Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

ALGORITMI CARE FOLOSESC TABELE DE DISPERSIE

Compresia Lempel-Ziv-Welch

- Algoritmul de compresie LZW (Lempel-Ziv-Welch), în diferite variante, este cea mai folosită metodă de compresie a datelor, deoarece nu necesită informații prealabile despre datele comprimate (este o metodă adaptivă) și este cu atât mai eficace cu cât fișierul inițial este mai mare și conține mai multă redundanță

Compresia LZW

- Pentru texte, rezultatele sunt foarte bune, deoarece acestea folosesc în mod repetat anumite cuvinte uzuale, care vor fi înlocuite printr-un cod asociat fiecărui cuvânt
- Compresia LZW este folosită de multe programe (gzip, unzip), precum și în formatul GIF de reprezentare (compactă) a unor imagini grafice

Compresia LZW

- Se folosește o tabelă de dispersie, prin care se asociază unor șiruri de caractere de diferite lungimi coduri numerice întregi și se înlocuiesc secvențe de caractere din fișierul inițial prin aceste numere
- Această tabelă de dispersie este analizată la fiecare nou caracter extras din fișierul inițial și este extinsă de fiecare dată când se găsește o secvență de caractere care nu exista anterior în tabela de dispersie

Decompresia LZW

- Pentru decompresie se reface tabela de dispersie construită în etapa de compresie
- Tabela de dispersie nu trebuie transmisă împreună cu fișierul comprimat
- Dimensiunea uzuală a tabelei de dispersie este $4096 = 2^{12}$, dintre care primele $256 = 2^8$ poziții conțin toate caracterele individuale care pot apărea în fișierele de comprimat

Compresia LZW

- Șirurile de caractere se reprezintă prin numere, iar codurile asociate pot avea lungimi diferite
- Se poate folosi o tabelă de dispersie (un dicționar) formată dintr-un singur vector de șiruri (pointeri la șiruri), iar codul asociat unui șir este chiar poziția în vector unde este memorat șirul

Compresia LZW

- Șirul inițial (de comprimat) este analizat și codificat într-o singură trecere, fără revenire
- La stânga poziției curente sunt subșiruri deja codificate, iar la dreapta poziției curente se caută cea mai lungă secvență care există deja în tabela de dispersie

Compresia LZW

- Când este găsită această secvență, ea este înlocuită prin codul asociat deja și se adaugă la tabela de dispersie o secvență cu un caracter mai lungă

Exemplu

- Se presupune că textul de codificat conține numai două caractere (‘a’ și ‘b’) (sub text sunt trecute codurile asociate secvențelor respective):
- a b b a a b b a a b a b b a a a a b a a b b a
- 0 | 1| 1| 0 | 2 | 4 | 2 | 6 | 5 | 5 | 7 | 3 | 0

Tabela de dispersie / Dicționar

- Tabela de dispersie va conține:
- 0=a / 1=b / 2=0b (ab) / 3=1b (bb) / 4=1a (ba) /
- 5=0a (aa) / 6=2b (abb) / 7=4a (baa)
- 8=2a (aba) / 9=6a (abba) / 10=5a (aaa) /
- 11=5b (aab) / 12=7b (baab) / 13=3a (bba)

Compresia LZW

Inițializare dicționar cu toate caracterele din șir și codurile asociate

P = { }; C = caracterul curent din șir

cât timp C nu este ultimul caracter ***repetă***

    citește un caracter C

    dacă P+C există în dicționar ***atunci*** P=P+C

    altfel /* P+C nu este în dicționar */

        determină codul lui P din dicționar și depune-l în șirul de ieșire

        adaugă P+C în dicționar

        P=C

    C trece la caracterul următor

Depune în șirul de ieșire codul asociat lui P

Șirul de ieșire este codificarea dorită

Intrare: w a b b a w a b b a

1	a		
2	b		
3	w		
P = {} C = w P+C = w			
P = w C = a P + C = wa		adaug 4 wa	output 3
P = a C = b P+C = ab		adaug 5 ab	output 1
P = b C = b P+C = bb		adaug 6 bb	output 2
P = b C = a P+C = ba		adaug 7 ba	output 2
P = a C = w P+C = aw		adaug 8 aw	output 1
P = w C = a P+C = wa			
P = wa C = b P+C = wab		adaug 9 wab	output 4
P = b C = b P+C = bb			
P = bb C = a P+C = bba		adaug 10 bba	output 6
P = a		output 1	

Ieșire: 3 1 2 2 1 4 6 1

Observații

- Rezultatul codificării este un șir de coduri numerice, cu mai puține elemente decât caractere în șirul inițial, dar câștigul obținut depinde de mărimea acestor coduri
- Dacă toate codurile au aceeași lungime (12 biți pentru 4096 de coduri diferite), atunci pentru un număr mic de caractere în șirul inițial nu se obține nicio compresie (poate chiar un șir mai lung de biți)
- Compresia efectivă începe numai după ce s-au prelucrat câteva zeci de caractere din șirul analizat

Decompresia LZW

- La decompresie se analizează un șir de coduri numerice, care pot reprezenta caractere individuale sau secvențe de caractere
- Cu ajutorul dicționarului se decodifică fiecare cod întâlnit

Algoritmul Rabin-Karp

- Se consideră **T** un șir de **n** caractere și **P** un pattern de **m** caractere
- Algoritmul verifică dacă **P** apare sau nu ca subsecvență în șirul **T**

Algoritmul Rabin-Karp

- Michael Rabin și Richard Karp au conceput un algoritm de **pattern-matching**, bazat pe tabele de dispersie (**hashing**)
- Dacă două șiruri au aceeași valoare hash, atunci ar putea să coincidă
- În caz contrar, sigur sunt diferite

Algoritmul Rabin-Karp

- Răspunsul **Nu** al algoritmului este întotdeauna corect
- Răspunsul **Da** al algoritmului este corect cu o probabilitate mare
- Algoritmul preprocesează patternul în **O(m)**
- Căutarea se execută în cazul cel mai defavorabil în **O((n-m+1)m)**, dar, în medie, timpul de execuție este mai bun

Algoritmul Rabin-Karp

- Se notează cu `d` numărul de caractere distincte care apar în text
- Un șir de lungime `m` poate fi considerat un număr în baza `d`, având `m` cifre
- Se notează cu `p` valoarea în baza `10` asociată patternului `P`

Algoritmul Rabin-Karp

- Se notează cu t_k valoarea în baza 10 a subsecvenței din T care începe la poziția k, adică $T[k...k+m-1]$
- $p=t_k$ dacă și numai dacă $P=T[k...k+m-1]$

Algoritmul Rabin-Karp

- Valorile t_k pentru $k=0,...,n-m+1$ se pot calcula în $O(n-m+1)$, deoarece t_{k+1} se poate determina din t_k cu formula:
- $t_{k+1}=d*(t_k-d^{m-1}*T[k])+T[k+m]$
- Dificultatea constă în faptul că p și t_k pot fi numere foarte mari
- Din această cauză, operațiile cu ele nu se execută în timp constant

Algoritmul Rabin-Karp

- Pentru a rezolva această problemă, se lucrează cu `p` și t_k modulo `q`, unde `q` este un număr prim convenabil ales, astfel încât `d*q<MAXLONGINT`, deci se utilizează o funcție hash
- Problema care apare este că $p\%q==t_k\%q$ nu implică faptul că $p==t_k$, adică apar coliziuni

Algoritmul Rabin-Karp

- Dacă $p\%q!=t_k\%q$ atunci sigur $P!=T[k...k+m-1]$
- Se poate utiliza testul modulo q ca o euristică, urmând să testăm egalitatea dintre P și $T[k...k+m]$

Exemplu

- Se consideră
- P="31415" și
- T="2359023141526739921"
- În acest caz d=10 (sunt doar cele 10 cifre), q=13, m=|P|=5 (P are 5 caractere) și n=|T|=19 (T are 19 caractere)
- Valoarea p a lui P este p=31415%13=7

Valorile funcției hashing pentru subsecvențele de lungime 5 ale textului

t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
8	9	3	11	0	1	7	8	4	5	10	11	7	9	11

- Se observă că $p=t_6$ și $p=t_{12}$
- Prima este o potrivire corectă, în schimb a doua este o potrivire falsă

Algoritmul Rabin-Karp

```
Rabin-Karp (T, P, d, q) {
n=strlen(T); m=strlen(P);
h=d^(m-1)%q;
p=0; t0=0;
for (i=0; i<m; i++) {                    //preprocesare pattern
        p=(d*p+P[i])%q;                  //calcul p inițial
        t0=(d*t0+T[i])%q; }              //calcul t0 inițial
for (k=0; k<=n-m; k++) {
        if (p==t0)    //testare toate deplasamentele posibile
                if (P==T[k…k+m-1]) {   //găsit la poziția k
                        printf("Da"); return; }
        t0=(t0+d*q-T[k]*h)%q;                //recalculare t0
        t0=(t0*d+T[k+m])%q; }
printf("Nu"); return; }
```