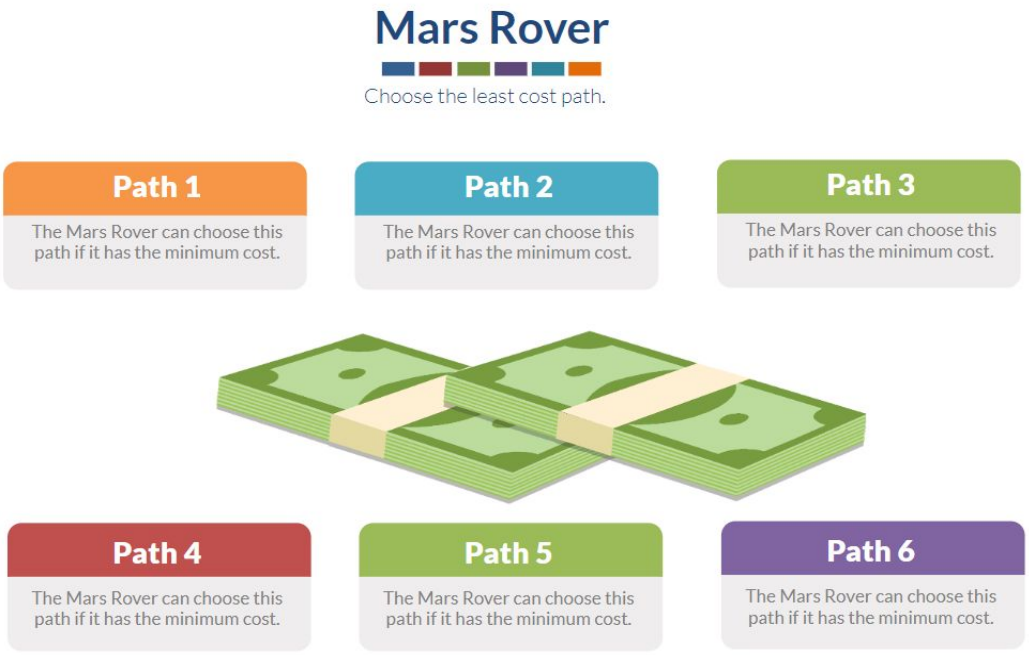
Kruskal's Algorithm

Kruskal's Algorithm can be used to find the cost of a particular path of the Mars Rover. If multiple paths of the same length exist, that is the shortest path out of all possible paths the Rover can take, then the Rover can choose the path that has the minimum cost using this algorithm.

The time complexity of this path is O(VlogV+ElogV).



Program:

//Header Section

```
#include <bits/stdc++.h>
using namespace std;

class Edge
{
   public:
   int src, dest, weight;
};

class Graph
{
```

```
    public:
    // V-> Number of vertices, E-> Number of edges
    int V, E;

    Edge* edge;
};

Graph* createGraph(int V, int E)
{
    Graph* graph = new Graph;
    graph->V = V;
    graph->E = E;

    graph->edge = new Edge[E];

    return graph;
}

class subset
{
    public:
    int parent;
    int rank;
};

int find(subset subsets[], int i)
{
    if (subsets[i].parent != i)
        subsets[i].parent = find(subsets, subsets[i].parent);

    return subsets[i].parent;
}
void Union(subset subsets[], int x, int y)
{
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);
    if (subsets[xroot].rank < subsets[yroot].rank)
        subsets[xroot].parent = yroot;
    else if (subsets[xroot].rank > subsets[yroot].rank)
        subsets[yroot].parent = xroot;

    else
    {
```

```
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
  int myComp(const void* a, const void* b)
{
    Edge* a1 = (Edge*)a;
    Edge* b1 = (Edge*)b;
    return a1->weight > b1->weight;
}

//Functions

void KruskalMST(Graph* graph)
{
    int V = graph->V;
    Edge result[V]; // Tnis will store the resultant MST
    int e = 0; // An index variable, used for result[]
    int i = 0; // An index variable, used for sorted edges

    qsort(graph->edge, graph->E, sizeof(graph->edge[0]), myComp);
    subset *subsets = new subset[( V * sizeof(subset) )];
    for (int v = 0; v < V; ++v)
    {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    while (e < V - 1 && i < graph->E)
    {
        Edge next_edge = graph->edge[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y)
        {
            result[e++] = next_edge;
            Union(subsets, x, y);
        }
        // Else discard the next_edge
    }
    cout<<"Following are the edges in the constructed MST\n";
    for (i = 0; i < e; ++i)
```

```
        cout<<result[i].src<<" -- "<<result[i].dest<<" == "<<result[i].weight<<endl;
    return;
}

//Main Function
int main()
{
    /* Let us create following weighted graph
            10
        0--------1
        | \ |
    6| 5\ |15
        | \ |
        2--------3
            4 */
    int V = 4; // Number of vertices in graph
    int E = 5; // Number of edges in graph
    Graph* graph = createGraph(V, E);


    // add edge 0-1
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[0].weight = 10;

    // add edge 0-2
    graph->edge[1].src = 0;
    graph->edge[1].dest = 2;
    graph->edge[1].weight = 6;

    // add edge 0-3
    graph->edge[2].src = 0;
    graph->edge[2].dest = 3;
    graph->edge[2].weight = 5;

    // add edge 1-3
    graph->edge[3].src = 1;
    graph->edge[3].dest = 3;
    graph->edge[3].weight = 15;

    // add edge 2-3
    graph->edge[4].src = 2;
    graph->edge[4].dest = 3;
```

```
    graph->edge[4].weight = 4;

    KruskalMST(graph);

    return 0;
}
```

Work Cited:
https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/