```
import numpy as np
import time
import argparse
import sys

"""
Below is code for the PageRank algorithm (power iteration).

This code assumes that the node IDs start from 0 and are contiguous up to
max_node_id.
You are required to implement the functionality in the space provided.

Because computing the adjacency matrix for large graph requires to load large
graph dataset to
computer memory, thus, in order to calculate the PageRank value of each node,
you need to iterate
over dataset multiple times and update the PageRank value based on equation
mentioned in the question.
"""

def author():

        return "***" # replace gburdell3 with your Georgia Tech username.


def gtid():

    return 903638876 # replace with your GT ID number

class PageRank:
    def __init__(self, edge_file):

        self.node_degree = {}
        self.max_node_id = 0
        self.edge_file = edge_file

    def read_edge_file(self, edge_file):
        with open(edge_file) as f:
            for line in f:
                val = line.split('\t')
                yield int(val[0]), int(val[1])

    """
    Step1: Calculate the out-degree of each node and maximum node_id of the
graph.
    Store the out-degree in class variable "node_degree" and maximum node id to
"max_node_id".
    """
    def calculate_node_degree(self):
        for source,target in self.read_edge_file(self.edge_file):

        ### Implement your code here
        #############################################
            self.node_degree[source] = self.node_degree.get(source,0) + 1
            if source > self.max_node_id:
                self.max_node_id = source
            if target > self.max_node_id:
                self.max_node_id = target


        #############################################

        print("Max node id: {}".format(self.max_node_id))
    def get_max_node_id(self):
        return self.max_node_id

    def run_pagerank(self, node_weights,  damping_factor=0.85, iterations=10):

        pr_values = [1.0 / (self.max_node_id + 1)] * (self.max_node_id + 1)
        start_time = time.time()
        """
        Step2: Implement pagerank algorithm as mentioned in lecture slides and 
the question.

        Incoming Parameters:
            node_weights: Probability of each node to flyout during random walk
            damping_factor: Probability of continuing on the random walk
            iterations: Number of iterations to run the algorithm 
            check the __main__ function to understand node_weights and 
max_node_id

        Use the calculated out-degree to calculate the pagerank value of each 
node
        """
        for it in range(iterations):

            new_pr_values = [0.0] * (self.max_node_id + 1)
            for source, target in self.read_edge_file(self.edge_file):

        ### Implement your code here
        #############################################
                if new_pr_values[target] == 0:
                    new_pr_values[target] = (1 - damping_factor) * 
node_weights[target]
                new_pr_values[target] += damping_factor * pr_values[source] / 
self.node_degree[source]

            for i in range(self.max_node_id):
                if new_pr_values[i] == 0:
                    new_pr_values[i] = (1-damping_factor)*node_weights[i]
            pr_values = new_pr_values

        #############################################

        print ("Completed {0}/{1} iterations. {2} seconds elapsed.".format(it + 
1, iterations, time.time() - start_time))

        return pr_values

def dump_results(command, iterations, result):
    print("Sorting...", file=sys.stderr)
    sorted_result = sorted(enumerate(result), key=lambda x: x[1], reverse=True)
    output_result = "node_id\tpr_value\n"
    for node_id, pr_value in sorted_result[:10]:
        output_result += "{0}\t{1}\n".format(node_id, pr_value)
    print(output_result)

    with open(command+'_iter'+str(args.iterations)+".txt","w") as output_file:
        output_file.write(output_result)


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="sample command: python 
pagerank.py -i 5 -d 0.85 simplified_pagerank network.tsv")
    parser.add_argument("command", help="Sub-command to execute. Can be 
simplified_pagerank or personalized_pagerank.")
    parser.add_argument("filepath", help="path of the input graph 
file(network.tsv)")
    parser.add_argument("-i", "--iterations", dest="iterations",
                        help="specify the number of iterations to  the
algorithm. Default: 10",
                        default=10, type=int)
    parser.add_argument("-d", "--damping-factor", dest="damping_factor",
                        help="specify the damping factor for pagerank. Default:
0.85",
                        default=0.85, type=float)

    args = parser.parse_args()

    if args.command == "simplified_pagerank":
        pr = PageRank(args.filepath)
        pr.calculate_node_degree()
        max_node_id = pr.get_max_node_id()
        node_weights = np.ones(max_node_id + 1) / (max_node_id + 1)
        result = pr.run_pagerank(node_weights=node_weights,
iterations=args.iterations, damping_factor=args.damping_factor)
        dump_results(args.command, args.iterations, result )

    elif args.command == "personalized_pagerank":
        pr = PageRank(args.filepath)
        pr.calculate_node_degree()
        max_node_id = pr.get_max_node_id()

        np.random.seed(gtid())
        node_weights = np.random.rand(max_node_id + 1)
        node_weights = node_weights/node_weights.sum()
        result = pr.run_pagerank(node_weights=node_weights,
iterations=args.iterations, damping_factor=args.damping_factor)
        dump_results(args.command, args.iterations, result)

    else:
        sys.exit("Incorrect command")
```