Algoritmi de procesare a şirurilor de caractere. Trie şi distanţa Levenshtein

Grigore Iulia-Anita

Facultatea de Automatică şi Calculatoare
iulia_anita.grigore@stud.acs.upb.ro

1 Descrierea problemei

Într-o lume extrem de digitalizată ca cea în care trăim acum, şirurile de caractere sunt peste tot: mesaje trimise online, căutări efectuate cu ajutorul diverselor browsere de internet şi chiar în propriul organism (secvenţele de ADN sunt uneori tratate ca o serie de simboluri cu semnificaţii bine cunoscute).

Din această cauză a apărut necesitatea implementării unor algoritmi din ce în ce mai eficienţi de procesare a şirurilor de caractere.

Câteva necesităţi din viaţa reală sunt, spre exemplu, programele de spell checking implementate de majoritatea editoarelor de text, sau chiar sugestiile primite în momentul în care cauţi ceva pe internet.

Un exemplu care nu este atât de la îndemână, dar care are o influenţă majoră în dezvoltarea ştiinţei, este alinierea secvenţială, un termen întâlnit în bioinformatică. Aceasta constă în aranjarea unor secvenţe de ADN şi de ARN pentru a putea observa dacă au un predecesor comun (un şir de date comun) şi pentru a depista mutaţiile (de deleţie sau de inserţie a unor gene) apărute la nivelul acestora. [2]

2 Soluţii alese

Ca soluţii la problema prezentată anterior, am ales structura de date Trie şi un algoritm de edit distance, distanţa Levenshtein. Pe parcursul lucrării, voi compara, în linii mari, mai multe distanţe (precum Hamming distance), însă doar din punct de vedere al eficientizării sau în privinţa completărilor aduse soluţiei deja alese.

2.1 Trie

Trie este o structură de date implementată sub forma unui arbore, în care root-ul este gol, iar descendenţii săi sunt reprezentaţi de un alfabet. Pentru a stoca un şir de caractere, este necesară parcurgerea arboreului în adâncime, pe copilul corespondent caracterului căutat.

Datorită acestei implementări, se foloseşte mai putină memorie (în majoritatea cazurilor), nefiind necesar să salvăm fiecare şir individual, ci doar să adăugăm la şirurile existente caracterele lipsă. [3]

O utilizare des întâlnită a trie-ului este oferirea unor sugestii de căutare pe baza unor caractere scrise anterior. De exemplu, daca vom căuta pe Google „tree“, vom primi sugestii precum „tree of life“ sau „treehouse“, toate acestea având un şir în comun.

2.2 Distanţa Levenshtein

Distanţa Levenshtein măsoară *diferenţa* dintre doua şiruri de caractere. Această diferenţă este calculată conform unei formule matematice, iar operaţiile care le calculează sunt ştergerea, adaugarea şi înlocuirea unui caracter. [1]

Formula de calcul a distanţei Levenshtein este:

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j), & \text{if } \min(i,j) = 0 \\ \min \begin{cases} \mathrm{lev}a,b(i-1,j)+1 \\ \mathrm{lev}a,b(i,j-1)+1 \\ \mathrm{lev}a,b(i-1,j-1)+1 \end{cases} & \text{otherwise} \end{cases} \tag{1}$$

3 Criterii de evaluare a soluţiilor

Pentru evaluarea acestor soluţii, îmi propun să creez 2 tipuri de teste. Pentru verificarea corectitudinii codului, voi folosi unit tests create cu date de intrare variate, care verifică atât cazuri generale, cât şi edge case-uri, iar pentru a testa eficienţa algoritmilor, voi scrie benchmark-uri.

Unit test-urile vor fi scrise astfel încât să verifice funcţionalitatea Trie-ului, fără a ţine cont de performanţa algoritmilor. Cu toate acestea, toţi algoritmii vor fi optimizaţi din punct de vedere al alocarilor dinamice, astfel încât memoria RAM să influenţeze cât mai puţin acurateţea benchmark-urilor.

Pentru verificarea performanţei operaţiilor în Trie, voi construi benchmark-uri bazate pe cuvintele din dictionarul englezei americane. Dimensiunea dicţionarul ce urmează a fi inserat într-o instanţă de Trie este de aproximativ 4.5MB.

Fisierul în format CSV urmează să fie procesat cu un script de Python pentru a obţine şi compara:

- inserţia a aproximativ 460 de mii de cuvinte de lungimi variate ordonate alfabetic, şi
- inserţia aceluiaşi set de cuvinte, dar ordonate aleatoriu.

Ulterior inserţiei voi măsura performanţa căutării cuvintelor din acelaşi dicţionar (intrări ce există în Trie) şi a unor secvenţe generate aleatoriu, astfel încât căutarea să nu întoarcă niciun rezultat.

Voi procesa dicţionarul astfel încât să fie eliminate toate intrările ce contin alte caractere decât `[a-zA-Z]`.

References

1. Black, P.E.: "levenshtein distance", dictionary of algorithms and data structures. U.S. National Institute of Standards and Technology (2016)
2. DM, M.: Bioinformatics: Sequence and genome analysis (2nd ed.). Cold Spring Harbor Laboratory Press: Cold Spring Harbor, NY (2004)
3. Sedgewick, R., Wayne, K.: Digit-based sorting and data structures (????)