**DAA RECORD (Additional questions)**

1. **AIM:** To check connected graph status using DFS using adjacent list(iterative)

***PROGRAM:***

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string.h>*

*#define MAX 1000*

**typedef** **struct** Node {

**int** vertex;

**struct** Node\* next;

} Node;

Node\* adj[MAX];

Node\* rev[MAX];

**int** visited[MAX], V;

Node\* createNode(**int** v) {

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

newNode->vertex = v;

newNode->next = NULL;

**return** newNode;

}

**void** addEdge(Node\* graph[MAX], **int** u, **int** v) {

Node\* newNode = createNode(v);

newNode->next = graph[u];

graph[u] = newNode;

}

**void** dfsIterative(**int** start, Node\* graph[MAX]) {

**int** stack[MAX], top = -1;

stack[++top] = start;

visited[start] = 1;

**while** (top != -1) {

**int** v = stack[top--];

Node\* temp = graph[v];

**while** (temp) {

**int** adjVertex = temp->vertex;

**if** (!visited[adjVertex]) {

stack[++top] = adjVertex;

visited[adjVertex] = 1;

}

temp = temp->next;

}

}

}

**int** isStronglyConnected() {

memset(visited, 0, **sizeof**(visited));

dfsIterative(0, adj);

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

**if** (!visited[i]) **return** 0;

memset(visited, 0, **sizeof**(visited));

dfsIterative(0, rev);

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

**if** (!visited[i]) **return** 0;

**return** 1;

}

**int** main() {

**int** E, u, v;

scanf(*"%d %d"*, &V, &E);

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

adj[i] = NULL;

rev[i] = NULL;

}

**for** (**int** i = 0; i < E; i++) {

scanf(*"%d %d"*, &u, &v);

addEdge(adj, u, v);

addEdge(rev, v, u);

}

printf(isStronglyConnected() ? *"Strongly connected****\n****"* : *"Not strongly connected****\n****"*);

**return** 0;

}

***Output:***

******

1. **AIM:** To check connected graph status using DFS using adjacent list(recursion)

***PROGRAM:***

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string.h>*

*#define MAX 1000*

**typedef** **struct** Node {

**int** vertex;

**struct** Node\* next;

} Node;

Node\* adj[MAX];

**int** visited[MAX], V;

Node\* createNode(**int** v) {

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

newNode->vertex = v;

newNode->next = NULL;

**return** newNode;

}

**void** addEdge(**int** u, **int** v) {

Node\* newNode = createNode(v);

newNode->next = adj[u];

adj[u] = newNode;

}

**void** dfsRecursive(**int** v) {

visited[v] = 1;

printf(*"%d "*, v);

Node\* temp = adj[v];

**while** (temp) {

**int** adjVertex = temp->vertex;

**if** (!visited[adjVertex]) {

dfsRecursive(adjVertex);

}

temp = temp->next;

}

}

**int** main() {

**int** E, u, v;

scanf(*"%d %d"*, &V, &E);

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

adj[i] = NULL;

}

**for** (**int** i = 0; i < E; i++) {

scanf(*"%d %d"*, &u, &v);

addEdge(u, v);

}

memset(visited, 0, **sizeof**(visited));

printf(*"DFS starting from vertex 0:****\n****"*);

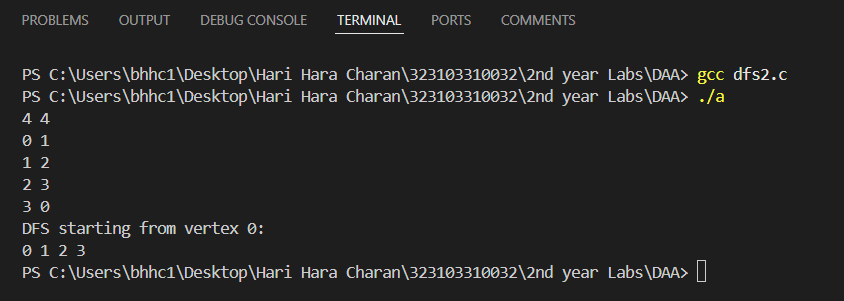
dfsRecursive(0);

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

**return** 0;

}

***Output:***



1. **AIM:** To check connected graph status using DFS using adjacency matrix iterative

***PROGRAM:***

*#include <stdio.h>*

*#include <stdlib.h>*

*#include <string.h>*

*#define MAX 1000*

**int** adj[MAX][MAX], rev[MAX][MAX], visited[MAX], V;

**void** dfs\_iterative(**int** start, **int** graph[MAX][MAX]) {

**int** stack[MAX], top = -1;

stack[++top] = start;

**while** (top != -1) {

**int** v = stack[top--];

**if** (!visited[v]) {

visited[v] = 1;

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

**if** (graph[v][i] && !visited[i]) {

stack[++top] = i;

}

}

}

}

}

**int** isStronglyConnected() {

memset(visited, 0, **sizeof**(visited));

dfs\_iterative(0, adj);

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

**if** (!visited[i]) **return** 0;

memset(visited, 0, **sizeof**(visited));

dfs\_iterative(0, rev);

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

**if** (!visited[i]) **return** 0;

**return** 1;

}

**int** main() {

**int** E, u, v;

scanf(*"%d %d"*, &V, &E);

memset(adj, 0, **sizeof**(adj));

memset(rev, 0, **sizeof**(rev));

**for** (**int** i = 0; i < E; i++) {

scanf(*"%d %d"*, &u, &v);

adj[u][v] = 1;

rev[v][u] = 1;

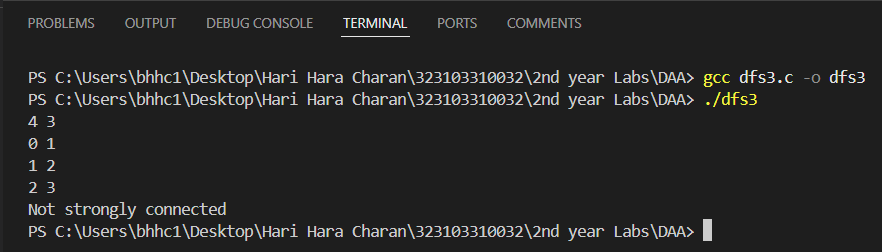
}

printf(isStronglyConnected() ? *"Strongly connected****\n****"* : *"Not strongly connected****\n****"*);

**return** 0;

}

***Output:***



1. **AIM:** To demonstrate binary search using iterative and recursion method

***PROGRAM:***

*#include <stdio.h>*

*#include <time.h>*

**int** binarySearchIterative(**int** arr[], **int** n, **int** k) {

**int** low = 0, high = n - 1;

**while** (low <= high) {

**int** mid = (low + high) / 2;

**if** (arr[mid] == k)

**return** mid;

**else** **if** (arr[mid] < k)

low = mid + 1;

**else**

high = mid - 1;

}

**return** -1;

}

**int** binarySearchRecursive(**int** arr[], **int** low, **int** high, **int** k) {

**if** (low <= high) {

**int** mid = (low + high) / 2;

**if** (arr[mid] == k)

**return** mid;

**else** **if** (arr[mid] < k)

**return** binarySearchRecursive(arr, mid + 1, high, k);

**else**

**return** binarySearchRecursive(arr, low, mid - 1, k);

}

**return** -1;

}

**int** main() {

**int** n, k;

printf(*"Enter the number of elements: "*);

scanf(*"%d"*, &n);

**int** arr[n];

printf(*"Enter sorted elements: "*);

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

scanf(*"%d"*, &arr[i]);

printf(*"Enter an element to search: "*);

scanf(*"%d"*, &k);

**clock\_t** start, end;

**double** cpu\_time\_used;

*// Iterative binary search*

start = clock();

**int** resultIterative = binarySearchIterative(arr, n, k);

end = clock();

cpu\_time\_used = ((**double**) (end - start)) / CLOCKS\_PER\_SEC;

**if** (resultIterative != -1)

printf(*"Iterative: Element found at position %d****\n****"*, resultIterative + 1);

**else**

printf(*"Iterative: Element not found****\n****"*);

printf(*"Iterative binary search time: %f seconds****\n****"*, cpu\_time\_used);

*// Recursive binary search*

start = clock();

**int** resultRecursive = binarySearchRecursive(arr, 0, n - 1, k);

end = clock();

cpu\_time\_used = ((**double**) (end - start)) / CLOCKS\_PER\_SEC;

**if** (resultRecursive != -1)

printf(*"Recursive: Element found at position %d****\n****"*, resultRecursive + 1);

**else**

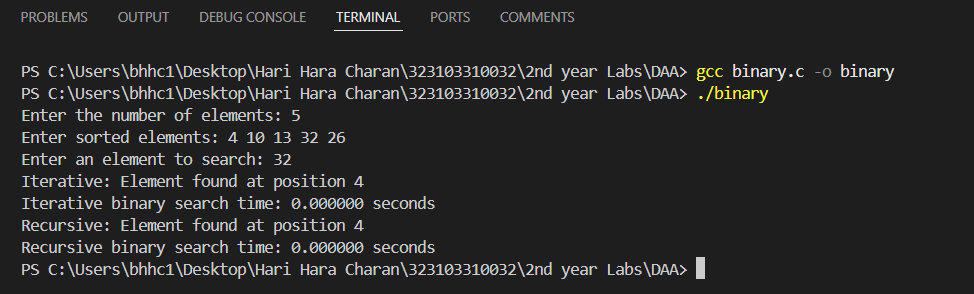
printf(*"Recursive: Element not found****\n****"*);

printf(*"Recursive binary search time: %f seconds****\n****"*, cpu\_time\_used);

**return** 0;

}

***Output:***



1. **AIM:** To perform minimum and maximum terms using recursion and iterative method

***PROGRAM:***

*#include <stdio.h>*

*// Iterative functions*

**void** findMinMaxIterative(**int** arr[], **int** n, **int** \*min, **int** \*max) {

\*min = arr[0];

\*max = arr[0];

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

**if** (arr[i] < \*min)

\*min = arr[i];

**if** (arr[i] > \*max)

\*max = arr[i];

}

}

*// Recursive functions*

**int** findMinRecursive(**int** arr[], **int** n) {

**if** (n == 1)

**return** arr[0];

**int** min = findMinRecursive(arr, n - 1);

**return** (arr[n - 1] < min) ? arr[n - 1] : min;

}

**int** findMaxRecursive(**int** arr[], **int** n) {

**if** (n == 1)

**return** arr[0];

**int** max = findMaxRecursive(arr, n - 1);

**return** (arr[n - 1] > max) ? arr[n - 1] : max;

}

**int** main() {

**int** n;

printf(*"Enter the number of elements: "*);

scanf(*"%d"*, &n);

**int** arr[n];

printf(*"Enter elements: "*);

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

scanf(*"%d"*, &arr[i]);

*// Iterative approach*

**int** minIter, maxIter;

findMinMaxIterative(arr, n, &minIter, &maxIter);

printf(*"Iterative approach:****\n****"*);

printf(*"Minimum element: %d****\n****"*, minIter);

printf(*"Maximum element: %d****\n****"*, maxIter);

*// Recursive approach*

**int** minRec = findMinRecursive(arr, n);

**int** maxRec = findMaxRecursive(arr, n);

printf(*"Recursive approach:****\n****"*);

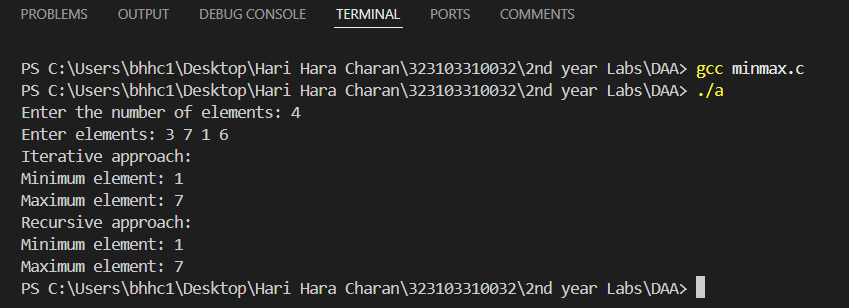
printf(*"Minimum element: %d****\n****"*, minRec);

printf(*"Maximum element: %d****\n****"*, maxRec);

**return** 0;

}

***Output:***

******