**Abordarea mea pentru rezolvarea problemei "Anagram Finder"**

**1. Alegerea abordării principale**

Ideea de bază a fost să găsesc o „formă comună” pentru toate anagramele. Am decis că două cuvinte sunt anagrame dacă, după sortarea literelor lor în ordine alfabetică, devin identice. Versiunea sortată devine forma canonică în cazul meu. De exemplu, pentru „race” și „care”, forma canonică este „acer”.

Pentru a grupa anagramele, am folosit java.util.HashMap. Cheia în această mapă este forma canonică a cuvântului, iar valoarea este o listă cu toate cuvintele originale care corespund acelei forme.

**2. Decizii de design și justificarea lor**

În timpul dezvoltării, m-am concentrat pe câteva aspecte-cheie:

Claritatea codului: Am încercat să scriu cod clar și ușor de înțeles. Logica este împărțită: controlul principal al programului se află în metoda main, iar sarcina de sortare a literelor unui cuvânt este extrasă într-o metodă auxiliară numită sortWord. Acest lucru face ca programul să fie mai ușor de întreținut și înțeles.

Dependențe minime: Am evitat folosirea bibliotecilor externe. Totul este implementat cu clase standard din Java (HashMap, ArrayList, BufferedReader, FileReader). Astfel, programul este mai ușor de rulat și mai puțin predispus la probleme de compatibilitate.

**3. Considerații privind performanța**

Sortarea cuvintelor: Metoda sortWord folosește Arrays.sort(), care are o complexitate temporală de O(LlogL), unde L este lungimea cuvântului. Pentru cuvinte de lungime medie, acest lucru este foarte rapid.

Operații cu HashMap: În medie, inserarea și extragerea valorilor din HashMap au complexitate O(1). În cazuri rare (coliziuni), pot ajunge la O(N), dar acest lucru este rar cu o funcție hash bine implementată.

Complexitatea generală: Dacă fișierul conține M cuvinte și lungimea maximă a unui cuvânt este Lmax, complexitatea este aproximativ O(M⋅Lmax⋅logLmax), ceea ce este eficient pentru majoritatea scenariilor.

Consum de memorie: Programul stochează toate cuvintele și formele lor canonice în memorie. Complexitatea spațială este O(M⋅Lavg), unde Lavg este lungimea medie a cuvântului.

**4. Scalabilitatea soluției**

**4.1. Procesarea a 10 milioane de cuvinte**

Implementarea actuală în Java poate gestiona 10 milioane de cuvinte fără probleme. Limitarea principală este memoria RAM. Cu o lungime medie de 7–10 caractere pe cuvânt, va fi nevoie de câțiva GB de RAM — acceptabil pentru sistemele moderne. Performanța va crește liniar cu volumul de date.

**4.2. Procesarea a 100 de miliarde de cuvinte**

Pentru un volum atât de mare, soluția actuală, care rulează complet în memorie pe un singur server, nu mai este viabilă. Ar fi necesari pași majori de scalare:

Calcul distribuit: Ar fi necesar să se treacă la framework-uri distribuite, precum Apache Spark sau Hadoop. Datele sunt împărțite între mai multe servere, fiecare procesând o parte. Rezultatele sunt apoi combinate pentru a obține toate anagramele.

Stocare externă și distribuită: În loc să încarce toate datele în RAM, cuvintele sunt stocate pe discuri în sisteme de fișiere distribuite sau în baze de date NoSQL scalabile, precum Apache Cassandra sau HBase.

**Rezumat :** Pentru volume uriașe de date, trebuie făcută trecerea de la procesare locală la sisteme distribuite care pot folosi simultan resursele mai multor mașini.