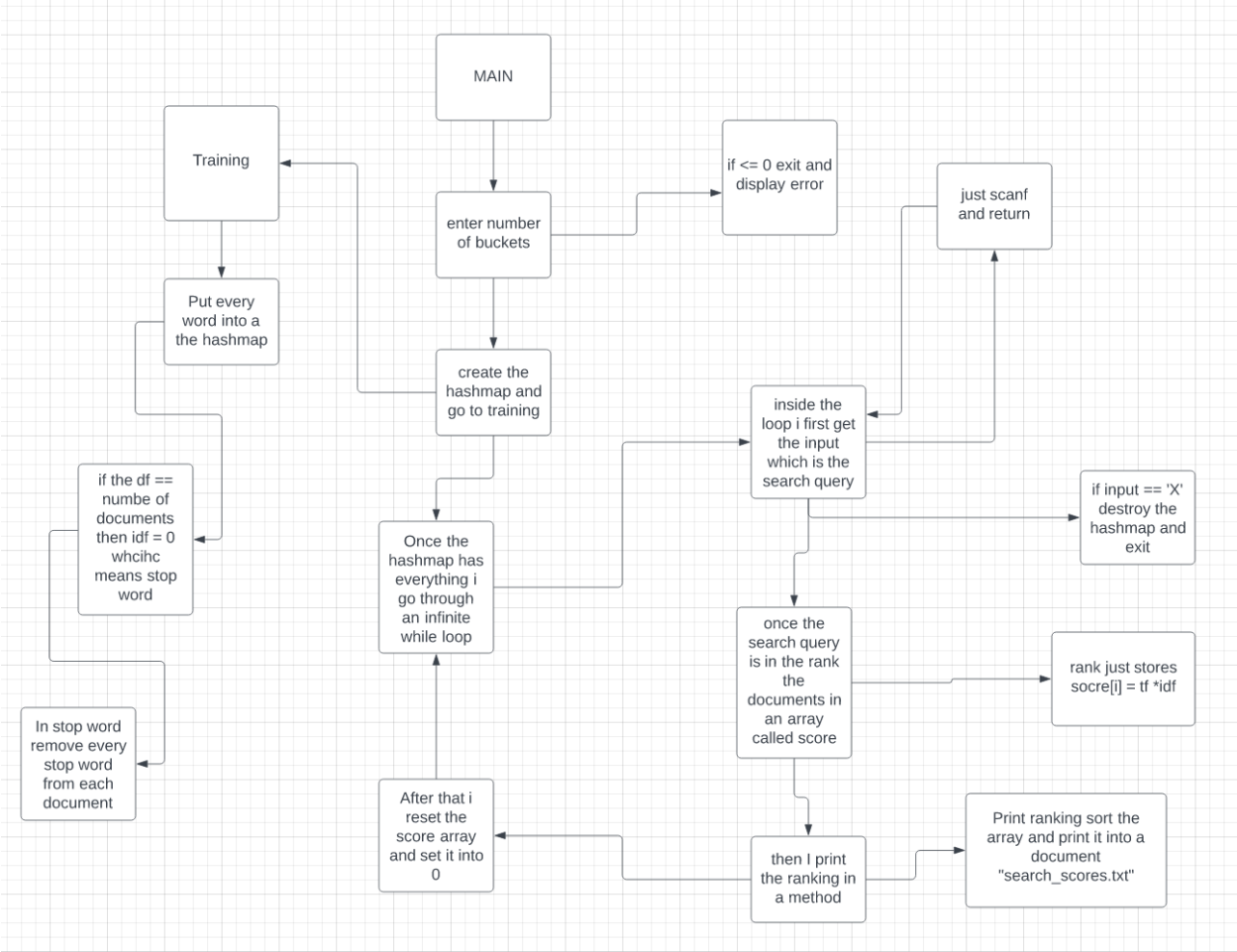
Project 5 report

I choose option 2 reading (arbitrary files from a directory)

My flow charts

The link to my flow chart is https://lucid.app/lucidchart/5c9532d7-f572-4ebe-b37b-1373da0940fd/edit?viewport_loc=-691%2C-32%2C3072%2C1577%2C0_0&invitationId=inv_726993d9-cf78-4dee-a057-d66f02e2b0da



My search.h file

In this file i defined the methods as follows the explanation of each method in detail is located in the header for search.c:

```
void training(struct hashmap * hm, int number_of_documents);
```
This method puts every word in the hashmap and stops repeated words more details in the training method
```
char * read_query(void);
```
This basically is a search query scan. More details in the read query method
```
void rank(char * input, double score[], int number_of_documents, struct hashmap * hm);
```
This puts the scores for each document by getting the tf and idf of each word and storing the score like this score[i] = tf*idf;

```
void stop_word(struct hashmap * hm, int number_of_documents, char * input, struct hashmap * removed);
```
This removes the stop word and puts multiple fil
```
void printranking(double score[],int number_of_documents);
```
This was not asked for but i added to make my main look neater it sorts prints the file into search_scores.txt

My search.c file

main(void):

In my main i first ask the user how many buckets they want if its less than 1 I exit the program and tell the user to restart the program. After that I create a hashmap with the number of documents. Then i use glob to get the number of files (more details on the Glob header). Then i do the training method which takes care of all stop words and puts other words into the hashmap. I declare the array scores to keep track of scores. Then inside a while(true) i first read query (more details on the header of read_query) and check if the user entered and X with strcmp if user entered X then i would destroy the hashmap and exit the program with a return 0; If the user put in the search_query then i would rank and print the sorted ranking into search_scores.txt after that i would reset the array scores to 0's I would do that until the user enters X as an input.

Glob:

At the beginning of my main file i make glob_t mything; and then i creat a string "pathofdocuments" = "p5docs/D*.txt" The Asterisk means anything so any number from 1,2,.....n. I do this to know the number of documents in p5 docs. Then i do this glob(pathofdocuments, 0, NULL, &mything) so i can count the number of documents in the path. By doing mything.gl_pathc it returns the number of documents. Finally i free mything by globfree(&mything) thats everything we need from glob. Since I know the number of documents

and know the files are named from D1 through Dn. when knowing the number of documents we could just get everything

Training(struct hashmap * hm, int number_of_documents)

My training method basically puts everything in a hashmap then checks if there are repeated words. I get the path name by making a variable and storing the string "p5docs/D" and then i get the number by “i” and convert it to a string and concatenate to p5docs/D finally i concatenate a string that has .txt after i put every word of document[i] into the hashmap i close document[i], then I reset the variables and do this for each i. To get the df I do a similar technique to get the df and i check if the df == number of documents then thats a stop word go to header stop_word for more details on this method. I reset the df each time however, i check for each document if the word is there once or more I increment the df. Also i create a hashmap that stores every stop word so it doesn't keep putting the same word into the hashmap if it comes later. Like architecture would still be in the hashmap if i didn't check the removed words.

read_query(void)

In my read query i first print "Enter search string or X to exit” then i create an array of characters with length of 1024 called input. Then i use this scanf(" %[^\n]%*c", input) the space before the percentage is needed because in a while loop it wouldn't work unless the space is there because the space will eat up the new line so that's why we need it. The number 1024 is the input length and the symbols are to allow the spaces without this my strtok would not work. Finally I create a character * and store the input in it and return. This is called in my main and passed on in the rank.

rank(char * input, double score[], int number_of_documents, struct hashmap * hm)

In rank i first i declare i as integers because of c we need to declare before for loops and create char D and numbers which are array. Numbers basically store the current i into a string and so i can update the current value to make it so i can use hm_get() to find the tf of a given word. But before that I created a variable called current_word and used strtok(input, “ “) . This method is to make the current word the first word. After that i check if the current word is not null as a condition in a while loop and before ending the while loop i current_word = strtok(NULL, " "); which basically goes to the next word. This happens because strtok makes the next characters in a string look like this \000 which is NULL. I did this in a debugger and found the \000. So i do strtok(NULL, " ") until the end of the input. Inside the while loop i first delete whats in D and whats in numbers by doing memset(D,0,20) which resets it amd make D[0] = ‘D’ and do the same for number. Then i use sprintf to make the number a string in number so i lets say i= 2; then number=”2” and after that i concatenate both to get “Di” where i is the number in the

for loop. For tf i use hm_get(hm, current_word, D) (this is where D comes in) to see how many times the word is in the doc and check if tf is = -1 if it is then i set it to 0. Then I check if df == 0 if it is then I would skip this word because it's not on the hashmap. If the df is not 0 then i would set idf = log(number_of_documents/df). Finally i just so score[i] += tf*idf;

stop_word(struct hashmap * hm, int number_of_documents, char * input, struct hashmap * removed)

In this method i do a for loop for each file of the document i would do the same thing i did to get the df for the files however inside the for loop i do a while loop and have hm_get(current word) != -1 and keep removing that word until its -1(which means there are no words on that file in the hashmap) I would do that for each of the documents. Doing this removes the word each time so once the df==number of documents i know that i have to stop that word. Finally when removing a word i put the word into a hashmap that stores stop words. This method is only called by the training method.

printranking(double score[],int number_of_documents)

In printranking i first create a variable fp1 open the file `"search_scores.txt"` with “a+” which will append if the file exists and create if the file doesn't exist.
Then I create a new array called numbers numbered from one to the number of documents with a for loop to keep track of the text file numbers D1,D2,...Dn.
Then I do a bubble sort with two temporary variables one for the scores which is a double and the other is for the number of the file which is an integer. This is because we want numbers[i] is still = to scores[i]
Finally I go through one last loop from 1 to the number of documents. And print it into the fp1 by using fprintf(fp1, “D%d: %0.3f\n”, numbers[i], score[i]) i put D first because printing the letter D then a number is a smart way to do it and there is no point of making an array that stores D1 and D2 and D3 so using the number makes it much simpler

My makefile

I copied the makefile for homework 6 into this project and I made two changes in my make file the first is to compile with the log we need -lm so I added it to the end of the compiling and secondly I changed test.c and test.o to search.c and search.o

My hashmap

I used the simple hashmap I used the same exact hashmap as homework 6. However, I changed the name of the function as hash_table_insert() is the same as put() before I just changed the name and hash_code() is the same as hash() in the homework. That is all the modifications that i have done.