MaratonaCIn

Seletiva 2020

MaratonaCIn

Dijkstra, Floyd-Warshall, DSU, MST

Aula 4

Dijkstra

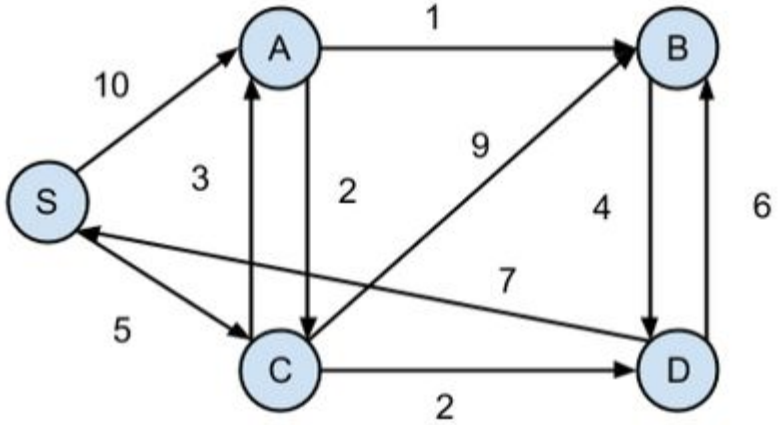
Dijkstra

- Calcula a distância de S aos demais vértices
- Arestas com pesos
  - Arestas de peso não negativo
- Busca gulosa
- Complexidade: O ((E+V) log V)
- Estruturas usadas:
  - Array com menores distâncias até agora
  - Min Heap com os próximos valores a analisar

Dijkstra

MaratonaCIn

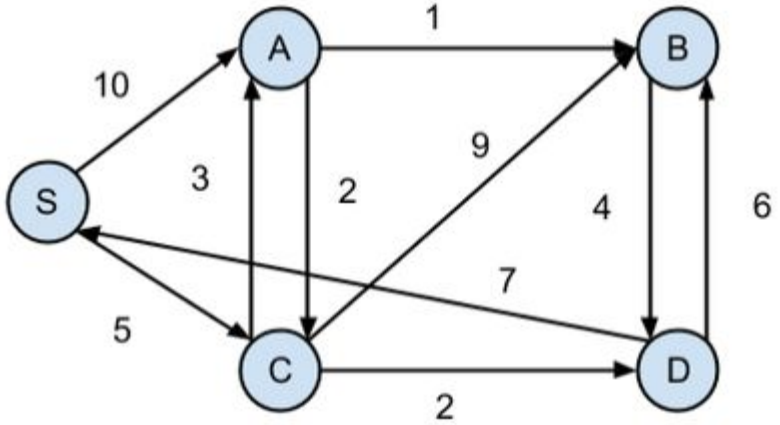
Node	d[v]	$\Pi[v]$
S	0	NIL
A	∞	NIL
B	∞	NIL
C	∞	NIL
D	∞	NIL



Dijkstra

MaratonaCIn

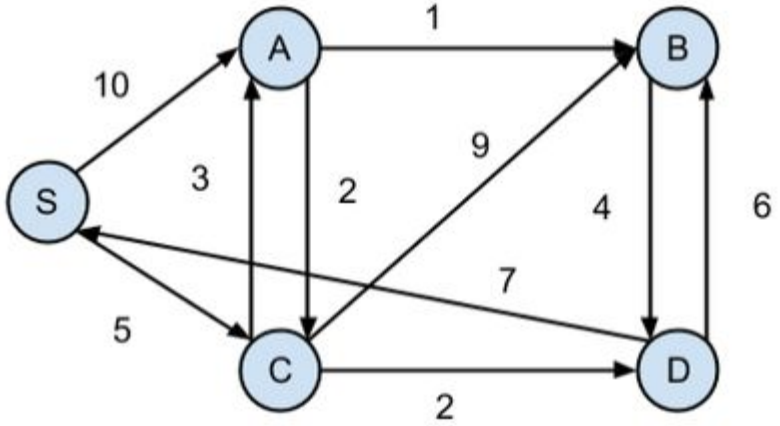
Node	d[v]	$\Pi[v]$
S	0	NIL
A	10	S
B	∞	NIL
C	5	S
D	∞	NIL



Dijkstra

MaratonaCIn

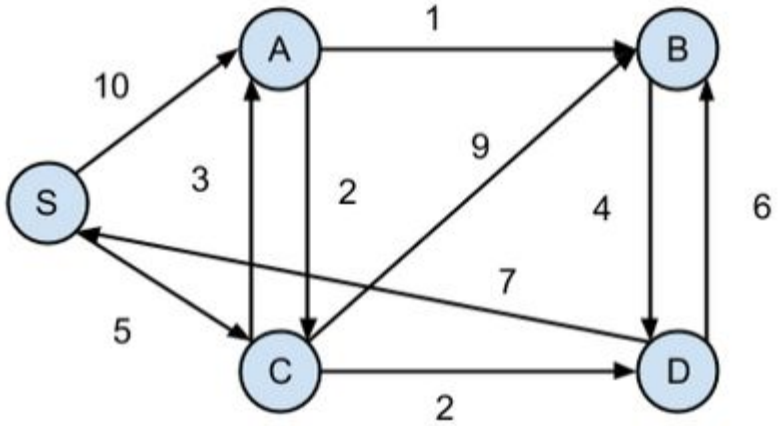
Node	d[v]	$\Pi[v]$
S	0	NIL
A	8	C
B	14	C
C	5	S
D	7	C



Dijkstra

MaratonaCIn

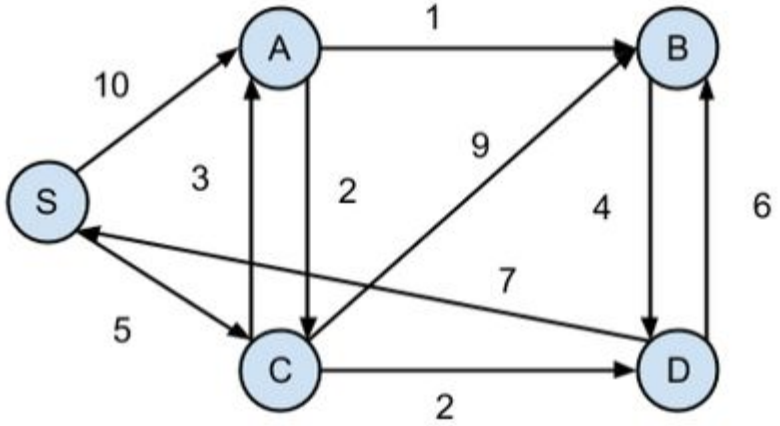
Node	d[v]	$\Pi[v]$
S	0	NIL
A	**8**	**C**
B	**13**	**D**
C	**5**	**S**
D	**7**	**C**



Dijkstra

MaratonaCIn

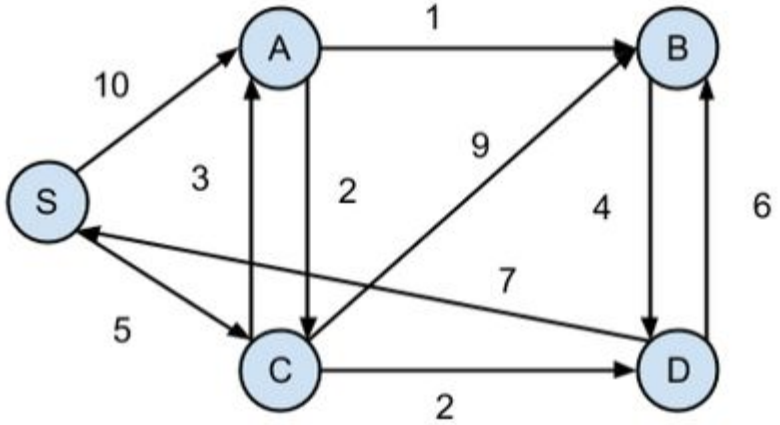
Node	d[v]	$\Pi[v]$
S	0	NIL
A	8	C
B	9	A
C	5	S
D	7	C



Dijkstra

MaratonaCIn

Node	d[v]	$\Pi[v]$
S	0	NIL
A	**8**	**C**
B	**9**	**A**
C	**5**	**S**
D	**7**	**C**



Dijkstra

MaratonaCIn

Código:

```
typedef pair<int, int> ii; const int inf = 0x3f3f3f3f;
#define inf 0x3f3f3f3f
const int N = 1e5+5;
int dist[N]; vector<ii> adjList[N];

void dijkstra(int source){
        for(int i = 0; i < n; i++) dist[i] = inf;
        dist[source] = 0;
        priority_queue<ii, vector<ii>, greater<ii>> pq;
        pq.emplace(0, source);

        while (!pq.empty()){
                int d = pq.top().first, u = pq.top().second;
                pq.pop();
                if (d > dist[u]) continue;

                for(auto e : adjList[u]){
                        int v = e.first, w = e.second;
                        if (dist[u] + w < dist[v]) {
                                dist[v] = dist[u] + w;
                                pq.emplace(dist[v], v);
                        }
                }
        }
```

Floyd Warshall

Floyd Warshall

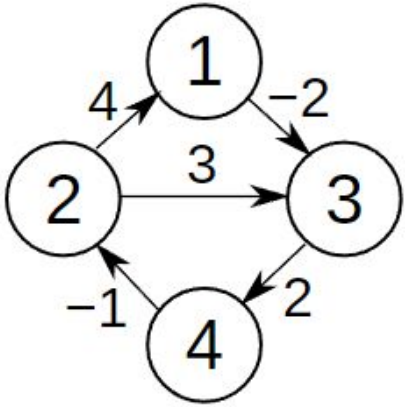
MaratonaCIn

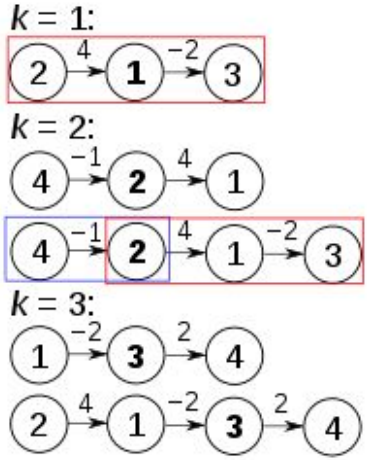
- Calcula o caminho mais curto entre todos os pares de vértices
- Programação dinâmica
- Recalcula as distâncias a partir de vértices intermediários
- Complexidade: O(V^3)

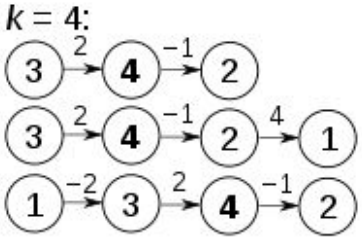
Floyd Warshall

MaratonaCIn

Exemplo:







$k=0$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	3	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=1$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	**2**	∞
3	∞	∞	0	2
4	∞	−1	∞	0

$k=2$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	∞
2	4	0	2	∞
3	∞	∞	0	2
4	**3**	−1	**1**	0

$k=3$	j			
i	**1**	**2**	**3**	**4**
1	0	∞	−2	**0**
2	4	0	2	**4**
3	∞	∞	0	2
4	3	−1	1	0

$k=4$	j			
i	**1**	**2**	**3**	**4**
1	0	**−1**	−2	0
2	4	0	2	4
3	**5**	**1**	0	2
4	3	−1	1	0

Floyd Warshall

MaratonaCIn

Código:

```
const int inf = 0x3f3f3f3f;
int dist[N][N];

void floyd(){
	for(int i = 0; i < n; i++) for(int j = 0; j < n; j++) {
		if(i == j) dist[i][j] = 0;
		else dist[i][j] = inf;
	}
	for(int i = 0; i < m; i++) {
		int u, v, w; cin >> u >> v >> w;
		dist[u][v] = min(dist[u][v], w);
	}
	for(int k = 0; k < n; k++) {
		for(int i = 0; i < n; i++) {
			for(int j = 0; j < n; j++) {
				dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
			}
		}
	}
}
```

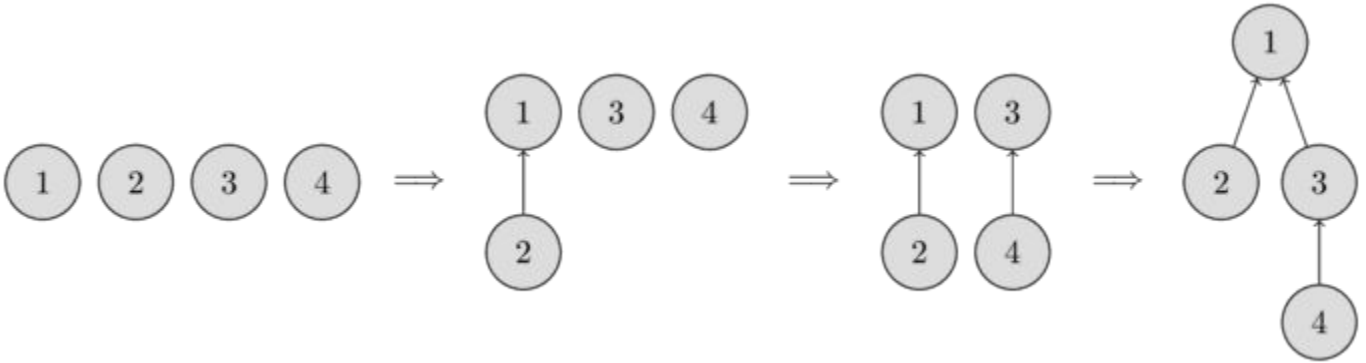
Disjoint Set Union(Union Find)

Disjoint-set Union

MaratonaCIn

Estrutura de dados capaz de rapidamente informar em qual conjunto determinado elemento está, se dois elementos estão em um mesmo conjunto e para unir dois conjuntos.

Visualização:https://s3.amazonaws.com/learneroo/visual-algorithms/DisjointSets.html



Código


```
void dsBuild() {
  for(int i = 0; i < n; i++) {
    ds[i] = i;
    sz[i] = 1;
  }
}

int dsFind(int i) {
  if(ds[i] != i) ds[i] = dsFind(ds[i]);
  return ds[i];
}

void dsUnion(int a, int b) {
  a = dsFind(a); b = dsFind(b);
  if(sz[a] < sz[b]) swap(a, b);
  if(a != b) sz[a] += sz[b];
  ds[b] = a;
}
```

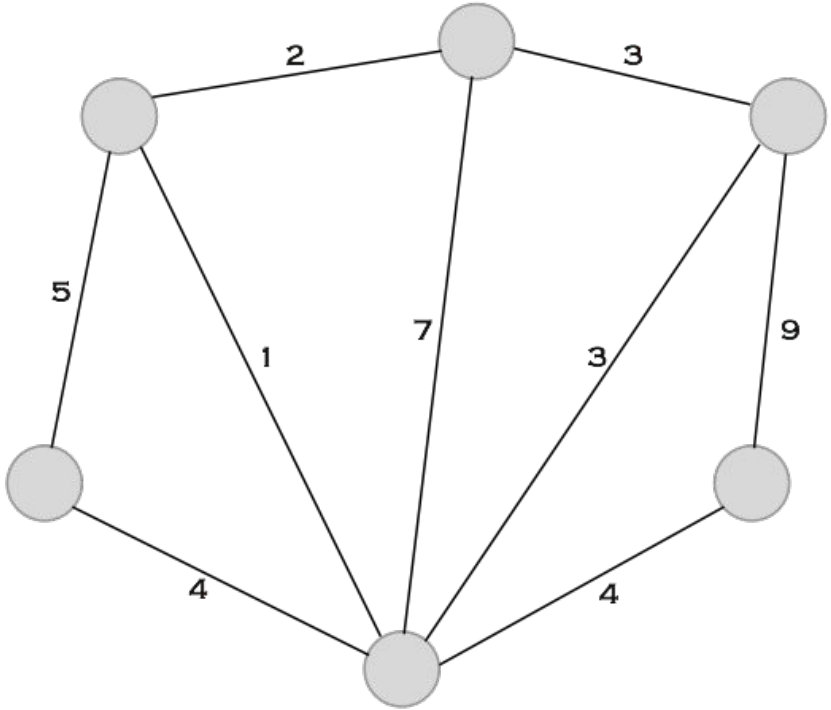
MST(Minimum spanning tree)

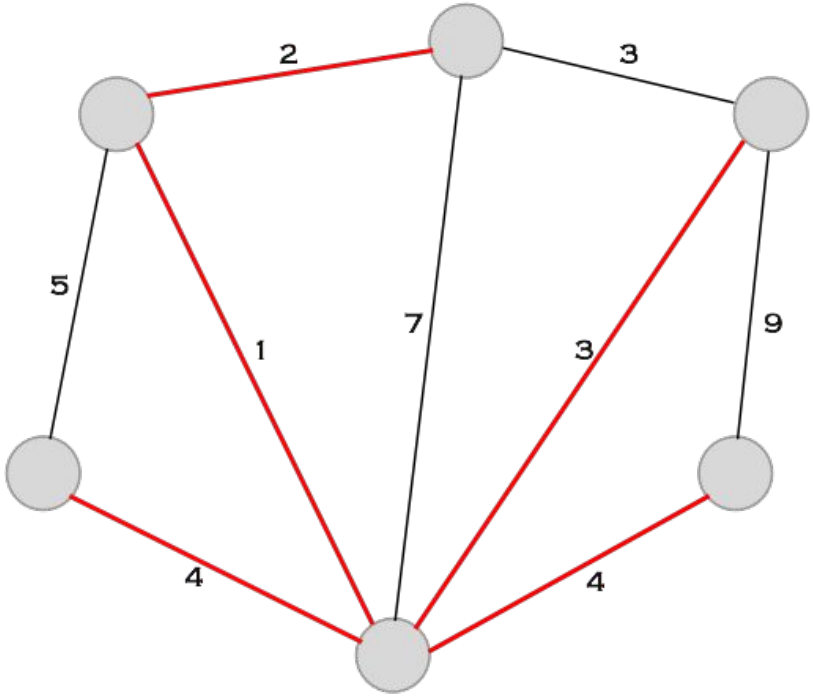
Minimum spanning tree

Uma MST é um conjunto de arestas de um grafo conectado, que conecta todos os vértices, sem formar ciclos e que a soma do peso dessas arestas seja o menor possível.

Exemplo

MaratonaCIn





MaratonaCIn

Kruskal

Kruskal

MaratonaCIn

Algoritmo:

1 - Ordene as arestas de acordo com o peso.

2 - Se os vértices da aresta estiverem na mesma componente, ignore e continue o algoritmo.

3 - Senão, inserir a aresta na MST

Kruskal

MaratonaCIn

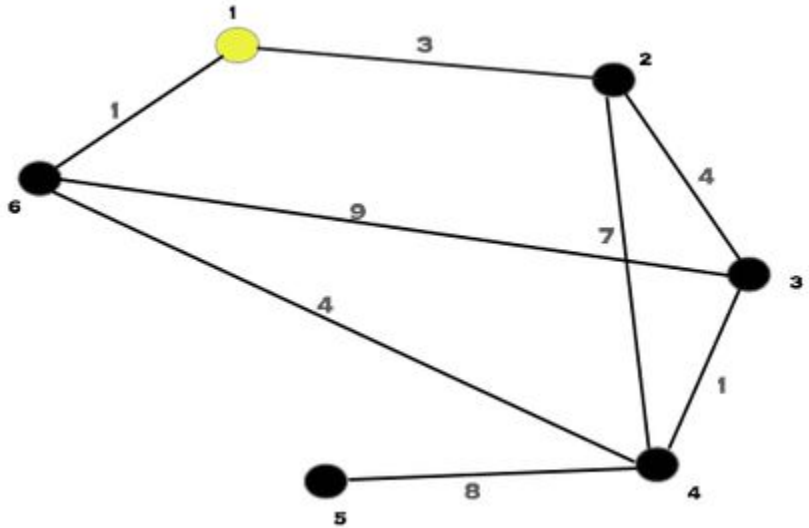
Código:

```
struct Edge{
	int u, v, cost;
	bool operator < (Edge b) {
		return cost < b.cost;
	}
};

void Kruskal() {
	vector<Edge> edge(m);
	for (int i = 0; i < m; i++) {
		cin >> edge[i].u >> edge[i].v >> edge[i].cost;
	}
	sort(edge.begin(), edge.end());
	long long total = 0;
	for (int i = 0; i < m; i++) {
		if (find(edge[i].u) != find(edge[i].v)) {
			merge(edge[i].u, edge[i].v);
			mst.push_back(edge[i]);
			total += edge[i].cost;
		}
	}
}
```

Prim

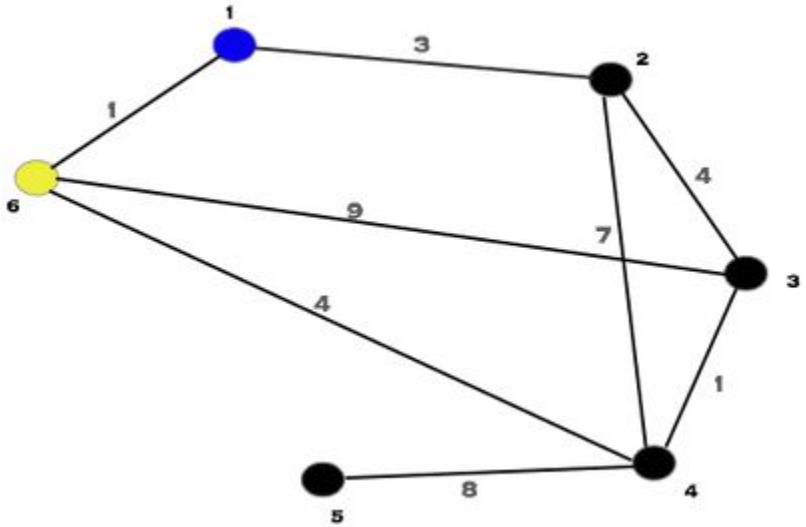
Prim



Nó	Distância
1	0
2	3
3	∞
4	∞
5	∞
6	1

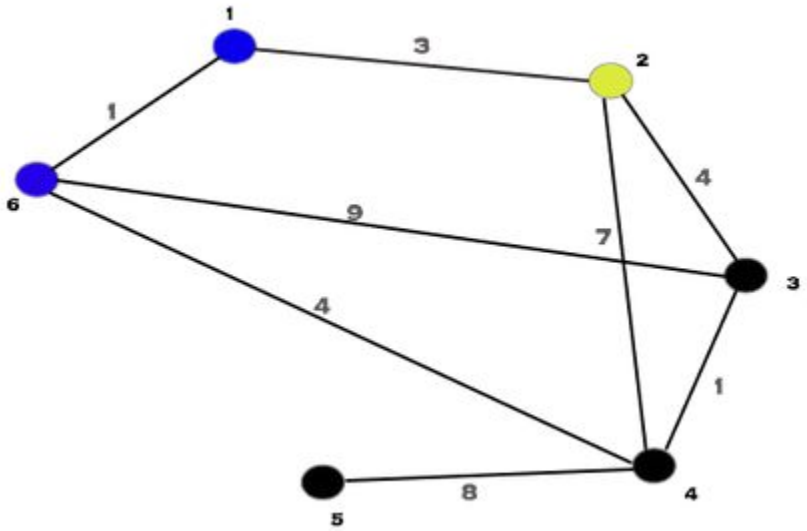
Prim

MaratonaCIn



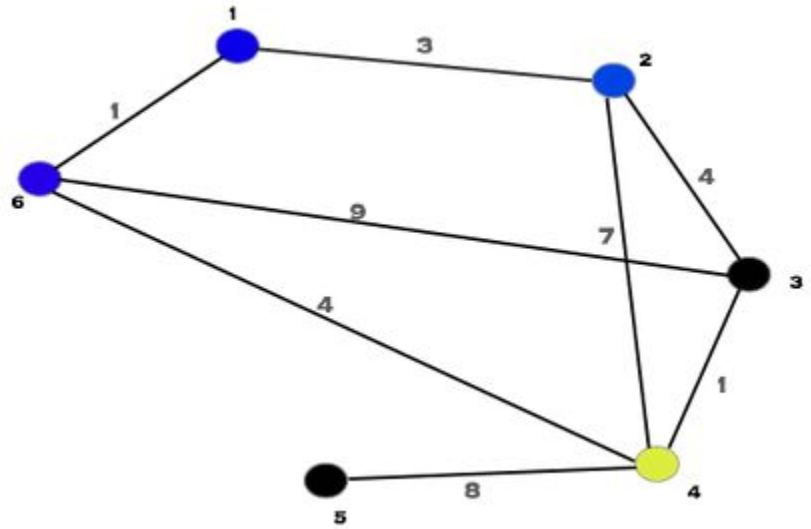
Nó	Distância
1	0
2	3
3	9
4	4
5	∞
6	1

Prim



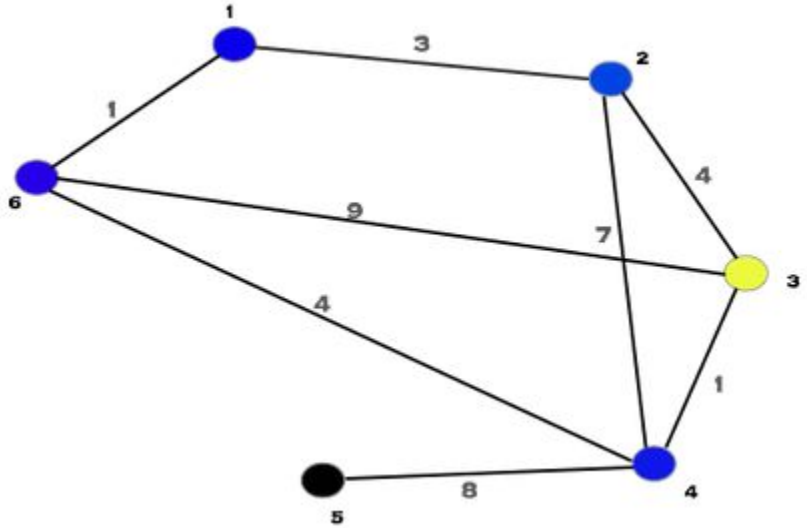
Nó	Distância
1	0
2	3
3	4
4	4
5	∞
6	1

Prim



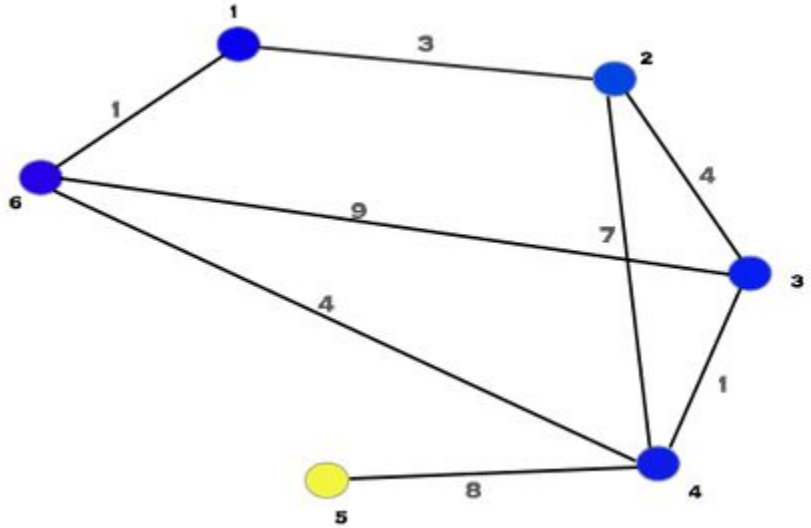
Nó	Distância
1	0
2	3
3	1
4	4
5	8
6	1

Prim



Nó	Distância
1	0
2	3
3	1
4	4
5	8
6	1

Prim



Nó	Distância
1	0
2	3
3	1
4	4
5	8
6	1

Prim

MaratonaCIn

Código:

```
long long Prim(int source = 0){
        for(int i = 0; i < n; i++) dist[i] = inf;
        dist[source] = 0;
        priority_queue<ii, vector<ii>, greater<ii>> pq;
        pq.emplace(0, source);
        while (!pq.empty()){
                int d = pq.top().first, u = pq.top().second;
                pq.pop();
                if (inMST[u]) continue;
                inMST[u] = true;

                for(auto e : adjList[u]){
                        int v = e.first, w = e.second;
                        if (!inMST[v] && w < dist[v]) {
                                // parent[v] = u;
                                dist[v] = w;
                                pq.emplace(dist[v], v);
                        }
                }
        }
}
```