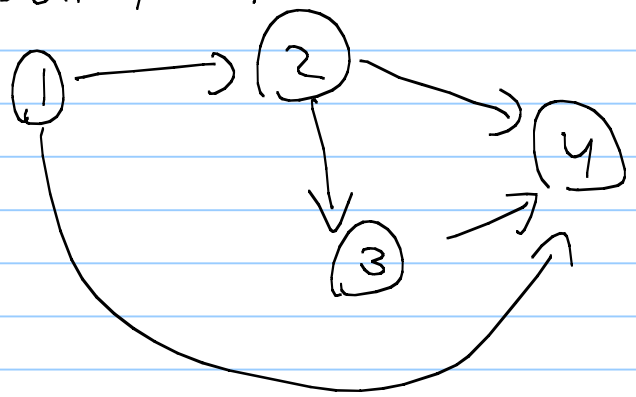
# Graph Algorithms

Note Title 9/25/2014

## Representation




Graph has $n$ nodes $m$ edges

Adjacency Matrix

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 0 |

directed

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 |
| 4 | 1 | 1 | 1 | 0 |

undirected

Storage alternatives

- Store matrix on disk (each row is line)
  $n^2 + n$

most graphs are sparse

- store coordinates of nonzero entries.
  $2m$

- Adjacency list

| node | node list of neighbors |
|---|---|
| 1 | 2 4 |
| 2 | 3 4 |
| 3 | 4 |
| 4 | |

$m+n$

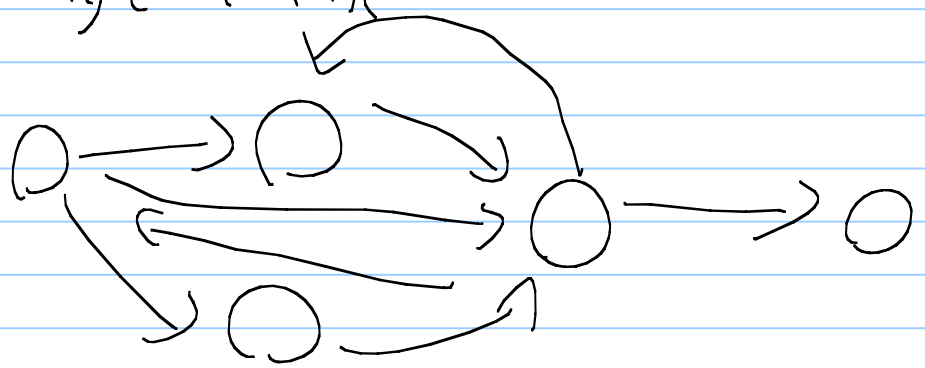
Shortest path Algorithm

Dijkstra's alg. Find shortest path between source and all other nodes in weighted graph, $w(u,v) \geq 0$

```
Dijkstra( G , w, s) = {
   d[s] = 0
   For every other node v:
      d[v] = infinity (or n+1)
   Priority Q ={s}   (priority based on shortest d value)
   while(Q is not empty) {
      v <- Q.getmin()
      for every out neighbor u of v:
         if d[u] > d[v] + w(v,u) then
            d[u] = d[v] + w(v,u)
            backpointer[u]=v (optional)
            Q.add(u)
```

```
Mapper
map(key: nodeid, value="(nid, weight) (nid, weight) .... d[nodeid]") {
    dist = d[nodeid]
    emit(key, value) //structure of graph passed to reducer
    for each node m in value:
        emit(m, dist + weight(nodeid, m))
Reducer
reduce(key: nodeid, valuelist = {adjlist, count1, count2, count3, ...}
     dmin = infinity
     for each value:
          if value is adjlist
               M <- value
          else //value is a number
               if val < dmin:
                    dmin <- value
      if M.distance < dmin:
           dmin = M.distance
      if dmin < M.distance: incremenet counter
      M.distance = dmin
      emit(nodeid , M)
```

# Page Rank



Random surfer

with prob $\alpha$ = goto random webpage

$(1-\alpha)$ = pick one of outgoing links from current page and follow it

importance = fraction of time surfer spends at web page

$$Pr(v) = \frac{\alpha}{n} + (1-\alpha) \sum_{\substack{\text{in links} \\ u}} \frac{Pr(u)}{\text{outdegree}(u)}$$

normalize so sums to 1

```
Mapper
  Map(key: nodeid, value = "n1 n2 n3 pr") {
      p <- pr / number of nodes nodeid points to
      emit(nodied, value)
      for every node m that nodeid points to:
          emit(m, p)


Reducer
   reduce(key: node id, value-list = {adj-list, p1 p2 p3 p4}
      inp = 0
      for each value in value-list:
          if value is adj-list
              M <- value
          else
              inp += value
       M.pr = inp
       emit(nodeid, M)
```

To add teleportation and reweighting so that pageranks sum up to 1, do the following

1. mapper multiplies p by 1-alpha
2. in reducer, inp is initialized with alpha/n where n is the number of total nodes in the graph.

3. Mapper maintains sum of the pageranks it emits. these sums get sent to all reducers.

4. Each reducer aggregates the special values from step 3 and adds alpha to them (this must be done first using the order inversion technique). This sum is the normalizing factor

5. Before emitting, the reducer divides inp by the normalizing factor.