BOYER-MOORE ALGORITMI: MATNDA QIDIRUVNING TEZ VA SAMARALI USULI

Onarkulov Maksadjon Karimberdiyevich
Farg'ona davlat univarsiteti Amaliy matematika va informatika kafedrasi dotsenti (PhD)
maxmaqsad@gmail.com
Zokirjonova Xushnozaxon Ulug'bek qizi
Farg'ona Davlat Universiteti 2-kurs talabasi
xushnozaxonzokirjonova0@gmail.com

Annotatsiya. *Ushbu ilmiy maqolada Boyer-Moore algoritmining nazariy asoslari, evristik yondashuvlari va amaliy qo'llanilishi chuqur tahlil qilinadi. Algoritm Robert S. Boyer va J Strother Moore tomonidan 1977 yilda ishlab chiqilgan bo'lib, belgilar qatori (string) ichida naqsh (pattern) izlashda eng samarali algoritmlardan biri sifatida tan olingan. Ushbu algoritmning asosiy afzalligi — u klassik qidiruv algoritmlaridan farqli ravishda, matn bo'yicha solishtirishni o'ngdan chapga amalga oshiradi va har bir mos kelmaslik holatida naqshni bir belgidan ko'ra ko'proq siljitishga imkon beruvchi ikki kuchli evristikani — "bad character" (yomon belgi) va "good suffix" (yaxshi suffiks) yondashuvlarini qo'llaydi. Bad character evristikasi — naqshdagi har bir belgining matnda uchrashishi asosida naqshni qanchalik siljitishni aniqlashga yordam beradi. Good suffix evristikasi esa naqshdagi mos kelgan suffix'lar asosida siljitish masofasini hisoblab, ortiqcha solishtirishlarni kamaytiradi.Maqolada ushbu algoritmning ishlash prinsipi, har ikki evristikaning matematik modeli, dasturlashda implementatsiyasi (ayniqsa C# tilida) va real amaliyotdagi misollar orqali ko'rsatilgan. Shuningdek, algoritmning vaziyatga qarab o'zgaruvchi ishlash samaradorligi (eng yaxshi, o'rtacha va eng yomon holatdagi vaqt murakkabligi) tahlil qilinadi. Boyer-Moore algoritmi keng o'lchamdagi matnlarda yoki bir naqsh bo'yicha ko'p martalik qidiruvlar talab etiladigan vaziyatlarda ayniqsa samarali hisoblanadi. Maqola davomida*

algoritmning afzalliklari va cheklovlari ham yoritilgan bo‘lib, u boshqa mashhur algoritmlar (Knuth-Morris-Pratt, Rabin-Karp, va boshqalar) bilan funksional va samaradorlik jihatidan taqqoslanadi. Bu esa uni real hayotdagi qidiruv tizimlarida (masalan, matn muharrirlari, biologik ketma-ketlik qidiruvi, veb-izlovchilar) qo‘llash imkoniyatlarini aniqlab beradi.

Kalit So‘zlar *.Boyer-Moore algoritmi, Matnda qidiruv (string search), Belgilar qatori (character sequence), Naqshni tanib olish (pattern recognition), Naqshni topish (pattern matching), Qidiruv algoritmlari (search algorithms), Matnli ma’lumotlar bilan ishlash.*

Kirish. Klassik hisoblash algoritmlari orasida eng mashhurlari Knuth-Morris-Pratt (KMP), Rabin-Karp, va Boyer-Moore algoritmlaridir. Ulardan Boyer-Moore algoritmi o'zining yuqori darajasi, aniqligi va kuchi bilan ajralib turadi. barcha, u katta o'lchamdagi matnlar yoki ko'p martalik naqsh izlashi uchun eng maqbul tanlovdir. Boyer-Moore algoritmi Robert S. Boyer va J Strother Moore tomonidan 1977 yilda taklif qilingan bo'lib, u matn ichida naqshnida o'ngdan chapga qarab taqqoslash tajribaga asoslanadi. Bu hisoblash algoritmga optimallashtirish beradi. Boyer-Moore algoritmining asosli kuchli jihati - bu uning ikki evristik belgilari **:** bad character (yomon belgi) va good suffix (yaxshi suffiks). Uch evristikalar orqali algoritm naqshni har bir mos kelmas holatida bir belgiga emas, balki bir necha belgiga siljitishi mumkin. Bu esa algoritmlarni kamaytirib, ko'plab narsalarni o'z ichiga oladi. Bundan tashqari, Boyer-Moore algoritmi takrorlangan naqshlar **,** uzun matnlar , yoki oldindan naqshlar asosida ko'p martalik izlanishlar talab qilgan uchun juda foydali. Shu sababli, u kompilyatorlar, antivirus tizimlari, genetik ma'lumotlarni tahlil qilish , va natural tilni qayta ishlash (NLP) soha keng qo'shimcha. Ushbuner-Moore maqolasida Boyer-Moore algoritmining asosiy g'oyalari, evristikalari, amaliy ishlash mexanizmi, tahlili, ustunliklari va topilmalari yoritiladi. Hisoblash, algoritm mashhur hisoblash algoritmlari bilan taqqoslangan tahlili ham keltiriladi.

Algoritmning Asosiy G‘oyasi.

Boyer-Moore algoritmi — matnda naqsh (pattern) izlash muammosiga innovatsion va samarali yechim taklif qiluvchi algoritmdir. Uning asosiy konsepsiyasi klassik algoritmlardan farqli o‘laroq, taqqoslashni naqshning oxiridan boshlab (o‘ngdan chapga) amalga oshirishda yotadi. Bu yondashuv orqali har bir mos kelmaslik holatida naqshni matn bo‘yicha bir belgi bilan emas, balki bir necha belgilar orqaga surish imkonini beradi, ya’ni algoritm qidiruv jarayonini sakrashlar bilan tezlashtiradi.Bunday sakrash (shift) mexanizmi algoritmda ikki asosiy evristik yondashuv orqali amalga oshiriladi:

1. **Bad Character (Yomon Belgi) evristikasi** – agar taqqoslash vaqtida biror belgi mos kelmasa, algoritm bu belgining naqshdagi oxirgi pozitsiyasiga qarab naqshni siljitadi. Agar bu belgi naqshda mavjud bo‘lmasa, u holda naqshni to‘liq uzunligi bo‘yicha suradi.
2. **Good Suffix (Yaxshi Suffiks) evristikasi** – agar naqshning oxirgi qismi matn bilan mos kelib, lekin undan oldingi belgi mos kelmasa, bu evristika shu mos kelgan suffiks naqsh ichida yoki uning prefiksida boshqa joyda uchrash-uchrashmasligini tahlil qiladi va mos ravishda naqshni eng katta samarali sakrash bilan siljitadi.

Bu yondashuvlar yordamida Boyer-Moore algoritmi bir nechta afzalliklarga ega bo‘ladi:

Keraksiz taqqoslashlarni chetlab o‘tadi, ya’ni har bir belgini alohida tekshirish o‘rniga, butun segmentlar ustidan ishlaydi.

Ortiqcha resurs sarfini kamaytiradi, bu esa katta matnlar bilan ishlashda muhim ahamiyat kasb etadi.

Qidiruv tezligi real holatlarda deyarli chiziqli vaqt (linear time) bo‘yicha bajariladi, bu esa uni amaliy dasturlarda afzal qiladi.

Algoritm Ishlash Jarayoni

Boyer-Moore algoritmi ikki asosiy bosqichda ishlaydi: **tayyorlov (preprocessing)** va **qidiruv (searching)** bosqichlari. Har ikki bosqich birgalikda algoritmning samarali va tez ishlashini ta’minlaydi.

1. Tayyorlov bosqichi (Preprocessing stage)

Bad Character jadvali (yomon belgi)

- Har bir belgining naqshdagi **oxirgi uchrashgan indeksini** saqlaydi.
- Belgilar to‘plamiga (masalan, ASCII) asoslangan massiv yoki lug‘at shaklida tuziladi.
- Agar belgi naqshda mavjud bo‘lmasa, jadvalda -1 bilan belgilanishi mumkin.
- Bu jadval yordamida mos kelmagan belgi uchraganda naqshni qanchaga siljitish kerakligi aniqlanadi.
- **Good Suffix jadvali (yaxshi suffiks)**

- Naqsh ichidagi suffixlarning takrorlanishiga asoslanadi.
- Agar naqshning oxiridan boshlab bir qismi matn bilan mos kelsa, lekin undan oldingi belgi mos kelmasa, bu suffix naqsh ichida boshqa joyda mavjudmi, deb tekshiriladi.
- Mos kelgan suffix naqsh ichida yoki uning prefiksida uchrasa, naqsh mos ravishda oldinga siljitiladi.
- Good suffix jadvali, shuningdek, maksimal sakrashni aniqlashga yordam beradi.

Bu bosqichda har ikki jadval O(m) vaqt (m — naqsh uzunligi) ichida tuziladi, va bu xarajat algoritmning keyingi ishlashi uchun foydali investitsiya hisoblanadi.

2. Qidiruv bosqichi (Searching stage)

Bu bosqichda tayyorlangan jadval va naqsh asosida matn bo‘yicha qidiruv amalga oshiriladi. Qidiruv quyidagi tartibda bajariladi:

1. **Naqsh matn ustida joylashtiriladi** – dastlabki pozitsiyadan boshlanadi.
2. **Taqqoslash o‘ngdan chapga qarab bajariladi** – naqsh oxiridan boshlanib har bir belgi mos kelishi tekshiriladi.
3. **Agar barcha belgilar mos kelsa**, naqsh mos kelgan joy aniqlanadi va natijaga qo‘shiladi.
4. **Agar nomuvofiqlik yuz bersa**, quyidagi qoidalar asosida naqsh siljitiladi:

   Bad Character evristikasi orqali siljitish masofasi hisoblanadi.

   Good Suffix evristikasi orqali ham siljitish masofasi hisoblanadi.

   Ushbu ikki qiymatdan **eng kattasi** tanlanadi va naqsh shu miqdorda oldinga suriladi.

5. **Qidiruv** matn uzunligi tugaguncha yoki oxirgi siljitish chegarasigacha davom etadi.

Boyer-Moore algoritmiga misol.

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

class BoyerMoore
{
    private const int AlphabetSize = 256;

    // Bad character jadvalini tayyorlash
    private static int[] BuildBadCharacterTable(string pattern)
```

```
    {
        int[] badChar = new int[AlphabetSize];
        for (int i = 0; i < AlphabetSize; i++)
            badChar[i] = -1;

        for (int i = 0; i < pattern.Length; i++)
            badChar[(int)pattern[i]] = i;

        return badChar;
    }

    // Good suffix jadvalini tayyorlash
    private static int[] BuildGoodSuffixTable(string pattern)
    {
        int m = pattern.Length;
        int[] goodSuffix = new int[m];
        int[] suffixes = new int[m];

        // Suffix massivini tayyorlash
        suffixes[m - 1] = m;
        int g = m - 1, f = 0;

        for (int i = m - 2; i >= 0; i--)
        {
            if (i > g && suffixes[i + m - 1 - f] < i - g)
                suffixes[i] = suffixes[i + m - 1 - f];
            else
            {
                if (i < g)
                    g = i;
```

```
            f = i;
            while (g >= 0 && pattern[g] == pattern[g + m - 1 - f])
                g--;
            suffixes[i] = f - g;
        }
    }

    // Good suffix massivini yaratish
    for (int i = 0; i < m; i++)
        goodSuffix[i] = m;

    int j = 0;
    for (int i = m - 1; i >= 0; i--)
    {
        if (suffixes[i] == i + 1)
        {
            for (; j < m - 1 - i; j++)
                if (goodSuffix[j] == m)
                    goodSuffix[j] = m - 1 - i;
        }
    }

    for (int i = 0; i <= m - 2; i++)
        goodSuffix[m - 1 - suffixes[i]] = m - 1 - i;

    return goodSuffix;
}

// Boyer-Moore qidiruv algoritmi
public static List<int> Search(string text, string pattern)
```

```
    {
        List<int> result = new List<int>();
        int n = text.Length;
        int m = pattern.Length;

        if (m == 0 || n < m)
            return result;

        int[] badChar = BuildBadCharacterTable(pattern);
        int[] goodSuffix = BuildGoodSuffixTable(pattern);

        int s = 0; // siljitish bosqichi

        while (s <= (n - m))
        {
            int j = m - 1;

            while (j >= 0 && pattern[j] == text[s + j])
                j--;

            if (j < 0)
            {
                result.Add(s); // topildi
                s += goodSuffix[0];
            }
            else
            {
                int badCharShift = j - badChar[(int)text[s + j]];
                int goodSuffixShift = goodSuffix[j];
                s += Math.Max(1, Math.Max(badCharShift, goodSuffixShift));
```

```
        }
    }

    return result;
}

// Asosiy method (test uchun)
static void Main()
{
    string text = "HERE IS A SIMPLE EXAMPLE";
    string pattern = "EXAMPLE";

    var matches = Search(text, pattern);

    if (matches.Count == 0)
        Console.WriteLine("Naqsh topilmadi.");
    else
    {
        Console.WriteLine($"Naqsh {matches.Count} marta topildi:");
        foreach (var index in matches)
            Console.WriteLine($"Pozitsiya: {index}");
    }
}
```

Xulosa. Boyer-Moore algoritmi — belgilar qatori (string) asosida naqshni aniqlash muammosini hal qilishda eng samarali yondashuvlardan biridir. Bu algoritm naqshni matnda izlashda klassik yondashuvlardan farqli o‘laroq, belgilarni **o‘ngdan chapga** qarab solishtiradi va **bad character** hamda **good suffix** evristikalari yordamida naqshni bir nechta belgiga siljitib, **keraksiz**

tekshiruvlar sonini kamaytiradi. Bu esa algoritmni nafaqat nazariy jihatdan, balki amaliy holatlarda ham juda samarali qiladi. Boyer-Moore algoritmi ayniqsa:

- Katta hajmdagi matnlar bilan ishlaganda,
- Naqsh bir necha bor takroran qidirilayotgan holatlarda,
- Ma'lumotlar qatorini tezlik bilan tahlil qilish zarur bo‘lgan tizimlarda

eng afzal tanlovlardan biri bo‘lib xizmat qiladi. Unga xos bo‘lgan optimallashtirish strategiyalari algoritmga **lineerga yaqin ishlash tezligi**ni ta’minlaydi. Shu sababli Boyer-Moore algoritmi nafaqat **matn muharrirlari** yoki **qidiruv tizimlarida**, balki **biotexnologiya (masalan, DNK ketma-ketligini izlash)**, **kompilyatorlar** va **ma'lumotlarni indekslash tizimlari** kabi turli sohalarda keng qo‘llaniladi.Algoritmning ba’zi kamchiliklari (masalan, tayyorlov bosqichining murakkabligi) mavjud bo‘lsa-da, ularning ta’siri amaliyotda sezilarli emas, chunki bu bosqich bir marta bajarilib, ko‘p marotaba foydalanishga yo‘l ochadi.

Foydalanilgan adabiyotlar:

1. **Boyer, R. S., & Moore, J. S.** (1977). *A Fast String Searching Algorithm*. *Communications of the ACM*, **20**(10), 762–772. https://doi.org/10.1145/359842.359859

2. **Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C.** (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. ISBN: 978-0262033848

3. **Gusfield, D.** (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press. ISBN: 978-0521585194

4. **Knuth, D. E., Morris, J. H., & Pratt, V. R.** (1977). *Fast Pattern Matching in Strings*. *SIAM Journal on Computing*, **6**(2), 323–350. https://doi.org/10.1137/0206024

5. **Navarro, G.** (2001). *A guided tour to approximate string matching*. *ACM Computing Surveys*, **33**(1), 31–88. https://doi.org/10.1145/375360.375365

6. **Sedgewick, R., & Wayne, K.** (2011). *Algorithms* (4th ed.). Addison-Wesley. ISBN: 978-0321573513

7. **Crochemore, M., & Rytter, W.** (2002). *Jewels of Stringology: Text Algorithms*. World Scientific Publishing. ISBN: 978-9810249841

8. **Watson, B. W.** (1995). *A taxonomy of string matching algorithms*. Eindhoven University of Technology, Computing Science Notes.