CSA0358 DATA STRUCTURES WITH GRAPH ALGORITHMS

DAY-5 : (12/08/2023)

QUESTION 1:

Write a C program to implement shortest path prims.

CODE:

```
#include <stdio.h>
#define INF 9999
#define MAX 10
void DijkstraAlgorithm(int Graph[MAX][MAX], int size, int start);
void DijkstraAlgorithm(int Graph[MAX][MAX], int size, int start) {
  int cost[MAX][MAX], distance[MAX], previous[MAX];
  int visited_nodes[MAX], counter, minimum_distance, next_node, i, j;
  for (i = 0; i < size; i++)
    for (j = 0; j < size; j++)
      if (Graph[i][j] == 0)
        cost[i][j] = INF;
      else
        cost[i][j] = Graph[i][j];
  for (i = 0; i < size; i++) {
    distance[i] = cost[start][i];
    previous[i] = start;
    visited_nodes[i] = 0;
  }
  distance[start] = 0;
  visited_nodes[start] = 1;
  counter = 1;
  while (counter < size - 1) {
    minimum_distance = INF;
    for (i = 0; i < size; i++)
      if (distance[i] < minimum_distance && !visited_nodes[i]) {
        minimum_distance = distance[i];
        next_node = i;
      }
    visited_nodes[next_node] = 1;
    for (i = 0; i < size; i++)
      if (!visited_nodes[i])
        if (minimum_distance + cost[next_node][i] < distance[i]) {
          distance[i] = minimum_distance + cost[next_node][i];
```

```
        previous[i] = next_node;
      }
    counter++;
  }
  for (i = 0; i < size; i++)
    if (i != start) {
      printf("\nDistance from the Source Node to %d: %d", i, distance[i]);
    }
}
int main() {
  int Graph[MAX][MAX], i, j, size, source;
  size = 7;
  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 0;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 8;
  Graph[0][6] = 0;
  Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 8;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 11;
  Graph[1][6] = 0;
  Graph[2][0] = 0;
  Graph[2][1] = 8;
  Graph[2][2] = 0;
  Graph[2][3] = 7;
  Graph[2][4] = 0;
  Graph[2][5] = 4;
  Graph[2][6] = 0;
  Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 7;
  Graph[3][3] = 0;
  Graph[3][4] = 9;
  Graph[3][5] = 14;
  Graph[3][6] = 0;
  Graph[4][0] = 0;
  Graph[4][1] = 0;
  Graph[4][2] = 0;
  Graph[4][3] = 9;
```

```
  Graph[4][4] = 0;
  Graph[4][5] = 10;
  Graph[4][6] = 2;
  Graph[5][0] = 0;
  Graph[5][1] = 0;
  Graph[5][2] = 4;
  Graph[5][3] = 14;
  Graph[5][4] = 10;
  Graph[5][5] = 0;
  Graph[5][6] = 2;
  Graph[6][0] = 0;
  Graph[6][1] = 0;
  Graph[6][2] = 0;
  Graph[6][3] = 0;
  Graph[6][4] = 2;
  Graph[6][5] = 0;
  Graph[6][6] = 1;
  source = 0;
  DijkstraAlgorithm(Graph, size, source);
  return 0;
}
```

OUTPUT:

```
Distance from the Source Node to 1: 4
Distance from the Source Node to 2: 12
Distance from the Source Node to 3: 19
Distance from the Source Node to 4: 12
Distance from the Source Node to 5: 8
Distance from the Source Node to 6: 10
--------------------------------
Process exited after 8.019 seconds with return value 0
Press any key to continue . . .
```

QUESTION 2:

Write a C program to implement Minimum spanning tree.

CODE:

```
#include <stdio.h>
#include <limits.h>
#define V 5
```

```
int minKey(int key[], int mstSet[]) {
   int min = INT_MAX, min_index;
   int v;
   for (v = 0; v < V; v++)
      if (mstSet[v] == 0 && key[v] < min)
         min = key[v], min_index = v;
   return min_index;
}
int printMST(int parent[], int n, int graph[V][V]) {
   int i;
   printf("Edge   Weight\n");
   for (i = 1; i < V; i++)
      printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}
void primMST(int graph[V][V]) {
   int parent[V];
   int key[V], i, v, count;
   int mstSet[V];
            for (i = 0; i < V; i++)
      key[i] = INT_MAX, mstSet[i] = 0;
   key[0] = 0;
   parent[0] = -1;
   for (count = 0; count < V - 1; count++) {
      int u = minKey(key, mstSet);
      mstSet[u] = 1;
      for (v = 0; v < V; v++)
         if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
            parent[v] = u, key[v] = graph[u][v];
   }
   printMST(parent, V, graph);
}
int main() {
   int graph[V][V] = { { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 },
         { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 }, };
   primMST(graph);
   return 0;
}
```

OUTPUT:

```
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5

--------------------------------
Process exited after 0.7467 seconds with return value 0
Press any key to continue . . .
```

QUESTION 3:

Write a C program to implement a Graph transversal depth first search.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#define MAX 5
struct Vertex {
  char label;
  bool visited;
};
int stack[MAX];
int top = -1;
struct Vertex* lstVertices[MAX];
int adjMatrix[MAX][MAX];
int vertexCount = 0;
void push(int item) {
  stack[++top] = item;
}
int pop() {
  return stack[top--];
}
int peek() {
  return stack[top];
}
bool isStackEmpty() {
  return top == -1;
}
void addVertex(char label) {
  struct Vertex* vertex = (struct Vertex*) malloc(sizeof(struct Vertex));
  vertex->label = label;
```

```
   vertex->visited = false;
   lstVertices[vertexCount++] = vertex;
}
void addEdge(int start,int end) {
   adjMatrix[start][end] = 1;
   adjMatrix[end][start] = 1;
}
void displayVertex(int vertexIndex) {
   printf("%c ",lstVertices[vertexIndex]->label);
}
int getAdjUnvisitedVertex(int vertexIndex) {
   int i;
   for(i = 0; i < vertexCount; i++) {
      if(adjMatrix[vertexIndex][i] == 1 && lstVertices[i]->visited == false) {
         return i;
      }
   }
   return -1;
}
void depthFirstSearch() {
   int i;
   lstVertices[0]->visited = true;
   displayVertex(0);
   push(0);
   while(!isStackEmpty()) {
      int unvisitedVertex = getAdjUnvisitedVertex(peek());
      if(unvisitedVertex == -1) {
         pop();
      } else {
         lstVertices[unvisitedVertex]->visited = true;
         displayVertex(unvisitedVertex);
         push(unvisitedVertex);
      }
   }
   for(i = 0;i < vertexCount;i++) {
      lstVertices[i]->visited = false;
   }
}
int main() {
   int i, j;
   for(i = 0; i < MAX; i++) {
      for(j = 0; j < MAX; j++)
         adjMatrix[i][j] = 0;
   }
```

```
  addVertex('S');
  addVertex('A');
  addVertex('B');
  addVertex('C');
  addVertex('D');
  addEdge(0, 1);
  addEdge(0, 2);
  addEdge(0, 3);
  addEdge(1, 4);
  addEdge(2, 4);
  addEdge(3, 4);
  printf("Depth First Search: ");
  depthFirstSearch();
  return 0;
}
```

OUTPUT:

```
Depth First Search: S A D B C
--------------------------------
Process exited after 1.142 seconds with return value 0
Press any key to continue . . .
```

QUESTION 4:

Write a C program to implement a Graph transversal Breadth first search.

CODE:

```
#include<stdio.h>
int queue[100];
int front=0,back=0;
void push(int var)
{
   queue[back] = var;
   back++;
}
void pop()
{
   queue[front] = 0;
   front++;
}
int visited[7] = {0};
```

```
int main()
{
   int N  = 6;
   int graph[6][6] = {{0,1,1,0,0,0},
                      {1,0,1,0,0,0},
                      {1,1,0,1,1,0},
                      {0,0,1,0,0,0},
                      {0,0,1,0,0,1},
                      {0,0,0,0,1,0}};
   push(1);
   visited[0] = 1;
   while(front != back)
   {
      int current = queue[front];
      printf("%d ", current);
      pop();
      for(int i=0;i<6;i++)
      {
         if((graph[current-1][i] == 1) && (visited[i] == 0))
         {
            visited[i] = 1;
            push(i+1);
         }
      }
   }
   return 0;
}
```

OUTPUT:

```
1 2 3 4 5 6
--------------------------------
Process exited after 3.271 seconds with return value 0
Press any key to continue . . .
```