# Database​ ​Theory​ ​And​ ​Design

The program below computes the counts, average age from examine table which is the outer relation joined with dogs table which is an inner relation. A simple nested loop join algorithm is used. First I read the of the DFile from where I get the did of the dogs whose age >=7 and then I read the contents of the EFile from where I use the search and sorting algorithms to compare the did’s of the dogs to the did’s of veterinarians who have examined the dog . Also the average fee that was charged for these dogs and the number of times it visited the vet is counted and computed

Source code:

#include <iostream>

#include <fstream>

#include <ctime>

#include <vector>

using namespace std;

int binarySearch(vector<int>&, int); //function to search

void quickSort(vector<int>&, vector<int>&, int, int);

int main()

{

ifstream examine\_File;

ifstream dog\_File;

int did,fee,vid1;

int ddid,dage=0;

int dindex=0;

clock\_t start,stop;

vector<int> temp\_did\_Array;// Vector to store did from examine after comparison of did's

vector<int> fee\_Array ;//Vector to store fee from examine after comparison of did's

vector<int> temp\_age\_Array;//vector to store age aftercomapring the value of age

vector<int> dog\_did\_Array;//vector to store did from dog

vector<int> dog\_age\_Array;// vector to store age from dog

examine\_File.open("EFile");

dog\_File.open("DFile");

start=clock();

while (dog\_File >> ddid >> dage)

{

dog\_did\_Array.push\_back(ddid); // storing did

dog\_age\_Array.push\_back(dage); // storing age

dindex++;

}

quickSort(dog\_did\_Array,dog\_age\_Array,0,dindex-1);

int k = 0;

// Starting to import info from xamine file and comparing did with did of 'dog'

while (examine\_File >> vid1 >> did >> fee)

{

int val = binarySearch(dog\_did\_Array, did); // checking if did of examine is on 'dog'

if (val != -1)

if(dog\_age\_Array[val]>=7) // if did matches, store age and corresponding fee to vectors

{

temp\_did\_Array.push\_back(did) ;

fee\_Array.push\_back(fee);

temp\_age\_Array.push\_back(dog\_age\_Array[val]);

k++;

}

}

examine\_File.close();

dog\_File.close();

quickSort(temp\_age\_Array,fee\_Array,0,k-1);

int count1=1;

float avg=0;

float sum=fee\_Array[0];

avg=sum;

for(int c1=0;c1<k;c1++)

{

if(temp\_age\_Array[c1]!=temp\_age\_Array[c1+1])

{

cout<<"\n";

cout<<"Age of the dog "<< temp\_age\_Array[c1] <<"count is "<< count1 <<" average is "<<avg<<"\n";;

count1=0;

sum=0;

avg=0.0;

}

count1++;

sum=sum+fee\_Array[c1+1];

avg=sum/count1;

}

stop=clock();

cout<< "\n time: "<<(double(stop-start)/CLOCKS\_PER\_SEC)<<" seconds"<<endl;

}

int binarySearch(vector<int>& V1, int num){

int mid, first = 0;

int last = V1.size();

while (first < last){

mid = first + (last - first)/2;

if (num > V1[mid])

first = mid+1;

else if (num < V1[mid])

last = mid;

else return mid;

}

return -1;

}

void quickSort(vector<int>& V1,vector<int>& V2, int left, int right) {

int i = left, j = right;

int tmp;

int pivot = V1[(left + right) / 2];

/\* partition \*/

while (i <= j) {

while (V1[i] < pivot)

i++;

while (V1[j] > pivot)

j--;

if (i <= j) {

tmp = V1[i];

V1[i] = V1[j];

V1[j] = tmp;

tmp = V2[i];

V2[i] = V2[j];

V2[j] = tmp;

i++;

j--;

}

}

/\* recursion \*/

if (left < j)

quickSort(V1, V2, left, j);

if (i < right)

quickSort(V1, V2, i, right);

}

Output:

