Page rank / HITS computation

IRWS course project

Compute efficient Link Analysis algorithms

PageRank

- $(1\text{-}d)E/n + dA^{\mathrm{T}}$ is a stochastic matrix. It is also irreducible and aperiodic
- **Note that**
  - $E = e\, e^{\mathrm{T}}$ where e is a column vector of 1's
  - $e^{\mathrm{T}} P = 1$ since P is a probability vector

$$A_{ji} = \begin{cases} \dfrac{1}{O_j} & if\,(j,i) \in E \\ 0 & otherwise \end{cases}$$

$$P = ((1-d)\frac{E}{n} + dA^T)P = (1-d)\frac{1}{n} e\, e^T P + dA^T P =$$

$$= (1-d)\frac{1}{n} e + dA^T P$$

Once solved the issue of dangling nodes

Compute PageRank

- Use the **power iteration** method

PageRank-Iterate(G)

$P_0 \leftarrow e/n$

$k = 0$

repeat

$P_{k+1} \leftarrow (1-d)\frac{e}{n} + dA^T P_k$;

$k = k + 1$;

until $||P_{k+1} - P_k||_1 < \varepsilon$

return P_{k+1}

Initialization

Norm 1 less than 10^{-6}

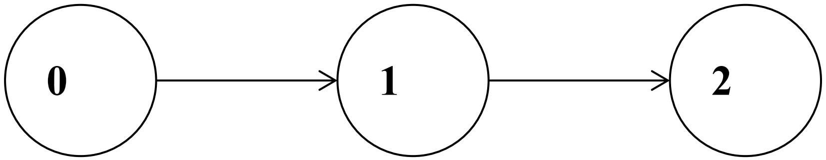
Fig. 6. The power iteration method for PageRank

Page Rank

- Write a sequential code (C or C++) that implements Pagerank
- Compile the code with `–O3` option, and measure the execution times (command `time`) for some inputs
- Input graphs: http://snap.stanford.edu/data
  - `e.g.:` Large Web graphs

- Naïve implementation of PageRank in Phython
  - https://colab.research.google.com/drive/1cyzyzKNXY4jA8wK9mvjH9UPcZS7ng9Mv

- Test example:



P[2] = 0.47441217
P[1] = 0.34117105
P[0] = 0.18441678

Data Format

```
# Directed graph (each unordered pair of nodes is saved once): web-NotreDame.txt
# University of Notre Dame web graph from 1999 by Albert, Jeong and Barabasi
# Nodes: 325729 Edges: 1497134
# FromNodeId    ToNodeId
0               0
0               1
0               2
0               3
0               4
0               5
…
```

- This is an excerpt from the compressed text file:
  `web-NotreDame.txt.gz`
  check if the numbering of the nodes starts always from 0 (to `325728` in this case)
- Note that if we represent the dense adjacency matrix (where each row i is divided by O_i) of the above graph (`325729 nodes`) using a float per entry (4 Bytes), we need about **395 GB**
- The pairs of nodes are generally unordered.
- Since we need to generate the TRANSPOSED matrix stored *row major*, we have to sort w.r.t. the 2nd **`ToNodeId`** field: from $(i,j,)$ to (j,i)
  - Look at the way we generate an inverted file!

Sparse matrix representation

Compressed sparse row (CSR or CRS)

Used for traversing matrix in
row major order

You have to implement the product:
sparse_matrix * dense_vector

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}.$$

val	10	-2	3	9	3	7	8	7	3	8	7	5	8	9	9	13	4	2	-1
col_ind	0	4	0	1	5	1	2	3	0	2	3	4	1	3	4	5	1	4	5

row_ptr	0	2	5	8	12	16	19

Start row 0

Start row 1

Start row 2

Start row 3

Start row 4

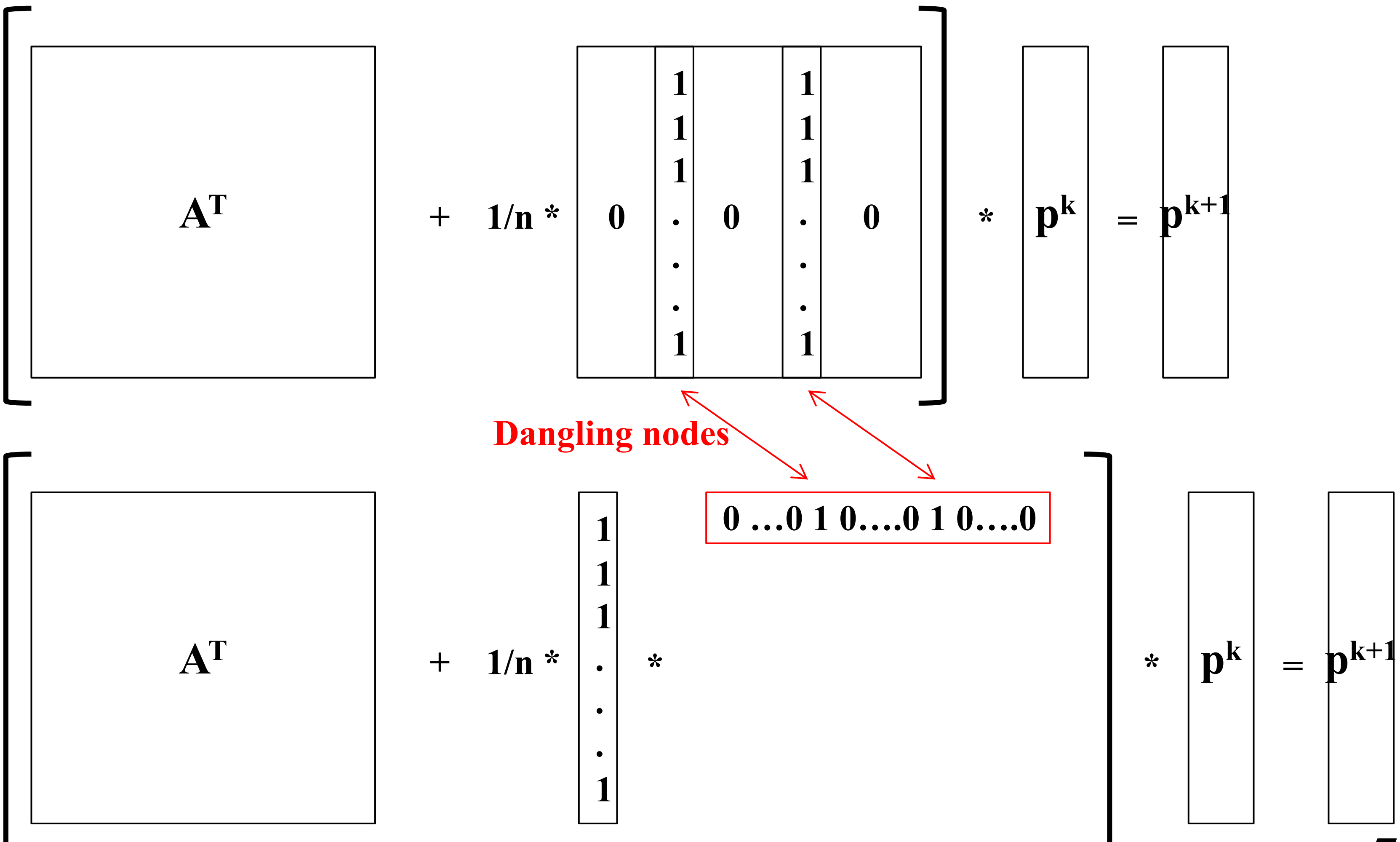
Start row 5

Start row 6 (1 more position)

Position where the n-th row should start. Note that the matrix is sparse: thus, the row could be completely zero. In this case **row_ptr[n] = row_prt[n+1]**

Optimizing Page Rank

- Given the original transposed adjacent matrix A^T, if we know the dangling node IDs, we can avoid filling zero-columns with values *1/n*



Optimizing Page Rank

$$A^T * p^k + 1/n * \begin{bmatrix} 1 \\ 1 \\ 1 \\ \cdot \\ \cdot \\ \cdot \\ 1 \end{bmatrix} * \overbrace{\left[\boxed{0 \ldots 0\ 1\ 0 \ldots. 0\ 1\ 0 \ldots. 0} * p^k \right]}^{\sum_{i \in Danglings} p^k[i]} = p^{k+1}$$

$$A^T * p^k + \begin{bmatrix} \sum_{i \in Danglings} \frac{p^k[i]}{n} \\ \cdot \\ \cdot \\ \cdot \\ \sum_{i \in Danglings} \frac{p^k[i]}{n} \end{bmatrix} = p^{k+1}$$

Optimizing Page Rank

$$\mathbf{A^T} * \mathbf{p^k} + \begin{bmatrix} \sum_{i \in Danglings} \frac{p^k[i]}{n} \\ \vdots \\ \sum_{i \in Danglings} \frac{p^k[i]}{n} \end{bmatrix} = \mathbf{p^{k+1}}$$

- Indeed, the above formula only compute: $p^{k+1} = A'^T p^k$ where A' is modified to become stochastic

- We must compute:

$$p^{k+1} = (1-d)/n\ \mathbf{e} + d\ A'^T p^k$$

mmap-ping large binary files

- Once produced the sparse array A^T[ ][ ] (CSR), store it to a *file* for future purposes
  – Instead of reading and storing A^T[ ][ ] by dynamically allocating memory (`malloc`) you can map the file to a memory region, and access it via pointers (just as you would access ordinary variables and objects)
  – You can mmap "specific section/partition of the file", and share the files between more threads

```
#include <stdio.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
    int i;
    float val;
    float *mmap_region;

    FILE *fstream;
    int fd;
```

mmap-ping large binary files

```
/* create the file */
fstream = fopen("./mmapped_file", "w+");
for (i=0; i<10; i++) {
    val = i + 100.0;

    /* write a stream of binary floats */
    fwrite(&val, sizeof(float), 1, fstream);
}
fclose(fstream);


/* map a file to the pages starting at a given address
   for a given length */
fd = open("./mmapped_file", O_RDONLY);
mmap_region = (float *)     mmap(0, 10*sizeof(float), PROT_READ,
                                 MAP_SHARED, fd, 0);
if (mmap_region == MAP_FAILED) {
    close(fd);
    printf("Error mmapping the file");
    exit(1);
}
close(fd);
```

Starting offset address in the file

mmap-ping large binary files

```
    /* Print the data mmapped */
    for (i=0; i<10; i++)
       printf("%f ", mmap_region[i]);
    printf("\n");



    /* free the mmapped memory */
    if (munmap(mmap_region, 10*sizeof(float)) == -1) {
       printf("Error un-mmapping the file");
       exit(1);
    }
}
```

HITS

- HITS also adopts the *power method* to produces the principal eingvectors ***a*[]** (authority scores) and ***h*[]** (hub scores) of the two matrixes $L^T L$ and $L L^T$ respectively

$$a_k = L^T L \, a_{k-1}$$

$$h_k = L \, L^T h_{k-1}$$

- L is the normal adjacency matrix of a graph (without dividing by the out-degree of the node as for PageRank)

- Naïve implementation of HITS in Phyton
  - **https://colab.research.google.com/drive/1PVg-uE-UOMpWgONf5WscN06HkqTiybUR**
  - we normalizes by dividing the authority and hub vectors by the sum of ***a*[]** and ***h*[ ]** (many normalization are proposed, as without HITS may not converge)

HITS

- Write a sequential code (C or C++) that implements HITS
- Compile the code with `–O3` option, and measure the execution times (command `time`) for some inputs
- Same input graphs used for PageRank:

  http://snap.stanford.edu/data

  – `e.g.:` Web graphs

- For the sparse adjacency matrix L and its transpose L^T used by HITS, consider all hints about matrix compression, etc.

- For each graph, you can rank nodes by (1) HITS authority, (2) PageRank and (3) InDegree
  - Compute the *top-k nodes*, ***k*=10,20,30** …, considering the 3 rankings above
  - Plot the 3 Jaccard coefficients (or Kendal's τ coefficient) between each pair of top-k set of nodes

Delivery of the project task

- Create a tar/zip file with:
  - your solution source code and the Makefile;
  - a readme file
  - a brief report (PDF)

- Groups of max 2 people

- Submit via moodle and send an email to: orlando@unive.it