Param Chhabra

22BCE0744

LAB ASSESSMENT 6

PRIM'S ALGORITHM

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

struct Adjacent {
   struct Node *to;
   int weight;
};

struct Node {
   int data;
   struct Adjacent *adjacents[4];
   bool visited;
};

struct Node *create_node(int data) {
```

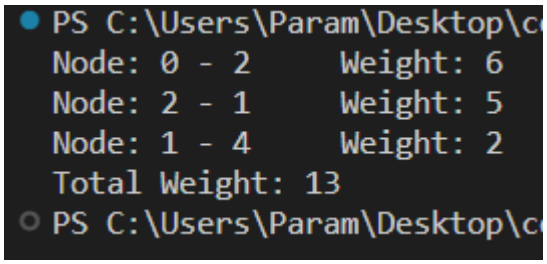
```
    struct Node *node = (struct Node *)malloc(sizeof(struct
Node));
    node->data = data;
    node->visited = false;
    for (int i = 0; i < 4; i++) {
        node->adjacents[i] = NULL;
    }
    return node;
}

void prim(struct Node *startNode) {
    startNode->visited = true;
    int totalWeight = 0;

    // Traverse until all nodes are visited
    while (true) {
        struct Node *minNode = NULL;
        struct Adjacent *minAdjacent = NULL;
        int minWeight = INT_MAX;

        // Find the minimum weight adjacent edge
        for (int i = 0; i < 4; i++) {
```

```
            struct Adjacent *adjacent = startNode->adjacents[i];
            if (adjacent != NULL && !adjacent->to->visited &&
adjacent->weight < minWeight) {
                minNode = startNode;
                minAdjacent = adjacent;
                minWeight = adjacent->weight;
            }
        }

        if (minNode == NULL) {
            // No more adjacent nodes to visit
            break;
        }

        struct Node *nextNode = minAdjacent->to;
        nextNode->visited = true;
        totalWeight += minWeight;
        printf("Node: %d - %d\tWeight: %d\n", minNode->data,
nextNode->data, minWeight);
        startNode = nextNode;
    }
```

```
    printf("Total Weight: %d\n", totalWeight);
}

int main() {
    // Create 5 nodes
    struct Node *nodes[5];
    for (int i = 0; i < 5; i++) {
        nodes[i] = create_node(i);
    }

    // Connect nodes
    nodes[0]->adjacents[0] = &(struct Adjacent){nodes[1], 10};
    nodes[0]->adjacents[1] = &(struct Adjacent){nodes[2], 6};
    nodes[1]->adjacents[0] = &(struct Adjacent){nodes[0], 10};
    nodes[1]->adjacents[1] = &(struct Adjacent){nodes[2], 5};
    nodes[1]->adjacents[2] = &(struct Adjacent){nodes[3], 15};
    nodes[1]->adjacents[3] = &(struct Adjacent){nodes[4], 2};
    nodes[2]->adjacents[0] = &(struct Adjacent){nodes[0], 6};
    nodes[2]->adjacents[1] = &(struct Adjacent){nodes[1], 5};
    nodes[3]->adjacents[0] = &(struct Adjacent){nodes[1], 15};
    nodes[4]->adjacents[0] = &(struct Adjacent){nodes[1], 2};
```

```
    // Start Prim's algorithm from the first node
    prim(nodes[0]);

    return 0;
}
```

OUTPUT-



KRUSHKAL'S ALGORITHM

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* parent;
    int rank;
};

struct Edge {
```

```
    struct Node* src;
    struct Node* dest;
    int weight;
};

struct Node* create_node(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct Node));
    node->data = data;
    node->parent = node;
    node->rank = 0;
    return node;
}

struct Edge* create_edge(struct Node* src, struct Node* dest, int
weight) {
    struct Edge* edge = (struct Edge*)malloc(sizeof(struct Edge));
    edge->src = src;
    edge->dest = dest;
    edge->weight = weight;
    return edge;
}

struct Node* find(struct Node* node) {
    if (node->parent != node) {
```

```
        node->parent = find(node->parent);
    }
    return node->parent;
}

void union_nodes(struct Node* x, struct Node* y) {
    struct Node* x_root = find(x);
    struct Node* y_root = find(y);
    if (x_root != y_root) {
        if (x_root->rank < y_root->rank) {
            x_root->parent = y_root;
        } else if (x_root->rank > y_root->rank) {
            y_root->parent = x_root;
        } else {
            y_root->parent = x_root;
            x_root->rank++;
        }
    }
}

void kruskal(struct Node* nodes[], struct Edge* edges[], int
num_nodes, int num_edges) {
    struct Edge* mst_edges[num_nodes - 1];
    int num_mst_edges = 0;
```

```
    int mst_weight = 0;

    for (int i = 0; i < num_edges; i++) {
        struct Edge* edge = edges[i];
        struct Node* src_root = find(edge->src);
        struct Node* dest_root = find(edge->dest);

        if (src_root != dest_root) {
            mst_edges[num_mst_edges] = edge;
            num_mst_edges++;
            mst_weight += edge->weight;
            union_nodes(src_root, dest_root);
        }
    }

    printf("Minimum Spanning Tree (MST):\n");
    printf("Edge \tWeight\n");

    for (int i = 0; i < num_mst_edges; i++) {
        struct Edge* edge = mst_edges[i];
        printf("%d - %d \t%d\n", edge->src->data, edge->dest->data,
edge->weight);
    }
```

```
    printf("Total Weight of MST: %d\n", mst_weight);
}

int main() {
    // Create 5 nodes
    struct Node* nodes[5];
    for (int i = 0; i < 5; i++) {
        nodes[i] = create_node(i);
    }

    // Create edges
    struct Edge* edges[7];
    edges[0] = create_edge(nodes[0], nodes[1], 10);
    edges[1] = create_edge(nodes[0], nodes[2], 6);
    edges[2] = create_edge(nodes[1], nodes[2], 5);
    edges[3] = create_edge(nodes[1], nodes[3], 15);
    edges[4] = create_edge(nodes[1], nodes[4], 2);
    edges[5] = create_edge(nodes[2], nodes[3], 4);
    edges[6] = create_edge(nodes[3], nodes[4], 3);

    // Apply Kruskal's algorithm
    kruskal(nodes, edges, 5, 7);

    // Cleanup - Free allocated memory
```

```
    for (int i = 0; i < 5; i++) {
        free(nodes[i]);
    }

    for (int i = 0; i < 7; i++) {
        free(edges[i]);
    }

    return 0;
}
```

OUTPUT-