ESTRUCTURA DE DADES I ALGORISMES II, 2022-2023

Pràctica final

Enric Martínez Morell (U215087)
Victor Martínez Navalón (U214560)
Javier Ocampo Tafur (U214966)
08/06/2023

ÍNDEX

INTRODUCCIÓ

El nostre grup ha fet una proposta de xarxa social enfocada a emparellar persones/grups de persones en quedades per a practicar esport.
Després d'ordenar i organitzar una extensa pluja d'idees inicial, el treball ha progressat sessió rere sessió fins tenir una xarxa social funcional amb totes les funcions dessitjades implementades.

Cada membre del grup s'ha basat en anteriors tasques realitzades i coneixements adquirits al llarg del curs per a complir el propòsit final d'aquesta pràctica. Per crear cert tipus de funcions ens hem inspirat en programes creats a Estructura de Dades i Algorismes i en alguna ocasió també hem compartit idees amb altres grups.

Per aconseguir la màxima productivitat possible cada membre s'ha especialitzat en funcions específiques on ha pogut explotar les seves capacitats de cara a la prosperitat del treball. La feina s'ha repartit, en la seva majoria, en tasques d'organització, programació, redacció de l'informe, revisió i correcció d'errors, entre d'altres. Cal afegir també, que tots els membres han col·laborat en tots els apartats referents al treball, encara que cadascun s'hagi enfocat més en una tasca.

La comunicació i un bon ambient de treball entre els membres de l'equip han afavorit molt positivament l'avanç d'aquest projecte.

OBJECTIUS DEL PROJECTE

En aquest secció del nostre informe ens centrarem en explicar els objectius inicials del projecte i, a més, els que ens hem proposat a mesura que anavem avançant el nostre codi. Dividim doncs aquest apartat en tres seccions:

- Objectius obligatoris
- Objectius desitjables
- Objectius lliures

En cadascun d'aquests apartats, explicarem les característiques principals de les estructures de dades, els algoritmes implementats per tal d'aconseguir el correcte funcionament del codi o el comportament esperat dins el programa.

OBJECTIUS OBLIGATORIS

Implantació d'una llista, una pila i una cua

Llista:

Per cada usuari, existeix una llista de posts que es diu posts. Cada post inclou un text i un marcador del temps en el qual es va fer aquella publicació.

També tenim un enter que conté el número de publicacions que es diu num_posts. Això ho fem servir per poder accedir al últim element o afegir un altre més fàcilment

El temps d'implementació d'aquesta estructura ha estat de una hora.

La declaració de la estructura està en les línies 13-16 del main.h.

Pila:

Les amistats de cada usuari estan estructurades com una pila, ón guardem només el punter del primer amic, i apilem els següents a sobre, canviant un punter en l'anterior. Com que les amistats no es poden rebatre, essencialment els amics són una pila sense Out.

Aquesta estructura es veu entre les línies 28-35 del main.h

Cua (queue):

S'ha implementat una estructura anomenada "Queue" amb dos nodes, el primer fa referència al primer objecte de la cua, "front"; el segon fa referència al final de la cua, "rear". Aquesta cua està organitzada amb una estructura "FIFO", First In First Out.

La cua s'utilitza per a organitzar les sol·licituds d'amistat que rep un usuari.

El temps d'implementació d'aquesta funció ha estat de tres hores i mitja. El procés de debugg i aconseguir el seu correcte funcionament ha estat més del previst.

La declaració de la estructura està en les línies 18 - 26 del main.h.

Implementació d'un algorisme de recerca

En aquest projecte hem implementat de forma repetida l'algoritme Linear Search. Ja a que el nombre d'usuari i dades és de tamany reduït s'ha decidit d'implementar la recerca secuencial. S'empra per a trobar els usuaris existents i/o la informació referent a ells.

Pren la forma "for (int i = 0; i < llista_a_cercar; i++){...}" per els arrays, com els posts, mentres per les llistes vinculades funciona amb punters, com és el cas el la funció trobar_user dins de usuari.c.

Emprant la recerca secuencial en una llista reduïda es pot trobar la informació de forma extremadament ràpida, sempre que aquesta estigui inclosa a la llista.

La complexitat de l'algoritme és $O(n)$, on n és el nombre d'objectes dintre la llista.

Per a millorar del projecte es podria implementar una recerca binaria; amb una complexitat $O\ log(n)$ es podria reduir el temps de recerca.

El temps d'implementació d'aquesta funció ha estat de cinc minuts per a cada iteració.

Implementació d'un algorisme per ordenar

Com a algoritme d'ordenament s'ha implementat el BubbleSort. La funció "Dictsort" s'encarrega d'ordenar les paraules més freqüents trobades en el diccionari. Les ordena en ordre descendent partint de la paraula utilitzada més freqüentment. Aquest algoritme té una complexitat de $O(n^2)$. Com a proposta de millora es podria implementant un algorisme MergeSort que té una complexitat de $O(n \cdot log(n))$ que resultaria en un temps d'execució inferior.
Les variables "i" i "j" s'utilitzen per a recorrer el diccionari. Les variables "value" i "key" s'empren per a ordenar-les en la de frecuencia en que apareixen.
El temps d'implementació d'aquesta funció ha estat d'hora i mitja, tenint en compte el temps de correcció i debug.
Les línies de codi encarregades d'aquesta funció són 30-44 en dict.c.

Implementació d'un diccionari
Per fer una anàlisi de les publicacions dels usuaris s'ha creat un diccionari nomenat "Dict" amb una estructura de tres variables, dues d'elles enteres. La primera rep el nom "value" la qual és un array d'enters de mida [MAX_WORDS]. La segona és "key", que és un array de les paraules en si. Finalment tenim "num", un contador de quantes paraules hi ha actualment. Per a ordenar les paraules en un factor ascendent de frecuencia de menció s'ha emprat l'algoritme Bubble Sort. La limitació d'aquest diccionari és la quantitat de paraules que pot incloure. Com a màxim, pot tenir 100 paraules ([MAX_WORDS] = 100).
El temps d'implementació d'aquesta funció ha estat de dues hores.
La declaració de la estructura la trobem entres les línies 50-54 de el main.h
L'apartat del codi encarregat del diccionari està en el fitxer dict.c, que inclou el sorteig amb bubble sort, procesar posts nous amb el diccionari, i llegir el fitxer dict.txt

OBJECTIUS DESITJABLES

Llegir dades d'una font externa
Les dades de aquest programa es guarden a 4 fitxers .txt diferents, que son 'users.txt', 'posts.txt', 'dict.txt' i 'friends.txt'
El temps d'implementació d'aquesta funció ha estat d'hora i mitja. En aquest temps està la escriptura del codi, la seva correcció, temps de debugg i finalment la creació dels fitxers.
Les línies de codi encarregades d'aquesta funció són les funcions ReadXXXXFromFile, totes trobades en el seu fitxer .c respectiu.
Això ha sigut la nostra manera de fer que la informació de la xarxa es conservés per cada ús, ja que els fitxers de text no es buiden ni perden informació al encendre el programa.

Red social temàtica
La nostre xarxa social busca motivar a la població a una vida més saludable facilitant les sortides grupals per a fer esport. El element hobby de cada persona inclou 5 esports, que tots els usuaris poden veure per incitar que es facin amics o que facin publicacions relacionades amb els esports

Vista per consola

El nostre programa es mostra exclusivament amb text. Això vol dir que hem hagut de prendre decisions per millorar la llegibilitat de la informació presentada, com per exemple separar cada publicació amb una línea de caràcters ‘-’, o també incloure tabulacions per mostrar la taula del diccionari adequadament.

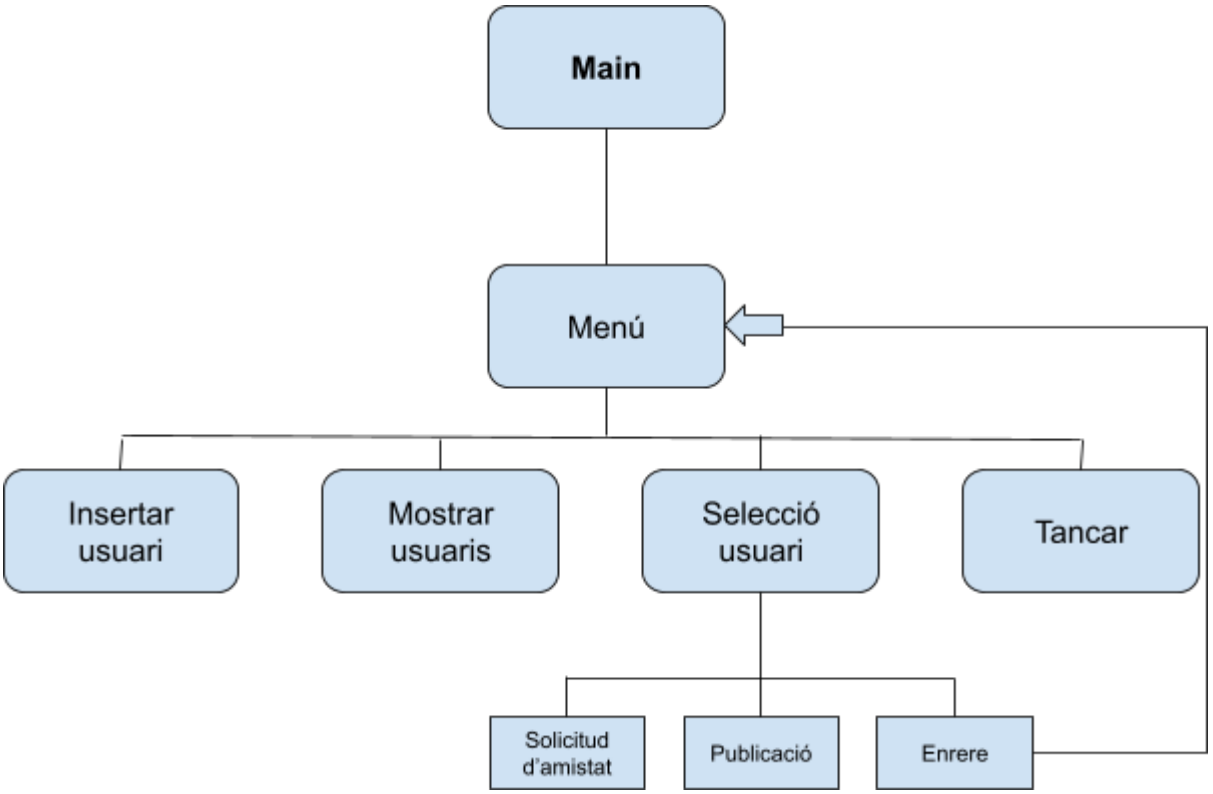
Mesura del temps d’execució

El nostre programa es basa en input constant de la persona, per tant era necessari intentar maximitzar la velocitat del programa, així l'experiència és més favorable.
Aquest objectiu es nota especialment en el bubble sort implementat per el diccionari, ja que es pot veure que els bucles for no recorren tota la llista, ja que podem suposar que per cada cicle un element es posa en la posició correcte al final del diccionari.

SOLUCIONS

En aquesta secció ens centrarem en les solucions que hem aconseguit mitjançant el treball de totes aquestes setmanes. A més mostrarem el procés de detecció i correcció d'errors i les parts del codi que hem anat modificant per tal de tenir un codi el més optimitzat possible. Per fer-ho dividirem també aquesta part en una serie d'apartats.

ARQUITECTURA DEL SISTEMA



Com podem veure en el diagrama, la nostra arquitectura consta de dos funcions principals que son:

- Main: que inicialitza la llista d'usuaris a més del diccionari i crida al menú.
- Menú: Ens dona les diverses opcions seleccionables i ens permet accedir a cadascuna d'elles.

Per altra banda, dins de la funció menú, tenim 4 opcions seleccionables que es mostren en pantalla en inicialitzar el programa:

- Insertar usuari: aquesta opció permet fer-ho de forma manual, ja que els fitxers amb usuaris adjunts es carreguen automàticament.

- Mostrar usuari: imprimeix tots els usuaris que s'han insertat fins ara, tant de forma manual com de forma automàtica.

- Seleccionar usuari: es selecciona de forma manual un usuari per tal de rebre les seves dades. Després entra en la funció *user_actions* per obrir el menú de l'usuari, que disposa de 3 opcions diferents:
    - Solicitud d'amistat
    - Publicació
    - Enrere(aquesta acció crida directament a la funció menú per tal de veure un altre cop les seves opcions)

- Tancar: finalitza el programa.

CONTROL D'ERRORS

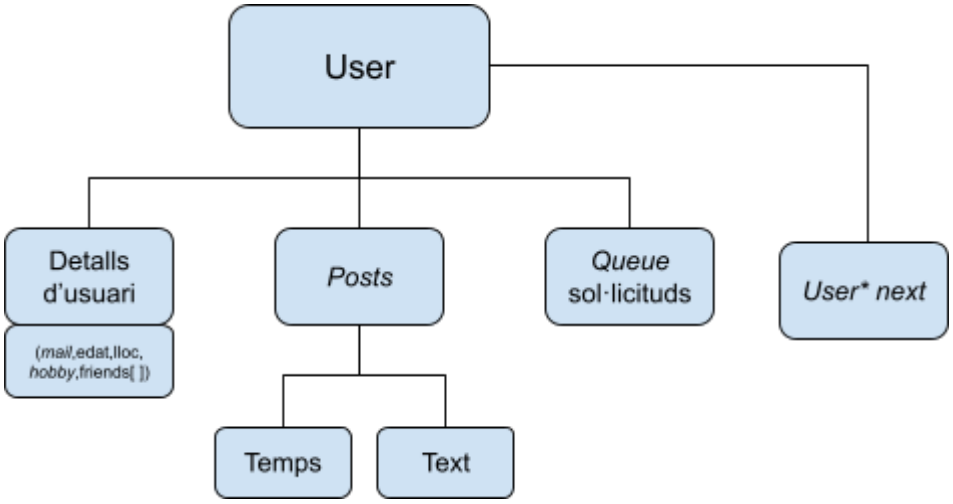
L'apartat d'errors ha sigut un tema que hem tractat de forma organitzada i on ha fet molta falta una comunicació constant .

Amb l'eina GitHub, qualsevol membre del grup gaudia d'accés total al codi en tot moment. Cadascú té horaris de treball propis , que no sempre coincideixen amb els dels altres participants, per tant, si en algun moment algú detectava un error dins del codi, es comunicaba dins el nostre chat de Whatsapp per tal de que quedi constància d'ell.

Un cop comunicat, era l'hora de buscar solucions que s'ens podien acudir i si cap d'elles funcionaba, anavem a seminaris o treballs anteriors que ens podien servir com a referència i ajuda.

Gràcies a aquest mètode hem pogut solventar la majoria d'errors sense molt problema tot i que hi han hagut excepcions que han requerit més temps de treball.

DISSENY DEL MODEL DE DADES



El **user** és l'agrupació més gran de la nostra estrucutra de dades. Conté la informació de cadascun dels usuaris dins del programa. Aquesta informació la dividim en quatre grans blocs:

- Detalls d'usuari: un llistat amb les següents dades; *mail, edat, localització,hobbies,amics*[ ].

- Posts: publicacions fetes pels usuaris que contenen també informació com el *temps* o el *text* de la publicació.

- *Queue* sol·licituds: aquest part inclou dues estructures:
    - La *queue*: guarda un punter del primer i l'últim node, les quals son sol·licituds.
    - Els nodes: tenen un enter que representa l'usuari i un punter al següent node.
- *User** next = un punter apuntant al següent usuari.

Utilitzem una variable User* head per indicar ón es troba el primer usuari.

DESCRIPCIÓ I PROCESSAMENT DEL CONJUNT DE DADES

S'han utilitzat tres grans conjunts de dades en fitxers de text per processar informació dins del programa:

- *Dict.txt*
- *Posts.txt*
- *Users.txt*
- *Friends.txt*

És important guardar aquesta informació en aquests formats per tal que es puguin conservar en tancar el programa.
S'ha implementat una funció per a cadascun dels fitxers *readDictFromFile readUsersFromFile readPostsFromFile readFriendsFromFile*
Aquestes serveixen per inicialitzar les estructures de dades corresponents. Primer tokenitzem el fitxer de text per línies i després,fent servir *sscanf* aconseguim guardar els respectius valors en les posicions correctes.

REFERÈNCIES

Aquest treball, com s'ha mencionat anteriorment, ha tingut referències de diferents fonts per tal de implementar el màxim el treball i de fer-ho de la manera més optimitzada possible.

Podem trobar exemples procedents d'internet però també s'han consultat treballs i documents d'entregues anteriors.

Es mostren a continuació:

***Chat GPT*. (s/f). Openai.com. Recuperat el 26 de abril de 2023, de https://chat.openai.com/**

***Seminari II public*. (s/f). Google Docs. Recuperat el 6 de maig de 2023, de https://docs.google.com/document/d/1M7iVUJ30E1u3YgUNKTe_R3Jx7EsUmO_Fq6EPqimb_fk/edit**

***Queue using linked list*. (s/f). Log2base2.com. Recuperat el 23 de maig de 2023, de https://www.log2base2.com/data-structures/queue/queue-using-linked-list.html**

Thakur, A. (s/f). *Clearing input buffer in C C*. Tutorialspoint.com. Recuperat el 7 de juny de 2023, de https://www.tutorialspoint.com/clearing-input-buffer-in-c-cplusplus