Grafos etiquetados

Algoritmos en grafos ponderados

- Existen muchas situaciones donde además del modelo de grafo, se requiere además asociar un peso/longitud a las artistas 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

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

Implementación

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

Implementación

Grafos **dirigidos** con pesos

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

Implementación

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

Implementación

Caminos más cortos

- 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→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→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→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