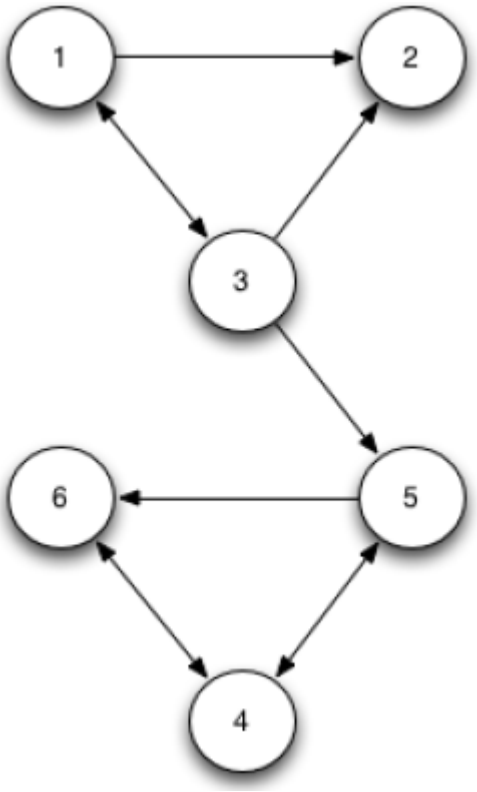
BLin_Assign10

Bin Lin

2017-4-12

1. Playing with PageRank You'll verify for yourself that PageRank works by performing calculations on a small universe of web pages. Let's use the 6 page universe that we had in the course notes. For this directed graph, perform the following calculations in R.



URL Page Universe

$$A = \begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Form the A matrix. Then, introduce decay and form the B matrix as we did in the course notes.

The following is the decay formula, which I am going to use in the calculation $B = d \times A + \frac{1-d}{n}$

```
A <- matrix(c(0, 1/2, 1/2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1/3, 1/3, 0, 0, 1/3, 0, 0, 0, 0, 0, 1/2, 1
/2, 0, 0, 0, 1/2, 0, 1/2, 0, 0, 0, 1, 0, 0), 6, 6, byrow = TRUE)
d <- 0.85
B <- d * A + (1 - d) / nrow(A)

# Matrix A has all 0s on the second row, which means the url 2 does not have any outlinks associ
ated with it. This is what we called the dangling node. This is also the only raw which does not
sum up to 1. Replace the row with uniform 1/6. (Their sum is 1)
B[2,] <- rep(1/6, 6)
B
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
## [1,] 0.0250000 0.4500000 0.4500000 0.0250000 0.0250000 0.0250000
## [2,] 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667 0.1666667
## [3,] 0.3083333 0.3083333 0.0250000 0.0250000 0.3083333 0.0250000
## [4,] 0.0250000 0.0250000 0.0250000 0.0250000 0.4500000 0.4500000
## [5,] 0.0250000 0.0250000 0.0250000 0.4500000 0.0250000 0.4500000
## [6,] 0.0250000 0.0250000 0.0250000 0.8750000 0.0250000 0.0250000
```

Start with a uniform rank vector r and perform power iterations on B till convergence. That is, compute the solution r = Bn * r. Attempt this for a sufficiently large n so that r actually converges.

```
convergence <- function(B, r)
{
  ri <- r
  rf <- t(B) %*% ri
  n <- 1

  while(all.equal(ri, rf) != TRUE)
  {
    ri <- rf
    rf <- t(B) %*% ri
    n <- n + 1
  }
  output <- list("Number of Iterations" = n, "r" = rf)
  return (output)
}
```

```
r <- matrix(c(1/6, 1/6, 1/6, 1/6, 1/6, 1/6), 6, 1, byrow = TRUE)
uniform_rank <- convergence(B, r)
uniform_rank$r
```

```
##            [,1]
## [1,] 0.05170475
## [2,] 0.07367927
## [3,] 0.05741241
## [4,] 0.34870368
## [5,] 0.19990381
## [6,] 0.26859608
```

Compute the eigen-decomposition of B and verify that you indeed get an eigenvalue of 1 as the largest eigenvalue and that its corresponding eigenvector is the same vector that you obtained in the previous power iteration method. Further, this eigenvector has all positive entries and it sums to 1.

```
#Just by eyeballing the eigen values, we can tell the largest eigen value is 1
Re(eigen(t(B))$values)
```

```
## [1]  1.00000000  0.57619235 -0.42500000 -0.42500000 -0.34991524 -0.08461044
```

```
Re(eigen(t(B))$vectors)
```

```
##           [,1]       [,2]          [,3]          [,4]        [,5]
## [1,] 0.1044385  0.2931457  2.519657e-15  2.519657e-15 -0.06471710
## [2,] 0.1488249  0.5093703 -8.507933e-16 -8.507933e-16  0.01388698
## [3,] 0.1159674  0.3414619 -1.930646e-15 -1.930646e-15  0.07298180
## [4,] 0.7043472 -0.5890805 -7.071068e-01 -7.071068e-01 -0.66058664
## [5,] 0.4037861 -0.1413606  7.071068e-01  7.071068e-01  0.73761812
## [6,] 0.5425377 -0.4135367  1.465801e-15  1.465801e-15 -0.09918316
##              [,6]
## [1,] -0.212296003
## [2,]  0.854071294
## [3,] -0.363638739
## [4,]  0.018399984
## [5,] -0.304719509
## [6,]  0.008182973
```

```
#After normalization of the vectors I got from previous power iteration method, we know it is id
entical to the eigen vector of B when its eigen value equals to 1.
normalized_rank <- uniform_rank$r[,1] / sqrt(sum(uniform_rank$r[,1] ^ 2))
normalized_rank
```

```
## [1] 0.1044385 0.1488249 0.1159674 0.7043472 0.4037861 0.5425377
```

```
#Both page rank vectors are identical.
all.equal(normalized_rank, Re(eigen(t(B))$vectors[,1]))
```

```
## [1] TRUE
```

```
#This code just prove the all values in the vectors I got from previous power iteration method a
re positive entries and it sums to 1.
sum(uniform_rank$r)
```

```
## [1] 1
```

Use the graph package in R and its page.rank method to compute the Page Rank of the graph as given in A. Note that you don't need to apply decay. The package starts with a connected graph and applies decay internally. Verify that you do get the same PageRank vector as the two approaches above.

```
library(igraph)
```

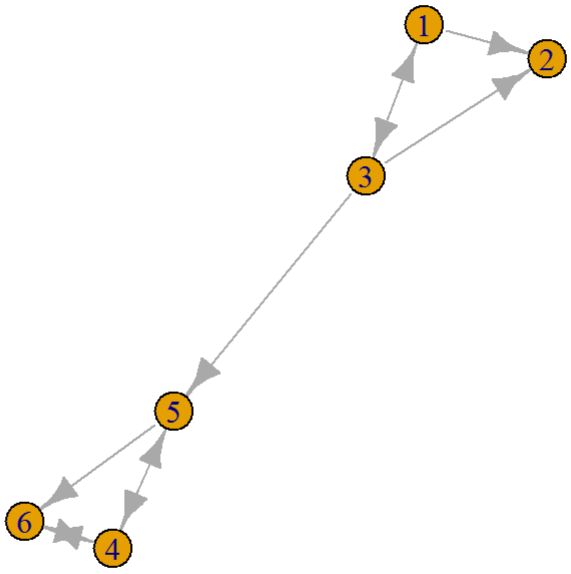
```
## Warning: package 'igraph' was built under R version 3.3.3
```

```
##
## Attaching package: 'igraph'
```

```
## The following objects are masked from 'package:stats':
##
##     decompose, spectrum
```

```
## The following object is masked from 'package:base':
##
##     union
```

```
pagerank_graph <- graph_from_adjacency_matrix(A, mode = c("directed"), weighted = TRUE)
plot(pagerank_graph)
```




```
pagerank_vector <- page_rank(pagerank_graph)$vector
pagerank_vector
```

```
## [1] 0.05170475 0.07367926 0.05741241 0.34870369 0.19990381 0.26859608
```

```
normalized_pagerank <- pagerank_vector / sqrt(sum(pagerank_vector ^ 2))
normalized_pagerank
```

```
## [1] 0.1044385 0.1488249 0.1159674 0.7043472 0.4037861 0.5425377
```

```
#It proves page.rank method generate same PageRank vector as other two approaches.
all.equal(normalized_rank, Re(eigen(t(B))$vectors[,1]), normalized_pagerank)
```

```
## [1] TRUE
```