Southern New Hampshire University

Chris Trimmer

CS-300-T1159 DSA: Analysis and Design

Milestone 4-3: Hash Table Data Structure Pseudocode

22EW1 – 09/23/2022

Southern
New Hampshire
University

The purpose of this document is to provide a pseudocode and algorithm analysis of code for a course planner that we are going to design for ABC University (ABCU). The pseudocode will consist of functions that pertain to file parsing, creating objects and storing them in a hash table data structure, searching the data structure, and printing the data. Where applicable, a runtime analysis will be documented for a function.

The main object used in the assignment is a Course class. Each course will contain a string value for its id, a string value for its title, and a list of Course objects that will store prerequisites courses. I use a list to store prerequisites because a course can have more than one prerequisite and a list is suitable for dynamic insertion. The hash table will represent the schedule of class objects. At a high-level representation, a schedule will contain multiple courses, and each course in the schedule can have zero or more prerequisite courses.

File Opening and Parsing

```
Void LoadCourses(string filepath, HashTable* hashTable) {

  Create infile ifstream object
  Use infile object to open the filepath

  If the infile object returns null
    return to caller immediately

  Create string object, line, to hold a line read from the file
  Create a char object, delim, which is a “,” (comma), to use as delimiter when reading each line
  Create a string object, word, to hold each word in the line
  Create a vector of strings, courseLine, that will hold each word

  Loop through infile object, and store each line in the line object
    Pass each line as an object to stringstream object, fullLine and parse using delim

    Loop through stringstream and push back each word in the line to temp vector

      // check file format
      If the line contains less than two words, then this is an incomplete record
        Print output to user regarding invalid file format
        Return to caller

      // verify that prereqs are valid
      For each word after the first two
        If the word is in not the course list
          Skip the word
    // Creating and storing the object is covered in AddCourse function
      Call AddCourse and pass vector of valid words

    Clear the courseLine vector before starting the next loop

} // end LoadSchedule
```

Code	Line Cost	Execution Times	Total Cost
Create ifstream object	1	1	1
Open filepath	1	1	1
if infile is not null	2	1	2
Create 5 local variables	5	1	5
for each line	1	n	n
create sstream object	1	1	1
for each sstream object	1	n	n
if courseLine less than 2	2	1	2
return to caller	1	1	1
for each word after 2	1	n	n
if the word not in course list	1	1	1
skip the word (continue)	1	1	1
add word to vector	1	1	1
call AddCourse	4n + 32	1	4n + 32
		Total Cost:	7n + 48
		Runtime	O(n)

Creating and Storing Objects

```
Void AddCourse(vector<string>& line, vector<Course>& courses) {

  Instantiate Course object

  For each word in line
    Set course id to line[0]
    Set course title to line[1]

  For each additional word in line
    Push back word into the prereq list stored in the Course object

  // the course now has all its data
  Call insert function of hash table class passing the Course object

}
```

Code	Line Cost	Execution Times	Total Cost
Instantiate blank course object	1	1	1
for each word in lines	1	n	n
set course id to line[0]	1	1	1
set course title to line[1]	1	1	1
for each additional word in lines	1	n	n
push back word to prereq vector	1	1	1
call insert function to pass course object	n + 14	1	n + 14
		Total Cost:	3n + 18
		Runtime	O(n)

Insertion function

```
Void HashTable::Insert(Bid bid) {
  Assign local bidKey variable by calling hash function

  Create pointer (curr) to the node at the index of the hashed bidKey

  If  curr is equal to nullptr
    Assign curr to the node at this index
  Else
    If the old key at this node is (UINT_MAX)
      Set curr key to bidKey
      Set curr bid to bid
      Set curr next to nullptr

    Else
      Loop through the list until we get to end
      Set next node of curr to be the new node
}
```

Code	Line Cost	Execution Times	Total Cost
set local var to result of hash	2	1	2

create pointer to index node	1	1	1
if curr is equal to nullptr	2	1	2
assign curr to the node index	1	1	1
else if old key is UINT_MAX	2	1	2
set curr key to bidKey	1	1	1
set curr bid to bid	1	1	1
set curr next to nullptr	1	1	1
else loop through list till end	1	n	n
set next of curr to new node	1	1	1
return key mod tableSize	2	1	2
Total Cost:			n + 14
Runtime			O(n)

Hash function

```
Unsigned int HashTable::hash(int key) {
  Return key % tableSize;
}
```

Code	Line Cost	Execution Times	Total Cost
return key mod tableSize	2	1	2
Total Cost:			2
Runtime			O(1)

Get number of prerequisites

```
Int GetNumberOfPrereqs(string key) {

  Initialize sum variable to 0

  Set curr to head node of prerequisites list

  While curr->next not equal to nullptr
    Increment sum

  Return sum
}
```

Code	Line Cost	Execution Times	Total Cost
Initialize sum variable to 0	1	1	1
set curr to head of prereq list	1	n	n
while curr->next not nullptr	1	n	n
increment sum	1	1	1
return sum	1	1	1
		Total Cost:	2n + 3
		Runtime	O(n)

Convert string to uppercase

```
string ConverToUpper(string key) {
  for each letter in key
    transform letter to uppercase

  return key
}
```

Code	Line Cost	Execution Times	Total Cost
for each letter in key	1	n	n
tranform letter to uppercase	2	1	2
return key	1	1	1
		Total Cost:	n + 3
		Runtime	O(n)

Remove a Course

```
Void HashTable::Delete(string key) {

  Call convert to uppercase with the key

  Create index variable by calling hash with the key

  Create iterator to loop through the table
  For each index of the table
    If the key matches the course id
      break from for loop (we now have the iterator)

  if the iterator is not null
    erase the Course that is at iterators position within the index list
}
```

Code	Line Cost	Execution Times	Total Cost
call convert to upper on key	n + 3	1	n + 3
create index variable by calling hash function	3	1	3
create iterator index	1	1	1
for each index of table	1	n	n
if key matches course id	2	1	2
break	1	1	1
if iterator not null	1	1	1
erase course at that iter	2	1	2
		Total Cost:	2n + 13
		Runtime	O(n)

Search for a Course

```
Void Search(string key) {

  Convert incoming key to uppercase
  Hash the key, and assign value to a variable named Index
  For each index in the table
    If the index->id is equal to the key, then we found a match
      Display the course
      For each course in the list of prepreqs
        Display the preprequisite information
}
```

Code	Line Cost	Execution Times	Total Cost
convert key to uppercase	n + 3	1	n + 3
index = hash(key)	1	1	1
for each course	1	n	n
if courseId equal to key	1	1	1
Print course information	1	1	1
For each prereq	1	n	n
Print prereq information	1	1	1
		Total Cost:	3n + 7

Runtime	O(n)

Print Courses

```
Void PrintHashTable() {

  For each index in table
    Print the course at this index
      For each prereq in a course
        Print the prereq

}
```

Code	Line Cost	Execution Times	Total Cost
For each course	1	n	n
Print course information	1	1	1
For each prereq	1	n	n
Print prereq information	1	1	1
		Total Cost:	2n + 2
		Runtime	O(n)