

Large Programming Project

CS470 Fall 2005

due date December 1, 2005, 11:59pm

[worth 150 points]

The web is so big, that you cannot possibly hope to compute PageRank by keeping the web graph entirely in memory, as we did in Project 6.1. Thus you may need to use disk to store the graph, and possibly partial results of PageRank computation.

In this project your goal will be to implement the PageRank computation of a possibly large graph. Assume that the graph is provided in a file in the format as defined in Project 6.1. The file may be very big, but its size is at most 10GB. The disk has at least 70GB of free space. Each graph node will have at most about 500 edges leading to other nodes, but on average that number will be at most a few dozen. You need to write an algorithm that computes PageRank using a limited amount of memory. The memory bound is provided as a parameter to your algorithm. It is possible that the number of nodes in the graph is too big to fit the rank vector entirely in memory, not to mention the graph itself. You may need to use both disk and memory in your algorithm. Notice that reading and writing to memory is several orders of magnitude faster than reading and writing to disk. Thus try to use as much memory as allowed by the parameter. It is faster to read n bytes of a very big file sequentially, rather than read n bytes of the file at randomly selected locations (the disk head will move more in the latter case). Design the sequence of disk reads and writes accordingly.

Implementation

Implement the following function

```
float computePageRankExternal( char *fileName, float c, int numIterations,
                              long maxMemory, char *outputFile)
```

that computes PageRanks of every node of a graph. The graph is encoded in file `fileName` as defined in Project 6.1. The computation can use at most `maxMemory` bytes of memory. It must take exactly `numIterations` iterations (i.e., the epsilon stop condition is not tested). The resulting ranks must be saved to a file `outputFile`. The output file contains floating point numbers representing ranks of successive nodes of the graph, each number is a `float` (4 bytes). Example of saving ranks:

```
float r[100];
FILE *f = fopen("ranks.bin", "wb");
if( NULL==f )
    throw;
if( 100!=fwrite(r, sizeof(float), 100, f) )
    throw;
fclose(f);
```

The function must return the similarity between `rank[]` and `rankDubPrime[]` of the latest iteration (the most recent rank vectors, see the example solution to Project 6.1). Your code should be implemented in VC++ on a Windows machine.

Testing

Test your code on example graphs provided here

http://www.cs.ua.edu/470/fall2005/graph_0xff.zip,
http://www.cs.ua.edu/470/fall2005/graph_0xfff.zip, and
http://www.cs.ua.edu/470/fall2005/graph_0xffff.zip.

Report the running time of your implementation invoked on each file with `maxMemory` set to 100,000 (about 100KB) and `numIterations` set to 10. You can explore how your algorithm scales. For this purpose generate larger graphs using the following code

```

unsigned __int32 numNodes,i,j,noEdges,dst;
numNodes=0xffff; // 0xffffffff yields a file of about 1.1GB
FILE *f = fopen("graph.bin","wb");
if( NULL==f ) throw;
if( 1!=fwrite(&numNodes,sizeof(unsigned __int32),1,f) ) throw;
for( i=0; i<numNodes; i++ ) {
    // pick number of edges
    noEdges = (unsigned __int32)pow(rand(),0.3);
    for( j=0; j<noEdges; j++ ) {
        // pick a destination node
        dst = (((rand()<<15) ^ rand()) << 15) ^rand() % numNodes;
        if( 1!=fwrite(&dst,sizeof(unsigned __int32),1,f) ) throw;
    };
    // marker ending adjacency list
    dst = 0xffffffff;
    if( 1!=fwrite(&dst,sizeof(unsigned __int32),1,f) ) throw;
};
fclose(f);

```

Documentation

Provide a pseudocode of your algorithm along with an English language outline of the algorithm. In particular discuss how you use memory and disk (e.g., data structures, file structures). Provide comments inside your C++ code.

Submission

Submit the code, results of tests, and the documentation.

Competition

There will be a competition among students for the fastest algorithm. Files with various sizes will be used to rank the speed of the algorithms. The winner of the competition will receive extra credit, and so will these students in the 2nd and 3rd places.