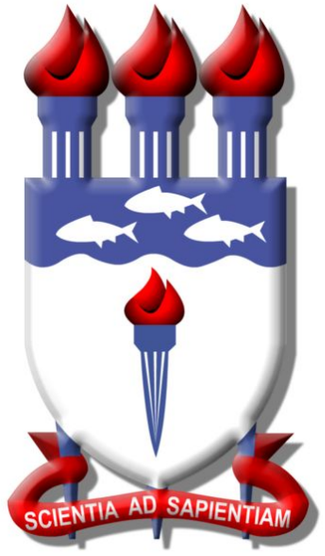


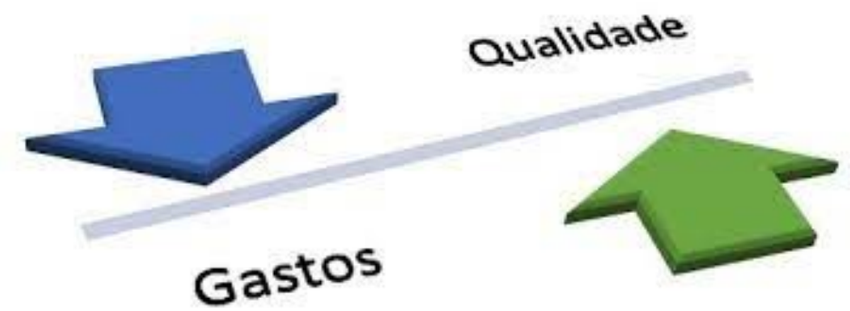
Algoritmo Kruskal

Equipe

[Link do github do projeto]

Motivação

- Marcinho, um empreendedor visionário, concretizou o sonho de construir um complexo de 9 mansões de luxo em uma região isolada. Estas mansões, verdadeiras jóias arquitetônicas, estão distribuídas por uma vasta extensão de terra. Agora, enfrentando o desafio logístico de fornecer energia elétrica de forma eficiente para todas as mansões, Marcinho procura uma solução que minimize os custos sem comprometer a qualidade do serviço.



Algoritmo Kruskal

- É um método para encontrar uma Minimum Spanning Tree (MST) em um grafo conexo ponderado.

- É utilizado em projetos de rede de computadores, sistemas de distribuição elétrica, e planejamento de transporte, dentre outros.

Definições

- Grafo não-dirigido:

  Grafo onde a ordem dos vértices das arestas não importa.

- Subfloresta:

  Conjunto de árvores derivadas de uma floresta maior.

- Floresta geradora

  Conjunto de árvores que conecta todos os vértices de um grafo sem formar ciclos.

Estruturas

```c
typedef struct Edge{
    int src, dest, weight;
}Edge;
```

```c
typedef struct Subset {
    int parent, rank;
}Subset;
```

Código Union-Find

```
int find(Subset subsets[], int i){
    if (subsets[i].parent != i){
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}
```

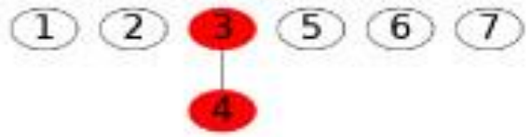
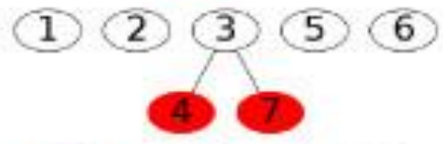
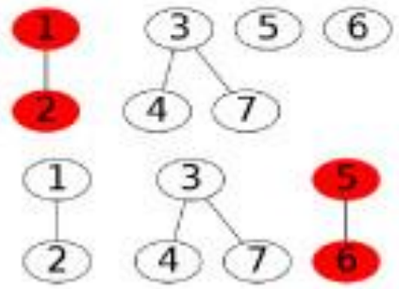
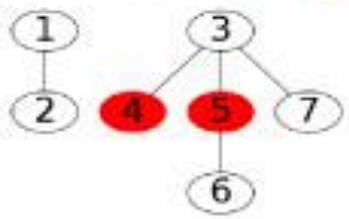
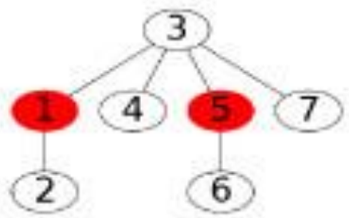
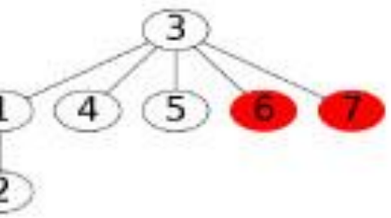
```
void Union(Subset subsets[], int x, int y){
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    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++;
    }
}
```

Codigo Extra

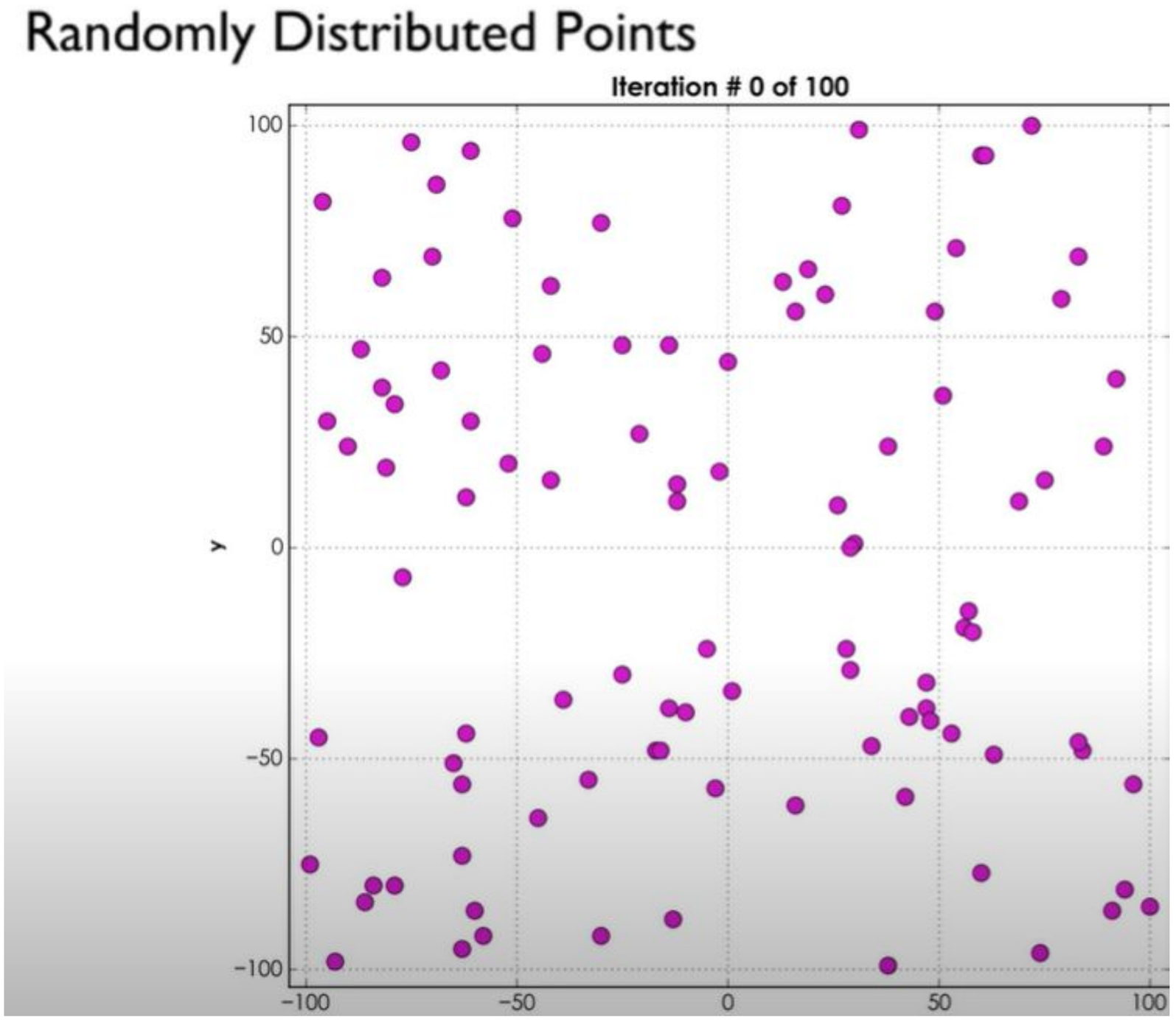
```c
void shuffle(struct Edge edges[], int n) {
    srand(time(NULL));  // Inicializa o ger
    for (int i = n - 1; i > 0; i--) {
        int j = rand() % (i + 1);
        struct Edge temp = edges[i];
        edges[i] = edges[j];
        edges[j] = temp;
    }
}
```

```c
void BubbleSortEdge(Edge edges[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (edges[j].weight > edges[j + 1].weight) {
                struct Edge temp = edges[j];
                edges[j] = edges[j + 1];
                edges[j + 1] = temp;
            }
        }
    }
}
```

Princípio de execução

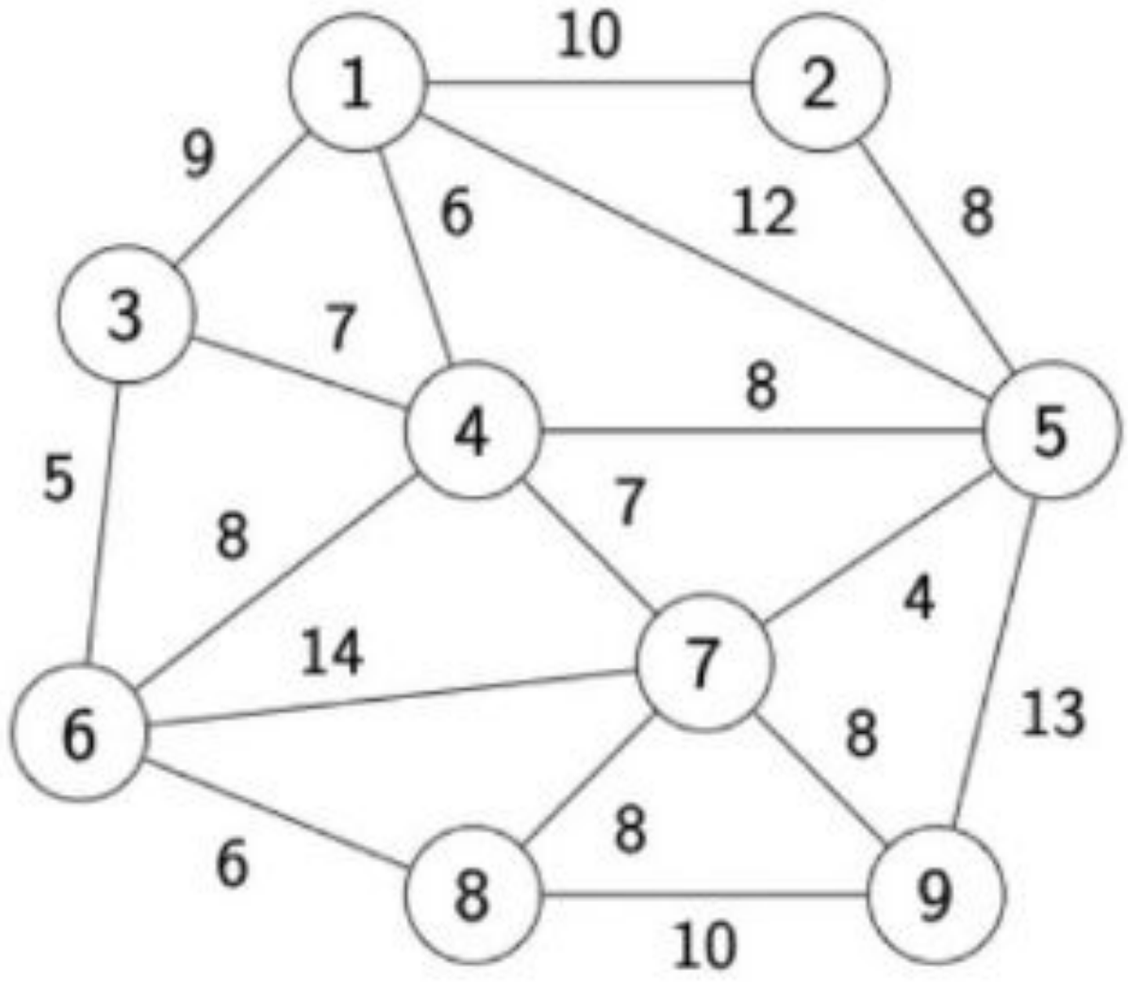
Edge	*parent*	Tree representation
	1 2 3 5 6 7	
(4,3)	1 2 3 3 5 6 7	
(7,4)	1 2 3 3 5 6 3	
(2,1)	1 1 3 3 5 6 3	
(6,5)	1 1 3 3 5 5 3	
(5,4)	1 1 3 3 3 5 3	
(5,1)	3 1 3 3 3 5 3	
(7,6)	3 1 3 3 3 3 3	

Animação


Randomly Distributed Points
Iteration # 0 of 100
100
50
0
−50
−100
y
−100
−50
0
50
100

De volta ao problema do Marcinho…

No grafo, cada vértice representa uma das 9 mansões de luxo construídas por Marcinho, as arestas ligam essas mansões e seus pesos correspondem às distâncias para fornecer energia elétrica entre elas.



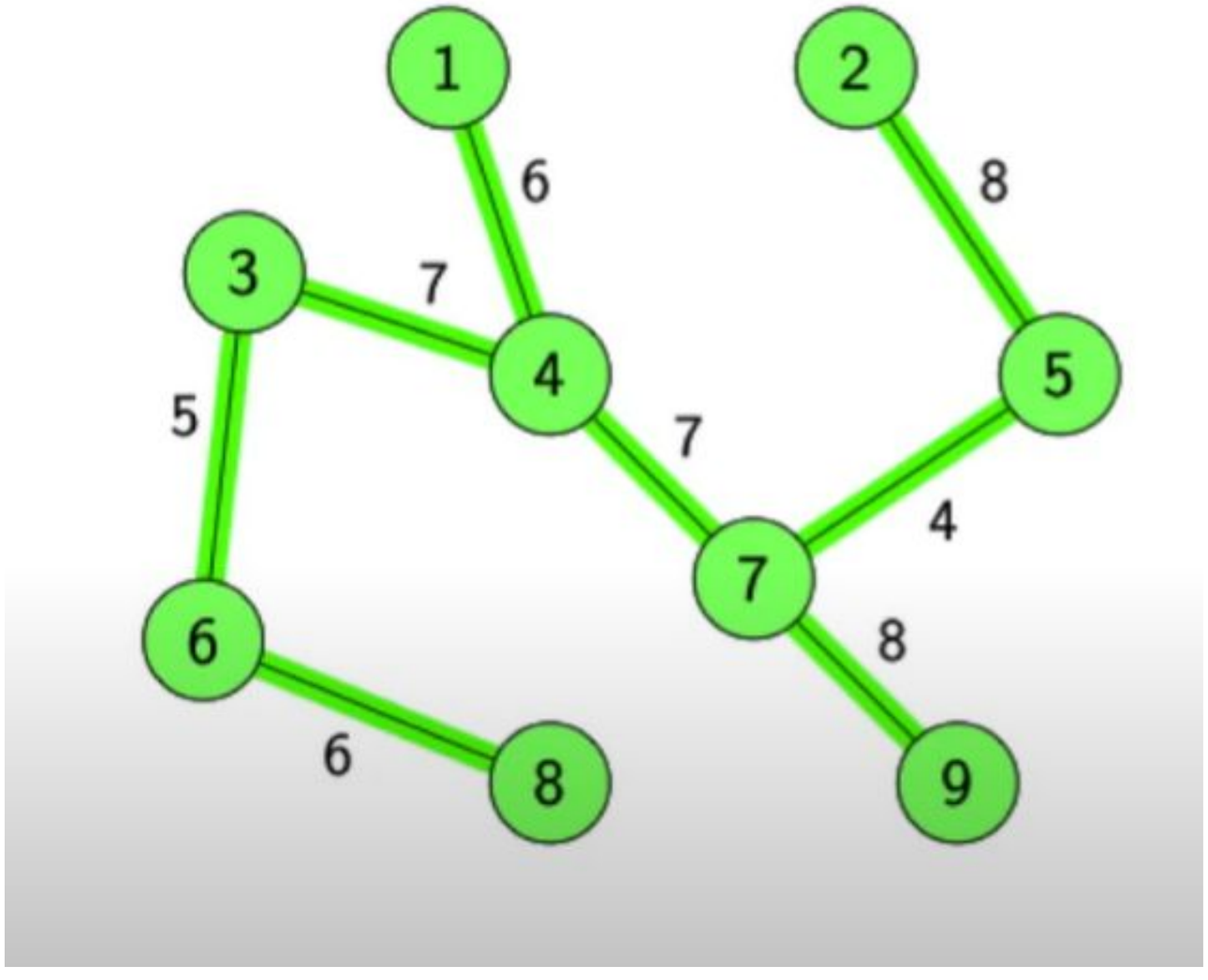
Construção do grafo

```
edges[0].src = 1;  edges[0].dest = 3;  edges[0].weight = 9;
edges[1].src = 1;  edges[1].dest = 4;  edges[1].weight = 6;
edges[2].src = 1;  edges[2].dest = 2;  edges[2].weight = 10;
edges[3].src = 1;  edges[3].dest = 5;  edges[3].weight = 12;
edges[4].src = 2;  edges[4].dest = 5;  edges[4].weight = 8;
edges[5].src = 3;  edges[5].dest = 4;  edges[5].weight = 7;
edges[6].src = 3;  edges[6].dest = 6;  edges[6].weight = 5;
edges[7].src = 4;  edges[7].dest = 6;  edges[7].weight = 8;
edges[8].src = 4;  edges[8].dest = 7;  edges[8].weight = 7;
edges[9].src = 4;  edges[9].dest = 5;  edges[9].weight = 8;
edges[10].src = 5; edges[10].dest = 7; edges[10].weight = 4;
edges[11].src = 5; edges[11].dest = 9; edges[11].weight = 13;
edges[12].src = 6; edges[12].dest = 7; edges[12].weight = 14;
edges[13].src = 6; edges[13].dest = 8; edges[13].weight = 6;
edges[14].src = 7; edges[14].dest = 8; edges[14].weight = 8;
edges[15].src = 7; edges[15].dest = 9; edges[15].weight = 8;
edges[16].src = 8; edges[16].dest = 9; edges[16].weight = 10;
```

Construindo MST

```
Subset* subsets = (Subset*) malloc((V+1) * sizeof(Subset));
for (int v = 0; v <= V; ++v) {
    subsets[v].parent = v;
    subsets[v].rank = 0;
}
Edge result[V-1];
int e = 0, i = 0;
while (e < V-1 && i < E) {
    Edge nextEdge = edges[i++];
    int x = find(subsets, nextEdge.src);
    int y = find(subsets, nextEdge.dest);

    if (x != y) {
        result[e++] = nextEdge;
        Union(subsets, x, y);
    }
}
```

Resultado




```
Arestas da AGM:
5 -- 7 == 4
3 -- 6 == 5
1 -- 4 == 6
6 -- 8 == 6
4 -- 7 == 7
3 -- 4 == 7
2 -- 5 == 8
7 -- 9 == 8
```

That's all Folks!