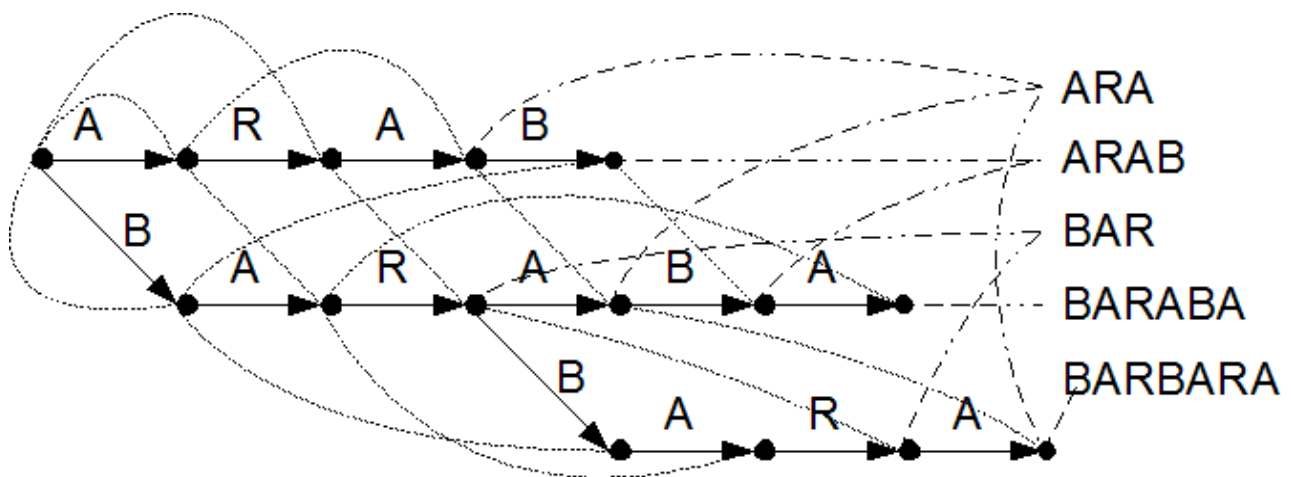


Vyhledávání v textu - algoritmus Aho - Corasick

Algoritmus Aho - Corasickové (vynalezený Alfredem Ahem a Margaret J. Corasickovou) je jeden z algoritmů pro vyhledávání vzorů v textu. Algoritmus zároveň vyhledává výskyt všech vzorů a má lineární časovou složitost $O(n + l + p)$, kde n je celková délka prohledávaného textu, l je součet délek všech vzorů a p je počet výskytů. Na vstupu dostane text, ve kterém budeme vyhledávat, a množinu vzorů. Výstupem jsou potom dvojice s pozicí a vzorem, který na této pozici končí. Celý algoritmus je možné rozdělit do dvou základních celků. Prvním je sestavení vyhledávacího automatu ze zadaných vzorů. Ve druhé části probíhá samotné vyhledávání. Vyhledávací automat se skládá ze tří funkcí - dopředné, zpětné a výstupní. Při vyhledávání potom znak po znaku procházíme prohledávaný text a podle aktuálního znaku přejdeme v automatu do dalšího stavu, pokud je definována dopředná funkce, přejdeme podle ní, jinak se vracíme zpět pomocí zpětné funkce dokud není přechod definován, kořen stromu je zarážka (zpětná funkce z kořene vede znovu do kořene). Vše lépe osvětlí příklad.

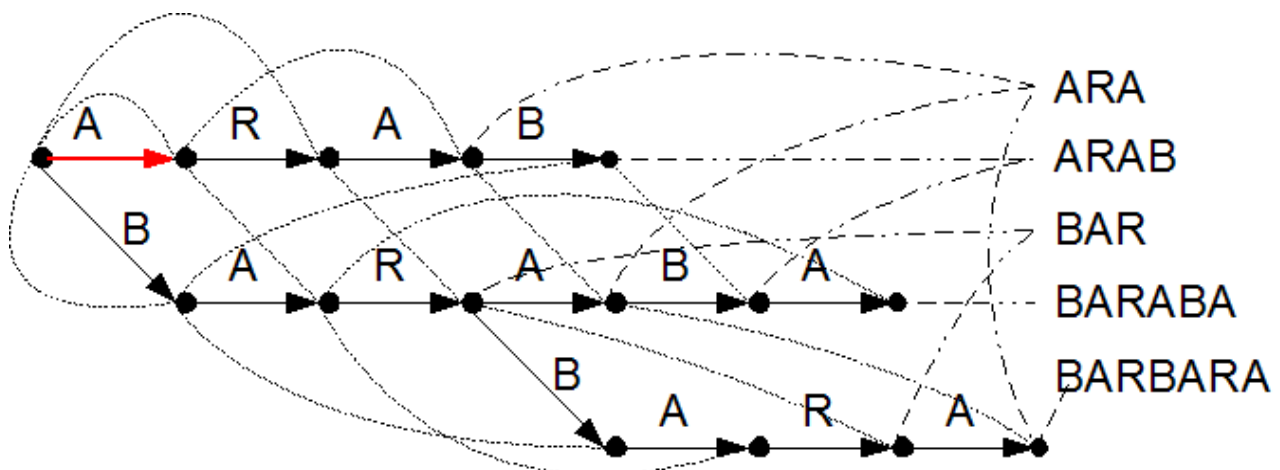
Vzory: ARA, ARAB, BAR, BARABA, BARBARA

Automat vytvořený z těchto vzorů:

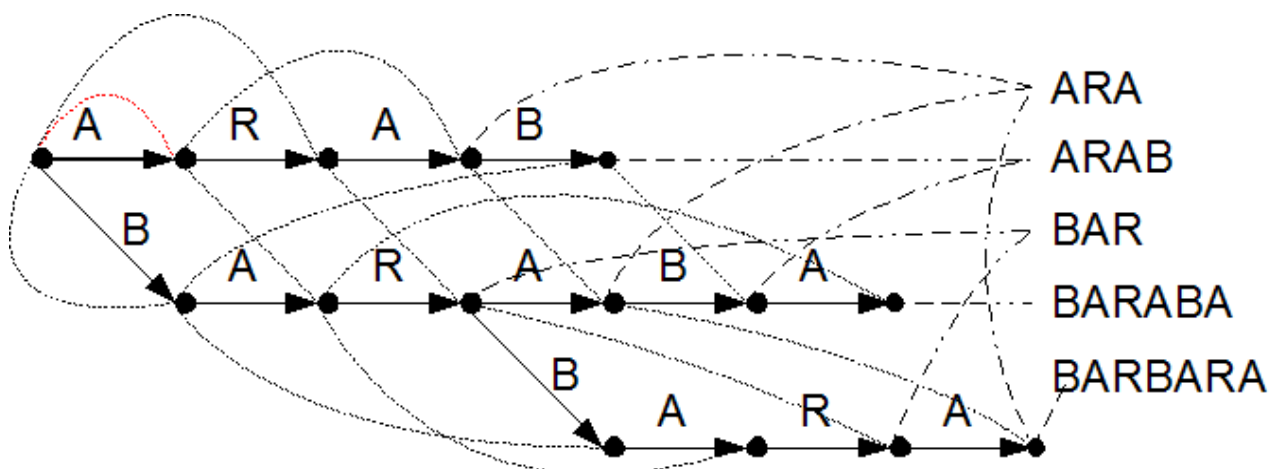


Body grafu reprezentují stavy automatu, plné šipky reprezentují dopřednou funkci, tečkované čáry reprezentují zpětnou funkci, čerchované reprezentují funkci výstupní.

Nyní například dostaneme zadaný text ACBARABARBARA. Algoritmus načte první písmeno A. Jsme v počátečním stavu, dopředná funkce pro A je definovaná, tedy přejdeme podle šipky A do následujícího stavu. Z tohoto stavu není definována výstupní funkce, tedy budeme pokračovat další iterací pro následující znak zadaného textu.



Dále načteme písmeno C. Dopředná funkce není definována, tedy budeme se vracet zpět podle zpětné funkce, která je značena čárkovanou čarou. Vrátime se tak do kořene, z kterého také není definován přechod pro C, ale dál už se vracet nedá, tudíž budeme pokračovat další iterací pro další znak.



Dalším znakem je B. Pro něj přechod existuje stejně tak jako pro následující znaky A, R. V této chvíli už se bude na výstup vypisovat výskyt slova BAR. Po přechodu o další A se vypíše ARA. Dále se opět přejde o B a A, vypíše se BARABA. Přechod z tohoto stavu není pro R definován, tedy se vracíme zpětnou funkcí a vidíme, že ze stavu, do kterého jsme se vrátili, přechod pro R definován je, čili zastavíme se zde a přejdeme podle R. Další postup je zřejmý.

Pseudokód vyhledávání

```
stav s = koren;
int pozice = 0;
foreach (znak c in prohledavany_text)
{
    while (neni_definovan_prechod_z_s_podle_c or s!=koren)
    {
        s = vrat_se_zpet_podle_zpetne_funkce;
    }
    s = prejdi_podle_c;
    Vypis_vystupy(pozice, s);
    pozice++;
}
```

Vlastnosti vyhledávacího automatu

Každá cesta začínající v kořenu je předponou nějakého vyhledávaného vzoru. Naopak každá předpona vzoru popisuje cestu k nějakému stavu. Tato předpona reprezentuje příslušný stav. Hloubka stavu s je rovna délce reprezentujícího slova.

Pro každý stav s reprezentovaný slovem u je stav, odkazovaný z s zpětnou funkcí, reprezentovaný nejdelší vlastní příponou slova u , která je současně předponou nějakého vzoru.

Pro každý stav s reprezentovaný slovem u a každý vzor y platí, že y bude vypsán výstupní funkcí ve stavu s právě když y je příponou u .

```
foreach (vzor)
{
    int i = 0;
    while (definován_přechod_pro_vzor[i])
    {
        Přejdi;
        ++i;
    }
    while (i < vzor.Length)
    {
        Přidej_stav;
        ++i;
    }
    Vytvoř_výstup_na_konci_vzoru;
}
```

Sestavení zpětné funkce a zbytku funkce výstupní

Poté co máme sestavenou dopřednou funkci a polotovar funkce výstupní, vytvoří se zpětná funkce a zbytek funkce výstupní. Do šířky projdeme celý strom sestavený v předchozí části. Vždy vybereme z fronty stav s , do kterého jsme přišli písmenem c , a vrátíme se do stavu z , kam vede zpětná funkce předchůdce stavu s . Pokud v tomto stavu bude definována dopředná funkce pro písmeno c (vedoucí do stavu q), bude zpětná funkce stavu s ukazovat do stavu q , jinak se vrátíme podle zpětné funkce zpět případně až do kořene.

```
while (fronta not empty)
{
    stav s = vyber_prvni_prvek_fronty();
    znak c = znak_kterym_se_dostaneme_do_stavu_s;
    p = s.predchudce;
    while (neni_definovan_prechod_z_p_znakem_c)
    {
        p = vrat_se_zpetnou_funkci;
    }
    q = prejdi_z_p_po_znaku_c;
    definuj_zpetnou_funkci_pro_stav_s_do_stavu_q;
    pridej_do_vystupni_funkce_stavu_s_vystupy_stavu_q;
    zarad_do_fronty_nasledovniky_s;
}
```