Apuntes Examen Final EDA

Anexo 1: Algoritmo de Prim

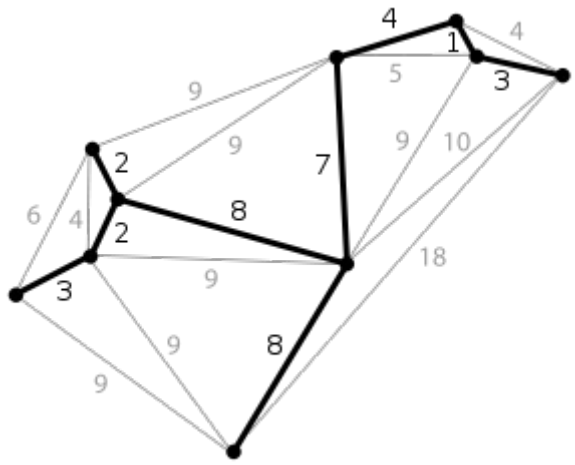
Grafos

Algoritmo de Prim

Este algoritmo encuentra el **árbol de expansión mínimo** de un **grafo no dirigido con pesos**

MST (minimum spanning tree)

El MST de un grafo **G** es un subgrafo **G'** que es un árbol y conecta todos los vértices de **G**



Las aristas del MST son las de trazo grueso

El algoritmo de Prim puede resumirse en 3 pasos:

- Inicializar el árbol **G'** con un vértice de **G** elegido al azar

- Añadir al árbol **G'** la arista de menor peso que conecta un vértice de **G'** con otro vértice que aún no pertenezca a G'

- Repetir el anterior paso hasta que **G'** contenga todos los vértices de **G**

Complejidad: **O(|E| log |V|)**

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;


typedef vector<int> IV;
typedef pair<int, int> IP;
typedef vector<IP> IPV;
typedef vector<IPV> G;

// struct que representa una arista
struct E {
  int w;     // peso de la arista
  int x, y;  // vertices de la arista

  // operador menor-que
  // se necesita para la priority queue, que debe ordenar por el peso 'w'
  inline friend bool operator<(const E &a, const E &b) {
    return a.w > b.w;
  }
};
typedef priority_queue<E> EQ;


G g;     // grafo: lista de adyacencias (peso, vertice adyacente)
IV par;  // array de padres del MST


void prim() {
  // inicializar el array de padres
  // al principio, ningun vertice tiene padre (indicado con -1)
  par = IV(g.size(), -1);
  // inicializar la cola de prioridad
  // añadimos una arista "fantasma" que une el vertice origen consigo mismo
  // asi conseguimos que este vertice origen sea la raiz del MST
  EQ que;
  que.push({ 0, 0, 0 });
  while (!que.empty()) {
    // la arista que sacamos es la de menor peso de todas las que hemos examinado
    const E &e = que.top();
    int x = e.x;
    int y = e.y;
    que.pop();
    if (par[y] == -1) {
      // si el vertice 'y' no tenia padre, se lo asignamos ahora
      // luego añadimos a la cola todas las aristas que conectan con 'y'
      par[y] = x;
      for (const IP &a : g[y]) {
        int w = a.first;
        int z = a.second;
        que.push({ w, y, z });
      }
    }
  }
}
```