**TOPIC**

Design and develop a question paper formation tool using an efficient data structure. Create three categories for pool of questions namely EASY, MEDIUM and HARD and store in separate files with 25 questions in each category for different topics of data structure subject. System should select questions randomly based on the count and difficulty level entered by the user from each category. Finally, your system should generate a question paper (Nirma University Format) for specified marks entered by user.

**Constraints:**

1. No question(s) must be repeated in the generated paper.
2. There must be less similarity between multiple variants of papers generated.

**METHODOLOGY**

**1.**

#include <bits/stdc++.h>

#include <fstream>

**<bits/stdc++.h> -** it typically includes a large number of standard C++ headers.

**#include <fstream> -** is used in C++ to include the standard C++ library header for file input and output operations.

2.

using namespace std;

const int MAX\_LINES = 10000;

constant integer named **MAX\_LINES** with a value of **10,000**. It limits the maximum number of lines that can be read from each file.

**ifstream (Input File Stream):**

The ifstream class is used for reading data from files. You can use it to open and read data from a file.

**ofstream (Output File Stream):**

The ofstream class is used for writing data to files. You can use it to open and write data to a file.

**3.**

// Function to save selected question indices to a file

void saveSelectedIndices(const vector<int> &indices, const string &saveSelIndices)

{

ofstream file(saveSelIndices);

for (int index : indices)

{

file << index << endl;

}

file.close();

}

* This function is responsible for saving a vector of selected question indices to a file. It takes two parameters:

**indices:** A constant reference to a vector of integers, which contains the question indices that were selected for a particular category (easy, medium, or hard).

**saveSelIndices:** in this file the indices will be saved.

* It iterates through the indices vector and writes each index to the file.
* The purpose of this function is to store the selected question indices so that they can be reused in future runs of the program. This way, the program avoids selecting the same questions in subsequent runs.

**4.**

// Function to load selected question indices from a file

vector<int> loadSelectedIndices(const string &loadselInd)

{

vector<int> indices;

ifstream file(loadselInd);

int index;

while (file >> index)

{

indices.push\_back(index);

}

return indices;

}

* This function is responsible for loading previously selected question indices from a file.
* It reads integers from the file, assuming one index per line, and stores them in a vector.
* It returns the vector containing the loaded indices.

The purpose of this function is to retrieve the previously selected question indices that were saved during previous runs. These indices are used to prevent the selection of the same questions in the current run of the program. By comparing the current selections with previously saved indices, the program ensures that questions are not duplicated across different runs, maintaining variety in the generated question paper.

So basically, **saveSelectedIndices** is used to store the selected question indices for future reference.

And **loadSelectedIndices** is used to retrieve these indices from previous runs.

This functionality helps maintain the integrity of the generated question paper and prevents the same questions from being selected repeatedly.

**5.**

int main()

{

ifstream easy\_file("/Users/nishitasolanki/Downloads/EASY (1).txt");

ifstream med\_file("/Users/nishitasolanki/Downloads/MEDIUM.txt");

ifstream hard\_file("/Users/nishitasolanki/Downloads/HARD.txt");

if (!easy\_file || !med\_file || !hard\_file)

{

cout << "Failed to open one or more of the files." << endl;

return 1;

}

string easy[MAX\_LINES];

string med[MAX\_LINES];

string hard[MAX\_LINES];

In the above ,

* **easy\_file, med\_file, and hard\_file,** each opened with a specific file path. They are used for reading the contents of text files.
* **if condition** – checks if any of them failed to open and it prints an error message and returns with an exit code of 1 to indicate an error.
* Declares arrays of strings named **easy, med, and hard**. These arrays are used to store the lines of questions read from the respective files.

**6.**

// Read questions from the easy file

while (getline(easy\_file, ques) && cnt < MAX\_LINES)

{

easy[cnt] = ques;

cnt++;

}

easy\_file.close();

cnt = 0;

string ques1;

// Read questions from the medium file

while (getline(med\_file, ques1) && cnt < MAX\_LINES)

{

med[cnt] = ques1;

cnt++;

}

med\_file.close();

cnt = 0;

string ques2;

// Read questions from the hard file

while (getline(hard\_file, ques2) && cnt < MAX\_LINES)

{

hard[cnt] = ques2;

cnt++;

}

hard\_file.close();

* Initializes an integer variable cnt to 0, which will be used to count the number of lines read.
* Defines a string variable ques to temporarily store each line of a question.
* Reads questions from the "EASY (1).txt" file It reads lines using getline, stores them in the ques variable, and assigns them to the easy array at the position indicated by cnt. It also increments cnt. The loop continues as long as there are lines to read (getline returns true) and the number of lines read is less than MAX\_LINES.
* And at last after reading all the questions, it closes the easy\_file stream.

**7.**

int n1, n2, n3, marks;

cout << "Enter the number of easy questions: " << endl;

cin >> n1;

cout << "Enter the number of medium questions: " << endl;

cin >> n2;

cout << "Enter the number of hard questions: " << endl;

cin >> n3;

cout << "Enter the marks" << endl;

cin >> marks;

Declares integer variables **n1, n2, n3, and marks** to store the user's input for the number of easy, medium, and hard questions, as well as the total marks for the question paper.

**8.**

int m1 = (2 \* n1) + (5 \* n2) + (10 \* n3); // check for marks validity

if (m1 > marks)

{

cout << "ERROR!" << endl;

}

Checks if the total marks entered by the user doesn’t exceeds the marks calculated of the total ques entered by the user.

**9.**

else

{

string head = "NIRMA UNIVERSITY QUESTION PAPER";//padding to print it in the middle.

int totalWidth = 120;

int padding = (totalWidth - head.length()) / 2;

for (int i = 0; i < padding; i++)

{

cout << ' ';

}

cout << head << endl;

For designing – It prints the Nirma University Question Paper in the middle.

**10.**

srand(static\_cast<unsigned>(time(nullptr))); // Seed with current time

int qno = 1; // Initialize the question number to 1

* It seeds the random number generator using the current time, ensuring that the generated random numbers will be different in each program run.
* It initializes the variable qno to 1, which represents the question number on the question paper.

**11.**

vector<int> selectedEasyIndices, selectedMediumIndices, selectedHardIndices;

// Load previously selected question indices

vector<int> previousEasyIndices = loadSelectedIndices("selected\_easy\_indices.txt");

vector<int> previousMediumIndices = loadSelectedIndices("selected\_medium\_indices.txt");

vector<int> previousHardIndices = loadSelectedIndices("selected\_hard\_indices.txt");

* creates three vectors, **selectedEasyIndices, selectedMediumIndices, and selectedHardIndices** to store the indices of selected questions from each category (easy, medium, hard).
* It loads previously selected question indices from separate files for each category into three vectors: **previousEasyIndices, previousMediumIndices, and previousHardIndices.** These vectors help prevent selecting the same questions in the current run that were selected in previous runs.

**12.**

cout << endl;

string head1 = "SECTION-A(2 Marks)";

int padding1 = (totalWidth - head1.length()) / 2;

for (int i = 0; i < padding1; i++)

{

cout << ' ';

}

cout << head1 << endl;

For designing – It prints the SECTION\_A(2 Marks) in the middle.

**13.**

for (int i = 0; i < n1; i++)

{

int q;

do

{

q = rand() % cnt;

} while (find(selectedEasyIndices.begin(), selectedEasyIndices.end(), q) != selectedEasyIndices.end() ||

find(previousEasyIndices.begin(), previousEasyIndices.end(), q) != previousEasyIndices.end());

selectedEasyIndices.push\_back(q);

cout << qno++ << easy[q] << endl;

}

* q = rand() % cnt; - This line generates a random number q within the range [0, cnt) using the rand() function. The variable cnt represents the total number of questions available in the "easy" category.

The do-while loop begins here. The purpose of this loop is to keep generating a random number q until two conditions are met:

* **find(selectedEasyIndices.begin(), selectedEasyIndices.end(), q) != selectedEasyIndices.end()** - This condition checks if the randomly generated q is already present in the selectedEasyIndices vector. If it is, it means that this question has already been selected in the current run, and the loop continues to generate a new random number.
* **find(previousEasyIndices.begin(), previousEasyIndices.end(), q) != previousEasyIndices.end()** - This condition checks if the randomly generated q is present in the previousEasyIndices vector, which stores indices of questions selected in previous runs. If it is found in the previous indices, it means the question was selected in a previous run, and the loop continues to generate a new random number.

The loop continues until it finds a random number q that satisfies both conditions, ensuring that the selected question is neither a duplicate within the current run nor a repeat from previous runs.

This code selects new questions that haven't been chosen in the current run or any previous runs. It helps maintain variety and prevents the same questions from being included in the generated question paper.

**14.**

cout << endl;

string head2 = "SECTION - B(5 Marks)";

int padding2 = (totalWidth - head2.length()) / 2;

for (int i = 0; i < padding2; i++)

{

cout << ' ';

}

cout << head2 << endl;

for (int i = 0; i < n2; i++)

{

int q;

do

{

q = rand() % cnt;

} while (find(selectedMediumIndices.begin(), selectedMediumIndices.end(), q) != selectedMediumIndices.end() ||

find(previousMediumIndices.begin(), previousMediumIndices.end(), q) != previousMediumIndices.end());

selectedMediumIndices.push\_back(q);

cout << qno++ << med[q] << endl;

}

cout << endl;

string head3 = "SECTION-C(10 MARKS)";

int padding3 = (totalWidth - head3.length()) / 2;

for (int i = 0; i < padding3; i++)

{

cout << ' ';

}

cout << head3 << endl;

for (int i = 0; i < n3; i++)

{

int q;

do

{

q = rand() % cnt;

} while (find(selectedHardIndices.begin(), selectedHardIndices.end(), q) != selectedHardIndices.end() ||

find(previousHardIndices.begin(), previousHardIndices.end(), q) != previousHardIndices.end());

selectedHardIndices.push\_back(q);

cout << qno++ << hard[q] << endl;

}

* **Same thing for medium and hard questions.**

**15**

// Save the new selected indices for the next run

saveSelectedIndices(selectedEasyIndices, "selected\_easy\_indices.txt");

saveSelectedIndices(selectedMediumIndices, "selected\_medium\_indices.txt");

saveSelectedIndices(selectedHardIndices, "selected\_hard\_indices.txt");

}

return 0;

}

* **saveSelectedIndices(selectedEasyIndices, "selected\_easy\_indices.txt"):** This line calls the saveSelectedIndices function to save the indices of the newly selected "easy" questions in the current run. The selectedEasyIndices vector contains these indices. They are saved to a text file named "selected\_easy\_indices.txt."

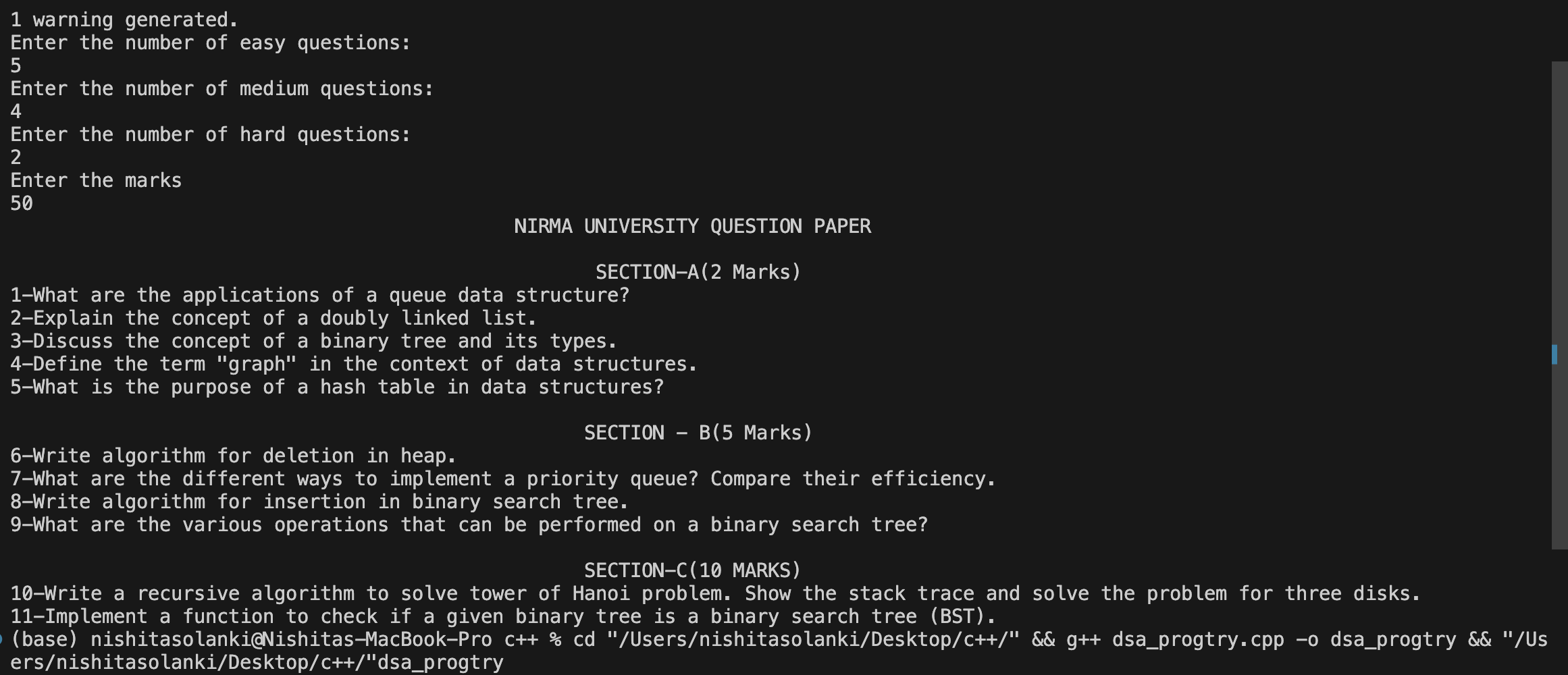
**And similarly saves medium and hard questions.**

* It allows the program to remember which questions were selected in the current run. This prevents the same questions from being selected again in the same session, ensuring that the question paper has variety.
* It enables the program to maintain a history of selected questions across different runs. By checking against the indices stored in these files, the program ensures that questions selected in previous runs are not duplicated in future runs.

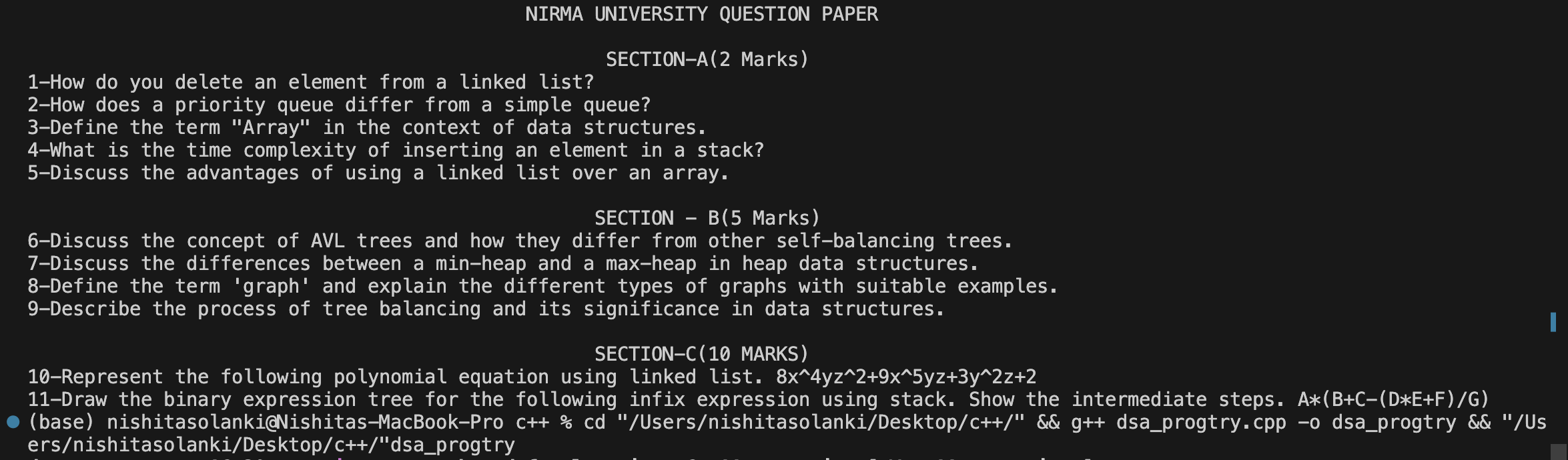
This part of the code is responsible for saving the selected question indices to files, both for maintaining session-to-session consistency and avoiding repetition of questions in future runs of the program.

**OUTPUT**

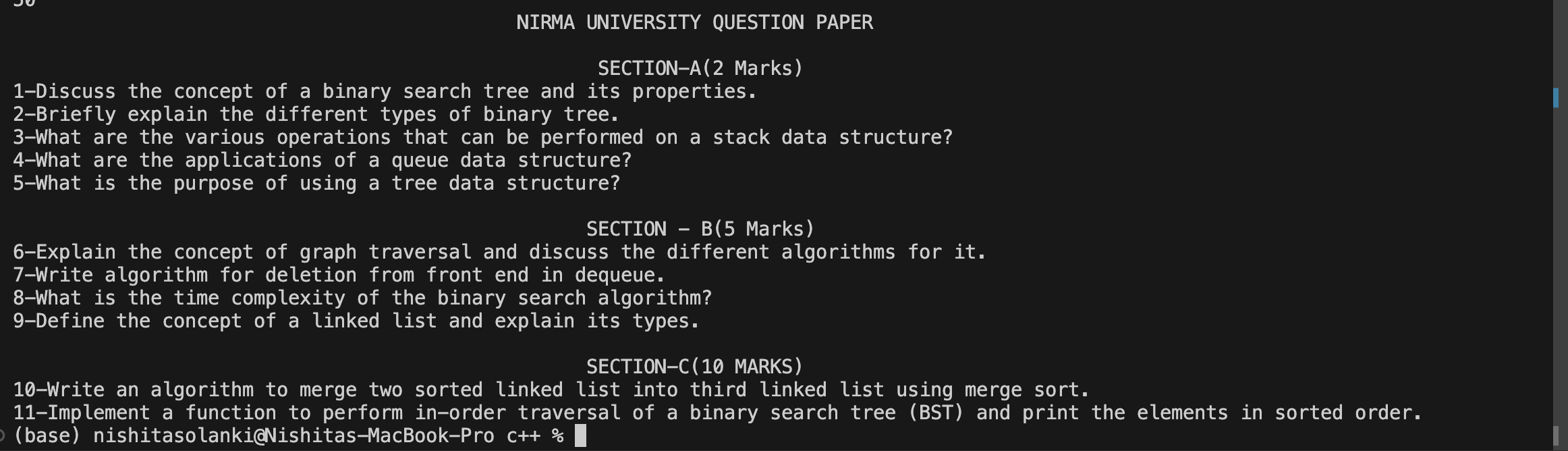
**PAPER -1**



**PAPER -2**



**PAPER- 3**



From the above we can see that there are no questions repeated in the 3 paper generated.