Caminos más cortos

Bellman-Ford

Motivación

- Otra alternativa para encontrar “el árbol de caminos más cortos” en un grafo, desde un vértice inicial *s*.
- Permite aristas con longitudes positiva, negativa o cero.
- Presenta la ventaja que fácilmente puede llevarse a una implementación distribuida: Un ejemplo práctico de esta situación es el problema del enrutamiento en redes, e.g. RIP.

Bellman-Ford

- Asume un grafo dirigido con pesos.
- Encuentra las rutas más cortas desde un vértice inicial *s* a todos los demás vértices, o
- Si el grafo tienen ciclos negativos, los detecta.

Estructuras utilizadas por el algoritmo

- Similares a las usanda en Dijkstra:

```
distTo  = new double[G.V()];
edgeTo  = new DirectedEdge[G.V()];
onQueue = new boolean[G.V()];
Queue<Integer> queue;
```

Inicialización

- Distancia inicial infinita
- Aristas entrantes: null
- Cola: Nodo origen

```
for (int v = 0; v < G.V(); v++)
    distTo[v] = Double.POSITIVE_INFINITY;
distTo[s] = 0.0;

queue = new Queue<Integer>();
queue.enqueue(s);
onQueue[s] = true;
```

Operación de ‘relajación’

- Invocada cada que se visita un nodo.

```
private void relax(EdgeWeightedDigraph G, int v) {
        for (DirectedEdge e : G.adj(v)) {
            int w = e.to();
            if (distTo[w] > distTo[v] + e.weight()) {
                distTo[w] = distTo[v] + e.weight();
                edgeTo[w] = e;
                if (!onQueue[w]) {
                    queue.enqueue(w);
                    onQueue[w] = true;
                }
            }
            if (++cost % G.V() == 0) {
                findNegativeCycle();
                if (hasNegativeCycle()) return;
            }
        }
    }
```

Bucle central

- Sobre cada uno de los nodos en la cola, aplicar el proceso de relajación.
- Observar diferencias con Dijkstra:
  - Cola simple, no en orden de distancia
  - Un mismo nodo podría visitarse muchas veces

```
while (!queue.isEmpty() && !hasNegativeCycle()) {
     int v = queue.dequeue();
     onQueue[v] = false;
     relax(G, v);
}
```

Implementación completa

Complejidad de Bellman-Ford

- En el caso de un grafo acíclico, cada arista se relaja como máximo V-1 veces.
- De lo contrario, el grafo contiene un ciclo.
- Esto lleva a un tiempo total de peor caso:

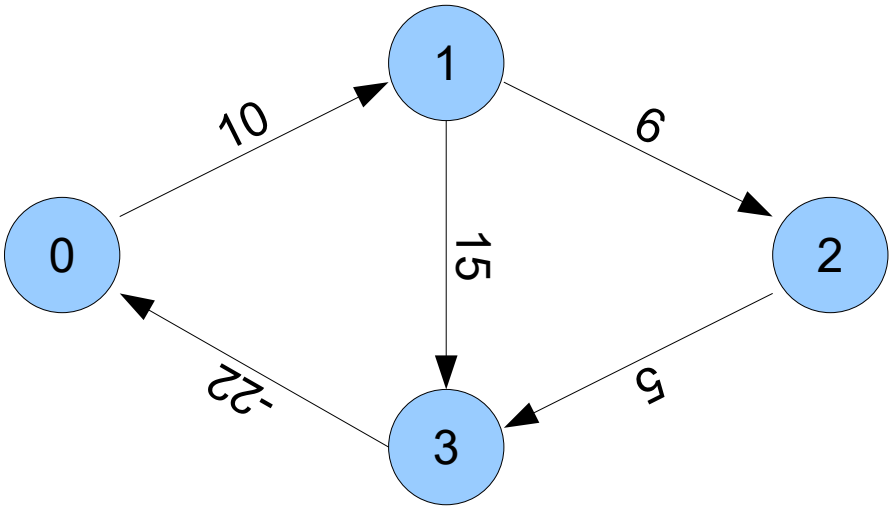
$$\sim VE$$

- en la práctica suele ser mucho menor.

Encontrando ciclos negativos

- Un ciclo negativo tiene el efecto de hacer que Bellman-Ford entre en un ciclo sin fin.

e.g.



Detección de ciclos negativos

- Cada que se completan V iteraciones del algoritmo de relajación (se han relajado V aristas) se invoca el algoritmo de detección de ciclos.
- El primer paso en la detección de ciclos es construir un grafo auxiliar que solo contiene las aristas predecesoras encontradas hasta el momento.
- Un segundo paso detecta el ciclo utilizando un recorrido DFS.

Grafo de predecesores

```
private void findNegativeCycle() {
    int V = edgeTo.length;
    EdgeWeightedDigraph spt = new EdgeWeightedDigraph(V);
    for (int v = 0; v < V; v++)
        if (edgeTo[v] != null)
            spt.addEdge(edgeTo[v]);

    EdgeWeightedDirectedCycle finder = new EdgeWeightedDirectedCycle(spt);
    cycle = finder.cycle();
}
```

Búscando ciclos mediante DFS

```
public EdgeWeightedDirectedCycle(EdgeWeightedDigraph G) {
    marked  = new boolean[G.V()];
    onStack = new boolean[G.V()];
    edgeTo  = new DirectedEdge[G.V()];
    for (int v = 0; v < G.V(); v++)
        if (!marked[v]) dfs(G, v);
}
```

```
private void dfs(EdgeWeightedDigraph G, int v) {
    onStack[v] = true;
    marked[v] = true;
    for (DirectedEdge e : G.adj(v)) {
        int w = e.to();
        if (cycle != null) return;
        else if (!marked[w]) {
            edgeTo[w] = e;
            dfs(G, w);
        }
        else if (onStack[w]) {
            cycle = new Stack<DirectedEdge>();
            DirectedEdge f = e;
            while (f.from() != w) {
                cycle.push(f);
                f = edgeTo[f.from()];
            }
            cycle.push(f);
            return;
        }
    }
    onStack[v] = false;
}
```

Implementación distribuida

- Cada nodo (*e.g.* cada enrutador) envía un mensaje a sus vecinos: Vector de distancias.
- Cuando w recibe el vector de distancias de v, comprueba si utilizando a v como intermediario hay una ruta más corta (paso de relajación). En caso afirmativo, actualiza su vector de distancias y lo distribuye a sus vecinos.