README

Nume: Gogea

Prenume: Mihail

Grupa: 324CC

Tema 1 Proiectarea Algoritmilor

Dificultate: 8.5/10

Timpul alocat acestei teme: 12-14 ore

Observatii:

Arhiva Gogea_Mihail_324CC.zip contine urmatoarele fisiere:

Problema1.java

Prob2.java

Makefile

README

In interiorul codului sursa la cele doua probleme sunt prezente si comenatarii.

Conform vmcheckeru-lui ambele probleme (1 si 2) au punctaj maxim ,iar toate testele se incadreaza in timpul impus in enunt.

Fisierele .java au fost scrise in ecplise, ambele fiind in java.

Detalii implementare:

Problema1: problema1.java

Am construit o functie ce avea ca argument un String(cuvant) numita palindromes . Am parcurs fiecare caracter pana la jumatea cuvantului, iar pentru fiecare caracter comparam cu caracterul opus acestuia adica cel cu indicele i cu cel cu indicele n-i-1 (n este lungimea cuvantului) , daca erau diferite atunci

insemna ca trebuiau facute interschimbari. Algoritmul era urmatoru: comparam cel cu indicele i la care eram cu cel din inaintea indicelui opus acestuia , adica n-i-2 , daca acestea erau diferite verificam cel de opus lui i adica n-i-1 cu cel de dupa caracterul cu indicele i , adica i+ 1 , daca nu erau nici acestea egale atunci comparam din nou caracterul de pe pozitia i cu caracterul de 2 ori in fata lui cel opus adica n-i-3 si tot asa pana gaseam. Daca gaseam la fel atunci cu ajutorul celor 3 variabile String taiam din cuvant si interschimbam pozitile acestuia ca sa poata devenii palindrom , iar numarul de interschimbari intre litere le retineam in variabila count. Daca ajungeam la un moment dat in care indicii i si j erau egali atunci insemna ca am un cuvant in care numarul de caractere cu aparitii impare este mai mare decat 1 adica imposibil de creat un palindrom , urmand sa returnez -1.

Citirea si afisarea au fost facute cu BufferReader si BufferWriter pentru un timp mai bun. Aceasta problema are o complexitate temporala de O(nr*n^2) nr fiind numarul de cuvinte iar n^2 este din cauza celor 2 foruri unul in altul din functia palindromes. Complexitatea spatiala fiind O(logn).

Problema2: prob2.java

La aceasta problema , m-am documentat de pe wikipedia Distante Levenshtein

https://en.wikipedia.org/wiki/Levenshtein_distance acesta este linkul , de distanta dintre 2 cuvinte adica numarul minim de ajustari ca 2 cuvinte sa fie la fel stiam de acest algoritm din anii precedenti , lovinduma de el la unele competitii . La aceasta problema am facut 2 functii separate , prima numinduse distance avand ca argument numele fisierului din care voi citii. Aici am aplicat algoritmul levenshtein cu un cuvant si un grup de cuvinte. Am creeat o matrice de M+ 1 x N+ 1 , in care prima linie si prima coloana erau indicii linilor respectiv indicii coloanelor , urmand apoi sa compar prima litera din cuvantul rostit cu prima litera din grupul cuvintelor variante daca este egala cu una din cele din grupul de variante atunci va lua acea celula valoarea din stanga sus adica fix diagonala ei mai sus in stanga(scad ambii indici) , iar daca este diferita litera din cuvantul rostit cu toate cele din grupul de variante atunci iau minumul dintre 1+ celula de sus , 1+ celula din stanga si 1+ celula stanga sus(cea despre care am vorbit mai devreme) , acest algoritm il continui cu litera la care sunt din cuvantul rostit cu toate literele din cuvintele din grupul de variante. In final dupa ce toata matricea este completata , numarul minim de ajustari este de pe ultima celula(ultimu rand ulima coloana). Cea dea doua functie numita scriere are 2 argumente primul este numele fisierului in care vreau sa scriu iar n este numarul pe care il voi scrie in fisier. Citirea si scrierea le-am facut cu clasele

BufferReader si BufferWriter pentru un timp mai bun. Aceasta problema de programare dinamica are o complexitate temporala de O(M*N*nrVar) si spatiala O(M*N*log(max(M,N))).