CURSO DE PROGRAMACIÓN COMPETITIVA
URJC - 2018

Sesión 4 (6ª Semana)

David Morán (ddavidmorang@gmail.com)
Juan Quintana (juandavid.quintana@urjc.es)
Sergio Pérez (sergioperezp1995@gmail.com)

Contenidos

- Grafos (Ponderados)
    - Introducción
    - Representación
    - Árbol de mínimo recubrimiento; Algoritmos de Prim y Kruskal

Grafos Ponderados

- En la vida real, el camino de un nodo a otro no siempre se cuenta como el número de aristas que cruza
- Autopista vs Carreteras cuando viajas entre dos ciudades, ¿Cuál es mejor?

Grafos Ponderados

- Un grafo ponderado es aquel que entre dos nodos existe un peso entre su arista, este peso suele ser un número
- Para representarlo, tenemos dos implementaciones

Grafos Ponderados

- Un array de vectors (arrayList) de una estructura llamada arista

```
struct edge{
  int from, to, weight;
  edge(){}
  edge(int f, int t, int w){
    from = f; to = t; weight = w;
  }
}; vector<edge> Graph[MAXN];
```

Grafos Ponderados

- Un array de vectors (arrayList) de un par (C++)

```
vector<pair<int, int> > Graph[MAXN];
```

Grafos Ponderados

- En Java un array de ArrayList de aristas:

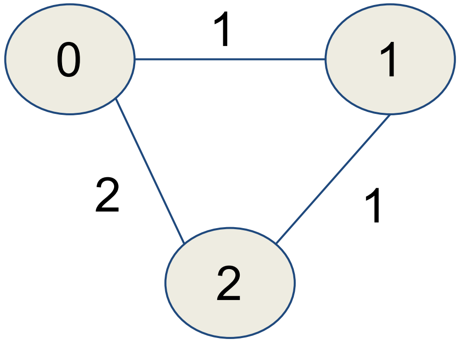
```
class Edge {
   public int from, to, weight;
   public Edge(int f, int t, int w){
      from = f; to = f; weight = w;
   }
}

ArrayList<Edge>[] graph = new ArrayList[MAXN];
```

Grafos Ponderados

- Por simplicidad, trabajaremos con la primera
- También podemos representar las aristas con peso mediante una matriz de adyacencia

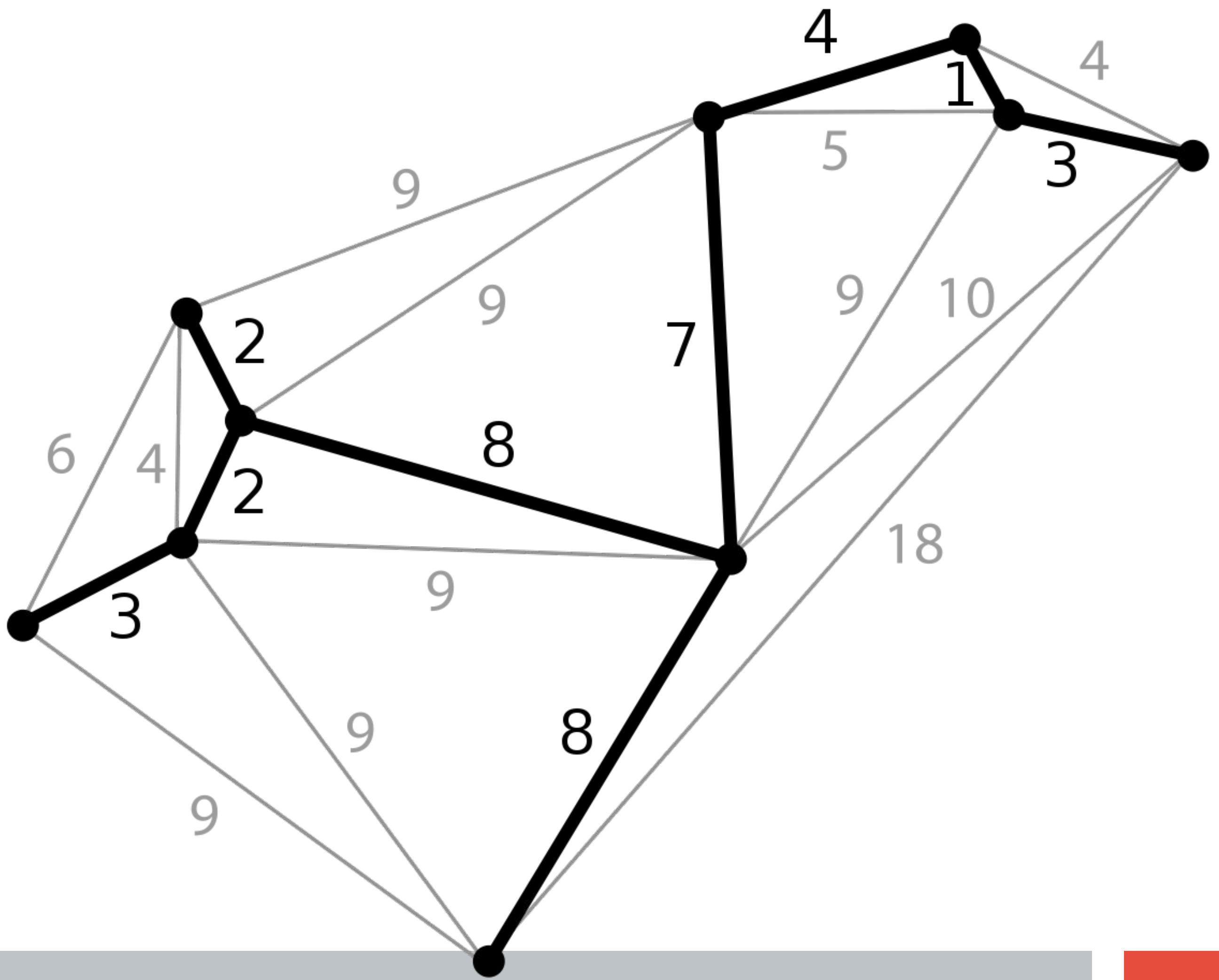
```
0 1 2
1 0 1
2 1 0
```



Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo voraz
- Dado un grafo, generar un arbol de N nodos minimizando la suma de sus N-1 aristas
- Nos centraremos en dos algoritmos conocidos:
  - Prim (cola de prioridad)
  - Kruskal (estructura Union-Find)

Grafos Ponderados
Árbol de recubrimiento mínimo



Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Prim (parecido a un BFS)
- Inicializamos un array de distancias a INF (inicializamos un array de visitados a 0)
- Ponemos un nodo arbitrario en la cola de prioridad
- Si el peso de la arista es menor a su valor dentro del array de distancias, encolamos (si el nodo no está visitado, encolamos)

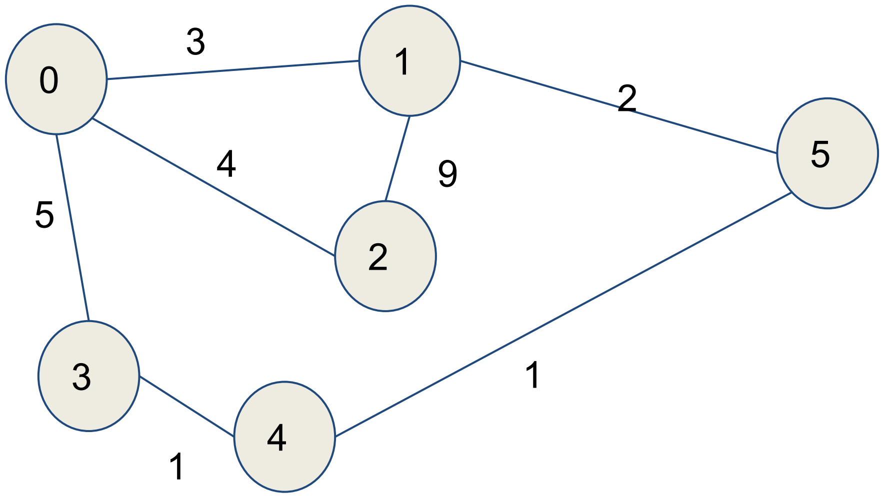
Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Prim

```
function prim(start):
  array_dist = set(INF)
  array_dist[start] = 0
  pq.encolar({start, 0})
  mientras no pq.vacía:
    N = pq.desencolar()
    si N.distancia <= array_dist[N.nodo]:
      para cada arista en N:
        si array_dist[N.to] > N.weight:
          array_dist[N.to] = N.weight
          pq.encolar({N.to, N.weight})
```

Grafos Ponderados
Árbol de recubrimiento mínimo

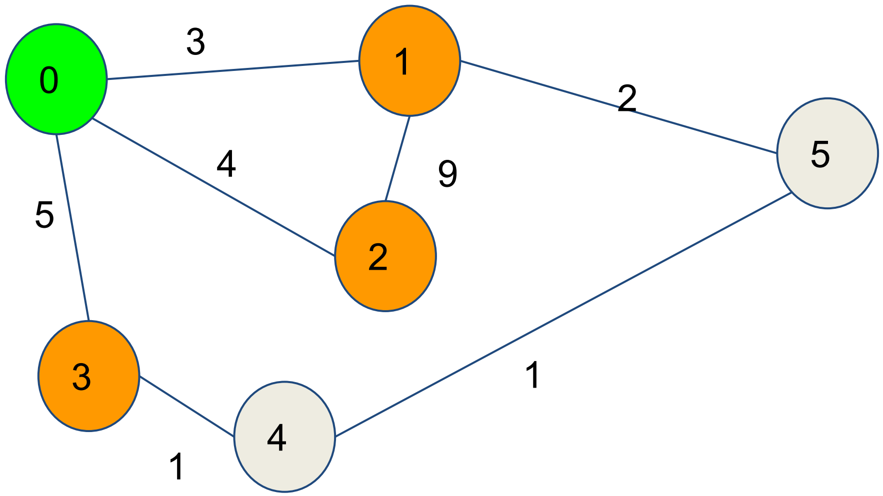
- Algoritmo de Prim



[0, INF, INF, INF, INF, INF], PQ = { 0 }

Grafos Ponderados
Árbol de recubrimiento mínimo

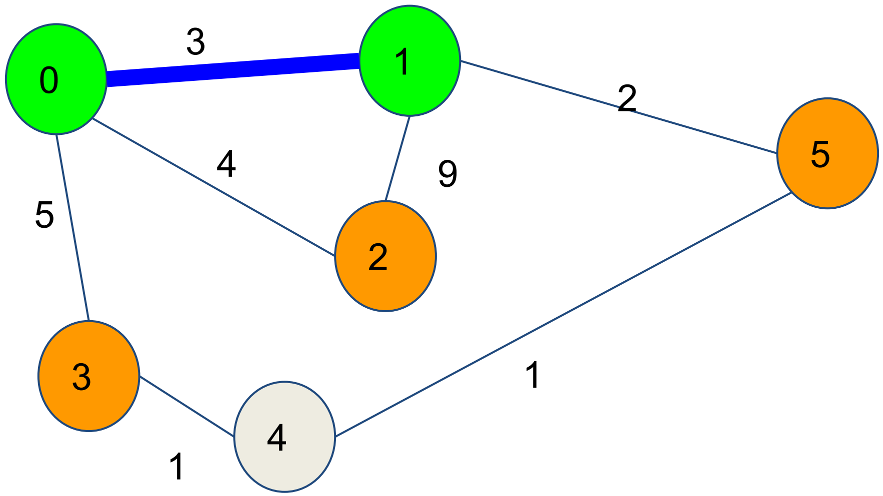
- Algoritmo de Prim



[0, 3, 4, 5, INF, INF], PQ = { (1,3), (2,4), (3,5) }

Grafos Ponderados
Árbol de recubrimiento mínimo

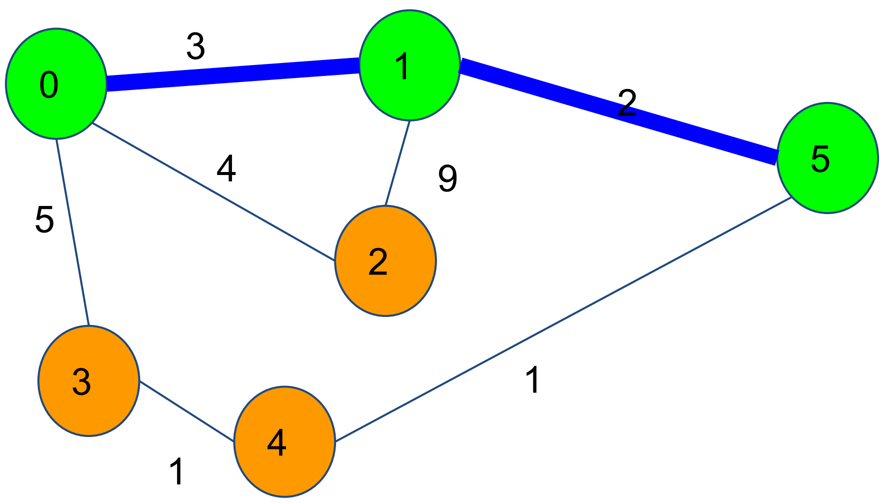
- Algoritmo de Prim



[0, 3, 4, 5, INF, 2], PQ = { (5,2), (2,4), (3,5) }

Grafos Ponderados
Árbol de recubrimiento mínimo

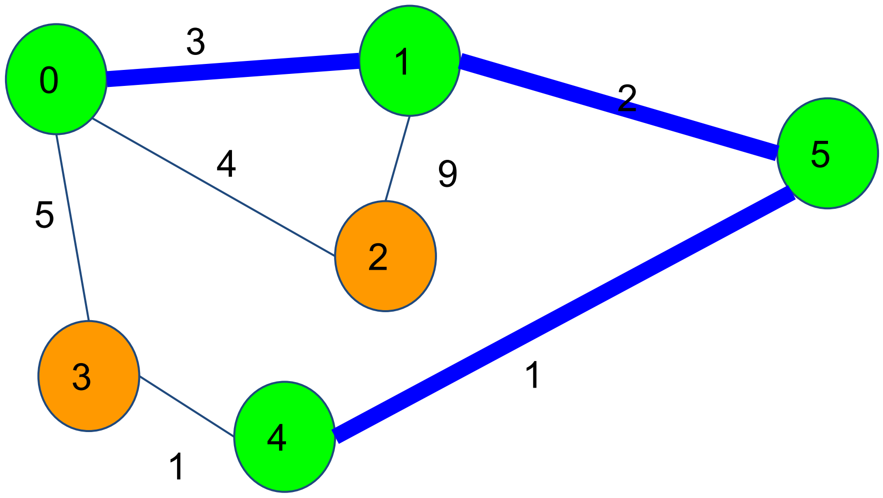
- Algoritmo de Prim



[0, 3, 4, 5, 1, 2], PQ = { (4,1), (2,4), (3,5) }

Grafos Ponderados
Árbol de recubrimiento mínimo

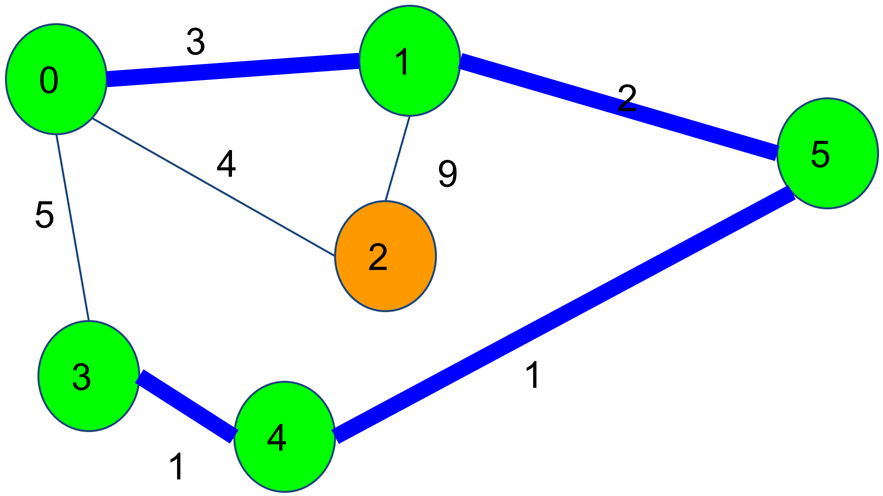
- Algoritmo de Prim



[0, 3, 4, 1, 1, 2], PQ = { (3,1), (2,4), (3,5) }

Grafos Ponderados
Árbol de recubrimiento mínimo

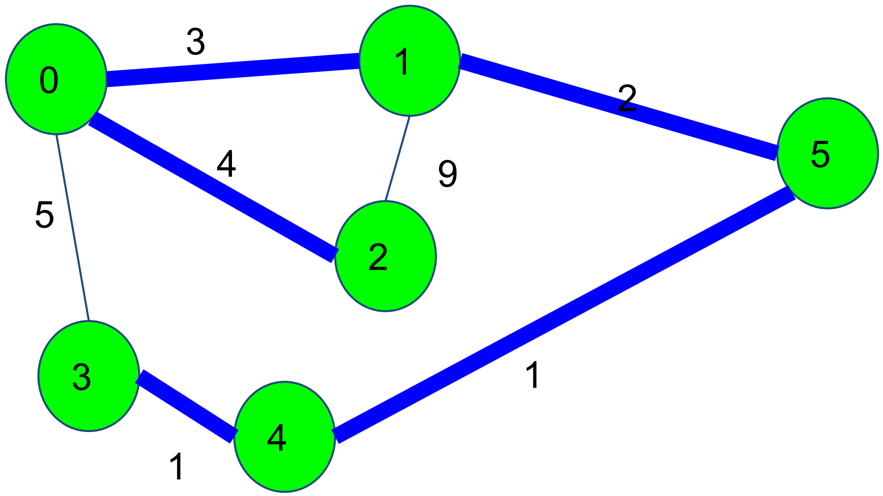
- Algoritmo de Prim



[0, 3, 4, 1, 1, 2], PQ = { (2,4), (3,5) }

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Prim



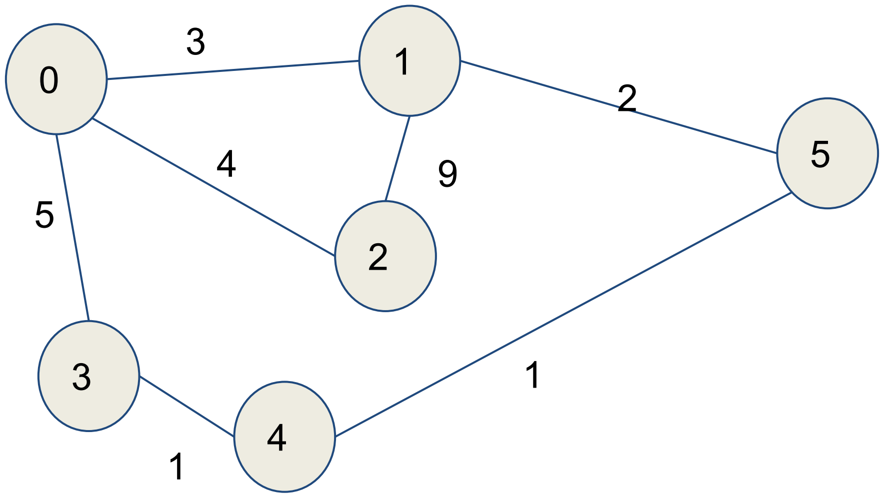
[0, 3, 4, 1, 1, 2], PQ = { (3,5) }

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal
- Partimos de una lista de aristas
- Ordenamos las aristas de menor a mayor
- Asumimos un grafo totalmente disconexo con cada nodo aislado
- Si para una arista de 0..N-1 con pares de nodos i,j no están conectados, conectamos i,j
- Conjuntos Disjuntos (Disjoint Sets)

Grafos Ponderados
Árbol de recubrimiento mínimo

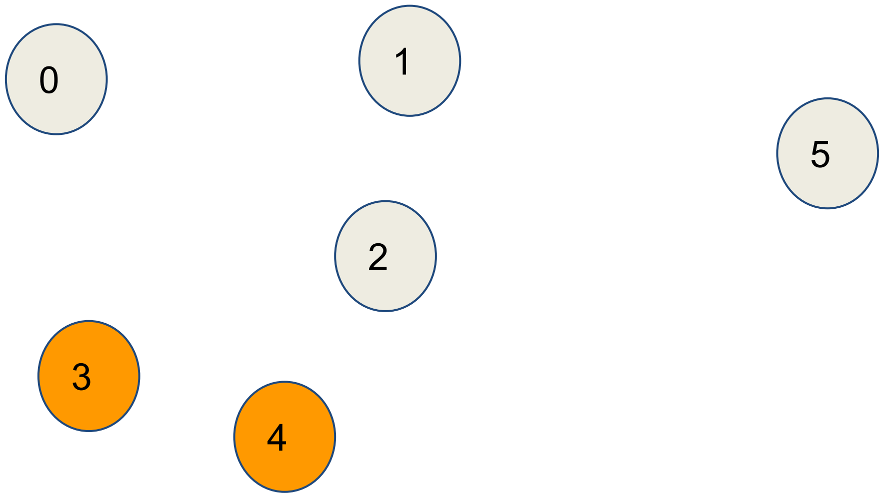
- Algoritmo de Kruskal



{0,1,3} {0,2,4} {0,3,5} {1,5,2} {1,2,9} {3,4,1} {5,4,1}

Grafos Ponderados
Árbol de recubrimiento mínimo

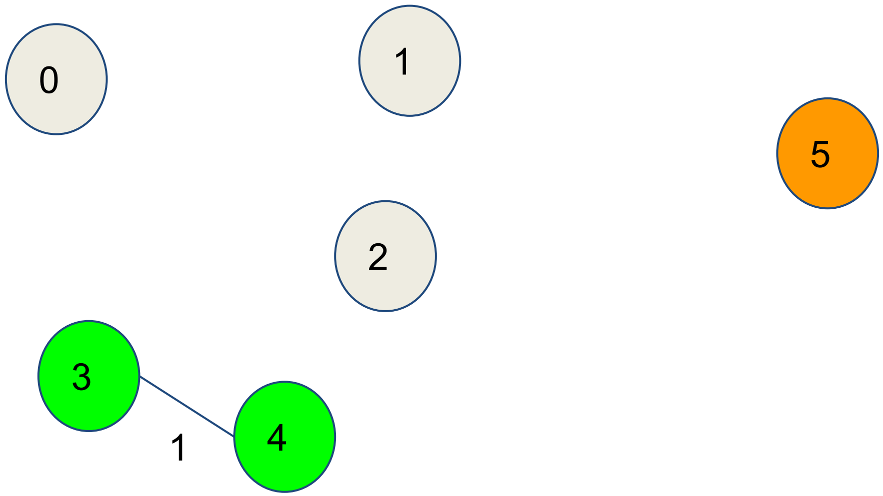
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

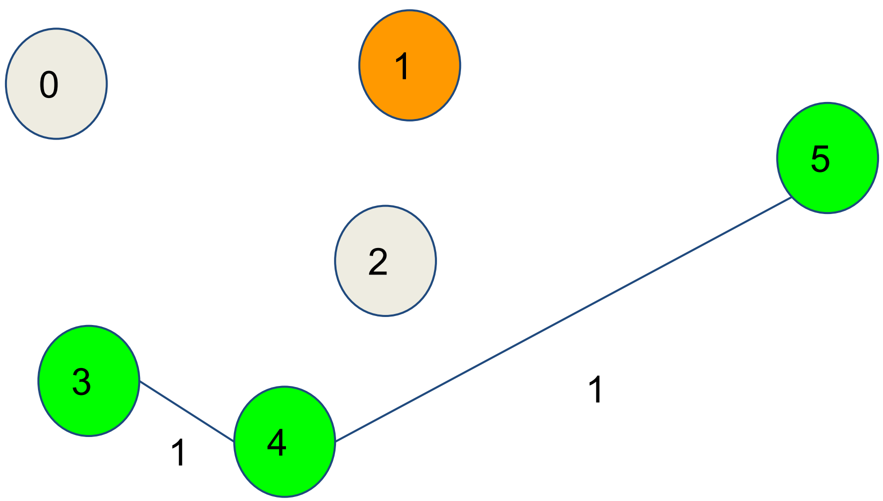
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

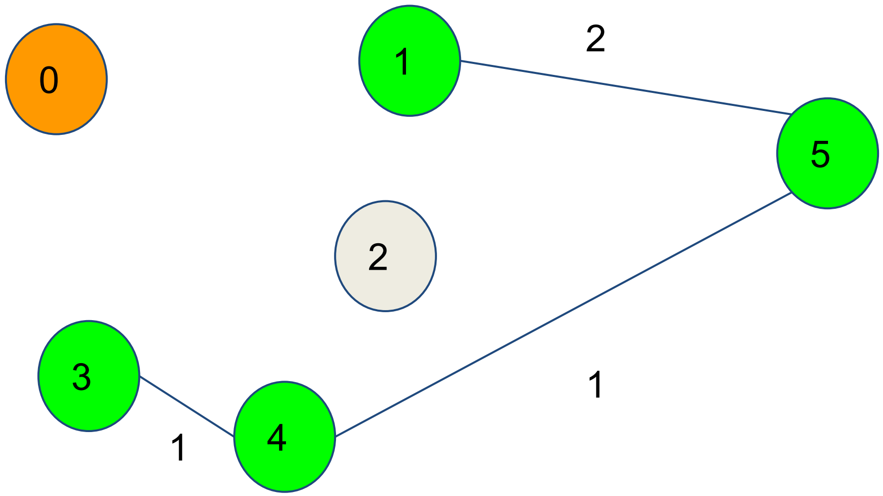
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

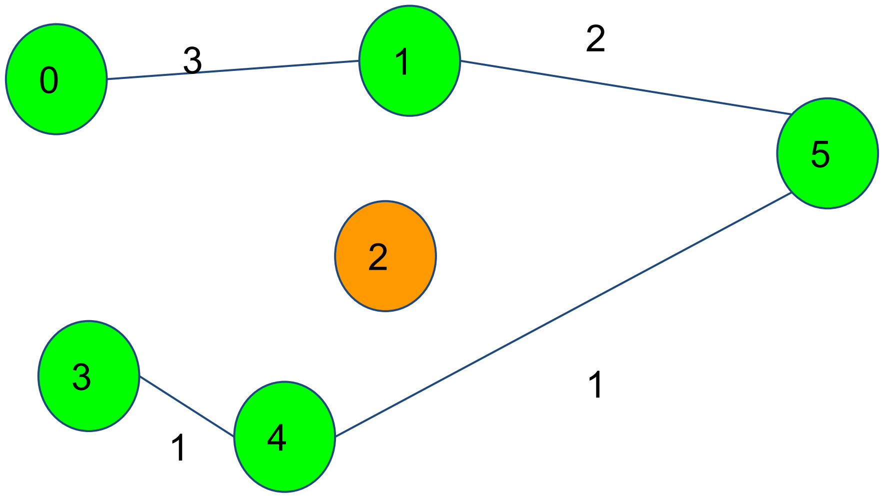
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

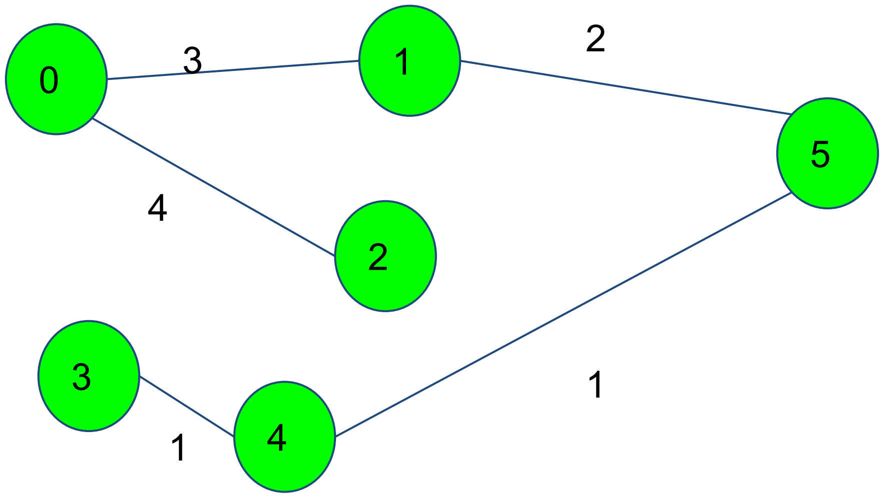
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

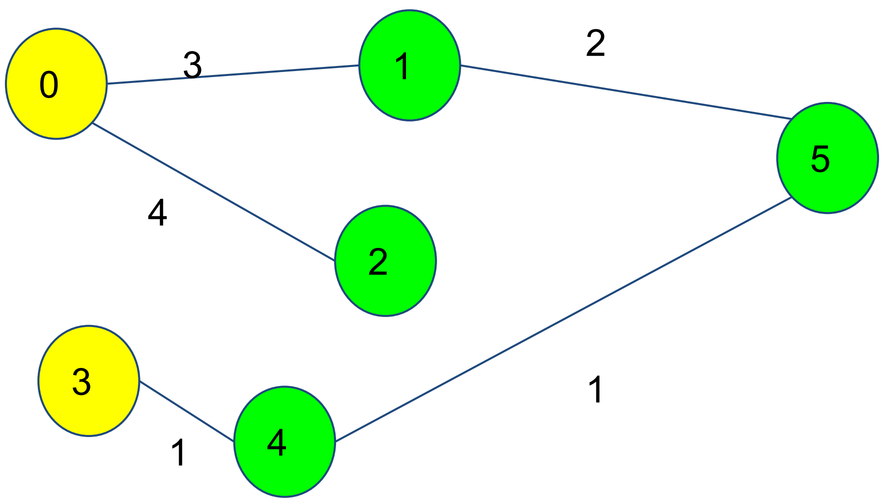
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

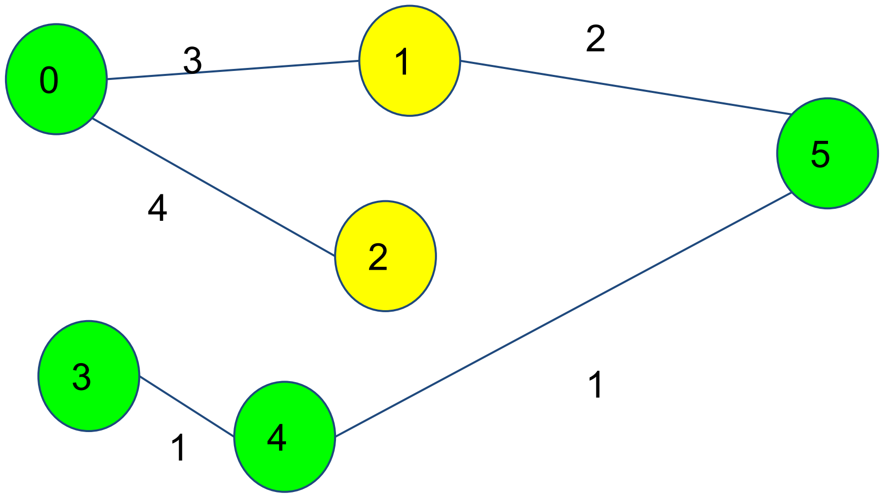
- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal



{3,4,1} {5,4,1} {1,5,2} {0,1,3} {0,2,4} {0,3,5} {1,2,9}

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal
- Verificar componentes conexas cada vez que comparamos dos nodos es muy costoso (N^2)
- Existe una estructura que nos facilita mucho esta comprobación llamada Union-Find
- Se basa en el mismo principio que Kruskal sobre conjuntos disjuntos

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal
- Llevamos cuenta en un array de quien es el padre del i-ésimo nodo (en inicio todos son padres de sí mismos)
- Llevamos cuenta en otro array del tamaño de cada componente (1 en principio)

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal
- Para unir las componentes vamos a comparar qué componente es la más grande
- La componente más grande absorbe a la más pequeña y el padre de la más pequeña pasa a ser el padre de la grande

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal

```
class UF{
   array parents[N]
   array size[N]
   for i from 0..N
      parents[i] = i
      size[i] = 1
.
.
.
```

- Algoritmo de Kruskal

```
class UF{ …
  function findRoot(n):
    if(n == parents[n]) return n
    return findRoot(parents[n])
.
.
.
```

{ 0, 3, 2, 4, 0 }

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal

```
class UF{ …
   function isConnected(a, b):
      return findRoot(a) == findRoot(b)
.
.
.
```

{ 0, 3, 2, 4, 0 }

Grafos Ponderados
Árbol de recubrimiento mínimo

- Algoritmo de Kruskal

```
class UF{ …
  function connect(a, b):
    if(isConnected(a, b)) return
    aRoot = findRoot(a)
    bRoot = findRoot(b)
    if size[aRoot] > size[bRoot]:
      size[aRoot] += size[bRoot]
      parents[bRoot] = aRoot
    else
      size[bRoot] += size[aRoot]
      parents[aRoot] = bRoot
```

{ 0, 3, 2, 4, 0 }

Semana que viene...

- Semana que viene:
    - Problemas prácticos sobre árboles de recubrimiento mínimo
    - Algoritmos de camino más corto; Floyd Warshall y Dijkstra

¡Hasta la próxima semana!

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (preferiblemente copia a los tres). También podéis utilizar el slack del curso:

https://urjc-cp.slack.com

David Morán (ddavidmorang@gmail.com)
Juan Quintana (juandavid.quintana@urjc.es)
Sergio Pérez (sergioperezp1995@gmail.com)