UNIVERZA V LJUBLJANI

FAKULTETA ZA MATEMATIKO IN FIZIKO

Matematika – praktična matematika (VSŠ)

Darko Brljak

Boyer-Mooreov algoritem

Diplomska naloga

Ljubljana, 2011

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Zahvala

Za mentorstvo pri diplomski nalogi gre iskrena zahvala mag. Matiji Lokarju.

Za spodbudo hvala staršem, Evi in Urošu.

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Vsebina

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Program diplomske naloge

V okviru diplomske naloge predstavite Boyer-Mooreov algoritem, s pomočjo katerega ugotavljamo, ali se v besedilu pojavlja določen podniz. Poleg osnovnega algoritma predstavite tudi nekatere izpeljanke tega algoritma, kot so Horspoolov algoritem, Raitov algoritem in druge.

Osnovna literatura:

- R. S. Boyer, J. S. Moore: A Fast String Searching Algorithm. Communications of the ACM, 20, 10, 762–772 (1977),
- C. Charras, T. Lecroq, Exact string matching algorithms, http://www-igm.univ-mlv.fr/~lecroq/string/index.html, (1997),
- R. N. Horspool: Practical Fast Searching in Strings. Software – Practice and Experience 10, 501–506 (1980),
- H. W. Lang, String matching algorithms, http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/indexen.htm, (2008).

mentor

mag. Matija Lokar

DIPLOMSKA NALOGA :
FAKULTETA ZA MATEMATIKO IN FIZIKO

Povzetek

V diplomskem delu sem predstavil nekaj algoritmov za iskanje podniza v danem nizu.

V prvem delu diplomske naloge sem podrobnejše opisal Boyer-Mooreov algoritem, v drugem delu sem predstavil nekaj različic tega algoritma (Horspool, Quick search …), medtem ko sem v tretjem delu vse opisane algoritme primerjal.

Abstract

In the thesis I have presented some string-matching algorithms.

In the first part of the thesis I have described in detail the Boyer-Moore algorithm. In the second part I have presented some versions of this algorithm (Horspool, Quick search …) and in the third I have compared all described algorithms.

Math. Subj. Class. (2011): 68Q25, 68W05, 68W32, 68W40

Computing Review Class. System (1998): F.2.2

Ključne besede: besedilo, vzorec, niz, podniz, algoritem, ujemanje, premik, pravilo dobre pripone, pravilo slabega znaka.

Keywords: text, pattern, string, substring, algorithm, matching, shift, good-suffix rule, bad-character rule.

DEFINICIJE

Vzorec/podniz in besedilo/niz sta sestavljena iz znakov/simbolov.

Primer:

Vzorec/podniz: "kuža"

Besedilo/niz: "Danes sije sonce."

Znak/simbol: 'a'

Znak označimo s *c* (ang. character).

c iz vzorca in besedila: 'a', 'c', 'D', 'e', 'i', 'j', 'k', 'n', 'o', 's', 'u', 'ž', ' ' (presledek), '.'

Vzorec označimo z $x = x[0 .. m-1]$. Dolžina vzorca je m.

x = "kuža", $x[0]$ = 'k', $x[1]$ = 'u', $x[2]$ = 'ž', $x[3]$ = 'a', m = 4

Besedilo označimo z $y = y[0 .. n-1]$. Dolžina besedila je n.

y = "Danes sije sonce.", $y[0]$ = 'D', ... , $y[16]$ = '.', n = 17

Znak na *j-tem* mestu v vzorcu označimo z $x[j]$.

$x[2]$ = 'ž'

Znak na *j-tem* mestu v besedilu označimo z $y[j]$.

$y[9]$ = 'e'

Vzorec in besedilo sta zgrajena s pomočjo končnega zaporedja znakov, ki ga imenujemo *abeceda*. Abeceda je označena s simbolom Σ, ki je velikosti σ.

Σ = {'a', 'c', 'D', 'e', 'i', 'j', 'k', 'n', 'o', 's', 'u', 'ž', ' ', '.'}, σ = 14

Beseda u je predpona besede w, če obstaja beseda v (lahko prazna), da velja $w = uv$.

Beseda v je pripona besede w, če obstaja beseda u (lahko prazna), da velja $w = uv$.

w = "Danes", u = "Da", v = "nes"

Beseda z je podniz ali del besede w, če obstajata dve besedi u in v (lahko prazni), da velja $w = uzv$.

w = "kuža", z = "ža", u = "ku", v = 0 (prazna beseda)

Število p je perioda besede w, če velja za i, $0 \leq i < m-p$, $w[i]=w[i+p]$. Najmanjšo periodo besede w imenujemo *perioda*, označimo jo $per(w)$.

w = "blablabla", $w[0] = w[3]$, $w[1] = w[4]$, $w[2] = w[5]$, $w[3] = w[6]$, $w[4] = w[7]$, $w[5] = w[8]$, $per(w)$ = 3

Beseda w, dolžine l, je periodična, če je njena najmanjša perioda manjša ali enaka $l/2$, drugače je beseda w neperiodična.

$w1$ = "hovhov", l = 6, $per(w1)$ = 3, $per(w1) \leq l/2$, zato je $w1$ periodična

$w2$ = magma, l = 5, $per(w2)$ = 3, $per(w2) > l/2$, zato je $w2$ neperiodična

UVOD

Pri delu z besedili je pogosto treba ugotoviti, ali niz vsebuje določen podniz. Zato ima večina operacijskih sistemov v programskih orodjih vgrajene **algoritme za ugotavljanje prisotnosti podnizov**.

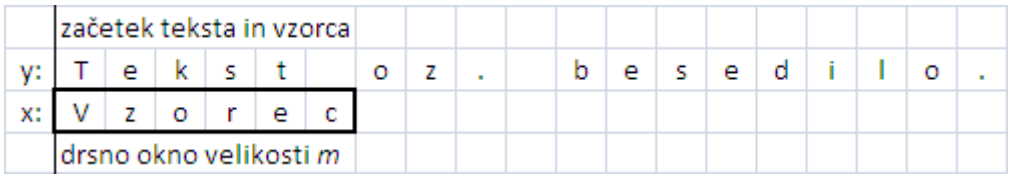
Hitrost obdelave posameznega podatka se na računalnikih redno povečuje, redno pa se povečuje tudi obseg podatkov. In čeprav so podatki dandanes shranjeni na različne načine, ostaja besedilo glavna odlika za izmenjavo informacij. To je razlog, zakaj morajo biti algoritmi za delo z nizi učinkoviti.

V diplomskem delu se bom osredotočil na iskanje podniza v nizu z Boyer-Mooreovim algoritmom (v nadaljevanju algoritem BM). Obstaja več različic tega algoritma, poleg BM pa bom opisal še naslednje algoritme: Horspool, Quick Search, Smith in Raita. Vsak od teh algoritmov ima svoje prednosti in slabosti. Različne so tudi njihove časovne in prostorske zahtevnosti, o katerih pa v diplomski nalogi ne bom pisal podrobnejše in jih bom le navedel.

Algoritmi, ki jih bom opisoval, iščejo v besedilu vse pojavitve vzorca. Če algoritem vzorec v besedilu najde, lahko na primer vrne indeks prve črke ujemanja v besedilu, ali pa kot v programu Microsoft Word, označi vzorec v besedilu. To je odvisno od različic algoritmov.

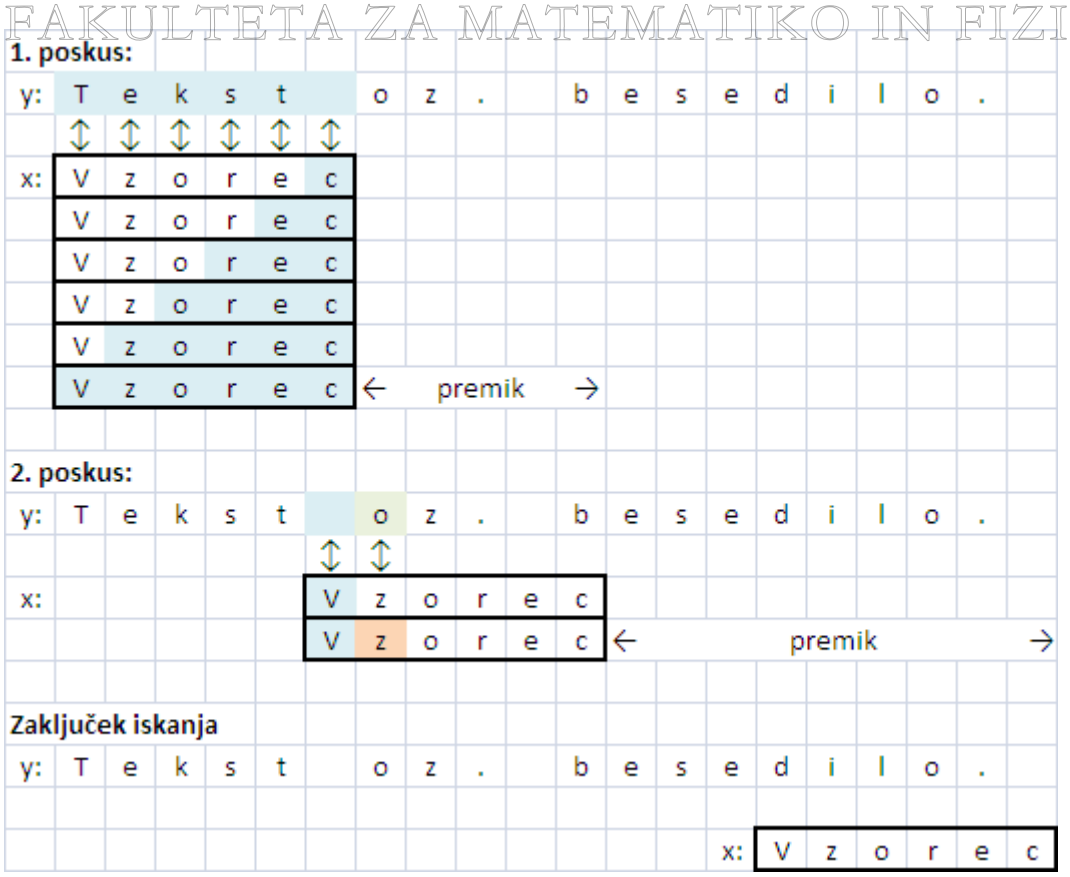
Vsi algoritmi, ki jih bom opisal, uporabljajo pri iskanju podniza v nizu **mehanizem drsnega okna**. Ta mehanizem deluje na naslednji način:

S pomočjo okna velikosti *m* označimo vzorec. Sledi poravnava med začetkom besedila in začetkom vzorca (glej slika 1). Zatem tehnika primerja istoležne simbole med besedilom in vzorcem. Takšni primerjavi rečemo **poskus**.



Slika 1: Poravnava drsnega okna z začetkom besedila

Če sta med poskusom znaka enaka, sledi primerjava med naslednjima znakoma. Če se vsi znaki v besedilu in vzorcu ujemajo (ko smo torej vzorec v besedilu našli), primerno ukrepamo (si zapomnimo mesto v besedilu, označimo ta del besedila …), čemur sledi **premik** vzorca z oknom v desno. Premik okna se zgodi tudi takrat, kadar si znaka, ki se primerjata, nista enaka. Različna je le dolžina tega premika. Ob premiku okna v desno pravimo, da je poskusa konec. Sledi naslednji poskus. Postopek ponavljamo, dokler se desni rob okna ne premakne bolj desno od desnega roba besedila. Takrat seveda ujemanja ne moremo več doseči (glej slika 2).



Slika 2: Poskus, primerjava znakov, premik in zaključek iskanja

Vsak poskus označimo glede na položaj j-tega simbola v besedilu. Okno (vzorec) je postavljeno pod simbole iz niza na položajih *y[j .. j+m-1]*.

Glede na vrstni red primerjanja znakov med nizom in podnizom v poskusu ločimo štiri kategorije algoritmov:

i. Algoritmi s primerjanjem z LEVE proti DESNI.
ii. Algoritmi s primerjanjem z DESNE proti LEVI (v praksi najučinkovitejši algoritmi).
iii. Algoritmi s SPECIFIČNIM vrstnim redom primerjanja (na primer najprej se primerja zadnji znak vzorca, nato prvi, sredinski, drugi … in predzadnji znak).
iv. Algoritmi s POLJUBNIM vrstnim redom primerjanja (vrstni red primerjanja ni pomemben, izbira se naključno).

1. BOYER-MOOREOV ALGORITEM

Osnovne lastnosti

Boyer-Mooreov algoritem velja za enega najbolj učinkovitih algoritmov za iskanje podniza v nizu. Uporabljen je v mnogih aplikacijah, kot na primer v raznih urejevalnikih besedila. Pri posameznem poskusu uporablja za primerjavo znakov tehniko **z desne proti levi**. Tehnika se imenuje tudi »zrcalna« (ang. »looking glass«), kar pomeni, da začne primerjavo med vzorcem in besedilom pri skrajno desnem znaku v vzorcu in se pomika proti levi. Povejmo še enkrat, da iščemo, kje v besedilu se (če sploh) pojavi vzorec (podniz).

Kot smo omenili že v uvodu, se tudi tu ob naslednjih poskusih vzorec pomika proti desni, za besedilo pa si mislimo, da ostaja na istem mestu. Pri premikanju vzorca proti desni Boyer-Mooreov algoritem uporabljamo za različne načine določanja premika. Pri računanju premikov se uporabljata dve tehniki: **»pravilo dobre pripone«** (good-suffix rule) in **»pravilo slabega znaka«** (bad-character shift). Natančneje ju bom opisal pozneje. Najprej si oglejmo njuno delovanje kar na primerih.

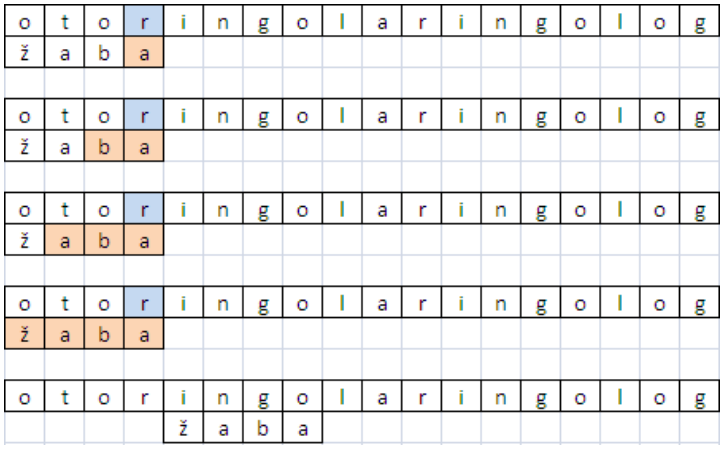
Primeri

Primer 1

V prvem poskusu (torej ob prvem primerjanju) se znak iz besedila, ki stoji na enakem mestu kot skrajni desni znak iz vzorca, v slednjem ne pojavi. Vzorec premaknemo za *m* mest v desno. Takšen primer na primer nastopi, če v nizu "otorinolaringolog" iščemo niz "žaba".

BESEDILO: "otoringolaringolog"

VZOREC: "žaba"



Primerjamo črki 'r' in 'a', dobimo neenakost.

Ugotovimo, da se črka 'r' v vzorcu sploh nikoli ne pojavi.

Poravnava vzorca in primerjanega dela niza s skupno črko torej ni mogoča, ne glede na to, kako bi premikali vzorec. Zato lahko vzorec premaknemo za štiri mesta naprej.

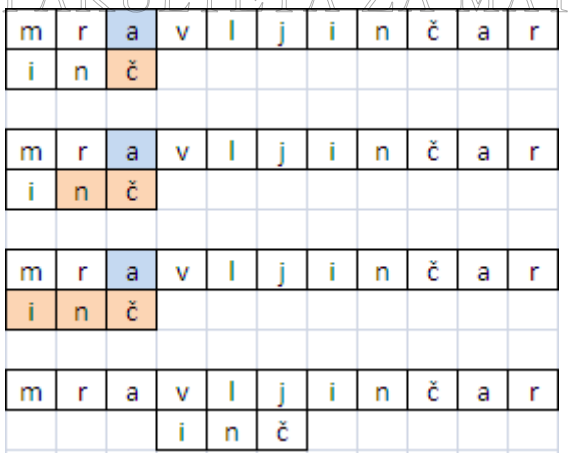
Slika 3: Črka iz besedila ne nastopa v vzorcu

Primer 2

Vzorec "inč" se v besedilu "mravljinčar" pojavi. Vseeno pa vzorec v prvem poskusu iskanja ne vsebuje znaka iz besedila, ki stoji nad zadnjim znakom vzorca. V takšnem primeru vzorec premaknemo za *m* mest.

BESEDILO: "mravljinčar"

VZOREC: "inč"



Primerjamo znaka 'a' in 'č', dobimo neujemanje.

Znak 'a' v vzorcu ne nastopa.

Vzorec premaknemo za tri mesta v desno, torej za znak 'a'.

Vzorec "inč" nastopa v nadaljevanju besedila, kar pa na iskanje v trenutnem poskusu ne vpliva.

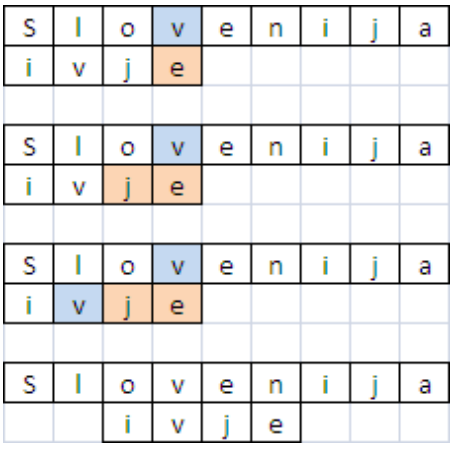
Slika 4: Črka iz besedila ne nastopa v vzorcu, vzorec nastopa v nadaljevanju besedila

Primer 3

Znak iz besedila, ki stoji nad zadnjim znakom vzorca, je od slednjega različen, pojavi pa se na kakšnem drugem mestu v iskanem vzorcu. Vzorec zato premaknemo za toliko mest, da sta znaka, ki se ujemata, poravnana eden z drugim.

BESEDILO: "Slovenija"

VZOREC: "ivje"



Primerjamo črki 'v' in 'e', dobimo neenakost.

Ugotovimo, da se črka 'v' v vzorcu pojavi.

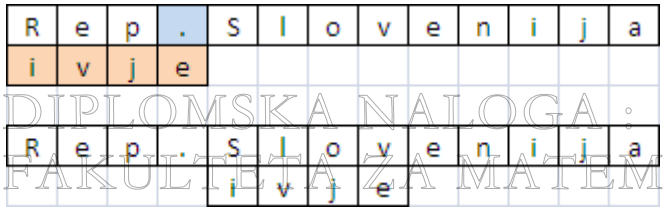
Zato vzorec in besedilo poravnamo tako, da črki 'v' stojita ena nad drugo. Vzorec torej premaknemo za dve mesti v desno.

Slika 5: Črka iz besedila nastopa v vzorcu

Seveda vsi omenjeni primeri delujejo enako tudi takrat, ko smo »sredi« algoritma, torej, ko se je določen del besedila že primerjal. Tako prvi poskus primerjanja pri besedilu "Rep.Slovenija" in vzorcu "ivje" premakne vzorec v položaj, kot ga vidimo pri primeru 3.

BESEDILO: "Rep.Slovenija"

VZOREC: "ivje"



Znak '.' v vzorcu ne nastopa, zato lahko vzorec premaknemo za štiri mesta v desno.

Sledi drugi poskus, kjer je postopek točno tak, kot smo opisali prej (glej slika 5).

Slika 6: Prvi poskus, sledi drugi poskus (primer 3)

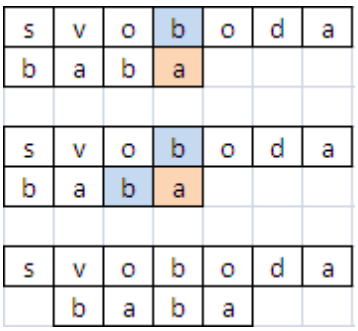
Primer 4

Znak iz besedila, ki ga iščemo v vzorcu, se v slednjem pojavi večkrat. Vzorec poravnamo tako, da se s simbolom v besedilu poravna skrajni desni simbol iz vzorca, ki ustreza primerjavi.

Če bi vzorec poravnali glede na levi ujemajoči se znak, lahko v tem primeru izgubimo morebitno ujemanje (glej primer 5).

BESEDILO: "svoboda"

VZOREC: "baba"



V vzorcu "baba" nastopata dve črki 'b'.

Vzorec poravnamo s tisto, ki je najdena prva.

Ker algoritem deluje z desne proti levi, je primerna skrajno desna črka 'b' v vzorcu.

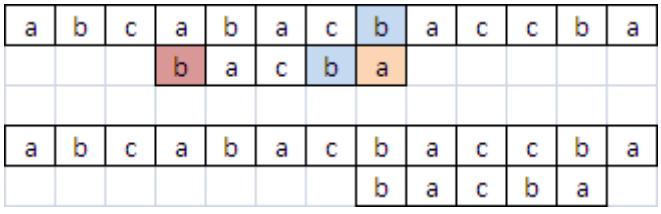
Slika 7: Črka iz besedila v vzorcu nastopa večkrat

Primer 5

V trenutnem poskusu (glej slika 8) iščemo znak 'b' v vzorcu. Pojavi se na prvem in četrtem mestu. Če bi z označenim znakom v besedilu poravnali črko 'b' na prvem mestu vzorca, bi izgubili rešitev.

BESEDILO: "abcabacbaccbab"

VZOREC: "bacba"



Slika 8: Nepravilen premik

Pri dosedanjih primerih se nista ujemala že prva primerjana znaka. Zato smo vzorec lahko takoj premaknili. Pri tem smo premik določili glede na to, kje se znak iz besedila nahaja v vzorcu.

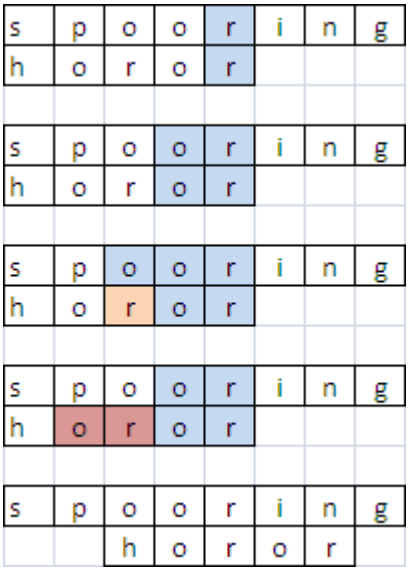
Sedaj pa si oglejmo sklop primerov, če se skrajni desni znak v vzorcu ujema z znakom, ki stoji v nizu nad njim. Primerjava se nadaljuje med znakoma, ki nastopata za eno mesto bolj levo, tako v besedilu, kot tudi v vzorcu. Ta postopek traja, dokler se znaka med seboj ne razlikujeta ali pa pridemo do levega roba vzorca (popolno ujemanje).

Primer 6

Pri primeru 6 imamo slučaj, ko metoda v vzorcu najde ujemajoči se sklop znakov. Pripona (zadnjih nekaj znakov) v vzorcu se torej pojavi v besedilu. Načeloma obstaja več možnih premikov. Izberemo maksimalnega (tistega, ki vzorec premakne najbolj desno) in glede na tega poravnamo ujemajoča se sklopa znakov.

BESEDILO: "spooring"

VZOREC: "horor"



Slika 9: Ujemajoča pripona se ponovi v vzorcu

Pri primerjanju ugotovimo, da se sklop znakov "or" ujema.

Pri primerjavi tretje črke dobimo neenakost. Algoritem BM predlaga dva možna premika.

Po pravilu slabega znaka iščemo črko 'o' (neujemajoči se znak iz besedila) v vzorcu. Najdemo jo na drugem in na četrtem mestu. Poravnava (poravnavamo »neujemajoči« se 'o' iz besedila, torej tistega, ki je pred podnizom "or") z bolj desnim znakom (torej z 'o' na četrtem mestu v vzorcu) bi pomenila premik v levo; to nam ne ustreza. Če pa poravnavamo 'o' na drugem mestu v vzorcu, bi opravili premik za en znak v desno. Pravilo slabega znaka nam torej omogoča premik za en znak v desno.

Pravilo dobre pripone pa poišče, če se sklop "or" v vzorcu pojavi še kje. Ker se (na drugem mestu v vzorcu), pravilo dobre pripone predlaga premik, in sicer za dve mesti v desno.

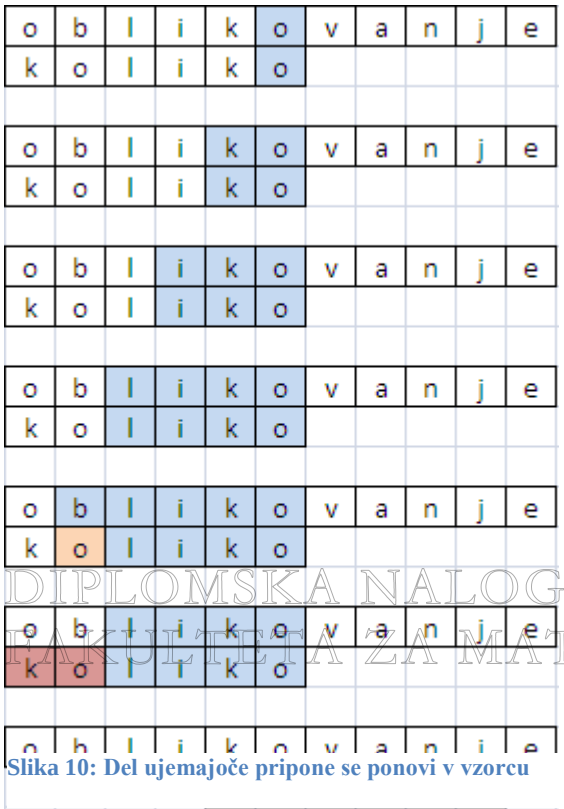
Torej pravilo dobre pripone omogoča večji premik kot pravilo slabega znaka. Zato seveda uporabimo to pravilo in vzorec premaknemo za dve mesti v desno.

Primer 7

Pri primeru 6 smo ujemajoči se sklop ponovno poiskali v vzorcu. Lahko pa se zgodi, da se celoten sklop v vzorcu ne pojavi več, pojavi pa se pripona tega sklopa. V vzorcu torej lahko najdemo le zaključek sklopa znakov, ki se ujema.

BESEDILO: "oblikovanje"

VZOREC: "koliko"



Slika 10: Del ujemajoče pripone se ponovi v vzorcu

Najdemo ujemajoči se sklop znakov "liko".

Ob neujemajočih se znakih ('b' in 'o') poskuša metoda prej omenjeni sklop najti v vzorcu še kje drugje.

Celotnega sklopa ne najde, najde le njegov del, to je pripono "ko".

Sledi poravnava vzorca z besedilom na mestu ujemajočega se dela sklopa "ko".

V tem poskusu predlaga premik tudi pravilo slabega znaka. Ob neujemanju črk 'b' in 'o' išče pravilo v vzorcu črko 'b'. Ker je ne najde, predlaga premik za dve mesti v desno, torej za mestom neujemanja. Pravilo dobre pripone predlaga premik za štiri mesta, pravilo slabega znaka pa za dve mesti v desno. Uporabimo večji premik, torej premik dobre pripone.

S temi sedmimi primeri smo pokrili večino primerov, ki lahko nastopijo med delovanjem algoritma. Sedaj pa si algoritem oglejmo podrobnejše.

Delovanje algoritma

Najprej si natančno oglejmo obe pravili za premike.

Pravilo dobre pripone (good-suffix rule)

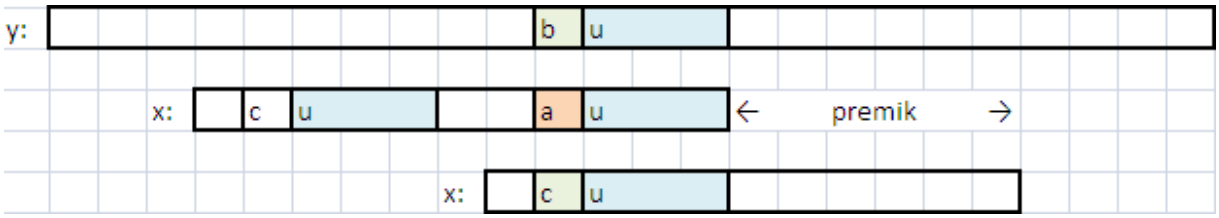
To pravilo nastopi takrat, ko se prvih nekaj (vsaj en) znakov pri primerjanju ujema. Denimo, da znaka $x[i] = a$ v vzorcu in $y[i + j] = b$ v besedilu nista enaka. Vsi istoležni predhodni znaki so bili prej paroma enaki. Velja torej:

$$x[i + 1 .. m - 1] = y[i + j + 1 .. j + m - 1] = u$$

in

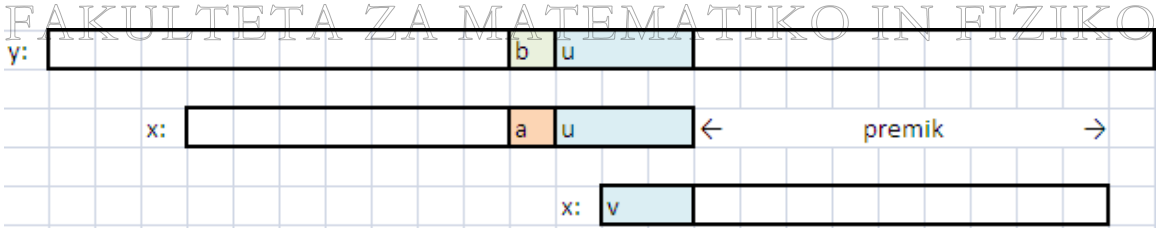
$$x[i] \neq y[i + j].$$

Pravilo dobre pripone v vzorcu poišče ponoven sklop znakov u, pred katerim ne stoji znak enak znaku $x[i] = a$. Če je takšnih sklopov v vzorcu več, uporabi najbolj desnega (glej primer 6).



Slika 11: Pravilo dobre pripone v vzorcu poišče sklop znakov, ki se je z besedilom že ujemal

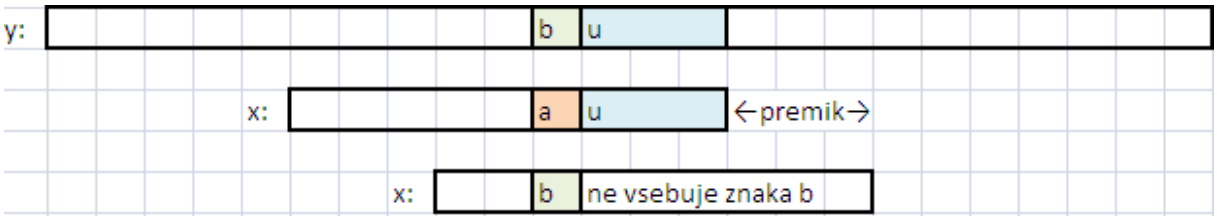
Če pravilo dobre pripone takšnega sklopa ne najde, poravna z besedilom pripono vzorca v. Del omenjene pripone je enak že ujemajočem se sklopu u (glej primer 7).



Slika 12: Pravilo dobre pripone v vzorcu poišče del sklopa znakov, ki se je z besedilom že ujemal

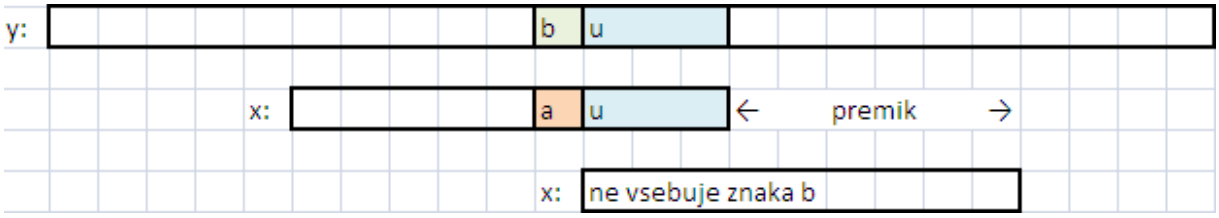
Pravilo slabega znaka (bad-character rule)

To pravilo pride v poštev vedno, ko naletimo na par različnih znakov. Takrat vzorec premaknemo za toliko mest v desno, da se poravna najbolj desni znak iz vzorca na mestih *x[0 .. m – 2]*, ki se ujema z znakom *y[i + j]* (glej primer 3).



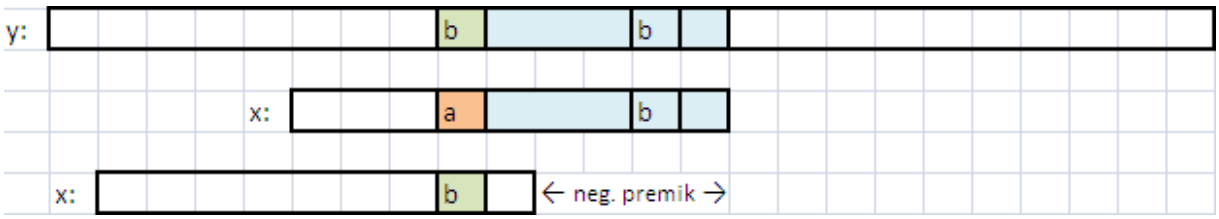
Slika 13: Pravilo slabega znaka v vzorcu poišče znak, ki se je z besedilom že ujemal

Če se znak *y[i + j]* v vzorcu ne pojavi, se slednji v celoti premakne na položaj za omenjenim znakom. Torej je znak *x[0]* pod znakom *y[i + j + 1]* (glej primer 1).



Slika 14: Pravilo slabega znaka v vzorcu ne najde znaka, ki je povzročil neujemanje, zato premakne vzorec za znak neujemanja

Obstaja možnost, da pravilo slabega znaka predlaga negativen premik (glej tudi primer 6).



Slika 15: Pravilo slabega znaka predlaga negativen premik

To bi bil pri iskanju vzorca v besedilu negativen korak in ga seveda ne naredimo.

V splošnem se lahko zgodi, da pravili predlagata različna premika. Takrat seveda izberemo boljšega, torej tistega, ki nam vzorec bolj premakne v desno.

Zato algoritem BM glede na pravilo dobre pripone in pravilo slabega znaka tvori dve tabeli: **bmGs** in **bmBc**. Tabeli vsebujeta vrednosti premikov pri določenem položaju (neujemanje/popolno ujemanje) v oknu. Pravilo dobre pripone predlaga premik, ki je enak vrednosti v tabeli bmGs. Vrednost premika iz tabele bmBc je treba še preračunati (glej enačba 1). Dobimo premik, ki ga predlaga pravilo slabega znaka. Oba od predlaganih premikov delujeta individualno in seveda omogočata pravilno iskanje. Ker vemo, da nobeden izmed njiju ne izpusti rešitve, algoritem BM vedno izbere večji premik.

Premik s pomočjo pravila dobre pripone imenujemo tudi **premik ob dobri priponi**, premik s pomočjo pravila slabega znaka pa imenujemo **premik ob slabem znaku**.

Gradnja tabel bmGs in bmBc

bmGs

Pravilo dobre pripone tvori tabelo **bmGs**, v kateri so shranjene vrednosti oziroma premiki za vse položaje znakov v vzorcu. Premik za vrednost iz tabele bmGs se zgodi na tistem položaju, kjer dobimo neujemanje ali pa pri prvem znaku vzorca ob popolnem ujemanju (vzorec nastopa v besedilu). Pravilo dobre pripone določi, kakšen del vzorca se je že ujemal z besedilom desno od trenutnega znaka (dobra pripona) in ali takšen del v vzorcu še kdaj nastopa na drugih mestih. Pred pripono in pred najdenim enakim sklopom znakov morata biti različna znaka, saj bi bil premik v nasprotnem primeru nesmiseln. Enak del znakov je treba premakniti pod tiste, ki so se že ujemali. Pred tem mora biti različen znak od tistega, ki je povzročil neujemanje. Podobno deluje pravilo slabega znaka (natančnejši opis spodaj), ko znak iz besedila iščemo v vzorcu. Če ga najdemo, enaka znaka postavimo enega pod drugega. Razlika je le, da pri pravilu dobre pripone, levo od neujemanja iščemo sklop znakov, ki se je z besedilom že ujemal. Če je v vzorcu takšnih sklopov več, se uporabi najbolj desni (glej Slika 11). V nasprotnem primeru lahko izgubimo rešitev, kar je še ena podobnost s pravilom slabega znaka (opis spodaj).

Pravilo dobre pripone lahko v vzorcu najde le del pripone, ki se je z besedilom ujemala. To je v primeru, ko je del začetnih znakov vzorca enak tistim na koncu, ki so se z besedilom že ujemali (ujemajoči se del je lahko večji od dela na začetku vzorca). V tem primeru se z besedilom poravna ujemajoči se del znakov (glej slika 12).

Vrednost tabele pri prvem znaku vzorca je enaka periodi vzorca ($per(x) = p$). Če se je neenakost zgodila šele pri prvem znaku, vemo, da se je preostali del vzorca z besedilom ujemal. To pomeni, da lahko prvi znak vzorca v besedilu nastopa za p mest bolj desno (če $p < m$). Potem tudi znaki desno od prvega nastopajo za p mest bolj desno. Zato se lahko z vzorcem premaknemo na ponovno pojavitev prvega znaka (kar je ravno za periodo). Premaknemo se pod del že ujemajoče se pripone. Če se prvi znak iz vzorca v nadaljevanju ne pojavi več, je vrednost periode enaka dolžini vzorca ($p = m$). Tudi v tem primeru je premik za p mest desno pravilen, saj se premaknemo za desni rob vzorca v trenutnem poskusu. Tudi ob popolnem ujemanju se zgodi, glede na prvi znak vzorca, premik za p. Prvi znak vzorca se namreč lahko ponovi v nadaljevanju vzorca za p mest desno ali pa se ne ponovi in je p enak m. Zato vzorec premaknemo za periodo in nadaljujemo z naslednjim poskusom.

Definicija tabele bmGs:

Tabela bmGs je velikosti m. Definirana sta 2 pogoja (ang. condition):

Cs(i, s): za vsak k, za katerega velja $i < k < m$, $\boldsymbol{s \geq k}$ *ali* $\boldsymbol{x[k-s] = x[k]}$,

Co(i, s): če $s < i$ *potem* $\boldsymbol{x[i-s] \neq x[i]}$

Cs … pogoj »pripone«. V vzorcu išče del znakov, ki ustrezajo priponi vzorca.

Co … pogoj neenakosti. V vzorcu išče znak, ki stoji pred delom, ki ustreza priponi, ta znak pa je različen tistemu pred pripono.

Potem velja za $0 < i < m$:

$bmGs[i + 1] = min\{s > 0;$ drži $Cs(i, s)$ in $Co(i, s)\}$

$bmGs[0] = per(x)$.

Opomba: Pri praktični uporabi algoritma (v različnih programih) je v kodi definirana še tabela suff. Z njeno pomočjo pravilo dobre pripone tvori tabelo bmGs, vendar je za osnovno razumevanje algoritma njena definicija nepotrebna. Njen opis in delovanje v kodi programa lahko najdete v knjigi C. Charras, T. Lecroq, Exact string matching algorithms.

bmBc

Pravilo slabega znaka tvori tabelo **bmBc**:

i. Znakom, ki nastopajo v besedilu, ne pa v vzorcu, določi za vrednost premika *m*, saj je premik vzorca v tem primeru predlagan na mesto za neujemanjem.
ii. Znakom, ki nastopajo v vzorcu (lahko tudi v besedilu), določi za vrednost premika obratno vrednost položaja v vzorcu (na primer predzadnji znak ima vrednost *1*, prvi znak *m-1*). Če je enakih znakov v vzorcu več, upošteva vrednost najbolj desnega znaka (izbere najmanjšo vrednost). Opomba: Izjema je zadnji znak vzorca. Pri prvem primerjanju v poskusu je ob neenakosti zadnji znak vzorca nepomemben, saj se znak iz besedila nad njim išče v nadaljevanju vzorca. Če enak znak v nadaljevanju obstaja, mu tabela bmBc določi vrednost predzadnje pojavitve (zadnja pojavitev je zadnji znak vzorca). Če takšen znak v nadaljevanju ne obstaja, mu tabela bmBc določi vrednost *m*.

Definicija tabele bmBc:

Za $c \in \Sigma$ (spomnimo se, da Σ označuje celotno abecedo, torej vse znake, ki sestavljajo vzorec in besedilo) velja:

$$bmBc[c] = \begin{cases} min\{i: 1 \leq i < m - 1 \text{ in } x[m-1-i] = c, \text{če se } c \text{ pojavi v } x, \\ m \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{sicer.} \end{cases}$$

Določena vrednost iz tabele bmBc še ne pomeni premika, saj je slednji odvisen predvsem od tega, kje ob neenakosti znak v vzorcu nastopa. Vzorec premaknemo bodisi pod bodisi za znak neujemanja. Ker lahko v različnih poskusih dva enaka znaka nastopata na različnih mestih v vzorcu, lahko pravilo slabega znaka predlaga dva različna premika. Zato dokončnih premikov ne moremo dati v tabelo bmBc.

OPOMBA: Predlagan premik s pravilom slabega znaka je potrebno določiti z **enačbo premika**:

Enačba 1: Enačba premika

$$\boldsymbol{Premik = bmBc[c] - m + (i + 1)}$$

bmBc[c] … vrednost v tabeli bmBc pri znaku *c*

m … dolžina vzorca

i … mesto neujemanja v vzorcu

Enačba 1 ob neujemanju znakov predlaga premik vzorca. Premakne ga bodisi pod znak iz besedila, kjer imamo neujemanje (če ta znak v vzorcu nastopa; glej slika 16) bodisi za njim (če ta znak v vzorcu ne nastopa; glej slika 17).

Opomba: V primerih spodaj je uporabljena tabela bmBc. Gradnja tabele na primerih je prikazana v razdelku bmBc na strani 26.

c	a	e	h	i	j	k	m	n	o	p	r
bmBc[c]	5	1	5	5	5	5	5	5	3	2	5

h	a	r	m	o	n	i	j	a
p	o	p	e	k				

Premik = bmBc[o] - m + (i + 1) = 3 - 5 + 5 = 3

h	a	r	m	o	n	i	j	a
			p	o	p	e	k	

Slika 16: Premik ob slabem znaku (znak ob neujemanju nastopa v nadaljevanju vzorca)

c	a	b	e	j	k	l	n	o	s
bmBc[c]	4	4	4	4	3	1	4	2	4

o	b	s	o	j	a	n	j	e
k	o	l	o					

Premik = bmBc[s] - m + (i + 1) = 4 - 4 + 3 = 3

o	b	s	o	j	a	n	j	e
			k	o	l	o		

Slika 17: Premik ob slabem znaku (znak ob neujemanju ne nastopa v nadaljevanju vzorca)

Primeri

bmGs

Legenda:		ujemanje
	X	znak ≠ X

Slika 18: Legenda

Primer 8

VZOREC: "poznana"

Iščemo sklop znakov, enak koncu vzorca, pri čemer morata znaka pred ujemajočima se deloma biti različna. Ker gradimo tabelo za zadnji znak vzorca, je do sedaj ujemajoči se sklop prazen. V vzorcu torej iščemo prazen sklop, znak na levi pa ne sme biti enak zadnjemu znaku vzorca. Črko, ki ni enaka zadnji, najdemo že na predzadnjem mestu vzorca ('n' ≠ 'a'). Torej je premik pri zadnji črki vzorca enak 1; bmGs[6] = 1 (glej slika 19).

						a
p	o	z	n	a	n	a
						1

Slika 19: Primer 8 (bmGs[6])

Sedaj iščemo premik pri ujemanju s črko 'a', znak za eno mesto v levo pa ne sme biti enak 'n'-ju. Da bi se lahko »poravnali« z 'a', pride v poštev le drugi 'a' z desne. Ker pa je pred njim 'n', to ni dobro, saj potrebujemo znak, različen od 'n'. Zato se po definiciji premaknemo za toliko mest v levo, da s sklopom pridemo izven levega roba vzorca. Premik pri x[5] je torej enak 7 (glej slika 20).

					n	a
p	o	z	n	a	n	a
					7	1

Slika 20: Primer 8 (bmGs[5])

Naslednje ujemanje mora biti enako sklopu "an", znak na levi pa ne sme biti enak 'a'-ju. Takšen sklop najdemo za dve mesti bolj levo v vzorcu. Zato je vrednost tabele bmGs pri znaku x[4] enaka 2, kar pomeni, da se v tem primeru premaknemo za dve mesti v desno (glej slika 21).

				a	n	a
p	o	z	n	a	n	a
				2	7	1

Slika 21: Primer 8 (bmGs[4])

Iščemo sklop "ana", znak na levi ne sme biti enak 'n'-ju. Sklop se v vzorcu več ne pojavi. Premik pri x[3] je 7 (glej slika 22).

			n	a	n	a
p	o	z	n	a	n	a
			7	2	7	1

Slika 22: Primer 8 (bmGs[3])

Tudi sklopa "nana" v nadaljevanju vzorca ni več, seveda tudi različne črke pred tem sklopom ne najdemo. Tudi premik pri x[2] je 7 (glej slika 23).

		z	n	a	n	a
p	o	z	n	a	n	a
		7	7	2	7	1

Slika 23: Primer 8 (bmGs[2])

V vzorcu več ne nastopa vzorec "znana" in pred njim različna črka od 'o'. Premik pri x[1] je ponovno 7 (glej slika 24).

	o	z	n	a	n	a
p	o	z	n	a	n	a
	7	7	7	2	7	1

Slika 24: Primer 8 (bmGs[1])

Takšen premik velja tudi za prvi znak vzorca (bmGs[0] = 7). BmGs je tako tudi definirana, saj velja za bmGs[0] = dolžina periode vzorca (glej slika 25). Ker se prvi znak vzorca v nadaljevanju več ne ponovi, je dolžina periode vzorca enaka dolžini vzorca (*per(x) = m*).

p	o	z	n	a	n	a
p	o	z	n	a	n	a
7	7	7	7	2	7	1

Slika 25: Primer 8 (bmGs[0])

Dobimo tabelo bmGs za besedo "poznana" (glej slika 26):

i	0	1	2	3	4	5	6
x[i]	p	o	z	n	a	n	a
bmGs[i]	7	7	7	7	2	7	1

Slika 26: Primer 8, tabela bmGs

Primer 9

VZOREC: "zlodelo"

Zgradimo tabelo bmGs še za besedo "zlodelo" (glej slika 27). Podrobnejše postopka ne bomo komentirali, saj je povsem podoben prejšnjemu zgledu.

1							o	**5**			o	d	e	l	o		
	z	l	o	d	e	l	o		z	l	o	d	e	l	o		
							1				7	7	4	7	1		
2						l	o	**6**		l	o	d	e	l	o		
	z	l	o	d	e	l	o		z	l	o	d	e	l	o		
						7	1			7	7	7	4	7	1		
3					e	l	o	**7**	z	l	o	d	e	l	o		
	z	l	o	d	e	l	o		z	l	o	d	e	l	o		
					4	7	1		7	7	7	7	4	7	1		
4				d	e	l	o	**bmGs:**									
	z	l	o	d	e	l	o		i		0	1	2	3	4	5	6
				7	4	7	1		x[i]		z	l	o	d	e	l	o
									bmGs[i]		7	7	7	7	4	7	1

Slika 27: Primer 9, tabela bmGs

Primer 10

VZOREC: "baraba"

Pri besedi "baraba" je stvar za odtenek drugačna. Tu poleg dveh ujemajočih se sklopov z različnima črkama pred njima najdemo še del ujemajočega se sklopa na začetku vzorca. To pomeni, da ima beseda "baraba" periodo, manjšo od *m*. Če torej dveh enakih sklopov z različnima znakoma spredaj v vzorcu ne najdemo, je poravnava možna z delom ujemajočega se sklopa, kar pomeni premik za vrednost periode (glej slika 12).

Iščemo različno črko od 'a' pred praznim sklopom. Najdemo jo za eno mesto bolj levo. Premik pri x[5] je 1 (glej slika 28).

					a
b	a	r	a	b	a
					1

Slika 28: Primer 10 (bmGs[5])

Sedaj potrebujemo ujemanje s črko 'a', črka bolj levo mora biti različna od 'b'. Takšen sklop najdemo za dve mesti bolj levo, ko najdemo sklop "ra". Premik pri x[4] je 2 (glej slika 29).

				b	a
b	a	r	a	b	a
				2	1

Slika 29: Primer 10 (bmGs[4])

Naslednji korak išče sklop "ba" in pred njim črko, ki ni 'a'. Takšne kombinacije v tem primeru ni več, saj ob enakem sklopu "ba" pridemo na začetek besede. Pred začetkom besede pa neujemanje s črko 'a' več ni mogoče. Vseeno je poravnava ujemajočih se sklopov pravilna. Premik pri x[3] je 4 (glej slika 30).

			a	b	a
b	a	r	a	b	a
			4	2	1

Slika 30: Primer 10 (bmGs[3])

Sledi iskanje sklopa "aba" in pred njim od 'r' različne črke. Tega ne najdemo, najdemo pa del ujemajočega se sklopa "ba", in sicer na začetku vzorca. Premik pri x[2] je 4 (glej slika 31).

		r	a	b	a
b	a	r	a	b	a
		4	4	2	1

Slika 31: Primer 10 (bmGs[2])

Enaka zadeva je pri naslednjem iskanju. Premik pri x[1] je 4 (glej slika 32).

	a	r	a	b	a
b	a	r	a	b	a
	4	4	4	2	1

Slika 32: Primer 10 (bmGs[1])

In še premik pri x[0] je 4 (glej slika 33), ki je po definiciji enak periodi vzorca. Vrednost periode vzorca ($per(x) = p$) določimo z $x[0] = x[0 + p]$, kar pomeni, da je p enak 4.

b	a	r	a	b	a
b	a	r	a	b	a
4	4	4	4	2	1

Slika 33: Primer 10 (bmGs[0])

Dobimo tabelo bmGs za besedo "baraba" (glej slika 34).

i	0	1	2	3	4	5
x[i]	b	a	r	a	b	a
bmGs[i]	4	4	4	4	2	1

Slika 34: Primer 10, tabela bmGs

Podobno velja za besedo "antiroman" (glej slika 35).

Primer 11

VZOREC: "antiroman"

1									n
	a	n	t	i	r	o	m	a	n
									1
2								a	n
	a	n	t	i	r	o	m	a	n
								9	1
3							m	a	n
	a	n	t	i	r	o	m	a	n
							7	9	1
4						o	m	a	n
	a	n	t	i	r	o	m	a	n
						7	7	9	1
5					r	o	m	a	n
	a	n	t	i	r	o	m	a	n
					7	7	7	9	1

6				i	r	o	m	a	n
	a	n	t	i	r	o	m	a	n
				7	7	7	7	9	1
7			t	i	r	o	m	a	n
	a	n	t	i	r	o	m	a	n
			7	7	7	7	7	9	1
8		n	t	i	r	o	m	a	n
	a	n	t	i	r	o	m	a	n
		7	7	7	7	7	7	9	1
9	a	n	t	i	r	o	m	a	n
	a	n	t	i	r	o	m	a	n
	7	7	7	7	7	7	7	9	1

i	0	1	2	3	4	5	6	7	8
x[i]	a	n	t	i	r	o	m	a	n
bmGs[i]	7	7	7	7	7	7	7	9	1

Slika 35: Primer 11, tabela bmGs

Primer 12

VZOREC: "abcxxx"

Do sedaj smo imeli pri vseh primerih tabele bmGs vrednost zadnjega znaka 1 (torej bmGs[*m-1*] = 1). V naslednjem primeru temu ne bo tako, saj se vzorec konča s tremi enakimi znaki. Zato pri gradnji tabele pri koraku 1 različno črko od zadnje najdemo šele tri mesta bolj v levo. Premik pri x[5] je torej 3 (glej slika 36).

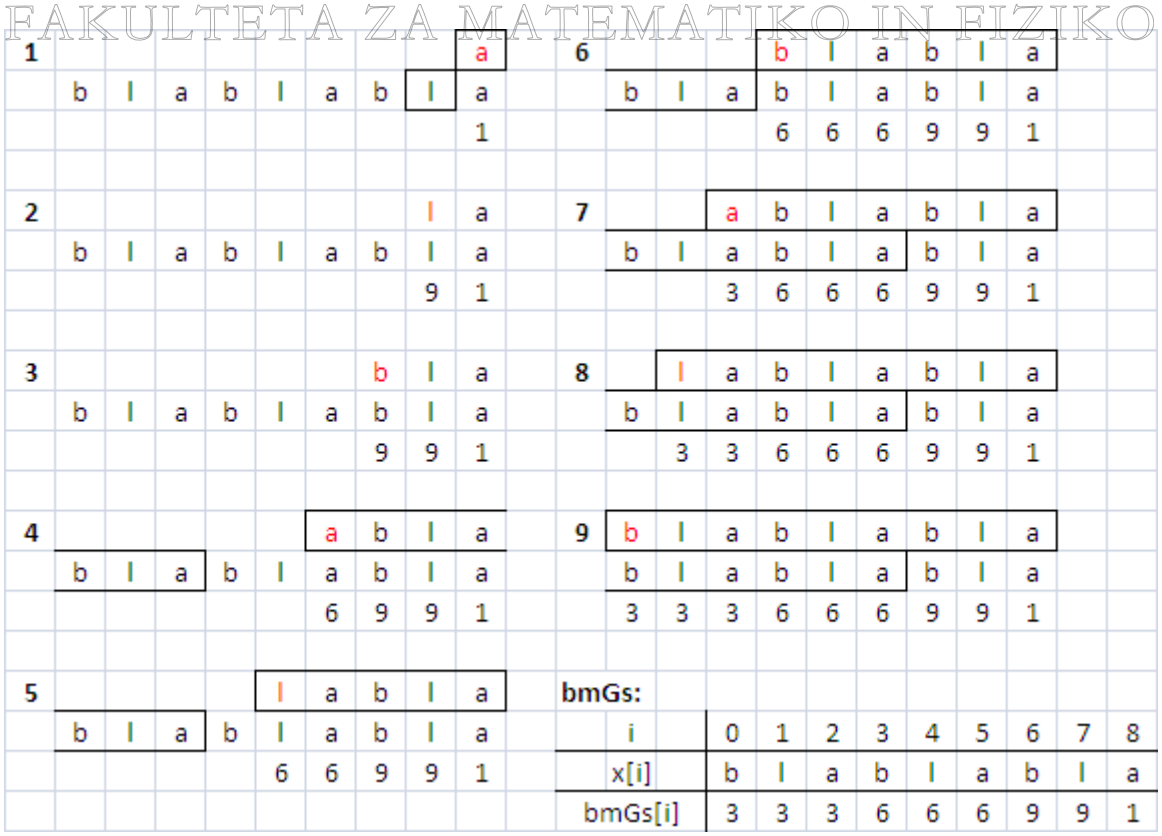
1						x		**4**			c	x	x	x
	a	b	c	x	x	x			a	b	c	x	x	x
						3					6	1	2	3
2					x	x		**5**		b	c	x	x	x
	a	b	c	x	x	x			a	b	c	x	x	x
					2	3				6	6	1	2	3
3				x	x	x		**6**	a	b	c	x	x	x
	a	b	c	x	x	x			a	b	c	x	x	x
				1	2	3			6	6	6	1	2	3
			bmGs:											
			i		0	1	2	3	4	5				
			x[i]		a	b	c	x	x	x				
			bmGs[i]		6	6	6	1	2	3				

Slika 36: Primer 12, tabela bmGs

Primer 13

VZOREC: "blablabla"

Pri besedi "blablabla" se pri premiku beseda poravna s svojo predpono (glej slika 12). Ugotovimo, da ima vzorec periodo 3, tako da je pri prvih treh znakih premik enak periodi (glej slika 37).



Slika 37: Primer 13, tabela bmGs

bmBc

Primer 14

BESEDILO: "katapult"

VZOREC: "sonce"

Najprej zapišemo vse znake, ki nastopajo v besedilu in vzorcu (najboljše kar po abecednem vrstnem redu). Črka 'a' v vzorcu ne nastopa, zato dobi vrednost 5, kar je dolžina vzorca "sonce". Če bomo namreč pri primerjanju v besedilu naleteli na črko 'a', se očitno lahko s celotnim vzorcem premaknemo desno od mesta neujemanja, saj poravnave med vzorcem in delom besedila prej ne bomo našli. Črka 'c' stoji na predzadnjem mestu v vzorcu, zato je vrednost pri njej enaka 1. Črka 'e' je zadnji znak vzorca (v njem je, razen na tem mestu, ni več), zato je vrednost pri 'e' enaka dolžini vzorca, torej 5. Črke 'k', 'l', 'p', 't' in 'u' v vzorcu ne nastopajo, zato prav tako dobijo vrednost 5. Znak 'n' stoji na drugem mestu od zadaj, zato ima vrednost 2. Na enak način imata črki 'o' in 's' vrednosti 3 in 4.

	c		a	c	e	k	l	n	o	p	s	t	u
bmBc[c]			5	1	5	5	5	2	3	5	4	5	5

Slika 38: Primer 14, tabela bmBc

Primer 15

BESEDILO: "Mali orangutan."

VZOREC: "orangutan"

Privzemimo, da med primerjanjem vzorca in besedila ločimo velike od malih črk. Kot samostojni znaki so mišljena tudi ločila ('.', ',', ' ' (presledek) …). Črka 'n' nastopa v vzorcu na dveh mestih, na četrtem in zadnjem mestu. Vemo, da zadnji znak v vzorcu iščemo še v preostalem delu. Če ga najdemo, uporabimo njegovo najbolj desno pojavitev (dobi vrednost položaja od zadnjega znaka). Kadar ga ne najdemo, uporabimo vrednost dolžine vzorca. Črka 'n' nastopa v vzorcu na tretjem mestu, če pa gledamo položaj iz obratne smeri, pa na petem mestu. Zato dobi v tabeli bmBc vrednost 5. Črke 'n', ki nastopa na zadnjem mestu vzorca, ne upoštevamo. Črka 'a' se v vzorcu pojavi na mestih 3 in 8. Uporabi se bolj desni 'a', ki ima iz obratne smeri v vzorcu položaj 1. Znak 'a' dobi v tabeli bmBc vrednost 1.

c	a	g	i	l	M	n	o	r	t	u		.
bmBc[c]	1	4	9	9	9	5	8	7	2	3	9	9

Slika 39: Primer 15, tabela bmBc

Primer 16

BESEDILO: "Anže joče"

VZOREC: "abcčd"

Gradnjo in razlago prepustimo bralcu. Dobljena tabela mora biti takšna, kot jo prikazuje slika 40.

c	a	A	b	c	č	d	e	j	n	o	ž	
bmBc[c]	4	5	3	2	1	5	5	5	5	5	5	5

Slika 40: Primer 16, tabela bmBc

S tem smo si ogledali način, kako tvorimo ustrezni tabeli. Potrebovali ju bomo pri izvajanju algoritma.

Postopek iskanja

Do sedaj smo spoznali pravila, ki jih algoritem BM potrebuje za iskanje vzorca v besedilu. Skupek teh pravil pa omogoča iskanje vzorca v besedilu. Poglejmo, kako poteka postopek iskanja. Spodaj je poenostavljen zapis javanske kode.

```
IskanjeBoyerMoore(niz y, niz x){//algoritem dobi v obdelavo besedilo in vzorec
        bmGs(x)//metoda bmGs (pravilo dobre pripone) iz znakov vzorca tvori tabelo bmGs
        bmBc(x, y)// metoda bmBc (pravilo slabega znaka) iz znakov vzorca in besedila tvori tabelo bmBc
        j = 0//vrednost premikanja okna po besedilu
        poravnava(y+j, x){//levi rob vzorca se pri prvem poskusu poravna z levim robom besedila, v nadaljnjih
poskusih se vzorec premakne na »j«-ti položaj v besedilu
                i = 1//vrednost premikanja po vzorcu (v smeri od zadaj proti naprej)
                primerjava(j, m-i){
                        enakost(y.znakPri(j+m-i) == x.znakPri(m–i))//znaka, ki se primerjata sta enaka
                                i++//i se poveča za 1, sledi primerjava znakov za 1 mesto bolj levo
                                če(i==m)//prišli smo do levega roba vzorca
                                        označi(y.odDo(j, j+m-i))//dobimo popolno ujemanje
```

```
                    poravnava(y+bmGs[0], x)//premik vzorca v desno za vrednost bmGs[0]
                    primerjava(j, m-i)//ponovi se postopek primerjave (tokrat za mesto bolj levo)
                neenakost
                    j = MAX(bmGs[m-i], bmBc[y.znakPri(j+m-i)]–m+1+i)//pogledamo katero pravilo predlaga večji premik (vrednost iz bmBc je preračunana še z enačbo premika)
                    poravnava(y+j, x)//premik vzorca v desno in ponovno iskanje
                če(j+m > y) konec//če desni rob premaknjenega vzorca pride desno od besedila
            }
        }
}
```

Časovna in prostorska zahtevnost algoritma

V fazi predobdelave podatkov algoritem BM tvori dve tabeli. Časovna zahtevnost tega dela je $O(m+\sigma)$, prostorska zahtevnost pa služi shranjevanju obeh tabel in je velikosti $O(m+\sigma)$. Spomnimo se, da je m dolžina vzorca, σ pa velikost abecede (torej število znakov, ki nastopajo v besedilu in vzorcu).

V fazi iskanja vzorca v besedilu je časovna zahtevnost kvadratična ($O(n^2)$), vendar je število primerjav znakov pri neperiodičnih vzorcih največ $3n$.

Same analize in razlaga zahtevnosti so dokaj zapletene in presegajo obseg in nivo te diplomske naloge.

Primeri – reševanje z algoritmom

V tem poglavju bomo prikazali delovanje algoritma BM na nekaj različnih primerih. Pri tem bomo v nasprotju s prejšnjimi primeri, ko smo pokazali, zakaj je prišlo do določenih premikov, preprosto pokazali, kaj se dogaja, če sledimo algoritmu.

Primer 17

BESEDILO: "Slovenska himna je dobra himna."

VZOREC: "himna"

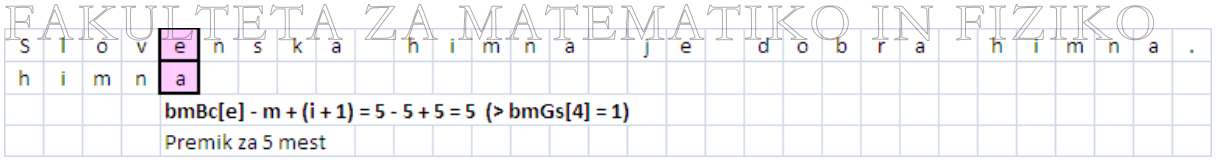
Najprej določimo premike, ki jih bomo potrebovali pri iskanju vzorca v besedilu. Zato tvorimo tabeli bmGs in bmBc (glej slika 41).

i	0	1	2	3	4
x[i]	h	i	m	n	a
bmGs[i]	5	5	5	5	1

c	a	b	d	e	h	i	j	k	l	m	n	o	r	s	S	v	.	
bmBc[c]	5	5	5	5	4	3	5	5	5	2	1	5	5	5	5	5	5	5

Slika 41: Primer 17, tabeli bmGs in bmBc

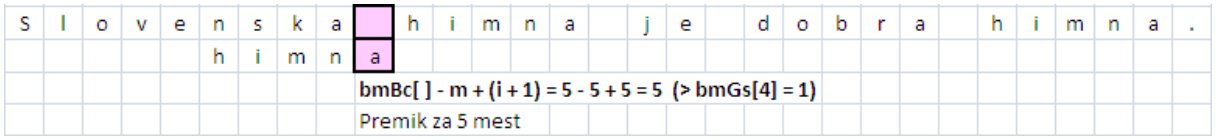
Sledi postopek iskanja vzorca v besedilu, prvi poskus (glej slika 42). Dolžina vzorca je 5.



Slika 42: Primer 17, prvi poskus

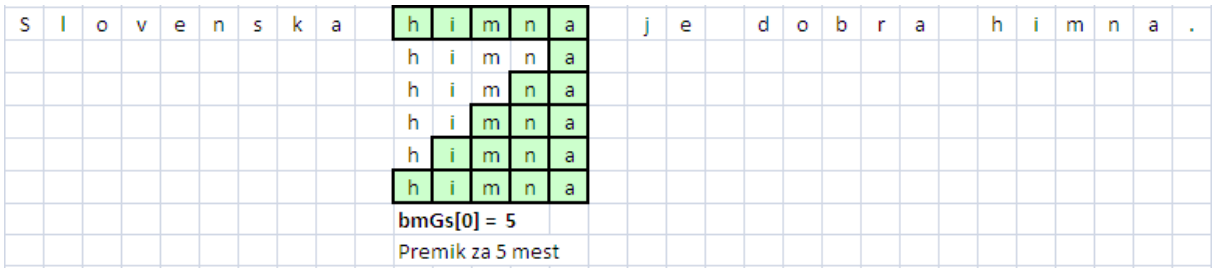
Opazimo neujemanje med črkama 'e' iz besedila in 'a' iz vzorca. Pogledamo možne premike. Premik s pomočjo pravila dobre pripone predlaga premik za eno mesto. »Ujema« se prazen sklop znakov, iščemo različno črko od 'a', najdemo jo na položaju za eno mesto bolj levo. Pravilo slabega znaka predlaga premik za pet mest. Vrednost pri črki neujemanja iz besedila, to je 'e', je 5. Po formuli premika (glej enačba 1) moramo tej vrednosti odšteti dolžino vzorca in prišteti mesto neujemanja v vzorcu. Torej dobimo: bmBc[e] – m + (i + 1) = 5 – 5 + 5 = 5. Premik s pomočjo bmBc je večji od bmGs, zato se z vzorcem premaknemo za pet mest.

Drugi poskus (glej elika 43):



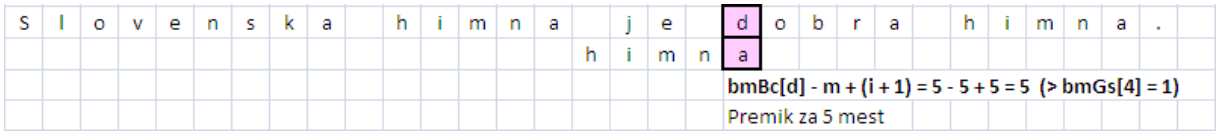
Slika 43: Primer 17, drugi poskus

Tokrat dobimo neenakost med znakoma ' ' (presledek) in 'a'. Predlagani premik s pravilom dobre pripone je manjši od premika s pomočjo pravila o slabem znaku, zato izberemo slednjega. Pri tretjem poskusu (glej slika 44) najdemo ujemanje celotnega vzorca. Če uporabljamo različico, ki išče le eno ujemanje, smo zaključili. Če pa nas zanima, ali se vzorec pojavlja še kje v besedilu, nadaljujemo. Pravilo slabega znaka išče neujemanja, zato pri ujemanju vseh znakov v poskusu ne predlaga premika. Kadar se torej ves vzorec ujema, sledi premik za vrednost pri bmGs[0].



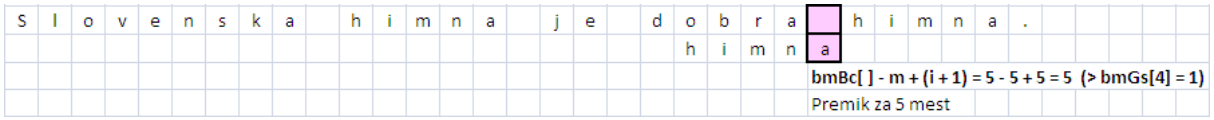
Slika 44: Primer 17, tretji poskus

Nadaljujemo s četrtim poskusom (glej slika 45).



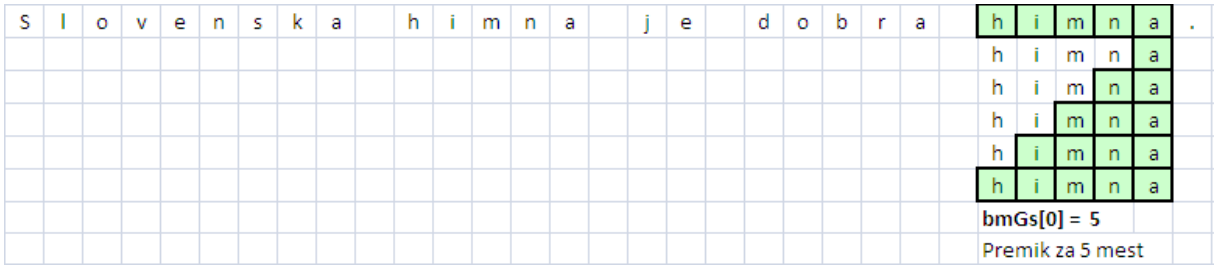
Slika 45: Primer 17, četrti poskus

Stvar je podobna poskusoma 1 in 2, kar pa velja tudi za peti poskus (glej slika 46).



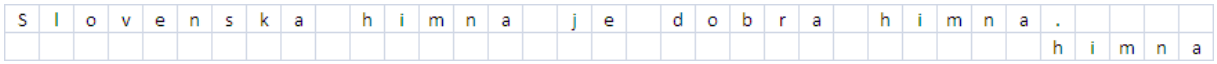
Slika 46: Primer 17, peti poskus

Šesti poskus (glej slika 47) je zelo podoben tretjemu, saj ponovno najdemo vzorec. Sledil bo torej premik za bmGs[0].



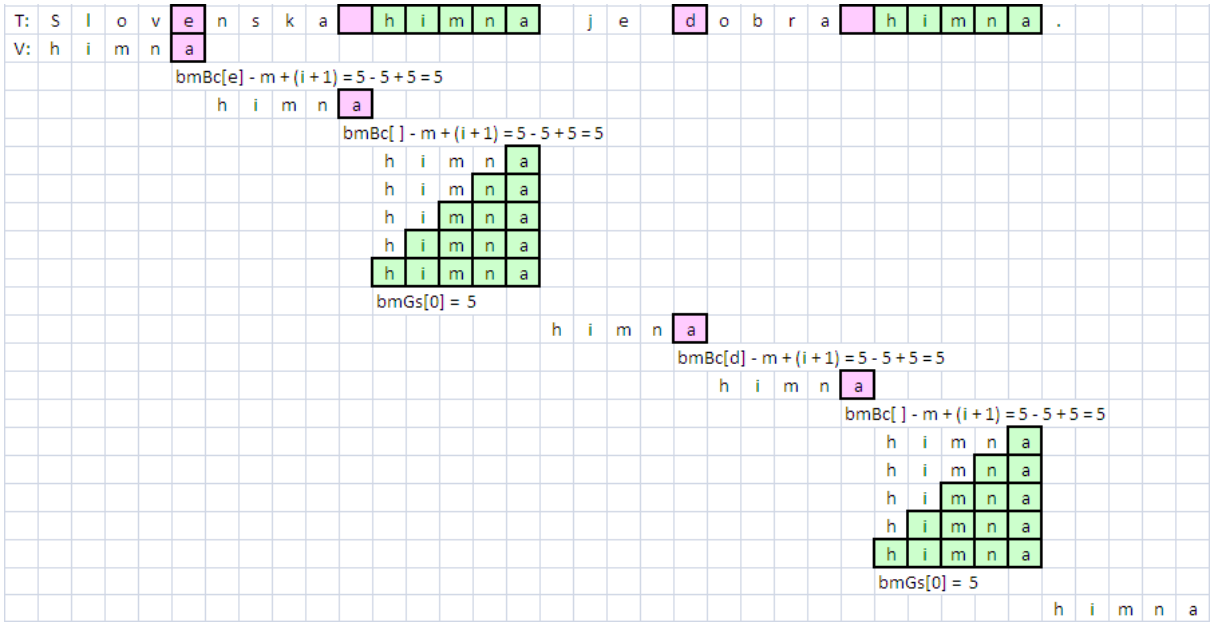
Slika 47: Primer 17, šesti poskus

Po tem premiku se desni rob drsnega okna vzorca premakne bolj desno od desnega roba besedila, zato z iskanjem vzorca v besedilu prenehamo (glej slika 48).



Slika 48: Primer 17, zaključek iskanja

Navedimo ponovno še celoten postopek iskanja (glej slika 49).



Slika 49: Primer 17, celoten postopek iskanja

Osnovni podatki primera 17:

- dolžina besedila … 31
- dolžina vzorca … 5
- število poskusov … 6
- število primerjav znakov … 14

Primer 18

BESEDILO: "uhanast"

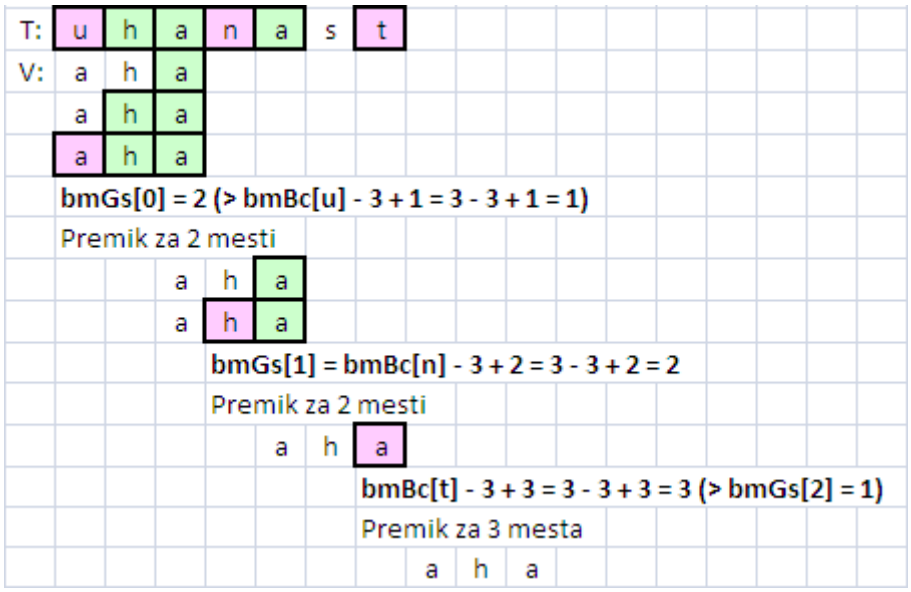
VZOREC: "aha"

Najprej naredimo tabeli bmGs in bmBc (glej slika 50).

i	0	1	2			
x[i]	a	h	a			
bmGs[i]	2	2	1			
c	a	h	n	s	t	u
bmBc[c]	2	1	3	3	3	3

Slika 50: Primer 18, tabeli bmGs in bmBc

Poglejmo postopek iskanja vzorca v besedilu (glej slika 51).



Slika 51: Primer 18, celoten postopek iskanja

Osnovni podatki primera 18:

- dolžina besedila … 7
- dolžina vzorca … 3

- število poskusov … 3
- število primerjav znakov … 6

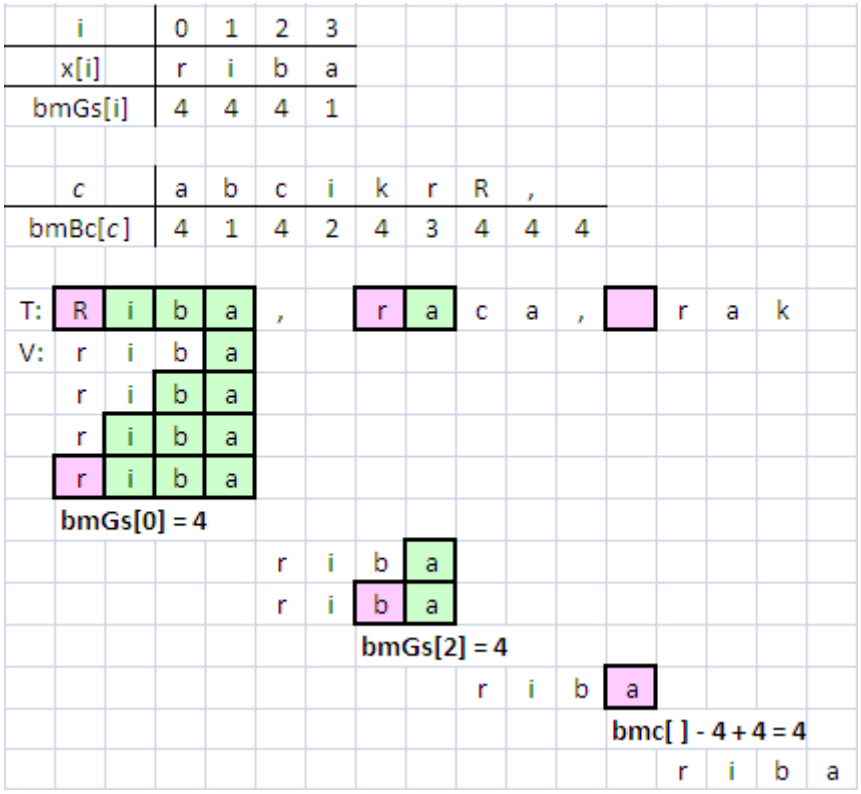
V zgornjem primeru vidimo, kako algoritem BM izbira med dvema funkcijama premika. Pri prvem poskusu se zgodi premik, predlagan s pravilom dobre pripone, saj je večji od premika s pomočjo pravila slabega znaka. Pri poskusu 2 sta premika obeh funkcij enaka. Pri tretjem poskusu pa se premaknemo glede na premik s pomočjo pravila slabega znaka, saj le-ta predlaga največji premik.

Opazimo, da takrat, kadar se ujemata vsaj prva znaka, prevladuje premik, predlagan s pravilom dobre pripone. Če pa sta v poskusu že prva primerjana znaka različna, običajno večji premik omogoči pravilo slabega znaka.

Primer 19

BESEDILO: "Riba, raca, rak."

VZOREC: "riba"



Slika 52: Primer 19, celoten postopek iskanja

Osnovni podatki primera 19:

- dolžina besedila … 15
- dolžina vzorca … 4
- število poskusov … 3
- število primerjav znakov … 7

Kot že rečeno, algoritem BM najprej tvori tabelo bmBc, v kateri nastopajo vsi znaki vzorca in besedila. Tako je tudi v zgornjem primeru. Ker opisujemo različico algoritma, ki loči majhne od velikih črk, sta v tabeli bmBc 'r' in 'R' zapisana kot različna znaka. To je glavni vzrok, da v prvem poskusu ne dobimo popolnega ujemanja. Pri ostalih dveh poskusih se je potrdil komentar iz prejšnjega primera (ob neujemanju prvega znaka v poskusu prevladuje premik s pomočjo pravila slabega znaka, ob vsaj enem ujemanju v poskusu prevladuje premik s pravilom dobre pripone).

2. RAZLIČICE ALGORITMA BM

HORSPOOLOV ALGORITEM

Po objavi Boyer-Mooreovega algoritma je nastalo več izpeljank tega algoritma. Avtorji so z njimi želeli optimizirati reševanje problemov z različnimi lastnostmi. Algoritmi se med seboj ločijo po prostorskih zahtevah, različna so števila znakovnih primerjav itd. Eden prvih praktično uporabnih algoritmov je bil Horspoolov algoritem (algoritem HP) iz leta 1980, torej tri leta za algoritmom BM.

Osnovne lastnosti in primerjava z algoritmom BM

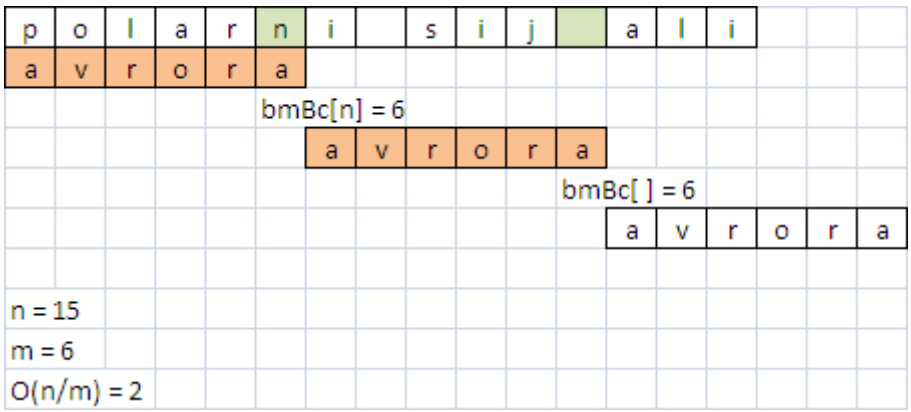
Gre za poenostavljeno različico algoritma Boyer-Moore. Algoritem HP uporablja le pravilo slabega znaka (glej stran 16). Pravilo dobre pripone je pri izvajanju algoritma v programih nekoliko bolj zahtevno, zato obstaja preprostejši algoritem, ki deluje le s pravilom slabega znaka.

Vrstni red primerjanja znakov je poljuben.

Algoritem HP je zelo učinkovit, ko je velikost abecede velika v primerjavi z dolžino vzorca. To se pogosto dogaja pri iskanju vzorca v besedilu v raznih urejevalcih besedila, zato je algoritem zelo uporaben.

Časovna zahtevnost faze pred iskanjem oziroma predobdelovalne faze je v primerjavi z algoritmom BM enaka (torej $O(m+\sigma)$). Kot smo omenili prej, nam gradnja tabele bmBc vzame $O(m+\sigma)$ časa. Seveda pa je prostorska zahtevnost manjša na prostorski ravni (in je $O(\sigma)$), saj algoritem HP ne sestavi tabele bmGs.

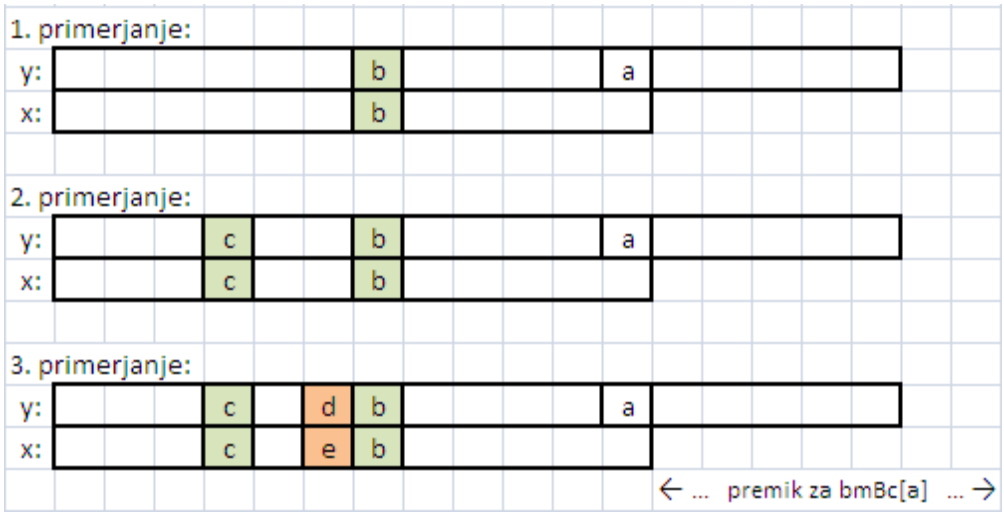
Časovna zahtevnost iskalne faze pri neperiodičnih vzorcih je v najslabšem primeru $O(mn)$, medtem ko je pri algoritmu BM $O(3n)$. Najmanj primerjav (n/m) algoritem HP naredi, ko znak iz besedila, ki stoji nad zadnjim znakom vzorca, v slednjem ne nastopa (glej slika 53, natančno delovanje algoritma je napisano v nadaljevanju).



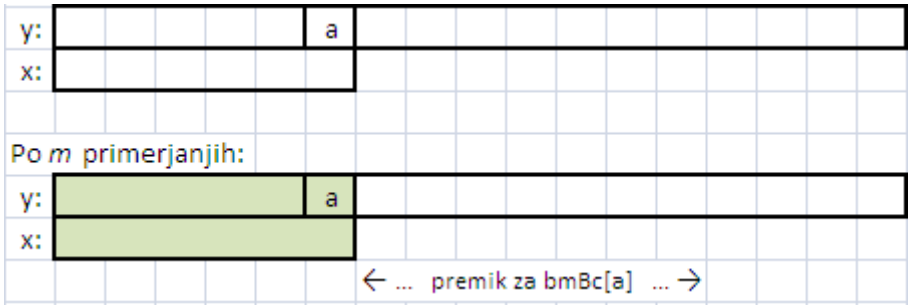
Slika 53: Minimalno število primerjav algoritma HP

Opis delovanja algoritma

Kot smo že omenili, algoritem HP uporablja le pravilo slabega znaka, ki pa je za malenkost različno od tistega, ki ga uporablja algoritem BM (tabela bmBc je enaka). V poskusu se primerjava med znakoma iz vzorca in besedila začne na poljubnem mestu, kar pomeni, da si lahko mesto primerjanja naključno izberemo. Ob morebitnem ujemanju sledi primerjava med naslednjima znakoma, prav tako na poljubnem položaju. Razlika z algoritmom BM je tudi v tem, da ob neujemanju dveh znakov (glej slika 54) ali pa ob popolnem ujemanju (vzorec je v besedilu najden, glej slika 55) pri algoritmu HP vedno sledi premik glede na znak iz besedila, ki stoji nad skrajno desnim znakom iz vzorca.



Slika 54: Algoritem HP, poljubna primerjava in neujemanje



Slika 55: Algoritem HP, popolno ujemanje

V nekaterih opisih algoritma HP je navedeno, da primerjanje poteka z desne strani proti levi. Dejansko pa je vrstni red primerjanja nepomemben, saj se premik vedno zgodi glede na znak iz besedila, ki stoji nad zadnjim znakom vzorca.

Pri opisu algoritma BM smo ugotovili, da se premik s pomočjo pravila slabega znaka največkrat zgodi ob takojšnjem neujemanju. Torej se največkrat zgodi premik enak premiku, ki ga uporablja algoritem HP.

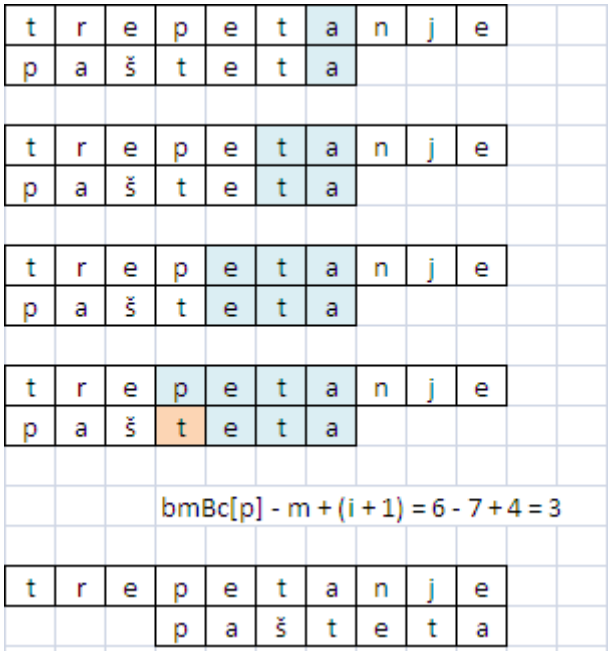
Še ena razlika med delovanjem pravila slabega znaka v algoritmu BM s HP je v tem, da slednji za premik ne naredi več nobenega preračuna. Premiki so shranjeni v tabeli bmBc, medtem ko algoritem BM te vrednosti še vstavi v enačbo premika (glej stran 18), saj upošteva že prej ujemajoči se sklop znakov.

Poglejmo primer delovanja pravila slabega znaka v poskusu pri obeh algoritmih. Najprej tvorimo tabelo bmBc, ki je pri obeh algoritmih enaka (glej slika 56).

c	a	e	j	n	p	r	t	š
bmBc[c]	5	2	7	7	6	7	1	4

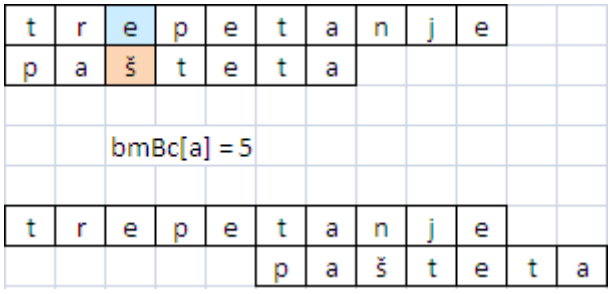
Slika 56: Tabela bmBc za algoritma BM in HP

Sedaj poglejmo delovanje slabega znaka pri algoritmu BM. Za predlog premika poleg tabele bmBc uporabi tudi enačbo premika (glej slika 57).



Slika 57: Primerjava algoritmov BM in HP, delovanje algoritma BM

Poglejmo še premik, ki ga izvede algoritem HP. Opazimo, da se primerjava med znakoma v poskusu začne na poljubnem položaju, vrednost premika pa je odvisna od znaka v besedilu, ki stoji nad zadnjim znakom vzorca (glej slika 58).



Slika 58: Primerjava algoritmov BM in HP, delovanje algoritma HP

Primeri

Za lažje razumevanje poglejmo delovanje algoritma HP še na dveh primerih.

Primer 20

BESEDILO: "garažiran avto"

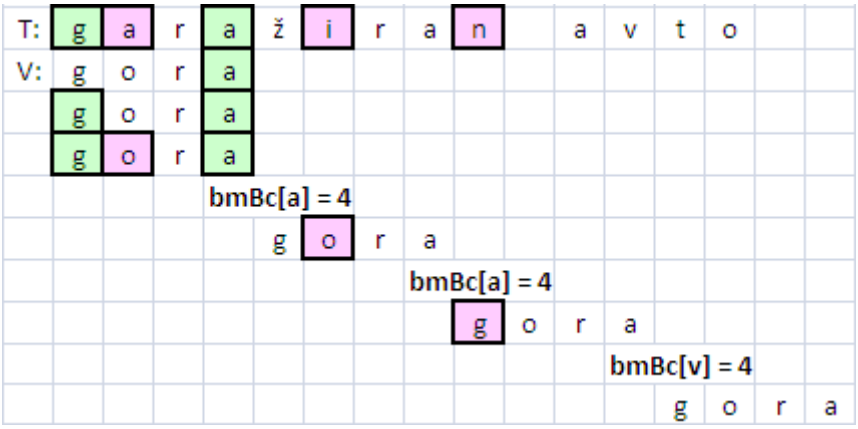
VZOREC: "gora"

Algoritem HP deluje s pomočjo tabele bmBc, zato najprej tvorimo omenjeno tabelo (glej slika 59).

	c		a	g	i	n	o	r	t	v	ž	
bmBc[*c*]			4	3	4	4	2	1	4	4	4	4

Slika 59: Primer 20, tabela bmBc

Sledi faza iskanja. Primerjavo v posameznem poskusu lahko začnemo na različnih položajih. V prvem poskusu smo jo začeli na četrtem mestu glede na vzorec, sledila je primerjava na prvem mestu, neujemanje pa smo našli na drugem mestu v vzorcu. Sledil je premik za vrednost znaka iz besedila, ki stoji nad zadnjim znakom vzorca. V drugem poskusu smo primerjali znak iz vzorca na drugem mestu z znakom iz besedila nad njim in takoj dobili neenakost. Premik smo opravili za vrednost znaka iz besedila, ki stoji nad zadnjim znakom vzorca. Zadnji, tretji poskus smo začeli med prvim znakom vzorca in znakom iz besedila nad njim. S tem poskusom smo zaradi neujemanja tudi takoj zaključili. Ponovno premik za vrednost tabele bmBc za znak iz besedila nad zadnjim znakom vzorca (glej slika 60).



Slika 60: Primer 20, celoten postopek iskanja

Osnovni podatki primera 20:

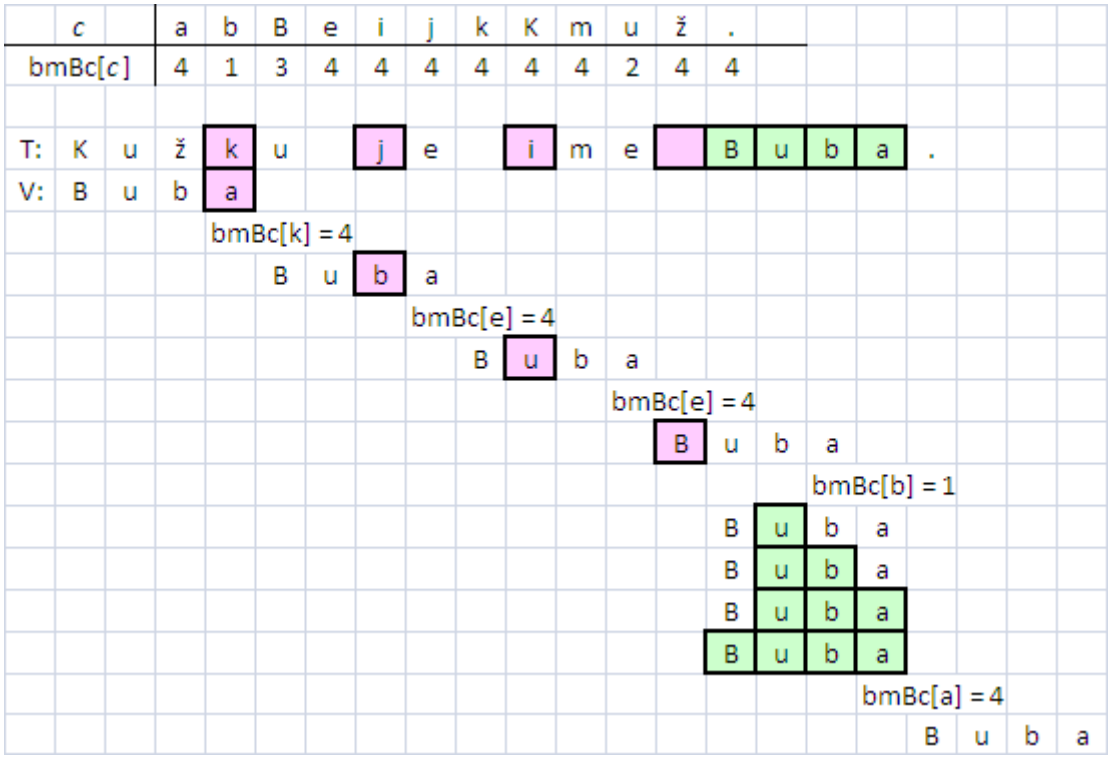
- dolžina besedila … 14
- dolžina vzorca … 4
- število poskusov … 3
- število primerjav znakov … 5

Primer 21

BESEDILO: "Kužku je ime Buba."

VZOREC: "Buba"

Pri tem primeru algoritem ponovno loči med velikimi in malimi črkami. Primerjava je poljubna, izbrali smo si primerjavo po vrsti. Pri prvem poskusu smo primerjali zadnji znak iz vzorca z znakom iz besedila nad njim, pri drugem poskusu predzadnji znak itd. Ko smo prišli do primerjave prvega znaka vzorca z znakom iz besedila nad njim, smo primerjavo nadaljevali z drugim znakom vzorca itd. (glej slika 61).



Slika 61: Primer 21, celoti postopek iskanja

Osnovni podatki primera 21:

- dolžina besedila … 18
- dolžina vzorca … 4
- število poskusov … 5
- število primerjav znakov … 8

ALGORITEM QUICK SEARCH

V praksi zelo uporaben algoritem je tudi algoritem Quick search (algoritem QS). Nastal je leta 1990, imenujemo pa ga tudi algoritem Sunday.

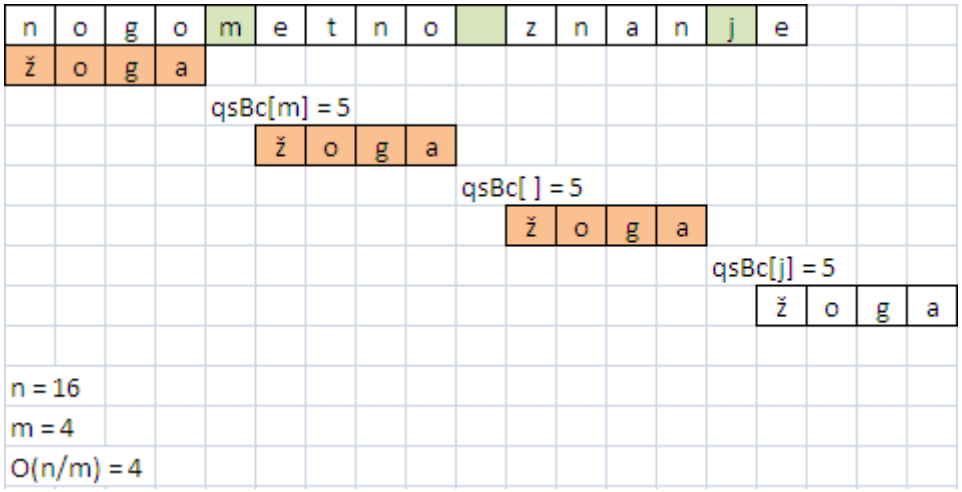
Osnovne lastnosti in primerjava z algoritmom BM

Gre za poenostavljeno različico algoritma BM. Kot Horspoolov algoritem tudi algoritem QS uporablja le pravilo slabega znaka, ki pa je v primerjavi z algoritmom BM nekoliko spremenjeno.

V poskusu se primerjava med znakoma v besedilu in vzorcu začne pri poljubnem znaku, tudi vrstni red naslednjih primerjav je poljuben.

Ker algoritem QS, podobno kot algoritem HP, za delovanje uporablja le pravilo slabega znaka, sta pred iskanjem časovni in prostorski zahtevnosti enaki, in sicer $O(m+\sigma)$ ter $O(\sigma)$.

Iskalna faza je v primerjavi z algoritmom HP na časovni ravni enaka, in sicer pri neperiodičnih vzorcih naredi algoritem QS največ mn primerjav (algoritem BM jih naredi $3n$). Najmanj primerjav (n/m) naredi algoritem QS, ko znak iz besedila na mestu bolj desno od zadnjega znaka vzorca v slednjem ne nastopa (glej slika 62, natančno delovanje algoritma je zapisano v nadaljevanju). V tem primeru naredi algoritem QS tri primerjave, kar je še za eno manj od n/m primerjav.



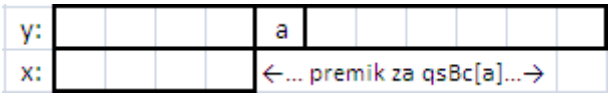
Slika 62: Minimalno število primerjav algoritma QS

V praksi je algoritem zelo hiter pri iskanju kratkih vzorcev v velikih abecedah. To je pogosto primer v raznih urejevalnikih besedil, kjer v besedilu po navadi iščemo eno besedo (ki ni zelo dolga). Takrat lahko rečemo, da je dolžina besede v primerjavi z velikostjo abecede majhna.

Opis delovanja algoritma

Algoritem QS uporablja le pravilo slabega znaka, vendar se od istoimenskega pravila, definiranega pri algoritmu BM (glej stran 16), nekoliko razlikuje. Pri pravilu slabega znaka (pri algoritmih BM in HP) je pri določenem poskusu okno postavljeno pod besedilo na položaju $y[j \,..\, j+m-1]$. Glede na to, da je dolžina vzorca enaka vsaj 1, je tudi dolžina premika enaka vsaj 1. Torej je v naslednjem poskusu (po premiku) eden od znakov vzorca vsekakor postavljen pod znak $y[j+m]$. Zato algoritem QS (za razliko od algoritma HP) upošteva znak $y[j+m]$ že pri poskusu, ko stoji zadnji znak vzorca pod znakom $y[j+m-1]$. V primerjavi z algoritmom HP (premik glede na znak iz besedila nad zadnjim znakom

vzorca) se zgodi premik glede na znak iz besedila desno od zadnjega znaka vzorca (glej slika 63). Zato je tudi maksimalen premik pri algoritmu QS (*m+1*) za ena večji od maksimalnega premika pri algoritmu HP (*m*).



Slika 63: Algoritem QS, premik

Zaradi tega algoritem QS tvori tabelo qsBc, ki se od bmBc razlikuje po tem, da upošteva tudi zadnji znak vzorca. Tabela qsBc shrani najbolj desno pojavitev določenega znaka v vzorcu, zapisano v obratnem vrstnem redu (zadnji znak vzorca dobi vrednost 1, predzadnji znak vrednost 2 … prvi znak vzorca pa vrednost *m*). Znaki, ki v vzorcu ne nastopajo, dobijo vrednost *m + 1*.

Definicija tabele qsBc:

Za $c \in \Sigma$ velja:

$$qsBc[c] = \begin{cases} \min\{i: 0 < i \leq m \ in \ x[m-i] = c, če \ se \ c \ pojavi \ v \ x, \\ m+1 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad sicer. \end{cases}$$

Primer gradnje tabele qsBc

Primer 22

BESEDILO: "Šola je nora"

VZOREC: "nesmisel!"

Tabelo tvorimo za vse znake, ki nastopajo v besedilu in vzorcu. Vrednost pri znakih, ki v vzorcu ne nastopajo, je 10. Znaki, ki v vzorcu nastopajo, dobijo obratne vrednosti najbolj desnega položaja. Zadnji znak ima vrednost 1, predzadnji 2 … prvi znak pa ima vrednost 9 (glej slika 64).

c	a	e	i	j	l	m	n	o	r	s	Š	!	
qsBc[*c*]	10	3	5	10	2	6	9	10	10	4	10	1	10

Slika 64: Primer 22, tabela qsBc

Primeri

Oglejmo si delovanje algoritma QS na treh primerih.

Primer 23

BESEDILO: "opažanje in razumevanje"

VZOREC: "predvidevanje"

Prvi korak je gradnja tabele qsBc (glej slika 65).

c	a	d	e	i	j	m	n	o	p	r	u	v	z	ž	
qsBc[c]	4	7	1	8	2	14	3	14	13	12	14	5	14	14	14

Slika 65: Primer 23, tabela qsBc

Nato se začne prvi poskus. Primerjava med znakoma v besedilu in vzorcu se lahko začne na katerem koli mestu. V primeru 23 smo se odločili za primerjavo znakov na začetku okna, ob ujemanju se bomo premikali za eno mesto v desno. Ker ob prvi primerjavi dobimo neenakost, sledi premik. Gledamo vrednost znaka v tabeli qsBc, in sicer za znak v besedilu, ki stoji za eno mesto bolj desno od zadnjega znaka vzorca, torej za znak 'a' (glej slika 66).

T:	o	p	a	ž	a	n	j	e		i	n		r	a	z	u	m	e	v	a	n	j	e
V:	p	r	e	d	v	i	d	e	v	a	n	j	e										
	qsBc[a] = 4																						

Slika 66: Primer 23, prvi poskus

Sledita drugi (glej slika 67) in tretji (glej slika 68) poskus, kjer ni nobenih razlik glede na prvi poskus.

T:	o	p	a	ž	a	n	j	e		i	n		r	a	z	u	m	e	v	a	n	j	e
V:					p	r	e	d	v	i	d	e	v	a	n	j	e						
					qsBc[e] = 1																		

Slika 67: Primer 23, drugi poskus

T:	o	p	a	ž	a	n	j	e		i	n		r	a	z	u	m	e	v	a	n	j	e
V:						p	r	e	d	v	i	d	e	v	a	n	j	e					
						qsBc[v] = 5																	

Slika 68: Primer 23, tretji poskus

Pri četrtem poskusu pa opazimo, da se okno poravna z desnim koncem besedila. Algoritem QS v določenem poskusu uporabi znak desno od okna, v tem primeru na tem mestu ni nobenega znaka več. Zato sledi kar premik za vrednost znakov, ki v vzorcu ne nastopajo, torej qsBc[0] = 14 (glej slika 69). To je tudi zadnji poskus, saj se desni rob vzorca po tem premiku očitno postavi za desni rob besedila.

T:	o	p	a	ž	a	n	j	e		i	n		r	a	z	u	m	e	v	a	n	j	e
V:											p	r	e	d	v	i	d	e	v	a	n	j	e
											qsBc[0] = 14												

Slika 69: Primer 23, četrti poskus

Osnovni podatki primera 23:

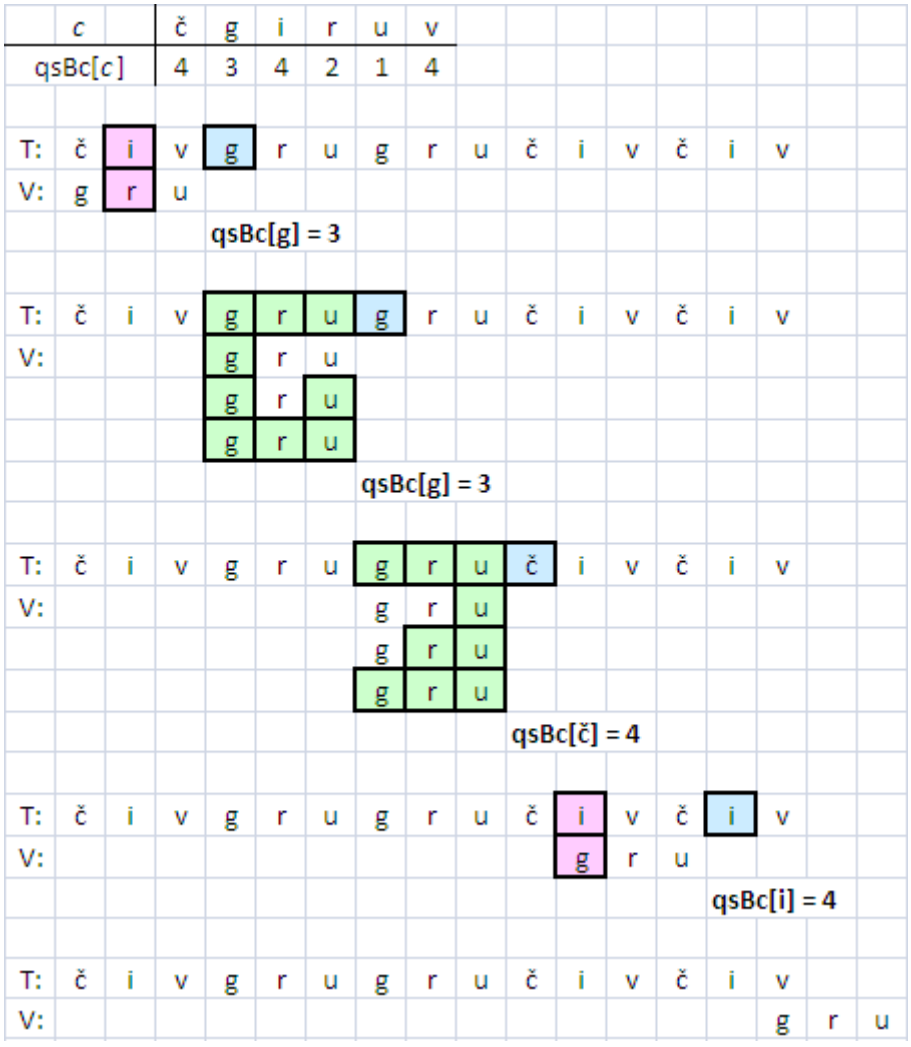
- dolžina besedila … 23
- dolžina vzorca … 13
- število poskusov … 4
- število primerjav znakov … 4

Primer 24

BESEDILO: "čivgrugručivčiv"

VZOREC: "gru"

V primeru 24 smo znake med seboj primerjali po naključnem vrstnem redu. Za primerjavo smo si izbrali naključen znak iz vzorca in ga primerjali z znakom iz besedila nad njim (glej slika 70).



Slika 70: Primer 24, celoten postopek iskanja

Osnovni podatki primera 24:

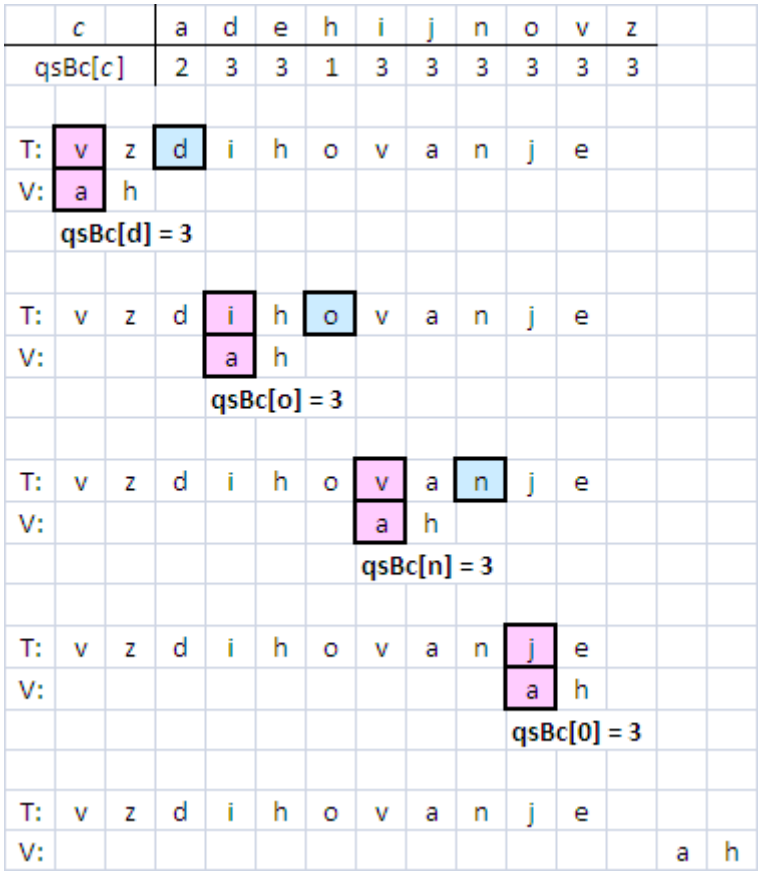
- dolžina besedila … 15
- dolžina vzorca … 3
- število poskusov … 4
- število primerjav znakov … 8

Primer 25

BESEDILO: "vzdihovanje"

VZOREC: "ah"

Pri tem primeru opazimo, da je za krajše vzorce QS algoritem res hiter. Opravi minimalno število primerjav znakov, manj kot *n/m*. Kot vedno pri algoritmu QS, je vrstni red primerjanja poljuben. Mi smo v vseh poskusih začeli pri prvem znaku vzorca (glej slika 71).



Slika 71: Primer 25, celoten postopek iskanja

Osnovni podatki primera 25:

- dolžina besedila … 11
- dolžina vzorca … 2
- število poskusov … 4
- število primerjav znakov … 4

SMITHOV ALGORITEM

Nastal je leta 1991, ko je P. D. Smith algoritma HP in QS, opisana v prejšnjih dveh poglavjih, združil v nov algoritem.

Osnovne lastnosti in primerjava z algoritmom BM

Algoritem, tako kot algoritma HP in QS, za določanje premika uporabi le pravilo slabega znaka. Pri tem pa izračuna premik tako po načinu, ki ga uporablja Horspoolov algoritem, kot tudi po načinu, uporabljenem pri algoritmu Quick Search. Potem seveda uporabi večji premik.

Vrstni red primerjanja znakov je, tako kot pri algoritmih HP in QS, poljuben.

Predobdelovalna faza ima enako časovno in prostorsko zahtevnost kot algoritma HP in QS, in sicer $O(m+\sigma)$ ter $O(\sigma)$. Pri tem je razlika z algoritmom BM le pri prostorski zahtevnosti faze pred iskanjem. Omenjena faza je pri algoritmu BM zahtevnosti $O(m+\sigma)$, saj algoritem tvori še tabelo dobre pripone.

Tudi iskalna faza je v primerjavi z algoritmi BM, HP in QS na časovni ravni enaka, in sicer $O(mn)$.

Opis delovanja algoritma

Kot smo že omenili, Smithov algoritem med premikoma, predlaganima po postopku algoritmov HP in QS, izbere večji premik. Poglejmo povzetek pravil slabega znaka pri obeh algoritmih:

- **Horspool**: premik se določi glede na znak iz besedila, ki stoji nad zadnjim znakom vzorca; tabela bmBc hrani najbolj desne pojavitve znakov v vzorcu, zadnji znak vzorca ne upošteva;
- **Quick search**: premik se določi glede na znak iz besedila, ki stoji eno mesto desno od zadnjega znaka vzorca; tabela qsBc hrani najbolj desne pojavitve znakov v vzorcu, zadnji znak vzorca upošteva.

Podrobnejši opis obeh pravil si lahko ogledamo na straneh 34 (HP) in 39 (QS).

Za delovanje uporablja Smithov algoritem obe tabeli, **bmBc** (stran 18) in **qsBc** (stran 40). Glede na v tabelah hranjene vrednosti se lahko zgodijo tri možnosti:

i. Algoritem HP predlaga večji premik od algoritma QS (glej slika 72);
ii. Algoritma HP in QS predlagata enaka premika (glej slika 73);
iii. Algoritem QS predlaga večji premik od algoritma HP (glej slika 74).

Vemo, da sta za premike pri Smithovem algoritmu odločilna znaka iz besedila nad zadnjim znakom vzorca (HP) in desno od zadnjega znaka vzorca (QS), torej na položajih v besedilu $m-1$ in m. Na spodnjih slikah smo znak, ki je pomemben za premik, s pomočjo tabele bmBc označili z 'b', znak, ki mu določi vrednost premika s tabelo qsBc, pa 'a'.

Najprej si poglejmo možnost, ko algoritem HP predlaga večji premik od algoritma QS. To se lahko zgodi na štiri različne načine (glej slika 72):

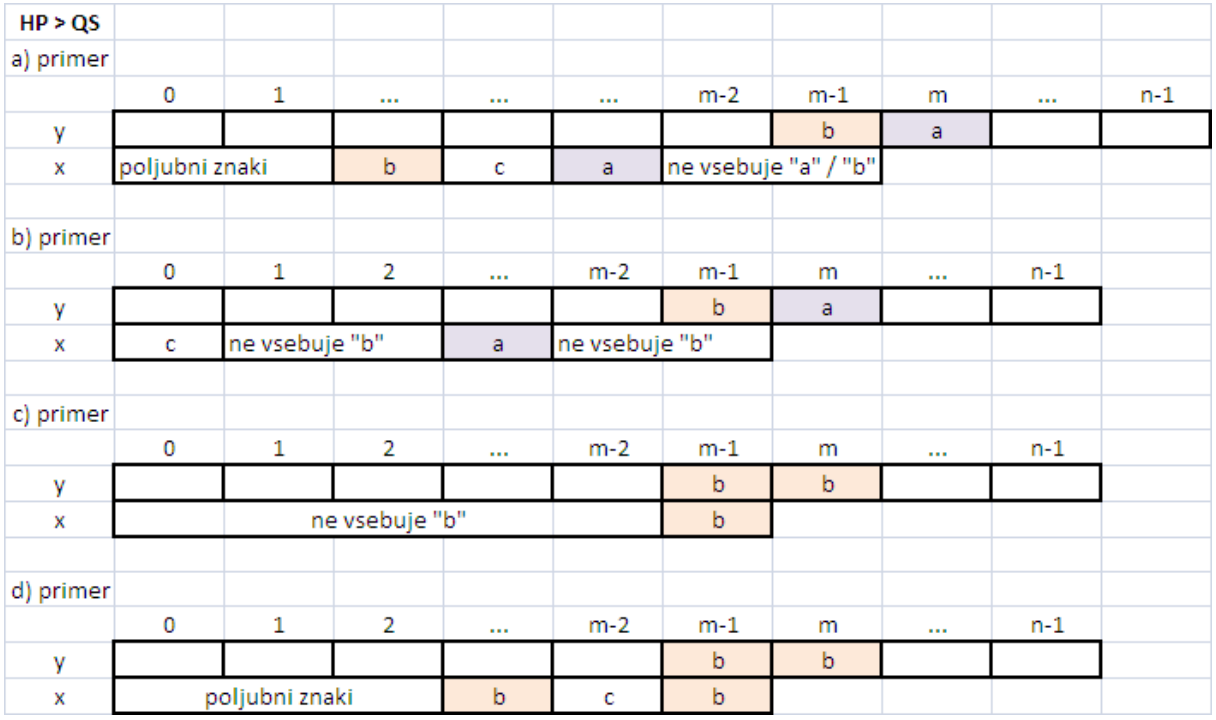
a) V besedilu je na položaju $m-1$ znak 'b', na položaju m pa znak 'a'. Najbolj desni položaj znakov 'a' in 'b' v vzorcu je takšen, da ima znak 'b' položaj bolj levo od znaka 'a', med njima pa stoji vsaj en znak, ki ni enak 'b'-ju. Ta vmesni znak označimo s 'c', 'c' je lahko enak tudi znaku 'a', kar pa na rezultat ne vpliva, saj se v tabeli qsBc upošteva najbolj desni znak 'a'. Med znakoma 'a' in 'b' je lahko tudi več znakov, pomembno je le, da stoji znak 'b' bolj levo od znaka 'a'. Na

mestih bolj levo od najbolj desnih pojavitev znakov 'a' in 'b' v vzorcu se lahko pojavijo poljubni znaki (tudi 'a' in 'b'), saj na premik vplivajo le najbolj desne pojavitve znakov 'a' in 'b'. Povejmo še, da lahko znak 'a' nastopa tudi na zadnjem mestu vzorca. Med znakom 'a' in najbolj desnim položajem znaka 'b' pa nastopa vsaj en od 'b' različen znak.
Premik s pomočjo algoritma HP se zgodi tudi takrat, ko sta 'a' in 'b' na že opisanem položaju, znak na zadnjem mestu vzorca pa je enak 'b'-ju. V tem primeru tabela bmBc zadnjega znaka ne upošteva.

b) V besedilu je na položaju *m-1* znak 'b', na položaju *m* pa znak 'a'. Znak 'a' lahko nastopa v vzorcu na vseh položajih, razen na prvem. Znak 'b' v vzorcu ne nastopa, lahko nastopa le na zadnjem mestu, ki ga tabela bmBc ne upošteva.

c) Znaka v besedilu nad zadnjim znakom vzorca in desno od zadnjega znaka vzorca sta enaka ('b'). Znak 'b' nastopa v vzorcu samo na zadnjem mestu. Tako algoritem QS predlaga premik za ena, algoritem HP pa premik za *m*, ki je seveda večji.

d) Ponovno sta znaka v besedilu nad zadnjim znakom vzorca in desno od zadnjega znaka vzorca enaka ('b'). V vzorcu znak 'b' nastopa vsaj dvakrat. Najbolj desna pojavitev je na zadnjem mestu vzorca. Druga najbolj desna pojavitev pa ni na predzadnjem mestu vzorca, na tem mestu je znak 'c', ki je od 'b' različen. Med zadnjo in predzadnjo pojavitvijo znaka 'b' v vzorcu je vsaj en znak od 'b' različen (lahko jih je tudi več). Tako algoritem QS predlaga premik za ena, algoritem HP pa premik, večji od ena.



Slika 72: Možnosti, ko algoritem HP predlaga večji premik od algoritma QS

Algoritma HP in QS predlagata enaka premika v naslednjih štirih možnostih (glej slika 73):

a) V besedilu je na položaju *m-1* znak 'b', na položaju *m* pa znak 'a'. Najbolj desna pojavitev znaka 'a' v vzorcu je na položaju desno od najbolj desne pojavitve znaka 'b' v vzorcu. Torej znaka stojita eden zraven drugega, znak 'b' je levo, znak 'a' desno. Znak 'a' je lahko tudi zadnji

znak vzorca, znak 'b' je potem predzadnji znak vzorca (v tem primeru je premik s pomočjo algoritmov HP in QS enak ena).

Na zadnjem mestu vzorca lahko nastopa tudi znak 'b', vendar njegov drugi najbolj desni položaj (najbolj desni položaj je zadnji znak vzorca) v vzorcu nastopa levo od znaka 'a'. Levo od obeh sosednjih znakov 'a' in 'b', ki vplivata na premik, nastopajo poljubni znaki. Med njimi so lahko tudi 'a'-ji in 'b'-ji, ki pa se v vzorcu pojavijo tudi na položajih bolj desno, zato na tabeli bmBc in qsBc ne vplivajo.

b) V besedilu je na položaju *m-1* znak 'b', na položaju *m* pa znak 'a'. Znak 'a' nastopa v vzorcu le na prvem mestu, znaka 'b' v vzorcu ni (lahko je le na zadnjem mestu, ki pa na tabelo bmBc ne vpliva). Tako sta premika s pomočjo algoritmov HP in QS enaka, in sicer *m*.

c) V besedilu sta na položajih *m-1* in *m* enaka znaka ('b'). V vzorcu sta znaka 'b' na položajih *m-2* in *m-1*, torej na predzadnjem in na zadnjem mestu. V tem primeru oba algoritma predlagata premik za ena. Levo od obeh znakov v vzorcu so poljubni znaki, med njimi so lahko tudi 'b'-ji, ki pa na tabeli bmBc in qsBc ne vplivajo.

HP = QS

a) primer

	0	1	...	...	m-2	m-1	m	...	n-1
y						b	a		
x	poljubni znaki		b	a	ne vsebuje "a" / "b"				

b) primer

	0	1	2	...	m-2	m-1	m	...	n-1
y						b	a		
x	a	ne vsebuje "a" / "b"							

c) primer

	0	1	2	...	m-2	m-1	m	...	n-1
y						b	b		
x		poljubni znaki			b	b			

Slika 73: Možnosti, ko algoritma HP in QS predlagata enaka premika

Večji premik lahko predlaga tudi algoritem QS. To se zgodi na pet različnih načinov (glej slika 74):

a) V besedilu je na položaju *m-1* znak 'b', na položaju *m* pa znak 'a'. V vzorcu znaka 'a' in 'b' ne nastopata ('b' lahko nastopa na zadnjem mestu vzorca, vendar ta položaj na tabelo bmBc ne vpliva), zato algoritem HP predlaga premik za *m*, algoritem QS pa za *m+1*, večji je seveda slednji premik.

b) V besedilu je na položaju *m-1* znak 'b', na položaju *m* pa znak 'a'. V vzorcu znak 'a' ne nastopa, znak 'b' pa nastopa na poljubnem mestu. Znak 'b' je lahko tudi na zadnjem položaju v vzorcu, vemo pa, da ta položaj na tabelo bmBc ne vpliva. Če je to njegova edina pojavitev v vzorcu, dobimo enako predlagane premike kot v a) primeru. V vzorcu se lahko 'b' pojavi večkrat, tabela bmBc uporabi njegov najbolj desni položaj (zadnjega znaka ne upošteva).

c) V besedilu je na položaju *m-1* znak 'b', na položaju *m* pa znak 'a'. V vzorcu se znaka 'a' in 'b' pojavita najbolj desno kot sosednja znaka, in sicer 'a' na levi in 'b' na desni strani. V tem primeru algoritem QS predlaga premik za dve večji od premika algoritma HP. Če je med znakoma v enakem vrstnem redu ('a' levo, 'b' desno) še kakšen drug znak, premik tudi

predlaga algoritem QS. V tem primeru je razlika med premikoma algoritmov QS in HP več kot dva. Levo od obeh znakov so poljubni znaki.
Znak 'b' lahko nastopa tudi na zadnjem mestu vzorca (najbolj desna pojavitev), vendar mora biti njegova druga najbolj desna pojavitev na položaju desno od najbolj desnega znaka 'a'.

d) V besedilu sta na položajih *m-1* in *m* enaka znaka ('b'). Vzorec znaka 'b' ne vsebuje (niti na zadnjem mestu), zato algoritma predlagata premika *m+1* (QS) in *m* (HP). Večji je premik algoritma QS.

e) V besedilu sta na položajih *m-1* in *m* enaka znaka ('b'). V vzorcu nastopa natanko en znak 'b', ki pa ne sme biti na zadnjem mestu vzorca. Nastopa na katerem koli drugem položaju in v tem primeru algoritem QS predlaga premik, za ena večji od premika, ki ga predlaga algoritem HP.

QS > HP

a) primer

	0	1	2	...	m-2	m-1	m	...	n-1
y						b	a		
x			ne vsebuje "a" / "b"						

b) primer

	0	1	2	...	m-2	m-1	m	...	n-1
y						b	a		
x	ne vsebuje "a"			b	ne vsebuje "a" / "b"				

c) primer

	0	1	...	...	m-2	m-1	m	...	n-1
y						b	a		
x	poljubni znaki		a	b	ne vsebuje "a" / "b"				

d) primer

	0	1	2	...	m-2	m-1	m	...	n-1
y						b	b		
x			ne vsebuje "b"						

e) primer

	0	1	2	...	m-2	m-1	m	...	n-1
y						b	b		
x	poljubni znaki			b	ne vsebuje "b"				

Slika 74: Možnosti, ko algoritem QS predlaga večji premik od algoritma HP

Primeri

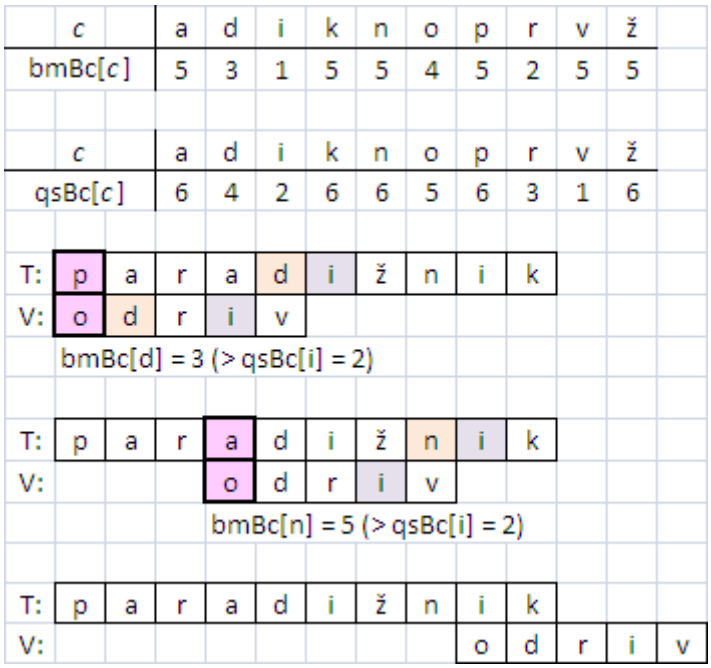
Poglejmo si delovanje Smithovega algoritma na nekaj primerih.

Primer 26

BESEDILO: "paradižnik"

VZOREC: "odriv"

Najprej tvorimo tabeli bmBc in qsBc. Ko sledimo postopku, vidimo, da pri obeh poskusih večji premik predlaga algoritem HP. Vrstni red primerjav je poljuben, v obeh poskusih pa smo primerjanje začeli na prvem mestu vzorca (glej slika 75).



Slika 75: Primer 26, celoten postopek iskanja

Osnovni podatki primera 26:

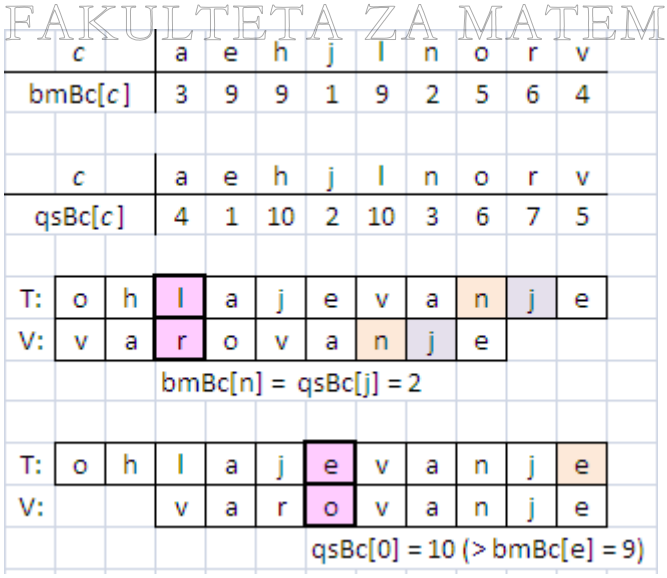
- dolžina besedila … 10
- dolžina vzorca … 5
- število poskusov … 2
- število primerjav znakov … 2

Primer 27

BESEDILO: "ohlajevanje"

VZOREC: "varovanje"

Najprej tvorimo obe tabeli, bmBc in qsBc. V prvem poskusu začnemo primerjavo na tretjem mestu v vzorcu, da dobimo neenakost. Algoritma HP in QS predlagata enaka premika. V drugem poskusu začnemo primerjavo na četrtem mestu vzorca, takoj dobimo neenakost. Opazimo, da sta besedilo in vzorec poravnana z desnima roboma. Algoritem QS predlaga premik glede na znak iz besedila desno od desnega roba vzorca. Ker se je besedilo tam že končalo, qsBc predlaga premik za $m+1$, saj »ničti« znak v vzorcu ne nastopa. Po tem premiku se levi rob vzorca premakne za dve mesti od desnega roba besedila, zato premika nismo prikazali (glej slika 76).



Slika 76: Primer 27, celotni postopek iskanja

Osnovni podatki primera 27:

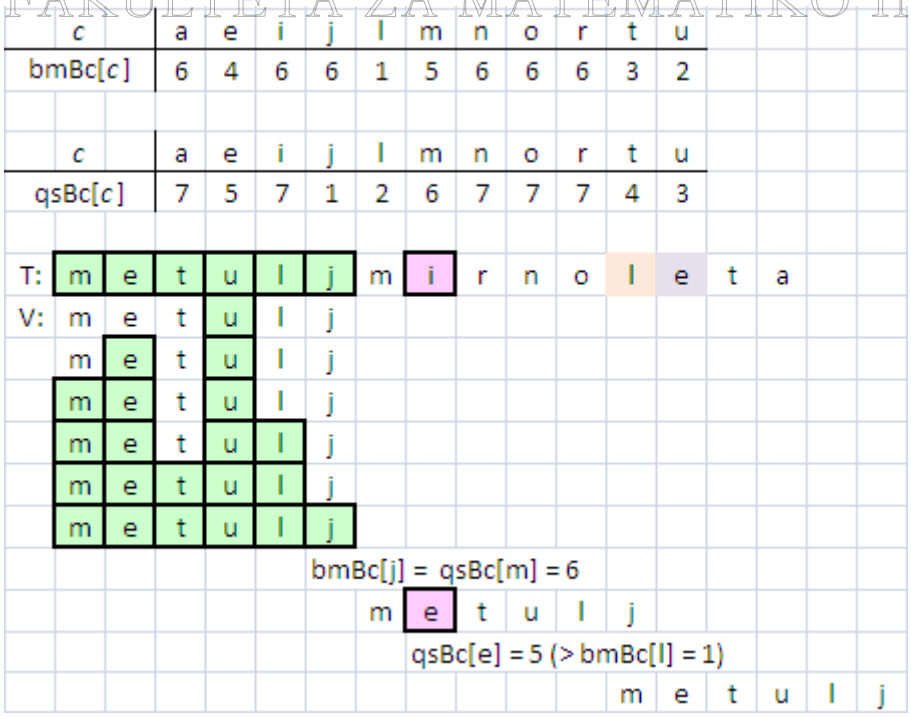
- dolžina besedila … 11
- dolžina vzorca … 9
- število poskusov … 2
- število primerjav znakov … 2

Primer 28

BESEDILO: "metuljmirnoleta"

VZOREC: "metulj"

V prvem poskusu dobimo popolno ujemanje, kar pri premiku ničesar ne spremeni. Ponovno predlagata premike algoritma HP in QS. Prvi predlaga premik glede na znak iz besedila nad zadnjim znakom vzorca, drugi pa glede na znak iz besedila desno od zadnjega znaka vzorca. V tem primeru oba algoritma predlagata enaka premika. V poskusu 2 večji premik predlaga algoritem QS, sledi pa zadnji premik vzorca. Pri primerjanju smo si ponovno izbrali poljubne položaje v vzorcu (glej slika 77).



Slika 77: Primer 28, celotni postopek iskanja

Osnovni podatki primera 28:

- dolžina besedila … 15
- dolžina vzorca … 6
- število poskusov … 2
- število primerjav znakov … 7

RAITOV ALGORITEM

Leta 1992 je T. Raita napisal algoritem, ki izhaja iz Horspoolovega algoritma. Ugotovil je, da je nov algoritem nekoliko hitrejši pri iskanju v angleških besedilih, takšno delovanje pa je pripisal obstoju odvisnosti med znaki.

Osnovne lastnosti in primerjava z algoritmom BM

Uporablja enake premike kot Horspoolov algoritem, kar pomeni, da ob neujemanju vedno stori premik glede na znak iz besedila nad zadnjim znakom vzorca.

Vrstni red primerjav znakov je specifičen, kar pomeni, da se pri primerjavah držimo določenega vrstnega reda. Ta je opisan v naslednjem poglavju, kjer opisujemo delovanje algoritma.

Časovna in prostorska zahtevnost predobdelovalne faze sta $O(m+\sigma)$ ter $O(\sigma)$.

Časovna zahtevnost iskalne faze je v teoriji enaka kot pri algoritmu HP, in sicer $O(mn)$. Raita pa je s primeri pokazal, da koda novega algoritma deluje 25 % hitreje. Natančnejši opis najdemo v članku, ki ga je Raita napisal leta 1992.

Opis delovanja algoritma

Algoritem HP ima dobro delovanje pri poljubnih besedilih. Velikokrat pa je besedilo določenega tipa, pri čemer je eden izmed tipov angleški jezik. V angleškem jeziku velja, da so znaki v besedi med seboj odvisni, razlaga je v članku, ki sem ga omenil zgoraj. Najmanjša odvisnost je med znakoma na začetku in koncu besede. Zato Raiti primerjava na primer z desne proti levi ni bila všeč, saj je verjetnost ujemanja znakov na koncu besede večja, kot pri ujemanju znakov na zadnjem in na prvem mestu. Zato ima algoritem Raita specifični vrstni red primerjav. V posameznem poskusu se primerjava začne pri zadnjem znaku vzorca. Če se znaka ujemata, se z ustreznim znakom primerja prvi znak vzorca. Če se ujemata še ta dva znaka, je na vrsti primerjanje pri sredinskem znaku vzorca. Če pa je tudi ta znak enak ustreznemu znaku iz besedila, sledijo primerjave ostalih znakov po vrsti od drugega do predzadnjega. Sredinski znak vzorca se torej ob ujemanju znakov, ki so na vrsti pred njim, primerja dvakrat; programska koda le za enkratno primerjanje sredinskega znaka je preveč zapletena, zato poznamo algoritem z dvojnim primerjanjem sredinskega znaka. Na sliki 78 je s številkami označen vrstni red primerjav.

y				2	4	5	3, 6	7	8	1			
x										1			
				2						1			
				2			3			1			
				2	4		3			1			
				2	4	5	3			1			
				2	4	5	6			1			
				2	4	5	3, 6	7		1			
				2	4	5	3, 6	7	8	1			

Slika 78: Vrstni red primerjav pri algoritmu Raita

Sredinski znak vzorca določimo s celoštevilskim deljenjem dolžine vzorca s številom 2. Ker se indeksacija znakov na položajih vzorca začne z 0, pomeni, da je sredinski znak pri sodih dolžinah vzorca prvi znak druge polovice vzorca.

Povedali smo, da algoritem Raita izhaja iz algoritma HP. Razlika med njima je le v vrstnem redu primerjanja. Premiki so torej enaki kot pri algoritmu HP. Ob popolnem ujemanju ali ob neujemanju na katerem koli mestu vedno sledi premik glede na znak iz besedila, ki stoji nad zadnjim znakom vzorca. Torej tudi tu tvorimo tabelo bmBc, ki je enaka kot pri algoritmu HP (podrobnejši opis delovanja algoritma HP je na strani 34).

Primeri

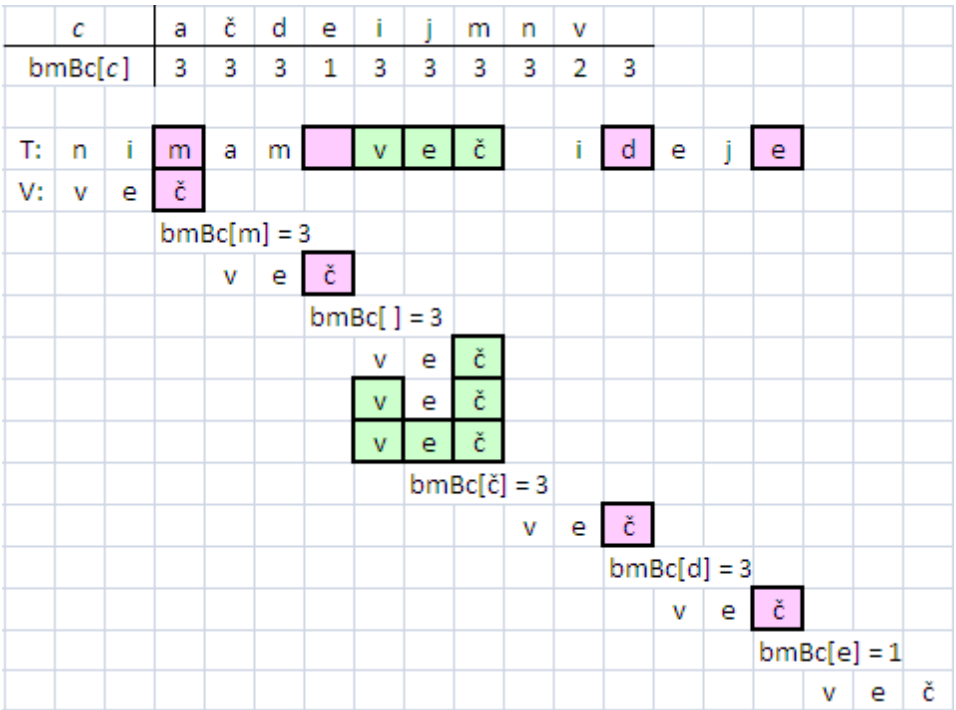
Poglejmo delovanje Raitovega algoritma na dveh primerih.

Primer 29

BESEDILO: "nimam več ideje"

VZOREC: "več"

Najprej tvorimo tabelo bmBc. Nato izvedemo ustrezne poskuse. Pri primerjanju je vrstni red primerjav takšen, da iz vzorca najprej primerjamo zadnji znak, zatem prvi znak in, če je treba, še sredinski znak vzorca (glej slika 79).



Slika 79: Primer 29, celotni postopek iskanja

Osnovni podatki primera 29:

- dolžina besedila … 15
- dolžina vzorca … 3

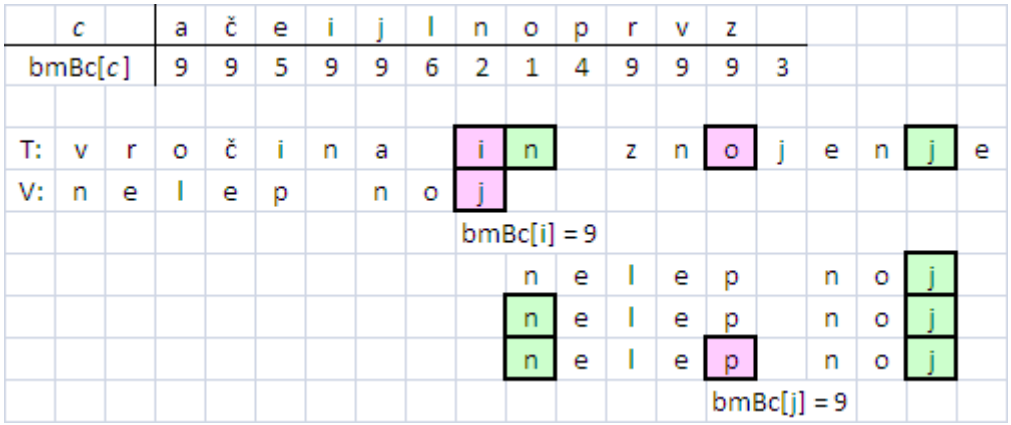
- število poskusov … 5
- število primerjav znakov … 8

Primer 30

BESEDILO: "vročina in znojenje"

VZOREC: "nelep noj"

V tem primeru vsebuje vzorec dve besedi, vmes pa je presledek. To ni nič posebnega, saj presledek obravnavamo kot samostojen znak. V zadnjem poskusu ob neujemanju bmBc predlaga premik, vendar ga zaradi prostorske stiske na sliki nismo prikazali (glej slika 80). Ta premik je tolikšen, da gremo z vzorcem že izven območja primerjanja.



Slika 80: Primer 30, celotni postopek iskanja

Osnovni podatki primera 30:

- dolžina besedila … 19
- dolžina vzorca … 9
- število poskusov … 2
- število primerjav znakov … 4

PRIMERJAVA ALGORITMOV

To poglavje je sestavljeno iz dveh razdelkov. V vsakem od poglavij bomo na enakem primeru izvedli vse opisane algoritme. Osnovne podatke delovanja vseh algoritmov, kot so število poskusov (POS), število primerjav znakov (PZ) in število POS & PZ do prvega popolnega ujemanja, bomo zapisali v skupno tabelo in jih med seboj primerjali.

Primer A

BESEDILO: "vsevednavedavedoslovje"

VZOREC: "vida"

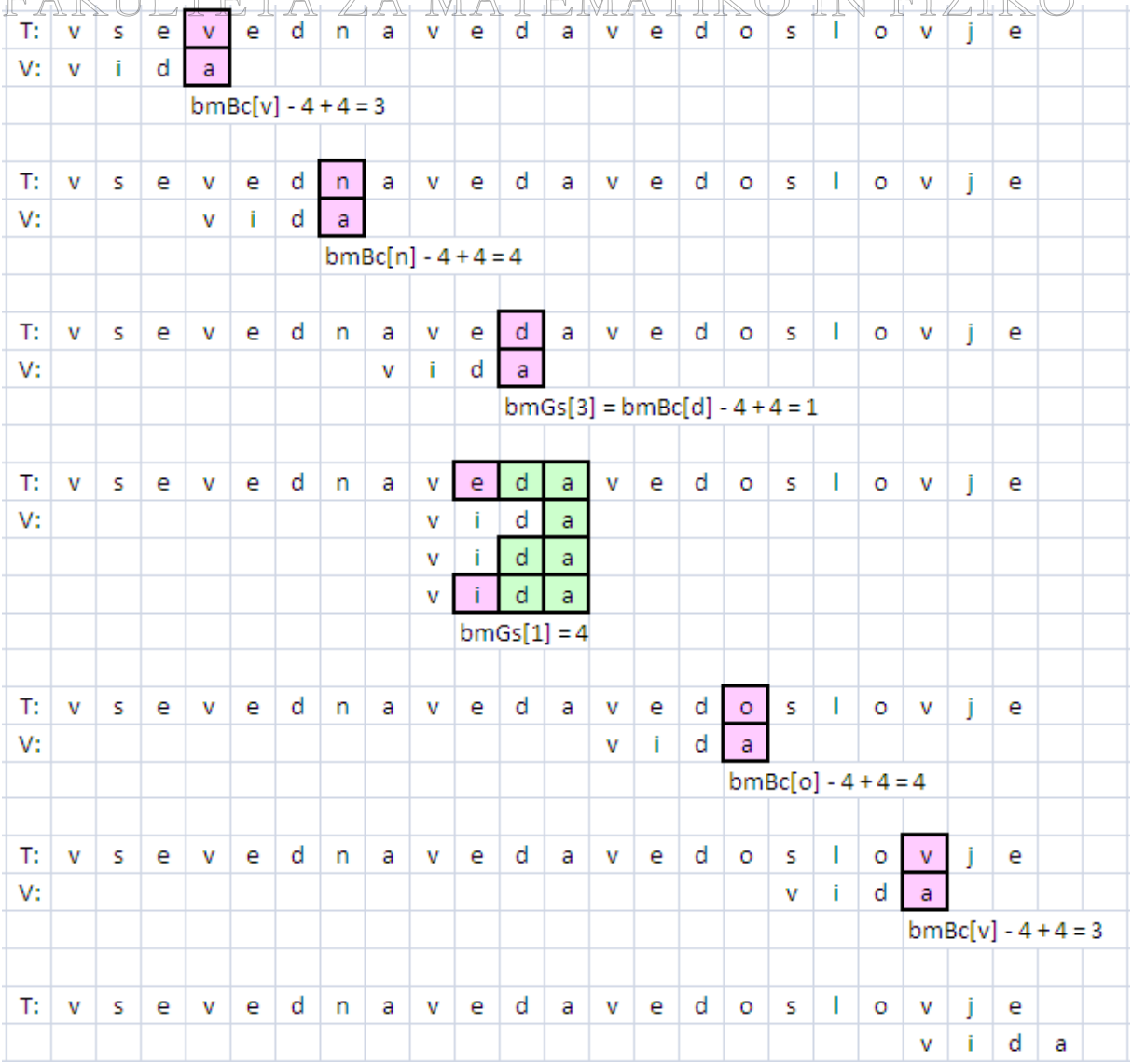
Boyer-Mooreov algoritem

Najprej tvorimo ustrezni tabeli – tabelo za premike po pravilu dobrega znaka in tabelo za premike po pravilu dobre pripone.

i	0	1	2	3						
x[i]	v	i	d	a						
bmGs[i]	4	4	4	1						
c	a	d	e	i	j	l	n	o	s	v
bmBc[*c*]	4	1	4	2	4	4	4	4	4	3

Slika 81: Primer A, Boyer-Mooreov algoritem, tabeli bmGs in bmBc

Oglejmo si sedaj, kako poteka primerjanje.



Slika 82: Primer A, Boyer-Mooreov algoritem, celotni postopek iskanja

Kot vidimo, smo za premik štirikrat uporabili pravilo slabega znaka, enkrat pravilo dobre pripone, enkrat pa sta obe pravili predlagali enak premik.

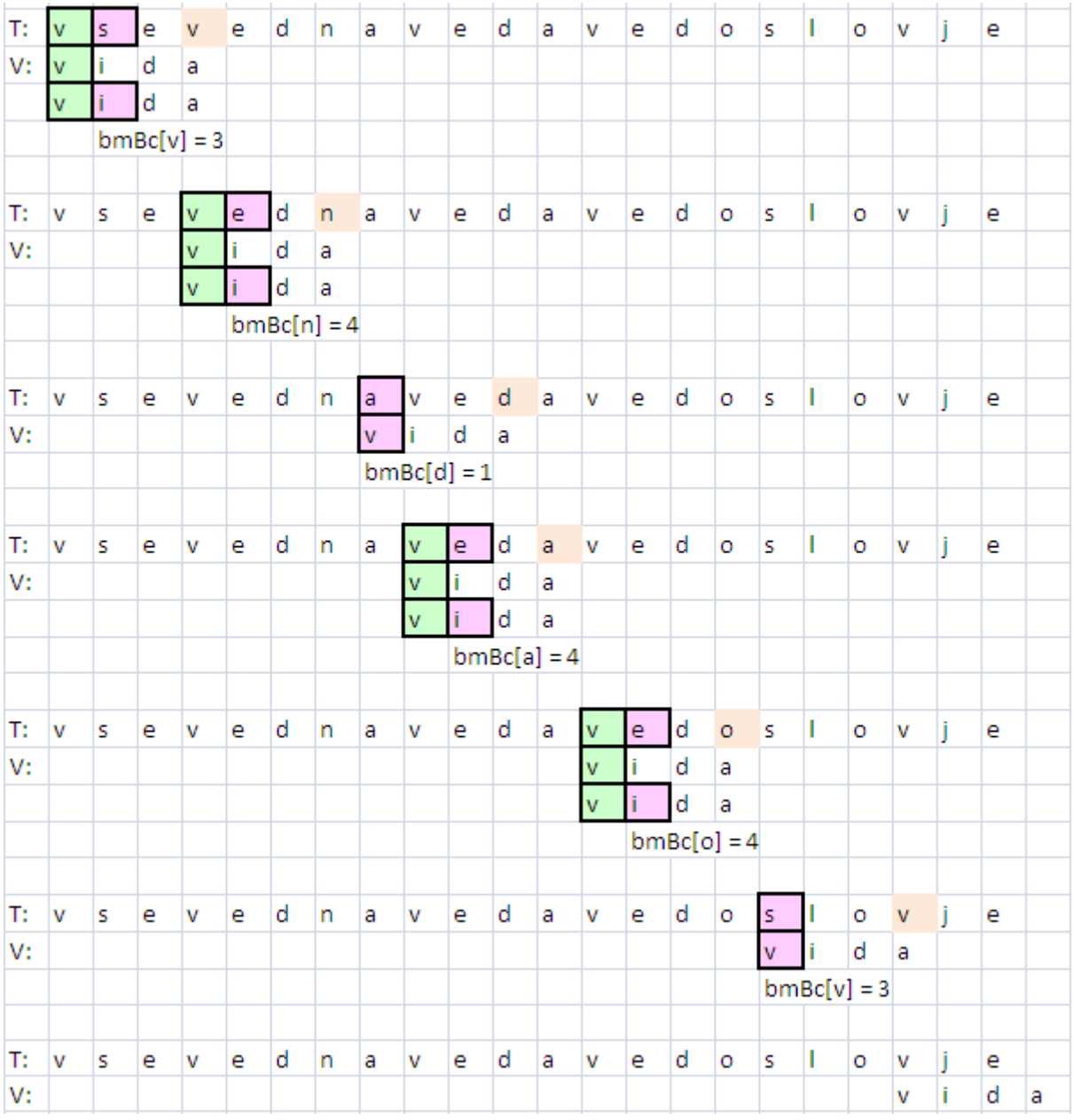
Da smo preiskali celotno besedilo, smo opravili osem primerjav posameznih znakov in vzorec premaknili šestkrat. Vzorca "vida" v besedilu nismo našli.

Horspoolov algoritem

Tu je potrebna le tabela bmBc. Vrstni red primerjanja je pri Horspoolovem algoritmu poljuben, odločili smo se za vrstni red primerjanja z leve proti desni.

c	a	d	e	i	j	l	n	o	s	v
bmBc[c]	4	1	4	2	4	4	4	4	4	3

Slika 83: Primer A, Horspoolov algoritem, tabela bmBc



Slika 84: Primer A, Horspoolov algoritem, celotni postopek iskanja

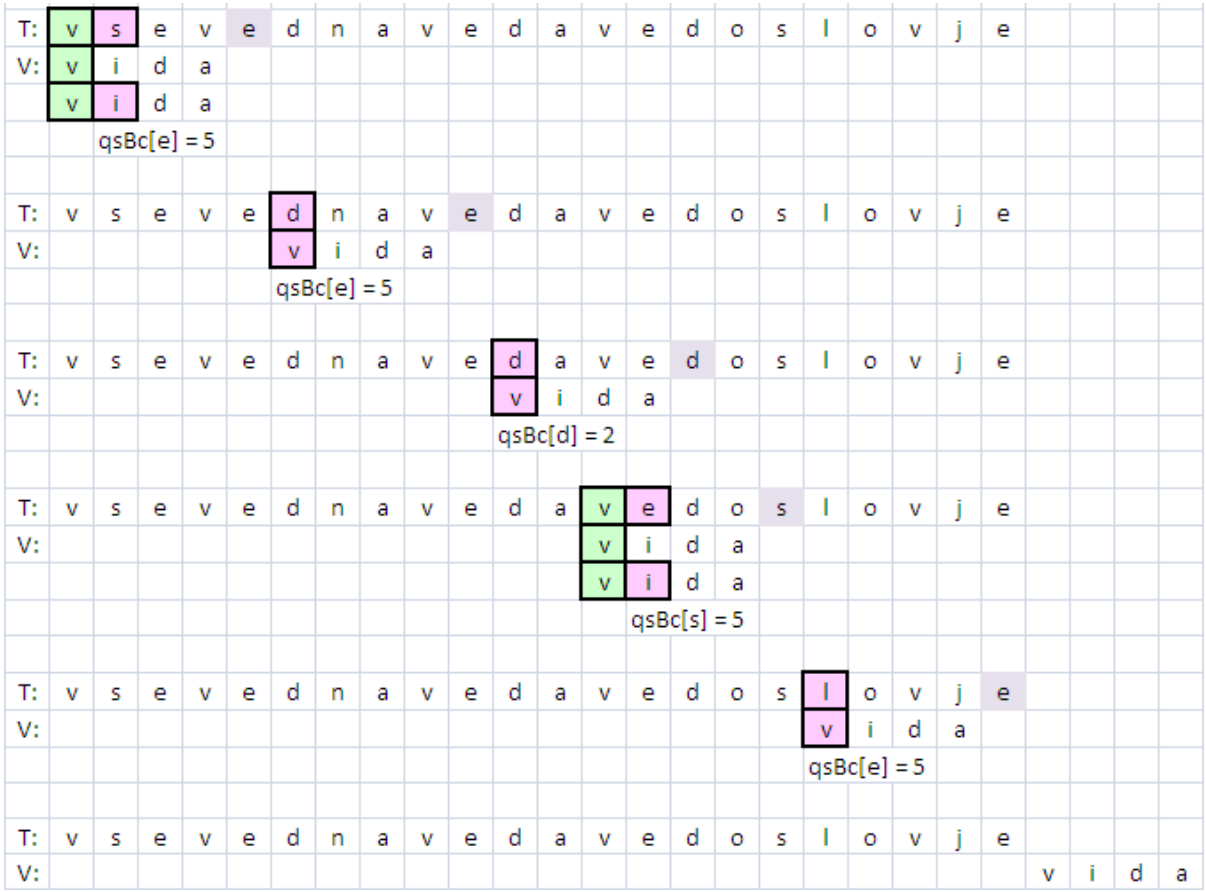
Seveda vzorca "vida" v besedilu nismo našli. Smo pa pri premikanju okna čez besedilo naredili deset primerjav posameznih znakov. Število poskusov je bilo šest.

Algoritem Quick search

Najprej tvorimo tabelo qsBc. Potem začnemo z iskanjem, ker pa je vrstni red primerjav znakov poljuben, si izberemo primerjavo z leve proti desni.

	c		a	d	e	i	j	l	n	o	s	v
qsBc[c]			1	2	5	3	5	5	5	5	5	4

Slika 85: Primer A, algoritem Quick search, tabela qsBc



Slika 86: Primer A, algoritem Quick search, celotni postopek iskanja

Z algoritmom pri primerjanju vzorca z besedilom naredimo sedem primerjav posameznih znakov v skupno petih poskusih. Popolnega ujemanja ne dobimo.

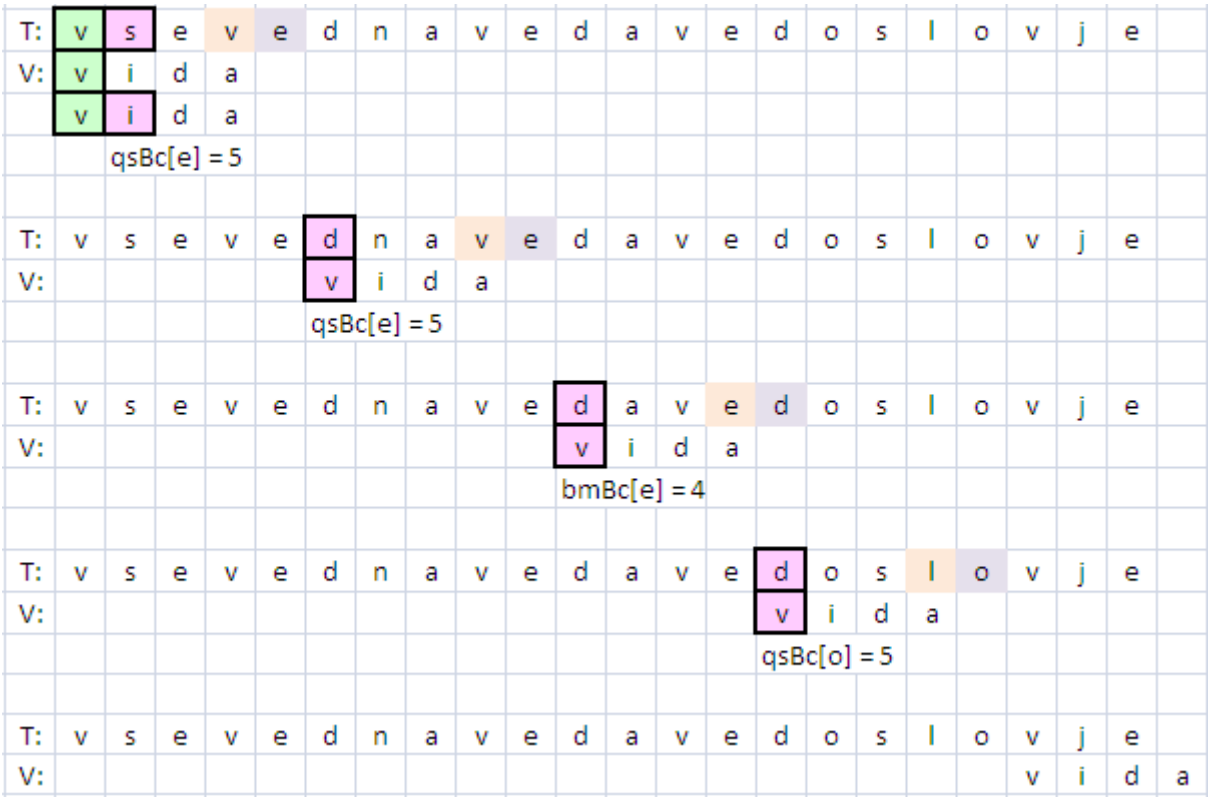
Smithov algoritem

V fazi pred iskanjem tvorimo dve tabeli, bmBc in qsBc. Sledi faza iskanja, kjer smo si izbrali vrstni red primerjanja z leve proti desni.

c	a	d	e	i	j	l	n	o	s	v
bmBc[c]	4	1	4	2	4	4	4	4	4	3

c	a	d	e	i	j	l	n	o	s	v
qsBc[c]	1	2	5	3	5	5	5	5	5	4

Slika 87. Primer A, Smithov algoritem, tabeli bmBc in qsBc



Slika 88: Primer A, Smithov algoritem, celotni postopek iskanja

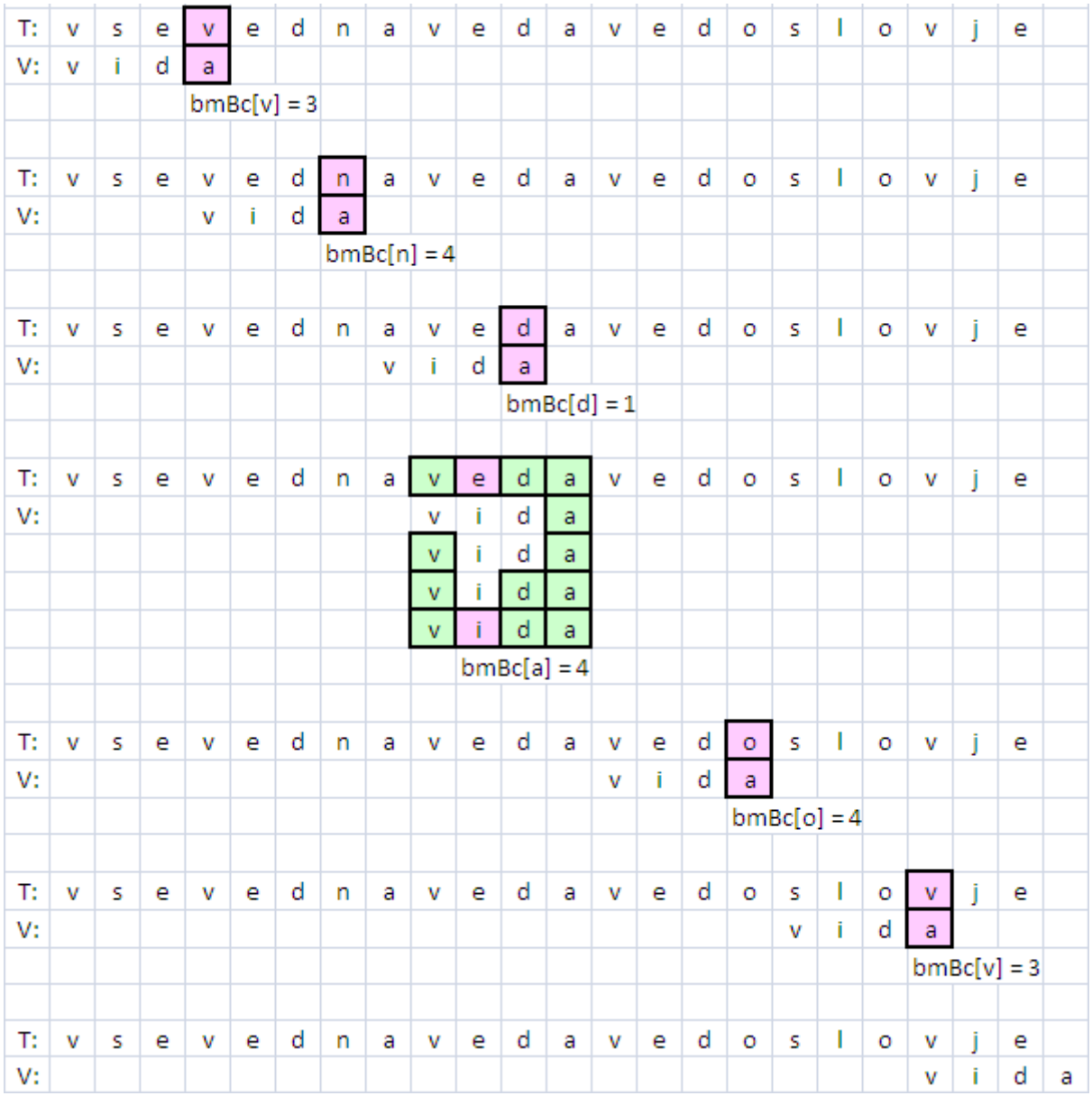
V fazi iskanja smo naredili en premik s pomočjo algoritma HP (tabela bmBc) in tri premike s pomočjo algoritma QS (tabela qsBc). Torej smo naredili štiri premike vzorca, slednji v besedilu "vsevednavedavedoslovje" ne nastopa. Število primerjav posameznih znakov v postopku iskanja je 5.

Raitov algoritem

Najprej tvorimo tabelo bmBc. Sledi postopek iskanja, v katerem je vrstni red primerjav znakov natančno določen. In sicer najprej primerjamo z znakom iz besedila zadnji znak vzorca, če se ujemata, sledi primerjava prvega znaka. Če je tudi ta primerjava uspešna, sledi primerjava tretjega znaka in ob enakosti še primerjava drugega znaka vzorca.

c	a	d	e	i	j	l	n	o	s	v
bmBc[c]	4	1	4	2	4	4	4	4	4	3

Slika 89: Primer A, Raitov algoritem, tabela bmBc



Slika 90: Primer A, Raitov algoritem, celotni postopek iskanja

Z Raitovim algoritmom naredimo šest poskusov in skupno devet primerjav posameznih znakov. Vzorec "vida" v besedilu ne nastopa.

Analiza

Povzemimo dogajanje v vseh petih algoritmih in osnovni podatke za primer A zložimo v tabelo Tabela 1.

	Boyer-Moore	Horspool	Quick search	Smith	Raita
niz	"vsevednavedavedoslovje"				
dolžina niza	22				
vzorec	"vida"				
dolžina vzorca	4				
št. poskusov (POS)	6	6	5	4	6
št. primerjav znakov (PZ)	8	10	7	5	9
št. POS, PZ do prvega popolnega ujemanja	ni popolnega ujemanja	ni popolnega ujemanja	ni popolnega ujemanja	ni popolnega ujemanja	ni popolnega ujemanja

Tabela 1: Primer A, osnovni podatki delovanja algoritmov

Že pri tako kratkem besedilu opazimo razlike pri delovanjih različnih algoritmov. Algoritem BM naredi v primerjavi z algoritmom HP manj primerjav znakov, saj poleg pravila slabega znaka, ki jima je skupno, uporablja še pravilo dobre pripone.

Algoritem QS gre še korak naprej. Pravilo slabega znaka je v tem algoritmu spremenjeno. V večini primerov predlaga za eno mesto večje premike kot algoritem HP. Zato naredi algoritem QS pri danem primeru tudi poskus manj. Prav tako opravi tri primerjanja znakov manj kot njegov neposredni »tekmec«, algoritem HP.

Smithov algoritem od vseh opisanih algoritmov naredi najmanj poskusov in primerjav znakov. To je bilo glede na neperiodičnost vzorca pričakovano, saj bi v primeru, če bi bil vzorec periodičen, najmanj poskusov in primerjav znakov naredil algoritem BM (glej primer C). Smithov algoritem je v primeru A zelo dober, saj v vsakem poskusu izbere večjega od premikov, ki ju predlagata algoritma HP in QS.

Algoritem Raita se pri rezultatih delovanja ne razlikuje veliko od algoritma HP. Pri takšnem vrstnem redu primerjav znakov, kot sem ga izbral za algoritem HP, naredi algoritem Raita eno primerjavo manj.

Primer B

BESEDILO: "ples salsa za vsak dan"

VZOREC: "salsa"

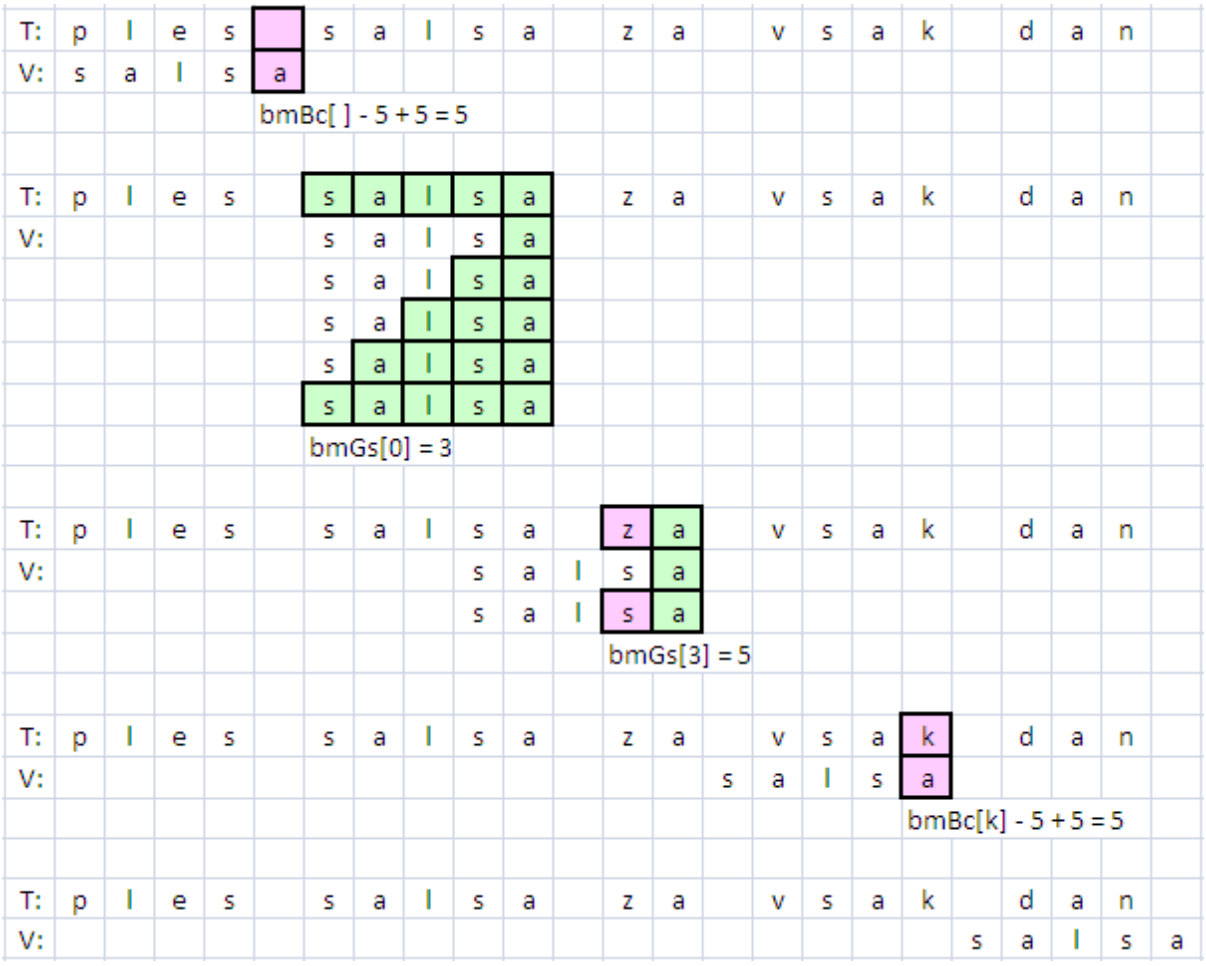
Boyer-Mooreov algoritem

Najprej tvorimo tabeli bmGs in bmBc, sledi postopek iskanja, pri katerem poteka vrstni red primerjav znakov od desne proti levi.

i	0	1	2	3	4
x[i]	s	a	l	s	a
bmGs[i]	3	3	3	5	1

c	a	d	e	k	l	n	p	s	v	z	
bmBc[c]	3	5	5	5	2	5	5	1	5	5	5

Slika 91: Primer B, Boyer-Mooreov algoritem, tabeli bmGs in bmBc



Slika 92: Primer B, Boyer-Mooreov algoritem, celotni postopek iskanja

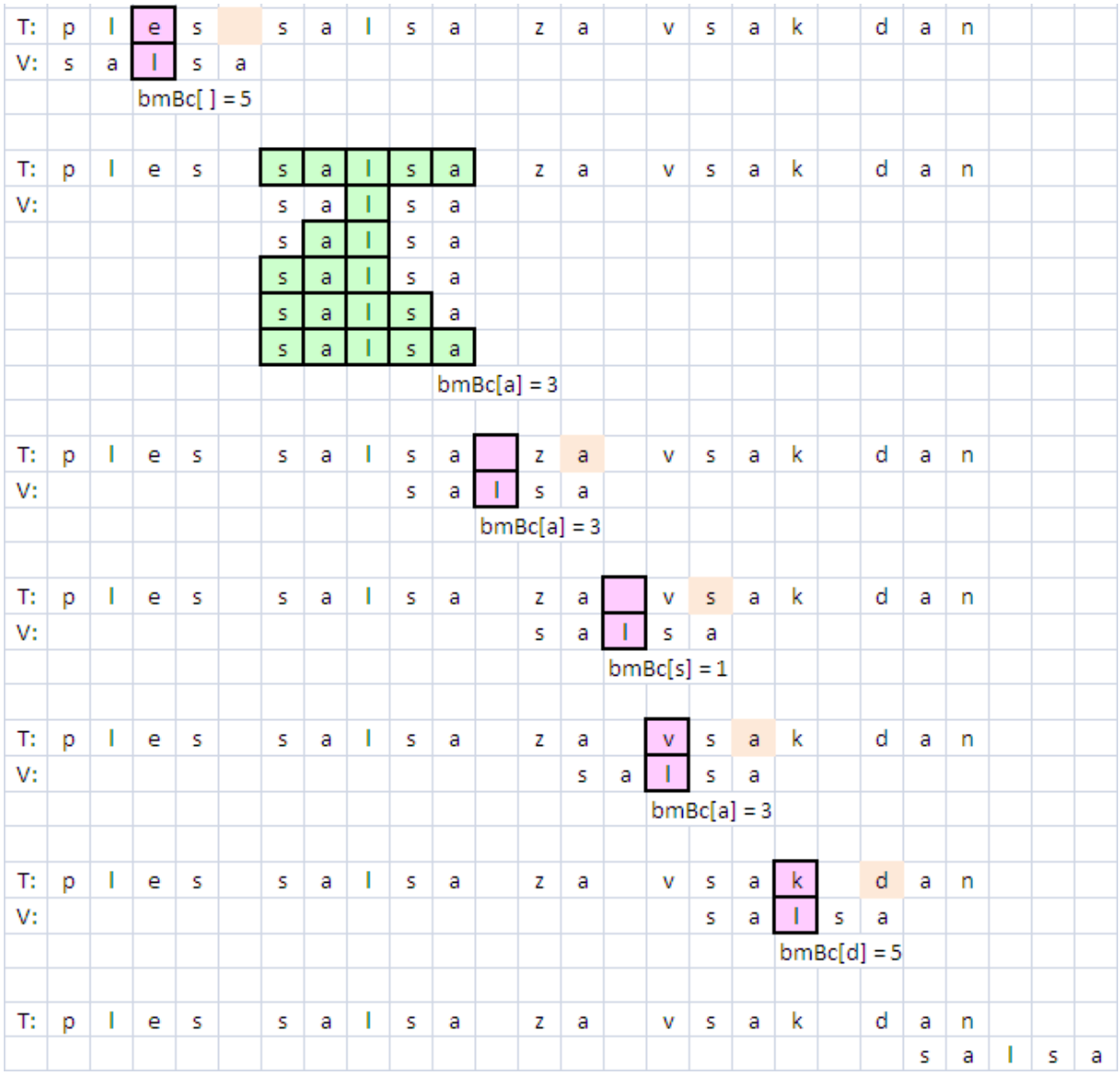
V postopku iskanja naredimo štiri premike vzorca, dva s pomočjo pravila dobre pripone in dva s pomočjo pravila slabega znaka. V celotni fazi iskanja naredimo devet primerjav posameznih znakov. Do popolnega ujemanja pridemo v drugem poskusu s šestimi primerjavami posameznih znakov.

Horspoolov algoritem

Najprej tvorimo tabelo bmBc. Pri algoritmu HP velja poljuben vrstni red primerjanja znakov, izberemo naslednji vrstni red: najprej primerjamo znak na tretjem mestu vzorca, nato na drugem, prvem, četrtem in nazadnje še na petem mestu.

	c		a	d	e	k	l	n	p	s	v	z	
bmBc[*c*]			3	5	5	5	2	5	5	1	5	5	5

Slika 93: Primer B, Horspoolov algoritem, tabela bmBc



Slika 94. Primer B, Horspoolov algoritem, celotni postopek iskanja

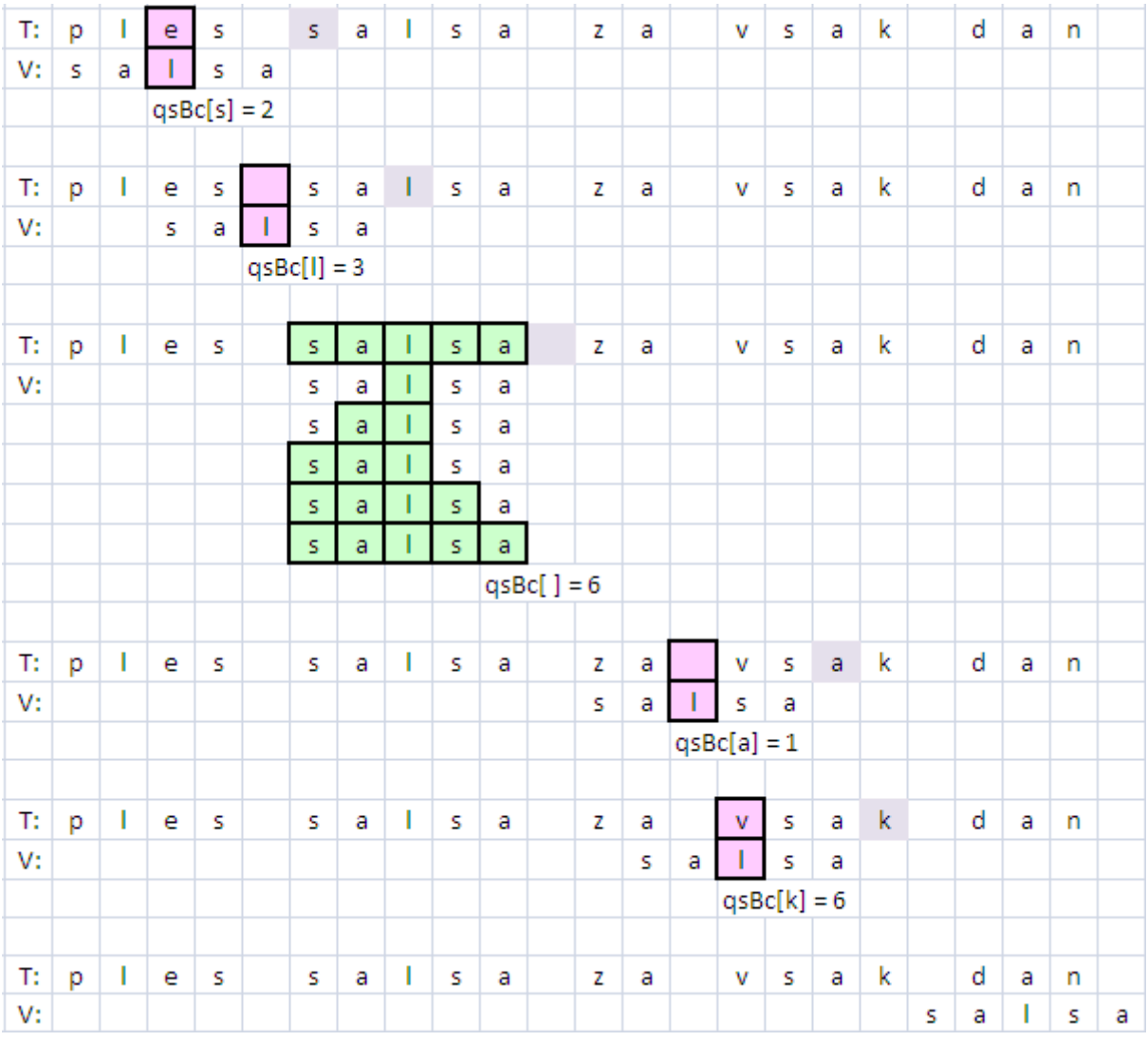
Algoritem HP do popolnega ujemanja naredi šest primerjav znakov, kar stori v dveh poskusih. Skupno je poskusov 6, primerjav posameznih znakov pa 10.

Algoritem Quick search

Najprej tvorimo tabelo qsBc. Vrstni red primerjave znakov je pri algoritmu QS poljuben. Izberemo enak vrstni red kot pri algoritmu HP, torej primerjamo najprej znak na tretjem mestu, nato na drugem, prvem, četrtem in petem mestu.

	c		a	d	e	k	l	n	p	s	v	z	
qsBc[*c*]			1	6	6	6	3	6	6	2	6	6	6

Slika 95: Primer B, Algoritem Quick search, tabela qsBc



Slika 96: Primer B, Algoritem Quick search, celotni postopek iskanja

Algoritem QS pride do popolnega ujemanja v treh poskusih, naredi sedem primerjav znakov. V celotni fazi iskanja naredi devet primerjav znakov in pet premikov vzorca (poskusov).

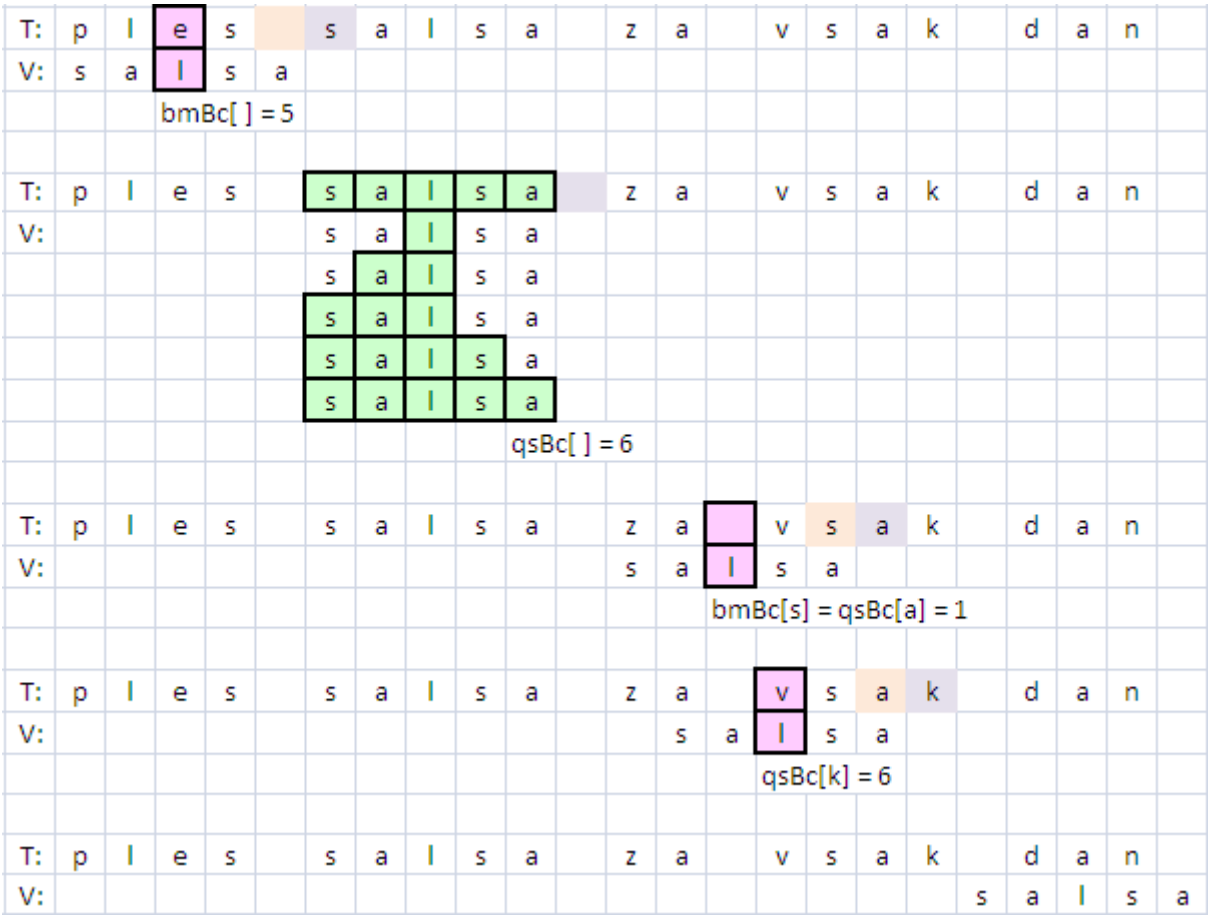
Smithov algoritem

Smithov algoritem tvori tabeli bmBc in qsBc. Za vrstni red primerjanja znakov izberemo enako zaporedje, kot pri algoritmih HP in QS. Torej se najprej primerja znak na tretjem mestu vzorca, nato na drugem, prvem, četrtem in petem mestu.

	c		a	d	e	k	l	n	p	s	v	z	
bmBc[c]			3	5	5	5	2	5	5	1	5	5	5

	c		a	d	e	k	l	n	p	s	v	z	
qsBc[c]			1	6	6	6	3	6	6	2	6	6	6

Slika 97: Primer B, Smithov algoritem, tabeli bmBc in qsBc



Slika 98: Primer B, Smithov algoritem, celotni postopek iskanja

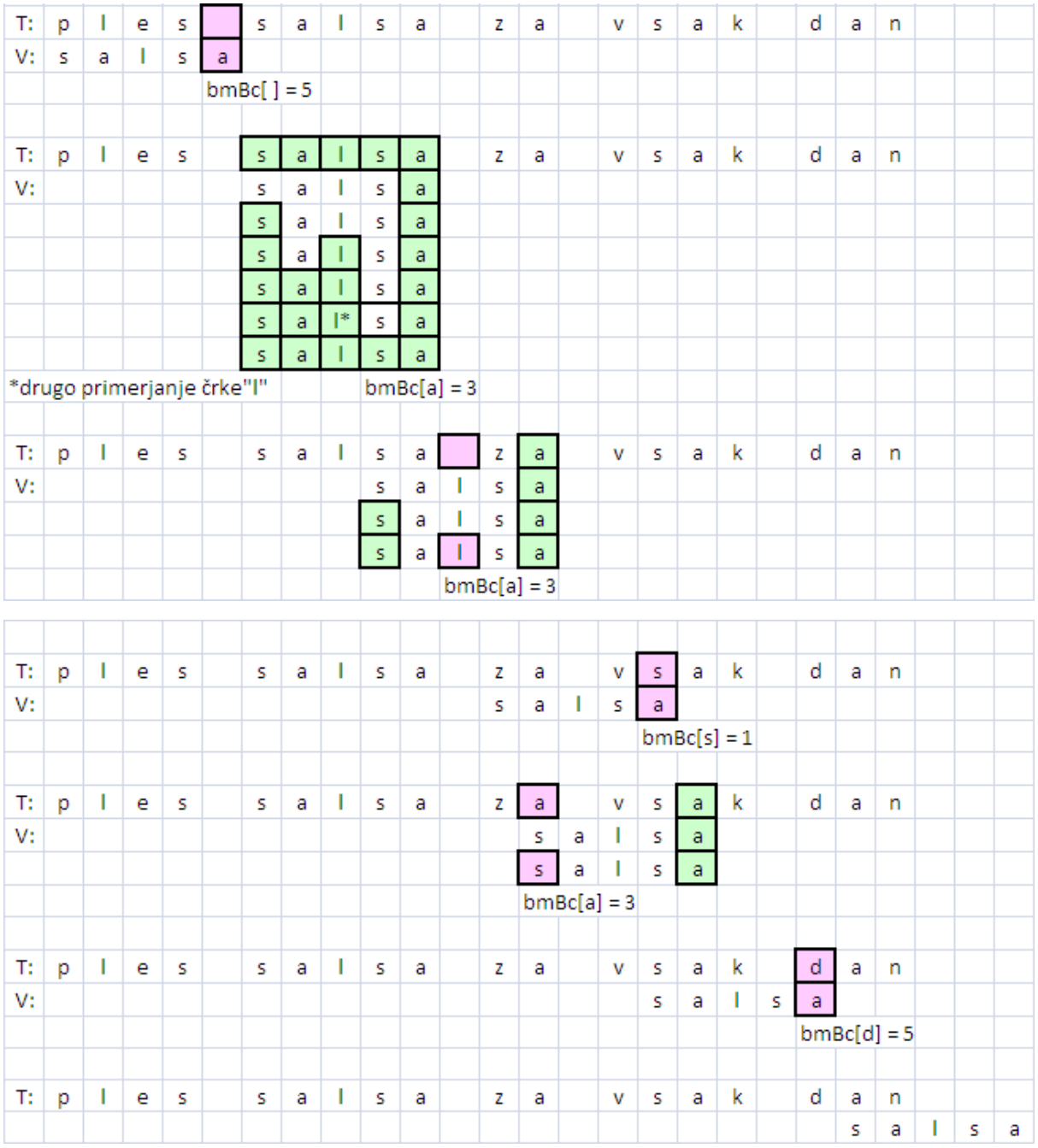
Celoten Smithov algoritem na tem primeru naredi osem primerjav znakov, do popolnega ujemanja pa šest. Število vseh poskusov je štiri, do popolnega ujemanja pa naredi dva poskusa. Od vseh štirih premikov dva predlaga algoritem QS, enega predlaga algoritem HP, pri enem premiku pa oba algoritma predlagata enako vrednost.

Raitov algoritem

Najprej tvorimo tabelo bmBc. V postopku iskanja se Raitov algoritem drži točno določenega vrstnega reda primerjanja znakov. V tem primeru se primerjava začne na petem mestu vzorca, sledi primerjava na prvem mestu, tretjem, drugem, tretjem (še enkrat) in četrtem.

	c		a	d	e	k	l	n	p	s	v	z	
bmBc[c]		3	5	5	5	2	5	5	1	5	5	5	

Slika 99: Primer B, Raitov algoritem, tabela bmBc



Slika 100: Primer B, Raitov algoritem, celotni postopek iskanja

Raitov algoritem naredi v šestih poskusih štirinajst primerjav posameznih znakov. Do popolnega ujemanja naredi dva poskusa in sedem primerjav znakov (preverjanje sredinskega znaka vzorca ob popolnem ujemanju je dvojno, tako da bi bil lahko Raitov algoritem še za malenkost hitrejši).

Analiza

Osnovne podatke delovanja algoritmov vstavimo v Tabela 2.

	Boyer-Moore	Horspool	Quick search	Smith	Raita
niz	"ples salsa za vsak dan"				
dolžina niza	22				
vzorec	"salsa"				
dolžina vzorca	5				
št. poskusov (POS)	4	6	5	4	6
št. primerjav znakov (PZ)	9	10	9	8	14
št. POS, PZ do prvega popolnega ujemanja	2, 6	2, 6	3, 7	2, 6	2, 7

Tabela 2: Primer B, osnovni podatki delovanja algoritmov

Podobno kot v primeru A naredi algoritem BM manj primerjav znakov in manj poskusov kot algoritem HP, saj poleg pravila slabega znaka, ki jima je skupno, uporablja še pravilo dobre pripone. Oba algoritma pa v besedilu najdeta vzorec v drugem poskusu s šestimi primerjavami znakov.

Algoritem QS naredi v primerjavi z algoritmom HP en poskus in eno primerjavo znakov manj, do popolnega ujemanja pa pride z enim poskusom in eno primerjavo več.

Ponovno je najhitrejši Smithov algoritem, v štirih poskusih naredi osem primerjav znakov. Do popolnega ujemanja pride enako hitro, kot algoritma BM in HP.

Vrstni red primerjav znakov pri Raitovem algoritmu za ta primer ni ravno najboljši, saj naredi največ primerjav znakov. Tudi do popolnega ujemanja pride šele po sedmih primerjavah znakov. Raitov algoritem naj bi najboljše deloval v angleških besedilih, kjer je odvisnost med znaki večja.

Primer C

BESEDILO: "hihihahahahojladrija"

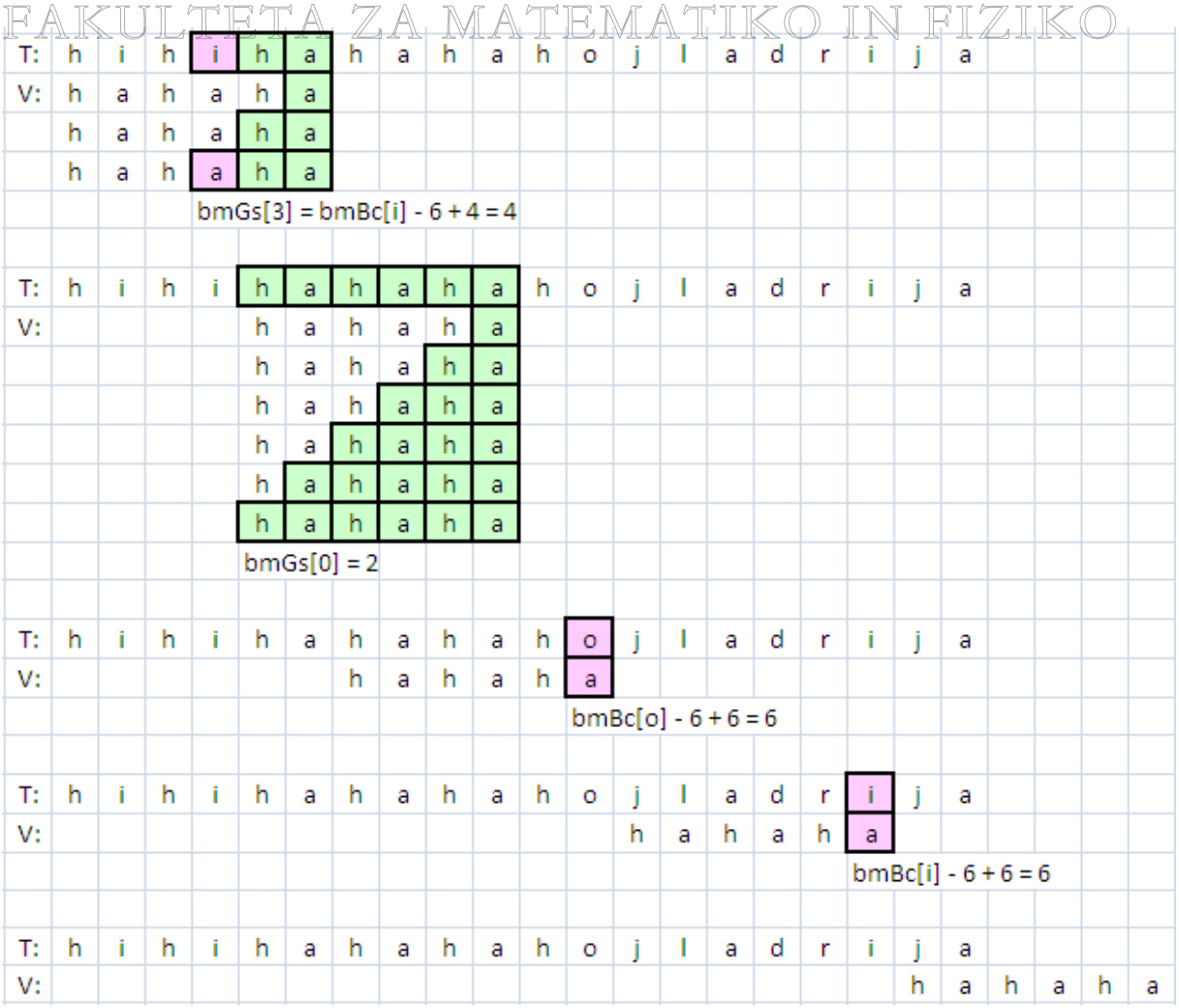
VZOREC: "hahaha"

Boyer-Mooreov algoritem

Najprej tvorimo tabeli bmGs in bmBc, sledi postopek iskanja, pri katerem poteka vrstni red primerjav znakov od desne proti levi.

i	0	1	2	3	4	5		
x[i]	h	a	h	a	h	a		
bmGs[i]	2	2	4	4	6	1		
c	a	d	h	i	j	l	o	r
bmBc[*c*]	2	6	1	6	6	6	6	6

Slika 101: Primer C, Boyer-Mooreov algoritem, tabeli bmGs in bmBc



Slika 102: Primer C, Boyer-Mooreov algoritem, celotni postopek iskanja

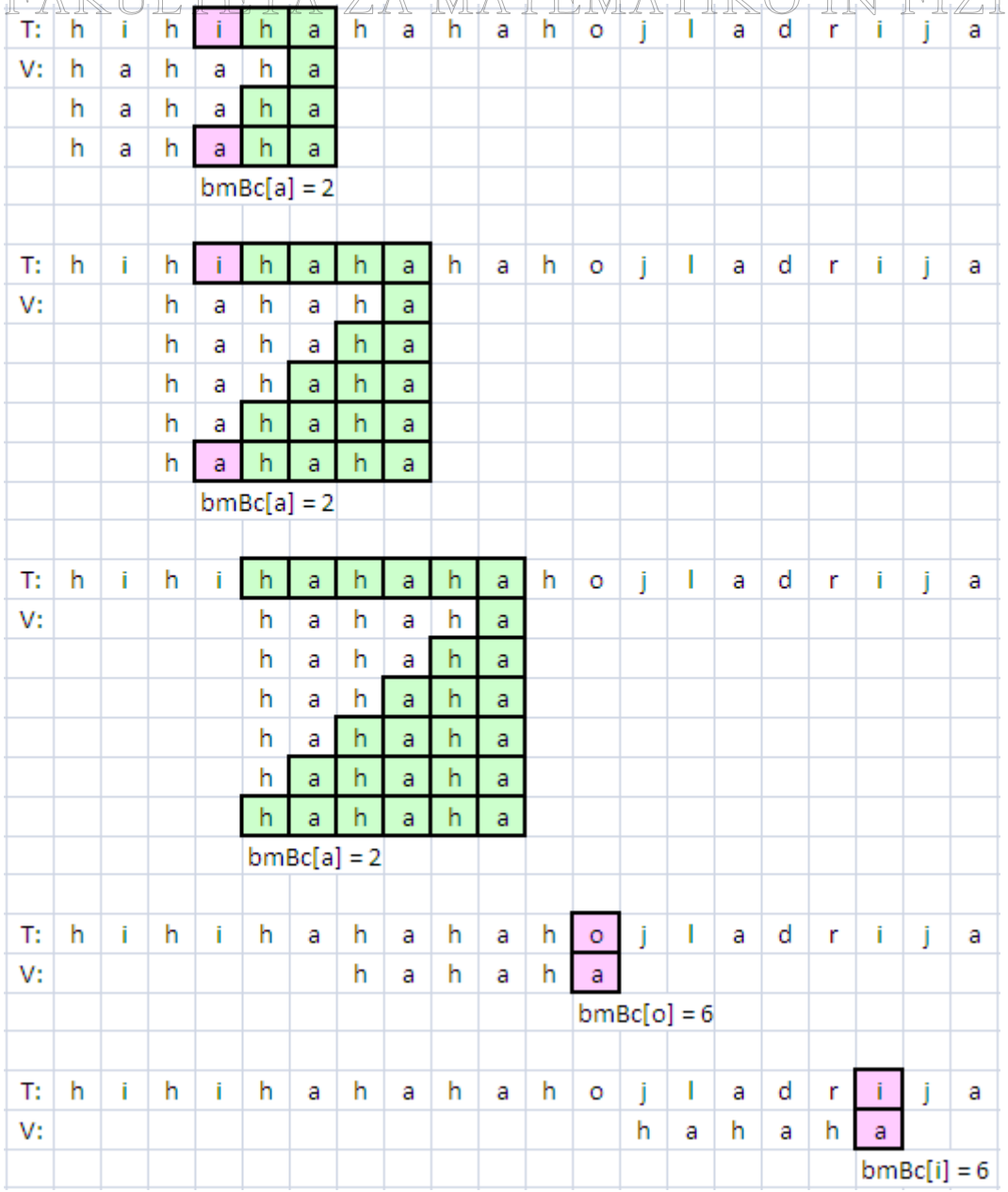
Boyer-Mooreov algoritem naredi v štirih poskusih enajst primerjav posameznih znakov. Do popolnega ujemanja naredi dva poskusa in devet primerjav znakov. V prvem poskusu predlagata enak premik pravili dobre pripone in slabega znaka. V drugem poskusu imamo popolno ujemanje, zato premik predlaga le pravilo dobre pripone. V tretjem in četrtem poskusu imamo neujemanje že pri prvem primerjanju, zato večji premik predlaga pravilo slabega znaka.

Horspoolov algoritem

Najprej tvorimo tabelo bmBc. Pri algoritmu HP velja poljuben vrstni red primerjanja znakov, izberemo primerjanje z desne proti levi.

c	a	d	h	i	j	l	o	r
bmBc[*c*]	2	6	1	6	6	6	6	6

Slika 103: Primer C, Horspoolov algoritem, tabela bmBc



Slika 104: Primer C, Horspoolov algoritem, celotni postopek iskanja

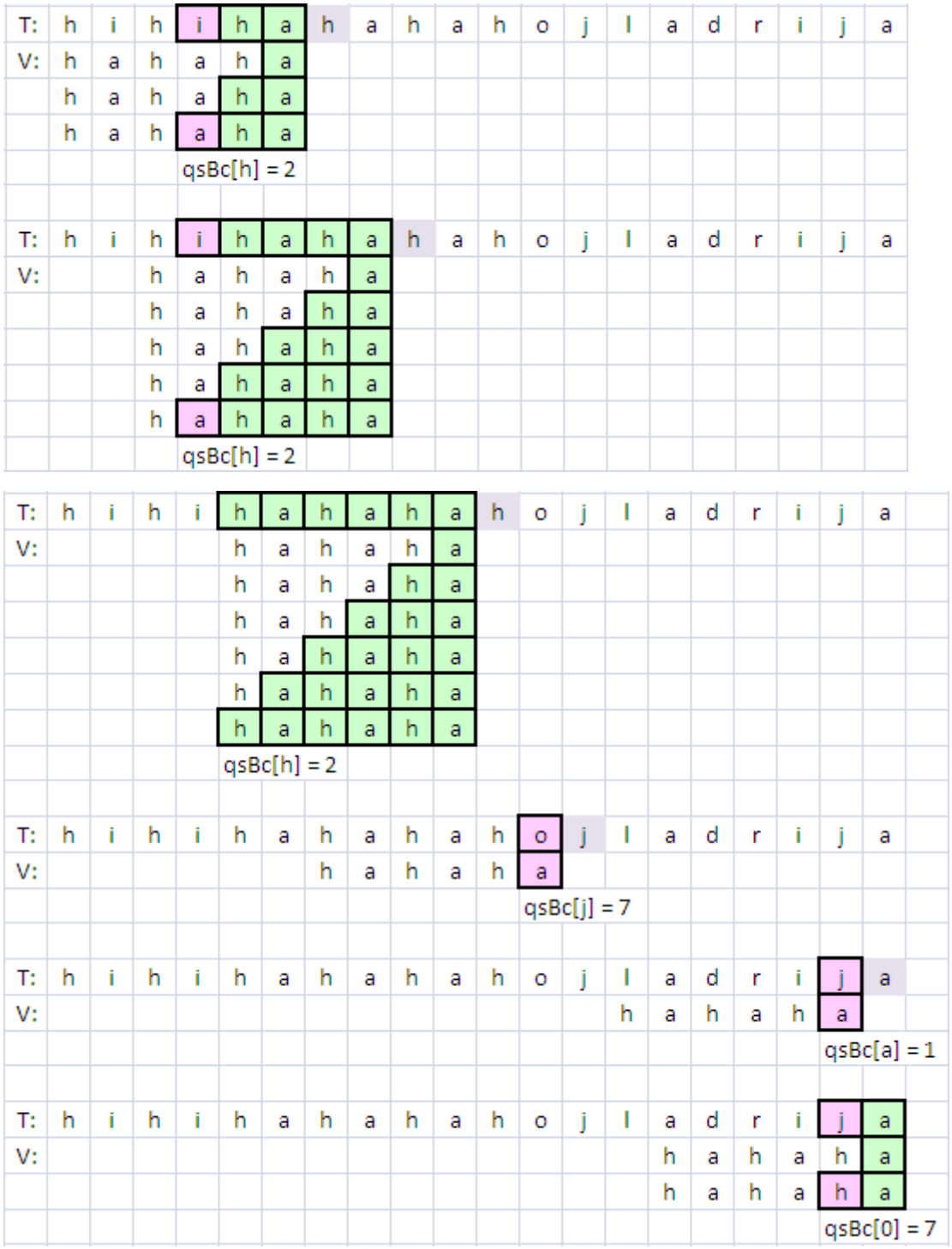
Celoten Horspoolov algoritem na tem primeru naredi šestnajst primerjav znakov, do popolnega ujemanja pa štirinajst. Število vseh poskusov je pet, do popolnega ujemanja pa naredi tri poskuse. Zadnjega premika vzorca zaradi prostorske stiske na sliki ne pokažemo.

Algoritem Quick search

Najprej tvorimo tabelo qsBc. Vrstni red primerjave znakov je pri algoritmu QS poljuben. Izberemo enak vrstni red kot pri algoritmu HP, torej primerjamo z desne proti levi.

c	a	d	h	i	j	l	o	r
qsBc[*c*]	1	7	2	7	7	7	7	7

Slika 105: Primer C, Algoritem Quick search, tabela qsBc



Slika 106: Primer C, Algoritem Quick search, celotni postopek iskanja

Algoritem QS pride do popolnega ujemanja v treh poskusih, naredi štirinajst primerjav znakov. V celotni fazi iskanja naredi osemnajst primerjav znakov in šest premikov vzorca (poskusov). Zadnjega premika na sliki zaradi prostorske stiske ne prikažemo.

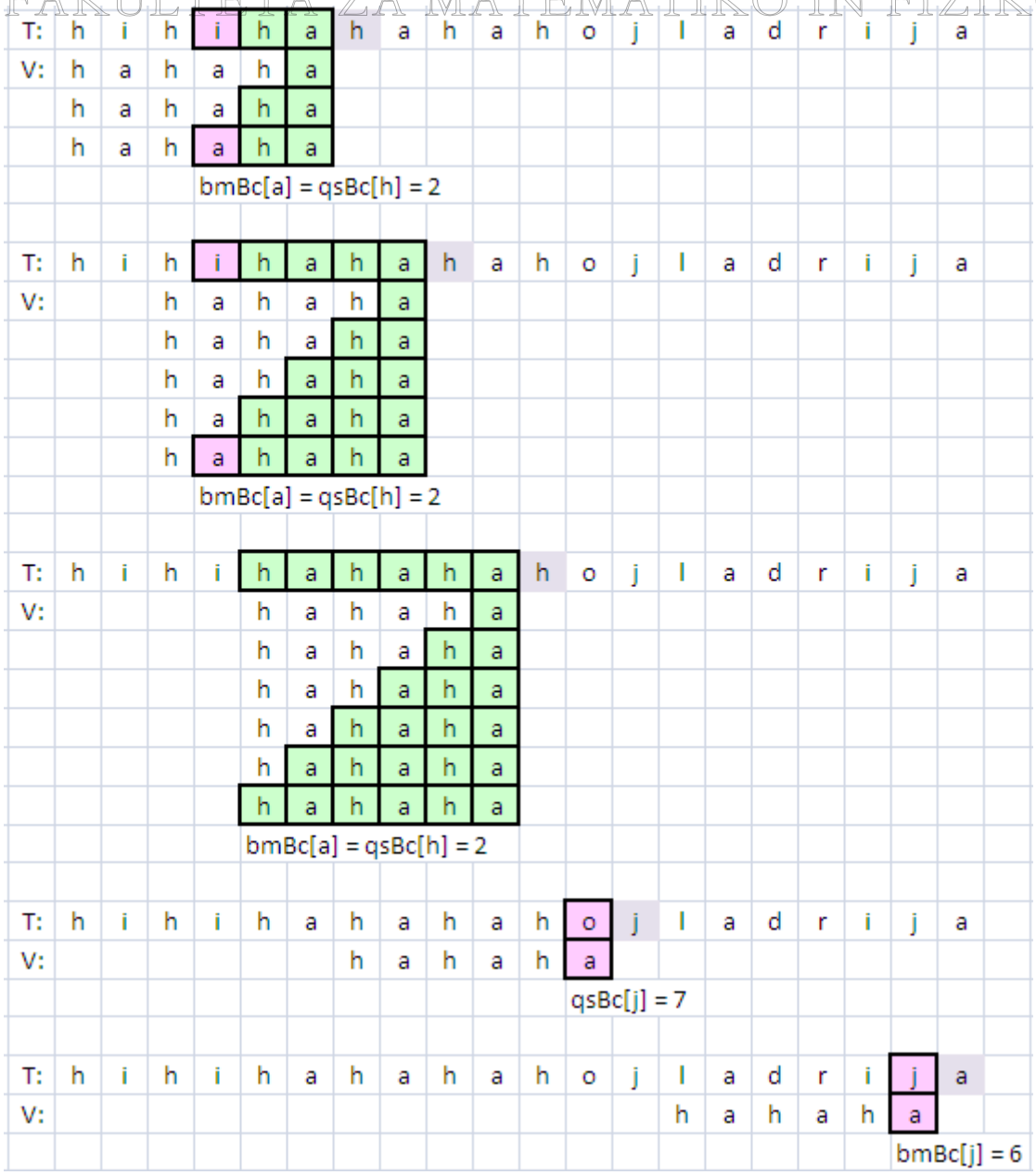
Smithov algoritem

Smithov algoritem tvori tabeli bmBc in qsBc. Za vrstni red primerjanja znakov izberemo enako zaporedje kot pri algoritmih HP in QS, torej z desne proti levi.

c	a	d	h	i	j	l	o	r
bmBc[*c*]	2	6	1	6	6	6	6	6

c	a	d	h	i	j	l	o	r
qsBc[*c*]	1	7	2	7	7	7	7	7

Slika 107: Primer C, Smithov algoritem, tabeli bmBc in qsBc



Slika 108: Primer C, Smithov algoritem, celotni postopek iskanja

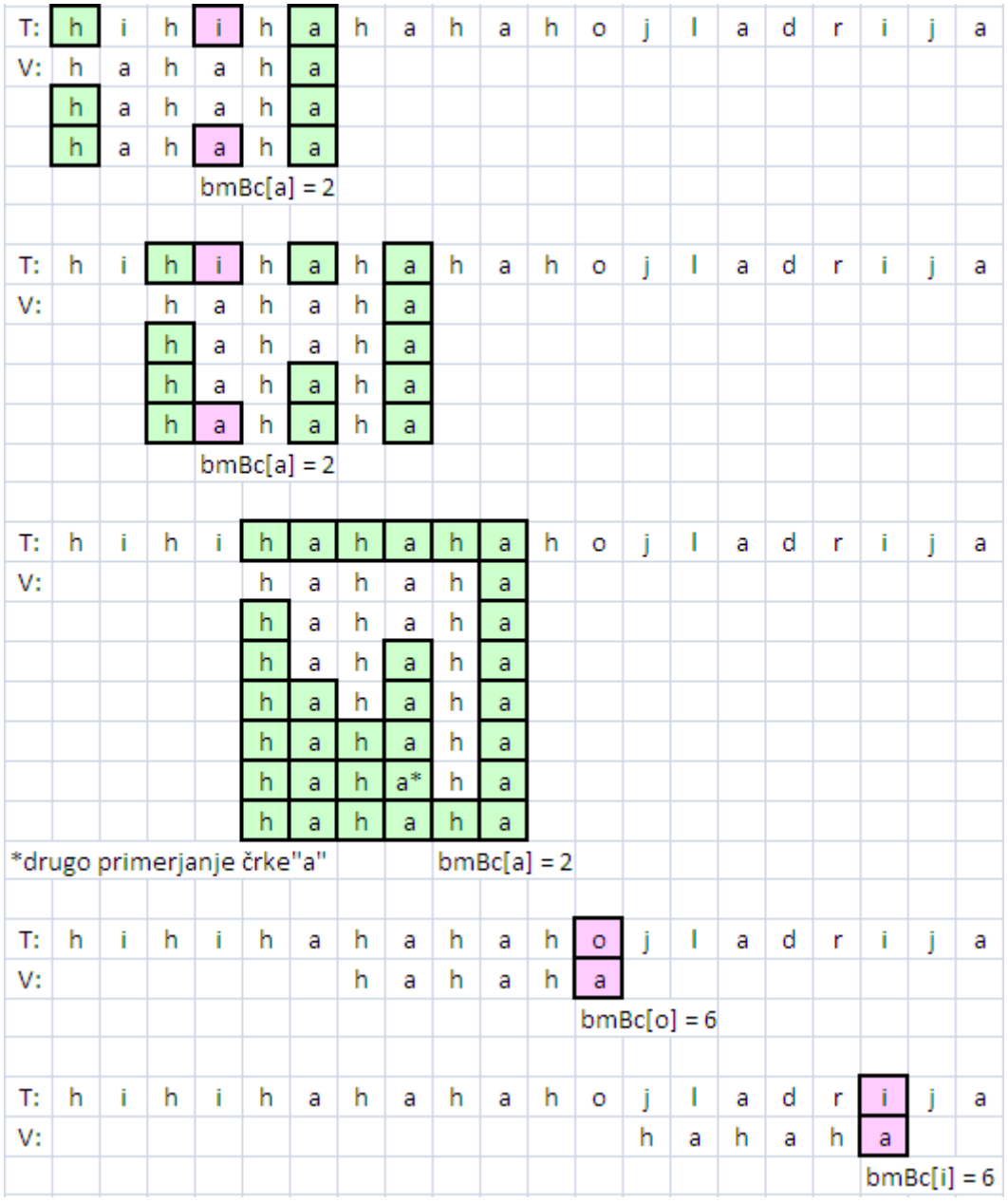
Smithov algoritem do popolnega ujemanja naredi štirinajst primerjav znakov, kar stori v treh poskusih. Skupno je poskusov pet, primerjav posameznih znakov pa šestnajst. V prvem, drugem in tretjem poskusu algoritma HP in QS predlagata enake premike. V četrtem poskusu večji premik predlaga algoritem QS, v petem pa algoritem HP. Zadnji premik vzorca zaradi prostorske stiske ni prikazan.

Raitov algoritem

Najprej tvorimo tabelo bmBc. V postopku iskanja se Raitov algoritem drži točno določenega vrstnega reda primerjanja znakov. Primerjava se začne na zadnjem mestu vzorca, sledi primerjava na prvem mestu, naslednji se primerja sredinski znak, nato pa sledijo primerjave od drugega do predzadnjega znaka vzorca (sredinski znak se primerja še enkrat).

	c		a	d	h	i	j	l	o	r
bmBc[c]			2	6	1	6	6	6	6	6

Slika 109: Primer C, Raitov algoritem, tabela bmBc



Slika 110: Primer C, Raitov algoritem, celotni postopek iskanja

V postopku iskanja naredimo pet premikov vzorca (poskusov) in šestnajst primerjav posameznih znakov. Do popolnega ujemanja pridemo v tretjem poskusu s štirinajstimi primerjavami posameznih znakov. Zadnjega premika vzorca zaradi prostorske stiske ne prikažemo.

Analiza

Osnovne podatke delovanja algoritmov vstavimo v Tabela 3: Primer C, osnovni podatki delovanja algoritmov.

	Boyer-Moore	Horspool	Quick search	Smith	Raita
niz	"hihihahahahojladrija"				
dolžina niza	20				
vzorec	"hahaha"				
dolžina vzorca	6				
št. poskusov (POS)	4	5	6	5	5
št. primerjav znakov (PZ)	11	16	18	16	16
št. POS, PZ do prvega popolnega ujemanja	2, 9	3, 14	3, 14	3, 14	3, 14

Tabela 3: Primer C, osnovni podatki delovanja algoritmov

Opazimo, da naredi v primeru C algoritem BM najmanj poskusov in najmanj primerjav znakov. Tudi vzorec najde v besedilu občutno najhitreje. To lahko pripišemo iskanju periodičnega vzorca v besedilu in pri tem upoštevanje del že ujemajočega se sklopa (pravilo dobre pripone).

Ostali algoritmu se v primeru C med seboj bistveno ne razlikujejo. Tudi Smithov algoritem, ki je bil v primerih A in B najboljši, ne izstopa.

ZAKLJUČEK

Boyer-Mooreov algoritem je le eden izmed mnogih algoritmov za iskanje podniza v nizu. Samo iz tega algoritma izhaja veliko različic, kar pomeni, da gre za obsežno temo v računalništvu. Delovanje je zelo uporabno v praksi, zato toliko izboljšav.

Sam velikokrat uporabljam funkcijo iskanja določene besede v besedilu, največkrat na spletnih straneh ali v programu Microsoft Word. Sedaj si lažje predstavljam »ozadje« iskanja določene besede. Ta na videz preprosta funkcija je v vsakdanji uporabi zelo pomembna, zato je dobila veliko izboljšav, tako da iskanje besede res poteka optimalno. Tudi zato, ker je zadeva uporabna v vsakdanjem življenju, sem se odločil za pisanje o njej.

LITERATURA

[1] R. S. Boyer, J. S. Moore: A Fast String Searching Algorithm. Communications of the ACM, 20, 10, 762–772 (1977).

[2] C. Charras, T. Lecroq, Exact string matching algorithms. Dostopno preko: http://www-igm.univ-mlv.fr/~lecroq/string/index.html, (1997) (zadnji obisk: 1. 9. 2011).

[3] Anja Rožac, 2009: Diplomska naloga Tekstovni algoritmi (online). Dostopno preko: http://rc.fmf.uni-lj.si/matija/OpravljeneDiplome/Diploma_TekstovniAlgoritmi_AnjaRozac.pdf (zadnji obisk: 5. 9. 2011).

[4] R. N. Horspool: Practical Fast Searching in Strings. Software – Practice and Experience 10, 501–506 (1980).

[5] Boyer-Moore string search algorithm. Wikipedia. Dostopno preko: http://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string_search_algorithm (zadnji obisk: 29. 8. 2011).

[6] H. W. Lang, String matching algorithms. Dostopno preko: http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/indexen.htm, (2008) (28. 8. 2011).

[7] String searching. Dostopno preko: http://www.grouse.com.au/ggrep/string.html (zadnji obisk: 28. 8. 2011).

[8] Slovar slovenskega knjižnega jezika (online). Dostopno preko: http://bos.zrc-sazu.si/sskj.html (zadnji obisk: 28. 8. 2011).

[9] Raita T., 1992, Tuning the Boyer-Moore-Horspool string searching algorithm, Software – Practice & Experience, 22(10):879–884.

[10] Spletni naslov: https://docs.google.com/viewer?a=v&q=cache:29mpwYwHprUJ:algorithm.cs.nthu.edu.tw/WSPAA07/slides/8.ppt+National+Chi+Nan+University+string+algorithm+raita&hl=sl&gl=si&pid=bl&srcid=ADGEESiX5E73k7Hp1eI0l-rdjbF45Az2qKVtMH1TSJihBWoml6uz9kURPRSekG18h250DcmSl3C8IAXa6BBkD--X1gUtaBlpyaF5gE3OenvfkOtZ_eD3WXcns-0UNm7lgg9tiiFdaDDq&sig=AHIEtbTeJYNizP58bppyd4rle28-xEbLqg (zadnji obisk: 29. 8. 2011).