

DERSİN ADI: Algoritma Analizi

DERSİN EĞİTMENİ: Dr. Öğr. Üyesi Mehmet Amaç GÜVENSAN

ÖĞRENCİ ADI: Ertuğrul ŞENTÜRK

ÖĞRENCİ NO: 18011028

ÖĞRENCİ MAIL: mdesenturk@gmail.com

DÖNEM: 3

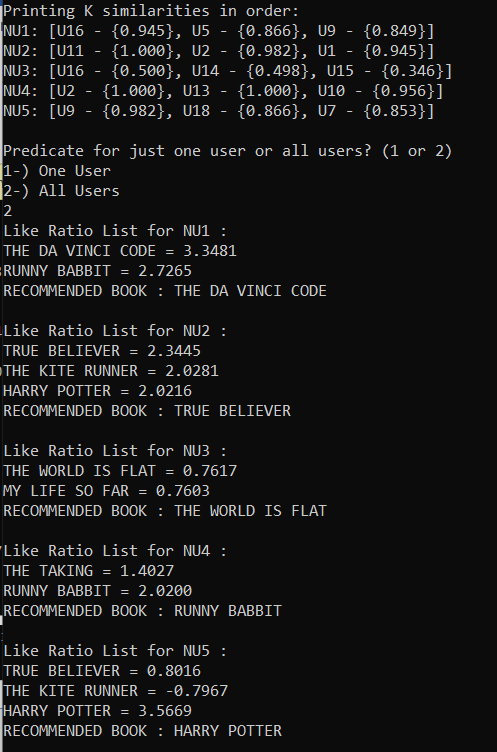
GRUP NO: 2

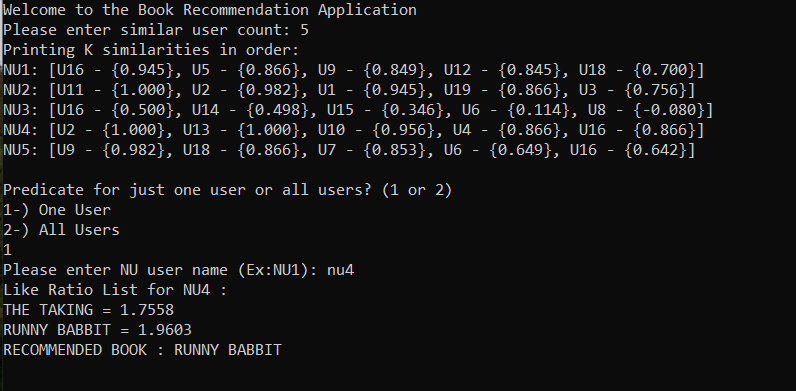
ÖDEV NO: Proje

ÖDEV KONUSU: Kitap Öneri Sistemi

Algoritma:

1. Verilen dosyadaki veriler uygun şekilde okunup dosyaya kaydedildi.
2. Kullanıcıdan hesaplamada kullanılacak benzer kullanıcı sayısı alındı.
3. Verilen inputlar ile her nu üyesinin u üyesi ile pearson kaysayısı hesaplandı.
4. İstenilen benzer kullanıcı sayısı kadar maximum eleman dizinin başına taşındı ve indexleri k\_max adından bir diziye kaydedildi.
5. Bu kullanıcıların benzerlik oranı hakkında bilgi yazdırıldı.
6. Sonrasında kullanıcıdan tek kullanıcıya ait mi yoksa tüm kullanıcıya ait mi öneri yapılacağının bilgisi alındı.
7. Alınan bilgiye göre seçilen nu kullanıcıların okumadığı kitaplar hakkında predicate fonksiyonu çağırılarak tahmin yapıldı.
8. Tahmin yapılan kitapların tahmin değerleri arasından maksimumu kullanıcı için önerilen kitap olacağından o kitabın bilgisi yazdırıldı.

Ekran görüntüleri:



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#include <math.h>

#include <float.h>

// Max available book count

#define MAX\_BOOK\_COUNT 50

// Max available book name

#define MAX\_BOOK\_NAME 200

// Max available user count

#define MAX\_USER\_COUNT 100

// Buffer size for a row

#define ROW\_BUFFER 10

// Database file name

const static char data[] = "RecomendationDataSet.csv";

// Struct definition for keeping user info

// name = user name

// book\_list = book ratings giving by the user

struct{

char name[ROW\_BUFFER];

int\* book\_list;

}typedef user;

// Database reading functions

int read\_book\_names(FILE\* database,char \*\*\* book\_names);

int read\_users(FILE\* database, user users[], int book\_count);

bool read\_database(const char\* file\_name,FILE\*\* database);

bool skip\_line(FILE\* database);

// Pearson calculation

float calculate\_pearson(user u1, user u2, int book\_count);

float\* calculate\_all\_similarities(user nu\_user,user\* u\_users,int u\_count,int book\_count);

// K max element calculation

int\* find\_kmax(int k, float\* values, int size);

// Predication calculation for a book

float calculate\_pred(user nu, int book\_number, user u\_users[],const int\* kmax, int k,int book\_count);

void predicate\_user(user nu\_user,int nu\_order,user u\_users[],int k,int\*\* kmax\_list,char\*\* book\_names,int book\_count);

// Test function

int \*\* test1(int k, user nu\_users[],int nu\_count,user u\_users[],int u\_count,int book\_count);

void test2and3(user nu\_users[], int nu\_count, user u\_users[], int\*\* kmax\_list, int k, char\*\* book\_names,int book\_count);

// Dellocation functions

void free\_book\_names(char\*\* book\_names);

void free\_users(user users[],int u\_count);

void free\_kmax\_list(int\*\* kmax\_list,int nu\_count);

int main(){

// variable definitions

int k;

int book\_count;

int u\_count;

int nu\_count;

char\*\* book\_names;

int\*\* kmax\_list;

user u\_users[MAX\_USER\_COUNT];

user nu\_users[MAX\_USER\_COUNT];

// reading database

FILE\* database;

if (!read\_database(data,&database))

return 0;

// initializing variables

book\_count = read\_book\_names(database,&book\_names);

u\_count = read\_users(database,u\_users,book\_count);

skip\_line(database);

nu\_count = read\_users(database,nu\_users,book\_count);

fclose(database);

printf("Welcome to the Book Recommendation Application\n");

printf("Please enter similar user count: ");

scanf("%d",&k);

if (k>u\_count || k<1){

printf("k value is invalid");

return 0;

}

// Starting test1 and generating kmax\_list

kmax\_list = test1(k,nu\_users,nu\_count,u\_users,u\_count,book\_count);

// Starting test2 and test 3

test2and3(nu\_users, nu\_count, u\_users, kmax\_list, k, book\_names, book\_count);

// memory is freed to recover from memory leak

free\_book\_names(book\_names);

free\_users(u\_users, u\_count);

free\_users(nu\_users, nu\_count);

free\_kmax\_list(kmax\_list,nu\_count);

return 0;

}

/\* Book name reading function

param:

database = file pointer to read database from file

book\_names == array which is keeps book names;

var:

i = iterator;

book\_count = count of books;

buff = buffer to keep a row content;

c = temporary char for reading character

return:

book\_count = count of books;

\*/

int read\_book\_names(FILE\* database,char \*\*\* book\_names){

int i;

int book\_count;

char buff[MAX\_BOOK\_NAME];

char c;

// allocate memory for book list

\*book\_names = (char\*\*) malloc(sizeof(char\*) \* MAX\_BOOK\_COUNT);

for(i = 0; i < MAX\_BOOK\_COUNT; i++){

// allocate memory for each book

(\*book\_names)[i] = (char\*) malloc(sizeof(char) \* MAX\_BOOK\_NAME);

}

// Remove first , for first empty

c = fgetc(database);

i = 0;

book\_count = 0;

while(c != '\n' && c != EOF){

c = fgetc(database);

if(c!=','){

// Add characters to buffer

buff[i] = c;

i++;

}

else{

// Add endl at the end of the buffer for stoping buffer

buff[i] = '\0';

// Copy buffer to array

strncpy((\*book\_names)[book\_count],buff,i+1);

// Increase book count

book\_count++;

// Start buff from begining

i = 0;

}

}

// Add last cell.

buff[i-1] = '\0';

strncpy((\*book\_names)[book\_count], buff, i+1);

// Return book count

return book\_count+1;

}

/\* User reading function

param:

database = file pointer to read database from file

users == array which is users;

book\_count = count of books;

var:

i,j,k = iterator;

buff = buffer to keep a row content;

c = temporary char for reading character

temp = keeps integer value of a row

return:

user count;

\*/

int read\_users(FILE\* database, user users[], int book\_count){

int i, j, k;

char buff[ROW\_BUFFER];

char c;

int temp;

// Read first char

c = fgetc(database);

i = 0;

// If first char is , that means first cell is empty

while(c!=',' && c!=EOF){

// Set values for new user

users[i].book\_list = (int\*) malloc(sizeof(int)\*book\_count);

j = 0;

// Read until end of the line

while (c!='\n' && c != EOF){

k=0;

// Read a cell

while(c != ',' && c != '\n' && c != EOF){

buff[k] = c;

c = fgetc(database);

k++;

}

// For end of the cells

if(c != '\n'){

// Copt last cell

buff[k] = '\0';

// For first cell read username

if(j == 0){

strcpy(users[i].name, buff);

}

// For other cells convert value to integer and add it into book list

else{

temp = atoi(buff);

users[i].book\_list[j-1] = temp;

}

c = fgetc(database);

// Clear buffer for end of the line

if(c == '\n')

buff[0] = '\0';

}

j++;

}

// Add last cell

temp = atoi(buff);

if(temp != 0){

j--;

}

users[i].book\_list[j-1] = temp;

while(c == '\n')

c = fgetc(database);

i++;

}

// Return user count

return i;

}

/\* File reading function

param:

file\_name = file location for reading

database = file pointer to read database from file;

return:

success of reading

\*/

bool read\_database(const char\* file\_name,FILE\*\* database){

\*database = fopen(file\_name,"r");

if(!\*database){

printf("File not found\n");

return false;

}

return true;

}

/\* File reading function

param:

database = file pointer to read database from file;

var:

c = temporary char for reading character

return:

success of reading

\*/

bool skip\_line(FILE\* database){

char c = '.';

while(c != '\n')

c = fgetc(database);

}

/\* Pearson coefficent calculation function

param:

u1 = first user

u2 = second user

book\_count = book count;

var:

i = iterator

numerator = top of the division

denominator1, denominator2 = botton of the division

average1, average2 = average values

common\_book = book count readed by both user

return:

pearson coefficent

\*/

float calculate\_pearson(user u1, user u2, int book\_count){

int i;

float numerator = 0;

float denominator1 = 0;

float denominator2 = 0;

float average1 = 0;

float average2 = 0;

int common\_book = 0;

//Calculate sum value for books both readed

for(i=0;i<book\_count;i++){

// Only calculate if both user readed same book

if(u1.book\_list[i]>0 && u2.book\_list[i]>0){

average1+=u1.book\_list[i];

average2+=u2.book\_list[i];

common\_book++;

}

}

// Calculate average by dividing common book count

average1 /= (float)common\_book;

average2 /= (float)common\_book;

for(i=0;i<book\_count;i++){

// Only calculate if both user readed same book

if(u1.book\_list[i]>0 && u2.book\_list[i]>0){

numerator += ((float)u1.book\_list[i] - average1) \* ((float)u2.book\_list[i] - average2);

denominator1 += pow(((float)u1.book\_list[i] - average1), 2);

denominator2 += pow(((float)u2.book\_list[i] - average2), 2);

}

}

// calculate result

return numerator / (sqrt(denominator1) \* sqrt(denominator2));

}

/\* Pearson coefficent array generator for a nu user

param:

nu\_user = nu user, u\_users = u user list , u\_count = u user count , book\_count = book count;

var:

i = iterator , pearson\_values = list of pearson values for a nu user

return:

list of pearson values for a nu user\*/

float\* calculate\_all\_similarities(user nu\_user,user\* u\_users,int u\_count,int book\_count){

int i;

float\* pearson\_values = (float\*) malloc(sizeof(float)\*u\_count);

for(i=0;i<u\_count;i++){

pearson\_values[i] = calculate\_pearson(nu\_user,u\_users[i],book\_count);

}

return pearson\_values;

}

/\* Calculates first k maxiumum

param:

k = similar user count

values = pearson values for a nu user

size = u user count

var:

i,j = iterator

t,temp = temp values

kmax = array to keeps k max users index;

return:

array to keeps k max users index

\*/

int\* find\_kmax(int k, float\* values, int size){

int i,j;

int t;

float temp;

int\* kmax = (int\*) malloc(sizeof(int)\*size);

for(i=0;i<size;i++){

kmax[i] = i;

}

for(i=0;i<k;i++){

for(j=i+1;j<size;j++){

// swap index array and value array partial bubble sort with k step

if(values[i]<values[j]){

temp = values[i];

values[i] = values[j];

values[j] = temp;

t = kmax[i];

kmax[i] = kmax[j];

kmax[j] = t;

}

}

}

return kmax;

}

/\* Calculation of predication value for a book from first k user

param:

nu = nu user to generate recommendation

book\_number = book order for predication

kmax = k max u user's index

k = similar user count

book\_count = count of all books

var:

i = iterator

numerator = top part of division

denominator = botton part of division

similarity = keeps pearson value between u user and nu user

user u = temp value for keeping each u user

u\_average = average value for u\_users;

nu\_average = average value for nu\_user;

read\_count = temporary value for readed book count for any user;

return:

predication value for selected book and nu user

\*/

float calculate\_pred(user nu, int book\_number, user u\_users[],const int\* kmax, int k,int book\_count){

int i,j;

float numerator = 0;

float denominator = 0;

float similarity;

user u;

float u\_average;

float nu\_average = 0;

int read\_count = 0;

for(i = 0;i<book\_count;i++){

if(nu.book\_list[i]!=0){

read\_count++;

nu\_average += (float)nu.book\_list[i];

}

}

nu\_average /= (float) read\_count;

for(i = 0; i < k; i++){

u = u\_users[kmax[i]];

u\_average = 0;

read\_count = 0;

for(j = 0; j<book\_count; j++){

if(u.book\_list[j]!=0){

u\_average += (float)u.book\_list[j];

read\_count++;

}

}

u\_average /= (float) read\_count;

similarity = calculate\_pearson(nu,u,book\_count);

numerator += similarity\*((float)u.book\_list[book\_number]-u\_average);

denominator += similarity;

}

return nu\_average + (numerator/denominator);

}/\* Makes predication for one nu user

param:

nu\_user = nu user to generate recommendation, nu\_order = order no to user

u\_users = u user array

k = similar user count, kmax\_list = k max index array

book\_names = all book names

book\_count = count of all books

var:

i = iterator

pred = predication value for each unreaded book

denominator = botton part of division

max\_index = keeps index of max predication

max\_value = keeps value of max predication

\*/

void predicate\_user(user nu\_user,int nu\_order,user u\_users[],int k,int\*\* kmax\_list,char\*\* book\_names,int book\_count){

int i;

float pred;

int max\_index = 0;

float max\_value=FLT\_MIN;

printf("Like Ratio List for %s : \n",nu\_user.name);

for(i=0;i<book\_count;i++){

// If NU user not readed that book yet

if(nu\_user.book\_list[i] == 0){

// Calculate each non readed book predication

pred = calculate\_pred(nu\_user,i,u\_users,kmax\_list[nu\_order],k,book\_count);

// Print the values and book names

printf("%s = %0.4f\n",book\_names[i],pred);

// Keep track of maximum

if(max\_value<pred){

max\_value = pred;

max\_index = i;

}

}

}

// Print maximum as a recommended book

printf("RECOMMENDED BOOK : %s\n\n",book\_names[max\_index]);

}

/\* Calculation of predication value for a book from first k user

param:

k = similar user count

nu\_users = nu user array to generate recommendation

nu\_count = nu user count

u\_users = u user array for predication

u\_count = u user count

book\_count = count of all books

var:

i,j = iterator

kmax\_list = k max index array value for selected book and nu user

similarities = array of similartiy values for a nu user;

similaritiy\_list = array of similarties values for each nu user;

return:

kmax\_list = k max index array value for selected book and nu user

\*/

int \*\* test1(int k, user nu\_users[],int nu\_count,user u\_users[],int u\_count,int book\_count){

int i,j;

// Memory allocation for calculating arrays

int \*\* kmax\_list = (int\*\*) malloc(sizeof(int\*)\*nu\_count);

float\* similarities;

float\*\* similaritiy\_list = (float\*\*) malloc(sizeof(float\*)\*nu\_count);

// For each nu user

for(i = 0;i<nu\_count;i++){

// Calculate similarity values for each u user

similarities = calculate\_all\_similarities(nu\_users[i],u\_users,u\_count,book\_count);

// Find k max;

kmax\_list[i] = find\_kmax(k,similarities,u\_count);

// Generate similarity array for printing

similaritiy\_list[i] = similarities;

}

// Print similarity values and deallocate arrays

printf("Printing K similarities in order:\n");

for(i=0;i<nu\_count;i++){

printf("NU%d: [",i+1);

for(j=0;j<k-1;j++){

printf("U%d - {%0.3f}, ",kmax\_list[i][j]+1,similaritiy\_list[i][j]);

}

printf("U%d - {%0.3f}]\n",kmax\_list[i][j]+1,similaritiy\_list[i][j]);

free(similaritiy\_list[i]);

}

free(similaritiy\_list);

printf("\n");

// Return k min list

return kmax\_list;

}/\* Calculation of predication value for a book from first k user

param:

nu\_users = nu user array to generate recommendation

nu\_count = nu user count

u\_users = u user array for predication

kmax\_list = k max index array value for selected book and nu user

k = similar user count, book\_names = names of all books

book\_count = count of all books

var:

i = iterator

choice = print option - input from user

user\_name = user name - input from user

\*/

void test2and3(user nu\_users[], int nu\_count, user u\_users[], int\*\* kmax\_list, int k, char\*\* book\_names,int book\_count){

int i;

char choice;

char user\_name[ROW\_BUFFER];

// Get input for print one selected user's result or all users result

printf("Predicate for just one user or all users? (1 or 2)\n1-) One User\n2-) All Users\n");

scanf(" %c",&choice);

// Print all user

if(choice == '2'){

for(i=0;i<nu\_count;i++){

predicate\_user(nu\_users[i],i,u\_users,k,kmax\_list,book\_names,book\_count);

}

return;

}

else{

// Get user name

printf("Please enter NU user name (Ex:NU1): ");

scanf("%s",user\_name);

// Find user index

i=0;

while(i<nu\_count && strcasecmp(user\_name,nu\_users[i].name) != 0)

i++;

// If user exist call predicate for that user

if(i== nu\_count)

printf("User not found");

else

predicate\_user(nu\_users[i],i,u\_users,k,kmax\_list,book\_names,book\_count);

}

}/\* Memory deallocation for book names

param:

book\_names = array of book names

var:

i = iterator

\*/

void free\_book\_names(char\*\* book\_names){

int i;

for(i=0;i<MAX\_BOOK\_COUNT;i++){

free(book\_names[i]);

}

free(book\_names);

}

/\* Memory deallocation for user array

param: users = user array , u\_count = array size

var: i = iterator

\*/

void free\_users(user users[],int u\_count){

int i;

for(i=0;i<u\_count;i++){

free(users[i].book\_list);

}

}

/\* Memory deallocation for k mix list array

param: kmax\_list = k mix list array , nu\_count = array size

var: i = iterator

\*/

void free\_kmax\_list(int\*\* kmax\_list,int nu\_count){

int i;

for(i=0;i<nu\_count;i++){

free(kmax\_list[i]);

}

free(kmax\_list);

}