CS-300 22EW2

Project One

Jared Smith

12/4/22

**Psuedocode for BST:**

DECLARATIONS:

DEFINE struct: course with four string parameters: course ID, course name, prerequisite one and prerequisite two, and two pointers, one left and one right

DEFINE a default constructor and an overloaded constructor with course ID and course name. Both should initialize the pointers as nullptr

DEFINE methods for adding prerequisite one and two

**PARSING THE FILE:**

OPEN file

WHILE not at the end of the file

READ and SET course ID

IF there is a second item on the list

READ and SET course name

ELSE

THROW error, move to next item

DECLARE new course struct, using overloaded constructor

IF there is a third item on the list

CALL search() for the course ID

IF course ID is found

DEFINE prerequisite one as element in CSV

ELSE

THROW error, continue to next item

IF there is a fourth item on the list

CALL search() for the course ID

IF course ID is found

DEFINE prerequisite two as element in CSV

ELSE

THROW error, continue to next item

CALL AddNode() with course node object

**AddNode()**

DECLARE current node, set to root

IF course id is less than our current node’s

IF current node’s left pointer is null

Set current node’s left pointer to new Node object

ELSE

SET current node to its left pointer

CALL AddNode, feeding it the new current node and same course id

ELSE

IF current node’s right branch pointer is null

SET current node’s right branch pointer to new node object

ELSE

SET current node to its right pointer

CALL AddNode, feeding it the new current node and same course id

RETURN

**InOder()** – For printing in order….

IF node fed to function is not NULL

CALL InOrder with current Node’s left pointer

PRINT course information

CALL InOrder with current Node's right pointer

RETURN

**Search()**

SET current node to root

WHILE current node is not NULL

IF current node is a match

RETURN current node

ELSE IF our current node’s bid ID is less than the bid id we’re searching for

SET current node to current node’s right pointer

ELSE

SET current node to current node’s left pointer

IF no matching bid is found

RETURN empty node

**Psuedocode for Hash Table:**

DECLARATIONS:

DEFINE struct: course with four string parameters: course ID, course name, prerequisite one and prerequisite two, an uint param for hash key and a pointer for next.

DEFINE a default constructor and an overloaded constructor with course ID and course name. Both should initialize the key as UINT\_MAX and the pointer as nullptr

DEFINE methods for adding prerequisite one and two

DECLARE a vector of 100 empty course object

DEFINE a hash method which takes the course ID modulo the vector length and returns the hash value

**PARSING THE FILE:**

OPEN file

WHILE not at the end of the file

READ and SET course ID

IF there is a second item on the list

READ and SET course name

ELSE

THROW error, move to next item

DECLARE new course struct, using overloaded constructor

IF there is a third item on the list

SEARCH course vector for hash

IF prerequisite ID at current hash location is the current Course element’s course ID

RETURN TRUE

ELSE

WHIILE check node’s next pointer is not nullptr

CHECK each next until we find the correct course ID

RETURN TRUE

IF no correct course id is found

RETURN FALSE

IF prerequisite ID is a course ID in course vector

DEFINE prerequisite one as element in CSV

ELSE

THROW error

IF there is a fourth item on the list

SEARCH course vector for hash

IF prerequisite ID at current hash location is the current Course element’s course ID

RETURN TRUE

ELSE

WHIILE check node’s next pointer is not nullptr

CHECK each next until we find the correct course ID

RETURN TRUE

IF no correct course id is found

RETURN FALSE

IF prerequisite ID is a course ID in course vector

DEFINE prerequisite two as element in CSV

ELSE

THROW error

HASH course id

IF node of vector at course’s hash key is not UINT\_MAX

WHILE check node’s next pointer is not NULL

CHECK next node

APPEND new node at end of bucket linked list

ELSE

SET node at hash key to current course node

**SEARCHING THE VECTOR AND PRINTING RESULT:**

GET course ID from user

HASH course ID

IF course id matches the course id at the hash key index

RETURN that object

ELSE

WHILE current course’s next pointer is not null

CHECK next object in linked list for matching course ID

IF matching course ID found, return next object

IF no matching course ID found

PRINT no matching course found

**PRINT ALL:**

FOR each item in course vector

IF current object key is not UINT\_MAX

PRINT course object

WHILE pointer is not nullptr

SET current object as current’s next pointer

PRINT current object

SET current object as next in the vector

**Psuedocode for Vector:**

DECLARATIONS:

DEFINE struct: course with four string parameters: course ID, course name, prerequisite one and prerequisite two

DEFINE a default constructor and an overloaded constructor with course ID and course name

DEFINE methods for adding prerequisite one and two

DECLARE an empty vector of courses

**PARSING THE FILE:**

OPEN file

WHILE not at the end of the file

READ and SET course ID

IF there is a second item on the list

READ and SET course name

ELSE

THROW error, move to next item

DECLARE new course struct, using overloaded constructor

IF there is a third item on the list

CHECK course vector for ID

FOR all courses in course vector

IF prerequisite ID is the current Course element’s course ID

RETURN TRUE

ELSE

Increment and check next

RETURN FALSE

IF prerequisite ID is a course ID in course vector

DEFINE prerequisite one as element in CSV

ELSE

THROW error

IF there is a fourth item on the list

CHECK course vector for ID

FOR all courses in course vector

IF prerequisite ID is the current Course element’s course ID

RETURN TRUE

ELSE

Increment and check next

RETURN FALSE

IF prerequisite ID is a course ID in course vector

DEFINE prerequisite two as element in CSV

ELSE

THROW error

**SEARCHING THE VECTOR AND PRINTING RESULT:**

GET key from user

FOR all courses in course vector

IF course number matches key

PRINT course information

IF prerequisite one is not null

PRINT prerequisite one

IF prerequisite two is not null

PRINT prerequisite two

**PRINT ALL:**

FOR each item in the vector

PRINT current object

**Psuedocode for Menu:**

WHILE choice is not ‘9’

SWITCH CASE:

CASE 1:   
 CALL CSV parser

CASE 2:

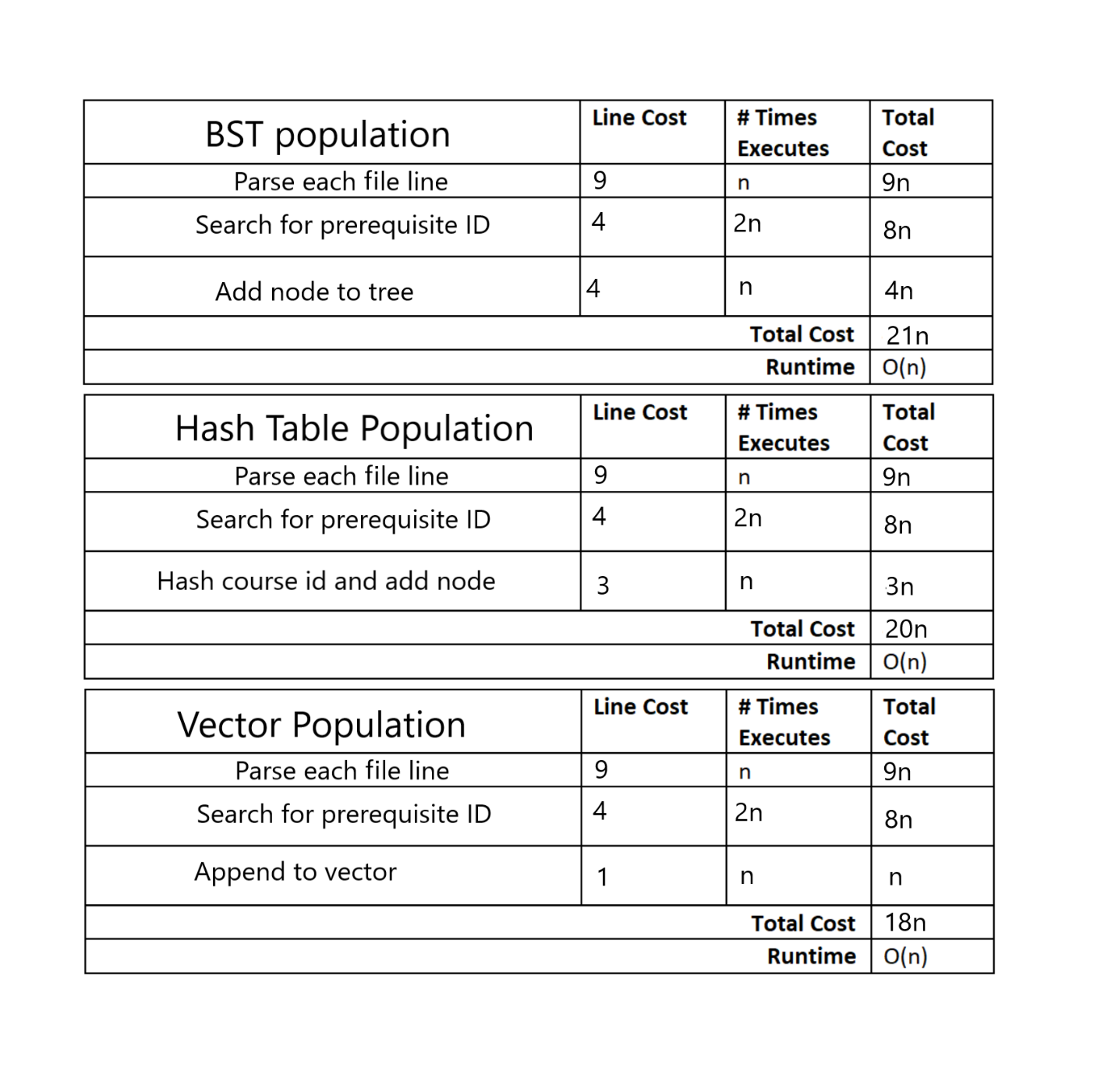
CALL printall function

CASE 3:

CALL Search and Print

CASE 9:

EXIT

**Evaluations:**

**Analysis:**

BST

A BST is a data structure that uses nodes with two pointers, organized by placing each key that is less than its root to the left and vice versa. The benefit of using this data structure is quick access for data structures of limited size, requiring fewer average comparisons than using a traditional vector. The downside of using this data structure is it can become cumbersome with large amounts of data. Using a BST can also be problematic when our tree becomes lopsided, meaning it essentially just becomes a linked list.

Hash Table

A hash table is a data structure that has the fastest access time, given you use an effective hash function. This is done by creating a function that takes unique key identifiers and using a uniform function to translate those keys into indices on another list of some kind. This can be a very efficient structure time wise, but can be inefficient storage wise, with indices in a vector going unused.

Vector

The vector is the simplest data structure used for this project, simply appending another memory location to the end of the last until all nodes are compiled into a list. The advantage of this data structure is primarily the storage space and simplicity. This advantage comes at the cost of accessibility, as the list can be difficult to access, averaging list length divided by two comparisons.

**Recommendations:**

For the purposes of this project, I recommend using a BST. The course ID’s are all unique and spread fairly evenly. Given the limited size of the data set, neither memory storage nor quick access should be much of an issue, so I feel the BST is a good middle ground that covers all the bases we need. Additionally, our worst case O value for each data structure is the same, so there will be no major consideration in that regard.