CSCI 2270
Project

ANY USE OF TOOLS SUCH AS CHATGPT WILL BE CONSTRUED AS PLAGIARISM AND RESULT IN A 0

Learning Objectives:

- Hashing
- Chaining
- Priority Queues

Grading policy

- You are required to schedule a **mandatory interview grading** with course staff.
- With out interview grading your project grade will be 0

Please read this document carefully before you start working

In this project you will create and maintain a data structure for efficient storing, retrieving and manipulating of movie reviews. Each movie review consists of four informations given as

1. **movieName:** Name of the movie
2. **comment:** review/comment about the movie
3. **user:** this will store the commenter's name
4. **time:** it will store the information about when the comment was made by the user. Time id in 24h format

Here is one example of a movie review

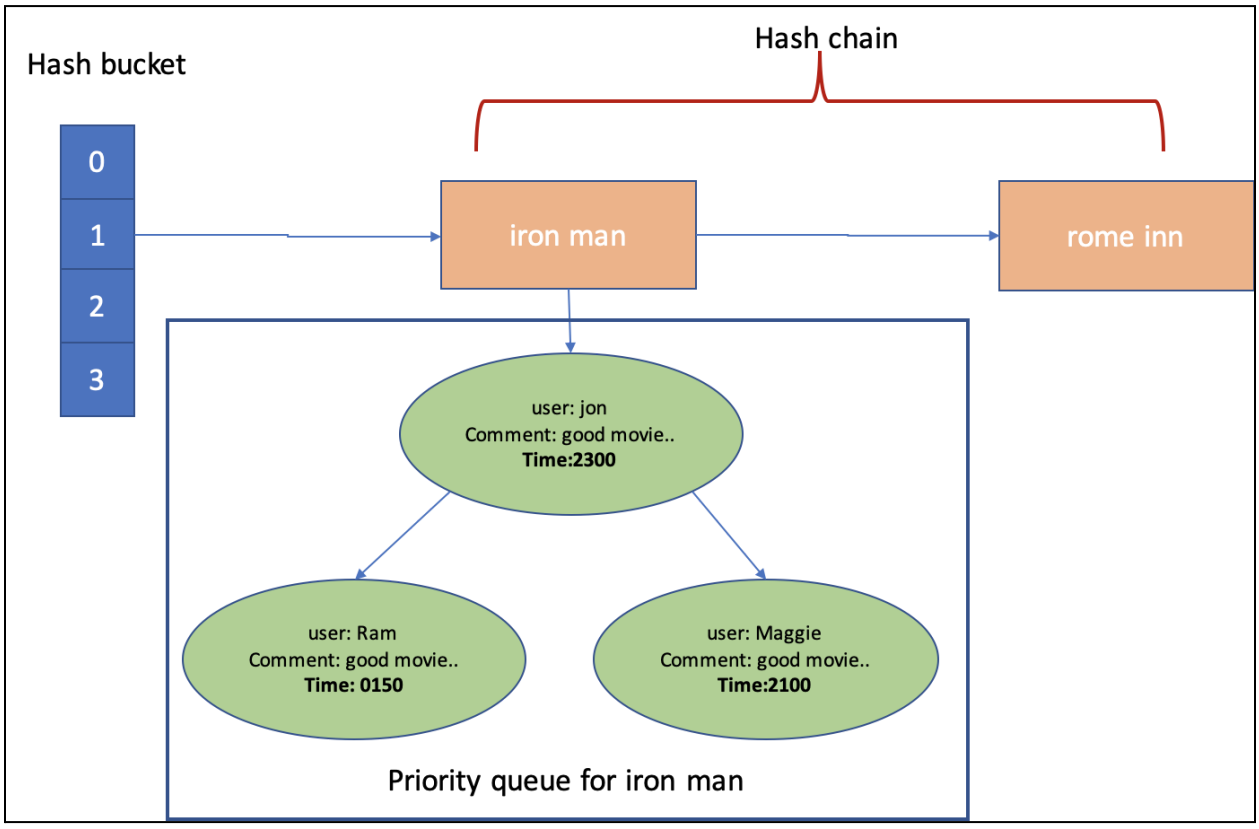
movieName: iron man
comment: The first Iron Man film was intense, right from the get go.
user: A. Mark Wilburn
time: 0317

Therefore you will be using the following structure (can be found at PriorityQueue.hpp)-

```
struct ReviewInfo{
    string movieName;
    string comment;
    string user;
    int time;
};
```

Overview of the data structure

You are supposed to store the movie reviews in a hash table with chaining. Each node in the chain will represent a distinct movie that has the same hash values. For example ‘iron man’ and ‘roam inn’ will have the same hash value and they will reside in the same chain. With each movie node there will be a priority queue. You need to organize the review information for a particular movie in the priority queue of the node based on the time. Most recent review info should be at the front of the queue.



During hashing the hash function will take the movieName and will give you the bucket index in the hashtable. Based on the index, retrieve the head appropriate chain. Each movie will have a distinct node in the chain. Therefore, find the node responsible for the current movieName. And then depending on the purpose, manipulate the node. Note, as mentioned above and depicted in the image each node will have a priority queue associated with it. Node structure is defined in the hash.hpp as follows-

```
struct node
{
    string movieName;
    PriorityQ pq;
    struct node* next;
};
```

Priority Queue

Each node of hash table chain will store a priority queue to store the information about a review. Note each review is stored as an instance of structure ReviewInfo. The queue will be organized on the basis of time subfield of ReviewInfo. The higher value of time signifies higher priority. Therefore choose proper heap implementation (think about the question - do you need a max heap or minheap)? This is an array based implementation of heap. The array will be created dynamically from the constructor. Note each element of the array is of type ReviewInfo

- `PriorityQ(int capacity)`:constructor
- `int parent(int index)`: returns parent index for an index
- `int leftChild(int index)`: return the index of the left child for the given index
- `int rightChild(int index)`: return the index of the right child for the given index
- `void peek()`:
    - If queue is empty print `"no record found"`
    - Else print the information about the item at the top (item with highest priority)
        - `cout<< "Movie: "<<rw.movieName<<endl;`
        - `cout<< "User:"<<rw.user<<endl;`
        - `cout<< "Comment:"<<rw.comment<<endl;`
        - `cout<< "Time:"<<rw.time<<endl`
- `void heapify(int index)`: maintain the heap as per the priority
- `void pop()`: remove the top priority element from the queue
- `void insertElement(ReviewInfo value)`: Insert an element in the queue. After insertion you need to make sure that the heap property is maintained.
- `void print()`: print the contents of the queue. Go over the array and print
    - `cout<<"\t"<<"User: "<<heapArr[i].user<<endl;`
    - `cout<<"\t"<<"Comment: "<<heapArr[i].comment<<endl;`
    - `cout<<"\t"<<"Time: "<<heapArr[i].time<<endl;`
    - `cout<<"\t"<<"====="<<endl;`

- `void deleteKey(string user):` Delete an entry with a specific user from the queue.

Hashing

For this project you are required to implement hashing with chaining for collision resolution. Hash.cpp should implement all the functions required for hashing. Please refer to the hash.hpp for class definition and function declaration. Note each node will contain a priority queue to store the information such as comment,user and time. Node structure was discussed above and can be found in hash.hpp.

HashTable will have a dynamically allocated array pointer given as **`node* *table`**. Note, the array will store the heads of the linked list chains. Therefore each element of the array is of type `node*` and the other '*' represents the fact that an array is created with a pointer (recall how with a pointer we created dynamic memory allocation for arrays).

The class of HashTable will also store a variable for **`numCollision`** to keep track of the number of collisions. Remember if two keys x and y are such that x!=y but hash(x)==hash(y) then only it is a collision. So, for example you entered a record for say '`iron man`' and then you entered a second record for '`iron man`'. Here there is no collision. Now you entered a record for '`roam inn`'. And hash(`iron man`) == hash(`roam inn`). Now you have a collision. So the variable **`numCollision`** should be adjusted accordingly. Beyond this any other insertion of '`iron man`' or '`roam inn`' will not change **`numCollision.`**

Member functions for HashTable are listed as follows-

- **`HashTable(int bsize):`** This is the constructor. Create a HashTable of size bsize.
- **`node* createNode(string movieName, node* next):`** This function will create a linked list node for the chain with movieName. During creation of the node, create the priority queue of size 50. You will call it from the insertItem function.
- **`unsigned int hashFunction(string movieName):`** This function calculates the hash value for a given string. To calculate the hash value of a given string, sum up the ascii values of all the characters from the string. Then take the% operator with respect to tableSize.

$$hashFunction(S) = (\sum_{ch \in S} ch)\%tableSize$$

- `node* searchItem(string movieName)`: Search for a movieName in the hash table. If found returns the node, else returns null.
    a. Calculate the hash function for the argument movieName.
    b. Retrive the chain head from the table
    c. Traverse the chain to find the node containing the movieName.
    d. On successful search return the node else return null.
- `void insertItem(ReviewInfo movie)`: Insert a movie to the hash table.
    a. First search for the node with movie.movieName
    b. If a node exists in the hash table for that movie, insert the ReviewInfo movie in the priorityQueue of that node.
    c. If the search method returns a null then
        - Create a new node for that movie.movieName.
        - Add the ReviewInfo movie in the priorityQueue of that node. Use `insertElement` of the priority queue.
        - If the corresponding chain head in the hashTable bucket is null then this is the first node of that chain. So update the chain head in the table accordingly.
        - If the corresponding chain head in the hashTable bucket is not null, then this is a collision. Update the numCollision. And add the node the linkedlist accordingly.
- `void buildBulk(string fname)`:This function will populate the hash table from a file. Path of the file will be passed from driver code. The file path will be passed as fname. The file is ; separated file and the format of a line is-

    movieName;comment;user;time

    For example

    ```
    iron man;The first Iron Man film was intense, right from the get go.;A. Mark Wilburn;0317
    ```

Inside the buildBulk function you will read each line from the file. For each line of the file, create a ReviewInfo structure instance out of the line you just read from the file. Call the insertItem function with the instance of ReviewInfo.

- `int getNumCollision()`:Returns the numCollision.

- `void printTable()` : It will print the chains of the tables along with bucket indices. It will not print any priority queue information
    a. Basically iterate over the table
    b. For each entry in the table, traverse the linked list and print the movieName of the node.
    c. An example is given here with a table of size 5 and the provided test.txt. For more detail example please refer to the run example at the end of the document.

```
0|NULL
1|Avengers: Endgame-->Pair Jars-suck-->Jurassic -Park-->Jura-ssic Park-->NULL
2|aladin-->NULL
3|roam inn-->iron man-->NULL
4|NULL
```

- `void deleteEntry(string movieName, string user)` : Delete a movie record with movieName and user as specified in the argument.
    a. First search in hashTable for the movieName.
    b. If the search returns null, cout `"no record found".`
    c. If search returns a valid node, delete the record with movieName and user from the priority queue associated with the concerned node. Use the `deleteKey` function of the priority queue.
    d. If this deletion makes the priority queue of the concerned node empty delete the node. Handle the boundary cases like deletion of head accordingly.

Driver code

You should pass two command line arguments-

1. Initial file from which hashtable will be populated. The file is ; separated file and the format of a line is-
   movieName;comment;user;time
   For example

iron man;The first Iron Man film was intense, right from the get go.;A. Mark Wilburn;0317

2. Size of the hash table

Driver code should be a menu driven function. It should present the following menu options to the user and it must not stop until the user chooses to exit.

1: Build the datastructure (call it only once)

2: Add a movie review tweet

3: Retrieve most recent review for a movie

4: Delete a review

5: Print reviews for a movie

6: Get number of collision

7: Print the table

8: Exit

The displayMenu() function is given as part of the starter code. Kindly see the following for more details on the menu options-

1: Build the datastructure (call it only once): Build the hash table from the filename passed as command line arguments. Call the `buildBulk` of hash table. Note during the one execution this option can be chosen once only. Therefore any subsequent attempt of choosing option 1 should not be allowed.

```
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print review for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>1
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print review for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>1
Already built.
```

2: Add a movie review tweet. This will prompt for information about the movie. User will input moviename, user, comment and time accordingly. Then create an instance of `ReviewInfo` and will insert it to the hash table. Refer to the example at the end of this document.

```
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print review for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>2
Moviename: ironman
Username: taylor
Comment: this is the best movie of MCU
Time: 1011
```

3. Retrieve the most recent review for a movie: First ask the user to provide the name of the movie. Then look for that movie in the hashtable. If found, print the most recent review information from the priority queue. Use the search function of the hash table and peek function of the priority queue. If no such movie exists in the hash table print `"no record found".`

```
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print review for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>3
movie name:iron man
retrieved result
Movie: iron man
User:Mark
Comment:This is the movie that started it all! and my first Mcu movie.
Time:2008
```

4. Delete a review: Prompt for the movie name and the user name. Search for the movie in the hash table. If found, delete the `ReviewInfo` with the user from the priority queue of the hashtable node. Use the function **`deleteEntry`** of hash table.

```
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print review for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>4
movie name:iron man
user name:Mark
```

5. Print reviews for a movie: Ask for the movieName and read the movieName from the consol. Then search for the movie in the hash table. If found, print the review information from the associated priority queue.

```
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print review for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>5
movie name:iron man
Movie: iron man
        User: Jerry
        Comment: Great acting by RDJ.
        Time: 1703
        =====
        User: Carl
        Comment: Since Marvel Comics and CGI started their mystical marriage in the late 1990s, we've had superheroes galore.
        Time: 1543
        =====
        User: McKein Mull
        Comment: What a hell of a way to kick off the character of Tony Stark aka Iron-Man as well as the Marvel Cinematic Universe, little di
d we know it at the time of this movie's release but it had quite a big burden riding on its success.
        Time: 1011
        =====
        User: A. Mark Wilburn
        Comment: The first Iron Man film was intense, right from the get go.
        Time: 317
        =====
        User: Shaheed Gonyatta
        Comment: In my eyes Iron Man is what could be called a perfect hero's journey.
        Time: 1236
        =====
```

If the movie is not found in the hash table, then print `"no record found".`

6. Get number of collision: Print the number of collisions

7. Print the table: Call the `printTable` of the hash table.

8. Exit: exit the execution

An example run is given here with respect to the test.txt file-

./a.out test.txt 5

```
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print reviews for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>1
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print reviews for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>7
0|NULL
1|Avengers: Endgame-->Pair Jars-suck-->Jurassic -Park-->Jura-ssic
Park-->NULL
2|aladin-->NULL
```

```
3|roam inn-->iron man-->NULL
4|NULL
------------------
Give your choice >>6
Number of collision:4
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print reviews for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>5
movie name:aladin
Movie: aladin
        User: Ana Carone
        Comment: Ok. Don't you dare compare to the original because the story got changed in many many ways and you will see as soon as the movie starts.
        Time: 132
        =====
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print reviews for a movie
6: Get number of collision
7: Print the table
8: Exit
```

```
------------------
Give your choice >>4
movie name:aladin
user name:Ana Carone
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print reviews for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>7
0|NULL
1|Avengers: Endgame-->Pair Jars-suck-->Jurassic -Park-->Jura-ssic
Park-->NULL
2|NULL
3|roam inn-->iron man-->NULL
4|NULL
------------------
1: Build the datastructure (call it only once)
2: Add a movie review tweet
3: Retrieve most recent review for a movie
4: Delete a review
5: Print reviews for a movie
6: Get number of collision
7: Print the table
8: Exit
------------------
Give your choice >>3
movie name:iron man
```

retrieved result

Movie: iron man

User:Mark

Comment:This is the movie that started it all! and my first Mcu movie.

Time:2008