CIS421 – Professor Zhu

Assignment 5 Report

The University of Michigan – Dearborn

Thomas Opolski
Due: 12-12-2019

Program Overview

In this assignment, you will have a chance to implement the hash-join algorithm for a DBMS. You can use C, C++, Java or any other high-level programming language to write the program. However, your program should include sufficient comments to make it readable. You need to turn in (1) a brief report/description about your program design and implementation (e.g., high-level program diagram and data/file structures) and program usage; (2) your program source code; (3) proof of compilation (e.g., the screen snapshot of a successful compilation); and (4) sample execution outputs. Please assemble all the above required contents in a single Word or PDF file for your submission. The program specification is given as follows.

Let R1(a1, a2, a3) and R2(b1, b2, b3, b4) be two relations with all integer attributes. Tuples in these two relations are sequentially stored in two data files, respectively.

• Use the hash-join algorithm to implement a join (equijoin) of R1 and R2. Assume that the hash function is f(k) = k mod N, where N is the number of buckets allowed in your hash structure/table.

• Your program should allow a user to choose the joining attributes from the two relations, i.e., performing R1 ./R1.ai=R2.bj R2 for any chosen pair of ai and bj , where ai is the i-th attribute in R1 (1 ≤ i ≤ 3) and bj is the jth attribute in R2 (1 ≤ j ≤ 4). For example, a user may want to perform R1 ./R1.a2=R2.b3 R2.

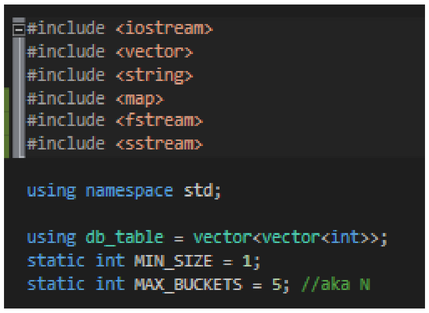
• Your program should display the join result and output the selectivity of the join.

• You may request a user to interactively input the necessary parameters, such as the data file names for R1 and R2, the number of tuples in each relation, and the joining attributes (e.g., 1 for the 1st attribute, 3 for the 3rd attribute).

• Use your program to perform several joins for different relation instances of R1 and R2.

Program Description and Implementation

Global Declarations



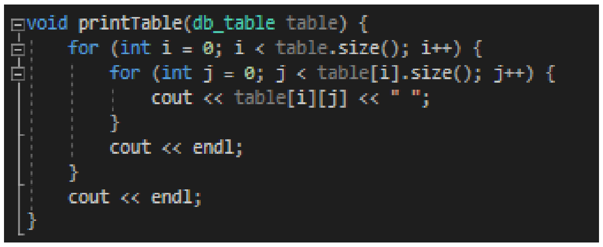
```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <fstream>
#include <sstream>

using namespace std;

using db_table = vector<vector<int>>;
static int MIN_SIZE = 1;
static int MAX_BUCKETS = 5; //aka N
```

using db_table = vector<vector<int>>;
This is used to define the table structure in the form of a 2D vector, or a vector in a vector.

static int MIN_SIZE = 1;
This is used to declare the minimum/starting attribute for joining tables

static int MAX_BUCKETS = 5;
This is used to declare the number of buckets for the hashing algorithm.

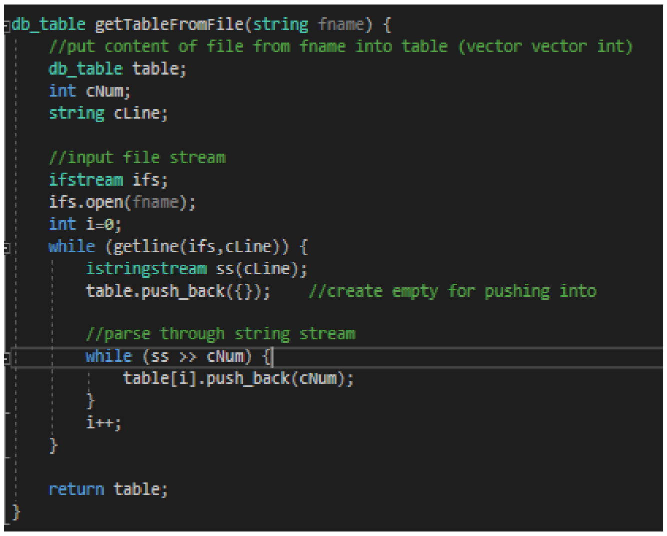
printTable function



```
void printTable(db_table table) {
    for (int i = 0; i < table.size(); i++) {
        for (int j = 0; j < table[i].size(); j++) {
            cout << table[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;
}
```

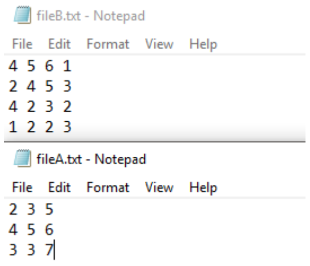
This function takes in a table to be printed out, and simply traverses through all its elements to output the contents in the console.

getTableFromFile function



```
db_table getTableFromFile(string fname) {
    //put content of file from fname into table (vector vector int)
    db_table table;
    int cNum;
    string cLine;

    //input file stream
    ifstream ifs;
    ifs.open(fname);
    int i=0;
    while (getline(ifs,cLine)) {
        istringstream ss(cLine);
        table.push_back({});    //create empty for pushing into

        //parse through string stream
        while (ss >> cNum) {
            table[i].push_back(cNum);
        }
        i++;
    }

    return table;
}
```

This function takes in a string filename and opens up that file to be read. Files are parsed with an input stringstream to get stringstream lines, and parsed again to get individual integers. The format of these files are expected to be as follows:

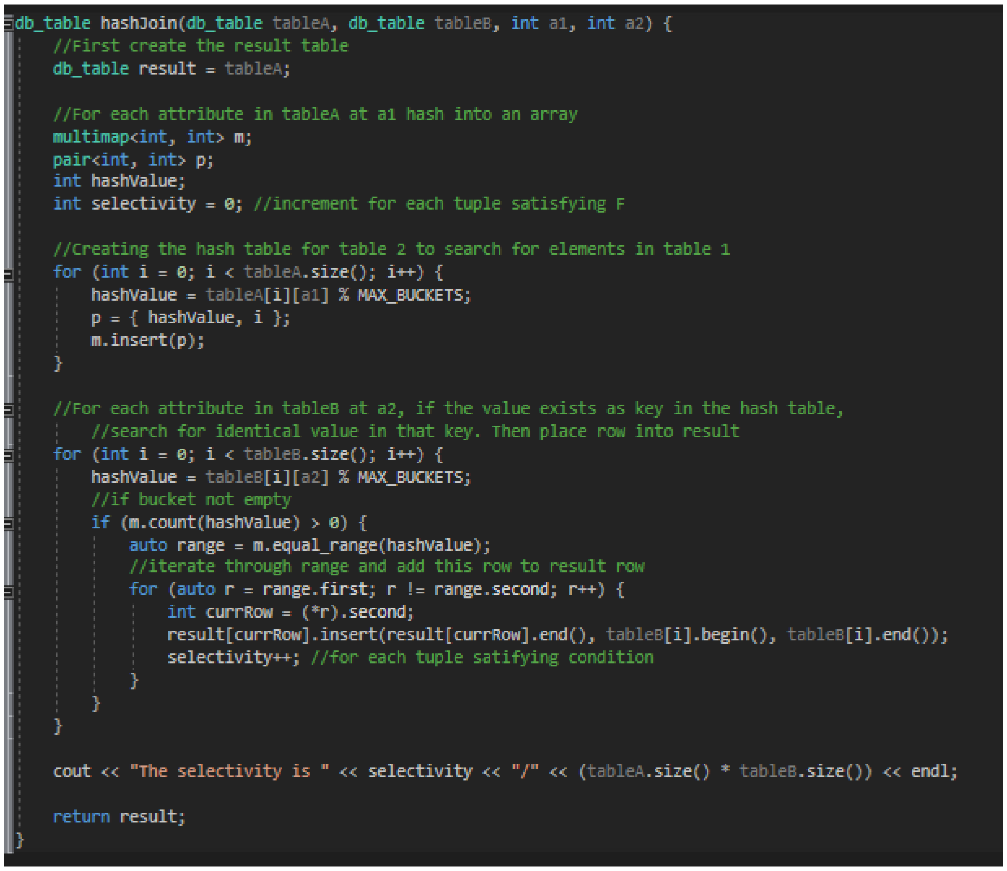


```
fileB.txt - Notepad
4 5 6 1
2 4 5 3
4 2 3 2
1 2 2 3
```

```
fileA.txt - Notepad
2 3 5
4 5 6
3 3 7
```

Where each tuple is a row of space separated integers and each column is an attribute.

hashJoin function



```
db_table hashJoin(db_table tableA, db_table tableB, int a1, int a2) {
    //First create the result table
    db_table result = tableA;

    //For each attribute in tableA at a1 hash into an array
    multimap<int, int> m;
    pair<int, int> p;
    int hashValue;
    int selectivity = 0; //increment for each tuple satisfying F

    //Creating the hash table for table 2 to search for elements in table 1
    for (int i = 0; i < tableA.size(); i++) {
        hashValue = tableA[i][a1] % MAX_BUCKETS;
        p = { hashValue, i };
        m.insert(p);
    }

    //For each attribute in tableB at a2, if the value exists as key in the hash table,
        //search for identical value in that key. Then place row into result
    for (int i = 0; i < tableB.size(); i++) {
        hashValue = tableB[i][a2] % MAX_BUCKETS;
        //if bucket not empty
        if (m.count(hashValue) > 0) {
            auto range = m.equal_range(hashValue);
            //iterate through range and add this row to result row
            for (auto r = range.first; r != range.second; r++) {
                int currRow = (*r).second;
                result[currRow].insert(result[currRow].end(), tableB[i].begin(), tableB[i].end());
                selectivity++; //for each tuple satifying condition
            }
        }
    }

    cout << "The selectivity is " << selectivity << "/" << (tableA.size() * tableB.size()) << endl;

    return result;
}
```

The main functionality of the program, the hashJoin function takes 4 parameters which are the two tables to be joined, along with the attribute of each at which they wish to be joined.

First, I declared a table called result that is identical to the first table. I then used a multimap for storing the hashed values of the first table as the key, and then an int i at which the tables tuple is stored. Next, I iterated through the second table with an identical hashing method, as to which I would then check if the bucket for that hash value was empty or not. If not, I iterated through the range of values and added the current row of the second tuple into the row of the hashed row in the table result. I then increased int selectivity for each of the values added, to get the number of tuples satisfying the condition. I took this value and using the equation provided below:

* **Selectivity of** $\sigma_F(R)$

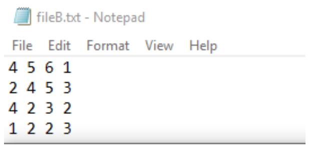
$$= \frac{\text{\# of tuples satisfying F}}{|\text{R}|}$$

I calculated and output the selectivity of the join to the console. I then returned the result table as to which I would simply print it out afterwards.

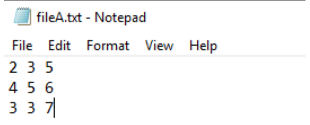
Program Usage:

Inputs: filenameA, filenameB, atrbA, atrbB
Output: joined table and selectivity
You will be prompted to input two files that you wish to get your tables from. These are expected to be simple txt files.
Next you will be prompted to input the attributes from both of the tables that you wish to join.
Then the outputs will show the tables parsed into the data structures in the program, along with the selectivity and the final output table at the end.

Example Input:



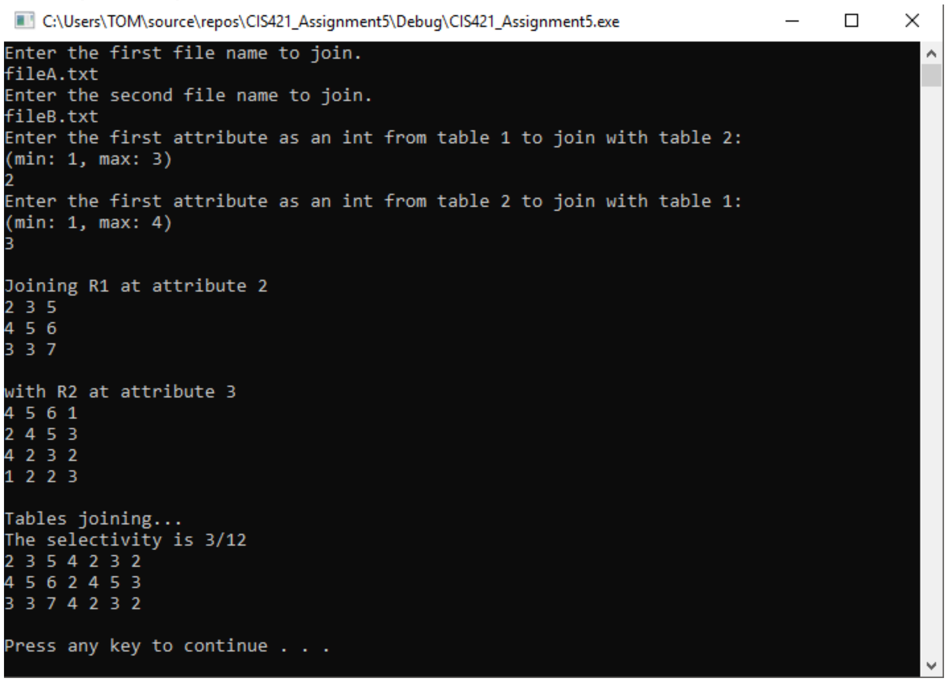
```
4 5 6 1
2 4 5 3
4 2 3 2
1 2 2 3
```



```
2 3 5
4 5 6
3 3 7
```

fileA.txt
fileB.txt
2
3

Example Output:


C:\Users\TOM\source\repos\CIS421_Assignment5\Debug\CIS421_Assignment5.exe

```
Enter the first file name to join.
fileA.txt
Enter the second file name to join.
fileB.txt
Enter the first attribute as an int from table 1 to join with table 2:
(min: 1, max: 3)
2
Enter the first attribute as an int from table 2 to join with table 1:
(min: 1, max: 4)
3

Joining R1 at attribute 2
2 3 5
4 5 6
3 3 7

with R2 at attribute 3
4 5 6 1
2 4 5 3
4 2 3 2
1 2 2 3

Tables joining...
The selectivity is 3/12
2 3 5 4 2 3 2
4 5 6 2 4 5 3
3 3 7 4 2 3 2

Press any key to continue . . .
```

Full Source Code:

FileA.txt

```
2 3 5
4 5 6
3 3 7
```

FileB.txt

```
4 5 6 1
2 4 5 3
4 2 3 2
1 2 2 3
```

assignment5_source.cpp

```
/*
Creator: Thomas Opolski
Task: Use the hash-join algorithm to implement a join (equijoin) of R1 and R2. Assume that the
hash function is f(k) = k mod N, where N is the number of buckets allowed in your hash
structure/table.
Date Created: 12/9/19
Date Modified: 12/10/19
*/

#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <fstream>
#include <sstream>

using namespace std;

using db_table = vector<vector<int>>;
static int MIN_SIZE = 1;
static int MAX_BUCKETS = 5; //aka N

/*
--Tables for testing purposes--

db_table table1 = {
        {2, 3, 5},
        {4, 5, 6},
        {3, 3, 7}
};

db_table table2 = {
        {4, 5, 6, 1},
        {2, 4, 5, 3},
        {4, 2, 3, 2},
        {1, 2, 2, 3}
};
*/
```

```
void printTable(db_table table) {
	for (int i = 0; i < table.size(); i++) {
		for (int j = 0; j < table[i].size(); j++) {
			cout << table[i][j] << " ";
		}
		cout << endl;
	}
	cout << endl;
}

db_table getTableFromFile(string fname) {
	//put content of file from fname into table (vector vector int)
	db_table table;
	int cNum;
	string cLine;

	//input file stream
	ifstream ifs;
	ifs.open(fname);
	int i=0;
	while (getline(ifs,cLine)) {
		istringstream ss(cLine);
		table.push_back({});		//create empty for pushing into

		//parse through string stream
		while (ss >> cNum) {
			table[i].push_back(cNum);
		}
		i++;
	}

	return table;
}

db_table hashJoin(db_table tableA, db_table tableB, int a1, int a2) {
	//First create the result table
	db_table result = tableA;

	//For each attribute in tableA at a1 hash into an array
	multimap<int, int> m;
	pair<int, int> p;
	int hashValue;
	int selectivity = 0; //increment for each tuple satisfying F

	//Creating the hash table for table 2 to search for elements in table 1
	for (int i = 0; i < tableA.size(); i++) {
		hashValue = tableA[i][a1] % MAX_BUCKETS;
		p = { hashValue, i };
		m.insert(p);
	}

	//For each attribute in tableB at a2, if the value exists as key in the hash table,
		//search for identical value in that key. Then place row into result
	for (int i = 0; i < tableB.size(); i++) {
				hashValue = tableB[i][a2] % MAX_BUCKETS;
				//if bucket not empty
				if (m.count(hashValue) > 0) {
					auto range = m.equal_range(hashValue);
					//iterate through range and add this row to result row
					for (auto r = range.first; r != range.second; r++) {
						int currRow = (*r).second;
						result[currRow].insert(result[currRow].end(), tableB[i].begin(), tableB[i].end());
						selectivity++; //for each tuple satifying condition
					}
				}
		}

		cout << "The selectivity is " << selectivity << "/" << (tableA.size() * tableB.size()) << endl;

		return result;
}


int main() {

		int atrbA, atrbB;
		string filenameA, filenameB;
		db_table tableA, tableB;
		db_table result;

		//User filename inputs
		cout << "Enter the first file name to join." << endl;
		cin >> filenameA;
		cout << "Enter the second file name to join." << endl;
		cin >> filenameB;

		//Getting tables from files
		tableA = getTableFromFile(filenameA);
		tableB = getTableFromFile(filenameB);

		//Getting join attributes from user
		cout << "Enter the first attribute as an int from table 1 to join with table 2:" << endl;
		printf("(min: %d, max: %d)\n", MIN_SIZE, tableA.size());
		cin >> atrbA;

		cout << "Enter the first attribute as an int from table 2 to join with table 1:" << endl;
		printf("(min: %d, max: %d)\n", MIN_SIZE, tableB.size());
		cin >> atrbB;

		//Printing tables
		cout << endl << "Joining R1 at attribute " << atrbA << endl;
		printTable(tableA);
		cout << "with R2 at attribute " << atrbB << endl;
		printTable(tableB);

		cout << "Tables joining..."<< endl;
		//Hash join tables
		result = hashJoin(tableA, tableB, atrbA-1, atrbB-1);
```

```
	//Print final table
	printTable(result);

	system("pause");
	return 0;
}
```