**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 1:- Write a C menudriven program to reverse a single linked list in O(N) time with a non-recursive function. \*/**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

// Structure to represent a node in the singly linked list

struct Node

{

int info; // Data stored in the node

struct Node\* next; // Pointer to the next node in the list

};

// Function to insert a node at the end of the list

struct Node\* insert(struct Node\* head)

{

struct Node\* t = (struct Node\*)malloc(sizeof(struct Node)); // Create a new node and allocate memory

if (t == NULL) // Check if memory allocation failed

{

printf("Memory allocation failed\n");

return head; // Return the head if memory allocation fails

}

printf("Enter the data to be inserted in the new node:\n");

scanf("%d", &(t->info)); // Input data for the new node

t->next = NULL; // Set the next pointer of the new node to NULL

if (head == NULL) // If the list is empty, the new node becomes the head

{

return t; // Return the new node as the head

}

struct Node\* p = head;

while (p->next != NULL) // Traverse to the end of the list

{

p = p->next;

}

p->next = t; // Insert the new node at the end of the list

return head; // Return the updated head

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1\*/**

}

// Function to reverse the linked list

struct Node\* reverse(struct Node\* head)

{

struct Node\* p1 = NULL; // Previous pointer

struct Node\* p2 = head; // Current pointer

struct Node\* p3 = NULL; // Next pointer

while (p2 != NULL) // Traverse through the list

{

p3 = p2->next; // Save the next node

p2->next = p1; // Reverse the current node's pointer to point to the previous node

p1 = p2; // Move p1 to the current node

p2 = p3; // Move p2 to the next node

}

return p1; // Return the new head of the reversed list

}

// Function to print the linked list

void print(struct Node\* head)

{

if (head == NULL) // Check if the list is empty

{

printf("List is empty.\n");

return; // Exit the function if the list is empty

}

struct Node\* p = head;

while (p != NULL) // Traverse through the list and print each node's data

{

printf("%d -> ", p->info);

p = p->next;

}

printf("NULL\n"); // Print "NULL" to indicate the end of the list

}

// Main function to control the flow of the program

int main()

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1\*/**

{

struct Node\* head = NULL; // Initialize the head of the list as NULL

int c; // Variable to store the user's choice

printf("\t\t\t\t\*\*\*\*\*INPUT\*\*\*\*\*\t\t\t\t\n");

printf("1. Insert at end\n");

printf("2. Display list\n");

printf("3. Reverse list\n");

printf("4. Exit\n");

printf("\t\t\t\t\*\*\*\*\*OUTPUT\*\*\*\*\*\t\t\t\t\n");

// User interface with options to interact with the linked list

do

{

printf("\nEnter your choice: ");

scanf("%d", &c); // Input the user's choice

switch (c) // Switch based on the user's choice

{

case 1:

head = insert(head); // Insert a new node at the end of the list

break;

case 2:

print(head); // Print the current linked list

break;

case 3:

head = reverse(head); // Reverse the linked list

printf("The list has been reversed.\n");

break;

case 4:

printf("Exit\n");

return 0; // Exit the program

default:

printf("Invalid choice.\n"); // Handle invalid input

}

}

while (1); // Continue until the user chooses to exit

return 0;

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1\*/**

snehagoyal@Desktop-36/ gcc reverse.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

1. Insert at end

2. Display list

3. Reverse list

4. Exit

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter your choice: 1

Enter the data to be inserted in the new node:

10

Enter your choice: 1

Enter the data to be inserted in the new node:

20

Enter your choice: 1

Enter the data to be inserted in the new node:

30

Enter your choice: 1

Enter the data to be inserted in the new node:

40

Enter your choice: 1

Enter the data to be inserted in the new node:

50

Enter your choice: 2

10 -> 20 -> 30 -> 40 -> 50 -> NULL

Enter your choice: 3

The list has been reversed.

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

Enter your choice: 2

50 -> 40 -> 30 -> 20 -> 10 -> NULL

Enter your choice: 4

Exit

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 2:- Write a menu driven program to delete duplicate name(if any) in the single Link list. The information field of each node should contain name and roll no. of students.\*/**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Structure to represent a node in the singly linked list

struct Node

{

char name[50]; // Name of the student (string of up to 50 characters)

int rollNo; // Roll number of the student (integer)

struct Node\* next; // Pointer to the next node in the list

};

// Function to insert a node at the end of the list

struct Node\* insert(struct Node\* head)

{

struct Node\* t = (struct Node\*)malloc(sizeof(struct Node)); // Allocate memory for the new node

if (t == NULL) // Check if memory allocation failed

{

printf("Memory allocation failed\n");

return head; // Return the original head if memory allocation fails

}

// Input name and roll number for the new node

printf("Enter name: ");

scanf("%s", t->name);

printf("Enter roll number: ");

scanf("%d", &t->rollNo);

t->next = NULL; // Set the next pointer of the new node to NULL (it will be the last node initially)

if (head == NULL) // If the list is empty, the new node becomes the head of the list

{

return t; // Return the new node as the head

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

struct Node\* p = head;

while (p->next != NULL) // Traverse the list to reach the last node

{

p = p->next;

}

p->next = t; // Link the last node's next pointer to the new node

return head; // Return the original head

}

// Function to delete duplicate names in the list

struct Node\* delDup(struct Node\* head)

{

if (head == NULL) // Check if the list is empty

{

printf("List is empty.\n");

return head; // Return the original head if the list is empty

}

struct Node\* p1 = head; // Pointer to traverse the list

while (p1 != NULL) // Traverse each node

{

struct Node\* p2 = p1; // Pointer to track the previous node

struct Node\* p3 = p1->next; // Pointer to check the remaining nodes for duplicates

while (p3 != NULL) // Traverse the list for duplicate nodes

{

if (strcmp(p1->name, p3->name) == 0) // Check if names are the same

{

p2->next = p3->next; // Remove the duplicate node by linking the previous node to the next of p3

free(p3); // Free the memory allocated to the duplicate node

p3 = p2->next; // Move p3 to the next node

}

else

{

p2 = p3; // Move p2 to the current node

p3 = p3->next; // Move p3 to the next node

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

}

p1 = p1->next; // Move p1 to the next node

}

printf("Duplicates removed successfully.\n");

return head; // Return the modified head of the list

}

// Function to display the linked list

void print(struct Node\* head)

{

if (head == NULL) // Check if the list is empty

{

printf("List is empty.\n");

return; // Exit the function if the list is empty

}

struct Node\* p = head; // Pointer to traverse the list

while (p != NULL) // Traverse through the list

{

// Print the name and roll number of each node

printf("Name: %s, Roll No: %d -> ", p->name, p->rollNo);

p = p->next; // Move to the next node

}

printf("NULL\n"); // Print "NULL" to indicate the end of the list

}

// Main function to control the program flow

int main()

{

struct Node\* head = NULL; // Initialize the head of the list as NULL

int c; // Variable to store the user's choice

printf("\t\t\t\t\*\*\*\*\* INPUT \*\*\*\*\*\t\t\t\t\n");

printf("1. Insert \n");

printf("2. Display list\n");

printf("3. Delete duplicate names\n");

printf("4. Exit\n");

printf("\t\t\t\t\*\*\*\*\* OUTPUT \*\*\*\*\*\t\t\t\t\n");

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

// Menu-driven interface to interact with the linked list

do

{

printf("\nEnter your choice: ");

scanf("%d", &c); // Input the user's choice

switch (c) // Switch based on the user's choice

{

case 1:

head = insert(head); // Call the insert function to add a node to the list

break;

case 2:

print(head); // Call the print function to display the list

break;

case 3:

head = delDup(head); // Call the delDup function to remove duplicate names

break;

case 4:

printf("Exiting program.\n");

return 0; // Exit the program

default:

printf("Invalid choice.\n"); // Handle invalid input

}

}

while (1); // Keep the program running until the user chooses to exit

return 0;

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

snehagoyal@Desktop-36/ gcc duplicate.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\* INPUT \*\*\*\*\*

1. Insert

2. Display list

3. Delete duplicate names

4. Exit

\*\*\*\*\* OUTPUT \*\*\*\*\*

Enter your choice: 1

Enter name: Sneha

Enter roll number: 60

Enter your choice: 1

Enter name: Vartika

Enter roll number: 18

Enter your choice: 1

Enter name: Sneha

Enter roll number: 12

Enter your choice: 1

Enter name: Bani

Enter roll number: 13

Enter your choice: 2

Name: Sneha, Roll No: 60 -> Name: Vartika, Roll No: 18 -> Name: Sneha, Roll No: 12 -> Name: Bani, Roll No: 13 -> NULL

Enter your choice: 3

Duplicates removed successfully.

Enter your choice:

2

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

Name: Sneha, Roll No: 60 -> Name: Vartika, Roll No: 18 -> Name: Bani, Roll No: 13 -> NULL

Enter your choice: 4

Exiting program.

**/\*Name:-SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 3:- Write a C program to find and delete the node with second minimum data in the single link list. \*/**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

// Structure for a node in the singly linked list

struct Node

{

int info; // Data stored in the node

struct Node\* next; // Pointer to the next node in the list

};

// Function to insert a node at the end of the list

struct Node\* insert(struct Node\* head)

{

struct Node\* t = (struct Node\*)malloc(sizeof(struct Node)); // Allocate memory for a new node

if (t == NULL) // Check if memory allocation failed

{

printf("Memory allocation failed\n");

return head;

}

printf("Enter the data to be inserted in the new node:\n");

scanf("%d", &(t->info)); // Input data for the new node

t->next = NULL; // Set the next pointer of the new node to NULL

if (head == NULL) // If the list is empty, make the new node the head

{

return t;

}

struct Node\* p = head;

while (p->next != NULL) // Traverse the list to find the last node

{

p = p->next;

}

p->next = t; // Insert the new node at the end of the list

return head;

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

// Function to display the linked list

void print(struct Node\* head)

{

if (head == NULL) // Check if the list is empty

{

printf("List is empty.\n");

return;

}

struct Node\* p = head;

while (p != NULL) // Traverse the list and print each node's data

{

printf("%d -> ", p->info);

p = p->next;

}

printf("NULL\n"); // Indicate the end of the list

}

// Function to find and delete the node with the second minimum data

struct Node\* delSecondMin(struct Node\* head)

{

if (head == NULL || head->next == NULL) // Check if the list has fewer than two nodes

{

printf("List does not have enough nodes.\n");

return head;

}

struct Node\* min = head; // Pointer to the node with the minimum data

struct Node\* secondMin = NULL; // Pointer to the node with the second minimum data

struct Node\* p = head->next;

while (p != NULL) // Traverse the list to find the minimum and second minimum nodes

{

if (p->info < min->info)

{

secondMin = min;

min = p;

}

else if (secondMin == NULL || p->info < secondMin->info)

{

secondMin = p;

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

}

p = p->next;

}

if (secondMin == NULL) // Check if a second minimum node was found

{

printf("Second minimum does not exist.\n");

return head;

}

struct Node\* prev = NULL; // Pointer to track the previous node

p = head;

while (p != NULL && p != secondMin) // Traverse the list to find and delete the second minimum node

{

prev = p;

p = p->next;

}

if (p == secondMin)

{

if (prev != NULL) // If the second minimum is not the head

{

prev->next = p->next;

}

else // If the second minimum is the head

{

head = p->next;

}

free(p); // Delete the second minimum node

printf("Second minimum node deleted successfully.\n");

}

return head;

}

// Main function

int main()

{

struct Node\* head = NULL;

int c;

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

printf("\t\t\t\t\*\*\*\*\*INPUT\*\*\*\*\*\t\t\t\t\n");

printf("1. Insert at end\n");

printf("2. Display list\n");

printf("3. Remove second minimum\n");

printf("4. Exit\n");

printf("\t\t\t\t\*\*\*\*\*OUTPUT\*\*\*\*\*\t\t\t\t\n");

do

{

printf("\nEnter your choice: ");

scanf("%d", &c);

switch (c)

{

case 1:

head = insert(head);

break;

case 2:

print(head);

break;

case 3:

head = delSecondMin(head);

break;

case 4:

printf("Exit\n");

return 0;

default:

printf("Invalid choice.\n");

}

} while (1);

return 0;

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

snehagoyal@Desktop-36/gcc mindata.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

1. Insert at end

2. Display list

3. Remove second minimum

4. Exit

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter your choice: 1

Enter the data to be inserted in the new node:

10

Enter your choice: 1

Enter the data to be inserted in the new node:

9

Enter your choice: 1

Enter the data to be inserted in the new node:

12

Enter your choice: 1

Enter the data to be inserted in the new node:

11

Enter your choice: 1

Enter the data to be inserted in the new node:

13

Enter your choice: 2

10 -> 9 -> 12 -> 11 -> 13 -> NULL

Enter your choice: 3

Second minimum node deleted successfully.

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

Enter your choice: 2

9 -> 12 -> 11 -> 13 -> NULL

Enter your choice: 4

Exit

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 4:- Write a C program to create a single linked list, then find whether there exist any two nodes in the linked list whose multiplication is less than the target value.\*/**

#include <stdio.h>

#include <stdlib.h>

#include <malloc.h>

// Structure for a node in the singly linked list

struct Node

{

int info; // Data field to store the information in the node

struct Node\* next; // Pointer to the next node in the list

};

// Function to insert a node at the end of the list

struct Node\* insert(struct Node\* head)

{

struct Node\* t = (struct Node\*)malloc(sizeof(struct Node)); // Allocate memory for the new node

if (t == NULL) // Check if memory allocation failed

{

printf("Memory allocation failed\n");

return head; // Return the head if memory allocation fails

}

// Prompt the user to input the data for the new node

printf("Enter the data to be inserted in the new node:\n");

scanf("%d", &(t->info)); // Read the integer data into the new node

t->next = NULL; // Set the next pointer of the new node to NULL (it will be the last node initially)

// If the list is empty, make the new node the head of the list

if (head == NULL)

{

return t; // Return the new node as the head of the list

}

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

struct Node\* p = head; // Pointer to traverse the list

// Traverse the list until the last node is reached

while (p->next != NULL)

{

p = p->next; // Move to the next node in the list

}

p->next = t; // Link the last node to the new node

return head; // Return the head of the list (unchanged)

}

// Function to display the linked list

void print(struct Node\* head)

{

// Check if the list is empty

if (head == NULL)

{

printf("List is empty.\n");

return; // Exit the function if the list is empty

}

struct Node\* p = head; // Pointer to traverse the list

// Traverse through the list and print each node's data

while (p != NULL)

{

printf("%d -> ", p->info); // Print the node's data

p = p->next; // Move to the next node in the list

}

printf("NULL\n"); // Print "NULL" to indicate the end of the list

}

// Function to check if any two nodes' multiplication is less than the target value

void multi(struct Node\* head, int t)

{

// If the list has fewer than two nodes, no pair can be formed

if (head == NULL || head->next == NULL)

{

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

printf("No such pair exists.\n");

return; // Exit the function if there are not enough nodes

}

struct Node\* p1 = head; // Pointer to the first node in the pair

// Traverse through the list with p1

while (p1 != NULL)

{

struct Node\* p2 = p1->next; // Pointer to the second node in the pair

// Traverse the remaining list with p2

while (p2 != NULL)

{

// If the multiplication of the two nodes' values is less than the target value, print "Yes"

if (p1->info \* p2->info < t)

{

printf("Yes\n");

return; // Exit after finding the first valid pair

}

p2 = p2->next; // Move to the next node

}

p1 = p1->next; // Move to the next node in the list

}

// If no valid pair is found after checking all pairs, print "No"

printf("No\n");

}

// Main function to control the program flow

int main()

{

struct Node\* head = NULL; // Initialize the head of the linked list as NULL

int c; // Variable to store the user's choice

// Display menu options

printf("\t\t\t\t\*\*\*\*\* INPUT \*\*\*\*\*\t\t\t\t\n");

printf("1. Insert at end\n");

printf("2. Display list\n");

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

printf("3. Check multiplication\n");

printf("4. Exit\n");

printf("\t\t\t\t\*\*\*\*\* OUTPUT \*\*\*\*\*\t\t\t\t\n");

do

{

printf("\nEnter your choice: ");

scanf("%d", &c); // Input the user's choice

switch (c) // Switch case based on user input

{

case 1:

head = insert(head); // Call insert function to add a node

break;

case 2:

print(head); // Call print function to display the list

break;

case 3:

{

int t;

printf("Enter the target value: ");

scanf("%d", &t); // Input the target value for multiplication check

multi(head, t); // Call the multi function to check if any pair's multiplication is less than the target value

}

break;

case 4:

printf("Exit\n");

return 0; // Exit the program

default:

printf("Invalid choice.\n"); // Handle invalid menu choices

}

}

while (1); // Keep looping until the user chooses to exit

return 0;

}

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

snehagoyal@Desktop-36/ gcc multi.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

1. Insert at end

2. Display list

3. Check multiplication

4. Exit

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter your choice: 1

Enter the data to be inserted in the new node:

10

Enter your choice: 1

Enter the data to be inserted in the new node:

20

Enter your choice: 1

Enter the data to be inserted in the new node:

3

Enter your choice: 1

Enter the data to be inserted in the new node:

5

Enter your choice: 1

Enter the data to be inserted in the new node:

30

Enter your choice: 2

10 -> 20 -> 3 -> 5 -> 30 -> NULL

Enter your choice: 2

10 -> 20 -> 3 -> 5 -> 30 -> NULL

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

Enter your choice: 3

Enter the target value: 20

Yes

Enter your choice: 3

Enter the target value: 10

No

Enter your choice: 4

Exit

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 5:- Write a C program to find the node in the doubly linked list at which the two linked lists are intersected. \*/**

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the doubly linked list

struct Node

{

int data; // Data stored in the node

struct Node\* next; // Pointer to the next node

struct Node\* prev; // Pointer to the previous node

};

// Function to create a new node with given data

struct Node\* create(int data)

{

struct Node\* t = (struct Node\*)malloc(sizeof(struct Node)); // Allocate memory for a new node

t->data = data; // Set the data for the new node

t->prev = NULL; // Set the previous pointer to NULL, as it will be the first node

t->next = NULL; // Set the next pointer to NULL, as it will be the last node initially

return t; // Return the newly created node

}

// Function to insert a node at the end of the list

void insert(struct Node\*\* head, int data)

{

struct Node\* t = create(data); // Create a new node with the provided data

if (\*head == NULL)

{

\*head = t; // Set the head to the new node

return;

}

struct Node\* p = \*head;

while (p->next != NULL)

{

p = p->next;

}

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

p->next = t; // Set the next pointer of the last node to the new node

t->prev = p; // Set the previous pointer of the new node to the last node

}

// Function to find the intersection point of two doubly linked lists

struct Node\* find(struct Node\* list1, struct Node\* list2)

{

struct Node\* p1 = list1; // Pointer to traverse the first list

struct Node\* p2; // Pointer to traverse the second list

while (p1 != NULL)

{

p2 = list2; // For each node in list1, start from the beginning of list2

while (p2 != NULL)

{

if (p1->data == p2->data)

{

return p1; // Return the node where intersection occurs

}

p2 = p2->next; // Move to the next node in list2

}

p1 = p1->next; // Move to the next node in list1

}

return NULL; // If no intersection is found, return NULL

}

// Function to print the doubly linked list

void print(struct Node\* head)

{

struct Node\* p = head; // Pointer to traverse the list

while (p != NULL)

{

printf("%d \t ", p->data); // Print the data of the current node

p = p->next; // Move to the next node

}

}

// Main function to execute the program

int main()

{

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

struct Node\* list1 = NULL; // Initialize the first list as empty

struct Node\* list2 = NULL; // Initialize the second list as empty

int n1, n2, data, choice; // Variables for storing user input

printf("\t\t\t\t\*\*\*\*\*INPUT\*\*\*\*\*\t\t\t\t\n");

printf("\nMenu:\n");

printf("1. Insert elements in both lists\n");

printf("2. Print both lists\n");

printf("3. Find the intersection of both lists\n");

printf("4. Exit\n");

printf("\t\t\t\t\*\*\*\*\*OUTPUT\*\*\*\*\*\t\t\t\t\n");

do

{

printf("Enter your choice: ");

scanf("%d", &choice); // Read the user's menu choice

switch (choice)

{

case 1:

printf("Enter the number of elements for List 1: ");

scanf("%d", &n1); // Read number of elements for list1

printf("Enter the elements for List 1:\n");

for (int i = 0; i < n1; i++) // Insert elements into list1

{

scanf("%d", &data); // Read the element

insert(&list1, data); // Insert the element into the list

}

printf("Enter the number of elements for List 2: ");

scanf("%d", &n2); // Read number of elements for list2

printf("Enter the elements for List 2:\n");

for (int i = 0; i < n2; i++) // Insert elements into list2

{

scanf("%d", &data); // Read the element

insert(&list2, data); // Insert the element into the list

}

break;

case 2:

printf("\nList 1: ");

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

print(list1); // Print list1

printf("\n");

printf("List 2: ");

print(list2); // Print list2

printf("\n");

break;

case 3:

{

struct Node\* inter = find(list1, list2); // Find intersection

if (inter != NULL)

{

printf("Intersection at node with data: %d\n", inter->data); // Print the intersection data

}

else

{

printf("No intersection found.\n"); // Print if no intersection found

}

}

break;

case 4:

printf("Exiting program...\n");

break;

default:

printf("Invalid choice! Please try again.\n");

}

} while (1); // Continue until the user chooses to exit

return 0;

}

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

snehagoyal@Desktop-36/ gcc intersection.c

snehagoyal@Desktop-36/ ./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

Menu:

1. Insert elements in both lists

2. Print both lists

3. Find the intersection of both lists

4. Exit

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter your choice: 1

Enter the number of elements for List 1: 4

Enter the elements for List 1:

1

2

3

4

Enter the number of elements for List 2: 5

Enter the elements for List 2:

3

4

5

6

7

Enter your choice: 3

Intersection at node with data: 3

Enter your choice: 2

List 1: 1 2 3 4

List 2: 3 4 5 6 7

Enter your choice: 4

Exiting program...

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1**

**Question 6:- W.A.P. to create a binary search tree and perform the following operations:**

**1) Find a node with second highest data.**

**2) Delete a node from the tree.**

**3) Count total number of nodes having common parent.**

**4) Find the height of a binary search tree**

**5) Count total numbers of nodes from left hand side of the root node. \*/**

#include <stdio.h>

#include <stdlib.h>

// Structure for the Binary Search Tree (BST)

typedef struct node

{

int info; // Data stored in the node

struct node\* left; // Pointer to the left child

struct node\* right; // Pointer to the right child

} bst;

// Function to create a new node

bst\* create\_node(int info)

{

bst\* new = (bst\*)malloc(sizeof(bst)); // Allocate memory for the new node

new->info = info; // Set the data for the node

new->left = NULL; // No left child

new->right = NULL; // No right child

return new; // Return the new node

}

// Function to insert a node into the BST

bst\* insert(bst\* root, int data)

{

if (root == NULL) // If the tree is empty, create a new node

return create\_node(data);

if (data < root->info) // If data is smaller, insert in the left subtree

root->left = insert(root->left, data);

else // Otherwise, insert in the right subtree

root->right = insert(root->right, data);

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

return root; // Return the (unchanged) root pointer

}

// Function to find the maximum node in a tree (rightmost node)

bst\* find\_max(bst\* root)

{

while (root && root->right != NULL) // Traverse the tree to the rightmost node

root = root->right;

return root; // Return the maximum node

}

// Function to find the second highest node

int find\_secondmax(bst\* root)

{

bst\* parent = NULL;

while (root && root->right) // Traverse to the rightmost node

{

parent = root; // Keep track of the parent

root = root->right; // Move right

}

if (root && root->left) // If the rightmost node has a left child, the second max is the max of the left subtree

return find\_max(root->left)->info;

return parent ? parent->info : -1; // If no second max, return the parent or -1 if there is no parent

}

// Function to delete a node from the BST

bst\* delete\_node(bst\* root, int key)

{

if (root == NULL) // Base case: If the tree is empty, return NULL

return root;

if (key < root->info) // If key is smaller, move to the left subtree

root->left = delete\_node(root->left, key);

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

else if (key > root->info) // If key is larger, move to the right subtree

root->right = delete\_node(root->right, key);

else // Node to be deleted is found

{

// Node with only one child or no child

if (root->left == NULL)

{

bst\* temp = root->right; // If no left child, link to right child

free(root); // Free the node

return temp; // Return the right child

}

else if (root->right == NULL) // If no right child, link to left child

{

bst\* temp = root->left;

free(root);

return temp;

}

// Node with two children: Get the inorder successor (smallest in the right subtree)

bst\* temp = find\_max(root->left); // Find the maximum node in the left subtree

root->info = temp->info; // Replace the value of the current node with the max node value

root->left = delete\_node(root->left, temp->info); // Delete the inorder successor

}

return root; // Return the (possibly unchanged) root pointer

}

// Function to count nodes with a common parent (nodes with both left and right children)

int count\_common\_pnodes(bst\* root)

{

if (root == NULL) // Base case: If the tree is empty, return 0

return 0;

int count = 0;

if (root->left != NULL && root->right != NULL) // If both left and right children exist, it's a common parent node

count = 2;

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

else if (root->left != NULL || root->right != NULL) // If only one child exists, it's still a parent node

count = 1;

// Recursively count the common parent nodes in the left and right subtrees

return count + count\_common\_pnodes(root->left) + count\_common\_pnodes(root->right);

}

// Function to find the height of the BST

int find\_height(bst\* root)

{

if (root == NULL) // Base case: If the tree is empty, height is 0

return 0;

int lheight = find\_height(root->left); // Find the height of the left subtree

int rheight = find\_height(root->right); // Find the height of the right subtree

return 1 + (lheight > rheight ? lheight : rheight);

}

// Function to count the number of nodes in the left subtree

int count\_lsubtree\_nodes(bst\* root)

{

if (root == NULL || root->left == NULL) // If the root or left subtree is NULL, return 0

return 0;

// Count the current node, the left child, and recursively count the nodes in the left and right subtrees of the left child

return 1 + count\_lsubtree\_nodes(root->left) + count\_lsubtree\_nodes(root->left->right);

}

// Function for inorder traversal of the BST (left, root, right)

void inorder\_traversal(bst\* root)

{

if (root == NULL) // Base case: If the tree is empty, return

return;

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

inorder\_traversal(root->left); // Traverse the left subtree

printf("%d ", root->info); // Print the data of the current node

inorder\_traversal(root->right); // Traverse the right subtree

}

int main()

{

bst\* root = NULL; // Initialize an empty BST

int choice, value;

printf("\t\t\t\t\*\*\*\*\*INPUT\*\*\*\*\*\t\t\t\t\n");

printf("\nMenu:\n1. Insert\n2. Find second highest\n3. Delete a node\n4. Count nodes with common parent\n5. Find height of tree\n6. Count nodes in left subtree\n7. Display tree (Inorder)\n8. Exit\n");

printf("\t\t\t\t\*\*\*\*\*OUTPUT\*\*\*\*\*\t\t\t\t\n");

// Continuously prompt the user for input until they choose to exit

while (1)

{

printf("Enter your choice: ");

scanf("%d", &choice); // Read the user's choice

switch (choice)

{

case 1:

printf("Enter value to insert: ");

scanf("%d", &value);

root = insert(root, value); // Insert the value into the tree

break;

case 2:

value = find\_secondmax(root); // Find the second highest value

if (value != -1)

printf("Second highest value: %d\n", value);

else

printf("Tree has less than two nodes.\n");

break;

case 3:

printf("Enter value to delete: ");

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

scanf("%d", &value);

root = delete\_node(root, value); // Delete the specified node from the tree

break;

case 4:

printf("Total nodes with common parent: %d\n", count\_common\_pnodes(root)); // Count nodes with a common parent

break;

case 5:

printf("Height of tree: %d\n", find\_height(root)); // Find the height of the tree

break;

case 6:

printf("Nodes in left subtree: %d\n", count\_lsubtree\_nodes(root)); // Count nodes in the left subtree

break;

case 7:

printf("Inorder Traversal: ");

inorder\_traversal(root); // Print the inorder traversal of the tree

printf("\n");

break;

case 8:

return 0; // Exit the program

default:

printf("Invalid choice!\n"); // Handle invalid choice

}

}

return 0;

}

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

snehagoyal@Desktop-36/ gcc BST.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

Menu:

1. Insert

2. Find second highest

3. Delete a node

4. Count nodes with common parent

5. Find height of tree

6. Count nodes in left subtree

7. Display tree (Inorder)

8. Exit

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter your choice: 1

Enter value to insert: 80

Enter your choice: 1

Enter value to insert: 20

Enter your choice: 1

Enter value to insert: 90

Enter your choice: 1

Enter value to insert: 10

Enter your choice: 1

Enter value to insert: 100

Enter your choice: 1

Enter value to insert: 5

Enter your choice: 2

Second highest value: 90

Enter your choice: 3

Enter value to delete: 5

Enter your choice: 4

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

Total nodes with common parent: 4

Enter your choice: 5

Height of tree: 3

Enter your choice: 6

Nodes in left subtree: 2

Enter your choice: 7

Inorder Traversal: 10 20 80 90 100

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1**

**Question 7:- Write a C program to sort N names given by user in an array, using Quick sort technique. \*/**

#include <stdio.h>

#include <string.h>

#define MAX 100

#define NAME\_LENGTH 50

// Function to swap two strings

void swap(char copy[][NAME\_LENGTH], int i, int j)

{

char x[NAME\_LENGTH];

strcpy(x, copy[i]); // Copy the first string to temporary variable 'x'

strcpy(copy[i], copy[j]); // Copy the second string to the first position

strcpy(copy[j], x); // Copy the temporary variable 'x' (first string) to the second position

}

// Partition function for Quick Sort

int part(char copy[][NAME\_LENGTH], int l, int h)

{

char pivot[NAME\_LENGTH];

strcpy(pivot, copy[h]); // Select the pivot element (last element)

int i = l - 1; // Initialize the index for smaller element

for (int j = l; j < h; j++)

{

if (strcmp(copy[j], pivot) < 0)

{

i++; // Increment index of smaller element

swap(copy, i, j); // Swap the current element with the smaller element

}

}

swap(copy, i + 1, h); // Swap the pivot element with the element at index i+1

return i + 1; // Return the partition index

}

// Quick Sort function

void QuickSort(char copy[][NAME\_LENGTH], int l, int h)

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

{

if (l < h)

{

int pi = part(copy, l, h); // Get partition index

// Recursively sort elements before and after partition

QuickSort(copy, l, pi - 1);

QuickSort(copy, pi + 1, h);

}

}

int main()

{

int n;

char names[MAX][NAME\_LENGTH];

printf("Enter the number of names: ");

scanf("%d", &n); // Take input for the number of names

printf("Enter the names:\n");

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

{

scanf(" %[^\n]", names[i]); // Read each name including spaces

}

QuickSort(names, 0, n - 1); // Sort the names using QuickSort

printf("\nSorted names:\n");

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

printf("%s\n", names[i]); // Print the sorted names

return 0;

}

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

snehagoyal@Desktop-36/ gcc quicksort.c

snehagoyal@Desktop-36/./a.out

**\*\*\*INPUT\*\*\***

Enter the number of names: 5

Enter the names:

Varima

Sneha

Sanmita

Anushka

Vaishnavi

**\*\*\*OUTPUT\*\*\***

Sorted names:

Anushka

Sanmita

Sneha

Vaishnavi

Varima

**\*\*\*INPUT\*\*\***

Enter the number of names: 6

Enter the names:

VarimaDudeja

SnehaGoyal

Sanchi Kala

**/\*Name :- SnehaGoyal**

**University Roll No.:- 2023934**

**Section :- DS1 \*/**

VanshikaAgarwal

KopalAgarawal

AanchalGarg

**\*\*\*OUTPUT\*\*\***

Sorted names:

AanchalGarg

KopalAgarawal

Sanchi Kala

SnehaGoyal

VanshikaAgarwal

VarimaDudeja

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1**

**Question 8:- Write a C program to implement time sharing environment using circular linked list to allocate time slots of 10ns for N given processes, and then print which process will be completed in how much time.\*/**

#include <stdio.h>

#include <stdlib.h>

// Structure for a Process in the Round Robin scheduling

typedef struct Process

{

int id;

int gtime;

struct Process\* next;

} pr;

// Function to create a new process node

pr\* create\_pr(int id, int Time)

{

pr\* new\_pr = (pr\*)malloc(sizeof(pr)); // Allocate memory for a new process

new\_pr->id = id;

new\_pr->gtime = Time;

new\_pr->next = NULL; // Initialize the next pointer to NULL

return new\_pr;

}

// Function to add a new process to the circular linked list

void add\_pr(pr\*\* head, int id, int Time)

{

pr\* new\_pr = create\_pr(id, Time); // Create a new process with the given ID and burst time

if (\*head == NULL) // If the list is empty, make the new process the head

{

\*head = new\_pr;

new\_pr->next = \*head; // Point the new process back to the head (circular link)

}

else

{

pr\* temp = \*head;

// Traverse the list to find the last process

while (temp->next != \*head)

{

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1 \*/**

temp = temp->next;

}

// Insert the new process at the end and link it to the head

temp->next = new\_pr;

new\_pr->next = \*head;

}

}

// Function to perform the time-sharing simulation using Round Robin scheduling

void time\_sharing(pr\* head, int tslot)

{

if (head == NULL) // Check if the list is empty

{

printf("No processes available.\n");

return;

}

pr\* current = head; // Start from the head of the list

int etime = 0; // Variable to track the elapsed time

printf("Time-sharing simulation:\n");

// Continue simulation until all processes are completed

while (current != NULL)

{

if (current->gtime > 0) // Only execute processes that still have remaining burst time

{

// Print the process execution details

printf("Process %d executed for %d ns\n", current->id, tslot);

// Update the elapsed time and the remaining burst time for the process

etime += (current->gtime < tslot) ? current->gtime : tslot;

current->gtime -= tslot; // Decrease the remaining burst time of the process

// If the process is finished, print its completion details

if (current->gtime <= 0)

{

printf("Process %d completed at time %d ns\n", current->id, etime);

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1 \*/**

}

}

current = current->next; // Move to the next process in the circular list

pr\* temp = head; // Temporary pointer to check if all processes are completed

int all\_completed = 1;

do

{

if (temp->gtime > 0) // If any process still has remaining burst time

{

all\_completed = 0; // Not all processes are completed

break;

}

temp = temp->next; // Move to the next process

} while (temp != head); // Continue until we circle back to the head

if (all\_completed) // If all processes are completed, break out of the loop

{

break;

}

}

}

// Function to free the memory allocated for the process list

void free\_list(pr\* head)

{

if (head == NULL) // If the list is empty, there's nothing to free

{

return;

}

pr\* current = head;

pr\* temp;

do

{

temp = current->next; // Get the next process

free(current); // Free the current process

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1 \*/**

current = temp; // Move to the next process

} while (current != head); // Continue until we circle back to the head

}

// Main function to simulate Round Robin scheduling

void main()

{

int n, t;

pr\* head = NULL; // Initialize an empty process list

printf("Enter the number of processes: ");

scanf("%d", &n); // Take input for the number of processes

// Input the burst time for each process and add them to the list

for (int i = 1; i <= n; i++)

{

printf("Enter burst time for process %d (in ns): ", i);

scanf("%d", &t); // Input the burst time for the process

add\_pr(&head, i, t); // Add the process to the list

}

// Simulate the time-sharing scheduling with 10 ns time slots

printf("\nAllocating time slots of 10 ns...\n");

time\_sharing(head, 10);

// Free the memory allocated for the process list

free\_list(head);

}

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1**

snehagoyal@desktop-36/ gcc process\_time.c

snehagoyal@desktop-36/ ./a.out

**\*\*\*INPUT\*\*\***

Enter the number of processes: 6

Enter burst time for process 1 (in ns): 12

Enter burst time for process 2 (in ns): 8

Enter burst time for process 3 (in ns): 2

Enter burst time for process 4 (in ns): 4

Enter burst time for process 5 (in ns): 7

Enter burst time for process 6 (in ns): 1

**\*\*\*OUTPUT\*\*\***

Allocating time slots of 10 ns...

Time-sharing simulation:

Process 1 executed for 10 ns

Process 2 executed for 10 ns

Process 2 completed at time 18 ns

Process 3 executed for 10 ns

Process 3 completed at time 20 ns

Process 4 executed for 10 ns

Process 4 completed at time 24 ns

Process 5 executed for 10 ns

Process 5 completed at time 31 ns

Process 6 executed for 10 ns

Process 6 completed at time 32 ns

Process 1 executed for 10 ns

Process 1 completed at time 34 ns

**\*\*\*INPUT\*\*\***

Enter the number of processes: 5

Enter burst time for process 1 (in ns): 1

Enter burst time for process 2 (in ns): 2

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1 \*/**

Enter burst time for process 3 (in ns): 3

Enter burst time for process 4 (in ns): 4

Enter burst time for process 5 (in ns): 5

**\*\*\*OUTPUT\*\*\***

Allocating time slots of 10 ns...

Time-sharing simulation:

Process 1 executed for 10 ns

Process 1 completed at time 1 ns

Process 2 executed for 10 ns

Process 2 completed at time 3 ns

Process 3 executed for 10 ns

Process 3 completed at time 6 ns

Process 4 executed for 10 ns

Process 4 completed at time 10 ns

Process 5 executed for 10 ns

Process 5 completed at time 15 ns

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 9:- Write a C program to store the details of a weighted graph.\*/**

#include <stdio.h>

#define MAX\_NODES 100

// Function to input the graph details

void input\_graph(int graph[MAX\_NODES][MAX\_NODES], int nodes, int edges)

{

int x, y, weight;

// Loop to input edges and their weights

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

{

printf("Enter edge %d (source, destination, weight): ", i + 1);

scanf("%d %d %d", &x, &y, &weight); // Read source, destination, and weight

graph[x][y] = weight; // Set the weight for the directed edge from x to y

graph[y][x] = weight; // Set the weight for the directed edge from y to x (for undirected graph)

}

}

// Function to display the graph as an adjacency matrix

void display\_graph(int graph[MAX\_NODES][MAX\_NODES], int nodes)

{

printf("\nAdjacency Matrix:\n");

// Loop to print the adjacency matrix

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

{

for (int j = 0; j < nodes; j++)

{

printf("%d ", graph[i][j]); // Print the weight of the edge from node i to node j

}

printf("\n"); // New line for the next row in the matrix

}

}

// Main function to input and display the graph

void main()

{

int graph[MAX\_NODES][MAX\_NODES] = {0}; // Initialize graph with all values set to 0

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1 \*/**

int nodes, edges;

// Input the number of nodes and edges

printf("Enter the number of nodes: ");

scanf("%d", &nodes); // Read the number of nodes

printf("Enter the number of edges: ");

scanf("%d", &edges); // Read the number of edges

// Input the graph data (edges and weights)

input\_graph(graph, nodes, edges);

// Display the adjacency matrix of the graph

display\_graph(graph, nodes);

}

**/\*Name: – SnehaGoyal**

**University Roll No.: – 2023934**

**Section: – DS1 \*/**

snehagoyal@Desktop-36/ source.c

snehagoyal@Desktop-36/ ./a.out

**\*\*\*INPUT\*\*\***

Enter the number of nodes: 5

Enter the number of edges: 4

Enter edge 1 (source, destination, weight): 1 2 1

Enter edge 2 (source, destination, weight): 2 3 8

Enter edge 3 (source, destination, weight): 3 4 2

Enter edge 4 (source, destination, weight): 4 1 4

**\*\*\*OUTPUT\*\*\***

Adjacency Matrix:

0 0 0 0 0

0 0 1 0 4

0 1 0 8 0

0 0 8 0 2

0 4 0 2 0

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 10:- Write a C program to implement DFS and BFS. Make sure that the program is in the format I gave you, and it should be dynamic in nature and menu-driven. \*/**

#include <stdio.h>

#include <stdlib.h>

// Structure for a node in the adjacency list (for each vertex)

struct Node

{

int vertex;

struct Node\* next;

};

// Structure for the graph

struct Graph

{

int V; // Number of vertices

struct Node\*\* adjList; // Pointer to array of adjacency lists

};

// Function to create a new node in the adjacency list

struct Node\* createNode(int v)

{

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (!newNode)

{

printf("Memory allocation failed\n");

exit(1);

}

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

// Function to create a graph with V vertices

struct Graph\* createGraph(int V)

{

struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));

if (!graph)

{

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

printf("Memory allocation failed\n");

exit(1);

}

graph->V = V;

graph->adjList = (struct Node\*\*)malloc(V \* sizeof(struct Node\*));

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

{

graph->adjList[i] = NULL; // Initialize each adjacency list to NULL

}

return graph;

}

// Function to add an edge to the graph (undirected)

void addEdge(struct Graph\* graph, int src, int dest)

{

// Add edge from src to dest

struct Node\* newNode = createNode(dest);

newNode->next = graph->adjList[src];

graph->adjList[src] = newNode;

// As the graph is undirected, add the reverse edge as well

newNode = createNode(src);

newNode->next = graph->adjList[dest];

graph->adjList[dest] = newNode;

}

// Function to perform DFS

void DFS(struct Graph\* graph, int vertex, int\* visited)

{

printf("Visited %d\n", vertex);

visited[vertex] = 1;

struct Node\* adjList = graph->adjList[vertex];

while (adjList != NULL)

{

int connectedVertex = adjList->vertex;

if (!visited[connectedVertex]) // If the connected vertex is not visited

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

{

DFS(graph, connectedVertex, visited);

}

adjList = adjList->next;

}

}

// Function to perform BFS

void BFS(struct Graph\* graph, int startVertex)

{

int visited[graph->V];

for (int i = 0; i < graph->V; i++) // Initialize all vertices as not visited

{

visited[i] = 0;

}

int queue[graph->V];

int front = 0, rear = 0;

visited[startVertex] = 1;

queue[rear++] = startVertex; // Enqueue starting vertex

while (front < rear) // While there are vertices to be processed in queue

{

int currentVertex = queue[front++];

printf("Visited %d\n", currentVertex);

struct Node\* adjList = graph->adjList[currentVertex];

while (adjList != NULL) // Visit all adjacent vertices

{

int connectedVertex = adjList->vertex;

if (!visited[connectedVertex]) // If adjacent vertex not visited

{

visited[connectedVertex] = 1;

queue[rear++] = connectedVertex; // Enqueue adjacent vertex

}

adjList = adjList->next;

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

}

}

}

// Function to print the adjacency list (for debugging)

void printGraph(struct Graph\* graph)

{

for (int v = 0; v < graph->V; v++) // Loop through each vertex

{

struct Node\* temp = graph->adjList[v];

printf("\nVertex %d: ", v);

while (temp) // Print each node in the adjacency list

{

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("NULL\n");

}

}

// Main function

int main()

{

int choice, V, E, src, dest;

printf("\t\t\t\t\*\*\*\*\*INPUT\*\*\*\*\*\t\t\t\t\n");

printf("Enter the number of vertices: ");

scanf("%d", &V);

struct Graph\* graph = createGraph(V);

printf("\t\t\t\t\*\*\*\*\*OUTPUT\*\*\*\*\*\t\t\t\t\n");

int visited[V]; // Move visited array declaration outside of the switch block

do

{

printf("\nMenu:\n");

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

printf("1. Add an edge\n2. Perform DFS\n3. Perform BFS\n4. Display graph\n5. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter the source and destination vertices for the edge: ");

scanf("%d %d", &src, &dest);

if (src >= 0 && src < V && dest >= 0 && dest < V)

{

addEdge(graph, src, dest);

}

else

{

printf("Invalid vertices. Please enter values between 0 and %d.\n", V - 1);

}

break;

case 2:

printf("Enter the starting vertex for DFS: ");

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

{

visited[i] = 0; // Reset the visited array

}

scanf("%d", &src);

if (src >= 0 && src < V)

{

printf("\nDFS Traversal starting from vertex %d:\n", src);

DFS(graph, src, visited);

}

else

{

printf("Invalid vertex.\n");

}

break;

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

case 3:

printf("Enter the starting vertex for BFS: ");

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

{

visited[i] = 0; // Reset the visited array

}

scanf("%d", &src);

if (src >= 0 && src < V)

{

printf("\nBFS Traversal starting from vertex %d:\n", src);

BFS(graph, src);

}

else

{

printf("Invalid vertex.\n");

}

break;

case 4:

printGraph(graph);

break;

case 5:

printf("Exiting...\n");

break;

default:

printf("Invalid choice. Please try again.\n");

}

} while (choice != 5);

return 0;

}

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

snehagoyal@Desktop-36/ gcc bfsdfs.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

Enter the number of vertices: 5

\*\*\*\*\*OUTPUT\*\*\*\*\*

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 1

Enter the source and destination vertices for the edge: 0 1

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 1

Enter the source and destination vertices for the edge: 0 2

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 1

Enter the source and destination vertices for the edge: 1 3

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 1

Enter the source and destination vertices for the edge: 3

4

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 4

Vertex 0: 2 -> 1 -> NULL

Vertex 1: 3 -> 0 -> NULL

Vertex 2: 0 -> NULL

Vertex 3: 4 -> 1 -> NULL

Vertex 4: 3 -> NULL

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 2

Enter the starting vertex for DFS: 0

DFS Traversal starting from vertex 0:

Visited 0

Visited 2

**/\*Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

Visited 1

Visited 3

Visited 4

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 3

Enter the starting vertex for BFS: 0

BFS Traversal starting from vertex 0:

Visited 0

Visited 2

Visited 1

Visited 3

Visited 4

Menu:

1. Add an edge

2. Perform DFS

3. Perform BFS

4. Display graph

5. Exit

Enter your choice: 5

Exiting...

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1**

**Question 11:- Write a C program implement Kurskal’s algorithm to find minimal spanning tree from a given graph.\*/**

#include <stdio.h>

#include <stdlib.h>

// Structure to represent an edge in the graph

struct Edge

{

int src, dest, weight; // Source, destination, and weight of the edge

};

// Structure to represent the graph

struct Graph

{

int V, E; // Number of vertices and edges

struct Edge\* edges; // Array of edges

};

// Function to create a graph with V vertices and E edges

struct Graph\* newGraph(int V, int E)

{

struct Graph\* g = (struct Graph\*)malloc(sizeof(struct Graph));

g->V = V;

g->E = E;

g->edges = (struct Edge\*)malloc(E \* sizeof(struct Edge)); // Dynamically allocate memory for edges

return g;

}

// Find function to find the set (or parent) of an element i

int find(int parent[], int i)

{

if (parent[i] != i)

{

parent[i] = find(parent, parent[i]); // Path compression

}

return parent[i];

}

// Union function to perform union of two sets x and y

void unionSets(int parent[], int x, int y)

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

{

int rootX = find(parent, x);

int rootY = find(parent, y);

parent[rootX] = rootY;

}

// Function to sort edges based on weight using bubble sort

void sortEdges(struct Edge edges[], int E)

{

for (int i = 0; i < E - 1; i++)

{

for (int j = 0; j < E - i - 1; j++)

{

if (edges[j].weight > edges[j + 1].weight)

{

struct Edge temp = edges[j];

edges[j] = edges[j + 1];

edges[j + 1] = temp;

}

}

}

}

// Function to implement Kruskal's algorithm for Minimum Spanning Tree

void kruskal(struct Graph\* g)

{

int V = g->V;

struct Edge result[V]; // Array to store the MST

int e = 0; // Count of edges in MST

int i = 0; // Index variable for sorted edges

sortEdges(g->edges, g->E);

int\* parent = (int\*)malloc(V \* sizeof(int)); // Array to store the parent of each vertex

for (int v = 0; v < V; v++)

{

parent[v] = v; // Initially, each vertex is its own parent

}

// Process each edge in the sorted order

while (e < V - 1 && i < g->E)

{

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

struct Edge nextEdge = g->edges[i++]; // Get the next edge

int x = find(parent, nextEdge.src); // Find the set of the source vertex

int y = find(parent, nextEdge.dest); // Find the set of the destination vertex

// If source and destination are not in the same set, include this edge in the MST

if (x != y)

{

result[e++] = nextEdge; // Add edge to result MST

unionSets(parent, x, y); // Perform union of sets

}

}

// Print the edges of the MST

printf("Edges in the Minimum Spanning Tree:\n");

for (i = 0; i < e; i++)

{

printf("%d -- %d == %d\n", result[i].src, result[i].dest, result[i].weight);

}

free(parent); // Free dynamically allocated memory for parent array

}

// Function to input graph details from the user

void input(struct Graph\* g)

{

printf("Enter the edges (source, destination, weight):\n");

for (int i = 0; i < g->E; i++)

{

// Get source, destination, and weight for each edge

scanf("%d %d %d", &g->edges[i].src, &g->edges[i].dest, &g->edges[i].weight);

}

}

// Main function that provides a menu-driven interface

int main()

{

int V = 0, E = 0; // Variables to store the number of vertices and edges

struct Graph\* g = NULL; // Pointer to store the graph

int choice; // Variable to store the menu choice

// Infinite loop to display menu options

// Display the menu

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

printf("\t\t\t\t\*\*\*\*\*INPUT\*\*\*\*\*\t\t\t\t\n");

printf("\nMenu:\n 1. Input graph details\n 2. Run Kruskal's MST algorithm\n 3. Exit\n ");

printf("\t\t\t\t\*\*\*\*\*OUTPUT\*\*\*\*\*\t\t\t\t\n");

while (1)

{

printf("Enter your choice: ");

scanf("%d", &choice); // Get user's choice

// Switch case to handle different menu choices

switch (choice)

{

case 1:

// Input graph details if choice is 1

printf("Enter the number of vertices: ");

scanf("%d", &V);

printf("Enter the number of edges: ");

scanf("%d", &E);

// Free any previously allocated memory for the graph

if (g != NULL)

{

free(g->edges);

free(g);

}

// Create a new graph and input the edges

g = newGraph(V, E);

input(g);

break;

case 2:

// Run Kruskal's algorithm if choice is 2

if (g == NULL)

{

printf("Graph has not been created yet.\n");

}

else

{

kruskal(g); // Execute Kruskal's algorithm

}

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

break;

case 3:

// Exit the program if choice is 3

if (g != NULL)

{

free(g->edges);

free(g); // Free memory before exiting

}

printf("Exiting program.\n");

return 0;

default:

// Handle invalid input

printf("Invalid choice, please try again.\n");

}

}

return 0;

}

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

snehagoyal@Desktop-36/ gcc kruskal.c

snehagoyal@Desktop-36/./a.out

\*\*\*\*\*INPUT\*\*\*\*\*

Menu:

1. Input graph details

2. Run Kruskal's MST algorithm

3. Exit

\*\*\*\*\*OUTPUT\*\*\*\*\*

Enter your choice: 1

Enter the number of vertices: 5

Enter the number of edges: 6

Enter the edges (source, destination, weight):

0

4

12

4

2

7

2

1

2

1

3

5

3

0

6

2

**/\* Name:- SnehaGoyal**

**University Rollno.:- 2023934**

**Section:- DS1 \*/**

3

3

Enter your choice: 2

Edges in the Minimum Spanning Tree:

2 -- 1 == 2

2 -- 3 == 3

3 -- 0 == 6

4 -- 2 == 7