Desete vaje APS2: Iskanje v nizih

Na kratko osvežimo terminologijo in notacijo, povezano z nizi. Niz $S = a_1 a_2 \dots a_n$ je končno zaporedje znakov iz končne množice (abecede) Σ. Naj bo $|S| = n$ (dolžina niza S), $S[i] = a_i$ (indeksi naj se pričenjajo z 1, ne z 0, kot smo bili vajeni doslej),[1] $S[i : j] = a_i a_{i+1} \dots a_j$, $S[i :] = a_i a_{i+1} \dots a_n$ in $S[: i] = a_1 a_2 \dots a_i$. Niz S je predpona niza T ($S \sqsubseteq T$) natanko tedaj, ko je $|S| \leqslant |T|$ in $S = T[: |S|]$. Niz S je pripona niza T ($S \sqsupseteq T$) natanko tedaj, ko je $|S| \leqslant |T|$ in $S = T[|T| - |S| + 1 :]$.

Pri problemu iskanja v nizih želimo poiskati vse pojavitve niza P (*vzorec*) dolžine m v nizu T (*besedilo*) dolžine n, torej vse položaje $i \in \{1, \dots, n\}$, tako da je $T[i : i + m - 1] = P$ oziroma (s kompaktnejšim zapisom) $P \sqsubseteq T[i :]$. Ta problem zlahka rešimo v času $O(mn)$, na predavanjih pa smo (in še bomo) spoznali tudi nekaj pristopov, ki vse pojavitve vzorca v besedilu poiščejo v času $O(n)$, če si pred tem vzamejo $\Omega(m)$ časa za preobdelavo vzorca. Tokrat bomo spoznali še en pristop, ki pojavitve niza P v nizu T poišče v času $O(n)$, pri čemer za predobdelavo vzorca ne potrebuje nič več kot $O(m)$ časa.

Z-funkcija

Naj bo S poljuben niz (v tem razdelku ne bomo ločevali med vzorcem in besedilom) in naj bo vrednost $Z_i(S)$ (ali kar Z_i, če je niz S znan iz konteksta) za $i \geqslant 2$ definirana kot dolžina najdaljšega podniza niza S, ki se prične na indeksu i in je hkrati predpona niza S:

$$Z_i(S) = \max\{k \mid S[i : i + k - 1] \sqsubseteq S\}.$$

Slika 1 prikazuje primer niza S in pripadajoče vrednosti njegove Z-funkcije. Na primer, $Z_4 = 2$, ker se na indeksu 4 v nizu S prične podniz dolžine 2 (`ab`), ki se pojavi tudi na začetku niza S, medtem ko podniz dolžine 3, ki se tam prične (tj. `abc`), ni predpona niza S.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$S[i]$	a	b	a	a	b	c	a	b	a	b	a	a	b	c
Z_i	–	0	1	2	0	0	3	0	6	0	1	2	0	0
Z-škatle														
r_i	–	–	3	5	5	–	9	9	14	14	14	14	14	14
ℓ_i	–	–	3	4	4	–	7	7	9	9	9	9	9	9

Slika 1: *Primer izračuna Z-funkcije in Z-škatel.*

Definirajmo še sledeče pojme:

- Če je $Z_i > 0$, je pripadajoča *Z-škatla* interval $[i, i + Z_i - 1]$ (i je levi, $i + Z_i - 1$ pa desni rob Z-škatle).
- Naj bo $\mathcal{Z}_i$ množica vseh Z-škatel, ki vsebujejo indeks i. Če je $|\mathcal{Z}_i| \geqslant 1$, potem naj bo r_i desni rob tiste Z-škatle iz množice $\mathcal{Z}_i$, ki se konča na največjem indeksu, l_i pa naj bo levi rob te iste Z-škatle. V našem primeru indeks 13 pripada Z-škatlama [12, 13] in [9, 14]. Ker se škatla [9, 14] zaključi na večjem indeksu kot škatla [12, 13] ($14 > 13$), je $r_{13} = 14$ in $l_{13} = 9$.

[1] To je konstantna dilema. Mislim, da se bodo naslednje leto tudi indeksi pri običajnih tabelah pričenjali z 1, da bom konsistenten s Cormenom *et al.* in standardno matematično prakso.

Če je indeks i vsebovan v več Z-škatlah z istim maksimalnim desnim robom, lahko za vrednost l_i vzamemo levi rob katerekoli od njih.

Izračun Z-funkcije v času $O(|S|)$

Z uporabo definicije lahko Z-funkcijo izračunamo v času $O(|S|^2)$, obstaja pa tudi ne pretirano zapleten algoritem, ki to doseže v času $O(|S|)$. Algoritem se sprehodi po nizu od leve proti desni (od $i = 2$ do $i = |S|$) in sproti računa Z_i, r_i in l_i. Ker vedno hrani zgolj vrednosti r_i in l_i za trenutni i, bomo namesto r_i in l_i pisali kar r in l.

Algoritem najprej neposredno po definiciji (s primerjavo znakov) izračuna vrednost Z_2 ter nastavi $l \leftarrow 2$ in $r \leftarrow 1 + Z_2$. Sedaj pa (induktivno) predpostavimo, da je algoritem pravilno izračunal vrednosti Z_i za vse $i < k$ in da sta l in r nastavljena na indeksa levega in desnega roba najdlje segajoče Z-škatle, ki se prične pred indeksom k in vsebuje indeks k, če takšna škatla obstaja (v nasprotnem primeru pa naj bo $r < k$). Vrednost Z_k ter posodobljeni vrednosti l in r izračunamo takole:

Primer 1. Če je $r < k$, potem se trenutni indeks (k) ne nahaja v nobeni od doslej odkritih Z-škatel, zato vrednost Z_k izračunamo po definiciji. Nastavimo $l \leftarrow k$ in $r \leftarrow k + Z_k - 1$ (tudi v primeru $Z_k = 0$).

Primer 2. Za lažje razumevanje tega primera spremljajmo sliki 2 in 3. Če je $r \geqslant k$, potem se trenutni indeks (k) nahaja v Z-škatli $[l, r]$. Ta Z-škatla določa podniz $S[l : r]$ (na slikah 2 in 3 je prikazan z modro), ki se po definiciji Z-funkcije ponovi na začetku niza S. To pomeni, da velja $S[l : r] = S[1 : r - l + 1]$ in od tod tudi $S[k] = S[k']$, pri čemer je $k' = k - l + 1$.

Recimo, da je $Z_{k'} > 0$ (primer $Z_{k'} = 0$ ni zanimiv, a ga lahko obravnavamo skupaj s primerom $Z_{k'} > 0$). To pomeni, da se po definiciji Z-funkcije niz $S[k' : k' + Z_{k'} - 1]$ ponovi na začetku niza S, hkrati pa se vsaj del tega niza (tisti, ki je na slikah 2 in 3 prikazan z zeleno) ponovi tudi na indeksih od k naprej, saj velja $S[l : r] = S[1 : r - l + 1]$ in $l < k \leqslant r$.

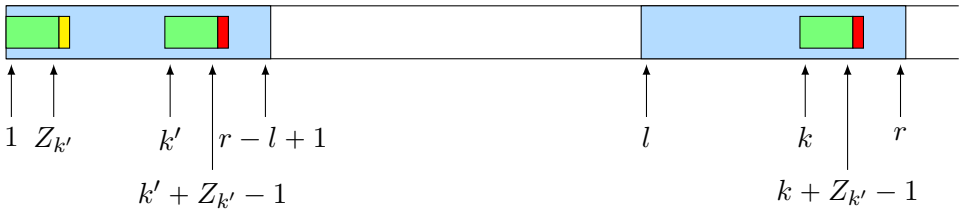
V odvisnosti od tega, kolikšen del niza $S[k' : k' + Z_{k'} - 1]$ se ponovi na indeksih od k naprej, ločimo dva podprimera:

Primer 2a. Če je $Z_{k'} \leqslant r - k$ (slika 2), potem se celoten zeleni niz $S[k' : k' + Z_{k'} - 1]$ ponovi na indeksih od k naprej. Velja torej $S[1 : Z_{k'}] = S[k' : k' + Z_{k'} - 1] = S[k : k + Z_{k'} - 1]$. Od tod neposredno sledi $Z_k \geqslant Z_{k'}$, v resnici pa je kar $Z_k = Z_{k'}$. Zakaj? Zato, ker je znak $S[k + Z_{k'}] = S[k' + Z_{k'}]$ (rdeč) različen od znaka $S[Z_{k'} + 1]$ (rumen); če bi bili vsi trije znaki med seboj enaki, bi bil $Z_{k'}$ vsaj za 1 večji, kot je dejansko. Skratka, v tem primeru lahko nastavimo $Z_k = Z_{k'}$, l in r pa pustimo pri miru.

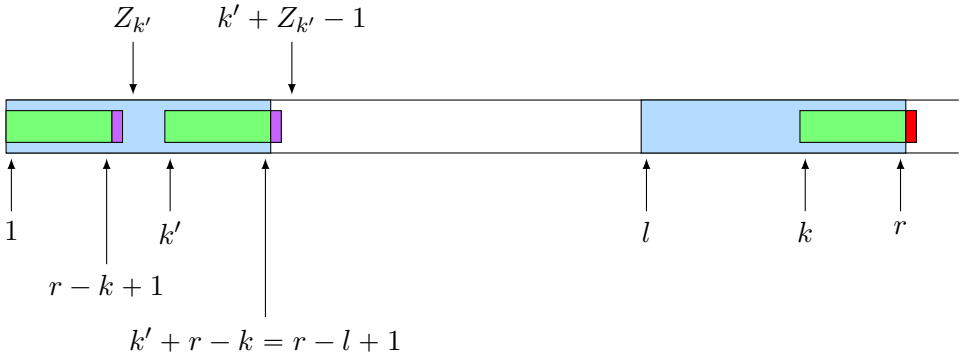
Primer 2b. V primeru $Z_{k'} > r - k$ (slika 3) se gotovo vsaj prvih $r - k + 1$ znakov niza $S[k' : k' + Z_{k'} - 1]$ ponovi na indeksih od k naprej, hkrati pa se *celoten* niz $S[k' : k' + Z_{k'} - 1]$ ponovi na začetku niza S, ker je vrednost Z-funkcije na indeksu k' pač enaka Z'_k. Ker je $Z_{k'} \geqslant r - k + 1$, gotovo velja $Z_k \geqslant r - k + 1$.

Za koliko je Z_k lahko večji od $r - k + 1$? Ne vemo, ker smo niz S prebrali samo do indeksa r. Če torej želimo določiti vrednost Z_k, moramo znake na indeksih $r + 1$, $r + 2$ itd. primerjati z znaki na indeksih $r - k + 2$, $r - k + 3$ itd., dokler ne naletimo na prvo neujemanje — npr. $S[r + p] \neq S[r - k + 1 + p]$. Tedaj nastavimo $Z_k = r - k + p$ ter posodobimo $r \leftarrow r + p - 1$ in $l \leftarrow k$.

Celoten postopek izračuna Z-funkcije je prikazan kot algoritem 1. Psevdokoda natančno



Slika 2: *Primer 2a. Modra podniza sta si med seboj enaka, pa tudi vsi trije zeleni podnizi so si med seboj enaki. Rdeča znaka sta si med seboj enaka, rumeni pa je drugačen.*



Slika 3: *Primer 2b. Modra podniza sta si med seboj enaka, pa tudi vsi trije zeleni podnizi so si med seboj enaki. Vijolična znaka sta si med seboj enaka, rdeči pa je lahko enak vijoličnima, lahko pa je tudi različen od njiju.*

sledi opisu algoritma, dejansko pa je Z-funkcijo, kot si lahko ogledate npr. na spletišču GeeksforGeeks, mogoče sprogramirati s precej manjšo količino kode.

Kolikšna je časovna zahtevnost algoritma za izračun Z-funkcije v odvisnosti od dolžine niza S? Zaradi dvojne zanke bi jo na prvi pogled ocenili na $O(|S|^2)$. Ta zgornja meja je sicer (tudi) pravilna, tesna zgornja meja pa je $O(|S|)$. Če se osredotočimo na funkcijo ŠteviloUjemanj, v kateri se izvajajo primerjave med znaki, opazimo sledeče:

- Funkcija ŠteviloUjemanj se zaključi, ko naletimo na prvo neujemanje znakov. Ker funkcijo pokličemo največ $|S|$-krat, je skupno število odkritih neujemanj znakov kvečjemu enako $|S|$.
- Ocenimo še število odkritih ujemanj znakov. Funkcijo ŠteviloUjemanj vsakokrat pokličemo s parametrom $p \geqslant r+1$. Recimo, da v funkciji izvršimo s medsebojnih primerjav znakov, od katerih je $s-1$ uspešnih, zadnja pa neuspešna. Ko se funkcija zaključi, povečamo r vsaj za $s-1$. Ker je vrednost parametra p ob vsakem klicu funkcije vsaj za s večja od vrednosti tega parametra ob predhodnem klicu funkcije, skupno število odkritih ujemanj znakov ne more biti večje od $|S|$.

Skupno število primerjav znakov (uspešnih in neuspešnih) tako znaša $O(|S|)$. Režijski stroški te zgornje meje ne spremenijo.

Uporaba Z-funkcije za iskanje v času $O(n+m)$

Precej časa smo porabili za izračun Z-funkcije, dejansko pa še vedno ne vemo, kako bi jo lahko izkoristili za učinkovito iskanje pojavitev vzorca P dolžine m v nizu T dolžine n. Takole gre: sestavimo niz $S = P\$T$, pri čemer je \$ znak, ki ne nastopa niti v P niti v T, nato pa izračunamo Z-funkcijo niza S. Če najdemo indeks $i \geqslant m+2$ z lastnostjo $Z_i = m$, potem vemo, da je $S[i : i+m-1] = T[i-m-1 : i-2] = P$. (Primer $Z_i > m$

Algoritem 1 Izračun Z-funkcije.

```
function ŠteviloUjemanj(S, p, q)
    ▷ Vrne max{t | S[p : p + t − 1] = S[q : q + t − 1]}. Predpostavlja p > q. ◁
    n ← |S|
    t ← 0
    while p + t ⩽ n ∧ S[p + t] = S[q + t] do
        t ← t + 1
    return t

function Z(S)
    n ← |S|
    Z_2 ← ŠteviloUjemanj(S, 2, 1)
    l ← 2
    r ← 1 + Z_2
    for k ← 3 to n do
        if r < k then
            Z_k ← ŠteviloUjemanj(S, k, 1)
            l ← k
            r ← l + Z_k − 1
        else
            k′ ← k − l + 1
            if Z_k′ ⩽ r − k then
                Z_k ← Z_k′
            else
                Z_k ← r − k + 1 + ŠteviloUjemanj(S, r + 1, r − k + 2)
                l ← k
                r ← l + Z_k − 1
    return ⟨Z_2, . . . , Z_n⟩
```

zaradi ločila \$ ni mogoč.) Izračun Z-funkcije terja čas $O(n + m + 1)$ oziroma preprosto $O(n)$, če predpostavimo $n \geqslant m$, dodatno preverjanje lastnosti $Z_i = m$ pa te meje seveda ne spremeni.