Hanyang Univ.

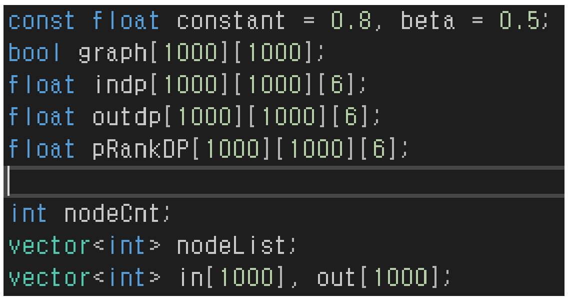
PRank

Data Science Term project

Sung Ki Hun

2019-6-10

1. Data Structures


```
const float constant = 0.8, beta = 0.5;
bool graph[1000][1000];
float indp[1000][1000][6];
float outdp[1000][1000][6];
float pRankDP[1000][1000][6];

int nodeCnt;
vector<int> nodeList;
vector<int> in[1000], out[1000];
```

Vector<int> **nodeList** : contains all nodes actually used in graph

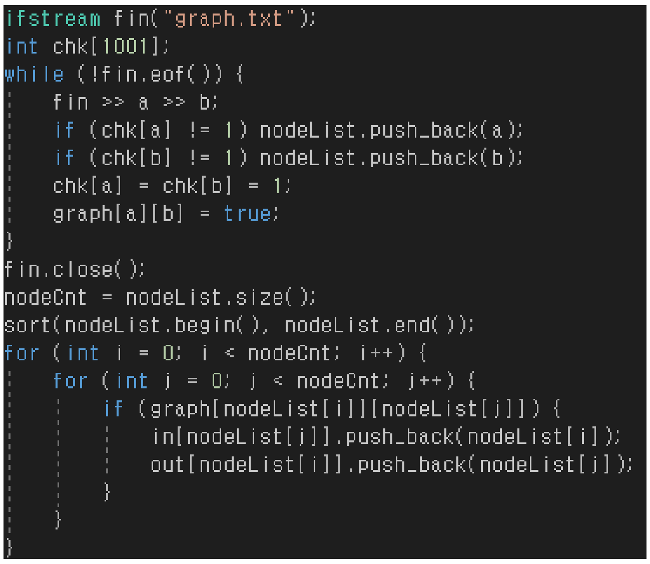
bool[][] **graph** : raw input data, will be processed as in, out vector.

Vector<int>[] **in**, **out** : get I, O of each node. index is corresponding by nodeList.

float[][][] **indp**, **outdp**, **pRankDP** : memoization array, third axis means step.

2. Source Explanation

Vector used for optimized memory usage

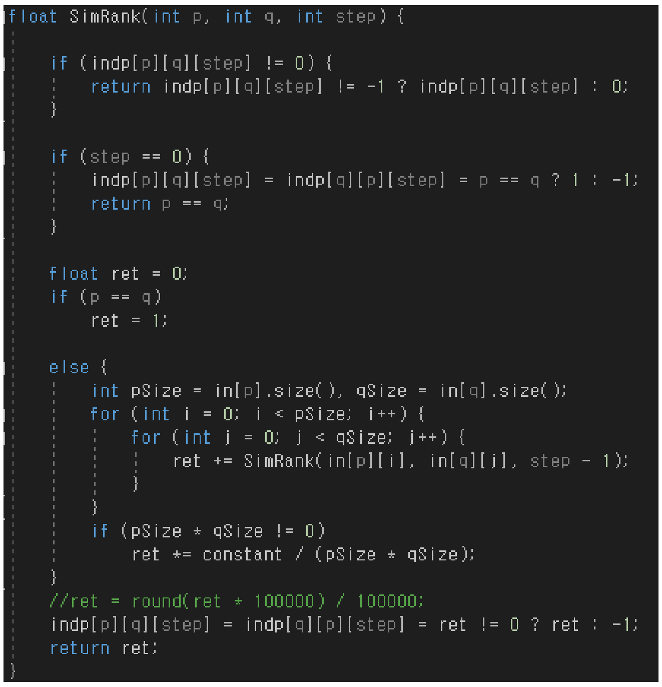

```
ifstream fin("graph.txt");
int chk[1001];
while (!fin.eof()) {
    fin >> a >> b;
    if (chk[a] != 1) nodeList.push_back(a);
    if (chk[b] != 1) nodeList.push_back(b);
    chk[a] = chk[b] = 1;
    graph[a][b] = true;
}
fin.close();
nodeCnt = nodeList.size();
sort(nodeList.begin(), nodeList.end());
for (int i = 0; i < nodeCnt; i++) {
    for (int j = 0; j < nodeCnt; j++) {
        if (graph[nodeList[i]][nodeList[j]]) {
            in[nodeList[j]].push_back(nodeList[i]);
            out[nodeList[i]].push_back(nodeList[j]);
        }
    }
}
```

nodeList -> added node which is in actual use.

To use vector, don't need to iterate whole.

Create two different vector which means In and Out.

Used memorization to speed up PRank Algorithm



```
float SimRank(int p, int q, int step) {

    if (indp[p][q][step] != 0) {
        return indp[p][q][step] != -1 ? indp[p][q][step] : 0;
    }

    if (step == 0) {
        indp[p][q][step] = indp[q][p][step] = p == q ? 1 : -1;
        return p == q;
    }

    float ret = 0;
    if (p == q)
        ret = 1;

    else {
        int pSize = in[p].size(), qSize = in[q].size();
        for (int i = 0; i < pSize; i++) {
            for (int j = 0; j < qSize; j++) {
                ret += SimRank(in[p][i], in[q][j], step - 1);
            }
        }
        if (pSize * qSize != 0)
            ret *= constant / (pSize * qSize);
    }
    //ret = round(ret * 100000) / 100000;
    indp[p][q][step] = indp[q][p][step] = ret != 0 ? ret : -1;
    return ret;
}
```

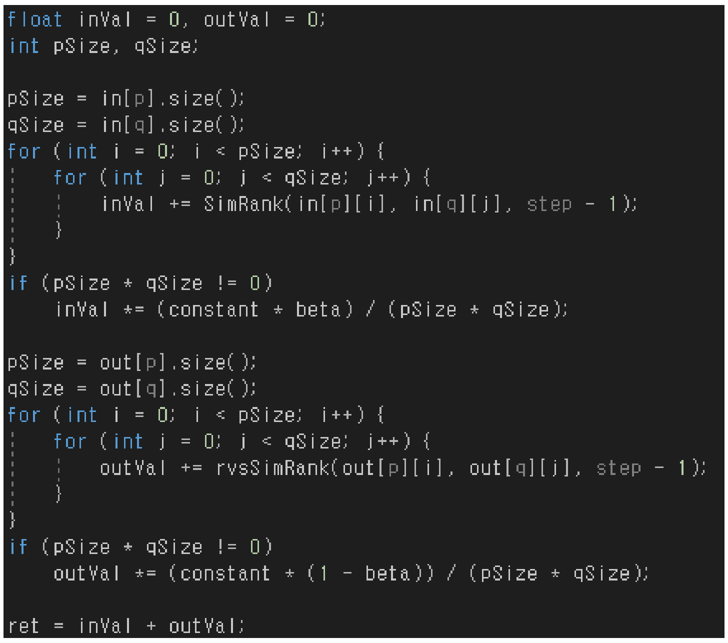
On S0, check whether p and q are same or not as p==q.

(pSize * qSize != 0):

Check whether denominator is 0 for prevent to make value inf

-1 means 0.

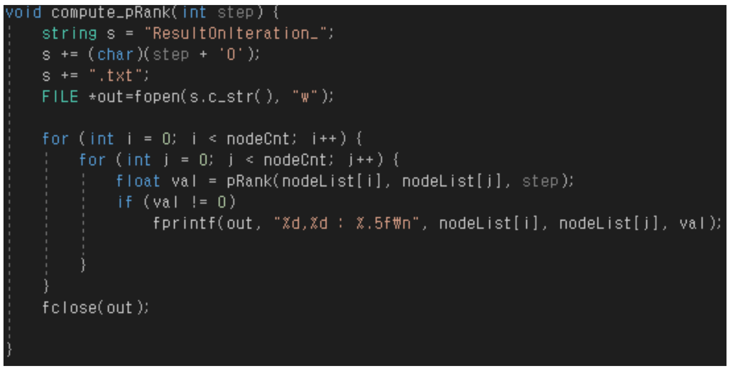
PRank Implementation



```
float inVal = 0, outVal = 0;
int pSize, qSize;

pSize = in[p].size();
qSize = in[q].size();
for (int i = 0; i < pSize; i++) {
    for (int j = 0; j < qSize; j++) {
        inVal += SimRank(in[p][i], in[q][j], step - 1);
    }
}
if (pSize * qSize != 0)
    inVal *= (constant * beta) / (pSize * qSize);

pSize = out[p].size();
qSize = out[q].size();
for (int i = 0; i < pSize; i++) {
    for (int j = 0; j < qSize; j++) {
        outVal += rvsSimRank(out[p][i], out[q][j], step - 1);
    }
}
if (pSize * qSize != 0)
    outVal *= (constant * (1 - beta)) / (pSize * qSize);

ret = inVal + outVal;
```

Compute PRank is almost same as SimRank and rvsSimRank. Just added both of them.

Print each step of pRank implementation



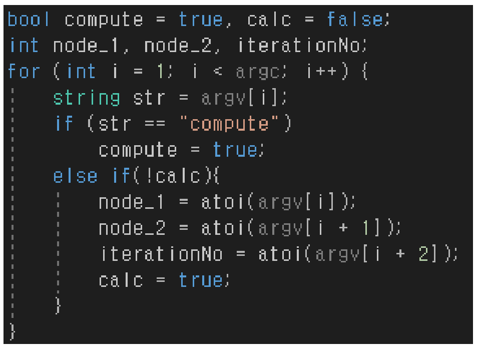
```
void compute_pRank(int step) {
    string s = "ResultOnIteration_";
    s += (char)(step + '0');
    s += ".txt";
    FILE *out=fopen(s.c_str(), "w");

    for (int i = 0; i < nodeCnt; i++) {
        for (int j = 0; j < nodeCnt; j++) {
            float val = pRank(nodeList[i], nodeList[j], step);
            if (val != 0)
                fprintf(out, "%d,%d : %.5f₩n", nodeList[i], nodeList[j], val);

        }
    }
    fclose(out);

}
```

Compute all pRank between nodes.

Variable ‘val’ check seems useless but I didn’t test without it.

Implementation slightly changed after write document as “**fopen**” to “**fopen_s**”

Parameter handling Implementation

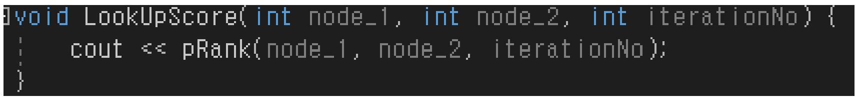


```
bool compute = true, calc = false;
int node_1, node_2, iterationNo;
for (int i = 1; i < argc; i++) {
    string str = argv[i];
    if (str == "compute")
        compute = true;
    else if(!calc){
        node_1 = atoi(argv[i]);
        node_2 = atoi(argv[i + 1]);
        iterationNo = atoi(argv[i + 2]);
        calc = true;
    }
}
```

Check whether parameter is “compute” or “node_1 node_2 iterationNo”.

“node_1 node_2 iterationNo compute” will be work fine too.

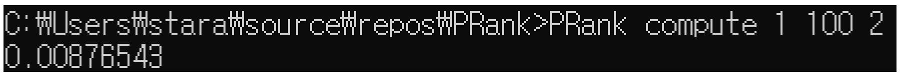
LookupScore



```
void LookUpScore(int node_1, int node_2, int iterationNo) {
    cout << pRank(node_1, node_2, iterationNo);
}
```

Just same as pRank. Added for criteria on document.

3. Program Usage



```
C:₩Users₩stara₩source₩repos₩PRank>PRank compute 1 100 2
0.00876543
```

PRank parameters: Can use both of them or single of them too.

- Compute
- Node_1 Node_2 iterationNo