ba'zi holatlarda samaradorligini kamaytiradi. Masalan, agar grafda mavjud bo'lgan ma'lumotlarning ba'zilari sezilarli darajada ko'proq bo'lsa, DFS ular ustida ishlashni boshlaydi va vaqtni keraksiz sarflaydi, bu esa ba'zi muammolarni tezda hal qilishni qiyinlashtiradi. DFS o'zining chuqur tekshirish usuli bilan yaxshi ishlaydi, lekin kengaytirilgan yoki katta graflar uchun bu usul ko'proq vaqt va xotira sarfini talab qiladi. DFS algoritmining boshqa bir muhim tomoni — uning xotira ishlatish usuli. DFSni amalga oshirishda, barcha tekshirilgan tugunlar va ularning holatlari (masalan, o'rganilgan yoki yo'q) xotirada saqlanadi. DFS, grafdagi tugunlarni tekshirishda "visited" (ko'rilgan) massividan foydalanadi, bu massivda har bir tugunning ko'rilganligini yoki ko'rilmaganligini tekshirish imkoniyatini beradi. Agar biror tugun allaqachon tekshirilgan bo'lsa, u holda uni qayta tekshirishning hojati yo'q. DFSning

ishlash tamoyilida yana bir jihat — barcha tugunlar to'liq o'rganilmagan holda, ortga qaytish va boshqa yo'nalishlarni tekshirishdir. Bu, ayniqsa, rekursiv yondashuvda o'zini namoyon qiladi, chunki algoritm qaysidir nuqtada borib tugalganida, yana orqaga qaytib boshqa tugunlarni o'rganishga o'tadi. DFS algoritmi ba'zi holatlarda iterativ usulda ham amalga oshirilishi mumkin, bunda rekursiya o'rniga, stack yordamida har bir tugun tekshiriladi. Stack yordamida ishlashda, DFSning ishlash tamoyilida rekursiv chaqiriqlar o'rniga, har bir yangi tugun stack'ga qo'shiladi va undan keyingi tugunlar stack yordamida o'rganiladi. Bu usulda, algoritm stackni ishlatib, har bir tugun va uning qo'shnilarini tekshiradi, va har bir yangi tugun uchun stackga qo'shilgan tugunlar keyingi navbatda tekshiriladi. Bu yondashuvda DFS, stack orqali rekursiyani simulyatsiya qiladi va shu tarzda chuqur izlashni amalga oshiradi. DFS algoritmining ishlash tamoyilini tushunishda yana bir muhim jihat — uni graflarda va daraxtlarda qanday ishlatishning farqidir. Daraxtda, DFSning ishlash tamoyili nisbatan soddaroq, chunki har bir tugun faqat bitta ota tuguniga ega bo'ladi va har bir tugunni faqat bir marotaba tekshirish kifoya qiladi. Graflarda esa, tugunlar orasidagi bog'lanishlar murakkablashadi va ba'zida grafda tsikllar bo'lishi mumkin. Shuning uchun, DFS algoritmida tsikllarni aniqlash muhim ahamiyatga ega. Agar biror tugun avval o'rganilgan bo'lsa, bu tugunni qayta tekshirishning hojati yo'q, va bu holda algoritm tsiklni aniqlaydi.

DFS algoritmini implementatsiya qilishning asosiy bosqichlari – graflarni to'g'ri ta'riflash, har bir tugunni tekshirish, va rekursiv yoki iterativ yondashuv orqali grafni o'rganish. Graflarni ta'riflashda, odatda, tugunlar va ularning o'zaro bog'lanishlari ro'yxatini saqlash uchun ma'lumotlar tuzilmasi sifatida ro'yxat (list) yoki lug'at (dictionary) ishlatiladi. Har bir tugun graflarda boshqa tugunlar bilan bog'langan bo'ladi, shuning uchun bu bog'lanishlar grafning tuzilmasi sifatida saqlanadi. DFSning asosiy tamoyili bu boshlang'ich tugundan boshlab uning qo'shni tugunlariga ketma-ket kirish, har bir tugunni chuqur o'rganishdir. Har bir tugun tekshirilganda, u boshqa tugunlarga kirishga harakat qiladi, shu bilan rekursiv chaqiruvlar orqali grafni tahlil qiladi. DFS algoritmining implementatsiyasida, ba'zi holatlarda, tugunlar ko'rilganmi yoki yo'qmi ekanligini bilish uchun qo'shimcha strukturalar kerak bo'ladi. Masalan, ko'rilgan tugunlar ro'yxati yoki seti (set) orqali har bir tugunni tekshirish va uni qayta tekshirmaslik ta'minlanadi. DFS algoritmi rekursiv usulda amalga oshirilsa, bunda har bir chaqiruv davomida yangi tugunlar tekshiriladi va yangi tugunlar yordamida yana yangi chaqiruvlar amalga oshiriladi. Boshqacha qilib aytganda, DFS rekursiv ravishda graflarni chuqur o'rganadi, har bir tugunni tekshiradi va keyin unga bog'langan tugunlarni tekshirish uchun yangi chaqiruvlar yuboradi. Bu jarayon har bir tugun o'rganilgach, orqaga qaytish orqali amalga oshiriladi. DFSning implementatsiyasida, ba'zi hollarda, bu jarayonni stack yordamida iterativ ravishda ham amalga oshirish mumkin. Iterativ yondashuvda, stackga yangi tugunlar qo'shiladi va tugunlar stackdan birin-ketin olinadi, har biri o'zining qo'shni

tugunlari bilan tekshiriladi. Bu usulda, rekursiyaning o'rniga stackning yordamida DFS amalga oshiriladi. Stack yordamida DFS algoritmning o'rganishi va rekursiyaning takrorlanishidan saqlanish mumkin, bu esa xotira sarfini va rekursiyaning chuqurligini nazorat qilishga yordam beradi. DFSning implementatsiyasi jarayonida eng muhim qadamlar - bu har bir tugunni tekshirish, uni qayta tekshirmaslik uchun belgilash va tegishli ma'lumotlar tuzilmasini yangilashdir. DFSni ishlatish uchun, dastlab, boshlang'ich tugun ko'rsatiladi va undan keyin har bir tugun rekursiv yoki iterativ ravishda tekshiriladi. Bu jarayonda, DFS algoritmi har bir yangi tugun uchun qo'shni tugunlarga o'tib, ulardan yana yangi tugunlar olish orqali grafni o'rganadi. DFSni implementatsiya qilishda bir nechta masalalarni hal qilish mumkin. Masalan, graflarda tsikl mavjudligini aniqlash, grafni to'liq o'rganish, yoki graflarning bog'lanish komponentlarini tahlil qilishda DFS algoritmiga murojaat qilish mumkin. Agar DFS graflarda tsiklni aniqlash kerak bo'lsa, bu uchun algoritmning rekursiv stekini tekshirish yordamida bu masalani hal qilish mumkin. Agar rekursiv stekda allaqachon mavjud bo'lgan tugun qayta o'rganilsa, demak, grafda tsikl mavjud deb hisoblanadi. DFS algoritmining implementatsiyasini amalga oshirayotganda, asosan ikkita muhim holatni hisobga olish kerak. Birinchidan, rekursiya orqali ishlashda xotira chegaralari bilan bog'liq muammolar yuzaga kelishi mumkin. Agar graf juda katta bo'lsa yoki rekursiya chuqurligi yuqori bo'lsa, bu holda tizimda xotira cheklovlari tufayli **stack overflow** (rekursiya stekining to'lishi) xatoliklari yuzaga kelishi mumkin. Bunday holatlarda iterativ yondashuvni qo'llash yoki grafni kichik qismlarga bo'lib o'rganish kerak bo'ladi. Ikkinchidan, DFS algoritmi barcha yo'llarni o'rganish imkonini bersa-da, uning eng qisqa yo'lni topishda samaradorligi pastroq bo'lishi mumkin. DFS grafikdagi barcha mumkin bo'lgan yo'llarni o'rganadi, lekin har doim eng qisqa yo'lni topa olmaydi, chunki u avvalgi yo'llarni to'liq tekshirishni afzal ko'radi. Boshqa yondashuvlar, masalan, BFS (Breadth-First Search) algoritmi, eng qisqa yo'lni topishda yaxshiroq ishlashi mumkin, chunki u tugunlarni kengaytirish orqali eng qisqa yo'lni tezda topadi. DFS algoritmi, shuningdek, kichik va o'rta graf tuzilmalarida juda samarali ishlaydi. Graflar tuzilmasi nisbatan sodda bo'lsa yoki katta hajmdagi graflar bilan ishlanayotgan bo'lsa, DFS juda qulay yondashuvni taqdim etadi. Rekursiv yondashuv orqali graflarda chuqur izlanishlar amalga oshiriladi va har bir tugun faqat bir marta tekshiriladi, bu esa algoritmning samaradorligini oshiradi. DFS algoritmi bilan o‘rganilgan grafning har bir tuguni boshqa tugunlar bilan aloqasini chuqur tahlil qilish mumkin, bu esa uning afzalliklarini kuchaytiradi.

DFSning boshqa bir katta afzalligi uning graflarda tsikllarni aniqlashda yordam berishidir. Agar grafda tsikl mavjud bo'lsa, DFS rekursiv stek orqali bu tsiklni aniqlay oladi. Grafda tsikl mavjud bo'lsa, DFS algoritmi qaytib kelgan tugunni qayta tekshirishni boshlaydi va shu tariqa tsiklni topish imkonini beradi. DFS algoritmi yordamida

graflarning bog'lanish komponentlarini aniqlash ham mumkin, bu esa grafning ichki tuzilmasini tahlil qilishda juda foydalidir. DFS orqali o‘rganiladigan graflar yordamida bog'lanish tarmog'ining tuzilishi yoki alohida bog'lanish komponentlarini aniqlash mumkin bo‘ladi. Bu, ayniqsa, tarmoqlarni tahlil qilish, grafni ajratish yoki bo‘linmalarni aniqlashda muhim rol o‘ynaydi. DFS algoritmi muayyan graf tuzilmalarida, masalan, labirintlarni echishda yoki o‘rganishda ham samarali ishlaydi. DFSning yana bir kamchiligi, uning eng qisqa yo‘lni topishdagi samaradorligining pastligi bilan bog‘liqdir. DFS algoritmi har bir yo'lni chuqur o'rganadi, ammo bu yo'llar bir-biriga teng bo‘lmasligi mumkin. Agar maqsad eng qisqa yo'lni topish bo'lsa, DFSning eng yaxshi tanlov bo'lmasligi mumkin, chunki u eng qisqa yo'lni topishdan ko‘ra, eng chuqur yo‘llarni o‘rganishni afzal ko‘radi. DFSda yo‘llar birin-ketin, chuqur tekshiriladi, lekin eng qisqa yo‘lni topishda samarali bo‘la olmaydi. Bu holatda, kengaytirilgan izlash algoritmlari, masalan, BFS (Breadth-First Search) yoki DFS algoritmi, eng qisqa yo‘lni topishda yaxshiroq ishlashi mumkin. Shuningdek, DFS algoritmi ba'zida keraksiz takroriy tekshiruvlar olib kelishi mumkin, chunki u har bir tugunni yangi yo‘llar orqali chuqurroq o‘rganadi. Bu esa vaqtni isrof qilishga olib keladi, ayniqsa, katta va murakkab graflarda. DFSning boshqa bir kamchiligi shundan iboratki, agar grafda tsikllar bo‘lsa, algoritm qayta-qayta ayni tugunni tekshirishni boshlashi mumkin. Agar grafikda tsikl mavjud bo‘lsa, DFS bu tsiklni aniqlamasa, grafni to‘liq o‘rganishda xatolik yuzaga kelishi mumkin. DFSda bu kabi holatlarni oldini olish uchun qo‘shimcha tekshiruvlar yoki tugunlar ro‘yxatini ishlatish zarur, lekin bu uning soddaligini kamaytiradi va qo‘shimcha resurslarni talab qiladi. DFSni ishlatishda, grafdagi tugunlarning bir-biriga bog'lanishini yoki tsiklni oldini olishni nazorat qilish uchun qo‘shimcha mexanizmlar zarur bo‘ladi. Bu holatda, algoritmning to‘liq va samarali ishlashini ta'minlash uchun ba'zi optimallashtirishlar talab qilinadi. DFS algoritmining implementatsiyasida yirik graflarda yuzaga keladigan xatoliklar, kamchiliklar va resurslar bilan bog‘liq muammolarni hal qilish uchun ba'zan algoritmni optimallashtirish zarur. Masalan, ba'zi hollarda DFSni iterativ ravishda amalga oshirish, grafni kichik qismlarga bo'lish yoki eng qisqa yo‘lni topish uchun boshqa algoritmlarni ishlatish kerak bo‘lishi mumkin. DFS algoritmining eng katta afzalliklaridan biri bu uning oddiyligi va ko‘p maqsadlarda qo‘llanilishi, lekin uning kamchiliklari ham mavjud. Bu kamchiliklar DFSning to‘liq ishlashini ta'minlash uchun qo‘shimcha chora-tadbirlarni talab qiladi. DFSni ishlatishda uning afzalliklari va kamchiliklarini hisobga olib, eng to‘g‘ri yondashuvni tanlash muhimdir. DFSning samaradorligini oshirish uchun ba'zi optimallashtirishlar va qo‘shimcha mexanizmlar ishlatilishi mumkin, bu esa uning ishlashini sezilarli darajada yaxshilaydi.

Masala: Berilgan yo'naltirilgan grafda tsikl mavjudligini aniqlash va uni chiqarish. Grafda tsikl bo'lsa, bu tsiklni topish va ekranga chiqarish kerak. Misolda grafdagi har bir

tugundan boshlanib, uning qo'shni tugunlarini rekursiv tarzda DFS yordamida tekshirib chiqamiz.

Masala Misoli:

Graf (yo'naltirilgan):

0 -> 1 -> 2 -> 0

|

v

3 -> 4

Bu grafda tsikl mavjud: (0 -> 1 -> 2 -> 0).

```
using System;
using System.Collections.Generic;

class Graph
{
    private int vertices;
    private List<int>[] adjList;
    public Graph(int v)
    {
        vertices = v;
        adjList = new List<int>[v];
        for (int i = 0; i < v; i++)
        {
            adjList[i] = new List<int>();
        }
    }
    public void AddEdge(int v, int w)
    {
        adjList[v].Add(w);
    }
    private bool DFS(int v, bool[] visited, bool[] recStack, List<int> cycle)
    {
        if (recStack[v])
        {
            cycle.Add(v);
            return true;
        }
        if (visited[v]) return false;
```

```
        visited[v] = true;
        recStack[v] = true;
        foreach (int neighbor in adjList[v])
        {
            if (DFS(neighbor, visited, recStack, cycle))
            {
                if (cycle.Contains(v) == false)
                    cycle.Add(v);
                return true;
            }
        }
        recStack[v] = false;
        return false;
    }
    public List<int> FindCycle()
    {
        bool[] visited = new bool[vertices];
        bool[] recStack = new bool[vertices];
        List<int> cycle = new List<int>();
        for (int i = 0; i < vertices; i++)
        {
            if (!visited[i] && DFS(i, visited, recStack, cycle))
            {
                cycle.Reverse();
                return cycle;
            }
        }

        return null;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Graph g = new Graph(5);
```

```
            g.AddEdge(0, 1);
            g.AddEdge(1, 2);
            g.AddEdge(2, 0);
            g.AddEdge(3, 4);
            List<int> cycle = g.FindCycle();

            if (cycle != null)
            {
                Console.WriteLine("Grafda tsikl mavjud: ");
                foreach (int v in cycle)
                {
                    Console.Write(v + " ");
                }
            }
            else
            {
                Console.WriteLine("Grafda tsikl mavjud emas.");
            }
        }
    }
```

Natija:

Grafda tsikl mavjud:
0 1 2 0

FOYDALANILGAN ADABIYOTLAR:

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
2. Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.). Addison-Wesley.
3. Kleinberg, J., & Tardos, E. (2006). *Algorithm Design*. Pearson Education.
4. Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley.
5. Tarjan, R. E. (1972). Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2), 146-160.
6. Knuth, D. E. (1973). *The Art of Computer Programming, Volume 1: Fundamental Algorithms* (3rd ed.). Addison-Wesley.
7. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

8. Strogatz, S. H. (2001). *Sync: The Emerging Science of Spontaneous Order*. Hyperion.

9. Raxmonjonovich, F. S. (2024). MA'LUMOTLARNI SIQISHDA BITLI ALGORITMLARDAN FOYDALANISH. Modern education and development, 15(5), 320-328.

10. Маматкулов, Б. М., & Тураев, Б. Ш. (2023). *ПРОБЛЕМЫ МУЖСКОГО БЕСПЛОДИЯ В УЗБЕКИСТАНЕ* (Doctoral dissertation).

11. Mamatqulov, B., Tolipova, G., & To‘Rayev, B. (2023). VACCINATION AND CONTRAINDICATIONS BASED ON THE NATIONAL IMMUNIZATION CALENDAR AND THEIR BASIS. *Science and innovation*, *2*(D11), 316-321.

12. Mamatqulov, B., To‘Rayev, B., Ochilova, G., & Tolipova, G. (2023). REGULATORY LEGAL BASES OF MEDICAL AND SOCIO-HYGIENIC MEASURES AIMED AT MAINTAINING REPRODUCTIVE HEALTH IN THE REPUBLIC OF UZBEKISTAN. *Science and innovation*, *2*(D11), 384-394.

13. Маматкулов, Б., & Тураев, Б. (2022). Ўзбекистон Республикасида демографик ҳолат, фарзандсиз оилаларнинг тарқалганлиги ва бунда эркаклар бепуштлигининг ўрни.

14. Raxmonjonovich, F. S. (2024). AXBOROTLARNI SHIFRLASHDA MATEMATIK ALGORITMLARDAN FOYDALANISH. Modern education and development, 15(5), 338-344.

15. Raxmonjonovich, F. S. (2024). BIR SHAHARDAN BOSHQASIGA YUK YETKAZIB BERISHDA ENG OPTIMAL VA KAM XARAJAT SARFLANADIGAN YO’LNI TOPISHDA BELLMAN-FORD ALGORITMIDAN FOYDALANISH. Ta'lim innovatsiyasi va integratsiyasi, 34(2), 72-78.

16. Raxmonjonovich, F. S. (2024). KOMPYUTER TARMOQLARI SOHASIDA BITLI ALGORITMLAR. Modern education and development, 15(4), 50-59.

17. Raxmonjonovich, F. S., & Xurshidbek o‘g‘li, A. O. (2024). FORD-BELMAN ALGORITMI. Modern education and development, 15(4), 60-65.

18. Raxmonjonovich, F. S. (2024). IJTIMOIY TARMOQLAR TAHLILIDA BFS ALGORITMLARI. ОБРАЗОВАНИЕ НАУКА И ИННОВАЦИОННЫЕ ИДЕИ В МИРЕ, 58(7), 20-26.

19. Raxmonjonovich, F. S. (2024). DINAMIK DASTURLASH VA TARMOQ OQIMIDA FORD-BELMAN ALGORITMIDAN FOYDALANISH. ОБРАЗОВАНИЕ НАУКА И ИННОВАЦИОННЫЕ ИДЕИ В МИРЕ, 58(7), 13-19.

20. Raxmonjonovich, F. S. (2024). GRAFLARDA FLOYD-WARSHALL ALGORITMINING AHAMIYATI. ОБРАЗОВАНИЕ НАУКА И ИННОВАЦИОННЫЕ ИДЕИ В МИРЕ, 58(7), 6-12.

21. Raxmonjonovich, F. S., & Hamdamjon o'g'li, A. S. (2024). HISOBLASH MATEMATIKASI VA SONLI ANALIZ SOXASIDA DIFFERENSIAL TENGLAMALARNI YECHISHDA MATEMATIK ALGORITMLARNING AHAMIYATI. TADQIQOTLAR. UZ, 51(2), 37-44.

22. Raxmonjonovich, F. S. (2024). ARIFMETIK VA GEOMETRIK PROGRESSIYAGA OID MASALALARNING MATEMATIK ALGORITMLARI YORDAMIDA YECHISH. Лучшие интеллектуальные исследования, 34(1), 142-152.

23. Raxmonjonovich, F. S. (2024). XOFMAN KODLASH TIZIMI: AVIATSIYA VA PARVOZ MA'LUMOTLARINI SIQISHNING INNOVATSION YONDASHUVI. Лучшие интеллектуальные исследования, 34(1), 153-160.

24. Raxmonjonovich, F. S., & Botirali o'g'li, T. M. (2024). KAN ALGORITMINI GRAFLARDA QO'LLANILISHI. TADQIQOTLAR. UZ, 51(2), 27-36.

25. Raxmonjonovich, F. S. (2024). ROBOTOTEXNIKA SOHASIDA GEOMETRIK ALGORITMLARNING O'RNI. Лучшие интеллектуальные исследования, 34(1), 134-141.

26. Raxmonjonovich, F. S. (2024). MINIMAL BOG'LANISH DARAXTINI TOPISHDA PRIM ALGORITMIDAN FOYDALANISH. YANGI O'ZBEKISTON, YANGI TADQIQOTLAR JURNALI, 1(3), 436-443.

27. Raxmonjonovich, F. S., & Kudratullo o'g, K. U. B. (2024). C# VA NET FRAMEWORK ORQALI ZAMONAVIY VA XAVFSIZ TARMOQ DASTURLARINI ISHLAB CHIQISH. International journal of scientific researchers (IJSR) INDEXING, 5(2), 351-356.

28. Raxmonjonovich, F. S., & Azizjon o'g'li, N. A. (2024). WORKING WITH DATE AND TIME IN MODERN PROGRAMMING LANGUAGES. International journal of scientific researchers (IJSR) INDEXING, 5(2), 296-300.

29. Mamatkulov, B., Turayev, B., & Urinova, U. (2022). Assessment of Risk Factors Affecting the Reproductive Health of Men Living in Uzbekistan.

30. Тураев, Б. Ш. (2022). СОЦИАЛЬНО-ГИГИЕНИЧЕСКИЕ ФАКТОРЫ РИСКА, ВЛИЯЮЩИЕ НА РЕПРОДУКТИВНОЕ ЗДОРОВЬЕ МУЖЧИН. *ББК 51.1 G54*, 30.

31. Turayev, B. (2022). *LIFESTYLE AND MALE REPRODUCTIVE HEALTH* (Doctoral dissertation).

32. Ramanova, D., Urazalieva, I., Ishmukhamedova, S., Turayev, B., & Shoyusupova, H. (2020). The importance of family and family values in the formation of a healthy lifestyle. *Systematic Reviews in Pharmacy*, *11*(12).

33. Farmonov, S., & Rustamova, N. (2024, May). SINFLASHNING METRIK ALGORITMLARI, YAQIN QO'SHNI USULI VA UNI UMUMLASHTIRISH HAMDA

ULARNI NEYRON TARMOQ TEXNOLOGIYALARIDA QO‘LLANILISHI. In Международная конференция академических наук (Vol. 3, No. 5, pp. 71-75).

34. Farmonov, S., & Ergashaliyeva, B. (2024). QAT'IYMAS NEYRON TO'RLAR: MAMDANI QAT'IYMAS MANTIQIY XULOSASI, SUGENO QAT'IYMAS MANTIQIY XULOSASI. Development and innovations in science, 3(5), 62-70.

35. Raxmonjonovich, F. S. (2024). KOMPYUTER GRAFIKASI VA O’YIN DASTURLASHDA JOHNSON ALGORITMINING AHAMIYATI. Ta'lim innovatsiyasi va integratsiyasi, 34(2), 145-151.

36. Farmonov, S. R., & qizi Oktamjonova, M. I. (2024, November). FLOYD–UORSHELL ALGORITMI. In International Conference on World Science and Resarch (Vol. 1, No. 3, pp. 32-42).

37. Raxmonjonovich, F. S. (2024). KAN ALGORITMI VA UNING AMALIY QO ‘LLANILISHLARI. Ta'lim innovatsiyasi va integratsiyasi, 34(2), 139-144.

38. Farmonov, S. R. (2024). BFS ALGORITIMI ORQALI TOPOLOGIK TARTIBNI ANIQLASH. ОБРАЗОВАНИЕ И НАУКА В XXI ВЕКЕ, (56-5).

39. Raxmonjonovich, F. S., & Saidahmad o’g’li, I. S. (2024). BFS ALGORITMI VA UNING XAVFSIZLIK SOHASIDAGI ROLI. SCIENTIFIC APPROACH TO THE MODERN EDUCATION SYSTEM, 3(31), 117-123.

40. Raxmonjonovich, F. S. (2024). SUN’IY INTELEKT VA MASHINANI O’QITISHDA MATEMATIK ALGORITMDAN FOYDALANISH. Modern education and development, 15(5), 107-116.