CSCI 2270 – Data Structures and Algorithms
Instructor: Hoenigman/Jacobson
Final project
Due Friday, May 5, before 3pm

Hash table collision algorithm analysis

In this project, you need to build a hash table using two different algorithms for collision resolution that we discussed in lecture – open addressing and chaining. Your hash table should be build from the provided data set on Moodle called *playerData*. You also need to evaluate the performance of each algorithm in terms of how many collisions are generated while the table is built and the operations required to retrieve data from the table. Your evaluation and the results need to be described in a 2 - 3 page paper, single-spaced, 12pt. Times New Roman font.

Assignment data

There is a file on Moodle called *playerData* that includes every Major League Baseball player on every team since 1985. For example, the first two rows in the file show the header row, which shows what each column includes, and the first row of data. The first row of data is:

1985, ATL, NL, barkele01, 870000, Len, Barker, 1955, USA, 225, 77, R, R

The fields in the header row show:
yearID – year, between 1985 and 2016
teamID – three-letter abbreviation for team
leagueID – league that the team belongs to, either National League (NL) or American League (AL)
playerID – ID for the player, used in the Lahman database where I downloaded data
salary – how much money the player made that year
firstName – player's first name
lastName – player's last name
birthYear – year the player was born
birthCountry – country where the player was born
weight – player's weight
height – player's height
bats – how the player bats, either right (R) or left (L) handed, or switch (S).
throws – how the player throws, either right (R) or left (L) handed.

Assignment requirements

Build hash table

You need to process the data provided and build a hash table, where the key for entries in the hash table is the player's first and last name, concatenated. For example, for the first row in the file, the key would be *LenBarker*.

For a hash function, you can use the *hashSum* function discussed in lecture, or any other hash function that you want to try. In your project report, describe the hash function that you use.

Each player should only appear in the hash table one time. Each record in the hash table needs to include the player's information, including first and last name, birth year and country, height and weight, and batting and throwing. Each record also needs to include a list of all the teams and years that the player was active, and their salary for that year. For example, the hash table entry for Len Barker would be:

First name: Len
Last name: Barker
Year born: 1955
Country born: USA
Weight: 225
Height: 77
Bats: R
Throws: R
Teams and salary:
1985, ATL, NL, 870000
1986, ATL, NL, 880000
1987, ATL, NL, 890000
1987, ML4, AL, 72500
1988, ATL, NL, 900000

The teams and salary entries are taken from the *playerData* file for every row where the player's name appears. Note: you could have multiple players with the same name. You will need some method for verifying the uniqueness of a player. For example, if there were another Len Barker in the data file, you might want to check the height and weight or year born information to determine if it's the same person.

Set hash table size using command line argument
Use a command line argument to set the size of the hash table. There are 5072 unique players in the *playerData* file. Run your program multiple times, using 5072 as the minimum hash table size, and increase the size of the hash table up to 15K-20K. In your project write up, describe the effect that increasing the hash table size had on the number of collisions for both algorithms.

Collision analysis

As part of this assignment, you need to evaluate the performance of two collision resolution algorithms – open addressing and chaining. My suggestion for approaching this problem is to build two hash tables as you are reading the data from the file – one hash table resolves collisions using open addressing and the other hash table resolves collisions using chaining.

Collisions during building the table
As you are building the hash table, you need to count the number of collisions that each collision resolution algorithm generates. You also need to count the number of search operations needed to find a location for the record. For example, if you are using chaining and there is a collision and you traverse three records in a linked list to find an open location, then the number of search operations is 3. If you get a collision using open addressing and evaluate 4 entries in the hash table before finding an open location, then the number of search operations is 4.

Once the table is built, display the following information clearly for the user:

Hash table size: <table size>
Collisions using open addressing: <collisions>
Collisions using chaining: <chaining collisions>
Search operations using open addressing: <search ops>
Search operations using chaining: <search ops chaining>

User input

Your program needs to present the user with the option of looking up a player in the hash table. A suggested menu is as follows:

1. Query hash table
2. Quit program

If the user selects Query hash table, they should be asked to input the name of a player. If the player is in the hash table, display the record for that player, including all of the information shown on the previous page for the Len Barker example. If the player is not in the hash table, display "Player not found", and display the menu again so the user can select another player or quit the program.

Linear searches needed to retrieve data
In addition to displaying player information, your program should also display the number of search operations needed to find the player in the hash table for both open addressing and chaining.

For example, if the hash code for a player is 0, and no other players are stored at that index, then there would be zero searches needed to find the players information. However, if there are 5 players with the same hash code, and the player you need was stored at the end of the linked list for that index using chaining, then the number of searches is 5.

Display the following information, in addition to the player information:

Search operations using open addressing: <search ops>

Search operations using chaining: <search ops chaining>

Hints

Your code needs to be separated into multiple files, including a HashTable.h, HashTable.cpp, and FinalProject.cpp, similar to other assignments in this class.

If you use an array for your hash table, you will need to use dynamic memory to create the array. For the chaining algorithm, you store a dynamically created array of pointers. For the open addressing algorithm, you need a regular dynamic array.

Each player should only be stored in the hash table one time. The list of teams for that player needs to be an item in the player's record.

Written requirements

Your project needs to include a 2 - 3 page paper that describes your project and the results of using the two different collision resolution algorithms. Your paper needs to include the following sections.

Purpose - What is the purpose of this project? What are you evaluating?

Procedure - What algorithms are you using? This section should include a description of how each algorithm works, including the data structures that each algorithm uses. Also included in this section are the sizes of the hash tables you used.

Data – Describe the data set used for this project. This should include a description of the data, which field, or fields, was used to generate the hash table key, and the number of data points in the data set. This section should also include how the data was structured in each record in the hash table.

Results – Describe how each collision resolution algorithm performed, including number of collisions and search operations. If one algorithm performed better than the other, explain any intuition you have about the difference in performance.

Project Extension

If you have some extra time at the end of the semester here, and want to tackle a little extra data structures programming, you can choose to take on this additional task.

Answer the following question:

Given two players, modify your program to determine if the players shared a teammate, but never played on a team together.

The challenge here is that every player is an entry in the hash table, however, identifying connections between players requires knowing what teams they played for. You need to design a data structure that will allow you to follow connections between players. A brute-force solution is to identify two players and then loop

through all 26,000 player-team entries that might connect them. However, this approach is not worth any points.

To solve this problem, you can modify the hash table in any way you need to in order to store the required data, within reason. (If you assign all 26,000 data points to each player record in the hash table this is not an example of reasonable.)

What to display

If you take on the project extension, modify your user menu to add an item for Find Common Teammate. If the user selects this option, you should ask for the names of two players, and then search for the teammate, or teammates, that they share. Display the names, teams, and years for the common teammates. If there are no shared teammates, display "No shared teammates".

There is a separate link on Moodle to submit your solution to this challenge problem. I recommend that you get the required assignment working first and submit that. Then, copy the code and add the project extension code to that.

Working in teams

You can work on the code with one other person. Both of you need to submit the code as part of your final project submission. The project write up needs to be individual work - each person on the team needs to produce their own write up.

What to submit

Submit your code and report as FinalProject.zip to the Final Project Submit link on Moodle. If you take on the Project Extension, submit your code and a description of the data structure and algorithm you used to solve the problem, to the Project Extension Submit link on Moodle.