Resum Recuperació de la Informació

Miquel Perello Nieto

4 de noviembre de 2012

Índice general

Capítulo 1

String matching

1.1. Exact matching

1.1.1. One pattern

Buscant un patró exacte en un text.

Brute force

L'algoritme de força bruta consisteix en comprovar per totes les posicions del text entre 0 i n-m, si el principi del patró coincideix o no amb el text. Després de cada comprovació es shifta el patró una posició a la dreta.

Aquest algoritme no requereix cap fase de preprocés, i un espai extra constant adicionalment al patró i al text.

Durant la fase de cerca la comparació dels caràcters pot ser en qualsevol ordre. La complexitat temporal d'aquesta fase es O(mn). I el nombre de comparacions esperat es de 2n.

- No hi ha fase de preprocés;
- Espai extra necessari constant;
- Sempre mou la finestra una posició a la dreta;
- La comparació es pot efectuar en qualsevol ordre;
- Fase de cerca en temps O(mn);
- Comparacions de text esperat de 2n;

Horspool

Es crea una taula amb tots els simbols de l'alfabet, amb el salt més gran que es pot fer per cada un d'ells. Quan no hi ha coincidencia es fa un salt del valor de la taula del primer simbol del text de la finestra (no el que fa fallar, sinó el primer de la finestra), desde la posició comparada.

El shift per caràcter incorrecte de l'algoritme de Boyer-Moore no és molt eficient per alfabets petits, però quan els alfabets són grans comparats amb la longitud del patró és comporta molt bé.

Utilitzant-lo sol es produeix un algoritme molt eficient en la pràctica. El Horspool proposa utilitzar només el shift de caràcter incorrecte del caràcter més a la dreta de la finestra per calcular els shifts de l'algoritme de Boyer-Moore.

La fase de preprocés es en complexitat temporal $O(m + \sigma)$ i espaial $O(\sigma)$

La fase de cerca té el pitjor cas quadràtic però es pot demostrar que el nombre mig de comparacions en un text es entre $1/\sigma$ i $2/(\sigma + 1)$.

- Simplificació de l'algoritme Boyer-Moore;
- Temps de preprocés $O(m + \sigma)$ i espai $O(\sigma)$;
- Temps de fase de cerca en $O(mn)$;
- El nombre de comparacions esperat per caràcter és de $1/\sigma$ i $2/(\sigma + 1)$.

BNDM

Es crea una taula amb una entrada per cada simbol de l'alfabet, i amb una columna d'un bit per cada posició del patró. Aquests bits són un 1 si el simbol apareix a la posició del patró i 0 al contrari.

Exemple per el patró ATGTA.

símbol	A	T	G	T	A
B(A)	1	0	0	0	1
B(C)	0	0	0	0	0
B(G)	0	0	1	0	0
B(T)	0	1	0	1	0

Un cop es té la taula s'inicialitza amb el primer caracter a comparar del text (la posicio k-esima on k es la mida del patró) i es possa a D1.

Tot seguit es shifta a l'esquerra aquest valor i es fa una & bit a bit amb el següent simbol del text, si queda tot a zeros fem un salt de la mida del patró menys X on X es el DX més gran que tingui el primer bit a 1.

Si arribem a DX amb X la mida del patró, i aquest conté un 1 en el primer bit, en DX+1 al shiftar obtindrem un patró correcte.

L'algoritme BNDM (Backward Nondeterministic Dawg Matching) utilitza una taula B, la qual per cada caràcter guarda una mascara de bits. La màscara del caràcter es crea si i només si $xi = c$.

La fase de cerca es manté en una paraula $d = dm - 1 \ldots \mathrm{d}0$, on la mida del patró m es inferior o igual a la mida de paraula de l'ordinador.

El bit di a la iteració k es posa a 1 si i només si $x[m - i..m - 1 - i + k] = y[j + m - k..j + m - 1]$. A la primera iteració "d"s'inicialitza a 1m-1. La formula per actualitzar "d"segueix $d' = (d\&B[yj]) \ll 1)$.

Existeix una coincidència si i només si després de la iteració m el bit $dm-1 = 1$.

Quan apareix una coincidència el prefix més llarg vist fins la coincidència determina el salt de finestra per la nova posició inicial.

- Variant de l'algoritme Reverse Factor;
- Utilitza la simulació paral·lela de bit;
- Eficient si la mida del patró es inferior a la mida de paraula de l'ordinador.

BOM

Primer es crea un automat finit que reconeix el patró invers. Tot seguit es possa la finestra de la mida del patró, i es comença a seguir l'automat amb els simbols del text. Si en algun moment no existeix un camí per l'automat, es salta la finestra fins la posició del text següent.

Si en algun moment s'han llegit tans simbols com la mida del patró, i s'ha pogut seguir l'automat haurem trobat el patró en el text.

Els algoritmes de tipus Boyer-Moore troben alguns sufixes del patró, però és possible trobar alguns prefixes del patró comprovant els caràcters de la finestra de dreta a esquerra i determinar la llargada del salt. Això és possible utilitzant el sufixe d'Oracle en el patró invertit. Aquesta estructura de dades es un Automat molt compacte que es capaç de reconèixer al menys tots els sufixes del patró i algunes altres paraules. La cerca de patrons amb l'algoritme utilitzant el patró invertit amb oracle es diu Backward Oracle Matching algorithm (les inicials del qual donen nom a aquest algoritme).

El sufix oracle d'una paraula "w.^{es} un Automat Finit Determinista $O(w) = (Q, q0, T, E)$.

El llenguatge acceptat per $O(w)$ es tal que {u en * : existeix v en * tal que $w = vu$} en $L(O(w))$.

La fase de preprocés del Backward Oracle Matching algorithm consisteix en calcular el sufix d'oracle per l'invers del patró xR. A pesar del fet que aquest es capaç de reconèixer paraules que no son del patró, el sufix d'oracle pot ser utilitzat per fer una comparació de paraules tenint en compte que la única paraula de mida major o igual a m és reconeguda per el patró invers de oracle.

1.1.2. k patterns

Buscant k patrons exactes en un text.

Horspool

Primer hem de crear un trie amb tots els patrons invertits, on es bifurca si el patró no és igual al anterior en el trie.

Això te un cost lineal amb la suma de la mida dels patrons.

Calculem el *lmin* que es la mida mínima, i creem una taula amb el salt més petit per cada símbol, desde el node arrel fins la següent aparició del simbol.

En el moment que seguint el tree no es trobi un camí, es fa el salt indicat a la taula per el primer simbol que s'está comparant en la finestra actual.

Wu-Manber

Existeix el problema que a mida que hi ha més patrons els salts tendeixen a mida 1, per resoldre això es fa la taula amb més d'un simbol, d'aquesta manera s'augmenta la mida del salts pero es fan més comparacions.

Existeix una formula per crear la mida de simbols per la taula óptima log—E— 2*lmin*k (on E és la mida de l'alfabet, i k és el nombre de patrons).

SBOM

Es genera el factor-oracle invers de tots els patrons junts.

Primer es va fent el camí d'un patró, en el qual en cada nou simbol s'ha de seguir un sufix-link anterior, o crear-ne un cap a un estat que tingui com a procedent del simbol nou. Un cop seguit el sufix-link es mira si existeix algun camí amb el simbol nou desde aquest estat, si no hi és es crea cap a l'estat nou creat. I un cop fet això es torna a reseguir els sufix-links que quedin repetint el mateix.

Un cop creat per un patró, reseguir aquest amb els altres patrons.

Amb factor d'oracle creat, mirem el lmin, per fer la finestra d'aquesta mida, i comencem a reseguir el cami amb el text. En el moment que falli, podem fer un salt de finestra fins la següent posició.

Per trobar un patró s'ha de seguir el factor d'oracle correctament, arribant a una posició final. Un cop en aquesta posició s'ha de comprovar que realment existeix el patró, i tot seguit s'ha de seguir fos també un subpatró d'un altre.

1.1.3. Extensions en el text

Les extensions són simbols que poden ser varis simbols de l'alfabet, per tant vindría a ser una extensió dels simbols de l'alfabet.

Brute force

Es crea una taula amb una entrada per cada simbol (extesos inclosos) i una columna per cada simbol de l'alfabet. Aquestes columnes contenen un 1 si el simbol de la fila contempla aquest simbol de l'alfabet.

Despres al fer la cerca es comprova si la fila del simbol del patró a comparar es continguda per el simbol del text amb una and. Si aquesta and es més gran a zero seguim.

Horspool

Es crea la shift-table amb els minims salts incloent els nous simbols del patró, amb el salt mes petit que li correspon amb el simbol amb salt mes petit.

BNDM

Es crea la mateixa taula que en el cas normal de BNDM, pero afegint els simbols extesos, amb la posició del patró que li pertoca a 1.

La seva execució es exactament igual que en el anterior cas.

1.1.4. Extensions en el patró

Horspool

Es crea la shift-table només amb els simbols de l'alfabet original, pero els salts venen determinats per el patró. El problema que apareix es que els salts es fan molt petits.

BNDM

Es crea la taula amb una fila per cada simbol original de l'alfabet, i es posen a 1 les columnes que continguin en el patró el simbol original o un extesj que correspongui.

SBOM

S'utilitza el SBOM amb tots els diferents patrons que es poden generar.

BNDM amb Bounded Length Gaps

Per utilitzar l'algoritme de BNDM amb un nombre de gaps variable determinat en el patró s'utilitza la següent técnica:

Per el patró ATx(2,4)TA ; en el qual poden haver dos, tres o quatre gaps.

símbol	A	T	x	x	x	x	T	A
B(A)	1	0	1	1	1	1	0	1
B(C)	0	0	1	1	1	1	0	0
B(G)	0	0	1	1	1	1	0	0
B(T)	0	1	1	1	1	1	1	0

Un cop amb aquesta taula s'executa el BNDM tenint en compte que si apareix un 1 en la part esquerra dels gaps,

TODO: no ho tinc molt clar, deixo la formula i ja m'ho miraré.

$D \leftarrow [F - (I\&D)]\&\neg F$

1.1.5. Regular Expressions

El llenguatge definit per una expressió regular R es el conjunt de paraules generades amb R.

El problema de buscar una expressió regular en un text T es trobar tots els factors en T que pertanyen al llenguatge.

Per això hi ha dos metodes.

Cerca amb Automat Finit Determinista

Amb l'expressió regular es genera un NFA (Nonedeterministic Finite Automaton).

Es transforma en un DFA (Deterministic Finite Automaton).

Es busca amb el DFA.

Cerca amb Automat de bit-paral·lel de Thompson

Amb l'expressió regular fem un parse tree.

Tot seguit es passa a un NFA de Thompson.

Es crea una taula amb una fila per cada un dels estats que tenen e-closure (son buits), amb tots els estats als que poden anar saltan més estats e-closure.

Es crea una altra taula amb tots els simbols de l'expressió regular i un bit per cada estat. Aquest bit estará a 1 si en aquell estat s'ha arribat afegint aquest simbol.

A partir d'aqui es comença l'execució de l'algoritme Bit-parallel de Thompson.

Els estats es representen per una tira de bits amb mida "nombre d'estats".

La primera comparació amb el text es crea l'estat inicial D0 amb el primer bit a 1.

Es shifta a la dreta i es fa una and amb el simbol de la segona taula feta. Si el valor es zero finalitza la comprovació d'aquest caracter en el text.

Si no es zero es mira en quin estat es troba el 1, i s'inicialitza el nou D1 amb el valor de la primera taula i l'estat indicat. Es shifta a la dreta i es fa la and amb el simbol del text llegit.

Aixo es repeteix fins que apareix un 1 en l'ultim estat...

TODO: no ho explica en les transparencies.

1.1.6. The text

1.1.7. Suffix trees

Primer s'afegeix un simbol nou al final del text. Aquest simbol servirá per indicar el final dels sufixes.

Es crea un suffix tree del text, aquest arbre primer té tot el text. A mesura que s'incrementa la posició del text es va introduint de nou tot el text, pero desde la nova posició, i en cada diferencia en l'arbre s'afegeix una nova ramificació. Al final de cada inserció de sufix, s'indica quina era la seva posició inicial.

Un cop generat aquest arbre es pot buscar si un patró apareix en el text en temps lineal respecte la mida del patró. Saber la posició on es troba el patró en temps lineal respecte la mida del arbre.

El cost de generar l'arbre pot ser

1.1.8. Suffix arrays

Es creen tots els sufixes ordenats per posició del sufix. I es crea una taula amb aquests ordenats lexicograficament amb un cost nlog(n).

Un cop ordenat es fa una cerca del patró en la taula amb un cost $log(n)+|P|$.

1.2. Approximate matching

Buscar un patró en un text amb alguns errors.

Ens cal per tant crear el concepte de distancia, que ens indica quin nombre d'errors hi ha entre dos patrons.

1. Mismatch : T - A
2. Insertion : TT - TAT
3. Deletion : TAT - TT

1.2.1. Dynamic programming

Es crea una matriu amb les columnes amb el text, i les files amb els simbols del patró; tots dos començant amb GAP (element buit representat amb un guió ").

En cada casella es possa la distancia mínima entre el prefix del patró i el sufix del text en aquella posició. Per trobar aquesta distancia mínima, en cada casella es segueix la mateixa norma:

seleccionar el mínim de :

- Inserció : valor lateral + 1

- Missmatch : valor diagonal + 1
- Identitat : valor diagonal + 0
- Deleció : valor superior + 1

Si només estem interesats en la distancia del patró amb el text, sense tenir en compte el sufix del text sencer, el que es fa es inicialitzar la primera fila de la matriu a 0, donant a entendre que tot el sufix del text amb el primer caracter del patró només contindrá la distancia del propi error de la casella, per els criteris abanç esmentats.

1.2.2. Sequence alignment (pairwise and multiple)

Amb l'alineament de seqüencies cambiem els valors de distancies per els següents:

- Inserció : valor lateral - 2
- Missmatch : valor diagonal - 1
- Identitat : valor diagonal + 1
- Deleció : valor superior - 2

Per tant el que volem ara es maximitzar el valor, i penalitzar les insercions i delecions.

Per alinear parelles de seqüencies es faría exactament igual que en el cas de programació dinàmica, i amb un nombre k de seqüencies es crearía una matriu de k dimensions, i s'hauría de calcular de manera similar pero de totes les posicions adjacents ja visitades. Això tindría un cost exponencial (en concret $n^k 2^k k^2$) per tant aquest metode no es factible.

El que es fa es crear una matriu de puntuacions amb totes les seqüencies, per exemple de $k * k$. Es comprova quin es el millor alineament i un cop escollit es deixa la parella. Es repeteix el proces el següent cop amb una matriu de puntuació de $(k-1)*(k-1)$ i en les comparacións es te en compte la puntuació de la suma dels alineaments.

Al repetir aquest proces al final tenim el millor alineament.

1.2.3. Sequence assembly: hash algorithm

A partir de sequencies agafar els seus troços i sequenciar (per exemple un genoma). Existeixen dos tipus

1. Hibridization: informació sobre l-tuples de la seqüencia.
2. Shotgun: la seqüencia es trenca en fragments aleatoris.

Hibridization

S'escull un valor de l, i es parteix tota la seqüencia en trosos d'aquesta longitud (realment es possa la seqüencia en solucions que s'activen en cas de que contingui la seqüencia de mida l que s'está buscant. Un cop es sap ja tens les particions ficticies).

Un cop es tenen totes aquestes particions (sense saber la quantitat de cadascuna) s'intenta veure totes les tuples que es poden enllaçar entre elles per sufix-prefix en l-1 simbols.

Això te un cost $O(n^2 l)$ on n^2 són totes les tuples amb totes les tuples i l per cada una de les tuples.

Realment es busca un camí Hamiltoniá, en el que el camí passa per tots els nodes un unic cop, i això te un cost NP-Complet.

1. Buscar els l-mers: cost $simbols^l$
2. Buscar les coincidencies en sufix-prefix : cost $O(m^2 l)$
3. Buscar el camí Hamiltonia: cost NP-Complet

Una aproximació és trobar un camí Eulerià amb un cost lineal.

S'agafa un node com a inicial, i es va recorrent el camí apuntant totes les interseccions fins que ens topem amb un node que ja haviem visitat.

En aquest moment mirem el primer node que tenia intersecció i el reseguim fins que arribém a éll mateix. En aquest punt fem que l'antic camí contingui aquest de nou. I seguim per totes les interseccions que encara tinguin altres nodes no visitats.

1. Buscar els l-mers: cost $simbols^l$
2. Buscar les coincidencies en sufix-prefix : cost $O(m^2 l)$
3. Buscar el camí Eulerià: cost lineal.

Aquests tenen el problema de les repeticions. Les quals no tenen cabuda en el sistema.

Shotgun

Agafa moltes copies de la seqüencia, i les parteix a l'atzar, tot seguit intenta solapar les seqüencies per els punts que són iguales.

Per trobar totes les parelles sufix-prefix es pot fer amb programació dimàmica amb cost quadratic, o l'altre opció.

Pimer amb un hash trobar les parelles susceptibles a ser-ho.

Despres amb programació dynamica mirar si enganxen.

1.3. Probabilistic search: Hidden Markov Models

Es poden resoldre tres tipus de problemes amb aquests models:

- Evaluació del problema:

  Quina probabilitat hi ha de que una seqüencia donada hagi estat generada per un model concret?

  Entrada : Un HMM M i una seqüencia x

  Sortida : Trobar la $Prob[x|M]$

- Decodificant el problema:

  Identificar amb quins estats han estat generades les diferents parts de la seqüencia.

  Entrada : Un HMM M i una seqüencia x

  Sortida : Trobar la seqüencia π que maximitza $P[x, \pi|M]$

- Problema d'aprenentatge:

  Identificar el nombre d'estats que es probable que hi hagi, i quines probabilitats hi ha en cada un.

  Entrada : Un HMM M sense una especificació de les probabilitats θ transisió/emisió i una seqüencia x

  Sortida : Paràmetres de θ que maximitzen $Prob[x|\theta]$

1.3.1. Evaluació del problema

Per generar una seqüencia x existeixen $2^{|}x|$ camins.

La computació de Naïve es molt costosa, donats $|x|$ caracters i N estats existeixen $N^{|}x|$ possibles estats.

Per resoldre el problema fem servir el "Forward Algorithm".

Suposant que la primera probabilitat es 1, i que la probabilitat de cada estat es la suma de les probabilitats dels seus estats antecesors multiplicats per el camí que els porta al nou estat.

1.3.2. Decodificant el problema

1.3.3. Problema d'aprenentatge