Algoritmo Prim

EDNL

CAMAVINGAGII

2025

ÍNDICE GENERAL

CAPÍTULO 1

INTRODUCCIÓN

- Prim es otro algoritmo cuyo objetivo es el mismo que el de Kruskall, **conectar todos los nodos a coste mínimo sin ningún ciclo**.
- Para ello, partimos del tipo **arista**, que contiene nodo origen, destino y coste.
- También tenemos un vector **U**, un vector de booleanos que indica si los nodos están conectados o no.
- Utilizaremos también un **APO**, donde se irán almacenando **todas las aristas adyacentes a los diferentes nodos que vamos recorriendo**.
  Recordemos que **en la raíz de un APO se encuentra el elemento más pequeño**, por lo que el APO internamente pondrá en la raíz aquella arista que tenga coste menor (OJO, con coste menor, PERO NO SIGNIFICA QUE NO PUEDA FORMAR UN CICLO)
- Por último, tendremos un **grafo**, que obviamente, contendrá todos los nodos conectados a coste mínimo sin ciclos (que realmente, es un árbol)

CAPÍTULO 2

FUNCIONAMIENTO DE PRIM

- Prim sigue el siguiente procedimiento:
  1. Partes de un nodo origen, que puede ser cualquiera del grafo.
  2. Se añade al APO **todas las aristas adyacentes** al nodo origen.
  3. Posteriormente, **se coge la arista de menor peso que conecte dos nodos y que no forme un ciclo** (sabemos que nodo conecta la arista por el tipo arista previamente definido, que contiene nodo origen, destino y coste, se comprueba que no forma un ciclo mirando si en el vector de booleanos U, esa posición está a true o false)
  4. Añadimos este nuevo nodo al árbol (lo metemos en el grafo g, y lo ponemos en el vector de booleanos a true).
     A este nuevo nodo el cual conecta la arista elegida, sus aristas adyacentes se meten en el APO y se vuelve a buscar aquella arista con menor coste que una dos vértices sin que forme un ciclo (esta arista no tiene por qué ser una que parta del nuevo nodo añadido)
- Ejemplo:

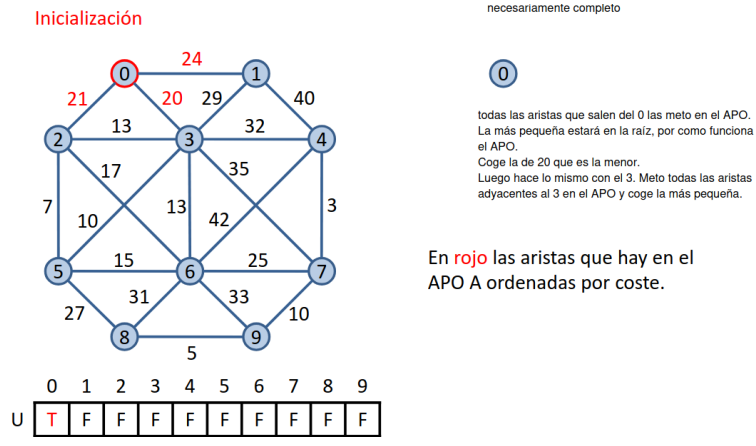

Figura 2.1: Ejemplo Prim

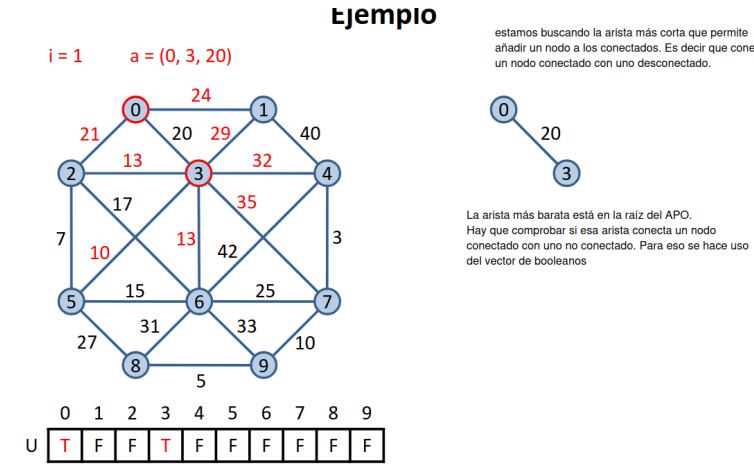

Figura 2.2: Ejemplo Prim

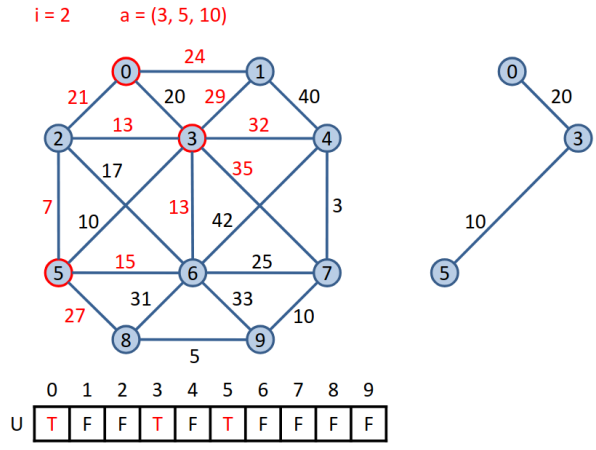

Figura 2.3: Ejemplo Prim

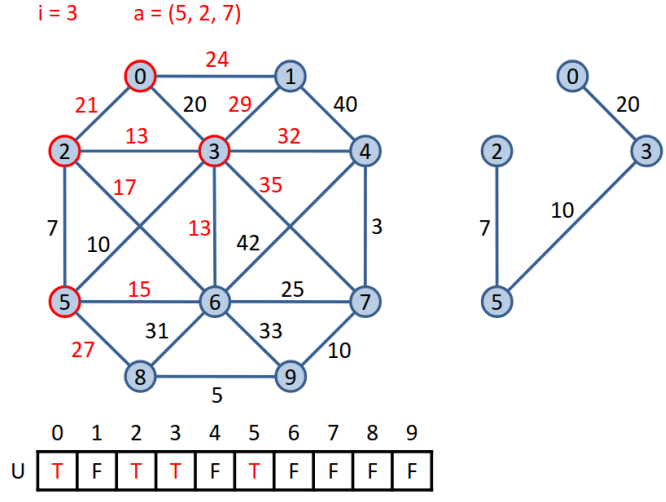

Figura 2.4: Ejemplo Prim

CAPÍTULO

3

CÓDIGO

- A continuación se da la implementación del algoritmo de Prim con comentarios para entender mejor su funcionamiento.

```
#include 'apo.h'

template <typename tCoste>
GrafoP<tCoste> Prim(const GrafoP<tCoste>& G)
// Devuelve un arbol generador de coste minimo
// de un grafo no dirigido ponderado y conexo G.
{
typedef typename GrafoP<tCoste>::vertice vertice;
typedef typename GrafoP<tCoste>::arista arista; //tipo arista con origen
    destino y coste
const tCoste INFINITO = GrafoP<tCoste>::INFINITO;
arista a;
const size_t n = G.numVert();
GrafoP<tCoste> g(n); // Arbol generador de coste minimo.
vector<bool> U(n, false); // Conjunto de vertices incluidos en g.
Apo<arista> A(n*(n-1)/2-n+2); // Aristas adyacentes al arbol g  ordenadas
     por costes.


U[0] = true; // Incluir el primer vertice en U, conectamos el primer
    vertice
// Introducir en el APO las aristas adyacentes al primer vertice.
for (vertice v = 1; v < n; v++)
if (G[0][v] != INFINITO)
A.insertar(arista(0, v, G[0][v])); //0 origen, v destino, G[0][v] coste

for (size_t i = 1; i <= n-1; i++) { // Seleccionar n-1 aristas.
// Buscar una arista a de coste minimo que no forme un ciclo.
// Nota: Las aristas en A tienen sus origenes en el arbol g.
```

```
27 do { //en este bucle desecho aristas inservibles, y me quedo con la mas
       pequena que no forme ciclos
28 a = A.cima(); //cojo la arista mas pequena
29 A.suprimir(); //la elimino del grafo
30 } while (U[a.dest]); // a forma un ciclo (a.orig y a.dest estan en U y en
       g). verifico que no forma ciclo
31 // Incluir la arista a en el arbol g y el nuevo vertice u en U.
32 g[a.orig][a.dest] = g[a.dest][a.orig] = a.coste; //metemos la arista en
       el grafo (mas bien su coste)
33 vertice u = a.dest; //obtenemos el vertice destino de la arista
34 U[u] = true; //lo ponemos a verdadero
35 // Introducir en el APO las aristas adyacentes al vertice u
36 // que no formen ciclos.
37 for (vertice v = 0; v < n; v++)
38 if (!U[v] && G[u][v] != INFINITO)
39 A.insertar(arista(u, v, G[u][v]));
40 }
41 return g;
42 }
```