1 Programming Project (PrP) overview

Your submission will support

- algorithmic-related programming,
- command-line processing support, and
- file-based input and/or output.

If you are not familiar with those topics and requirements we urge you to become so as soon as possible. Thus start familiarizing with them early in the semester. The more you procrastinate the more problems you will potentially face when you integrate these components with the algorithmic-related material. The function names shown in the following pages are just for guidance. They help us organize the narrative and assist you in organizing your project. They are programming language agnostic and thus might feel irrelevant or foreign to you all. Feel free to ignore them but do not complain afterwards.

STEP-1. Read carefully Document 3. It saves you time later.

STEP-2. When the archive per Document 3 guidelines is ready, upload it and submit it to canvas through Canvas Assignments

BEFORE NOON-time as in CALENDAR (Document 1, Syllabus)

We provide descriptions that are to the extent possible language independent: thus a reference in Java is a pointer in C or C++ for example. Also the (function, class) names we use do not adhere to the naming conventions of Document 3 as the names we use are general and generic.

HITS and PageRank implementations. Implement Kleinberg's HITS Algorithm, and Google's PageRank algorithm in Java, C, or C++ as explained. The implementation should be optimized enough for a test run execution to take no more than few seconds, maximum 5 seconds for graphs with up to 100 vertices.

2 HITS and PageRank (130 points)

(A) Implement the HITS algorithm as explained in class/Subject notes adhering to the guidelines of Document 3. Pay attention to the sequence of update operations and the scaling step. Subject notes contains an example that can be used for reference; various initialization vectors are being utilized. You need to implement the equivalent of class or function or program `hits` (e.g. `hits1234` or `hits_5678` if the last four digits of your NJIT ID are 1234 or 5678).

```
% java hits iterations initialvalue filename.txt
% ./hits    iterations initialvalue somegraph.txt
```

For an explanation of the arguments see the discussion on PageRank to follow.

(B) Implement Google's PageRank algorithm as explained in class/Subject notes adhering also to the guidelines of Document 3. Parameter d would be fixed to 0.85. You need to implement the equivalent of class or function or program `pgrk` (e.g. `pgrk_1234` or `pgrk5678` if the last four digits of your NJIT ID are 1234 or 5678).

```
% ./pgrk     iterations initialvalue somegraph.txt
% java pgrk  iterations initialvalue somegraph.txt
```

The two algorithms are iterative (see subject notes). You implement the synchronous update version for both. In particular, at iteration t all new rank values are computed using results from iteration $t-1$ synchronously. In order to run the 'algorithm' we either run it for a fixed number of iterations and the argument `iterations` gives the number, or for a fixed `errorrate`, and a zero or negative value of the argument `iterations` is interpreted as a given errorrate. We stop at iteration t if t is equal to the positive value of argument `iterations`; if the argument `iterations` indicates an errorrate, we compute the PageRank values, or all of the authority and hub values at the completion of the three steps of iteration t, and if the absolute value of the difference between the values at iteration t and $t-1$ is less than or equal to the **errorrate** for $\forall u$, then and only then can we stop at iteration t. (Errorrate is satisfied then.)

The command-line interface is as follows. The first argument (argv[0] in some programming languages) is the executable file (e.g. pgrk, hits), or a virtual machine followed by a class name (e.g. java pgrk). Next we have an arguments that indicates either the number of iterations or the errorrate (see Subject 3) associated with the computation. If it is a positive integer value it refers to the number of `iterations` we run the corresponding algorithm (HITS or PageRank). If it is a zero it indicates a default `errorrate` of 10^{-5}. It it is a negative integer value `-1, -2, etc , -6` it indicates a corresponding errorrate of $10^{-1}, 10^{-2}, \ldots, 10^{-6}$ respectively. Argument `initialvalue` sets the initial vector values. If it is `0` they are initialized to 0, if it is `1` they are initialized to 1, if it is `-1` they are initialized to $1/N$, where N is the number of web-pages (vertices of the graph), and If it is `-2` they are initialized to $1/\sqrt{N}$.

The last argument is a string of characters with or without the suffix (.txt) indicating an arbitrary filename such as `filename.txt` or `somegraph.txt` that were used in the previous example invocations. The file describes a directed graph represented through an adjacency list representation having the following format. The first line contains two numbers: **the number of vertices first, followed by the number of edges next**. The latter is also the number of the remaining lines. PAY ATTENTION TO THE ORDER of APPEARANCE. The sample graph is treacherous: n and m have the same value 4! All vertices are labeled $0, \ldots, N-1$. Expect N to be at most $100,000$ and thus it can fit into a 32-bit integer. In each line an edge (i, j) is represented by `i j`. Thus our graph has (directed) edges $(0,2),(0,3),(1,0),(2,1)$. Vector values are printed to 7 decimal digits. If the graph has N GREATER than 10, then the values for `iterations,`

`initialvalue` are automatically set to `0` and `-1` respectively. In such a case the hub/authority/pageranks at the stopping iteration (i.e t) are ONLY shown, one per line. The graph below will be referred to as `samplegraph.txt`

```
4 4
0 2
0 3
1 0
2 1
```

The following two invocations are for `samplegraph.txt`, with a fixed number of iterations and the fixed error rate that determines how many iterations will run respectively. Your code should compute for this graph the same rank values (intermediate and final). A sample of the output for the case of $N > 10$ is shown (with output truncated to first 4 lines of it).

```
% ./pgrk  15 -1 samplegraph.txt
Base   :  0 :P[ 0]=0.2500000 P[ 1]=0.2500000 P[ 2]=0.2500000 P[ 3]=0.2500000
Iter   :  1 :P[ 0]=0.2500000 P[ 1]=0.2500000 P[ 2]=0.1437500 P[ 3]=0.1437500
Iter   :  2 :P[ 0]=0.2500000 P[ 1]=0.1596875 P[ 2]=0.1437500 P[ 3]=0.1437500
Iter   :  3 :P[ 0]=0.1732344 P[ 1]=0.1596875 P[ 2]=0.1437500 P[ 3]=0.1437500
Iter   :  4 :P[ 0]=0.1732344 P[ 1]=0.1596875 P[ 2]=0.1111246 P[ 3]=0.1111246
Iter   :  5 :P[ 0]=0.1732344 P[ 1]=0.1319559 P[ 2]=0.1111246 P[ 3]=0.1111246
Iter   :  6 :P[ 0]=0.1496625 P[ 1]=0.1319559 P[ 2]=0.1111246 P[ 3]=0.1111246
Iter   :  7 :P[ 0]=0.1496625 P[ 1]=0.1319559 P[ 2]=0.1011066 P[ 3]=0.1011066
Iter   :  8 :P[ 0]=0.1496625 P[ 1]=0.1234406 P[ 2]=0.1011066 P[ 3]=0.1011066
Iter   :  9 :P[ 0]=0.1424245 P[ 1]=0.1234406 P[ 2]=0.1011066 P[ 3]=0.1011066
Iter   : 10 :P[ 0]=0.1424245 P[ 1]=0.1234406 P[ 2]=0.0980304 P[ 3]=0.0980304
Iter   : 11 :P[ 0]=0.1424245 P[ 1]=0.1208259 P[ 2]=0.0980304 P[ 3]=0.0980304
Iter   : 12 :P[ 0]=0.1402020 P[ 1]=0.1208259 P[ 2]=0.0980304 P[ 3]=0.0980304
Iter   : 13 :P[ 0]=0.1402020 P[ 1]=0.1208259 P[ 2]=0.0970858 P[ 3]=0.0970858
Iter   : 14 :P[ 0]=0.1402020 P[ 1]=0.1200230 P[ 2]=0.0970858 P[ 3]=0.0970858
Iter   : 15 :P[ 0]=0.1395195 P[ 1]=0.1200230 P[ 2]=0.0970858 P[ 3]=0.0970858

% ./pgrk -3 -1 samplegraph.txt
Base   :  0 :P[ 0]=0.2500000 P[ 1]=0.2500000 P[ 2]=0.2500000 P[ 3]=0.2500000
Iter   :  1 :P[ 0]=0.2500000 P[ 1]=0.2500000 P[ 2]=0.1437500 P[ 3]=0.1437500
Iter   :  2 :P[ 0]=0.2500000 P[ 1]=0.1596875 P[ 2]=0.1437500 P[ 3]=0.1437500
Iter   :  3 :P[ 0]=0.1732344 P[ 1]=0.1596875 P[ 2]=0.1437500 P[ 3]=0.1437500
Iter   :  4 :P[ 0]=0.1732344 P[ 1]=0.1596875 P[ 2]=0.1111246 P[ 3]=0.1111246
Iter   :  5 :P[ 0]=0.1732344 P[ 1]=0.1319559 P[ 2]=0.1111246 P[ 3]=0.1111246
Iter   :  6 :P[ 0]=0.1496625 P[ 1]=0.1319559 P[ 2]=0.1111246 P[ 3]=0.1111246
Iter   :  7 :P[ 0]=0.1496625 P[ 1]=0.1319559 P[ 2]=0.1011066 P[ 3]=0.1011066
Iter   :  8 :P[ 0]=0.1496625 P[ 1]=0.1234406 P[ 2]=0.1011066 P[ 3]=0.1011066
Iter   :  9 :P[ 0]=0.1424245 P[ 1]=0.1234406 P[ 2]=0.1011066 P[ 3]=0.1011066
Iter   : 10 :P[ 0]=0.1424245 P[ 1]=0.1234406 P[ 2]=0.0980304 P[ 3]=0.0980304
Iter   : 11 :P[ 0]=0.1424245 P[ 1]=0.1208259 P[ 2]=0.0980304 P[ 3]=0.0980304
Iter   : 12 :P[ 0]=0.1402020 P[ 1]=0.1208259 P[ 2]=0.0980304 P[ 3]=0.0980304
Iter   : 13 :P[ 0]=0.1402020 P[ 1]=0.1208259 P[ 2]=0.0970858 P[ 3]=0.0970858

% ./pgrk  0  -1 verylargegraph.txt
Iter   : 4
P[ 0]=0.0136364
P[ 1]=0.0194318
P[ 2]=0.0310227
   ... other vertices omitted
```

For the HITS algorithm, you need to print two values not one. Follow the convention of the Subject notes

```
Base   :  0 :A/H[ 0]=0.3333333/0.3333333 A/H[ 1]=0.3333333/0.3333333 A/H[ 2]=0.3333333/0.3333333
Iter   :  1 :A/H[ 0]=0.0000000/0.8320503 A/H[ 1]=0.4472136/0.5547002 A/H[ 2]=0.8944272/0.0000000
```

or for large graphs

```
Iter   : 37
A/H[ 0]=0.0000000/0.0000002
A/H[ 1]=0.0000001/0.0000238
A/H[ 2]=0.0000002/1.0000000
A/H[ 3]=0.0000159/0.0000000
 ...
```

Deliverables. An archive per Document 3 guidelines.