

**Project one: Algorithmic Design**

**Hamza Malik**

**TABLE OF CONTENT**

[PSEUDOCODE- LinkedList 3](#_Toc127181320)

[Advantages and Disadvantages of Linked List: 7](#_Toc127181321)

[PSEUDOCODE-Hash Table 7](#_Toc127181322)

[Advantages and Disadvantages of Hash Table: 10](#_Toc127181323)

[PSEUDOCODE-Binary Search Tree 11](#_Toc127181324)

[Advantages and Disadvantages of Binary Search Tree: 14](#_Toc127181325)

[PSEUDOCODE-Menu: 15](#_Toc127181326)

[Recommendation: 17](#_Toc127181327)

# PSEUDOCODE- LinkedList

**Struct Course**

Declare courseName as a string

Declare courseNubmer as a string

Declare prerequisite as string

Then create the Vector<Course> loadCourses(string csvPath)

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| Declare courseName as a string | 1 | 1 | 1 |
| **Declare courseNubmer as a string** | 1 | 1 | 1 |
| **Declare prerequisite as string** | 1 | 1 | 1 |
| **Then create the Vector<Course> loadCourses(string csvPath)** | 1 | 1 | 1 |
| **Total Cost** | | | 4 |
| **Runtime** | | | O(1) |

**To read the InputFile or File**

**ReadFile(filename, Vector)**

First Initialize the fstream filestream

Then Declare the string linestrieam to get contents of each line from the input file

Then declare string tempInfo to hold each parameter of the input file

Then declare the variable int totalCount to verify count of parameters

Then declare the variable string counter

Open file with fileStream

Fill linestream with each line

Create new course

For each line with the ‘,’ csv file separated by this //here we have comma separated //file.

Set total count equal to count of ‘,’

If totalCount is less than 2

Print “missing a parameter”

Return

Else if totalCount is greater than equal to 2

Set formatIsCorrect to the true.

While loop to iterate through each line’s contents

If formatIsCorrect to true

Then set contents equal to the token for string

Set counter = to ‘,’ ocunt

If couner == 1

Then set courseNumber = tempInfo

Else if counter == 2

Then set courseName = tempInfo

Else if counter >=3

Then set prerequisite == tempInfo

Add the course to the vector (LinkedList using for this assignment)

Then clear the lineStream

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| First Initialize the fstream filestream | 1 | 1 | 1 |
| Then Declare the string linestrieam to get contents of each line from the input file | 1 | 1 | 1 |
| **Then declare string tempInfo to hold each parameter of the input file** | 1 | 1 | 1 |
| **Then declare the variable int totalCount to verify count of parameters** | 1 | 1 | 1 |
| **Then declare the variable string counter** | 1 | 1 | 1 |
| **Open file with fileStream** | 1 | 1 | 1 |
| **Fill linestream with each line** | 1 | 1 | 1 |
| **Create new course** | 1 | 1 | 1 |
| For each line with the ‘,’ csv file separated by this | 1 | N | N |
| **Set total count equal to count of ‘,’** | 1 | N | N |
| If totalCount is less than 2 | 1 | N | N |
| Print “missing a parameter” | 1 | N | N |
| **Return** | 1 | N | N |
| **Else if totalCount is greater than equal to 2** | 1 | N | N |
| **Set formatIsCorrect to the true** | 1 | N | N |
| **While loop to iterate through each line’s contents** | 1 | N | N |
| **If formatIsCorrect to true** | 1 | N | N |
| **Then set contents equal to the token for string** | 1 | N | N |
| **Set counter = to ‘,’ ocunt** | 1 | N | N |
| **If couner == 1** | 1 | N | N |
| **Then set courseNumber = tempInfo** | 1 | N | N |
| **Else if counter == 2** | 1 | N | N |
| **Then set courseName = tempInfo** | 1 | N | N |
| **Else if counter >=3** | 1 | N | N |
| **Then set prerequisite == tempInfo** | 1 | N | N |
| **Add the course to the vector (LinkedList using for this assignment)** | 1 | 1 | 1 |
| **Then clear the lineStream** | 1 | 1 | 1 |
| **Total Cost** | | | 17N + 10 |
| **Runtime** | | | O(N) |

**Prerequisites**

**PrintCourseInformation**

Node\* temp = head;

for all courses until it found the nullpointer

check if the course is the same as courseNumber

print out the course information like the name of the course so you can use any delimiter for that.

for each prerequisite of the course

print the prerequisite course information

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Node\* temp = head;** | 1 | 1 | 1 |
| **for all courses until it found the nullpointer** | 1 | N | n |
| **check if the course is the same as courseNumber** | 1 | n | N |
| **print out the course information like the name of the course so you can use any delimiter for that.** | 1 | n | n |
| **for each prerequisite of the course** | 1 | n | n |
| **print the prerequisite course information** |  | n | n |
| **Total Cost** | | | 2\*n^2 + 3\*n + 1 |
| **Runtime** | | | O(n^2) |

## Advantages and Disadvantages of Linked List:

Using linked lists has a few advantages, including:

* Nodes can be added and removed with ease.
* Most computer languages include built-in mechanisms for creating linked lists.

On the other hand, linked lists have some drawbacks, including:

* Nodes must always be visited in a sequential manner, which takes time.
* Linked lists demand additional memory since they employ pointers.

# PSEUDOCODE-Hash Table

1. **Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors**

Opened the file "filename/filepath"

For every file line:

line breaks with commas

Verify if that there are at least two parameters.

Verify if the file's second argument to see if it's listed as a course.

Raise an exception if any mistakes are discovered.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Opened the file "filename/filepath"** | 1 | 1 | 1 |
| **For every file line:** | 1 | N | n |
| **line breaks with commas** | 1 | n | N |
| **Verify if that there are at least two parameters.** | 1 | n | n |
| **Verify if the file's second argument to see if it's listed as a course.** | 1 | n | n |
| **Raise an exception if any mistakes are discovered.** |  | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

1. **Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

First create the an empty hash table which store the bids

Then for each line in file to read the file

Then separate the each line by comma

Then create a new course object

Declare courseTitle as a string

Declare courseNubmer as a string

Declare prerequisite as string

Set the first parameter as courseNumber, second as courseTitle and third is prerequisites

Then insert the course object into the hashtable

Using the courseNumber as the key of the hashtable for each course.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **First create the an empty hash table which store the bids** | 1 | 1 | 1 |
| **Then for each line in file to read the file** | 1 | N | n |
| **Then separate the each line by comma** | 1 | n | N |
| **Then create a new course object** | 1 | n | n |
| **Declare courseTitle as a string** | 1 | n | n |
| **Declare courseNubmer as a string** | 1 | n | n |
| **Declare prerequisite as string** | 1 | 1 | 1 |
| **Set the first parameter as courseNumber, second as courseTitle and third is prerequisites** | 1 | 1 | 1 |
| **Then insert the course object into the hashtable** | 1 | n | N |
| **Using the courseNumber as the key of the hashtable for each course.** | 1 | 1 | 1 |
| **Total Cost** | | | 6n + 4 |
| **Runtime** | | | O(n) |

1. **Design pseudocode that will print out course information and prerequisites.**

Then we need to loop through the each key which is courseNumber in this case in Hashtable.

To retrieve the courseObject against the each courseNumber from the hashTable.

Then print the information about the course

Like courseNumber and the courseTitle

For each course And also print the lists of the prerequisites

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Then we need to loop through the each key which is courseNumber in this case in Hashtable.** | 1 | N | n |
| **To retrieve the courseObject against the each courseNumber from the hashTable.** | 1 | N | n |
| **Then print the information about the course** | 1 | n | N |
| **Like courseNumber and the courseTitle** | 1 | n | n |
| **Verify if the file's second argument to see if it's listed as a course.** | 1 | n | n |
| **For each course And also print the lists of the prerequisites** |  | n | n |
| **Total Cost** | | | 6n |
| **Runtime** | | | O(n) |

## Advantages and Disadvantages of Hash Table:

As a result, the following are some key advantages of using hash tables:

* Operations like insert, remove, and search can be completed in O(1) time, which is extremely quick.
* Large volumes of data can be stored in hash tables.

On the other hand, some drawbacks of using hash tables are:

* Hash functions frequently result in duplicate keys, which lead to collisions—storage issues—with data values.
* Good hash functions are expensive and challenging to develop since they yield unique keys.

# PSEUDOCODE-Binary Search Tree

1. **Design pseudocode to define how the program opens the file, reads the data from the file, parses each line, and checks for file format errors**

Opened the file "filename/filepath"

For every file line:

line breaks with commas

Verify if that there are at least two parameters.

Verify if the file's second argument to see if it's listed as a course.

Raise an exception if any mistakes are discovered.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Opened the file "filename/filepath"** | 1 | 1 | 1 |
| **For every file line:** | 1 | N | n |
| **line breaks with commas** | 1 | n | N |
| **Verify if that there are at least two parameters.** | 1 | n | n |
| **Verify if the file's second argument to see if it's listed as a course.** | 1 | n | n |
| **Raise an exception if any mistakes are discovered.** |  | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

1. **Design pseudocode to show how to create course objects and store them in the appropriate data structure.**

Create a root node for the tree Create a node with no parent and no children

Load the data into the tree

For each course in the curriculum:

Create a node for the course

Set the parent of the node to the root node

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Create a root node for the tree Create a node with no parent and no children** | 1 | 1 | 1 |
| **Load the data into the tree** | 1 | N | n |
| **For each course in the curriculum:** | 1 | n | N |
| **Create a node for the course** | 1 | n | n |
| **Set the parent of the node to the root node** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

**How to add node to the Binary search tree using the course Number.**

if node is larger then add to left

// if no left node

// this node becomes left

// else recurse down the left node

// else

// if no right node

// this node becomes right

//else

// recurse down the left node

And for traversing the tree we have different method for that.

preOrder, InOrder and, postOrder.

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **if node is larger then add to left** | 1 | 1 | 1 |
| if no left node | 1 | N | n |
| **this node becomes left** | 1 | n | N |
| **else recurse down the left node** |  |  |  |
| **else** | 1 | n | n |
| **if no right node** | 1 | n | n |
| **this node becomes right** | 1 | n | N |
| else | 1 | n | n |
| recurse down the left node | 1 | n | n |
| **Total Cost** | | | 7n + 1 |
| **Runtime** | | | O(n) |

1. **Design pseudocode that will print out course information and prerequisites.**

Then we need to loop through or traverse through each courseObject using different traversing techniques which I mentioned above in BinarySearch tree

To retrieve the courseObject against the each courseNumber from the binaryTree.

Then print the information about the course

Like courseNumber and the courseTitle

For each course And also print the lists of the prerequisites

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Then we need to loop through or traverse through each courseObject using different** | 1 | 1 | 1 |
| traversing techniques which I mentioned above in BinarySearch tree | 1 | N | n |
| **To retrieve the courseObject against the each courseNumber from the binaryTree.** | 1 | n | N |
| **Then print the information about the course** | 1 | n | n |
| **Like courseNumber and the courseTitle** | 1 | n | n |
| **For each course And also print the lists of the prerequisites** | 1 | n | n |
| **Total Cost** | | | 5n + 1 |
| **Runtime** | | | O(n) |

## Advantages and Disadvantages of Binary Search Tree:

The simplicity of binary trees is its key benefit. A straightforward structure for managing and organising data is included in binary trees. Additionally, binary trees have the following advantages:

* They can be applied to reflect patterns in data.
* They are capable of holding any quantity of data values.

However, there are some drawbacks to adopting binary trees, including:

* Node deletion is a difficult process.
* The height of the tree affects insertion, deletion, and search processes.

# PSEUDOCODE-Menu:

int choice = 0;

while (choice != 9)

cout << "Menu:" << endl;

cout << " 1. Load Bids" << endl;

cout << " 2. Display All Bids" << endl;

cout << " 3. Find Bid" << endl;

cout << " 4. Remove Bid" << endl;

cout << " 9. Exit" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice)

case 1:

case 1 execution code

break;

case 2:

case 2 execution code

break;

case 3:

case 3 execution code

break;

case 4:

case 1 execution code

break;

end switch

end while

cout << "Good bye." << endl;

return 0;

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **int choice = 0;** | 1 | 1 | 1 |
| while (choice != 9) | 1 | N | n |
| **cout << "Menu:" << endl;** | 1 | N | N |
| **cout << " 1. Load Bids" << endl;** | 1 | N | n |
| **cout << " 2. Display All Bids" << endl;** | 1 | N | n |
| **cout << " 3. Find Bid" << endl;** | 1 | N | N |
| **cout << " 4. Remove Bid" << endl;** | 1 | N | n |
| **cout << " 9. Exit" << endl;** | 1 | N | n |
| **cout << "Enter choice: ";** | 1 | N | N |
| **cin >> choice;** | 1 | N | n |
| **switch (choice)** | 1 | N | N |
| **case 1:** | 1 | N | n |
| **Case 1 execution code** | 1 | N | n |
| **Break;** | 1 | N | n |
| **Case 2:** | 1 | N | n |
| **Case 2 execution code** | 1 | N | N |
| **Break;** | 1 | N | n |
| **Case 3:** | 1 | N | n |
| **Case 3 execution code** | 1 | N | n |
| **Break;** | 1 | N | n |
| **Case 4:** | 1 | N | n |
| **Case 4 execution code** | 1 | N | N |
| **Break;** | 1 | N | n |
| **End switch** | 1 | N | N |
| **End while** | 1 | 1 | 1 |
| **Print”Good Bye”** | 1 | 1 | 1 |
| **Return 0;** | 1 | 1 | 1 |
| **Total Cost** | | | 22n + 4 |
| **Runtime** | | | O(n) |

# Recommendation:

* The insertion, deletion and search in Binary Search tree is done in O(logn)
* In linked list it took the for search is done in O(n) and insertion and deletion is done in O(1)
* In Hash table it took the for insertion, search and deletion is O(1).

I would recommend to use the Binary Search Tree because it took the O(logn) because the data is already sorted in Binary search tree. That is why it is more easy or fast, I would also like to use the Binary search tree.