Data Structures Odyssey: Exploring the Foundations of Computing

Ex. No.:13	Graph Traversal	Date:16/05/2014

Write a C program to create a graph and find a minimum spanning tree using prim's algorithm.

Algorithm:

1) Start
2) Initialize an empty set to store the minimum spanning tree (MST) and a priority queue to store edges and their weights.
3) Choose a starting node and add it to the MST set.
4) For each edge connected to the starting node, add the edge to the priority queue.
5) While the priority queue is not empty:
a. Extract the edge with the smallest weight from the priority queue.
b. If adding the edge to the MST set does not create a cycle, add the edge to the MST set.
c. For each neighbour of the newly added node in the MST set:
i. If the neighbour is not already in the MST set, add the edge connecting the neighbor to the priority queue.
6) Repeat step 4 until all nodes are included in the MST set.
7) The edges in the MST set form the minimum spanning tree of the graph. 8) Stop

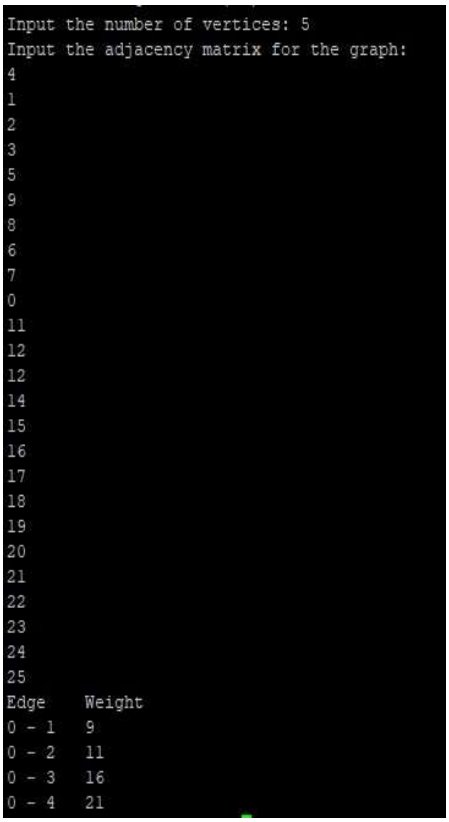
PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct node { int
vertex; struct
node* next;

}; struct adj_list {
struct node* head;
};
struct graph { int
num_vertices; struct
adj_list* adj_lists; int*
visited;
};
struct node* new_node(int vertex) { struct node* new_node =
(struct node*)malloc(sizeof(struct node)); new_node->vertex =
vertex; new_node->next = NULL; return new_node;
}
struct graph* create_graph(int n) { struct graph* graph = (struct
graph*)malloc(sizeof(struct graph)); graph->num_vertices = n;
graph->adj_lists = (struct adj_list*)malloc(n * sizeof(struct adj_list));
graph->visited = (int*)malloc(n * sizeof(int));

int i; for (i = 0; i< n;
i++) { graph-
>adj_lists[i].head =
NULL; graph-
>visited[i] = 0;
}

return graph;
```

```
}

void add_edge(struct graph* graph, int src, int dest) {
struct node* new_node1 = new_node(dest);
new_node1->next = graph->adj_lists[src].head;
graph->adj_lists[src].head = new_node1; struct
node* new_node2 = new_node(src); new_node2-
>next = graph->adj_lists[dest].head; graph-
>adj_lists[dest].head = new_node2;
}
void bfs(struct graph* graph, int v) { int queue[1000]; int
front = -1; int rear = -1; graph->visited[v] = 1;
queue[++rear] = v; while (front != rear) { int
current_vertex = queue[++front]; printf("%d ",
current_vertex); struct node* temp = graph-
>adj_lists[current_vertex].head; while (temp != NULL) { int
adj_vertex = temp->vertex; if (graph->visited[adj_vertex]
== 0) { graph->visited[adj_vertex] = 1; queue[++rear] =
adj_vertex;
}

temp = temp->next;
}
}
}

int main() { struct graph* graph =
create_graph(6); add_edge(graph, 0, 1);
add_edge(graph, 0, 2); add_edge(graph, 1, 3);
add_edge(graph, 1, 4); add_edge(graph, 2, 4);
add_edge(graph, 3, 4); add_edge(graph, 3, 5);
add_edge(graph, 4,5); printf("BFS traversal
starting from vertex 0: "); bfs(graph, 0);
```

```
return 0;
}
```

OUTPUT:



RESULT: Thus, the program was successfully executed.