



Sablon - Descrierea soluției

Autor prof. Radu Vișinescu, C.N. "I.L. Caragiale" Ploiești

Soluție 32 puncte (prof. Radu Vișinescu):

Se programează o funcție recursivă care decide dacă un șir de caractere S se potrivește cu o expresie șablon E. Cazurile se separă în funcție de primul caracter al expresiei E respectiv al șirului S. Exista și un caz în care se execută mai multe apeluri recursive și anume la întâlnirea caracterului * în expresia șablon E. Funcția se apelează din programul principal pentru fiecare valoare din șirul de cuvinte.

Soluție 48 puncte (prof. Daniel Popa):

Se transformă șablonul într-o expresie regulată care este folosită pentru căutare folosind biblioteca regex. Avantaj: Cod scurt. Dezavantaj: funcționează doar sub Linux, se consumă multă memorie, algoritmul implementat în librărie nu este foarte performant.

Soluție 64 puncte (student- Mihai Nițu):

Pentru a verifica dacă un cuvânt se potrivește cu un șablon dat, se poate utiliza un algoritm greedy sau o relație de recurență. Algoritmul greedy constă în verificarea dacă capetele șablonului (prefixul și sufixul până la '*') coincide cu capetele cuvântului, și apoi căutarea pe rând a fiecărei subsecvențe din șablon cuprinsă între '*' în cuvânt. Dacă acestea apar ca subsecvențe în cuvânt, în ordinea în care apar în și în șablon și nu se intersectează cu bucatile de care am stabilit că vor corespunde capetelor șablonului, atunci cuvântul se potrivește cu șablonul.

Soluția alternativă este data următoarea relație de recurență

$Match[i][j] = 1$ dacă sufixul $[j..m]$ din cuvântul c se potrivește cu sufixul $[i..n]$ din șablonul s.

$$Match[n+1][m+1] = 1$$

$$Match[i][j] = Match[i+1][j+1] \text{ dacă } s[i] == '?' \text{ sau } s[i] - \text{literă și } s[i] == c[j] \\ = 0, \text{ dacă } s[i] - \text{literă și } s[i] != c[j]$$

$= Match[i+1][j] \parallel Match[i+1][j+1] \parallel \dots Match[i+1][m+1]$, dacă $s[i] == '*'$, unde $\parallel$ este 'sau' logic. $\rightarrow$ se pot ține 'sau'-uri parțiale pe fiecare linie a matricei Match pentru a obține valoarea în $O(1)$.

Pentru a ține cont de posibilitatea eliminării unei litere, putem să aplicăm algoritmul, pentru șablonurile obținute prin eliminarea fiecărei litere. Complexitatea finală va fi $O(nr. \text{Cuvinte} * N^3)$ cu ambele abordări. Pentru algoritmul greedy, nu putem folosi KMP din cauza caracterelor ?, deci vom folosi algoritmul de potrivire brut. Aceasta soluție poate fi dificilă de implementat dar poate obține chiar 100 puncte deoarece algoritmul brut de potrivire se comportă foarte bine în practică.



Soluție 100 puncte (student - Mihai Nițu):

Putem sa imbunatatim, spre exemplu, solutia prin recurenta, pentru a numai fi nevoie să o recalculam de fiecare data cand eliminam o litera. Astfel, vom calcula aceeași recurenta de 2 ori, Match1 si Match2. Diferența dintre ele este ca Match1 va verifica potrivirea între un sufix al sablonului si un sufix al cuvântului iar Match2 va verifica potrivirea între un prefix al sablonului si un prefix al cuvântului. Putem, apoi, sa verificam pentru fiecare pozitie din sabloni, pe care exista o litera, daca $ok2[i-1][j] = 1$ si $ok1[i+1][j+1] = 1$, pentru oricare j de la 0 la m. Daca cel puțin una se evaluează la 1, atunci avem o potrivire. Complexitate: $O(nr. Cuvinte * N^2)$.