Note asupra realizării unui algoritm paralel de căutare a unui şablon într-un text

Lect. dr. Teodor Florin Fortiş
Universitatea de Vest, Timişoara

ABSTRACT. The purpose of this article is to realize an analysis for two methods that can be considered in order to create a parallel algorithm for pattern searching into a text. First, we are dealing with "Divide-and-Conquer" type approaches. Next, we are considering another type of approach, based on the decomposition of the initial problem into several simpler problems and with different initial data, but similar solutions. An algorithm for the later situation is sketched, in order to reveal some of the (possible) features of algorithms of this type.

Key words: pattern matching, string searching, text algorithms.

1 Probleme clasice de căutare

Problemele de căutare a şirurilor de caractere se dovedesc a fi un subiect extrem de important într-o varietate de domenii, în principal legate de procesarea textelor. Multe sisteme software se bazează pe funcţionarea unor algoritmi de căutare eficienţi, aceştia oferind de cele mai multe ori nu numai un sprijin sistemelor software dar chiar şi modele de programare.

Informaţia de bază este depozitată, în multe domenii ale ştiinţelor, într-o manieră lineară, similară şirurilor de caractere. Acest mod de depozitare a informaţiei este evident în domenii filologice, dar şi în alte domenii incluzând informatica (unde informaţia poate fi stocată în fişiere de dimensiuni mari) sau biologia (unde moleculele pot fi descrise oferind o secvenţă de nucleotide şi amino-acizi). Cantitatea mare de date care urmează să fie procesată presupune existenţa unor metode eficiente de identificare a informaţiei.

Tibiscus

O problemă de căutare presupune identificarea primei/unei apariţii (sau a tuturor apariţiilor) a unui şir de simboluri dintr-un alfabet, numit şablon, într-un text. Atât şablonul cât şi textul sunt construite peste o mulţime finită de caractere, numită alfabet.

În funcţie de modul în care sunt oferite informaţiile de bază, pot fi realizate mai multe abordări ale acestei probleme: cunoaşterea prealabilă a şablonului face posibilă utilizarea unui automat sau a anumitor proprietăţi ale şablonului, pe când cunoaşterea prealabilă a textului face posibilă utilizarea unor proprietăţi statistice ale acestuia în scopul îmbunătăţirii performanţelor algoritmilor.

Funcţionarea algoritmilor bazaţi pe cunoaşterea prealabilă a şirului de căutare poate fi aproximată astfel:

- textul este parcurs prin intermediul unei ferestre, a cărei dimensiune este egală cu dimensiunea şablonului;
- fereastra este aliniată iniţial la stânga cu textul, realizându-se o comparaţie a conţinutului ferestrei cu conţinutul şablonului. Această operaţie este numită încercare ;
- după fiecare încercare este realizată o translatare a ferestrei spre dreapta, algoritmul pregătindu-se pentru o nouă încercare;
- fereastra este mutată către dreapta până la atingerea marginii drepte a textului în care se realizează căutarea.

Aplicarea directă a acestei metode oferă algoritmul forţei brute, capabil să identifice toate apariţiile unui şablon într-un text într-un timp $O(m \times n)$, unde m este mărimea şablonului iar n este dimensiunea textului. Creşterea vitezei algoritmului este posibilă prin realizarea unor îmbunătăţiri aduse felului în care este realizată comparaţia în interiorul ferestrei: de la stânga la dreapta, de la dreapta la stânga, într-o ordine bine precizată sau fără restricţii.

În prima categorie intră algoritmii care se bazează pe funcţii de hashing (Harrison, Karp-Rabin) sau recunoaşterea unui şir de caractere cu ajutorul unui automat (Morris-Pratt, Knuth-Morris-Pratt). A doua categorie conţine unul dintre cei mai eficienţi algoritmi utilizaţi în practică (Boyer-Moore) sau variante ale acestuia (Colussi, Turbo-BM, şi altele).

A treia categorie de algoritmi (în care comparaţia este realizată într-o ordine bine precizată) oferă algoritmii cu cele mai bune performanţe teoretice, (Galil-Giancarlo, Galil-Seiferas, Crochemore-Perrin, etc.). Aceştia se bazează de cele mai multe ori pe algoritmi din primele două categorii, oferind o modificare simplă a acestora.

2 Abordări paralele pentru căutarea unui şablon într-un text

2.1 "Divide-and-conquer"

Cea mai simplă şi, probabil, cea mai directă modalitate de construire a unui algoritm paralel de căutare a unui şablon într-un text presupune utilizarea conceptelor "Divide-and-conquer" pentru construirea subproblemelor care urmează să fie abordate în paralel. În acest fel, problema de căutare principală este descompusă în mai multe probleme de căutare pe bucăţi distincte ale textului iniţial. Fiecare dintre procesele participante la rezolvarea paralelă va avea de realizat căutarea aceluiaşi şablon în bucăţi diferite de text.

În mod ideal, în condiţiile în care problema principală este descompusă în n subprobleme, viteza de căutare ar putea creşte de cel mult n ori. Cu toate acestea, nici unul dintre procesele participante nu va putea lua nici un fel de decizie la capetele textului pe care-şi desfăşoară activitatea, astfel încât încercarea de a păstra această viteză superioară va fi, inevitabil, însoţită de posibilitatea de a pierde una sau mai multe apariţii ale şablonului.

Pentru a trece peste această dificultate pot fi avute în vedere următoarele soluţii:

1. Extinderea uniformă a textului fiecărui proces cu câte $\left\lceil \frac{m}{2} \right\rceil$ simboluri, unde m este lungimea şablonului, în ambele capete. În felul acesta, textele pasate către două procese vecine se vor putea suprapune pe o lungime de cel mult m simboluri.
2. Realizarea de căutări parţiale (ipoteze) la capetele şirului, urmând ca acestea să fie confirmate în momentul în care căutarea în zona vecină este încheiată.

Alegerea metodei de căutare poate fi realizată în funcţie de modul în care are loc transferul de date între procesul principal şi cele n procese desemnate pentru rezolvarea subproblemelor.

În situaţia în care procesele au acces la o memorie comună prin intermediul căreia pot accesa textul în care se realizează căutarea, o alegere potrivită pentru rezolvarea fiecăreia dintre subproblemele de căutare este unul dintre algoritmii cu viteză de căutare mare, cum ar fi Boyer-Moore, Colussi, Turbo BM.

Tibiscus

Dacă, însă, este necesară transmiterea integrală a datelor către procesele participante, procesul central va putea realiza o estimare a frecvențelor caracterelor în timpul construirii pachetelor de date care urmează să fie transmise.

În acest mod ar putea fi ales un algoritm în care comparația caracterelor din şablon cu caracterele din fereastră este realizată într-o ordine bine precizată, alta decât de la stânga la dreapta sau de la dreapta la stânga.

2.2 Transformarea condițiilor problemei

Abordarea avută în vedere în acest articol oferă un punct de vedere uşor diferit. Paralelismul este introdus în acest caz prin transformarea uniformă atât a spațiului de lucru (textului) cât şi a şablonului utilizat. În această situație problema de căutare este redusă la mai multe subprobleme de căutare diferite, folosind şabloane şi texte modificate prin aplicarea unei proiecții.

Identificarea aparițiilor unui şablon într-un text este realizată prin combinarea rezultatelor întoarse de fiecare dintre subprobleme, această operație fiind realizată îndată ce este semnalată o posibilă potrivire a şablonului în text. Totodată, în acest mod este posibilă accelerarea vitezei de căutare, diferitele procese participante la proces având astfel capabilitatea de a "sări" peste bucăți mari de text, dacă este posibil.

Transformarea textului şi a şablonului este realizată într-o manieră extrem de simplă: fie V alfabetul folosit pentru scrierea textului, $V_1 \subset V$ o submulțime a lui V iar $\bar{b}$ un simbol (numit blank) care nu se găseşte în alfabetul V. Definim aplicația $pr_{V_1} : V \to V_1 \cup \{\bar{b}\}$ prin

$$pr_{v_1}(a) = \begin{cases} a, a \in V_1 \\ \bar{b}, a \notin V_1 \end{cases}$$

Pentru început, descompunem alfabetul V utilizat pentru descrierea textului într-un set de k alfabete disjuncte, formate, de preferat, din simboluri cu frecvențe cumulate de apariție apropiate. În plus, toate simbolurile utilizate în aceste mulțimi trebuie să apară în şablonul principal. Această descompunere limitează numărul de alfabete disjuncte (şi, implicit, numărul de procese participante la rezolvarea problemei) la numărul total de simboluri utilizate pentru scrierea şablonului.

Fiecare dintre procesele P_i cu $i = 1,2,\ldots,k$, participante la algoritmul de căutare recepționează proiecțiile peste alfabetul V_i ale textului şi ale şablonului utilizat. Mai departe, activitatea fiecărui proces P_i presupune căutarea proiecției şablonului original în proiecția textului original, fiecare apariție fiind numită ipoteză. Fiecare proces se orientează aşadar spre emiterea de ipoteze şi verificarea ipotezelor emise de alte procese.

2.2.1 Caracteristicile metodei

Subliniem, în cele ce urmează, câteva dintre caracteristicile care pot fi deduse din această metodă şi din algoritmul (vezi Secțiunea 2.2.2) utilizat pentru funcționarea fiecăruia dintre procesele P_i cu $i = 1,2,\ldots,k$.

Viteza de căutare

Procesele care realizează o căutare rapidă într-o porțiune a textului propriu pot influența accelerarea funcționării celorlalte procese în mai multe moduri:

- după emiterea unei ipoteze de către un astfel de proces, celelalte procese sunt "obligate" să avanseze la poziția ipotezei pentru verificarea acesteia;
- la verificarea unei ipoteze emisă de către un alt proces, P_l, acesta va putea avansa la poziția curentă a unui proces mai rapid.

Prin urmare procesele mai lente pe anumite porțiuni sunt obligate să realizeze salturi în funcționare pentru a ajunge la poziția proceselor mai rapide.

Transferul eficient al datelor

Fiecare proces P_i cu $i = 1,2,\ldots,k$, participant la algoritmul de căutare îşi desfăşoară activitatea în condiții deosebite: majoritatea simbolurilor din textul şi din şablonul utilizat (proiecția pr_{V_i} a textului şi a şablonului original) sunt simboluri blank, $\overline{b}$. Textul poate fi privit ca o matrice rară, cu numeroase goluri, posibil continue, marcate prin simboluri blank. Având în vedere această calitate a datelor, poate fi implementată o metodă pentru

Tibiscus

optimizarea modului în care urmează să fie transferate datele problemei către procese.

Adaptabilitate

Pregătirea fiecărei proiecții se realizează prin intermediul unui proces specializat. Prin intermediul acestui proces este posibilă culegerea informațiilor legate de frecvențele de apariție ale caracterelor. În acest mod, în situația în care datele nu sunt transmise către procese într-un singur transfer, este posibilă modificarea celor k mulțimi şi a proiecțiilor asociate astfel încât distribuția simbolurilor din cele k mulțimi să fie cât mai uniformă posibil.

Funcționare sincronă vs. funcționare asincronă

Implicit, algoritmul presupune un comportament sincron al proceselor P_i cu $i = 1,2,\ldots,k$. Totuşi, în ipoteza că este folosit un model client-server pentru a suporta verificarea ipotezelor, este posibilă bufferizarea ipotezelor emise de cele k procese, lăsând problema verificării ipotezelor şi a accelerării vitezei proceselor participante în seama unui proces central (serverul, în această ipoteză). În acest caz comportamentul celor k procese, P_i cu $i = 1,2,\ldots,k$, este asincron.

2.2.2 Descrierea algoritmului

Pentru descrierea algoritmului, vom presupune că fiecare dintre cele k procese participante, P_i cu $i = 1,2,\ldots,k$, utilizează o metodă M_i pentru identificarea următoarei apariții a şablonului în text.

Comunicarea între două procese P_a şi P_b presupune transmiterea următoarelor tipuri de mesaje:

- verificare ipoteză: este transmisă poziția ipotezei;
- cofirmare ipoteză: este returnată poziția ipotezei;
- infirmare ipoteză: este returnată cea mai avansată poziție dintre poziția curentă şi poziția următoare poziției ipotezei.

Fiecare dintre cele k procese participante, P_i cu $i = 1,2,\ldots,k$ îşi desfăşoară activitatea pe textul şi pe şablonul propriu în felul următor:

Libiscus

1) Procesul P_i utilizează metoda M_i pentru a identifica următoarea apariție a proiecției proprii a şablonului în proiecția textului;
2) Dacă este identificată o apariție a şablonului în text, atunci
   a) Dacă procesul P_i este în timpul verificării unei ipoteze şi poziția ipotezei verificate este identică cu poziția ipotezei obținute, emite mesajul de confirmare ipoteză;
   b) În caz contrar,
      i) emite câte o cerere de verificare a ipotezei către fiecare proces P_j, cu $j = 1,2,\ldots,k, j \neq i$.
      ii) Aşteaptă confirmarea ipotezei de la fiecare proces P_j, cu $j = 1,2,\ldots,k, j \neq i$.
      iii) Dacă ipoteza este confirmată, comunică apariția procesului central, apoi trece la Pasul 1.
      iv) Dacă ipoteza este infirmată de un proces P_j, trece la poziția returnată în mesajul de infirmare, apoi trece la Pasul 1.

3) Dacă este recepționată o cerere de verificare a unei ipoteze din partea unui proces P_j, atunci:
   a) Dacă poziția curentă depăşeşte poziția ipotezei, întoarce poziția curentă către procesul P_j în mesajul de infirmare a ipotezei.
   b) Dacă poziția ipotezei depăşeşte poziția curentă, avansează pe poziția ipotezei, apoi treci la Pasul 1.
4) Dacă este epuizat spațiul de lucru al procesului curent, emite o cerere fictivă de verificare a unei ipoteze pentru poziția finală, pentru a determina terminarea tuturor celorlalte procese. Încheie execuția procesului P_i.

Bibliografie

[AG92] **Apostolico, A., Giancarlo, R.**, *The Boyer-Moore-Galil string searching strategies revisited*, SIAM Journal on Computing, **15** (1), 98--105, 1992.

fibrscus

[Aho90] **Aho, A.V.**, *Algorithms for Finding Patterns in Strings*, Handbook of Theoretical Computer Science, Volume A, Algorithms and complexity, J. van Leeuwen ed., Cap. 5, pp **255 - 300**, Elsevier, Amsterdam, 1990.

[BM77] **Boyer, R.S., Moore, J.S.**, *A fast string searching algorithm*, Communications of the ACM, **20** (1977), 762-772.

[CR94] **Crochemore, M., Rytter, W.**, *Text Algorithms*, Oxford University Press, 1994.

[KMP77] **Knuth, D.E., Morris, Jr, J.H., Pratt, V.R.**, *Fast pattern matching in strings*, SIAM Journal on Computing, **6** (1), 323---350, 1977.

[Ste94] **Stephen, G.A.**, *String Searching Algorithms*, World Scientific, 1994.

Tibiscus