**SAVEETHA SCHOOL OF ENGINEERING**

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Academic Year 2022-23**

**Course Code – CSA03 43 Course Name - DATA STRUCTURES**

|  |  |
| --- | --- |
|  | **PROGRAM** |
|  | Write a C program to perform Matrix Multiplication  #include<stdio.h>  int main(){  int a[10][10],b[10][10],mul[10][10],r,c,i,j,k;  printf("Enter the number of row :");  scanf("%d",&r);  printf("Enter the number of column :");  scanf("%d",&c);  printf("Enter the elements of first :\n");  for(i=0;i<r;i++){  for(j=0;j<c;j++){  scanf("%d",&a[i][j]);  }  }  printf("Enter the elements of second matrix :\n");  for(i=0;i<r;i++){  for(j=0;j<c;j++){  scanf("%d",&b[i][j]);  }  }  for(i=0;i<r;i++){  for(j=0;j<c;j++){  mul[i][j]=0;  for(k=0;k<c;k++){  mul[i][j]+=a[i][k]\*b[k][j];  }  }  }  for(i=0;i<r;i++){  for(j=0;j<c;j++){  printf("%d\t",mul[i][j]);  }  printf("\n");  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 123159.png |
|  | Write a C program to find Odd or Even number from a given set of numbers  #include <stdio.h>  int main()  {  int n;  printf("Enter number of elements in the array: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements in the array: \n",n);  for(int i=0;i<n;i++){  scanf("%d",&arr[i]);  }  printf("Even numbers in the array are: ");  for(int i=0;i<n;i++){  if(arr[i]%2==0)  printf("\n%d ", arr[i]);  }  printf("\nOdd numbers in the array are: ");  for(int i=0;i<n;i++){  if(arr[i]%2==1)  printf("\n%d ", arr[i]);  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 123400.png |
|  | Write a C program to find Factorial of a given number without using Recursion  #include<stdio.h>  int main(){  int i,f=1, num;  printf("Enter a number: ");  scanf("%d", &num);  if((num==1)||(num==0)){  f=1;  }  else{  for(i=1;i<=num;i++){  f = f \* i;  }  printf("%d! = %d\n", num, f);  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 123524.png |
|  | Write a C program to find Fibonacci series without using Recursion  #include <stdio.h>  int main(){  int i, n, firstTerm=0, secondTerm=1, sum=0;  printf("\nEnter number of terms required in Fibonacci Series: ");  scanf("%d",&n);  if(n<=0){  printf("Error!! No Elements!!");  }  else if(n==1){  printf("\nFibonacci Series is:\n %d ", firstTerm);  }  else if (n==2){  printf("\nFibonacci Series is:\n\n\n %d %d ", firstTerm, secondTerm);  }  else{  i=2;  printf("\nFibonacci Series is:\n %d %d ", firstTerm, secondTerm);  while (i<n) {  sum=firstTerm+secondTerm;  firstTerm=secondTerm;  secondTerm=sum;  ++i;  printf("%d ",sum);  }  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 124041.png |
|  | Write a C program to find Factorial of a given number using Recursion  #include<stdio.h>  #include<conio.h>  int factorial(int number){  if(number==0 || number==1)  return 1;  else  return(number \* factorial(number-1));  }  int main(){  int num,f;  printf("\nEnter the number: ");  scanf("%d",&num);  f=factorial(num);  printf("\nThe factorial of the number %d is %d",num,f);  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 124127.png |
|  | Write a C program to find Fibonacci series using Recursion  #include<stdio.h>  int Fibonacci(int n){  if(n==0)  return 0;  else if(n==1)  return 1;  else  return (Fibonacci(n-1)+Fibonacci(n-2) );  }  int main(){  int n,i=0;  printf("Enter the number of terms in the series : ");  scanf("%d",&n);  printf("Fibonacci series\n");  for (int c=1;c<=n;c++){  printf("%d\n", Fibonacci(i));  i++;  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 124228.png |
|  | Write a C program to implement Array operations such as Insert, Delete and Display  #include <stdio.h>  void displayArray(int arr[], int size) {  printf("Array elements: ");  for (int i = 0; i < size; i++) {  printf("%d ", arr[i]);  }  printf("\n");  }  void insertElement(int arr[], int \*size, int position, int element) {  if (\*size >= position) {  for (int i = \*size; i > position; i--) {  arr[i] = arr[i - 1];  }  arr[position] = element;  (\*size)++;  printf("Element inserted successfully.\n");  } else {  printf("Invalid position for insertion.\n");  }  }  void deleteElement(int arr[], int \*size, int position) {  if (\*size > 0 && position >= 0 && position < \*size) {  for (int i = position; i < (\*size - 1); i++) {  arr[i] = arr[i + 1];  }  (\*size)--;  printf("Element deleted successfully.\n");  } else {  printf("Invalid position for deletion.\n");  }  }  int searchElement(int arr[], int size, int element) {  for (int i = 0; i < size; i++) {  if (arr[i] == element) {  return i;  }  }  return -1;  }  int main() {  int arr[100];  int size = 0;  printf("Enter the number of elements: ");  scanf("%d", &size);  printf("Enter the elements:\n");  for (int i = 0; i < size; i++) {  scanf("%d", &arr[i]);  }  displayArray(arr, size);  int newPosition, newValue;  printf("Enter the position and value to insert: ");  scanf("%d %d", &newPosition, &newValue);  insertElement(arr, &size, newPosition, newValue);  displayArray(arr, size);  int deletePosition;  printf("Enter the position to delete: ");  scanf("%d", &deletePosition);  deleteElement(arr, &size, deletePosition);  displayArray(arr, size);  int searchValue;  printf("Enter the value to search: ");  scanf("%d", &searchValue);  int searchResult = searchElement(arr, size, searchValue);  if (searchResult != -1) {  printf("Element %d found at position %d.\n", searchValue, searchResult);  } else {  printf("Element not found.\n");  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 125001.png |
|  | Write a C program to search a number using Linear Search method  #include <stdio.h>  int bubbleSort(int arr[], int size) {  for (int i = 0; i < size - 1; i++) {  for (int j = 0; j < size - i - 1; j++) {  if (arr[j] > arr[j + 1]) {  int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;}  }  }  }  int linearSearch(int arr[], int size, int target) {  for (int i = 0; i < size; i++) {  if (arr[i] == target) {  return i;  }  }  return -1;  }  int main() {  int n;  printf("Enter number of elements in the array: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements in the array: \n",n);  for(int i=0;i<n;i++){  scanf("%d",&arr[i]);  }  int size = sizeof(arr)/sizeof(arr[0]);  bubbleSort(arr, size);  printf("Sorted array: ");  for (int i = 0; i < size; i++) {  printf("%d ", arr[i]);  }  printf("\n");  int target;  printf("Enter the number to search: ");  scanf("%d", &target);  int result = linearSearch(arr, size, target);  if (result != -1) {  printf("Number %d found at index %d\n", target, result);  } else {  printf("Number %d not found in the array\n", target);  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 125314.png |
|  | Write a C program to search a number using Binary Search method  #include <stdio.h>  int bubbleSort(int arr[], int size) {  for (int i = 0; i < size - 1; i++) {  for (int j = 0; j < size - i - 1; j++) {  if (arr[j] > arr[j + 1]) {  int temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  }  }  int binarySearch(int arr[], int left, int right, int target) {  while (left <= right) {  int mid = left + (right - left) / 2;  if (arr[mid] == target) {  return mid;  } else if (arr[mid] < target) {  left = mid + 1;  } else {  right = mid - 1;  }  }  return -1;  }  int main() {  int n;  printf("Enter number of elements in the array: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements in the array: \n",n);  for(int i=0;i<n;i++){  scanf("%d",&arr[i]);  }  int size = sizeof(arr) / sizeof(arr[0]);  bubbleSort(arr, size);  printf("Sorted array: ");  for (int i = 0; i < size; i++) {  printf("%d ", arr[i]);  }  printf("\n");  int target;  printf("Enter the number to search: ");  scanf("%d", &target);  int result = binarySearch(arr, 0, size - 1, target);  if (result != -1) {  printf("Number %d found at index %d\n", target, result);  } else {  printf("Number %d not found in the array\n", target);  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 125436.png |
|  | Write a C program to implement Linked list operations  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*next;  };  int insertAtBeginning(struct Node \*\*head, int data) {  struct Node \*newNode = (struct Node \*)malloc(sizeof(struct Node));  newNode->data = data;  newNode->next = \*head;  \*head = newNode;  }  void deleteNode(struct Node \*\*head, int key) {  struct Node \*temp = \*head, \*prev;  if (temp != NULL && temp->data == key) {  \*head = temp->next;  free(temp);  return ;  }  while (temp != NULL && temp->data != key) {  prev = temp;  temp = temp->next;  }  if (temp == NULL)  return;  prev->next = temp->next;  free(temp);  }  int display(struct Node \*node) {  while (node != NULL) {  printf("%d -> ", node->data);  node = node->next;  }  printf("NULL\n");  }  int main() {  struct Node \*head = NULL;  int choice, data, key;  while (1) {  printf("1. Insert node\n");  printf("2. Delete node\n");  printf("3. Display linked list\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter data to insert: ");  scanf("%d", &data);  insertAtBeginning(&head, data);  break;  case 2:  printf("Enter key to delete: ");  scanf("%d", &key);  deleteNode(&head, key);  break;  case 3:  display(head);  break;  case 4:  exit(0);  default:  printf("Invalid choice!\n");  }  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 125609.png |
|  | Write a C program to implement Stack operations such as PUSH,  POP and PEEK  #include <stdio.h>  #define MAX\_SIZE 100  struct Stack {  int arr[MAX\_SIZE];  int top;  };  int initialize(struct Stack \*s) {  s->top = -1;  }  int isEmpty(struct Stack \*s) {  return s->top == -1;  }  int isFull(struct Stack \*s) {  return s->top == MAX\_SIZE - 1;  }  int push(struct Stack \*s, int value) {  if (isFull(s)) {  printf("Stack overflow\n");  } else {  s->top++;  s->arr[s->top] = value;  printf("Pushed %d onto the stack\n", value);  }  }  int pop(struct Stack \*s) {  if (isEmpty(s)) {  printf("Stack underflow\n");  return -1;  } else {  int value = s->arr[s->top];  s->top--;  return value;  }  }  int peek(struct Stack \*s) {  if (isEmpty(s)) {  printf("Stack is empty\n");  return -1;  } else {  return s->arr[s->top];  }  }  int main() {  struct Stack stack;  initialize(&stack);  while (1) {  int choice, value;  printf("\nStack Operations:\n");  printf("1. Push\n");  printf("2. Pop\n");  printf("3. Peek\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter value to push: ");  scanf("%d", &value);  push(&stack, value);  break;  case 2:  value = pop(&stack);  if (value != -1)  printf("Popped: %d\n", value);  break;  case 3:  value = peek(&stack);  if (value != -1)  printf("Top element: %d\n", value);  break;  case 4:  printf("Exiting...\n");  return 0;  default:  printf("Invalid choice\n");  }  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 125751.png |
|  | Write a C program to implement the application of Stack (Notations)  #include <stdio.h>  #include <stdlib.h>  #include <ctype.h>  #define MAX\_SIZE 100  struct Stack {  int top;  char items[MAX\_SIZE];  };  void push(struct Stack \*stack, char value) {  if (stack->top == MAX\_SIZE - 1) {  printf("Stack is full. Cannot push.\n");  } else {  stack->top++;  stack->items[stack->top] = value;  }  }  char pop(struct Stack \*stack) {  if (stack->top == -1) {  printf("Stack is empty. Cannot pop.\n");  return '\0';  } else {  char value = stack->items[stack->top];  stack->top--;  return value;  }  }  int precedence(char op) {  if (op == '+' || op == '-')  return 1;  if (op == '\*' || op == '/')  return 2;  return 0;  }  void infixToPostfix(char infix[], char postfix[]) {  struct Stack operatorStack;  operatorStack.top = -1;  int postfixIndex = 0;  for (int i = 0; infix[i] != '\0'; i++) {  if (isalnum(infix[i])) {  postfix[postfixIndex++] = infix[i];  } else if (infix[i] == '(') {  push(&operatorStack, '(');  } else if (infix[i] == ')') {  while (operatorStack.top != -1 && operatorStack.items[operatorStack.top] != '(') {  postfix[postfixIndex++] = pop(&operatorStack);  }  pop(&operatorStack);  } else {  while (operatorStack.top != -1 && precedence(infix[i]) <= precedence(operatorStack.items[operatorStack.top])) {  postfix[postfixIndex++] = pop(&operatorStack);  }  push(&operatorStack, infix[i]);  }  }  while (operatorStack.top != -1) {  postfix[postfixIndex++] = pop(&operatorStack);  }  postfix[postfixIndex] = '\0';  }  int main() {  char infix[MAX\_SIZE];  char postfix[MAX\_SIZE];  printf("Enter an infix expression: ");  scanf("%s", infix);  infixToPostfix(infix, postfix);  printf("Postfix expression: %s\n", postfix);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 125941.png |
|  | Write a C program to implement Queue operations such as ENQUEUE, DEQUEUE and Display  #include <stdio.h>  #include <stdlib.h>  #define MAX\_SIZE 10  struct Queue {  int items[MAX\_SIZE];  int front;  int rear;  };  typedef struct Queue Queue;  void initializeQueue(Queue \*q) {  q->front = -1;  q->rear = -1;  }  int isFull(Queue \*q) {  return (q->rear == MAX\_SIZE - 1);  }  int isEmpty(Queue \*q) {  return (q->front == -1);  }  void enqueue(Queue \*q, int value) {  if (isFull(q)) {  printf("Queue is full. Cannot enqueue.\n");  return;  }  if (isEmpty(q)) {  q->front = 0;  }  q->rear++;  q->items[q->rear] = value;  printf("%d enqueued to the queue.\n", value);  }  int dequeue(Queue \*q) {  int item;  if (isEmpty(q)) {  printf("Queue is empty. Cannot dequeue.\n");  return -1;  }  item = q->items[q->front];  q->front++;  if (q->front > q->rear) {  initializeQueue(q);  }  return item;  }  void display(Queue \*q) {  if (isEmpty(q)) {  printf("Queue is empty.\n");  return;  }  printf("Queue elements: ");  for (int i = q->front; i <= q->rear; i++) {  printf("%d ", q->items[i]);  }  printf("\n");  }  int main() {  Queue q;  initializeQueue(&q);  int choice, value;  while (1) {  printf("\nQueue Operations:\n");  printf("1. Enqueue\n");  printf("2. Dequeue\n");  printf("3. Display\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter value to enqueue: ");  scanf("%d", &value);  enqueue(&q, value);  break;  case 2:  printf("Dequeued: %d\n", dequeue(&q));  break;  case 3:  display(&q);  break;  case 4:  printf("Exiting the program.\n");  exit(0);  default:  printf("Invalid choice. Please enter a valid option.\n");  }  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 130056.png |
|  | Write a C program to implement the Tree Traversals (Inorder, Preorder, Postorder)  #include <stdio.h>  #include <stdlib.h>  struct TreeNode {  int data;  struct TreeNode\* left;  struct TreeNode\* right;  };  struct TreeNode\* createNode(int data) {  struct TreeNode\* newNode = (struct TreeNode\*)malloc(sizeof(struct TreeNode));  newNode->data = data;  newNode->left = NULL;  newNode->right = NULL;  return newNode;  }  struct TreeNode\* insertNode(struct TreeNode\* root, int data) {  if (root == NULL)  return createNode(data);  if (data < root->data)  root->left = insertNode(root->left, data);  else if (data > root->data)  root->right = insertNode(root->right, data);  return root;  }  void inorderTraversal(struct TreeNode\* root) {  if (root == NULL)  return;  inorderTraversal(root->left);  printf("%d ", root->data);  inorderTraversal(root->right);  }  void preorderTraversal(struct TreeNode\* root) {  if (root == NULL)  return;  printf("%d ", root->data);  preorderTraversal(root->left);  preorderTraversal(root->right);  }  void postorderTraversal(struct TreeNode\* root) {  if (root == NULL)  return;  postorderTraversal(root->left);  postorderTraversal(root->right);  printf("%d ", root->data);  }  int main() {  struct TreeNode\* root = NULL;  int numNodes, value;  printf("Enter the number of nodes: ");  scanf("%d", &numNodes);  printf("Enter the values of the nodes:\n");  for (int i = 0; i < numNodes; ++i) {  scanf("%d", &value);  root = insertNode(root, value);  }  printf("Inorder traversal: ");  inorderTraversal(root);  printf("\n");  printf("Preorder traversal: ");  preorderTraversal(root);  printf("\n");  printf("Postorder traversal: ");  postorderTraversal(root);  printf("\n");  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 130224.png |
|  | Write a C program to implement hashing using Linear Probing method  #include <stdio.h>  #define SIZE 10  int hash(int key) {  return key % SIZE;  }  void insert(int hashTable[], int key) {  int index = hash(key);  while (hashTable[index] != -1) {  index = (index + 1) % SIZE;  }  hashTable[index] = key;  }  void display(int hashTable[]) {  printf("Hash Table:\n");  for (int i = 0; i < SIZE; i++) {  printf("%d: %d\n", i, hashTable[i]);  }  }  int main() {  int hashTable[SIZE];  for (int i = 0; i < SIZE; i++) {  hashTable[i] = -1;  }  int numValues;  printf("Enter the number of values to insert: ");  scanf("%d", &numValues);  for (int i = 0; i < numValues; i++) {  int value;  printf("Enter value %d: ", i + 1);  scanf("%d", &value);  insert(hashTable, value);  }  display(hashTable);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 130418.png |
|  | Write a C program to arrange a series of numbers using Insertion Sort  #include <stdio.h>  void insertionSort(int arr[], int n) {  for (int i = 1; i < n; i++) {  int key = arr[i];  int j = i - 1;  while (j >= 0 && arr[j] > key) {  arr[j + 1] = arr[j];  j = j - 1;  }  arr[j + 1] = key;  }  }  void printArray(int arr[], int n) {  for (int i = 0; i < n; i++) {  printf("%d ", arr[i]);  }  printf("\n");  }  int main() {  int n;  printf("Enter the number of elements: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements:\n", n);  for (int i = 0; i < n; i++) {  scanf("%d", &arr[i]);  }  printf("Original array: ");  printArray(arr, n);  insertionSort(arr, n);  printf("Sorted array: ");  printArray(arr, n);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 130533.png |
|  | Write a C program to arrange a series of numbers using Merge Sort  #include <stdio.h>  void merge(int arr[], int left, int mid, int right) {  int n1 = mid - left + 1;  int n2 = right - mid;  int L[n1], R[n2];  for (int i = 0; i < n1; i++) {  L[i] = arr[left + i];  }  for (int j = 0; j < n2; j++) {  R[j] = arr[mid + 1 + j];  }  int i = 0, j = 0, k = left;  while (i < n1 && j < n2) {  if (L[i] <= R[j]) {  arr[k] = L[i];  i++;  } else {  arr[k] = R[j];  j++;  }  k++;  }  while (i < n1) {  arr[k] = L[i];  i++;  k++;  }  while (j < n2) {  arr[k] = R[j];  j++;  k++;  }  }  void mergeSort(int arr[], int left, int right) {  if (left < right) {  int mid = left + (right - left) / 2;  mergeSort(arr, left, mid);  mergeSort(arr, mid + 1, right);  merge(arr, left, mid, right);  }  }  void printArray(int arr[], int n) {  for (int i = 0; i < n; i++) {  printf("%d ", arr[i]);  }  printf("\n");  }  int main() {  int n;  printf("Enter the number of elements: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements:\n", n);  for (int i = 0; i < n; i++) {  scanf("%d", &arr[i]);  }  printf("Original array: ");  printArray(arr, n);  mergeSort(arr, 0, n - 1);  printf("Sorted array: ");  printArray(arr, n);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 130706.png |
|  | Write a C program to arrange a series of numbers using Quick Sort  #include <stdio.h>  void swap(int\* a, int\* b) {  int temp = \*a;  \*a = \*b;  \*b = temp;  }  int partition(int arr[], int low, int high) {  int pivot = arr[high];  int i = (low - 1);  for (int j = low; j <= high - 1; j++) {  if (arr[j] < pivot) {  i++;  swap(&arr[i], &arr[j]);  }  }  swap(&arr[i + 1], &arr[high]);  return (i + 1);  }  void quickSort(int arr[], int low, int high) {  if (low < high) {  int pi = partition(arr, low, high);  quickSort(arr, low, pi - 1);  quickSort(arr, pi + 1, high);  }  }  void printArray(int arr[], int size) {  for (int i = 0; i < size; i++) {  printf("%d ", arr[i]);  }  printf("\n");  }  int main() {  int n;  printf("Enter the number of elements: ");  scanf("%d", &n);  int arr[n];  printf("Enter %d elements:\n", n);  for (int i = 0; i < n; i++) {  scanf("%d", &arr[i]);  }  printf("Original array: ");  printArray(arr, n);  quickSort(arr, 0, n - 1);  printf("Sorted array: ");  printArray(arr, n);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 130812.png |
|  | Write a C program to implement Heap sort  #include <stdio.h>  void heapify(int arr[], int n, int i) {  int largest = i;  int left = 2 \* i + 1;  int right = 2 \* i + 2;  if (left < n && arr[left] > arr[largest])  largest = left;  if (right < n && arr[right] > arr[largest])  largest = right;  if (largest != i) {  int temp = arr[i];  arr[i] = arr[largest];  arr[largest] = temp;  heapify(arr, n, largest);  }  }  void heapSort(int arr[], int n) {  for (int i = n / 2 - 1; i >= 0; i--)  heapify(arr, n, i);  for (int i = n - 1; i > 0; i--) {  int temp = arr[0];  arr[0] = arr[i];  arr[i] = temp;  heapify(arr, i, 0);  }  }  int main() {  int n;  printf("Enter the number of elements: ");  scanf("%d", &n);  int arr[n];  printf("Enter the elements:\n");  for (int i = 0; i < n; i++)  scanf("%d", &arr[i]);  printf("Original array: ");  for (int i = 0; i < n; i++)  printf("%d ", arr[i]);  heapSort(arr, n);  printf("\nSorted array: ");  for (int i = 0; i < n; i++)  printf("%d ", arr[i]);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 131101.png |
|  | Write a program to perform the following operations:  a) Insert an element into a AVL tree  b) Delete an element from a AVL tree  c) Search for a key element in a AVL tree  #include <stdio.h>  #include <stdlib.h>  struct Node {  int data;  struct Node \*left;  struct Node \*right;  int height;  };  int max(int a, int b) {  return (a > b) ? a : b;  }  int getHeight(struct Node \*node) {  if (node == NULL)  return -1;  return node->height;  }  int getBalance(struct Node \*node) {  if (node == NULL)  return 0;  return getHeight(node->left) - getHeight(node->right);  }  struct Node \*newNode(int data) {  struct Node \*node = (struct Node \*)malloc(sizeof(struct Node));  node->data = data;  node->left = NULL;  node->right = NULL;  node->height = 0;  return node;  }  struct Node \*rightRotate(struct Node \*y) {  struct Node \*x = y->left;  struct Node \*T2 = x->right;  x->right = y;  y->left = T2;  y->height = max(getHeight(y->left), getHeight(y->right)) + 1;  x->height = max(getHeight(x->left), getHeight(x->right)) + 1;  return x;  }  struct Node \*leftRotate(struct Node \*x) {  struct Node \*y = x->right;  struct Node \*T2 = y->left;  y->left = x;  x->right = T2;  x->height = max(getHeight(x->left), getHeight(x->right)) + 1;  y->height = max(getHeight(y->left), getHeight(y->right)) + 1;  return y;  }  struct Node \*insert(struct Node \*root, int data) {  if (root == NULL)  return newNode(data);  if (data < root->data)  root->left = insert(root->left, data);  else if (data > root->data)  root->right = insert(root->right, data);  else  return root;  root->height = 1 + max(getHeight(root->left), getHeight(root->right));  int balance = getBalance(root);  if (balance > 1 && data < root->left->data)  return rightRotate(root);  if (balance < -1 && data > root->right->data)  return leftRotate(root);  if (balance > 1 && data > root->left->data) {  root->left = leftRotate(root->left);  return rightRotate(root);  }  if (balance < -1 && data < root->right->data) {  root->right = rightRotate(root->right);  return leftRotate(root);  }  return root;  }  struct Node \*minValueNode(struct Node \*node) {  struct Node \*current = node;  while (current->left != NULL)  current = current->left;  return current;  }  struct Node \*deleteNode(struct Node \*root, int data) {  if (root == NULL)  return root;  if (data < root->data)  root->left = deleteNode(root->left, data);  else if (data > root->data)  root->right = deleteNode(root->right, data);  else {  if (root->left == NULL || root->right == NULL) {  struct Node \*temp = root->left ? root->left : root->right;  if (temp == NULL) {  temp = root;  root = NULL;  } else  \*root = \*temp;  free(temp);  } else {  struct Node \*temp = minValueNode(root->right);  root->data = temp->data;  root->right = deleteNode(root->right, temp->data);  }  }  if (root == NULL)  return root;  root->height = 1 + max(getHeight(root->left), getHeight(root->right));  int balance = getBalance(root);  if (balance > 1 && getBalance(root->left) >= 0)  return rightRotate(root);  if (balance < -1 && getBalance(root->right) <= 0)  return leftRotate(root);  if (balance > 1 && getBalance(root->left) < 0) {  root->left = leftRotate(root->left);  return rightRotate(root);  }  if (balance < -1 && getBalance(root->right) > 0) {  root->right = rightRotate(root->right);  return leftRotate(root);  }  return root;  }  void inorderTraversal(struct Node \*root) {  if (root != NULL) {  inorderTraversal(root->left);  printf("%d ", root->data);  inorderTraversal(root->right);  }  }  int main() {  struct Node \*root = NULL;  int choice, data;  while (1) {  printf("\nAVL Tree Operations:\n");  printf("1. Insert\n");  printf("2. Delete\n");  printf("3. Display Inorder\n");  printf("4. Exit\n");  printf("Enter your choice: ");  scanf("%d", &choice);  switch (choice) {  case 1:  printf("Enter data to insert: ");  scanf("%d", &data);  root = insert(root, data);  break;  case 2:  printf("Enter data to delete: ");  scanf("%d", &data);  root = deleteNode(root, data);  break;  case 3:  printf("Inorder traversal of AVL tree: ");  inorderTraversal(root);  printf("\n");  break;  case 4:  exit(0);  default:  printf("Invalid choice\n");  }  }  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 131311.png  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 131423.png |
|  | Write a C program to Graph traversal using Breadth First Search  #include <stdio.h>  #include <stdlib.h>  #define MAX\_VERTICES 100  struct Queue {  int items[MAX\_VERTICES];  int front;  int rear;  };  struct Graph {  int vertices;  int\*\* adjacencyMatrix;  };  struct Queue\* createQueue() {  struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));  queue->front = -1;  queue->rear = -1;  return queue;  }  int isEmpty(struct Queue\* queue) {  return queue->rear == -1;  }  void enqueue(struct Queue\* queue, int value) {  if (queue->rear == MAX\_VERTICES - 1)  printf("Queue is full\n");  else {  if (queue->front == -1)  queue->front = 0;  queue->rear++;  queue->items[queue->rear] = value;  }  }  int dequeue(struct Queue\* queue) {  int item;  if (isEmpty(queue)) {  printf("Queue is empty\n");  item = -1;  } else {  item = queue->items[queue->front];  queue->front++;  if (queue->front > queue->rear) {  queue->front = queue->rear = -1;  }  }  return item;  }  struct Graph\* createGraph(int vertices) {  struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));  graph->vertices = vertices;  graph->adjacencyMatrix = (int\*\*)malloc(vertices \* sizeof(int\*));  for (int i = 0; i < vertices; i++) {  graph->adjacencyMatrix[i] = (int\*)malloc(vertices \* sizeof(int));  for (int j = 0; j < vertices; j++) {  graph->adjacencyMatrix[i][j] = 0;  }  }  return graph;  }  void addEdge(struct Graph\* graph, int src, int dest) {  graph->adjacencyMatrix[src][dest] = 1;  graph->adjacencyMatrix[dest][src] = 1;  }  void BFS(struct Graph\* graph, int startVertex) {  struct Queue\* queue = createQueue();  int visited[MAX\_VERTICES] = {0};  printf("Breadth First Traversal starting from vertex %d:\n", startVertex);  printf("%d ", startVertex);  visited[startVertex] = 1;  enqueue(queue, startVertex);  while (!isEmpty(queue)) {  int currentVertex = dequeue(queue);  for (int i = 0; i < graph->vertices; i++) {  if (graph->adjacencyMatrix[currentVertex][i] == 1 && !visited[i]) {  printf("%d ", i);  visited[i] = 1;  enqueue(queue, i);  }  }  }  printf("\n");  }  int main() {  int vertices, edges;  printf("Enter the number of vertices: ");  scanf("%d", &vertices);  struct Graph\* graph = createGraph(vertices);  printf("Enter the number of edges: ");  scanf("%d", &edges);  for (int i = 0; i < edges; i++) {  int src, dest;  printf("Enter edge %d (source destination): ", i + 1);  scanf("%d %d", &src, &dest);  addEdge(graph, src, dest);  }  int startVertex;  printf("Enter the starting vertex for BFS: ");  scanf("%d", &startVertex);  BFS(graph, startVertex);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 131647.png |
|  | Write a C program to Graph traversal using Depth First Search  #include <stdio.h>  #include <stdbool.h>  #define MAX\_NODES 100  struct Graph {  int vertices;  int adjMatrix[MAX\_NODES][MAX\_NODES];  };  void initGraph(struct Graph \*graph, int vertices) {  graph->vertices = vertices;  for (int i = 0; i < vertices; i++) {  for (int j = 0; j < vertices; j++) {  graph->adjMatrix[i][j] = 0;  }  }  }  void addEdge(struct Graph \*graph, int source, int destination) {  graph->adjMatrix[source][destination] = 1;  graph->adjMatrix[destination][source] = 1;  }  void DFS(struct Graph \*graph, int vertex, bool visited[]) {  visited[vertex] = true;  printf("%d ", vertex);  for (int i = 0; i < graph->vertices; i++) {  if (graph->adjMatrix[vertex][i] == 1 && !visited[i]) {  DFS(graph, i, visited);  }  }  }  int main() {  struct Graph graph;  int vertices, edges;  int source, destination;  printf("Enter the number of vertices: ");  scanf("%d", &vertices);  initGraph(&graph, vertices);  printf("Enter the number of edges: ");  scanf("%d", &edges);  for (int i = 0; i < edges; i++) {  printf("Enter edge %d (source destination): ", i + 1);  scanf("%d %d", &source, &destination);  addEdge(&graph, source, destination);  }  bool visited[MAX\_NODES] = {false};  printf("Depth First Search traversal starting from vertex 0: ");  DFS(&graph, 0, visited);  return 0;  } |
|  | Implementation of Shortest Path Algorithms using Dijkstra’s Algorithm  #include <stdio.h>  #include <limits.h>  #define V 10  int minDistance(int dist[], int visited[], int vertices) {  int min = INT\_MAX, min\_index;  for (int v = 0; v < vertices; v++) {  if (visited[v] == 0 && dist[v] <= min) {  min = dist[v];  min\_index = v;  }  }  return min\_index;  }  void printSolution(int dist[], int vertices) {  printf("Vertex Distance from Source\n");  for (int i = 0; i < vertices; i++) {  printf("%d %d\n", i, dist[i]);  }  }  void dijkstra(int graph[V][V], int src, int vertices) {  int dist[V];  int visited[V];  for (int i = 0; i < vertices; i++) {  dist[i] = INT\_MAX;  visited[i] = 0;  }  dist[src] = 0;  for (int count = 0; count < vertices - 1; count++) {  int u = minDistance(dist, visited, vertices);  visited[u] = 1;  for (int v = 0; v < vertices; v++) {  if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX &&  dist[u] + graph[u][v] < dist[v]) {  dist[v] = dist[u] + graph[u][v];  }  }  }  printSolution(dist, vertices);  }  int main() {  int vertices, edges;  printf("Enter the number of vertices and edges: ");  scanf("%d %d", &vertices, &edges);  int graph[V][V] = {0};  printf("Enter the edges and their weights (format: source destination weight):\n");  for (int i = 0; i < edges; i++) {  int source, destination, weight;  scanf("%d %d %d", &source, &destination, &weight);  graph[source][destination] = weight;  }  int source;  printf("Enter the source vertex: ");  scanf("%d", &source);  dijkstra(graph, source, vertices);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 132129.png |
|  | Implementation of Minimum Spanning Tree using Prim’s Algorithm  #include <stdio.h>  #include <stdbool.h>  #include <limits.h>  #define V 10  int minKey(int key[], bool mstSet[], int n) {  int min = INT\_MAX;  int min\_index = -1;  for (int v = 0; v < n; v++) {  if (!mstSet[v] && key[v] < min) {  min = key[v];  min\_index = v;  }  }  return min\_index;  }  void printMST(int parent[], int graph[V][V], int n) {  printf("Edge \tWeight\n");  int totalWeight = 0;  for (int i = 1; i < n; i++) {  printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);  totalWeight += graph[i][parent[i]];  }  printf("Total Spanning Tree Weight: %d\n", totalWeight);  }  void primMST(int graph[V][V], int n) {  int parent[V];  int key[V];  bool mstSet[V];  for (int i = 0; i < n; i++) {  key[i] = INT\_MAX;  mstSet[i] = false;  }  key[0] = 0;  parent[0] = -1;  for (int count = 0; count < n - 1; count++) {  int u = minKey(key, mstSet, n);  mstSet[u] = true;  for (int v = 0; v < n; v++) {  if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {  parent[v] = u;  key[v] = graph[u][v];  }  }  }  printMST(parent, graph, n);  }  int main() {  int n;  printf("Enter the number of vertices: ");  scanf("%d", &n);  int graph[V][V];  printf("Enter the adjacency matrix:\n");  for (int i = 0; i < n; i++) {  for (int j = 0; j < n; j++) {  scanf("%d", &graph[i][j]);  }  }  primMST(graph, n);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 132714.png |
|  | Implementation of Minimum Spanning Tree using Kruskal Algorithm  #include <stdio.h>  #include <stdlib.h>  struct Edge {  int source, destination, weight;  };  struct Graph {  int V, E;  struct Edge\* edge;  };  struct Graph\* createGraph(int V, int E) {  struct Graph\* graph = (struct Graph\*)malloc(sizeof(struct Graph));  graph->V = V;  graph->E = E;  graph->edge = (struct Edge\*)malloc(E \* sizeof(struct Edge));  return graph;  }  void sortEdges(struct Graph\* graph) {  for (int i = 0; i < graph->E - 1; i++) {  for (int j = 0; j < graph->E - i - 1; j++) {  if (graph->edge[j].weight > graph->edge[j + 1].weight) {  struct Edge temp = graph->edge[j];  graph->edge[j] = graph->edge[j + 1];  graph->edge[j + 1] = temp;  }  }  }  }  int findParent(int parent[], int vertex) {  if (parent[vertex] == -1)  return vertex;  return findParent(parent, parent[vertex]);  }  void unionSets(int parent[], int x, int y) {  int xset = findParent(parent, x);  int yset = findParent(parent, y);  parent[xset] = yset;  }  int kruskalMST(struct Graph\* graph) {  int V = graph->V;  struct Edge result[V];  int parent[V];  for (int i = 0; i < V; i++)  parent[i] = -1;  sortEdges(graph);  int edgeCount = 0;  int i = 0;  int totalSpanningWeight = 0;  while (edgeCount < V - 1 && i < graph->E) {  struct Edge nextEdge = graph->edge[i++];  int x = findParent(parent, nextEdge.source);  int y = findParent(parent, nextEdge.destination);  if (x != y) {  result[edgeCount++] = nextEdge;  totalSpanningWeight += nextEdge.weight;  unionSets(parent, x, y);  }  }  printf("Edges in MST:\n");  for (int i = 0; i < edgeCount; i++) {  printf("%d -- %d, Weight: %d\n", result[i].source, result[i].destination, result[i].weight);  }  return totalSpanningWeight;  }  int main() {  int V, E;  printf("Enter the number of vertices: ");  scanf("%d", &V);  printf("Enter the number of edges: ");  scanf("%d", &E);  struct Graph\* graph = createGraph(V, E);  printf("Enter edge details (source destination weight):\n");  for (int i = 0; i < E; i++) {  scanf("%d %d %d", &graph->edge[i].source, &graph->edge[i].destination, &graph->edge[i].weight);  }  int totalSpanningWeight = kruskalMST(graph);  printf("Total Spanning Weight: %d\n", totalSpanningWeight);  return 0;  }  C:\Users\mailm\OneDrive\Pictures\Screenshots\Screenshot 2023-08-09 133305.png |