Aho–Corasick

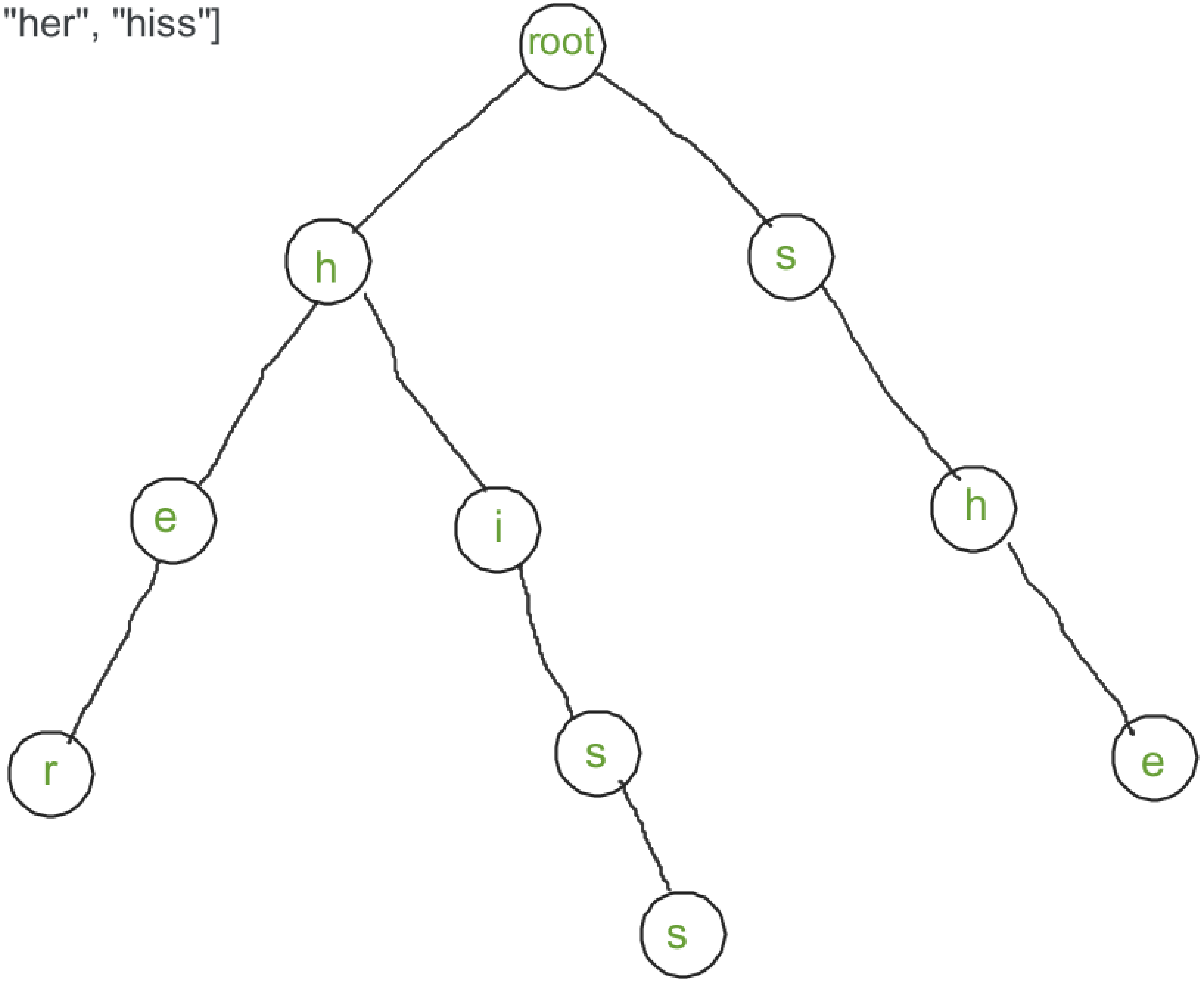
1 . Given a Text and some patterns
2 . For each pattern find how many times it occurs on the text

Algorithm

1 . Build a trie with the patterns
2 . Find failure link for each node using bfs
3 . Finally run the text on that trie

```
text         = "ahishers"
patterns = ["he", "his", "she", "her", "hiss"]
```

text = "ahishers"
patterns = ["he", "his", "she", "her", "hiss"]



What is failure link / suffix link ?

failure[node] = longest proper suffix which is also a prefix

What is failure link / suffix link ?

failure[node] = longest proper suffix which is also a prefix

= একটা node থেকে match করতে fail হলে,
অন্য কোন node থেকে আবার match করবো।

How to find failure link / suffix link ?

⇒ একটা BFS চালাবো trie এর উপর।

⇒ Base case: root এর fail root নিজেই

root এর child গুলার fail = root

How to find failure link / suffix link ?

x এর parent Px , Px এর fail $f[Px]$

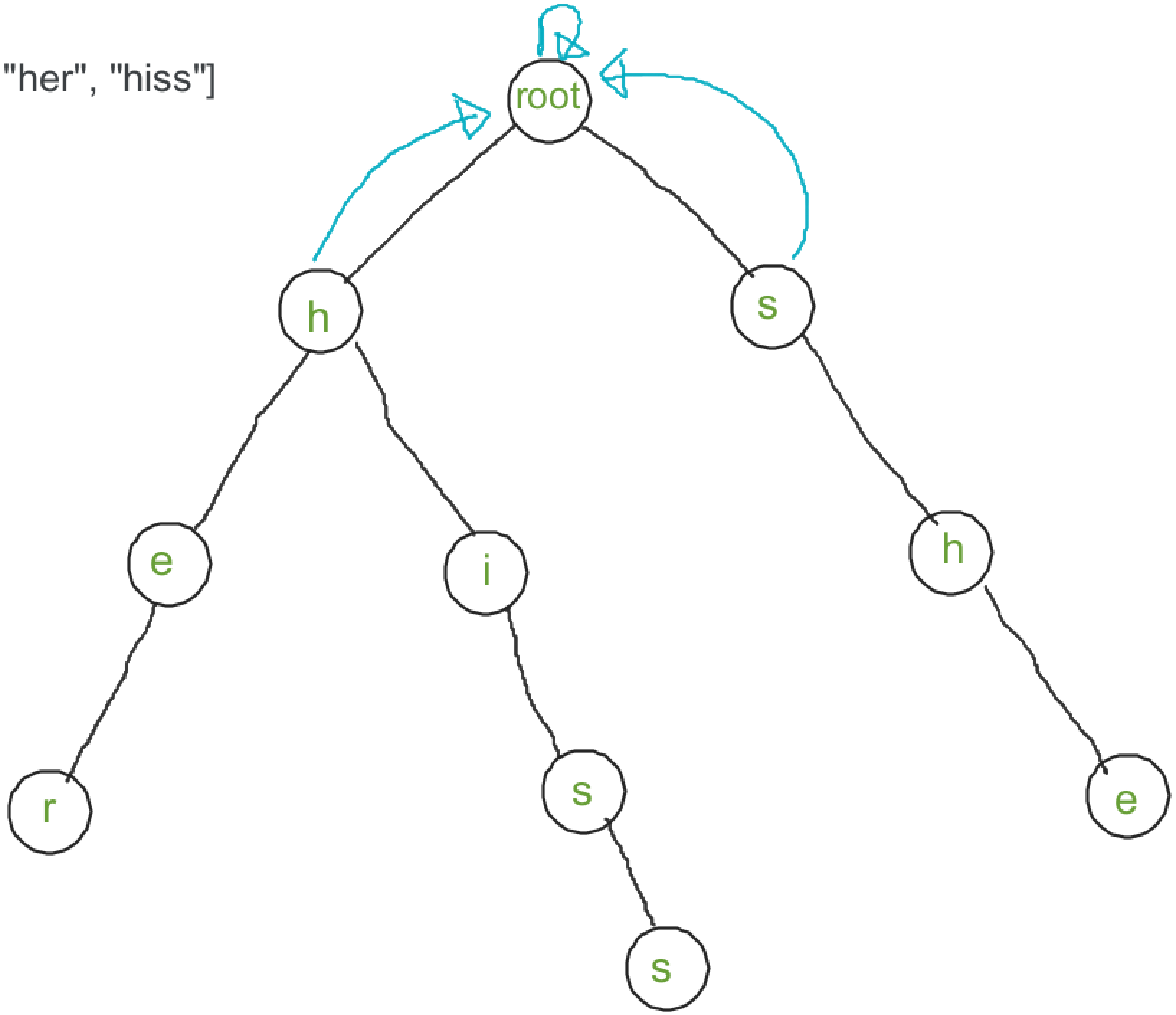
fail$[x]$ $\Rightarrow$ (i) x এর parent Px এ যাবো।

(ii) Px এর fail $f[Px]$ এ যাবো।

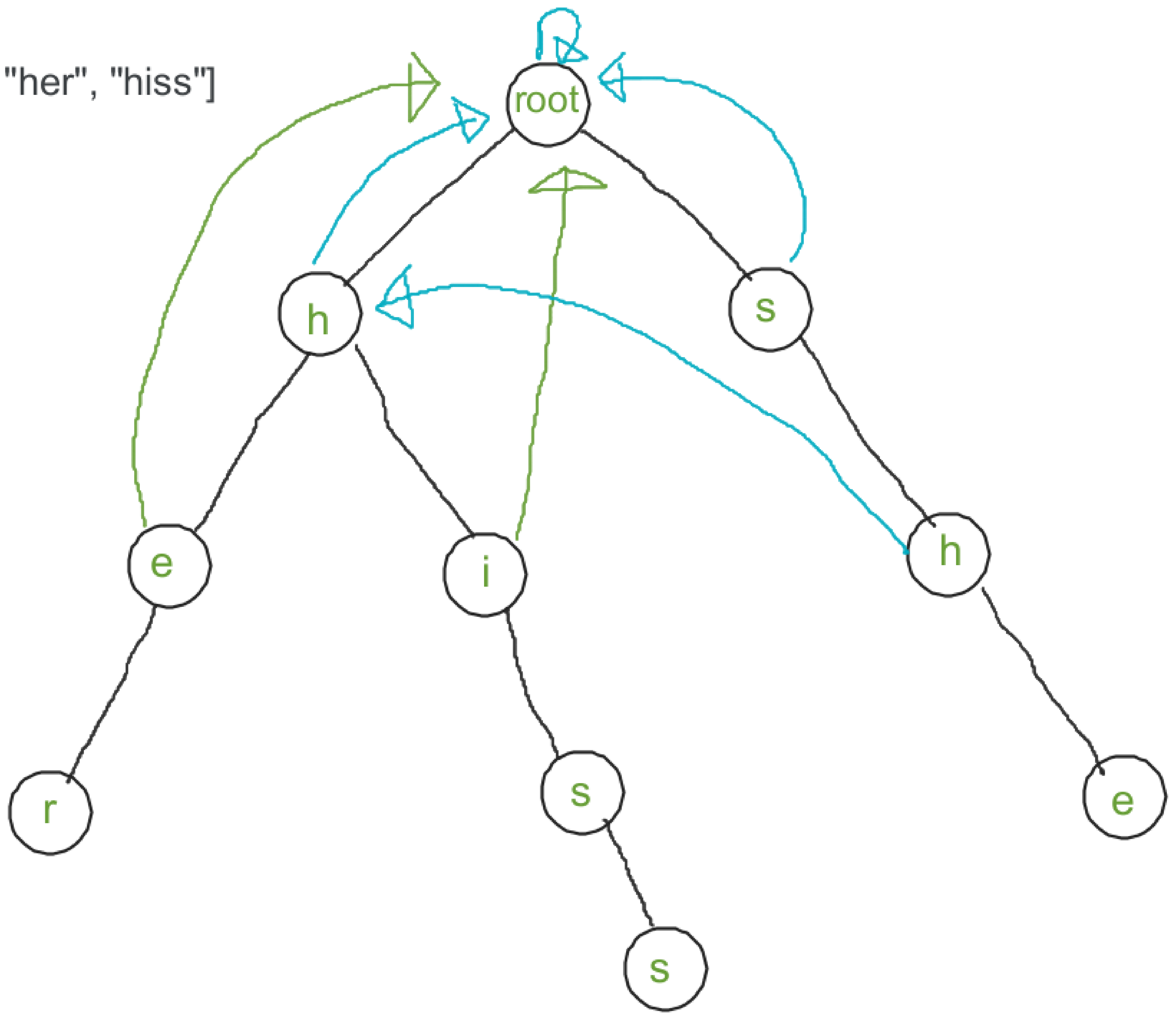
(iii) $f[Px]$ এর 'x' নামের child থাকলে ওটাই fail$[x]$

(iv) না থাকলে আবার $f[f[Px]]$ এ যাবো

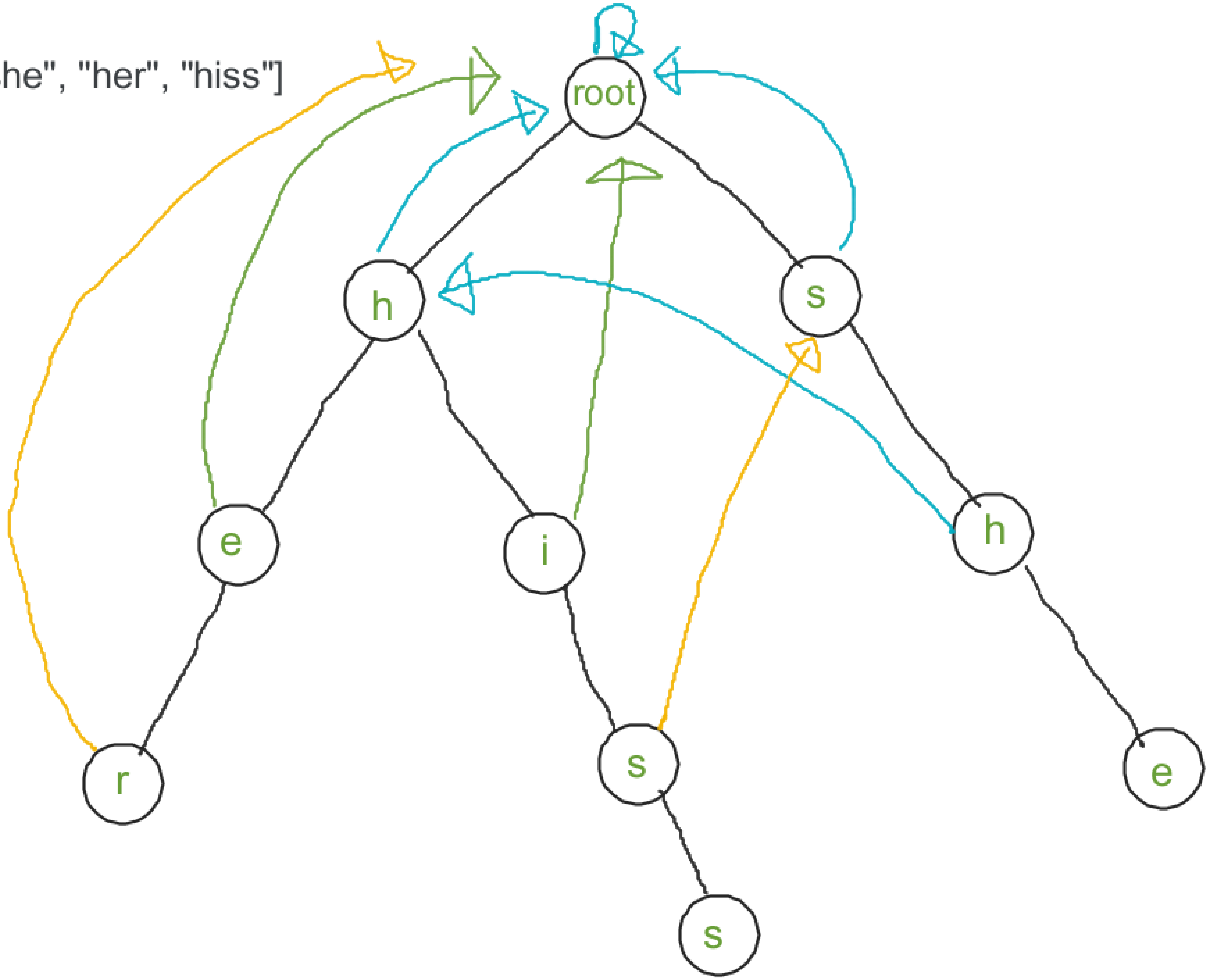
text = "ahishers"
patterns = ["he", "his", "she", "her", "hiss"]



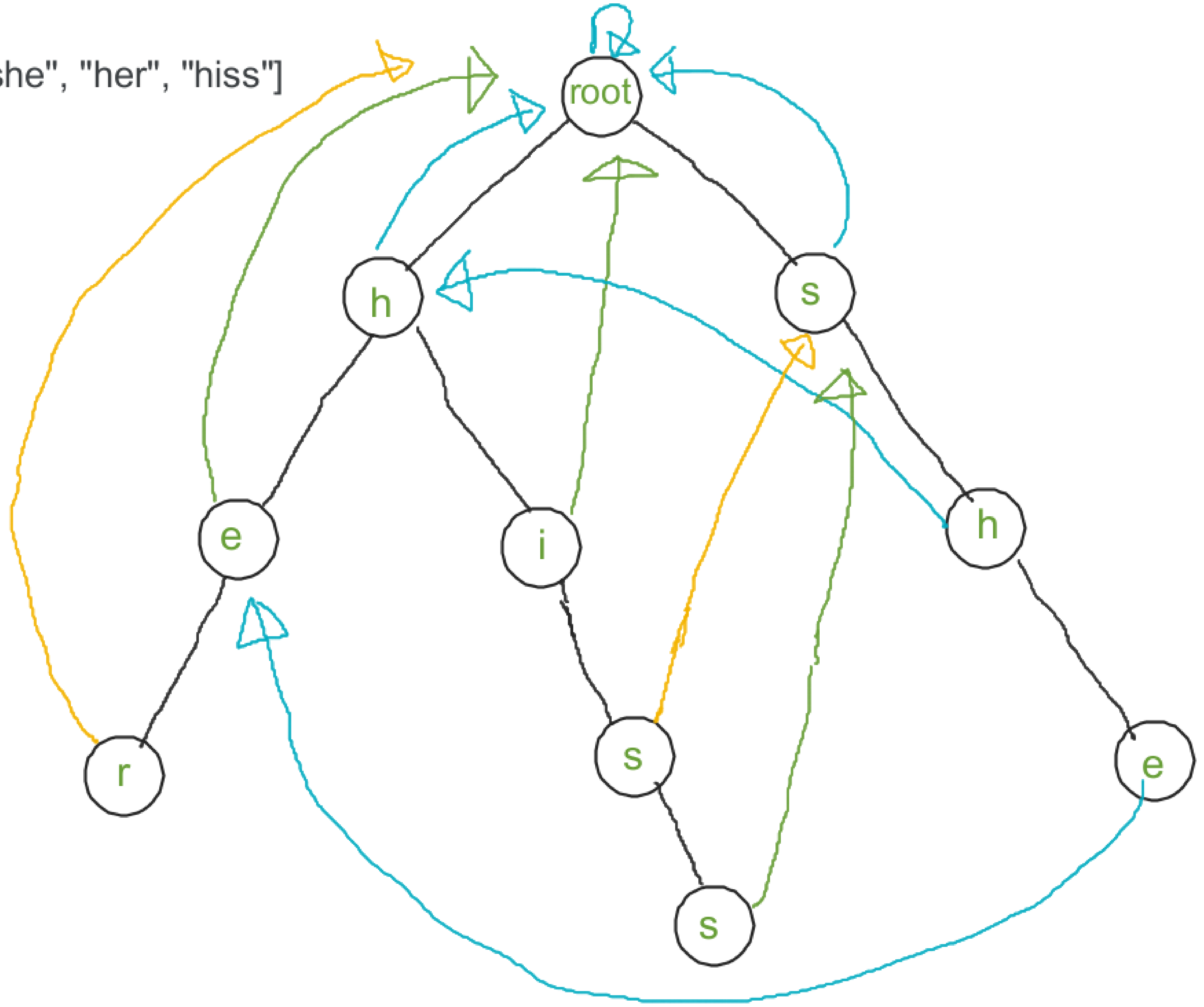
text = "ahishers"
patterns = ["he", "his", "she", "her", "hiss"]



```
text     = "ahishers"
patterns = ["he", "his", "she", "her", "hiss"]
```



text = "ahishers"
patterns = ["he", "his", "she", "her", "hiss"]



```
const int N = 1e4+7;        Number of characters in dictionary
 const int K = 10;          Alphabet size

   int nxt[N][K];           Children
   int go[N][K];            automaton

   int link[N];             Suffix link

   bool leaf[N];            isLeaf

   int par[N];              Parent
   char ch[N];              character of incoming edge
   int ex[N];               exit link
```

```
void addString(const string &s) {
        int cur = 0;
        for (char c: s) {
            int cc = c-'0';
            if (nxt[cur][cc] == -1) {
                nxt[cur][cc] = ++sz;
                ch[sz] = c;
                par[sz] = cur;
            }
            cur = nxt[cur][cc];
        }
        leaf[cur] = 1;
    }
```

```
///Amortized O(1)
    int getlink(int v) {
        if (link[v] != -1)  return link[v];
        if (v==0 || par[v] == 0)    return link[v] = 0;
        else return link[v] = Go(getlink(par[v]), ch[v]);
    }
```

```
///Amortized O(1)
    int Go (int v, char c) {
        int cc = c-'0';
        if (go[v][cc] != -1)     return go[v][cc];
        if (nxt[v][cc] != -1)    return go[v][cc] = nxt[v][cc];
        else return go[v][cc] = (v ? Go(getlink(v), c) : 0);
    }
```

```
///Amortized O(1)
    int exitlink(int v) {
        if (ex[v] != -1)                return ex[v];
        int nxt = getlink(v);
        if (nxt==0 || leaf[nxt])      return ex[v] = nxt;
        return ex[v] = exitlink(nxt);
    }
```

```
///returns number of matches (including multiple matches)
    ///O(no of matches + length of s)
    int match(string s) {
        int cur = 0;
        int ans = 0;
        for (auto c: s) {
            cur = Go(cur, c);
            int e = (leaf[cur] ? cur : exitlink(cur));
            while (e)
                ans++,
                e = exitlink(e);
        }
        return ans;
    }
```