Descrierea soluției - subsecvente

Student Marius Laurențiu Stroe - Universitatea București

Notăm prin $|S| = |S_1| + \ldots + S_{|M|}$, prin S' șirul de lungime minimă din M iar prin L lungimea maximă posibilă a unei subsecvențe.

Soluție 30p

Soluția naivă constă în a considera toate subsecvențele celui mai scurt șir S', în $O(L*|S'|)$, și a verifica dacă există în celelalte, în $O(|S|*L)$. Complexitatea finală: $O(L^2*|S|*|S'|)$. Cazul defavorabil se obține când există două șiruri de lungimi aproximativ egale, caz în care complexitatea finală este $O(L^2*|S|^2)$. Însă, în practică, dacă această abordare este optimizată efectuând cât mai puține comparații atunci se pot obține punctaje mai mari.

Căutarea unui subșir în celelalte șiruri poate profita de restricția că lungimea maximă a unei subsecvențe nu poate depăși 60. Înlocuind caracterele a și b cu 0 și 1, vom obține șiruri ale căror subsecvențe pot fi reprezentate ca numere întregi pe cel mult 64 de biți. Astfel, putem considera toate lungimile unei subsecvențe, în complexitate $O(L)$, considera apoi toate subsecvențele de această lungime din cel mai scurt șir, în complexitate $O(|S'|)$, și căuta în restul șirurilor această subsecvență, în complexitate $O(|S|+L)$. O subsecvență, reprezentată de un număr întreg, se poate căuta într-un șir considerând toate subsecvențele acestuia consecutiv, deplasând biții săi la stânga și înlăturându-i pe cei mai semnificativi, operații ce se pot efectua în $O(1)$. Astfel, complexitatea totală este $O(L*|S|*|S'|)$, care în cazul devarobabil, menționat anterior, devine $O(L*|S|^2)$. În practică, căutarea curentă se comportă puțin mai slab decât căutarea anterioară datorită operațiilor pe numere de 64 biți.

Soluție 40p

O altă restricție a problemei ce ne poate ajuta este cardinalul mulțimii M ce poate fi maxim 4. În cazul în care $|M| = 2$, problema se reduce la a determina cea mai lungă subsecvență comună a două șiruri. O soluție a acestei probleme se poate obține în complexitate $O(|S|^2)$ cu metoda programării dinamice. Recurența poate fi următoarea:

```
d[i][j] = {
      d[i - 1][j - 1] + 1, dacă S1[i] = S2[j]
      0, altfel
}
```

Soluția este $\max(d[i][j],$ iterând după toți indicii $i, j)$. Analog, se mai poate adăuga o dimensiune dacă avem trei șiruri. În cazul general, complexitatea este $O(|S|^{|M|})$. Implementarea recurenței cu număr variabil de șiruri vă va aduce acest punctaj.

Soluție 55p

Observăm că dacă o subsecvență `S'` se găsește în toate șirurile, atunci orice subsecvență a sa, se găsește de asemenea în toate șirurile. În particular, orice prefix al lui `S'` se găsește în toate șirurile. Se poate astfel căuta binar lungimea unei subsecvențe. Iar dacă extindem prima soluție înlocuind căutarea liniară, obținem o soluție cu complexitatea finală în cazul defavorabil: $O(\log(L)*|S|^2)$.

Soluție 55p, Adrian Panaete

Putem folosi o metodă constructivă începând cu subsecvența vidă corespunzătoare lungimii `0` și extinzând orice subsecvență comună în cele două moduri posibile (adăugând la final caracterul `'a'` sau caracterul `'b'`). Dacă prin extindere se obține subsecvență comună, atunci se realizează o îmbunătățire a soluției curente și se păstrează subsecvența extinsă. Algoritmul se incheie dacă pentru subsecvențele de o anumită lungime `L` nu se mai poate aplica nicio extindere și `L` va fi exact soluția problemei. Deoarece la fiecare pas corespunzător subsecvențelor de lungime `0`, `1`, …, `L-1` vom avea de procesat numai subsecvențe comune, atunci numărul acestora nu va putea depăși `|S'|`. În realitate acest număr va fi mai mic deoarece subsecvențele identice ale celui mai scurt string vor fi luate o singură dată. Complexitatea supraestimată a acestui algortim este $O(L*|S|*|S'|)$.

Soluție 80p

Ținând cont că pentru `80%` din teste o complexitate de memorie $O(L*|S|)$ este acceptabilă, putem menține într-o structură de date toate subsecvențele lui `S'` de lungime cel mult `L`. Subsecvențele trebuie ținute pe numere întregi de `64` de biți pentru a nu consuma memorie în exces și a compara două subsecvențe în timp constant. Astfel, iterând celelalte șiruri, vom calcula din nou toate subsecvențele, dar vom trece mai departe doar cele care se găsesc în șirul anterior. Astfel, setul de subsecvențe descrește semnificativ. Condiția pentru a obține acest punctaj este ca structura de date ce menține subsecvențele să permită operații de inserare și ștergere în timp logaritmic. Atunci complexitatea finală este $O(L*|S| \log (L*|S|))$.

Soluție 100p, Adrian Panaete

Se concateneaza cele n stringuri obținând astfel un string de lungime `|S|`. Pentru fiecare poziție din șirul concatenat memorăm un cod binar de cel mult `4` biți care ne indică din ce string inițial face parte poziția respectivă (pentru primul sir codul `1=0001` pentru al doilea codul `2=0010` pentru al treilea codul `4=0100` si pentru al patrulea codul `8=1000`) Vom începe să sortăm pozițiile utilizând drept criteriu ordinea lexicografică a subsecvențelor ce încep din poziția respectivă. Pentru sortare vom folosi idea algoritmului radix-sort grupând pozițiile în buchete care conțin toate pozițiile cu o aceeași subsecvență de o anumită lungime și descompunând

fiecare buchet în cate două buchete corespunzătoare prelungirii subsecvenței respective fie cu caracterul `'a'` fie cu caracterul `'b'`. Odată cu introducerea unei poziții în unul dintre buchete adaugam la un cod binar al buchetului și codul poziției din care provine subsecvența corespunzătoare. În acest fel putem urmări dacă un buchet mai merită sa fie descompus (în cazul în care codul buchetului are la final toți cei `n` biți deci conține subsecvențe din toate cele `n` stringuri inițiale). Tot codurile binare ale pozițiilor ne semnalizeaza daca la un moment dat încercăm să prelungim o subsecvență cu un caracter care nu mai aparține stringului. În acel caz poziția respectivă nu va mai fi plasată în niciunul dintre cele două buchete. Cum fiecare buchet descrie de fapt o subsecvență de o anumită lungime odată cu definitivarea unui buchet dacă se constată că el conține poziții din toate stringurile se poate folosi lungimea respective pentru a îmbunătăți eventual soluția problemei. Deoarece fiecare poziție poate fi utilizată de cel mult L ori (L=soluția testului) vom obține un algoritm de complexitate O(L*|S|).

Soluție 100p, Adrian Panaete

Construim un arbore binar, în care fiul stâng este `'a'` iar fiul dreapt este `'b'`. Considerăm toate subsecvențele de lungime `L` și le inserăm în acest arbore binar. Inserarea se face pornind din rădăcină și mergând spre stânga dacă primul caracter al subsecvenței este `'a'` sau în dreapta dacă acesta este `'b'`. Recursiv se procedează pentru restul nodurilor din arbore și restul subsecvenței. Complexitatea acestei operații este `O(L*|S|)`.

În acest arbore binar, fiecare nod memorează câte o subsecvență de lungime egală cu distanța de la nod la rădăcină. Pentru fiecare nod din arbore memorăm pe un număr de cel mult `|M|` biți din ce string face parte subsecvența corespunzătoare. Luăm ulterior celelalte șiruri și parcurgem arborele cu toate subsecvențele sale de lungime cel mult `60`. Însă, de data aceasta, avem două situații de întrerupere a parcurgerii:

1. parcurgerea se întrerupe deoarece un nod nu are continuare pe o literă; acest lucru înseamnă de fapt că subsecvența corespunzătoare nu e în primul șir, deci implicit, orice continuare a ei nu va fi în primul șir;
2. parcurgerea se întrerupe pentru că informația din nod îmi dovedește că un alt string anterior, altul decât primul șir, nu conține subsecvența corespunzătoare și, implicit, nici continuările ei.

Dacă am orice altceva însemna că de fapt subsecvența nu a fost marcată într-un șir anterior. În cazul preferabil, când informația mă asigură că sunt pe o cale bună, actualizez informația din nod adăugând un bit corespunzător șirului curent.

Soluția se actualizează în timp real odată cu introducerea subsecvențelor noului șir în arborele binar în funcție de cât de adânc reușesc acestea să pătrundă fără ca eventual să apară o întrerupere de cele două tipuri. Soluția de lungime maximă a ultimului șir este răspunsul corect. Complexitatea finală este `O(L*|S|)`.