PageRank

- Técnica para medir la importancia de los nodos de un grafo.
- Se base en que los nodos importantes van a recibir links de otros nodos importantes.
- Se basa en la idea de simular un navegante aleatorio en la web.
- Es un algoritmo iterativo.

PageRank de un nodo n en la iteración t:

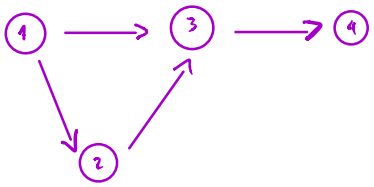
$$Pr_t(n) = \frac{1-d}{N} + d \cdot \sum_{\substack{\text{nodos } n' \\ \text{que apuntan} \\ \text{a } n}} \frac{Pr_{t-1}(n')}{\text{OutDegree}(n')}$$

Donde:

- d: damping factor (en general 0.85)
- N: número total de nodos

En la iteración inicial el PageRank de todos los nodos es el mismo (en general $\frac{1}{N}$).

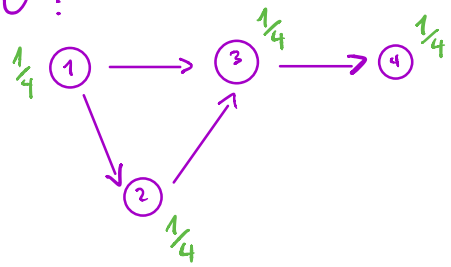
Ejemplo:



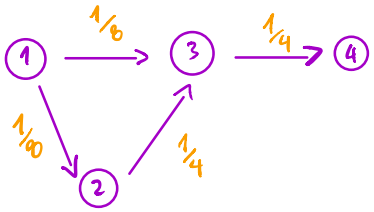
$d = 0.85$

$N = 4$

Iteración 0:



Iteración 1:

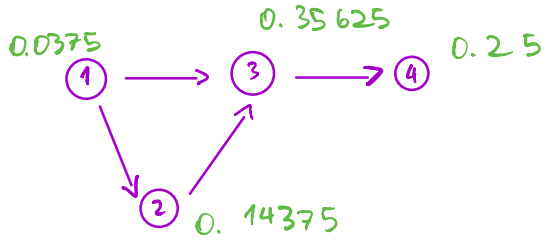


$$Pr_1(n_1) = \frac{1-0.85}{4} = 0.0375$$

$$Pr_1(n_2) = \frac{1-0.85}{4} + 0.85 \cdot \frac{1}{8} = 0.14375$$

$$Pr_1(n_3) = \frac{1-0.85}{4} + 0.85\left(\frac{1}{8} + \frac{1}{4}\right) = 0.35625$$

$$Pr_1(n_4) = \frac{1-0.85}{4} + 0.85 \cdot \frac{1}{4} = 0.25$$



Iteramos hasta que el P.R. de los nodos no cambie mucho en dos iteraciones consecutivas

Reduce

Recordemos la noción de reduce:

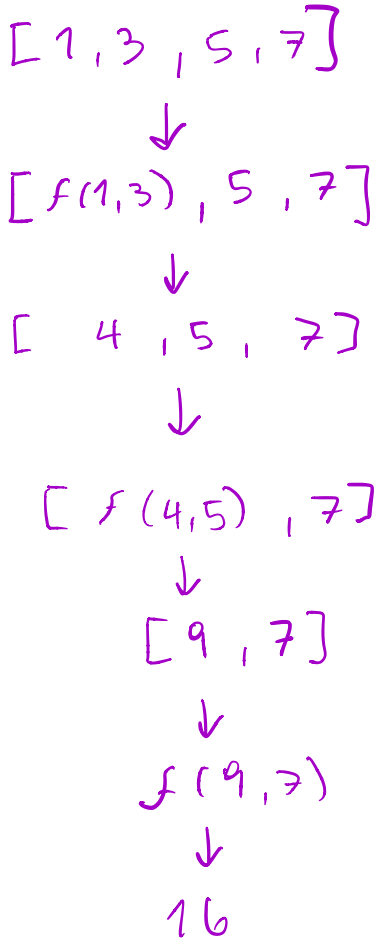
- Recibimos un iterable y una función que recibe dos argumentos
- Se va ejecutando la función de a pares

Ej. $l = [1, 3, 5, 7]$

$f(a,b) = a+b$

Entonces al hacer reduce:

reduce (l , f) =



PageRank con Pregel

La idea de computar PageRank con Pregel es la siguiente:

- Creamos un grafo en que cada nodo tiene su PageRank. Además cada arista guarda el valor $\frac{1}{outdegree}$ del nodo del que sale.
- Ejecutaremos un número fijo de iteraciones (10) con un damping factor de 0.85.
- En cada iteración, el mensaje que se emite es el P.R. del nodo, multiplicado por el valor guardado en la arista respectiva.
- A cada nodo le llegarán (potencialmente) varios de esos mensajes, el merge es un reduce del tipo a+b.
- Finalmente, actualizamos el P.R. del nodo.

1) Inicializar el grafo:

```scala
val initialGraph: Graph[Double, Double] = graph.
  outerJoinVertices(graph.outDegrees) {
    (vid, vdata, deg) => deg.getOrElse(0)
  }.
  mapTriplets(e => 1.0 / e.srcAttr).
  mapVertices((id, attr) => 1.0)
```

→ Valor de la Arista

→ Valor de los vértices

↓

Creamos un grafo en que los nodos son (id, P.R.) y las aristas tienen un Double que guarda el valor $\frac{1}{\text{outdegree}}$

2) Envier mensaje:

Tipo del vértice (sin id)

Tipo de la arista

```scala
def sendMsg(edge: EdgeTriplet[Double, Double]): Iterator[(VertexId, Double)] = {
  Iterator((edge.dstId, edge.srcAttr * edge.attr))
}
```

Para cada triplet, enviamos a los nodos que reciben la arista

Se envía $\frac{1}{\text{outdegree}} \cdot PR$ del nodo desde donde sale la arista

Id del nodo al que se envía el mensaje

Tipo del mensaje

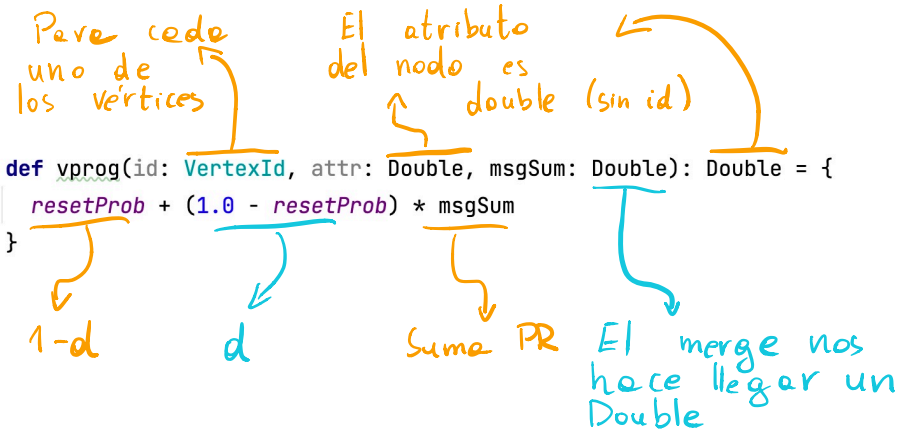
Todo se encapsula en un iterador

3) Merge de los mensajes:

```
def mergeMsg(a: Double, b: Double): Double = a + b
```

Aplicamos el reduce sobre la lista de mensajes que nos llega

4) Se reciben los mensajes:



5) Se llame la función Pregel y se imprime el grafo generado

```
val pagerankGraph = initialGraph.pregel(initialMsg, numIter, EdgeDirection.Out)(
  vprog, sendMsg, mergeMsg)

pagerankGraph.triplets.collect().foreach(t => println(s"${t}"))
```