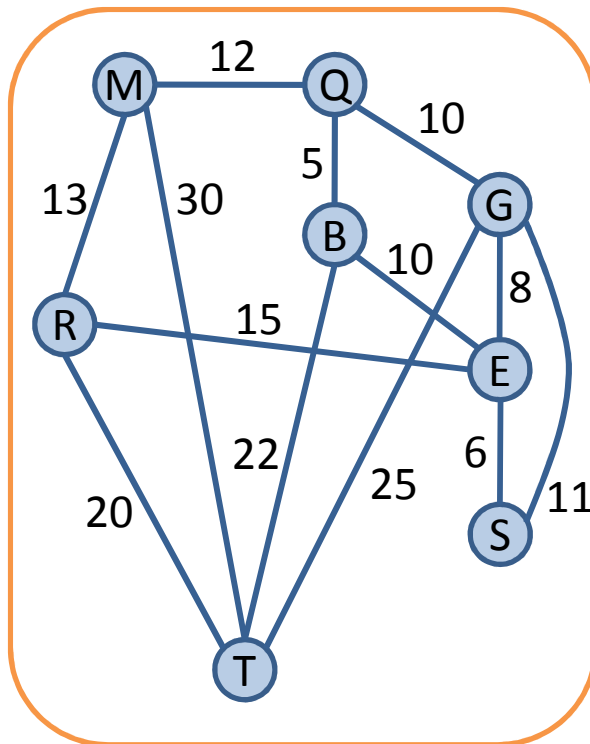
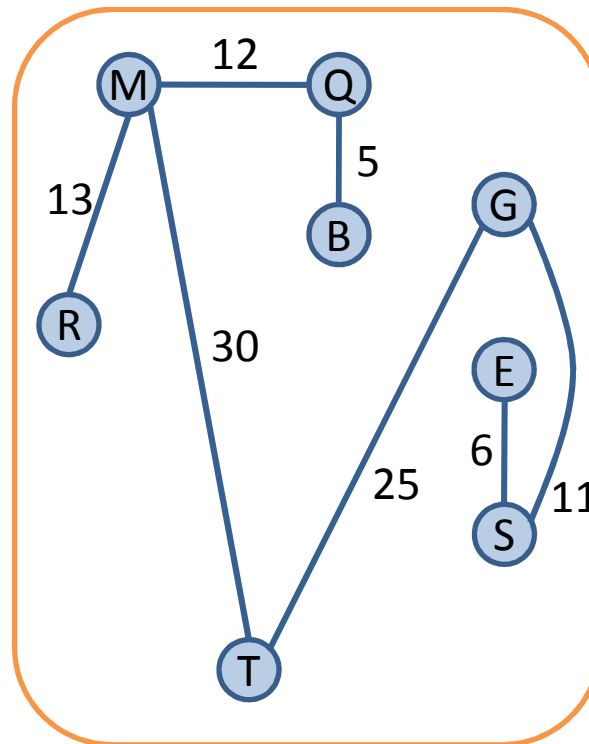


Árboles generadores de coste mínimo

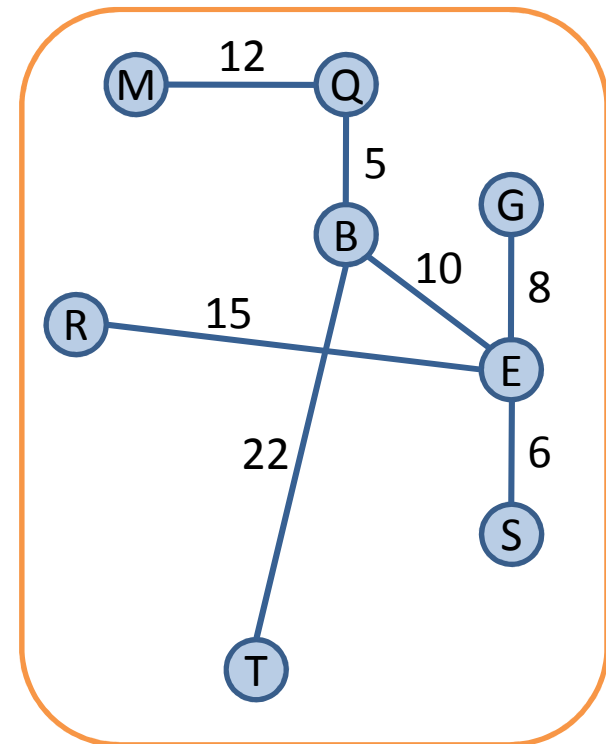
Dado un grafo no dirigido y conexo $G = (V, A)$, se define un **árbol generador (o de expansión) de G** como un árbol que conecta todos los vértices de V ; su coste es la suma de los costes de las aristas del árbol. Un árbol es un grafo conexo acíclico.



Grafo G



Árbol generador de G
Coste = 102



Árbol generador de G
Coste = 78

Árboles generadores de coste mínimo

Algoritmo de Prim

```
template <typename T> class GrafoP {
public:
    typedef T tCoste;
    typedef size_t vertice;
    struct arista {
        vertice orig, dest;
        tCoste coste;
        explicit arista(vertice v=vertice(), vertice w=vertice(),
                        tCoste c=tCoste()): orig(v), dest(w), coste(c) {}
        // Orden de aristas para Prim y Kruskall
        bool operator <(const arista& a) const
        { return coste < a.coste; }
    };
    // resto de miembros de la clase GrafoP<T> ...
};
```

```
#include "apo.h"
```

```
template <typename tCoste>
```

```
GrafoP<tCoste> Prim(const GrafoP<tCoste>& G)
```

```
// Devuelve un árbol generador de coste mínimo
```

```
// de un grafo no dirigido ponderado y conexo G.
```

{

```
typedef typename GrafoP<tCoste>::vertice vertice;
```

```
typedef typename GrafoP<tCoste>::arista arista;
```

```
const tCoste INFINITO = GrafoP<tCoste>::INFINITO;
```

```
arista a;
```

```
const size_t n = G.numVert();
```

```
GrafoP<tCoste> g(n);           // Árbol generador de coste mínimo.
```

```
vector<bool> U(n, false); // Conjunto de vértices incluidos en g.
```

```
Apo<arista> A( $n*(n-1)/2 - n + 2$ ); // Aristas adyacentes al árbol g
// ordenadas por costes.
```

```

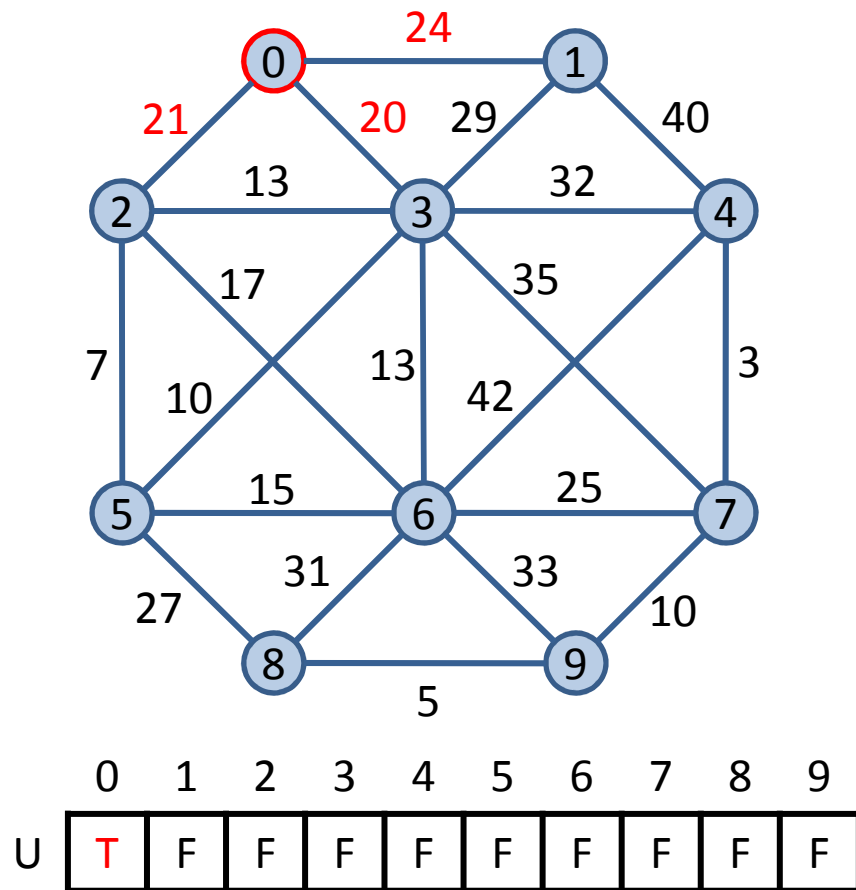
U[0] = true;    // Incluir el primer vértice en U.
// Introducir en el APO las aristas adyacentes al primer vértice.
for (vertice v = 1; v < n; v++)
    if (G[0][v] != INFINITO)
        A.insertar(arista(0, v, G[0][v]));
for (size_t i = 1; i <= n-1; i++) {    // Seleccionar n-1 aristas.
    // Buscar una arista a de coste mínimo que no forme un ciclo.
    do {
        a = A.cima();
        A.suprimir();
    } while (U[a.orig] && U[a.dest]);
    // Incluir la arista a en el árbol g y el nuevo vértice en U.
    g[a.orig][a.dest] = g[a.dest][a.orig] = a.coste;
    vertice u = U[a.orig] ? a.dest : a.orig;
    U[u] = true;
    // Introducir en el APO las aristas adyacentes al vértice u
    // que no formen ciclos.
    for (vertice v = 0; v < n; v++)
        if (!U[v] && G[u][v] != INFINITO)
            A.insertar(arista(u, v, G[u][v]));
}
return g;
}

```

Algoritmo de Prim

Ejemplo

Inicialización



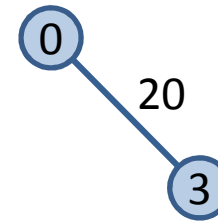
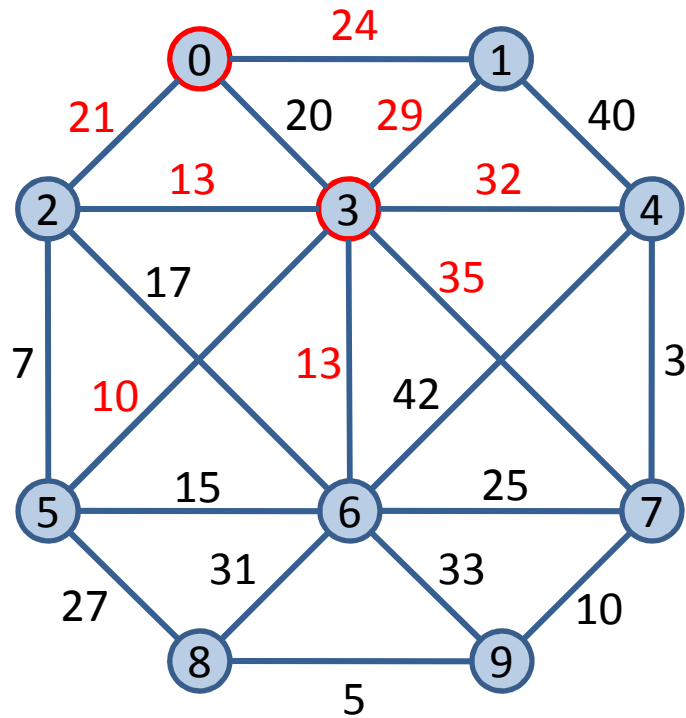
0

En **rojo** las aristas que hay en el APO A ordenadas por coste.

Algoritmo de Prim

Ejemplo

$i = 1$ $a = (0, 3, 20)$

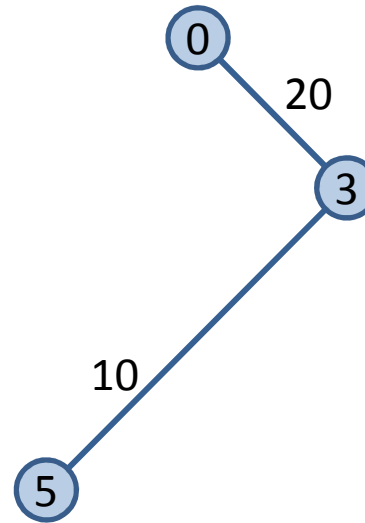
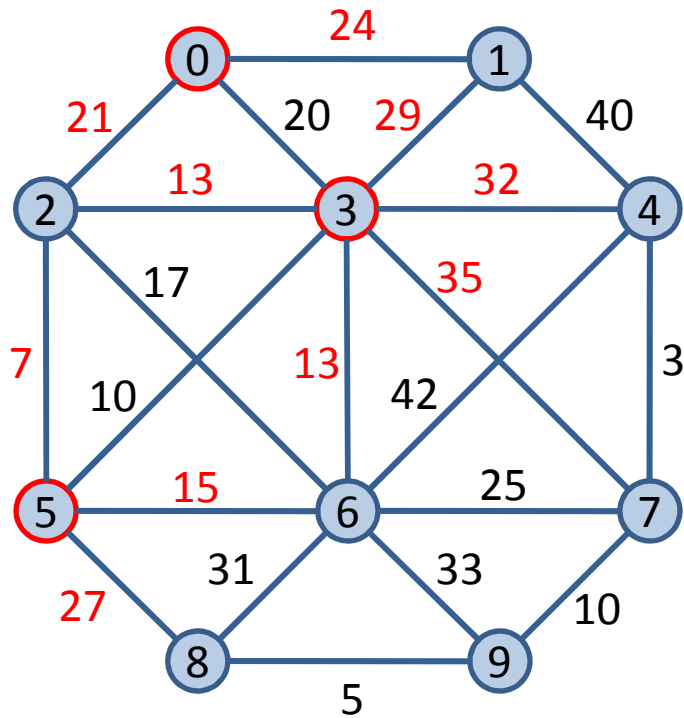


	0	1	2	3	4	5	6	7	8	9
U	T	F	F	T	F	F	F	F	F	F

Algoritmo de Prim

Ejemplo

$i = 2$ $a = (3, 5, 10)$

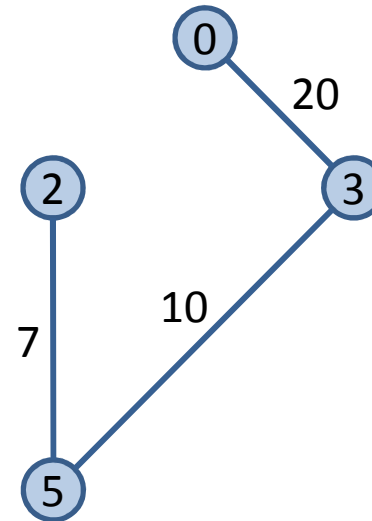
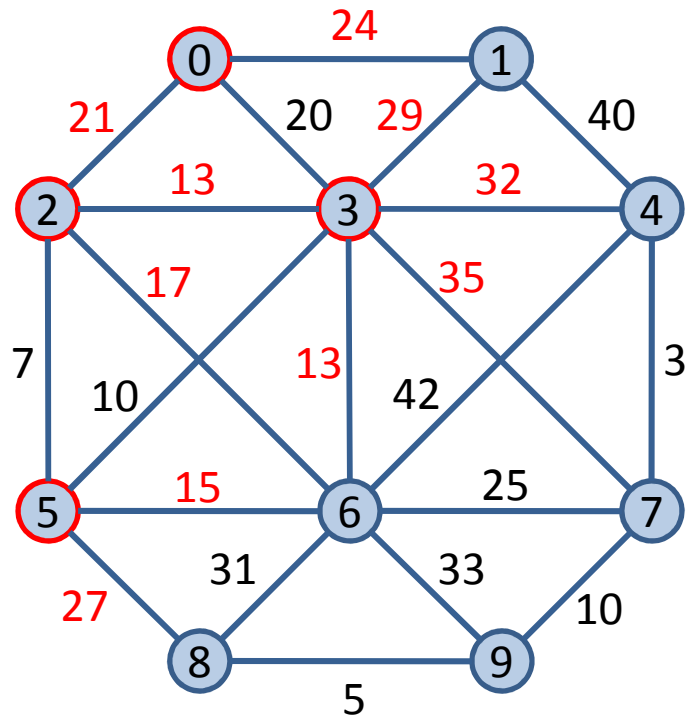


	0	1	2	3	4	5	6	7	8	9
U	T	F	F	T	F	T	F	F	F	F

Algoritmo de Prim

Ejemplo

$i = 3$ $a = (5, 2, 7)$

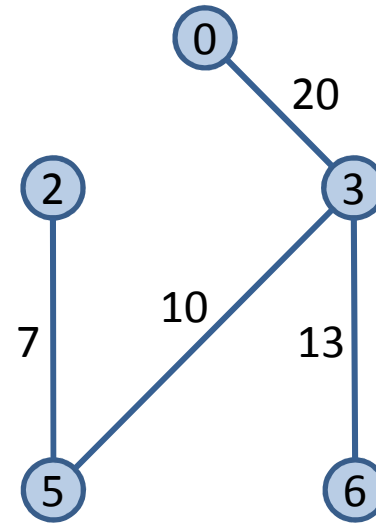
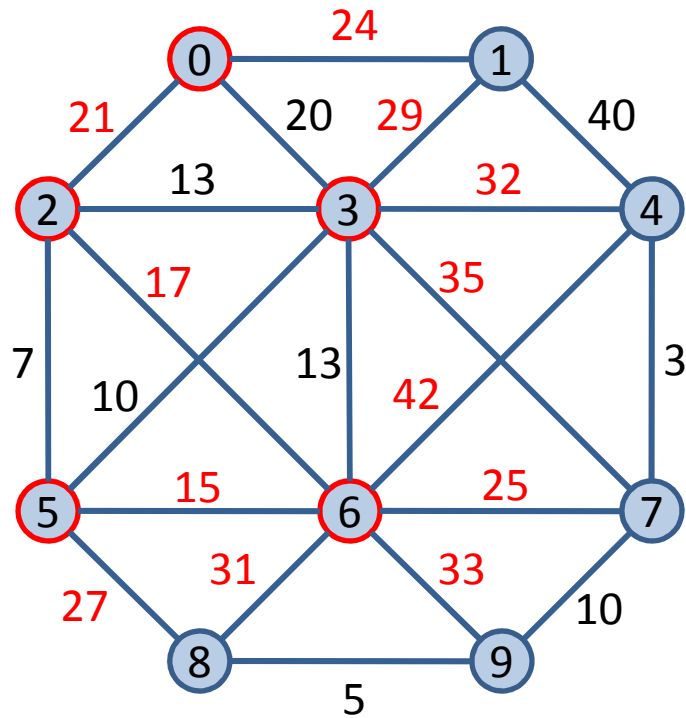


	0	1	2	3	4	5	6	7	8	9
U	T	F	T	T	F	T	F	F	F	F

Algoritmo de Prim

Ejemplo

$i = 4$ $a = (3, 6, 13)$

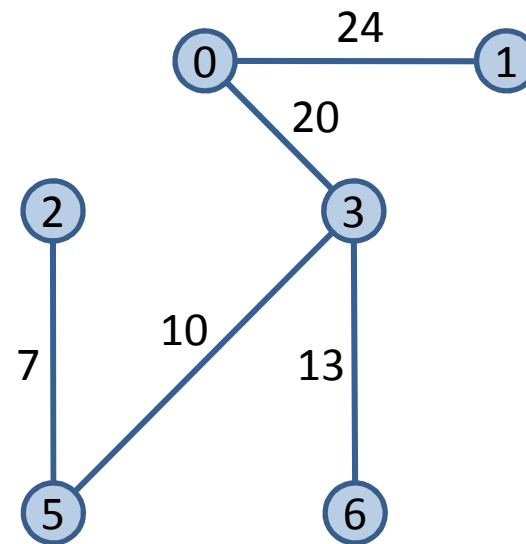
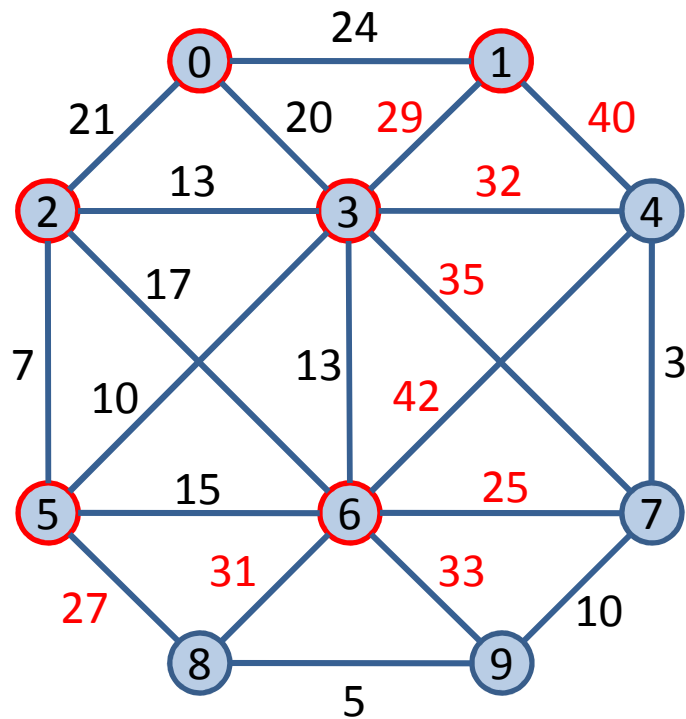


	0	1	2	3	4	5	6	7	8	9
U	T	F	T	T	F	T	T	F	F	F

Algoritmo de Prim

Ejemplo

$i = 5$ $a = (0, 1, 24)$

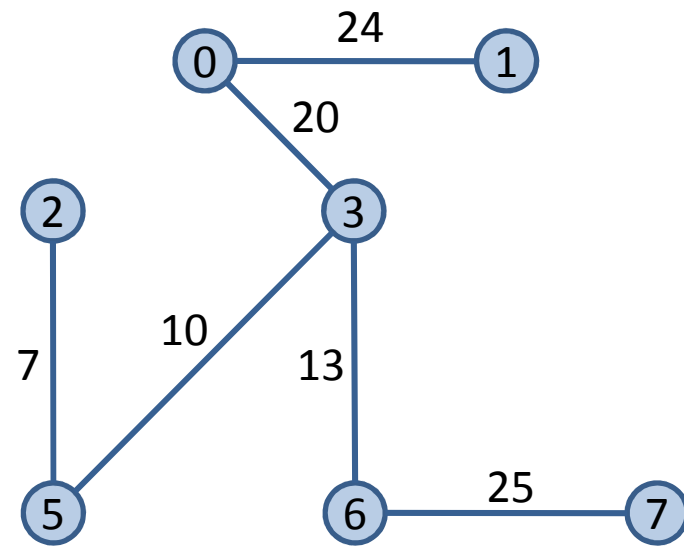
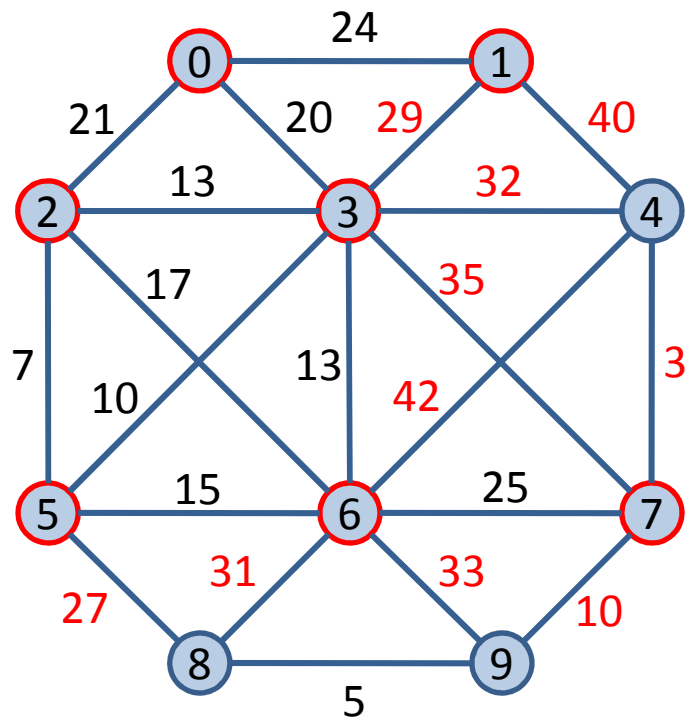


	0	1	2	3	4	5	6	7	8	9
U	T	T	T	T	F	T	T	F	F	F

Algoritmo de Prim

Ejemplo

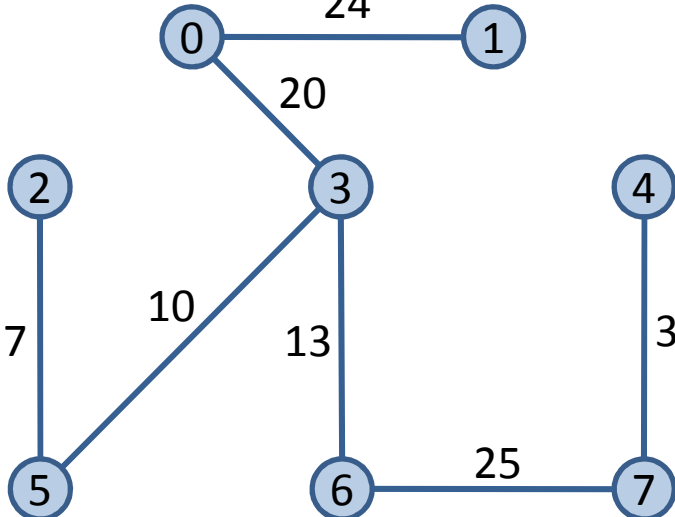
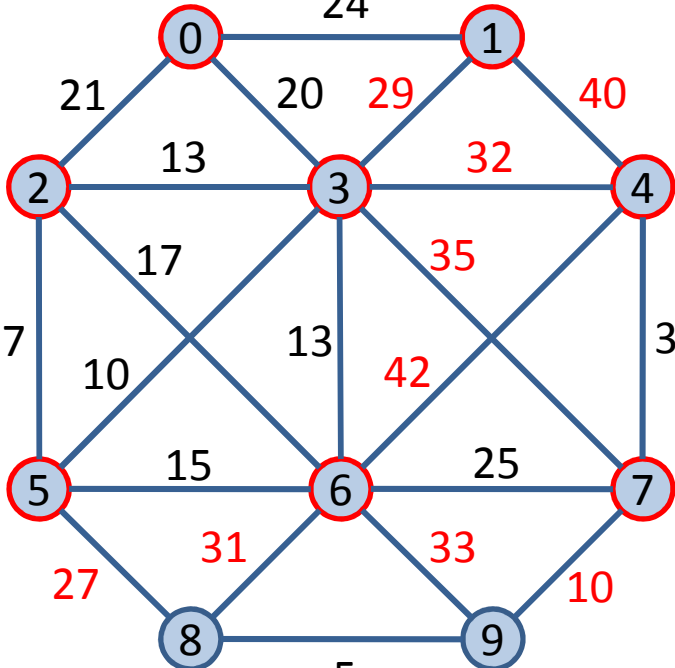
$i = 6$ $a = (6, 7, 25)$



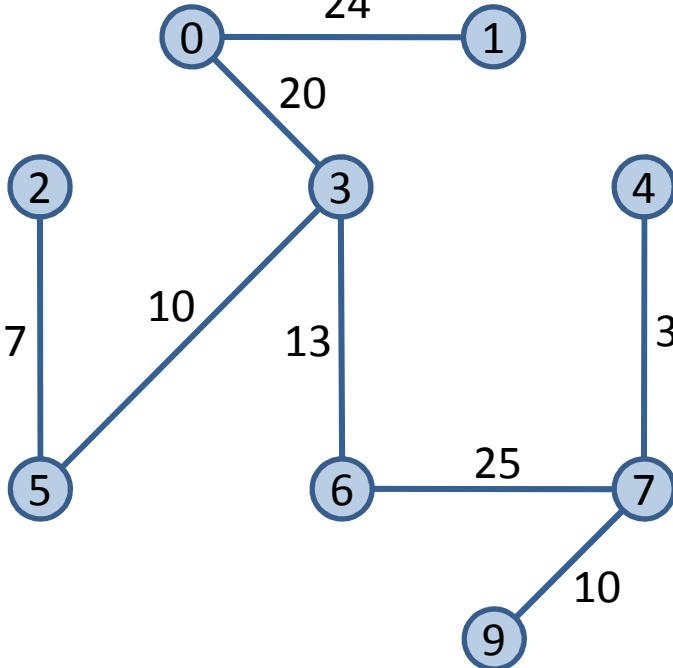
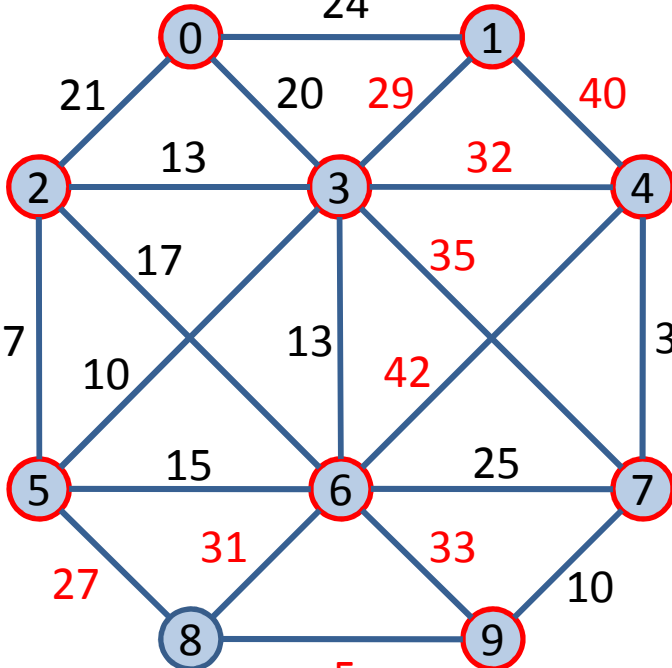
	0	1	2	3	4	5	6	7	8	9
U	T	T	T	T	F	T	T	T	F	F

Ejemplo

i = 7

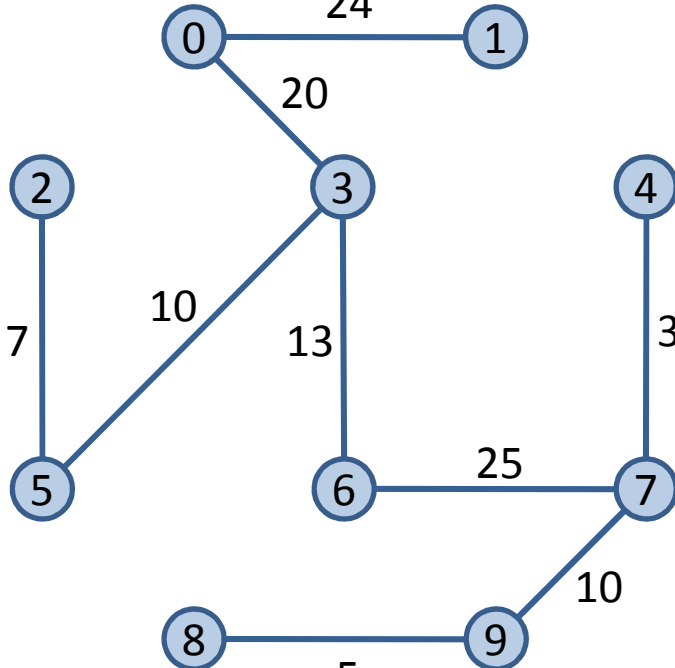
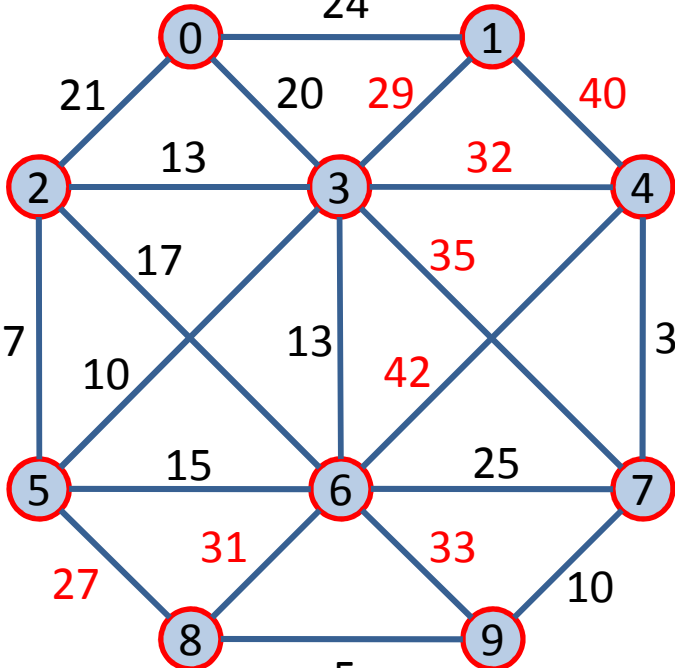
$$a = (7, 4, 3)$$
[illegible]

Ejemplo

$$a = (7, 9, 10)$$
[illegible]

Ejemplo

i = 9

$$a = (9, 8, 5)$$
[illegible]