BFS ALGORITMINING MURAKKABLIGI VA UNI OPTIMALLASHTIRISH USULLARI

Farmonov Sherzodbek Rahmonjonovich
Farg’ona davlat universiteti amaliy matematika va informatika kafedrasi katta o’qituvchisi
farmonovsh@gmail.com
Ro‘ziyev Abdulloxon Yorqinbek o‘g‘li
Farg‘ona davlat unversiteti talabasi
roziyevabdulloxon8@gmail.com

Annotatsiya. *BFS algoritmi graf va daraxt tuzilmalarida kenglik bo‘yicha qidiruvni amalga oshirishda keng qo‘llaniladi. Ushbu maqola BFS algoritmining murakkabligi va samaradorligini oshirish usullari, xususan Parallel BFS, Bi-Directional BFS, Memory-Efficient BFS va Heuristic BFS kabi optimallashtirish yondashuvlarini tahlil qiladi. Algoritmni katta grafiklarda yanada samarali ishlatish imkoniyatlarini ko‘rsatish bilan birga, kelajakdagi ilmiy izlanishlar uchun istiqbolli yo‘nalishlar ham bayon etiladi.*

Kalit so‘zlar: *BFS, Breadth-First Search, graf algoritmi, optimallashtirish usullari, Parallel BFS, Bi-Directional BFS, Memory-Efficient BFS, Heuristic BFS, algoritm samaradorligi, ma’lumot tuzilmalari, grafik qidiruv, katta hajmdagi ma’lumotlar, xotira optimallashtirish, resurslardan samarali foydalanish, eng qisqa yo‘lni topish*

Annotation. *The BFS algorithm is widely used for breadth-first search in graph and tree structures. This paper analyzes the methods of increasing the complexity and efficiency of the BFS algorithm, particularly optimization approaches such as Parallel BFS, Bi-Directional BFS, Memory-Efficient BFS and Heuristic BFS. Along with showing the possibilities of more effective use of the algorithm in large graphs, promising directions for future scientific research are also described.*

Keywords: *BFS, Breadth-First Search, graph algorithm, optimization methods, Parallel BFS, Bi-Directional BFS, Memory-Efficient BFS, Heuristic BFS, algorithm efficiency, data structures, graph search, big data , memory optimization, efficient use of resources, finding the shortest path*

Аннотация. *Алгоритм BFS широко используется для поиска в ширину в графовых и древовидных структурах. В этой статье анализируются методы повышения сложности и эффективности алгоритма BFS, в частности подходы к оптимизации,*

такие как параллельный BFS, двунаправленный BFS, эффективный с точки зрения памяти BFS и эвристический BFS. Наряду с показом возможностей более эффективного использования алгоритма на больших графах также описаны перспективные направления будущих научных исследований.

Ключевые слова: *BFS, поиск в ширину, графовый алгоритм, методы оптимизации, параллельный BFS, двунаправленный BFS, эффективное использование памяти BFS, эвристический BFS, эффективность алгоритма, структуры данных, поиск по графу, большие данные, оптимизация памяти, эффективное использование ресурсов. , нахождение кратчайшего пути*

BFS (Breadth-First Search) algoritmi graf va daraxt kabi ma’lumot tuzilmalarini qidirishda keng qo‘llaniladigan asosiy qidiruv algoritmlaridan biridir. BFSning asosiy ishlash prinsipi shundan iboratki, u qidiruvni berilgan tugundan boshlab, yaqin atrofdagi barcha tugunlarni birinchi bosqichda tekshiradi va shu tarzda qidiruv chuqurligini bosqichma-bosqich kengaytiradi. Algoritm "kenglik bo‘yicha qidirish" usulini qo‘llaydi, ya’ni u har bir tugunga yetib borganidan so‘ng, shu tugunga yaqin bo‘lgan boshqa tugunlarni ham izlaydi. Ushbu jarayon navbat (queue) yordamida boshqariladi, bunda algoritm yangi topilgan tugunlarni navbatga qo‘shib boradi va navbatning boshidan tugunlarni olib tekshirishni davom ettiradi. Shu sababli, BFS yaqin tugunlardan boshlab, qidiruv chegarasini kengaytirish orqali oxir-oqibat barcha tugunlarga yetib boradi. BFS algoritmining eng muhim xususiyatlaridan biri shundaki, u og‘irliksiz grafda eng qisqa yo‘lni topish imkonini beradi. Bu algoritmda tugunlar orasidagi masofa tugunning darajasiga qarab belgilanganligi sababli, u eng qisqa masofani kafolatlaydi.

BFS algoritmi turli ma’lumot tuzilmalarida keng qo‘llaniladi, eng ko‘p ishlatiladiganlar esa graf va daraxtlar hisoblanadi. Daraxtlar uchun BFS, odatda, har bir darajani o‘z navbatida tekshirish orqali ishlaydi, bunda tugunlar daraxtning ildizidan boshlanadi va har bir tugunning farzandlariga birinchi navbatda murojaat qilinadi. Daraxtning har bir darajasi to‘liq tekshirilgandan so‘ng keyingi darajaga o‘tiladi, bu esa qidiruv jarayonini tizimli ravishda amalga oshirishga imkon beradi. Graf strukturasida esa BFS qo‘shni tugunlar orasida bog‘lanishni tekshiradi va ularning qidiruv tartibida qaytarilishining oldini olish uchun "ko‘rilgan" tugunlar ro‘yxatini yuritadi. Graf uchun BFS algoritmi siklli tuzilmalar va bir nechta tugunlarga ega grafiklarda samarali ishlaydi. Ayniqsa, bu algoritm grafda bog‘liqlikni aniqlash uchun foydali bo‘lib, bir komponentdan ikkinchisiga yetib borish mumkinligini yoki grafikning turli qismlari orasidagi aloqalarni tahlil qilishda yordam beradi.

BFS algoritmi turli xil masalalarni yechishda ham juda qo‘lli vositadir. Masalan, u graf yoki xaritada eng qisqa yo‘lni topishda keng qo‘llaniladi. Bu, ayniqsa, yo‘l topish va marshrut belgilash kabi ilovalarda katta ahamiyatga ega. Masalan, shahardagi yo‘l tizimi, transport tarmog‘i, yoki internetdagi ma’lumot uzatish yo‘llari kabi grafiklarda foydalanuvchilar orasidagi masofani aniqlashda eng qisqa yo‘lni topishda BFSning samaradorligi ko‘rinadi. Bundan tashqari, BFS algoritmi bog‘liqlik komponentlarini topish masalasida ham qo‘llaniladi. Ma’lumotlar bazalarida yoki ijtimoiy tarmoqlarda bog‘langan tuzilmalarni tahlil qilishda, ya’ni foydalanuvchilar orasidagi bog‘liqlikni aniqlashda BFS yordamida bir tugundan boshqasiga o‘tish mumkin yoki yo‘qligini tekshirish oson bo‘ladi. Shu bilan birga, turli xil o‘yinlar va sun’iy intellekt sohalarida ham BFSning qo‘llanilishi keng tarqalgan. Masalan, labirint yechishda, imkoniyatlar daraxtidan optimal yo‘lni topishda yoki turli vaziyatlarda ko‘chish yo‘llarini qidirishda BFS algoritmi eng mos yechim topishda yordam beradi.

BFS algoritmi nafaqat masofalarni o‘lchash yoki bog‘liqlikni aniqlash, balki tahliliy va topologik masalalar uchun ham qulaydir. Ko‘pincha, grafikdagi har bir tugun va ularning yaqin aloqalarini birma-bir ko‘rish uchun BFS samarali ishlaydi. Shu sababli, katta miqyosdagi grafik ma’lumotlarini boshqarishda BFS algoritmi samarali vositalardan biri sifatida tan olinadi va ma’lumot tuzilmalari bilan ishlashda sezilarli darajada foydali hisoblanadi.

BFS algoritmining murakkabligi va amalga oshirish

BFS (Breadth-First Search) algoritmining murakkabligi va uni amalga oshirish usuli uning samaradorligini belgilovchi asosiy omillardandir. Vaqt murakkabligi nuqtai nazaridan, BFS algoritmi har bir tugunga bir marta murojaat qilgani va barcha bog‘lanishlarni ko‘rib chiqqani uchun O(V+E) (bu yerda V – tugunlar soni, E – qirralar soni) murakkablikka ega. Har bir tugun va qirra faqat bir marta tekshiriladi, chunki har bir qirra orqali borish va qaytish yo‘li bilan ikki tomonlama harakatlanilmaydi. Bu esa algoritmning vaqt samaradorligini oshiradi va uni katta grafiklar uchun ham samarali qiladi. Xotira murakkabligi esa O(V) ga teng, chunki algoritm qidirish davomida ko‘rilgan tugunlarni eslab qolish uchun qo‘shimcha joy talab qiladi. Shuningdek, tugunlar va qirralarni o‘zaro bog‘lashda navbat (queue)dan foydalaniladi, bu esa xotiradan optimal foydalanish imkonini beradi.

BFS algoritmini amalga oshirishda navbat asosiy rol o‘ynaydi. Algoritm biror bir tugundan boshlanadi va dastlabki tugunni navbatga qo‘yadi. Keyin, navbatning boshidagi tugun olib tashlanib, unga yaqin tugunlar ketma-ket ko‘rib chiqiladi. Har bir tugunning qo‘shnilari navbatga qo‘shiladi, va ular ko‘rilgan deb belgilanishi bilan takroriy ko‘rib chiqilishdan saqlanadi. Bu jarayon navbat bo‘shab qolmaguncha davom etadi, ya’ni barcha

tugunlar tekshirib chiqilgach algoritm to‘xtaydi. Shunday qilib, har bir tugun ketma-ket ravishda va o‘z yaqinlariga bog‘lanish ketma-ketligiga qarab tekshiriladi. Shu usul orqali grafdagi tugunlar kenglik bo‘yicha tekshiriladi va har bir tugun o‘z darajasiga qarab ketma-ket o‘rganiladi. Navbatdan foydalanish, algoritmni boshqarish va unga tartib berish imkonini beradi, bu esa katta grafiklar ustida ishlashda yanada samarali hisoblanadi.

BFS algoritmi odatda og‘irliksiz grafda eng qisqa yo‘lni topish uchun ishlatiladi. Og‘irliksiz grafda har bir qirraning qiymati bir xil deb qaraladi, shuning uchun BFS qidiruvi davomida qirralar soniga qarab eng qisqa yo‘lni kafolatlaydi. Masalan, yo‘l topish masalasida BFS algoritmi boshlang‘ich tugundan maqsad tugungacha bo‘lgan minimal masofani topadi, chunki u qidiruvda kenglik bo‘yicha harakatlanib, qirralar bo‘ylab bir xil masofani qamrab oladi. BFS boshqa turdagi graf strukturalarida ham samarali ishlaydi. Masalan, bog‘langan komponentlarni topishda, BFS grafikdagi barcha bog‘langan qismlarni tekshirish uchun foydalaniladi. Bu shuni anglatadiki, BFS algoritmi, grafikning har bir tuguniga yetib borish orqali uni bo‘laklarga ajratmasdan, butun grafikda bog‘liqlikni tekshirishi mumkin.

Bundan tashqari, BFS algoritmi keng qo‘llanilishi mumkin bo‘lgan boshqa sharoitlarda ham ishlatiladi. Masalan, ijtimoiy tarmoqlar yoki turli xil transport tizimlari kabi real dunyo grafiklarida foydalanuvchilar yoki stansiyalar orasidagi bog‘liqlikni tahlil qilishda qo‘llaniladi. Shu bilan birga, ayrim tarmoqlar yoki bog‘lanish tuzilmalarida eng qisqa yo‘l yoki eng qisqa vaqt masofasini topishda ham BFSni qo‘llash mumkin. Shuningdek, o‘yin dasturlarida labirintdan chiqish, yo‘l tanlash va qidiruv vazifalarini bajarayotganda, qidiruv samaradorligini oshirishda kenglik bo‘yicha qidirish juda qulay. Shu tarzda, BFS algoritmi nafaqat nazariy grafik masalalarni yechishda, balki amaliy va kundalik dasturlash masalalarida ham keng foydalaniladi, chunki u ko‘plab grafik va tarmoq bilan bog‘liq muammolar uchun samarali va oddiy yechim hisoblanadi.

BFS algoritmini optimizatsiya qilish usullari

BFS algoritmini optimallashtirish turli yondashuvlarni talab qiladi, chunki grafiklar va katta hajmdagi ma’lumotlar bilan ishlashda tezlik va xotira samaradorligi muhim ahamiyatga ega. Parallel BFS usuli - katta hajmdagi ma’lumotlar bilan ishlashda samarali yechimlardan biridir. Bu usulda qidiruv jarayoni bir nechta yadroli protsessorlar yordamida parallel ravishda bajariladi. Bu esa, qidiruvning barcha bosqichlarini bir vaqtda bir nechta qismga bo‘lib bajarish imkonini beradi. Misol uchun, biror tugun o‘ziga yaqin tugunlarni tekshirayotgan bo‘lsa, shu bilan bir vaqtda boshqa tugun ham o‘z yaqinlarini tekshirishi mumkin. Bu usul grafikning katta qismlarini tezroq qamrab olish va barcha tugunlarni tekshirish jarayonini sezilarli darajada tezlashtiradi. Parallel BFS ko‘pincha keng miqyosli grafiklarni o‘rganishda yoki katta miqdordagi tarmoq trafikini kuzatishda ishlatiladi, chunki

u bir nechta jarayonlarni bir vaqtning o‘zida bajarish orqali samaradorlikni oshiradi va qidiruvni umumiy vaqt bo‘yicha qisqartiradi.

Bi-Directional BFS algoritmi esa qidiruvni boshlang‘ich va maqsad tugunlardan bir vaqtda boshlash orqali samaradorlikni oshiradi. Bu usulda qidiruv jarayoni boshlang‘ich tugundan va maqsad tugundan ikki tomonlama olib boriladi. Har ikki tomonning qidiruvlari bir-biriga yaqinlashgan sari, qidiruv oralig‘i qisqaradi, natijada jarayonni tezroq yakunlash mumkin. Bu usulning asosiy afzalligi shundaki, grafikdagi masofalar uzoq bo‘lganda yoki qirralar ko‘p bo‘lganda, umumiy qidiruv jarayonini ikki baravarga qisqartiradi. Bi-Directional BFS usuli yo‘l topishda, masalan, xaritada yoki katta transport tizimlarida optimal marshrutni topishda juda qo‘llaniladi. Bu usul orqali algoritm umumiy qidiruv oralig‘ini qisqartirgani sababli, resurslardan yanada unumli foydalanish va qidiruv tezligini oshirish imkonini beradi.

Memory-Efficient BFS usuli esa grafdagi tugunlarni xotirada saqlashda samaradorlikka erishishga yordam beradi. Katta grafiklarda barcha tugunlarni va ularning yaqin aloqalarini xotirada saqlash katta xotira talab qiladi, shu bois murakkab ma’lumot tuzilmalarini qo‘llash orqali bu muammoni hal qilish mumkin. Masalan, har bir tugun haqida faqat eng zaruriy ma’lumotlarni saqlash yoki ko‘rilgan tugunlarni faqat kerakli miqdorda eslab qolish uchun maxsus strukturadan foydalanish xotira samaradorligini oshiradi. Bu usul ko‘p hollarda katta hajmdagi grafiklarda samarali bo‘ladi, chunki xotira talablarini kamaytirish orqali katta grafikni tekshirish imkonini beradi va qidiruvning umumiy samaradorligini oshiradi. Bu yondashuv ko‘pincha katta miqyosdagi ma’lumotlar bilan ishlash, masalan, ijtimoiy tarmoqlar yoki qidiruv tizimlarida qo‘llaniladi, chunki u xotiradan iqtisod qilish va jarayonni tezlatishga yordam beradi.

Masala

Har bir yo'lning uzunligi bitta (bir xil vaqt ketadi). Sizga shaharlar va yo'llar haqida quyidagi ma'lumotlar beriladi:

Shaharlar ro'yxati: Har bir shaharni boshqasi bilan bog'lovchi tarmoq (graf) mavjud.

Yo'llar ro'yxati: Shaharlar orasidagi yo'llar.

Boshlang'ich shahar: Qayerdan boshlash kerakligi berilgan.

Maqsad shahar: Borish kerak bo'lgan manzil.

Shart: Minimal qadamlar soni bilan maqsad shahariga yetadigan yo'lni toping.

Masalaning Kirish Ma'lumotlari:

Shaharlar: A, B, C, D, E, F

Yo'llar: (A-B), (A-C), (B-D), (C-D), (D-E), (E-F)

Boshlang'ich shahar: A

Maqsad shahar: F

Yechim:

Grafni yaratish: Shaharlarni va yo'llarni graf ko'rinishida ifodalash.

BFS algoritmi yordamida eng qisqa yo'lni topish.

Yechimning Algoritmi:

Qator (queue) dan foydalanib shaharlarni navbatma-navbat tekshirish.

Har bir shaharni ziyorat qilingan deb belgilash.

Boshlangan shahardan har bir keyingi shaharni navbatga qo'shish va ularning masofasini belgilash.

Maqsad shahariga yetganingizda eng qisqa yo'lni qaytarish.

C# Kod:

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

class BFSShortestPath
{
    static void Main(string[] args)
    {
        var graph = new Dictionary<string, List<string>>()
        {
            {"A", new List<string> { "B", "C" }},
            {"B", new List<string> { "A", "D" }},
            {"C", new List<string> { "A", "D" }},
            {"D", new List<string> { "B", "C", "E" }},
            {"E", new List<string> { "D", "F" }},
            {"F", new List<string> { "E" }}
        };

        string start = "A";
        string target = "F";

        var path = BFS(graph, start, target);

        if (path != null)
        {
            Console.WriteLine("Minimal yo'l: " + string.Join(" -> ", path));
        }
```

```
    else
    {
        Console.WriteLine("Yo'l topilmadi.");
    }
}

static List<string> BFS(Dictionary<string, List<string>> graph, string start, string target)
{
    var visited = new HashSet<string>();
    var queue = new Queue<List<string>>();


    queue.Enqueue(new List<string> { start });

    while (queue.Count > 0)
    {
        var path = queue.Dequeue();
        var city = path[^1];


        if (city == target)
            return path;

        if (!visited.Contains(city))
        {
            visited.Add(city);

            foreach (var neighbor in graph[city])
            {
                var newPath = new List<string>(path) { neighbor };
                queue.Enqueue(newPath);
            }
        }
    }

    return null;
```

```
        }
}
```

Dastur Natijasi:

Kirish ma'lumotlari:

Shaharlar: A, B, C, D, E, F

Yo'llar: (A-B), (A-C), (B-D), (C-D), (D-E), (E-F)

Boshlang'ich: A, Maqsad: F

Chiqish:

Minimal yo'l: A -> C -> D -> E -> F

Bu kod grafni aniqlash, BFS yordamida yo'lni qidirish va minimal masofani topish uchun ishlatiladi. Shaharlarni boshqa nomlarga o'zgartirish orqali boshqa masalalarni ham hal qilish mumkin.

BFS algoritmini optimallashtirish usullari uning umumiy samaradorligini sezilarli darajada oshiradi va algoritmni yanada keng qamrovli va resurslar samarador bo‘lgan holatlarda qo‘llash imkonini beradi. Parallel BFS usuli, masalan, bir nechta yadroli protsessorlar mavjud bo‘lgan tizimlarda katta hajmdagi ma’lumotlarni qidirishda juda qulay. Bu yondashuv katta grafiklarni qidirishda tezlikni oshirishga yordam beradi, chunki har bir yadroning o‘z ishini bir vaqtda bajarishi natijasida vaqt bo‘yicha samaradorlikka erishiladi. Bi-Directional BFS esa eng qisqa yo‘lni topish masalalarida samarali bo‘lib, masofa uzoq bo‘lgan grafiklarda yoki qirralar soni ko‘p bo‘lgan tizimlarda qidiruv jarayonini ikki tomondan olib borish orqali umumiy qidiruv oralig‘ini qisqartiradi. Shu bilan birga, Memory-Efficient BFS kabi yondashuvlar xotira resurslari cheklangan yoki grafning hajmi juda katta bo‘lganda juda qo‘l keladi. Xotiradan samarali foydalanish imkoniyatini oshirish orqali grafikdagi barcha ma’lumotlarni qamrab olish va natijalarni tezroq qaytarish imkonini beradi. Heuristic BFS usuli esa taxminlar asosida ishlagani uchun ma’lum qidiruv maqsadiga tez yetib borishni talab qiladigan real vaqtda ishlash tizimlari, masalan, yo‘l topish yoki o‘yinlar uchun juda qulay.

FOYDALANILGAN ADABIYOTLAR:

1. Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley Professional.
2. Knuth, D.E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley.
3. Aho, A.V., Hopcroft, J.E., & Ullman, J.D. (1974). The Design and Analysis of Computer Algorithms. Addison-Wesley.

4. Mehlhorn, K., & Sanders, P. (2008). Algorithms and Data Structures: The Basic Toolbox. Springer.
5. Even, S. (2011). Graph Algorithms. Cambridge University Press.
6. Banerjee, N., Chakraborty, S., & Raman, V. (2016, July). Improved space efficient algorithms for BFS, DFS and applications. In International Computing and Combinatorics Conference (pp. 119-130). Cham: Springer International Publishing.
7. Slota, G. M., Rajamanickam, S., & Madduri, K. (2014, May). BFS and coloring-based parallel algorithms for strongly connected components and related problems. In 2014 IEEE 28th International Parallel and Distributed Processing Symposium (pp. 550-559). IEEE.
8. Banerjee, N., Chakraborty, S., Raman, V., & Satti, S. R. (2018). Space efficient linear time algorithms for BFS, DFS and applications. Theory of Computing Systems, 62, 1736-1762.
9. Awerbuch, B., & Gallager, R. G. (1985, October). Distributed BFS algorithms. In 26th Annual Symposium on Foundations of Computer Science (sfcs 1985) (pp. 250-256). IEEE.
10. Kurant, M., Markopoulou, A., & Thiran, P. (2010, September). On the bias of BFS (breadth first search). In 2010 22nd International Teletraffic Congress (LTC 22) (pp. 1-8). IEEE.
11. Gazit, H., & Miller, G. L. (1988). An improved parallel algorithm that computes the BFS numbering of a directed graph. Information Processing Letters, 28(2), 61-65.
12. Parter, M., & Peleg, D. (2016). Sparse fault-tolerant BFS structures. ACM Transactions on Algorithms (TALG), 13(1), 1-24.