大数据课程第一次项目作业

计算机科学与技术一班 洪一帆 *1811363*
计算机科学与技术一班 丁一凡 *1811348*
计算机卓越班 林铸天 *1811554*

2021 年 4 月 27 日

目录

第 1 章　问题重述和本组实现

本次作业要求共 4 个，本组在完成基本作业要求和学有余力的基础上，完成了 2 个拓展功能，并且在此过程中运用了压缩技术和`cython`减少空间、时间开销。按照作业

序号	作业要求	是否要求	本组完成情况
1	基础 PageRank 代码	√	√
2	解决 Dead Ends 和 Spider Trap 问题	√	√
3	优化稀疏矩阵	√	√
4	实现分块矩阵	√	√
5	使用压缩技术减少存储过程硬盘开销		√
6	使用 cython 提高效率		√

表 1.1: 作业要求及本组完成情况

要求，本次结果我们汇报：

1. 源代码、报告
2. Top 100 的结点结果（依照实验要求）
3. 可执行文件

第 2 章　项目核心框架介绍

本次项目核心代码为如下文件`PageRank.py`。下面介绍文件结构：

1. `__init__`。初始化函数，配置参数，其中`beta=0.85`是默认值，这与作业要求一致。
2. `load_and_process_data`。负责导入和处理数据，并且构建列表，准备好数据结构。
3. `page_rank`。核心算法函数。
4. `save_result`。保存结果为 txt 类型，并且保证结果格式与要求一致。
5. `run_workflow`。工作流封装，包含时间测算和结果的封装，结果格式为：

```
statistics = {
    'Block Num.': self.block_num,
```

```
        'Alg. Time': end_page_rank - start_page_rank,
        'Total Time': end - start
    }
```

第 3 章　数据集说明

本次实验的数据为`WikiData.txt`，共 103689 行数据，其格式为 <node1_id><node2_id>。

第 4 章　项目配置方法

4.1　运行 exe

若您只需要运行 exe 文件，请将数据集放在与`PageRank.exe`同一个文件夹下，然后双击运行即可。

4.2　源代码

本部分供教授和助教检查所用。请打开项目，若您直接运行，则是**未分块**的状态，您可以调整如下代码来改变分块数:

```
pr = PageRank(
    beta=0.85,
    max_iter=100,
    tol=1e-16,
    block_num=1,   # 改变此参数
    data_path='WikiData.txt',
    result_path='result.txt',
    report_top_num=100
)
```

后续章节中有我们做实验的结果（遍历`tol`和`block_num`），若您想看这一部分结果是如何跑出来的，请注释掉前一段代码，并且对后一段代码解开注释即可。

第 5 章　项目逻辑介绍

5.1　核心功能

本小节直接切入核心的功能实现（PageRank），该算法的核心思想是流和投票，用其他节点的投票和赋权来更新本节点的权重。在算法中的体现为：

```
def page_rank(self):
    self.load_and_process_data()
    self.old_rank = np.full(self.N, 1 / self.N, dtype=float)
    for i in range(self.max_iter):
        self.new_rank = np.zeros(self.N, dtype=float)
        if self.block_num == 1:
            for node, [degree, links] in self.out_links.items():
                for link in links:
                    self.new_rank[link - 1] +=
                        self.beta * self.old_rank[node - 1]/
                        degree
        else:
            for block_path in self.blocks:
                block = np.load(block_path, allow_pickle=True).item()
                for node, links in block.items():
                    for link in links:
                        self.new_rank[link - 1] +=
                        self.beta *
                        self.old_rank[node - 1] /
                        self.out_degree[node]
        self.new_rank += (1 - np.sum(self.new_rank)) / self.N
        convergence = np.sum(np.fabs(self.old_rank - self.new_rank))
        print('iteration times:',
              i + 1,
              ', convergence:',
              np.round(convergence, 3))
        if convergence < self.tol:
            break
        self.old_rank = self.new_rank
```

注：受限于报告空间，以上代码部分提前换行以适应页边距，源码中缩进符合要求。

在本段代码中，主要按照如下路径实现：

- **迭代**。根据设置，整个训练会按照`self.max_iter`来进行迭代。
- **初始化**。对`self.old_rank`，采用的函数为：

  ```
  self.old_rank = np.full(self.N, 1 / self.N, dtype=float)
  ```

  也就是使用`numpy`进行向量填充，初始化为 $\frac{1}{N}$。
- **更新处理**。在此过程中，我们遵循：$r(t+1) = \mathbf{M} \cdot r^{(t)}$，但在算法实现上，我们仍然是按照迭代更新的形式（因为邻接关系不再是以邻接矩阵的方式组织，而是以列表的形式）。
- **收敛判定**。根据课堂所学知识，我们的收敛判定条件是 $\left|\mathbf{r}^{(t+1)} - \mathbf{r}^{(t)}\right|_1 < \varepsilon$，其算法实现则为：

  ```
  np.sum(np.fabs(self.old_rank - self.new_rank))
      < self.tol
  ```

- **分块**。为了对大数据过程中可能造成的内存不足，我们在主函数中就分块进行了处理，其算法请详见*拓展功能 3：分块处理*

5.2 拓展功能 1：解决 Dead End 和 Spider Trap 问题

我们可以把 Dead End 理解为只有一个节点的 Spider Trap 问题，解决该类方法是用理想化的局部结构修补非理性化理想局部结构的缺陷。在算法之中，我们将首先以 β 的概率完成原有结构的能量传输，而在结束本轮迭代后，用总体能量与 1 之间的差距填补每个结点，形成：

$$r_j^{\text{new}} = r_j'^{new} + \frac{1 - S}{N}$$

我们在算法中实现了该内容，其关键步骤在于：

```
self.new_rank[link - 1] +=
    self.beta
    * self.old_rank[node - 1]
    / degree
# ...
self.new_rank += (1 - np.sum(self.new_rank)) / self.N
```

5.3 拓展功能 2：优化稀疏矩阵的问题

在导入数据集的过程中，若是稀疏矩阵的整体导入，会因为存下来很多 0 而导致较为浪费存储空间，因此我们实现的列表存储格式如下所示：

$$< src. >, < degree >, < dest. >$$

本组的导入代码如下所示：

```
data = np.loadtxt(self.data_path, dtype=int)
for edge in data:
    self.out_links.setdefault(edge[0], [0, []])
    self.out_links[edge[0]][1].append(edge[1])
    self.out_links[edge[0]][0] += 1

    # 统计节点
    self.N = edge[0] if edge[0] > self.N else self.N
    self.N = edge[1] if edge[1] > self.N else self.N
```

5.4 拓展功能 3：分块处理

为了保证大数据也可以加载到内存来，我们采取分块处理，分块处理的代码如下所示：

```
with open(self.data_path, 'r') as f:
# 统计所有的节点个数
edge = f.readline()
while edge:
    edge = np.array(edge.split()).astype(int)
    self.N = edge[0] if edge[0] > self.N else self.N
    self.N = edge[1] if edge[1] > self.N else self.N
    edge = f.readline()
# 为节点分块并将块存储到磁盘
step = int(np.ceil(self.N / self.block_num))
for block_id, start_node in enumerate(range(1, self.N + 1, step)):
    f.seek(0)
    out_links = {}
    edge = f.readline()
    while edge:
```

```
            edge = np.array(edge.split()).astype(int)
            if start_node <= edge[1] < start_node + step:
                out_links.setdefault(edge[0], [])
                out_links[edge[0]].append(edge[1])
                self.out_degree.setdefault(edge[0], 0)
                self.out_degree[edge[0]] += 1
            edge = f.readline()
        self.blocks.append('block' + str(block_id + 1) + '.npy')
        np.save(self.blocks[-1], out_links)
```

5.5 拓展功能 4：Gamma 压缩

该功能本质上并不直接寄生于 PageRank 之中，但是会影响整个数据集在硬盘中存储的效率，我们采用的压缩算法是 Gamma 算法。

我们压缩的代码实现如下所示：

```
class GammaCompressor:
@staticmethod
def int_to_bin(int_in):
    if int_in == 0:
        return '0'
    # 去除第一个位，因此是print(bin(5)) 0b101
    ret = '1' * int(log(int_in, 2)) + '0' + bin(int_in)[3:]
    return ret

@staticmethod
@timethis
def encode(postings_list):
    ### Begin your code
    encoded_postings_list = ''
    for i in range(0, len(postings_list)):
        # 加一是为了处理0和1的情况
        encoded_postings_list +=
            GammaCompressor
            .int_to_bin(postings_list[i] + 1)
    return encoded_postings_list.encode()
```

```
@staticmethod
def decode(encoded_postings):
    res = []
    i = 0
    byteslen = len(encoded_postings)
    encoded_postings = encoded_postings.decode()
    while True:
        length = 0
        while encoded_postings[i] != '0':
            i += 1
            length += 1
        i += 1
        val = int('1' + encoded_postings[i:i + length], 2) - 1
        res.append(val)
        i = i + length
        if i >= byteslen:
            return res
```

5.6 拓展功能 5：cython 对代码的优化

我们对于 cython 的实现在 jupyter 中实现，其优化效果由后续实验结果进行报告：

```
%%cython
cimport numpy as np
cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
def update_rank_single_blocks(
    dict out_links,
    np.ndarray[double,ndim=1] new_rank,
    np.ndarray[double,ndim=1] old_rank,
    double alpha
    ):
    cdef int node, degree, link
    for node, [degree, links] in out_links.items():
        for link in links:
```

```
                new_rank[link - 1] +=
                alpha *
                old_rank[node - 1] /
                degree
        return new_rank

    @cython.boundscheck(False)
    @cython.wraparound(False)
    def update_rank_mtlti_blocks(
        dict blocks,
        np.ndarray[double,ndim=1] new_rank,
        np.ndarray[double,ndim=1] old_rank,
        double alpha,
        dict out_degree
        ):
        cdef int node, degree, link
        for node, links in blocks.items():
            for link in links:
                new_rank[link - 1] +=
                    alpha *
                    old_rank[node - 1] /
                    out_degree[node]
        return new_rank
```

本质上是将 $.pyx \rightarrow$ 编译为 c 文件 $\rightarrow$ 导入 python 文件。本组成员担心教授或助教的计算机中无 cython 环境可能导致影响整个程序的检查，因此在我们的源码中没有包含 cython 源码，而在报告中列出。其对于性能的提高请参考实验结果部分。

第 6 章　实验结果和对比

6.1　运行截图和结果格式

如图6.1所示，为我们程序的运行截图。结果的格式如下：

```
4037 0.004347713867098029
15 0.0034726271866630945
6634 0.0033848534953935423
2625 0.0030987322875347645
```

```
C:\Users\Lenovo\AppData\Local\Programs\Python\Python38\python.exe "
Start paging rank
iter 0, 1.0
iter 1, 1.0
iter 2, 1.0
iter 3, 1.0
iter 4, 1.0
iter 5, 1.0
iter 6, 1.0
iter 7, 1.0
iter 8, 1.0
iter 9, 1.0
iter 10, 1.0
iter 11, 1.0
iter 12, 1.0
iter 13, 1.0
iter 14, 1.0
iter 15, 1.0
Running time: 4.1199825 Seconds
Total running time: 4.1264341 Seconds
```

图 6.1: 运行过程截图

```
2398 0.002461726285154734
2470 0.002381641899254392
2237 0.0023560255735935462
4191 0.002140134444211437
7553 0.0020475389709032465
5254 0.0020290145327110805
...
```

6.2 实验 1：时间随块数的变化

6.2.1 实验流程

将分块数从 1（也就是没有分块）到 10 遍历，收敛阈值分别检查 $10^{-4}, 10^{-8}, 10^{-16}$ 并且分别记录实际算法的时间，我们将结果记录并且画图。

6.2.2 实验结果

得到随着分块数递增，可以得出时间变化如下所示：

```
{0.0001:
    [3.7367491722106934, 5.791176080703735, 6.607737064361572,
    6.853483438491821, 7.093072414398193, 7.528308153152466,
     8.490409851074219, 9.005643129348755, 10.196325063705444,
     10.316315174102783],
 1e-08:
    [8.076581239700317, 12.629549264907837, 12.315329313278198,
    13.377459526062012, 13.611292600631714, 14.708141565322876,
     14.341797590255737, 14.0680570602417, 14.621425151824951,
      14.886704683303833],
1e-16:
    [16.390163898468018, 23.26309561729431, 25.145299911499023,
    23.675409078598022, 24.756206035614014, 25.343189239501953,
    25.43118977546692, 26.774394989013672, 28.080437898635864,
    27.434569597244263]}
```

我们进行作图分析，可以得到如图6.2所示。其中图例表示的是收敛率对时间的影响

根据图像可知，时间近似于是线性增长的，这与预期是**一致**的。每轮迭代数量贡献的复杂度为 $O(max\_iter)$，而每一块都会运行 $O(max\_iter)$ 的代数，所以总体时间损耗大致为：

$$O(max\_iter \times Blocks \times Complexity\_of\_PageRank)$$

因为我们重点关注分块的影响，所以第三项复杂度就不详细说明了，因为它和 $Blocks$ 无关，以上我们可以发现，总复杂度确实随着分块数增加而线性增加。

6.3 实验 2：时间因 cython 而优化的情况

本部分汇报因为 cython 而带来的时间优化，本组做了和无 cython 情况下一样的时间检测实验。如图6.3所示。我们发现：

1. 在无分块的时候优化效率最高，原因是 IO 在分块情况下造成了主要是时间消耗，且关于 block 的循环并不在 cython 的优化过程中。

2. 效率最高可以达到 30 倍，足以见 C++ 和 Python 之间的效率差距。

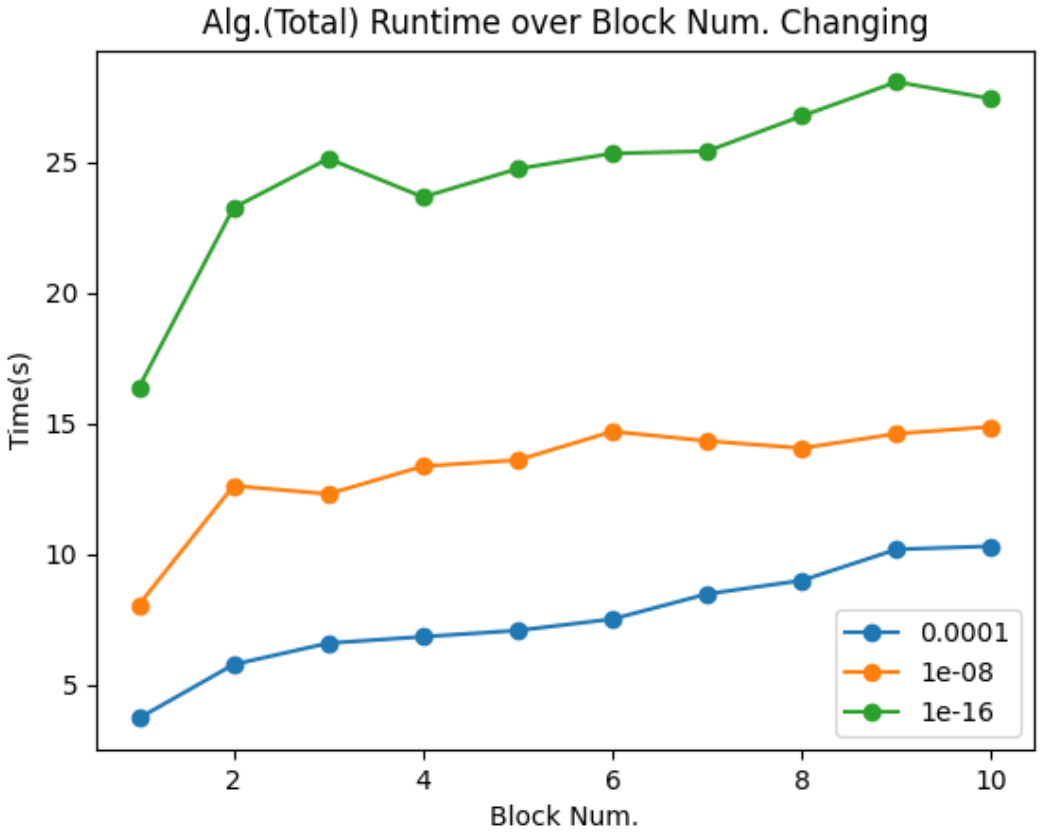

图 6.2: 运行时间随分块数变化情况

第 7 章　总结

本次作业中，小组成员均在代码、实验和报告中有所贡献，我们对 PageRank 算法较为感兴趣，较为深入地学习、实现和改进算法。科研和工程能力在本次作业中得到了很好地锻炼。同时，cython，Gamma压缩也是我们以前没有接触过的新知识，就像知识图谱一样，由一个源点（PageRank），联系了其他知识，也是高效学习的方法之一。

Appendices

第 A 章　cython 优化后的代码

请在 jupyter 中运行：

```
import os
import time
```

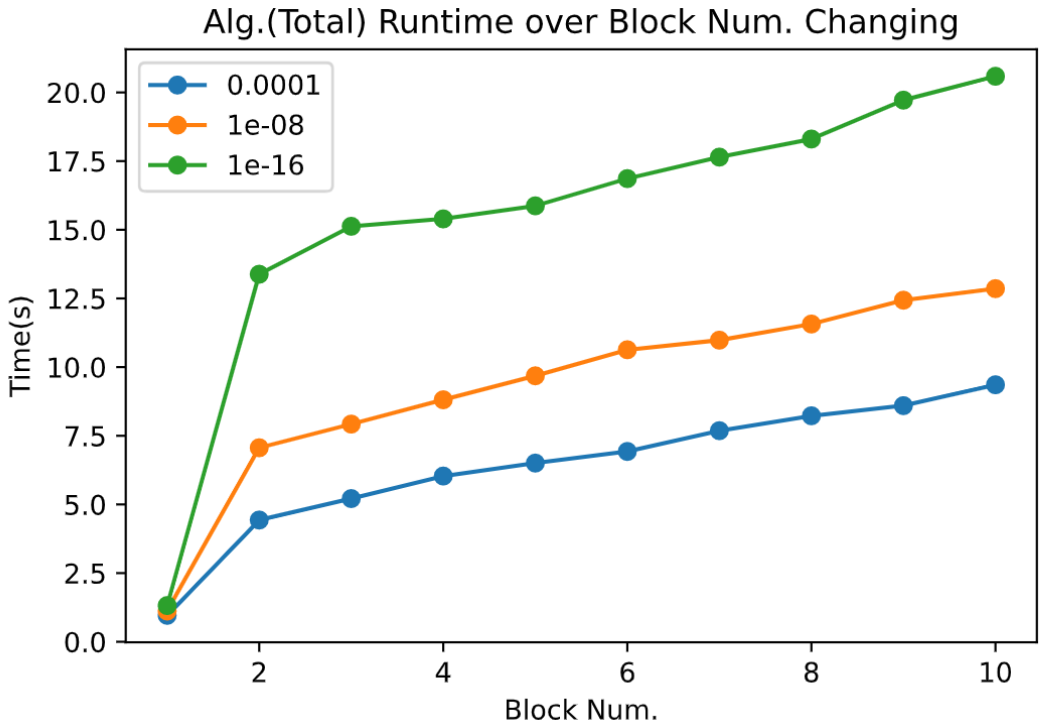

图 6.3: 有 cython 优化的情况下运行时间随分块数变化情况

```
import numpy as np
from functools import wraps
from math import log

%%cython



cimport numpy as np
cimport cython

@cython.boundscheck(False)
@cython.wraparound(False)
def update_rank_single_blocks(dict out_links,
np.ndarray[double,ndim=1] new_rank,
np.ndarray[double,ndim=1] old_rank,
double alpha):
    cdef int node, degree, link
    for node, [degree, links] in out_links.items():
```

```
        for link in links:
            new_rank[link - 1] +=
            alpha *
            old_rank[node - 1] /
             degree
    return new_rank

@cython.boundscheck(False)
@cython.wraparound(False)
def update_rank_mtlti_blocks(dict blocks,
np.ndarray[double,ndim=1] new_rank,
np.ndarray[double,ndim=1] old_rank,
double alpha,
dict out_degree):
    cdef int node, degree, link
    for node, links in blocks.items():
        for link in links:
            new_rank[link - 1] +=
             alpha *
             old_rank[node - 1] /
             out_degree[node]
    return new_rank

import time
import numpy as np
import matplotlib.pyplot as plt


class PageRank:
def __init__(self,
                 beta=0.85,
                 max_iter=100,
                 tol=1.0e-16,
                 block_num=0,
                 data_path='WikiData.txt',
                 result_path='result.txt',
                 report_top_num=100):
```

```
        self.beta = beta
        self.max_iter = max_iter
        self.tol = tol
        self.N = 0
        self.out_links = {}
        self.block_num = block_num
        self.blocks = []
        self.out_degree = {}
        self.data_path = data_path
        self.result_path = result_path
        self.report_top_num = report_top_num
        self.new_rank = None
        self.old_rank = None

    def run_workflow(self):
        self.load_and_process_data()
        start = time.time()

        print('Start paging rank')
        start_page_rank = time.time()
        self.page_rank()
        end_page_rank = time.time()
        print('Running time: %s Seconds'
        % (end_page_rank - start_page_rank))

        self.save_result()
        end = time.time()
        print('Total running time: %s Seconds' % (end - start))

        statistics = {
            'Block Num.': self.block_num,
            'Alg. Time': end_page_rank - start_page_rank,
            'Total Time': end - start
        }
        return statistics
```

```
def load_and_process_data(self):
    if self.block_num == 1:
        data = np.loadtxt(self.data_path, dtype=int)
        for edge in data:
            self.out_links.setdefault(edge[0], [0, []])
            self.out_links[edge[0]][1].append(edge[1])
            self.out_links[edge[0]][0] += 1

            # 统计节点
            self.N = edge[0] if edge[0] > self.N else self.N
            self.N = edge[1] if edge[1] > self.N else self.N

    else:
        with open(self.data_path, 'r') as f:
            # 统计所有的节点个数
            edge = f.readline()
            while edge:
                edge = np.array(edge.split()).astype(int)
                self.N = edge[0] if edge[0] >
                    self.N else self.N
                self.N = edge[1] if edge[1] >
                    self.N else self.N
                edge = f.readline()
            # 为节点分块并将块存储到磁盘
            step = int(np.ceil(self.N / self.block_num))
            for block_id, start_node in enumerate(
                    range(1, self.N + 1, step)
                ):
                f.seek(0)
                out_links = {}
                edge = f.readline()
                while edge:
                    edge = np.array(edge.split()).astype(int)
                    if start_node <= edge[1] < start_node + step:
                        out_links.setdefault(edge[0], [])
                        out_links[edge[0]].append(edge[1])
```

```
                        self.out_degree.setdefault(edge[0], 0)
                        self.out_degree[edge[0]] += 1
                    edge = f.readline()
                self.blocks.append('block' +
                     str(block_id + 1) +
                      '.npy')
                np.save(self.blocks[-1], out_links)

def page_rank(self):
    self.load_and_process_data()
    self.old_rank = np.full(self.N, 1 / self.N, dtype=float)
    for i in range(self.max_iter):
        self.new_rank = np.zeros(self.N, dtype=float)
        if self.block_num == 1:
                update_rank_single_blocks(self.out_links,
                    self.new_rank,
                    self.old_rank,
                    self.beta)
        else:
            for block_path in self.blocks:
                block = np.load(block_path, allow_pickle=True).item()
                update_rank_mtlti_blocks(block,
                self.new_rank,
                self.old_rank,
                self.beta,
                self.out_degree)
        self.new_rank += (1 - np.sum(self.new_rank)) / self.N
        convergence = np.sum(np.fabs(self.old_rank - self.new_rank))
        print('iteration times:',
         i + 1,
         ',convergence:',
         np.round(convergence, 3))
        if convergence < self.tol:
            break
        self.old_rank = self.new_rank
```

```
def save_result(self):
    result = sorted(zip(
        self.new_rank,
         range(1, self.N + 1)),
        reverse=True)
    with open(self.result_path, 'w') as f:
        for i in range(self.report_top_num):
            f.write(
                str(result[i][1]) + ' ' + str(result[i][0]) + '\n'
            )


if __name__ == '__main__':
'''
此处是可供您检查的代码，若您更改block_num参数，即可以调试分块的情况
'''
# pr = PageRank(
#     beta=0.85,
#     max_iter=100,
#     tol=1e-12,
#     block_num=2,
#     data_path='WikiData.txt',
#     result_path='result.txt',
#     report_top_num=100
# )

# pr.run_workflow().get('Alg. Time')

'''
下面是我们跑实验的代码，如果您需要检查最终结果是否正确，请不必解开注释。
若您想检查我们对于遍历<tol>和<blocks_num>对实验结果的检查，请解开注释

'''
block_num_range = range(1, 11)
tol_2_time = {
    1e-4: [],
```

```
    1e-8: [],
    1e-16: []
}

for tol in tol_2_time.keys():
    for block_num in block_num_range:
        pr = PageRank(
            beta=0.85,
            max_iter=100,
            tol=tol,
            block_num=block_num,
            data_path='WikiData.txt',
            result_path='result.txt',
            report_top_num=100
        )

        tol_2_time.get(tol).append(pr.run_workflow().get('Alg. Time'))
    plt.plot(list(range(1, 11)), tol_2_time.get(tol), 'o-')

print(tol_2_time)
plt.legend(tol_2_time.keys())
plt.xlabel("Block Num.")
plt.ylabel("Time(s)")
plt.title("Alg.(Total) Runtime over Block Num. Changing")
plt.show()
```