আহো-কোরাসিক(Aho-Corasick) অ্যালগোরিদম

Posted on September 28, 2017 by Sudip Sarker

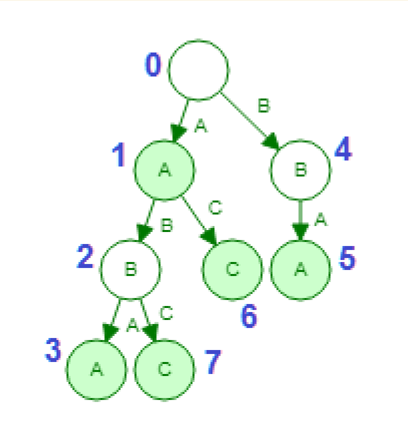
মনে আছে তো, আমার সেই কিপটে বন্ধুটাকে জব্দ করে বুফে খাওয়ার কথাটা! আহহ, সেই স্বাদ এখনও মুখে লেগে আছে! ☺ আর বন্ধুর করুণ মুখটা দেখলে আজও হাসি আটকে রাখতে পারি না। ☺ কিন্তু আমার সেই বন্ধুটাও কোনভাবেই ছাড়ার পাত্র নয়, সে এই টাকা আমার থেকে উঠিয়েই ছাড়বে। তাই এবার সে বাজি লাগল আরেকটা নতুন সমস্যা নিয়ে। প্রায় ১০০০০ শব্দ দিয়ে বলল, এগুলোর প্রত্যেকটা **‘Lord of the Rings’** বইয়ে কতবার করে আছে, সেটা বলতে হবে।

কিভাবে সম্ভব?! আমরা তো এর আগেই **KMP** এর মাধ্যমে টেক্সটে ১টা প্যাটার্ন কতবার আছে, সেটা বের করতে শিখে গেছি! যদি সেটা না শিখে থাকো, তাহলে এখান থেকে ঘুরে আসো।

মনে হতেই পারে, **KMP** জানা থাকলে এই কাজটা তো খুবই সহজ। কিন্তু এবার প্যাটার্ন কিন্তু ১টা নয়, ১০০০০টা। তার মানে আমাদের ১০০০০ বার **KMP** চালাতে হবে। বুঝতেই পারছো, সবগুলো শব্দ কতবার করে আছে, সেটা বের করতে আমাদের অনেক সময় লেগে যাবে।

এই কাজটাই সহজ করার জন্য আজকে আমরা শিখব **‘Aho-Corasick Algorithm’**. তবে এই অ্যালগোরিদম শেখার আগে ট্রাই সম্পর্কে জানা থাকা ভাল। তাই ট্রাই নিয়ে কোন আইডিয়া না থাকলে পুরোটা পড়ার আগে ট্রাই সম্পর্কে একটু জেনে আসো!

আচ্ছা, প্রথমে ধরে নেই, আমাদের টেক্সট হচ্ছে **T = abababacbabc**, আর প্যাটার্ন আছে ৫টা; **aba, ba, ac, a, abc**. প্রথমে আমরা এই ৫টা শব্দকে একটা ট্রাইতে রাখব। ট্রাইটা দেখতে হবে এরকমঃ

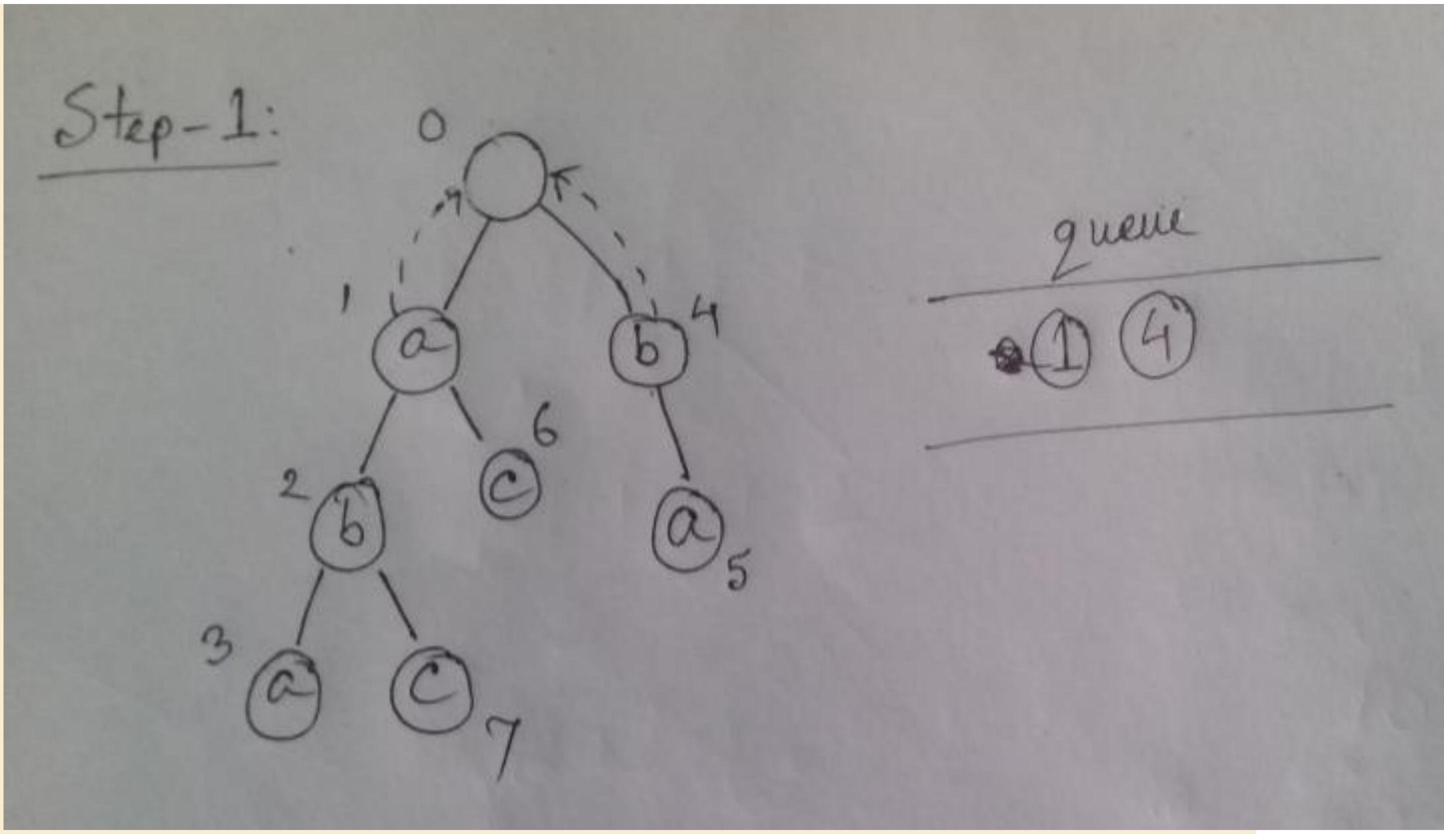


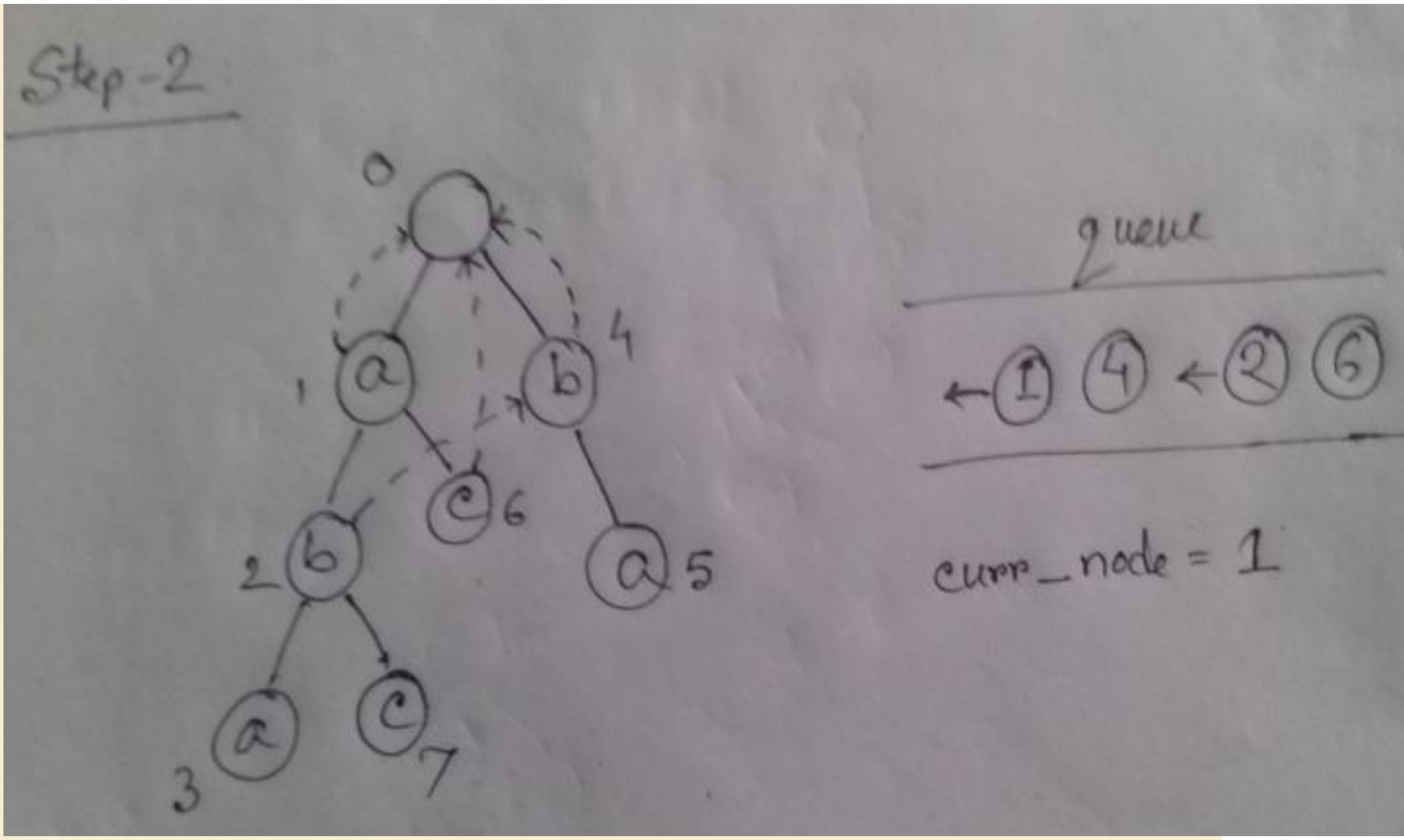
ট্রাইতে প্যাটার্নগুলো insert করার কোডটা দিয়েই দিলামঃ

```

void insert(string s)
{
    int u=0;
    for(int i = 0; i < s.size(); i++)
    {
        int c=s[i]-'a';
        if(!ch[u][c])
            ch[u][c]=ss++;;
        u=ch[u][c];
    }
    val[u]++;
    num[u]=0;
}

```

এরপর আমরা failure function নিয়ে কাজ করব। KMP তেও কিন্তু এরকম failure function ছিল, যেখানে কোন জায়গায় স্ট্রিং ম্যাচ না করলে আমরা কিছু ক্যারেক্টার স্কিপ করতে পারতাম। এখানেও ট্রাইতে কোথাও কোন ক্যারেক্টারের জন্যে এজ না থাকলে পরবর্তীতে কোন নোডে যাব, সেটার জন্যে একটা অ্যারে তৈরী করব। মাথার উপর দিয়ে যাচ্ছে? উদাহরণ দিয়ে দেখলেই বুঝতে পারবে।

এই ফাংশনটিতে আমাদের একটি queue থাকবে। যেখানে আমরা একে একে নোডগুলোকে রাখব, এবং সেগুলোকে নিয়ে কাজ করব। তো প্রথমে ০ এর 'child node' গুলোকে কিউতে ঢোকাব, এদের প্রত্যেকের 'fail node' হবে ০. এখন কিউয়ের প্রতি নোডের জন্যে আমার কাজ হবে, তার প্রত্যেকটা চাইল্ড নোডের জন্যে ফেইল নোড বের করা। 'current node' এর প্রত্যেকটি 'child node' এর 'closest ancestor' এর 'fail node' এর কোন 'child node' এ ওই সেইম ক্যারেক্টারের কোন নোড আছে কিনা সেটা চেক করতে হবে, থাকলে ওই নোডটাই হবে 'current node' এর জন্য 'fail node'. এভাবে খুঁজতে খুঁজতে যদি আমরা ০ পর্যন্ত পৌঁছে যাই এবং ০ এর 'child' এও ওই ক্যারেক্টার খুঁজে না পাই, তাহলে ০ হবে ওই চাইল্ড নোডের ফেইল নোড। কি, কঠিন লাগছে? আমরা এক এক করে দেখি। 'fail node' এর এজগুলোকে ডটেড লাইন দিয়ে দেখানো হয়েছে।


Step-1:
0
1
a
b
4
2
b
c
6
a
5
3
a
c
7
queue
1
4

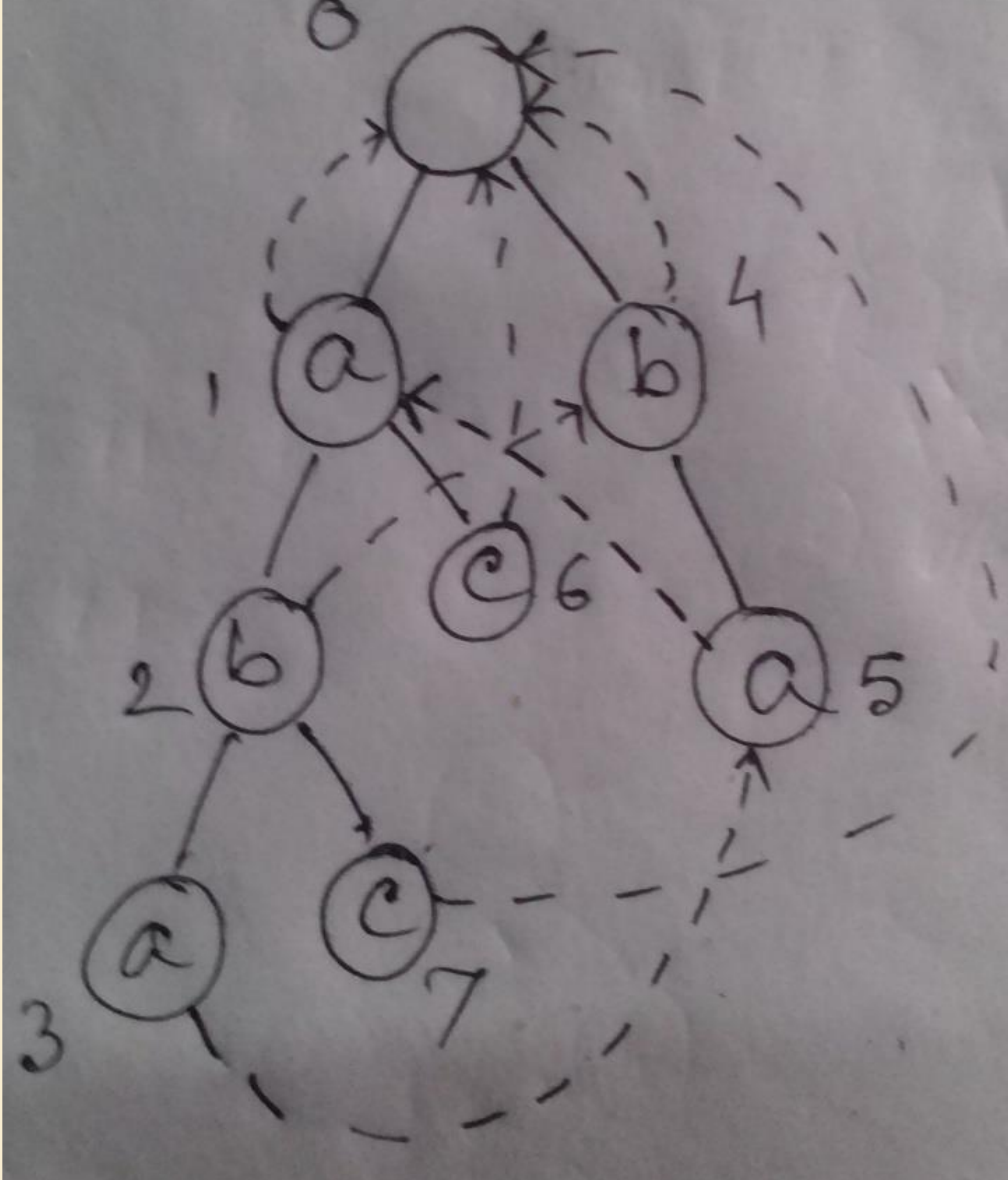


১নং স্টেপে ১,৪ নং নোডের ফেইল নোড করা হয়েছে ০ এবং এদের কিউতে পুশ করা হয়েছে। কিউ থেকে ১ নং নোড পপ করলাম ২নং স্টেপে এবং এখন কারেন্ট নোড হল ১। এর চাইল্ড নোড হচ্ছে ২, ৬। এদেরকেও কিউতে পুশ করে রাখলাম পরবর্তীতে এদের নিয়ে কাজ করার জন্য। এখন দেখো, ২ এর ‘closest ancestor’ হল ১, ১ এর ফেইল নোড ০. অর্থাৎ আমরা এখন ০ এর চাইল্ড নোডে দেখব ‘b’ ক্যারেক্টারটি আছে কিনা। খেয়াল করে দেখো, ০ এর চাইল্ড নোড ৪ এর ক্যারেক্টার ‘b’ অর্থাৎ আমরা ২ এর ফেইল নোড খুঁজে পেয়েছি। ২ থেকে ৪ এ ডটেড লাইন যোগ করলাম।

৬ এর জন্যেও সেইম কাজ করব। ৬ এর ‘closest ancestor’ হল ১, ১ এর ফেইল নোড ০. অর্থাৎ আমরা এখন ০ এর চাইল্ড নোডে দেখব ‘c’ ক্যারেক্টারটি আছে কিনা। যেহেতু c ক্যারেক্টারটি নেই এবং আমরা অলরেডি ০ নোডে পৌঁছে গেছি, সুতরাং ৬ এর জন্যে আমাদের ফেইল নোড হবে ০. ৬ থেকে ০ এ ডটেড লাইন যোগ করলাম।

আচ্ছা, ৩ নম্বর নোডের জন্য ফেইল নোড কোনটা হবে বলো তো? ৩ এর ‘closest ancestor’ হল ২, ২ এর ফেইল নোড ৪. অর্থাৎ আমরা এখন ৪ এর চাইল্ড নোডে দেখব ‘a’ ক্যারেক্টারটি আছে, সেটাই হবে আমাদের ফেইল নোড।

সবগুলো নোডের জন্যে এভাবে ফেইল নোডগুলো নিজে নিজে বের করে ফেলো। ☺ শেষ করে নিচের ছবিটার সাথে মিলিয়ে নাও, আহো কোরাসিক অ্যালগোরিদমের মূল কাজ কিন্তু এটাই।


0
a
1
b
4
c
6
b
2
a
5
a
3
c
7

ফেইলিওর ফাংশনের কোডটা এরকমঃ

```
void build_fail()
{
    queue q;
    int i;
    for(int i = 0; i < 26; i++)
    {
        if(ch[0][i])
            q.push(ch[0][i]);
    }

    int r,u,v;
    while(!q.empty())
    {
        r=q.front();
        q.pop();
        for(int c = 0; c < 26; c++)
        {
            u=ch[r][c];
            if(!u)
                continue;
            q.push(u);
            v=f[r];
            while(v && ch[v][c]==0)
                v=f[v];
            f[u]=ch[v][c];
        }
    }
}
```

এবার আমরা টেক্সটে এই প্যাটার্নগুলো খুঁজে বের করব। ট্রাইতে যেভাবে একটা স্ট্রিং খুঁজে বের করি, সেভাবেই খুঁজব, শুধু পার্থক্য হচ্ছে, কোন ক্যারেক্টার খুঁজে না পেলে কারেন্ট নোডের ফেইল নোডে চলে যাব এবং সেখান থেকে আবার খুঁজব। যেমন, টেক্সটের ‘aba’ খুঁজে পাওয়ার পরে আমাদের কারেন্ট নোড থাকবে ৩ আর ক্যারেক্টার হবে ‘b’ . ৩নং নোড থেকে যেহেতু ‘b’ এর কোন চাইল্ড নোড নেই, সেক্ষেত্রে ৩ এর ফেইল নোড ৫ এ যাব। ৫ থেকেও কোন চাইল্ড নোড নেই, তাই ৫ এর ফেইল নোড ১ এ যাব। ১ এর চাইল্ড নোডে ‘b’ ক্যারেক্টারটি আছে, অর্থাৎ আমরা ‘ab’ প্যাটার্নটি খুঁজে পেয়েছি টেক্সটের মধ্যে। এভাবে পুরো স্ট্রিংটাতেই সার্চ ফাংশন হাতে কলমে চালিয়ে দেখো। টেক্সটে প্যাটার্নগুলোর সংখ্যা নিচে দেয়া হলঃ

aba = 2, ba = 3, ac = 1, a = 4, abc = 1

সার্চ ফাংশনের কোডটা এরকমঃ

```
void search(string s)
{
    int j=0;
    for(int i = 0; i < s.size(); i++)
    {
        int c=s[i]-'a';
        while(j && ch[j][c]==0)
            j=f[j];
        j=ch[j][c];
        int temp=j;
        while(temp)
        {
            num[temp]++;
            temp=f[temp];
        }
    }
}
```

তো, কিপটে বন্ধু আবারও জব্দ, আর আমার আরেকবার ভূড়িভোজন হয়েই গেল! প্র্যাক্টিসের জন্যে নিচে কিছু প্রব্লেম দেয়া হলঃ

1427 – Substring Frequency (II) (LightOJ)

AHOCUR – Aho-Corasick Trie (SPOJ)