

Grafos etiquetados

Algoritmos en grafos ponderados

- Existen muchas situaciones donde además del modelo de grafo, se requiere además asociar un peso/longitud a las aristas del grafo.
- Estos grafos se denominan grafos ponderados, grafos etiquetados.
- Importantes problemas prácticos se abordan por medio de grafos ponderados. Por ejemplo:
 - Encontrar los caminos más cortos en un grafo
 - Encontrar el árbol de cubrimiento mínimo

Grafos no dirigidos con pesos

<code>EdgeWeightedGraph(V)</code>	// constructor
<code>int V()</code>	// Número de vértices
<code>int E()</code>	// Número de aristas
<code>void addEdge(Edge e)</code>	// Agregar una arista
<code>Iterable<Edge> adj(int v)</code>	// Aristas adyacentes a v
<code>int degree(int v)</code>	// Grado de un vértice

Implementación

<code>Edge(int v, int w, double w)</code>	// constructor
<code>int either()</code>	// Número de vértices
<code>int other(int v)</code>	// Número de aristas
<code>double weight()</code>	// Agregar una arista

Implementación

Grafos dirigidos con pesos

<code>EdgeWeightedDigraph(V)</code>	<code>// constructor</code>
<code>int V()</code>	<code>// Número de vértices</code>
<code>int E()</code>	<code>// Número de aristas</code>
<code>void addEdge(Edge e)</code>	<code>// Agregar una arista</code>
<code>Iterable<DirectedEdge> adj(int v)</code>	<code>// Aristas adyacentes a v</code>
<code>int outdegree(int v)</code>	<code>// Grado saliente de un vértice</code>
<code>int indegree(int v)</code>	<code>// Grado entrante a un vértice</code>

Implementación

<code>DirectedEdge(int v, int w, double w)</code>	<code>// constructor</code>
<code>int from()</code>	<code>// Número de vértices</code>
<code>int to()</code>	<code>// Número de aristas</code>
<code>double weight()</code>	<code>// Agregar una arista</code>

Implementación

Camino más corto

- En un grafo dirigido con pesos positivos, interesa encontrar el camino más corto desde un origen s a un nodo destino.
- Una de las soluciones más conocidas es el algoritmo de Dijkstra.
- Este algoritmo encuentra todas las rutas más cortas de s a los demás nodos.

Algoritmo de Dijkstra (1)

- Se mantienen tres estructuras auxiliares:
 - `distTo[i]` : Longitud del camino más corto de s a i .
 - `edgeTo[i]` : Última arista en el camino de s a i . Inicializado en `null`.
 - `pq` : Cola de prioridad mínima con los vértices indexados en función de su distancia al origen.

```
public class DijkstraSP {  
    private double[] distTo;  
    private DirectedEdge[] edgeTo;  
    private IndexMinPQ<Double> pq;  
  
    ...  
}
```

Algoritmo de Dijkstra (2)

- Inicialización
 - `distTo[i]` : Longitud de los caminos $s \rightarrow i$ inicialmente infinito.
 - `edgeTo[i]` : Inicialmente no se han identificado aristas en los caminos más cortos: `null`.
 - `pq` : El vértice origen `s`.

```
distTo = new double[G.V()];  
edgeTo = new DirectedEdge[G.V()];  
  
for (int v = 0; v < G.V(); v++)  
    distTo[v] = Double.POSITIVE_INFINITY;  
distTo[s] = 0.0;  
  
pq = new IndexMinPQ<Double>(G.V());  
pq.insert(s, distTo[s]);
```

Algoritmo de Dijkstra (3)

- Proceso de “relajación” de aristas:
 - Al considerar la arista $v \rightarrow w$, si la longitud del camino a w pasando por v es más corta:
 - actualizar la `distTo[w]` con la nueva distancia
 - actualizar la última arista hasta w con la arista $v \rightarrow w$.
 - actualizar la cola de prioridad acorde a la nueva distancia hasta w .

```
private void relax(DirectedEdge e) {  
    int v = e.from(), w = e.to();  
    if (distTo[w] > distTo[v] + e.weight()) {  
        distTo[w] = distTo[v] + e.weight();  
        edgeTo[w] = e;  
        if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);  
        else  
            pq.insert(w, distTo[w]);  
    }  
}
```


Algoritmo de Dijkstra (4)

- El ciclo central del algoritmo procede mientras la cola de prioridad no este vacía.
- Se obtiene el menor vértice de la cola y se relajan todas las aristas salientes de este vértice.

```
while (!pq.isEmpty()) {  
    int v = pq.delMin();  
    for (DirectedEdge e : G.adj(v))  
        relax(e);  
}
```

Dijkstra: Caso ejemplo

