# CS 300 Pseudocode Module 6

**I. VECTOR PSEUDOCODE**

**1. Text file manipulation**

IFSTREAM file(file to read.csv)

IF !file.is\_open()

PRINT error

RETURN 0

END IF

String line

WHILE (!file to read.csv.eof())

Getline(file, line)

Stringsteam ss(line)

Vector<Course> course

WHILE(getline(ss,course,’,’)

courseInfo.push\_back(course)

END WHILE

END WHILE

File.close()

**2. Creating course objects and store them in the appropriate data structure**

Vector<Course> course

FOR each int I = 0, I < file.size(); increment I

course.push\_back(courseName)

course.push\_back(courseTitle)

IF(course.coursePrereq.back() == true

Course.push\_back(coursePrereq)

END IF

END FOR

**3. Print out course information and prerequisite**

PRINT courseTitle and courseName

INPUT courseTitle to search

IF vector course is empty = false

FOR string course : courseTitle

IF course.coursePrerqe = courseTitle to search

PRINT course Prereq

END IF

END FOR

END IF

## 4. Runtime Analysis

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **Print courseTitle and courseName** | 1 | 1 | 1 |
| **Input courseTitle to search** | 1 | 1 | 1 |
| **If course vector is not empty** | 1 | 1 | 1 |
| **for each prerequisite of the course** | 1 | n | n |
| **If courseTitle to search match with coursePrereq in vector** | 1 | n | n |
| **Print course prerequisite** | 1 | 1 | 1 |
| **Total Cost** | | | 2n + 4 |
| **Runtime** | | | O(n) |

**II. HASH TABLE**

**1.Text file manipulation**

IFSTREAM file(file to read.csv)

IF !file.is\_open()

PRINT error

RETURN 0

END IF

String line

WHILE (!file to read.csv.eof())

Getline(file, line)

Stringsteam ss(line)

Vector<Course> course

WHILE(getline(ss,course,’,’)

courseInfo.push\_back(course)

END WHILE

END WHILE

File.close()

**2. Creating course objects and store them in the appropriate data structure**

CREATE structure to hold course info

courseNumber

courseName

coursePrereq

CREATE hashTable class

SET private:

CREATE structure Node to hold courses

Course

Key

Node\* next

Constructor

Vector<Node>nodes

Unsigned int tableSize

Unsigned int hash(string key)

SET public:

Hashtable()

HashTable(unsigned int size)

Virtual ~Hashtable()

Void insert(Course course)

Void printAll

Void remove(string courseNumber)

Void search(string courseNumber)

**3. Print out course information and prerequisite**

CREATE a node to key

SET a bool to false

CREATE a string courseName

WHILE node != NULL

IF node->courseNumber == courseNumber

Bool = true

courseName = node->courseName;

END IF

Node = node->next

END WHILE

IF true

PRINT coursePrereq

ELSE

PRINT “NOT FOUND”

**4. Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **While node is not null** | 1 | n | n |
| **If node->courseTitle = courseTitle from input** | 1 | n | n |
| **Set bool to true** | 1 | 1 | 1 |
| **courseTitle from input = node->courseTile** | 1 | 1 | 1 |
| **Point to next node** | 1 | n | n |
| **If condition is true** | 1 | n | n |
| **Print coursePrereq** | 1 | 1 | 1 |
| **Total Cost** | | | 4n + 3 |
| **Runtime** | | | O(n) |

**III. Binary Tree**

**1.Text file manipulation**

IFSTREAM file(file to read.csv)

IF !file.is\_open()

PRINT error

RETURN 0

END IF

String line

WHILE (!file to read.csv.eof())

Getline(file, line)

Stringsteam ss(line)

Vector<Course> course

WHILE(getline(ss,course,’,’)

courseInfo.push\_back(course)

END WHILE

END WHILE

File.close()

**2. Creating course objects and store them in the appropriate data structure**

CREATE a struct to hold course

String courseName

String courseTitle

Vector<String> coursePrereq

CREATE a struct to hold Binary Search Tree node

Course course

Node left

Node right

Define constructor

CREATE a Binary Search Tree class

SET public:

Binary Tree root node

FUNC addNode

FUNC inORder

SET private:

Default constructor

Destructor

FUNC search

FUNC printCourse

FUNC InOrder

FUNC loadCourse

**3. Print out course information and prerequisite**

WHILE currBstNode is not NULL

IF curBstNode -> course.courseTitle equal courseTitle

PRINT courseTitle and courseName

IF currBstNode->course.coursePrereq.empty() = false

FOR each string value in course prerequisite

IF string value match courseTitle

PRINT coursePrereq

END IF

END FOR

END IF

END IF

END WHILE

**4. Runtime Analysis**

| **Code** | **Line Cost** | **# Times Executes** | **Total Cost** |
| --- | --- | --- | --- |
| **While node is not null** | 1 | n | n |
| **If node->courseTitle match courseTitle** | 1 | n | n |
| **Print courseTile and courseName** | 1 | 1 | 1 |
| **If node->coursePrereq is not empty** | 1 | n | n |
| **For each string value in coursePrereq** | 1 | n | n |
| **If string value match courseTitle** | 1 | n | n |
| **Print coursePrereq** | 1 | 1 | 1 |
| **Total Cost** | | | 5n + 2 |
| **Runtime** | | | O(n) |

**IV. Evaluations:**

**Vector’s advantages**: size can be dynamic meaning vector will automatically increase its size to make room for new value and since size of vector can be increase, it’s not necessary to declare it every time, so it’s perfect to use insert function and remove function in the algorithm. Also, the time complexity is much faster than using array or any other types of linked lists

**Vector’s disadvantage**: same with linked list, every modification such as insert, remove will shift the position entirely, and search is the worst because it picks every value from the start to compare until the correspond value found.

**Hash table’s advantage**: simple compare key with search value with complexity for search is O(1). An ideal case for hash function is it’s well constructed with no collision or minimal collisions, even though, the hashing process keep looping until there is more collision. Also, as mention before, O(1) with search as well as insert

**Hash table’s disadvantage**: collision is the main reason making it worse as well as the code complexity, since we are going deal with several chainning method to avoid collision

**Binary Tree’s advantage**: since it’s a tree with root node and child node, search function seems to perform better than vector or linked list, because it traverses down left child and right child to find match value. Also, code complexity is not as much as hash table, and we don’t have to worry about collision because of its tree structure.

**Binary Tree’s disadvantage**: the problem when implementing remove function is we need to carefully traverse to the child node from the parent node, so it requires an ideal traversal sequence, or else the parent node will be deleted accidentally and mess up the binary tree.

**V. Recommendations:**

Since the process of binary search tree is very straightforward, because it simply traverses downward left and right to get the user input value out of the tree. I will choose that data structure to implement for the project two. Also, the project doesn’t deal with remove value from tree so it’s ideal for search function. If we had to implement a remove function, I would consider choose hash table because of its O(1) time complexity on every search, insert and remove function at an average case.