

Question 1: Process Interleaving: Straight-Line Code [15 marks]

- Consider whether all statements are atomic and transform the program to load/store architecture if needed.

In the current code, there are two atomic instructions:

P1: $n := k + 1$

Q1: $k := n - 1$

This can be achieved using synchronization mechanisms.

- Identify the conflicting pairs of atomic instructions.

Conflicting pair:

P1: $n := k + 1$

Q1: $k := n - 1$

These instructions conflict because they both access and modify shared variables (n and k).

- State an upper bound on the possible number of outcomes for this program based on the number of processes and atomic instructions in this program.

There are two processes (P and Q) and two atomic instructions. Each process executes one atomic instruction.

The upper bound on the possible number of outcomes is determined by the number of possible interleavings of these instructions. In this case, it's 2 factorial, which equals 2 possible outcomes:

1. P1 executes before Q1.
2. Q1 executes before P1.

- State an upper bound on the possible number of outcomes for this program based on the number of conflicting pairs found.

Since there is only one conflicting pair of atomic instructions (P1 and Q1), the upper bound on the possible number of outcomes is still 2. This is because the conflict between P1 and Q1 creates a partial order, and the order of execution between these two instructions is the only source of non-determinism.

- What are the possible final values of 'n' and 'k'? Justify your answer.

As mentioned above, there are two possible outcomes based on the order of execution of conflicting pairs:

1. P1 executes before Q1:

$$n := k + 1 \Rightarrow n := 1 + 1 \Rightarrow n := 2$$
$$k := n - 1 \Rightarrow k := 2 - 1 \Rightarrow k := 1$$

Final values: $n = 2, k = 1$

2. Q1 executes before P1:

$$k := n - 1 \Rightarrow k := 1 - 1 \Rightarrow k := 0$$
$$n := k + 1 \Rightarrow n := 0 + 1 \Rightarrow n := 1$$

Final values: $n = 1, k = 0$

Justify code:

```
1 import java.util.concurrent.atomic.AtomicInteger;
2
3 public class q1 {
4     public static void main(String[] args) {
5         int i = 1;
6         while(i ≤ 30){
7             AtomicInteger n = new AtomicInteger(1);
8             AtomicInteger k = new AtomicInteger(1);
9
10
11             Thread pThread = new Thread(() → {
```

```

12         n.set(k.get() + 1);
13     });
14     Thread qThread = new Thread(() → {
15         k.set(n.get() - 1);
16     });
17
18     pThread.start();
19     qThread.start();
20
21     try {
22         pThread.join();
23         qThread.join();
24     } catch (InterruptedException e) {
25         e.printStackTrace();
26     }
27
28     System.out.println(i + ": k = " + k + " n =
" + n);
29     i++;
30 }
31
32
33 }
34 }
35

```

out:

```

1 1: k = 1 n = 2
2 2: k = 1 n = 2
3 3: k = 1 n = 2
4 4: k = 1 n = 2
5 5: k = 1 n = 2
6 6: k = 1 n = 2
7 7: k = 1 n = 2
8 8: k = 1 n = 2
9 9: k = 1 n = 2
10 10: k = 0 n = 1
11 11: k = 1 n = 2
12 12: k = 0 n = 1
13 13: k = 1 n = 2

```

14	14: k = 1 n = 2
15	15: k = 1 n = 2
16	16: k = 1 n = 2
17	17: k = 1 n = 2
18	18: k = 1 n = 2
19	19: k = 1 n = 2
20	20: k = 1 n = 2
21	21: k = 1 n = 2
22	22: k = 1 n = 2
23	23: k = 1 n = 2
24	24: k = 1 n = 2
25	25: k = 1 n = 2
26	26: k = 1 n = 2
27	27: k = 1 n = 2
28	28: k = 1 n = 2
29	29: k = 1 n = 2
30	30: k = 1 n = 2

```
1: k = 1 n = 2
2: k = 1 n = 2
3: k = 1 n = 2
4: k = 1 n = 2
5: k = 1 n = 2
6: k = 1 n = 2
7: k = 1 n = 2
8: k = 1 n = 2
9: k = 1 n = 2
10: k = 0 n = 1
11: k = 1 n = 2
12: k = 0 n = 1
13: k = 1 n = 2
14: k = 1 n = 2
15: k = 1 n = 2
16: k = 1 n = 2
17: k = 1 n = 2
18: k = 1 n = 2
```

Question 2: Process Interleaving with Control Flow [10 marks]

- Consider whether all statements are atomic and transform the program to load/store architecture if needed

`int n = 1;` and `boolean flag = true;`: These are initialization statements and can be considered atomic since they are typically performed as a single operation.

`while(n == 1){};`: This is a loop that checks the value of 'n'. In a load/store architecture, reading 'n' would be a load operation, and the comparison can be considered atomic.

`flag = false;`: This is a store operation, and it can be considered atomic.

- Identify all conflicting pairs of atomic instructions

Conflicting Pair:

`while(n == 1){};` (Process P) and `n = 1 - n;` (Process Q)

- Construct an interleaving of instructions of the processes 'P' and 'Q' for which the program terminates

Construct need a fair scheduler that alternates between processes P and Q. Here's one possible interleaving that leads to program termination:

```
1 | P: int n = 1;
2 | P: boolean flag = true;
3 | Q: while(n == 1){};
4 | P: flag = false;
5 | Q: n = 1 - n;
```

- What are the possible values of 'n' when the program terminates? Can you argue that other values than those you have listed for 'n' are impossible?

In this program, 'n' is initially set to 1. The only operation that modifies 'n' is `n = 1 - n;` in Process Q. This operation toggles the value of 'n' between 0 and 1. Therefore, the possible values of 'n' when the program terminates are 0 and 1. Other values are impossible because no other part of the program modifies 'n'.

- Does the program terminate for all ways to interleave instructions assuming a fair scheduler? Does the program also terminate for execution scenarios that are unique to an unfair scheduler? Justify your answers.

(Reminder: with a fair scheduler no process is deferred forever).

With a fair scheduler, the program will terminate because neither Process P nor Process Q is deferred forever. The program may take some time to complete, but it will eventually do so.

With an unfair scheduler, the program may still terminate, but it's not guaranteed. If the scheduler always prioritizes one process (e.g., P) over the other, the program may run indefinitely because the non-prioritized process (e.g., Q) may never get a chance to execute.

Java code:

```
1 public class Q2 {
2     static int n = 1;
3     static boolean flag = true;
4
5     public static void main(String[] args) {
6         // Fair Scheduler
7         Thread threadP = new Thread(() → {
8             while (n == 1) {
9                 // Busy-wait until n is not equal to 1
10            }
11            flag = false;
12        });
13
14        Thread threadQ = new Thread(() → {
15            while (flag == true) {
16                n = 1 - n;
17            }
18        });
19
20        threadP.start();
21        threadQ.start();
22
23        try {
24            threadP.join();
25            threadQ.join();
26        } catch (InterruptedException e) {
```

```

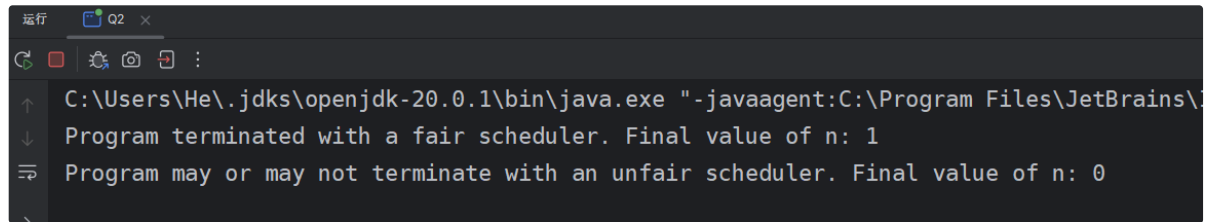
27         e.printStackTrace();
28     }
29
30     System.out.println("Program terminated with a
fair scheduler. Final value of n: " + n);
31
32     // Unfair Scheduler
33     n = 1;
34     flag = true;
35
36     Thread threadPUnfair = new Thread(() → {
37         while (n == 1) {
38             // Busy-wait until n is not equal to 1
39         }
40         flag = false;
41     });
42
43     Thread threadQUnfair = new Thread(() → {
44         while (flag == true) {
45             n = 1 - n;
46         }
47     });
48
49     threadPUnfair.start();
50     threadQUnfair.start();
51
52     // Sleep to simulate an unfair scheduler that
frequently selects P
53     try {
54         Thread.sleep(100);
55     } catch (InterruptedException e) {
56         e.printStackTrace();
57     }
58
59     threadPUnfair.interrupt();
60     threadQUnfair.interrupt();
61
62     System.out.println("Program may or may not
terminate with an unfair scheduler. Final value of n: "
+ n);
63 }

```



```
64 }  
65
```

out:



```
运行 Q2 x  
C:\Users\He\.jdk\openjdk-20.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\  
Program terminated with a fair scheduler. Final value of n: 1  
Program may or may not terminate with an unfair scheduler. Final value of n: 0
```

Question 3: Graph Processing - Introduction [15 marks]

- Record the residual error (printed to the console by the PageRank algorithm) at the end of each iteration of the power method for each of the three matrix formats. Plot these values in a chart (consider log-scale).
Summarise your observations

C00:

```
Format: C00  
Input file CSR: rMatGraph_J_5_100.csr  
Input file CSC: rMatGraph_J_5_100.csc  
Input file C00: rMatGraph_J_5_100.coo  
Algorithm: pr  
Number of threads: 1  
Output file: outputfileC00.txt  
Reading input: 0.0026684 seconds  
Number of vertices: 128  
Number of edges: 708  
Initialisation: 3.61E-5 seconds  
iteration 1: residual error=0.39273487186470324 xnorm=1.0  
time=0.0011051000000000001 seconds  
iteration 2: residual error=0.15082251467521354 xnorm=1.0  
time=6.97E-5 seconds
```

iteration 3: residual error=0.06098310915896902 xnorm=1.0
time=1.988E-4 seconds
iteration 4: residual error=0.03363108589428869 xnorm=1.0
time=6.790000000000001E-5 seconds
iteration 5: residual error=0.016629581949571277 xnorm=1.0
time=1.0630000000000001E-4 seconds
iteration 6: residual error=0.0095595094132368 xnorm=1.0
time=6.46E-5 seconds
iteration 7: residual error=0.005420271938283015 xnorm=1.0
time=6.3900000000000001E-5 seconds
iteration 8: residual error=0.003089775698722849 xnorm=1.0
time=6.2800000000000001E-5 seconds
iteration 9: residual error=0.00188182364267449 xnorm=1.0
time=6.06E-5 seconds
iteration 10: residual error=0.0010947688160630768
xnorm=1.0 time=6.15E-5 seconds
iteration 11: residual error=6.901694755585418E-4 xnorm=1.0
time=6.15E-5 seconds
iteration 12: residual error=4.0732532906687883E-4
xnorm=1.0 time=5.86E-5 seconds
iteration 13: residual error=2.622110797494007E-4 xnorm=1.0
time=4.8E-5 seconds
iteration 14: residual error=1.5764434900316813E-4
xnorm=1.0 time=4.7300000000000005E-5 seconds
iteration 15: residual error=1.0280682314865123E-4
xnorm=1.0 time=5.86E-5 seconds
iteration 16: residual error=6.208993245162718E-5 xnorm=1.0
time=6.37E-5 seconds
iteration 17: residual error=4.1069227137087265E-5
xnorm=1.0 time=6.2E-5 seconds
iteration 18: residual error=2.4937456845666815E-5
xnorm=1.0 time=6.21E-5 seconds
iteration 19: residual error=1.6657078937420185E-5
xnorm=1.0 time=6.54E-5 seconds
iteration 20: residual error=1.0190979647899025E-5
xnorm=1.0 time=6.35E-5 seconds
iteration 21: residual error=6.821510338905854E-6 xnorm=1.0
time=6.27E-5 seconds
iteration 22: residual error=4.212100413846474E-6 xnorm=1.0
time=6.31E-5 seconds
iteration 23: residual error=2.823012179009425E-6 xnorm=1.0

```
time=6.3E-5 seconds
iteration 24: residual error=1.749539231112561E-6 xnorm=1.0
time=6.54E-5 seconds
iteration 25: residual error=1.1744591493544412E-6
xnorm=1.0 time=5.9800000000000003E-5 seconds
iteration 26: residual error=7.296637554413295E-7 xnorm=1.0
time=6.18E-5 seconds
iteration 27: residual error=4.899707951033783E-7 xnorm=1.0
time=6.5600000000000001E-5 seconds
iteration 28: residual error=3.0571754504674087E-7
xnorm=1.0 time=6.35E-5 seconds
iteration 29: residual error=2.0493334797980906E-7
xnorm=1.0 time=6.91E-5 seconds
iteration 30: residual error=1.286083027587058E-7 xnorm=1.0
time=6.42E-5 seconds
iteration 31: residual error=8.596268075058283E-8 xnorm=1.0
time=6.24E-5 seconds
PageRank: total time: 0.031098100000000004 seconds
Writing file: 0.011592 seconds
All done
```

```
Format: COO
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCOO.txt
Reading input: 0.0026684 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 3.61E-5 seconds
iteration 1: residual error=0.39273487186470324 xnorm=1.0 time=0.0011051000000000001 seconds
iteration 2: residual error=0.15082251467521354 xnorm=1.0 time=6.97E-5 seconds
iteration 3: residual error=0.06098310915896902 xnorm=1.0 time=1.988E-4 seconds
iteration 4: residual error=0.03363108589428869 xnorm=1.0 time=6.7900000000000001E-5 seconds
iteration 5: residual error=0.016629581949571277 xnorm=1.0 time=1.0630000000000001E-4 seconds
iteration 6: residual error=0.0095595094132368 xnorm=1.0 time=6.46E-5 seconds
iteration 7: residual error=0.005420271938283015 xnorm=1.0 time=6.3900000000000001E-5 seconds
iteration 8: residual error=0.003089775698722849 xnorm=1.0 time=6.2800000000000001E-5 seconds
iteration 9: residual error=0.00188182364267449 xnorm=1.0 time=6.06E-5 seconds
iteration 10: residual error=0.0010947688160630768 xnorm=1.0 time=6.15E-5 seconds
iteration 11: residual error=6.901694755585418E-4 xnorm=1.0 time=6.15E-5 seconds
iteration 12: residual error=4.0732532906687883E-4 xnorm=1.0 time=5.86E-5 seconds
```

```
iteration 13: residual error=2.622110797494007E-4 xnorm=1.0 time=4.8E-5 seconds
iteration 14: residual error=1.5764434900316813E-4 xnorm=1.0 time=4.7300000000000005E-5 seconds
iteration 15: residual error=1.0280682314865123E-4 xnorm=1.0 time=5.86E-5 seconds
iteration 16: residual error=6.208993245162718E-5 xnorm=1.0 time=6.37E-5 seconds
iteration 17: residual error=4.1069227137087265E-5 xnorm=1.0 time=6.2E-5 seconds
iteration 18: residual error=2.4937456845666815E-5 xnorm=1.0 time=6.21E-5 seconds
iteration 19: residual error=1.6657078937420185E-5 xnorm=1.0 time=6.54E-5 seconds
iteration 20: residual error=1.0190979647899025E-5 xnorm=1.0 time=6.35E-5 seconds
iteration 21: residual error=6.821510338905854E-6 xnorm=1.0 time=6.27E-5 seconds
iteration 22: residual error=4.212100413846474E-6 xnorm=1.0 time=6.31E-5 seconds
iteration 23: residual error=2.823012179009425E-6 xnorm=1.0 time=6.3E-5 seconds
iteration 24: residual error=1.749539231112561E-6 xnorm=1.0 time=6.54E-5 seconds
iteration 25: residual error=1.1744591493544412E-6 xnorm=1.0 time=5.9800000000000003E-5 seconds
iteration 26: residual error=7.296637554413295E-7 xnorm=1.0 time=6.18E-5 seconds
iteration 27: residual error=4.899707951033783E-7 xnorm=1.0 time=6.560000000000001E-5 seconds
iteration 28: residual error=3.0571754504674087E-7 xnorm=1.0 time=6.35E-5 seconds
iteration 29: residual error=2.0493334797980906E-7 xnorm=1.0 time=6.91E-5 seconds
iteration 30: residual error=1.286083027587058E-7 xnorm=1.0 time=6.42E-5 seconds
iteration 31: residual error=8.596268075058283E-8 xnorm=1.0 time=6.24E-5 seconds
PageRank: total time: 0.031098100000000004 seconds
Writing file: 0.011592 seconds
All done
```

CSC:

```
Format: CSC
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCSC.txt
Reading input: 0.0017436 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 0.0015629 seconds
iteration 1: residual error=0.39273487186470324 xnorm=1.0
time=0.0010702 seconds
iteration 2: residual error=0.15082251467521357 xnorm=1.0
time=6.96E-5 seconds
iteration 3: residual error=0.06098310915896904 xnorm=1.0
time=6.29E-5 seconds
iteration 4: residual error=0.03363108589428868 xnorm=1.0
time=4.75E-5 seconds
iteration 5: residual error=0.016629581949571294 xnorm=1.0
time=4.2300000000000005E-5 seconds
iteration 6: residual error=0.009559509413236817 xnorm=1.0
time=4.2300000000000005E-5 seconds
iteration 7: residual error=0.005420271938283045 xnorm=1.0
```

time=4.8400000000000004E-5 seconds
iteration 8: residual error=0.0030897756987228217 xnorm=1.0
time=4.33E-5 seconds
iteration 9: residual error=0.0018818236426745034 xnorm=1.0
time=4.1500000000000006E-5 seconds
iteration 10: residual error=0.0010947688160630946
xnorm=1.0 time=6.950000000000001E-5 seconds
iteration 11: residual error=6.901694755585205E-4 xnorm=1.0
time=6.0400000000000004E-5 seconds
iteration 12: residual error=4.073253290668758E-4
xnorm=1.0000000000000002 time=7.620000000000001E-5 seconds
iteration 13: residual error=2.622110797494109E-4 xnorm=1.0
time=6.27E-5 seconds
iteration 14: residual error=1.576443490031703E-4 xnorm=1.0
time=6.18E-5 seconds
iteration 15: residual error=1.0280682314864386E-4
xnorm=1.0 time=5.84E-5 seconds
iteration 16: residual error=6.208993245161677E-5 xnorm=1.0
time=8.15E-5 seconds
iteration 17: residual error=4.1069227137101576E-5
xnorm=1.0000000000000002 time=6.2E-5 seconds
iteration 18: residual error=2.4937456845641444E-5
xnorm=1.0 time=6.730000000000001E-5 seconds
iteration 19: residual error=1.6657078937418884E-5
xnorm=1.0 time=6.77E-5 seconds
iteration 20: residual error=1.0190979647901194E-5
xnorm=1.0 time=6.57E-5 seconds
iteration 21: residual error=6.821510338883303E-6 xnorm=1.0
time=8.300000000000001E-5 seconds
iteration 22: residual error=4.212100413835199E-6 xnorm=1.0
time=6.63E-5 seconds
iteration 23: residual error=2.8230121790076904E-6
xnorm=1.0 time=6.8E-5 seconds
iteration 24: residual error=1.749539231122102E-6 xnorm=1.0
time=6.560000000000001E-5 seconds
iteration 25: residual error=1.174459149327553E-6 xnorm=1.0
time=6.450000000000001E-5 seconds
iteration 26: residual error=7.296637554530389E-7 xnorm=1.0
time=6.47E-5 seconds
iteration 27: residual error=4.899707950912352E-7 xnorm=1.0
time=6.69E-5 seconds

```
iteration 28: residual error=3.057175450502103E-7 xnorm=1.0
time=6.37E-5 seconds
iteration 29: residual error=2.0493334797460488E-7
xnorm=1.0 time=6.340000000000001E-5 seconds
iteration 30: residual error=1.286083027790888E-7
xnorm=1.0000000000000002 time=6.65E-5 seconds
iteration 31: residual error=8.596268069615588E-8 xnorm=1.0
time=6.66E-5 seconds
PageRank: total time: 0.0322483 seconds
Writing file: 0.010416700000000001 seconds
All done
```

```
Format: CSC
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCSC.txt
Reading input: 0.0017436 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 0.0015629 seconds
iteration 1: residual error=0.39273487186470324 xnorm=1.0 time=0.0010702 seconds
iteration 2: residual error=0.15082251467521357 xnorm=1.0 time=6.96E-5 seconds
iteration 3: residual error=0.06098310915896904 xnorm=1.0 time=6.29E-5 seconds
iteration 4: residual error=0.03363108589428868 xnorm=1.0 time=4.75E-5 seconds
iteration 5: residual error=0.016629581949571294 xnorm=1.0 time=4.2300000000000005E-5 seconds
iteration 6: residual error=0.009559509413236817 xnorm=1.0 time=4.2300000000000005E-5 seconds
iteration 7: residual error=0.005420271938283045 xnorm=1.0 time=4.8400000000000004E-5 seconds
iteration 8: residual error=0.0030897756987228217 xnorm=1.0 time=4.33E-5 seconds
iteration 9: residual error=0.0018818236426745034 xnorm=1.0 time=4.1500000000000006E-5 seconds
iteration 10: residual error=0.0010947688160630946 xnorm=1.0 time=6.950000000000001E-5 seconds
iteration 11: residual error=6.901694755585205E-4 xnorm=1.0 time=6.0400000000000004E-5 seconds
iteration 12: residual error=4.073253290668758E-4 xnorm=1.0000000000000002 time=7.620000000000001E-5 seconds
```

```
iteration 13: residual error=2.622110797494109E-4 xnorm=1.0 time=6.27E-5 seconds
iteration 14: residual error=1.576443490031703E-4 xnorm=1.0 time=6.18E-5 seconds
iteration 15: residual error=1.0280682314864386E-4 xnorm=1.0 time=5.84E-5 seconds
iteration 16: residual error=6.208993245161677E-5 xnorm=1.0 time=8.15E-5 seconds
iteration 17: residual error=4.1069227137101576E-5 xnorm=1.0000000000000002 time=6.2E-5 seconds
iteration 18: residual error=2.4937456845641444E-5 xnorm=1.0 time=6.730000000000001E-5 seconds
iteration 19: residual error=1.6657078937418884E-5 xnorm=1.0 time=6.77E-5 seconds
iteration 20: residual error=1.0190979647901194E-5 xnorm=1.0 time=6.57E-5 seconds
iteration 21: residual error=6.821510338883303E-6 xnorm=1.0 time=8.300000000000001E-5 seconds
iteration 22: residual error=4.212100413835199E-6 xnorm=1.0 time=6.63E-5 seconds
iteration 23: residual error=2.8230121790076904E-6 xnorm=1.0 time=6.8E-5 seconds
iteration 24: residual error=1.749539231122102E-6 xnorm=1.0 time=6.560000000000001E-5 seconds
iteration 25: residual error=1.174459149327553E-6 xnorm=1.0 time=6.450000000000001E-5 seconds
iteration 26: residual error=7.296637554530389E-7 xnorm=1.0 time=6.47E-5 seconds
iteration 27: residual error=4.899707950912352E-7 xnorm=1.0 time=6.69E-5 seconds
iteration 28: residual error=3.057175450502103E-7 xnorm=1.0 time=6.37E-5 seconds
iteration 29: residual error=2.0493334797460488E-7 xnorm=1.0 time=6.340000000000001E-5 seconds
iteration 30: residual error=1.286083027790888E-7 xnorm=1.0000000000000002 time=6.65E-5 seconds
iteration 31: residual error=8.596268069615588E-8 xnorm=1.0 time=6.66E-5 seconds
PageRank: total time: 0.0322483 seconds
Writing file: 0.010416700000000001 seconds
All done
```

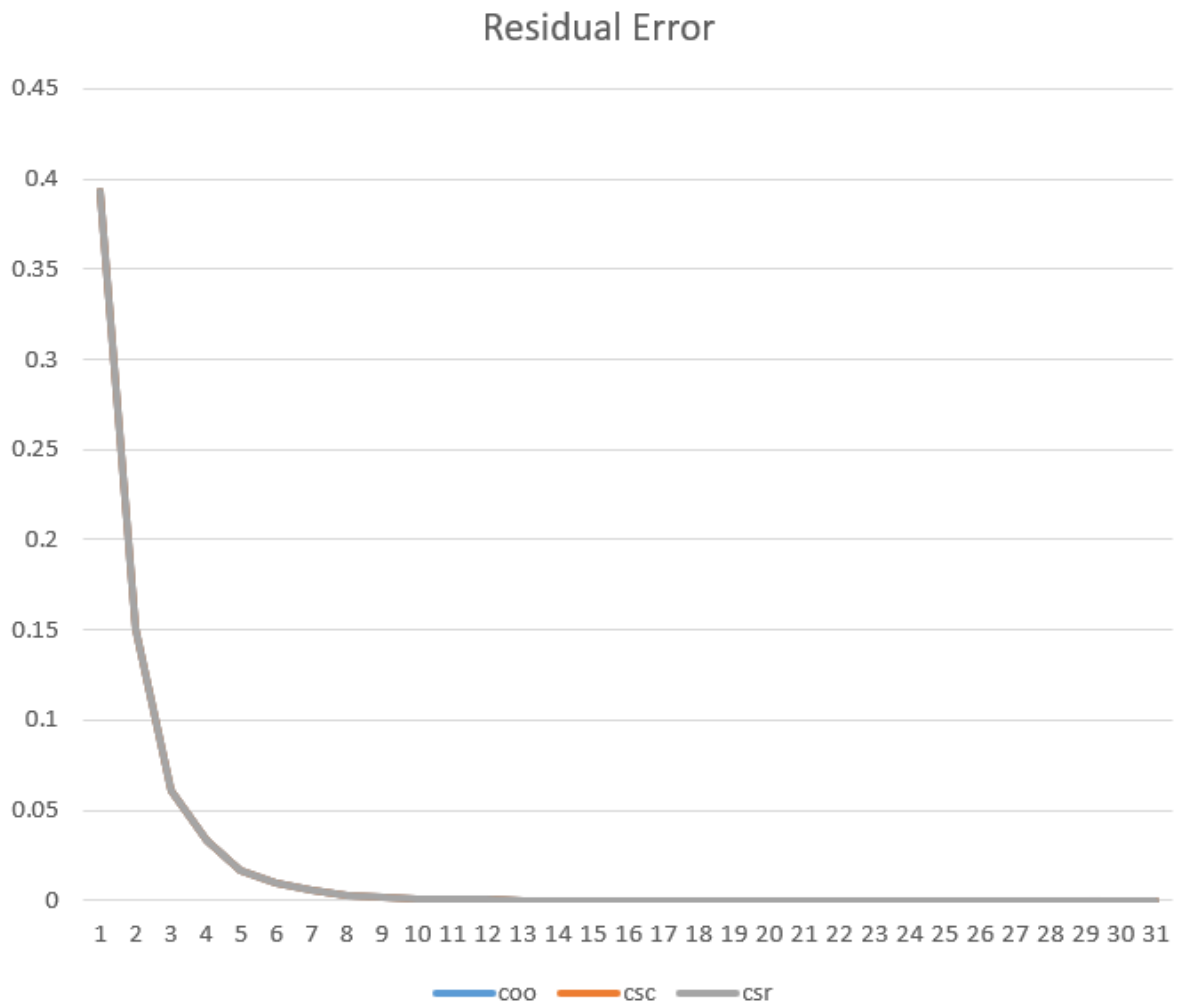
CSR:

Format: CSR
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCSR.txt
Reading input: 0.0016392000000000002 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 2.43E-5 seconds
iteration 1: residual error=0.39273487186470324 xnorm=1.0
time=0.0010069 seconds
iteration 2: residual error=0.15082251467521357 xnorm=1.0
time=6.37E-5 seconds
iteration 3: residual error=0.06098310915896904 xnorm=1.0
time=4.96000000000000006E-5 seconds
iteration 4: residual error=0.03363108589428868 xnorm=1.0
time=4.33E-5 seconds
iteration 5: residual error=0.016629581949571294 xnorm=1.0
time=4.52E-5 seconds
iteration 6: residual error=0.009559509413236817 xnorm=1.0
time=4.28000000000000004E-5 seconds
iteration 7: residual error=0.005420271938283045 xnorm=1.0
time=4.24E-5 seconds
iteration 8: residual error=0.0030897756987228217 xnorm=1.0
time=4.33E-5 seconds
iteration 9: residual error=0.0018818236426745034 xnorm=1.0
time=4.92E-5 seconds
iteration 10: residual error=0.0010947688160630946
xnorm=1.0 time=4.88E-5 seconds
iteration 11: residual error=6.901694755585205E-4 xnorm=1.0
time=4.89E-5 seconds
iteration 12: residual error=4.073253290668758E-4
xnorm=1.00000000000000002 time=4.32E-5 seconds
iteration 13: residual error=2.622110797494109E-4 xnorm=1.0
time=6.2200000000000001E-5 seconds
iteration 14: residual error=1.576443490031703E-4 xnorm=1.0
time=6.6E-5 seconds
iteration 15: residual error=1.0280682314864386E-4

xnorm=1.0 time=6.440000000000001E-5 seconds
iteration 16: residual error=6.208993245161677E-5 xnorm=1.0
time=1.078E-4 seconds
iteration 17: residual error=4.1069227137101576E-5
xnorm=1.0000000000000002 time=6.54E-5 seconds
iteration 18: residual error=2.4937456845641444E-5
xnorm=1.0 time=7.81E-5 seconds
iteration 19: residual error=1.6657078937418884E-5
xnorm=1.0 time=7.93E-5 seconds
iteration 20: residual error=1.0190979647901194E-5
xnorm=1.0 time=6.670000000000001E-5 seconds
iteration 21: residual error=6.821510338883303E-6 xnorm=1.0
time=6.450000000000001E-5 seconds
iteration 22: residual error=4.212100413835199E-6 xnorm=1.0
time=6.440000000000001E-5 seconds
iteration 23: residual error=2.8230121790076904E-6
xnorm=1.0 time=6.230000000000001E-5 seconds
iteration 24: residual error=1.749539231122102E-6 xnorm=1.0
time=6.76E-5 seconds
iteration 25: residual error=1.174459149327553E-6 xnorm=1.0
time=6.52E-5 seconds
iteration 26: residual error=7.296637554530389E-7 xnorm=1.0
time=6.610000000000001E-5 seconds
iteration 27: residual error=4.899707950912352E-7 xnorm=1.0
time=6.110000000000001E-5 seconds
iteration 28: residual error=3.057175450502103E-7 xnorm=1.0
time=6.47E-5 seconds
iteration 29: residual error=2.0493334797460488E-7
xnorm=1.0 time=6.31E-5 seconds
iteration 30: residual error=1.286083027790888E-7
xnorm=1.0000000000000002 time=6.63E-5 seconds
iteration 31: residual error=8.596268069615588E-8 xnorm=1.0
time=8.15E-5 seconds
PageRank: total time: 0.0325395 seconds
Writing file: 0.010609100000000002 seconds
All done


```
Format: CSR
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCSR.txt
Reading input: 0.001639200000000002 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 2.43E-5 seconds
iteration 1: residual error=0.39273487186470324 xnorm=1.0 time=0.0010069 seconds
iteration 2: residual error=0.15082251467521357 xnorm=1.0 time=6.37E-5 seconds
iteration 3: residual error=0.06098310915896904 xnorm=1.0 time=4.9600000000000006E-5 seconds
iteration 4: residual error=0.03363108589428868 xnorm=1.0 time=4.33E-5 seconds
iteration 5: residual error=0.016629581949571294 xnorm=1.0 time=4.52E-5 seconds
iteration 6: residual error=0.009559509413236817 xnorm=1.0 time=4.2800000000000004E-5 seconds
iteration 7: residual error=0.005420271938283045 xnorm=1.0 time=4.24E-5 seconds
iteration 8: residual error=0.0030897756987228217 xnorm=1.0 time=4.33E-5 seconds
iteration 9: residual error=0.0018818236426745034 xnorm=1.0 time=4.92E-5 seconds
iteration 10: residual error=0.0010947688160630946 xnorm=1.0 time=4.88E-5 seconds
iteration 11: residual error=6.901694755585205E-4 xnorm=1.0 time=4.89E-5 seconds
iteration 12: residual error=4.073253290668758E-4 xnorm=1.0000000000000002 time=4.32E-5 seconds
```

```
iteration 13: residual error=2.622110797494109E-4 xnorm=1.0 time=6.220000000000001E-5 seconds
iteration 14: residual error=1.576443490031703E-4 xnorm=1.0 time=6.6E-5 seconds
iteration 15: residual error=1.0280682314864386E-4 xnorm=1.0 time=6.440000000000001E-5 seconds
iteration 16: residual error=6.208993245161677E-5 xnorm=1.0 time=1.078E-4 seconds
iteration 17: residual error=4.1069227137101576E-5 xnorm=1.0000000000000002 time=6.54E-5 seconds
iteration 18: residual error=2.4937456845641444E-5 xnorm=1.0 time=7.81E-5 seconds
iteration 19: residual error=1.6657078937418884E-5 xnorm=1.0 time=7.93E-5 seconds
iteration 20: residual error=1.0190979647901194E-5 xnorm=1.0 time=6.670000000000001E-5 seconds
iteration 21: residual error=6.821510338883303E-6 xnorm=1.0 time=6.450000000000001E-5 seconds
iteration 22: residual error=4.212100413835199E-6 xnorm=1.0 time=6.440000000000001E-5 seconds
iteration 23: residual error=2.8230121790076904E-6 xnorm=1.0 time=6.230000000000001E-5 seconds
iteration 24: residual error=1.749539231122102E-6 xnorm=1.0 time=6.76E-5 seconds
iteration 25: residual error=1.174459149327553E-6 xnorm=1.0 time=6.52E-5 seconds
iteration 26: residual error=7.296637554530389E-7 xnorm=1.0 time=6.610000000000001E-5 seconds
iteration 27: residual error=4.899707950912352E-7 xnorm=1.0 time=6.110000000000001E-5 seconds
iteration 28: residual error=3.057175450502103E-7 xnorm=1.0 time=6.47E-5 seconds
iteration 29: residual error=2.0493334797460488E-7 xnorm=1.0 time=6.31E-5 seconds
iteration 30: residual error=1.286083027790888E-7 xnorm=1.0000000000000002 time=6.63E-5 seconds
iteration 31: residual error=8.596268069615588E-8 xnorm=1.0 time=8.15E-5 seconds
PageRank: total time: 0.0325395 seconds
Writing file: 0.010609100000000002 seconds
All done
```



These observations suggest that the PageRank algorithm successfully converges in different matrix formats and produces very similar PageRank values in the end.

These data also demonstrate that the PageRank algorithm continuously reduces residual errors during iterations until it converges to a small value.

- Analyse your code and describe for each array in each of the CSR, CSC and COO data structures, as well as the pagerank arrays with old and new values, which array is accessed using a sequential access pattern, and which are apparently random. Why do you think the distinction between these access patterns matters for concurrent programs?

CSR Pattern (Compressed Sparse Rows):

- The index array in CSR is likely accessed in a sequential pattern because it represents the row indices (source vertices), and these are typically processed sequentially in algorithms.

CSC Pattern (Compressed Sparse Columns):

- The index array in CSC is likely accessed sequentially, similar to CSR, because it represents the column indices (destination vertices) and is often processed sequentially in algorithms.

COO Pattern (Coordinate Format):

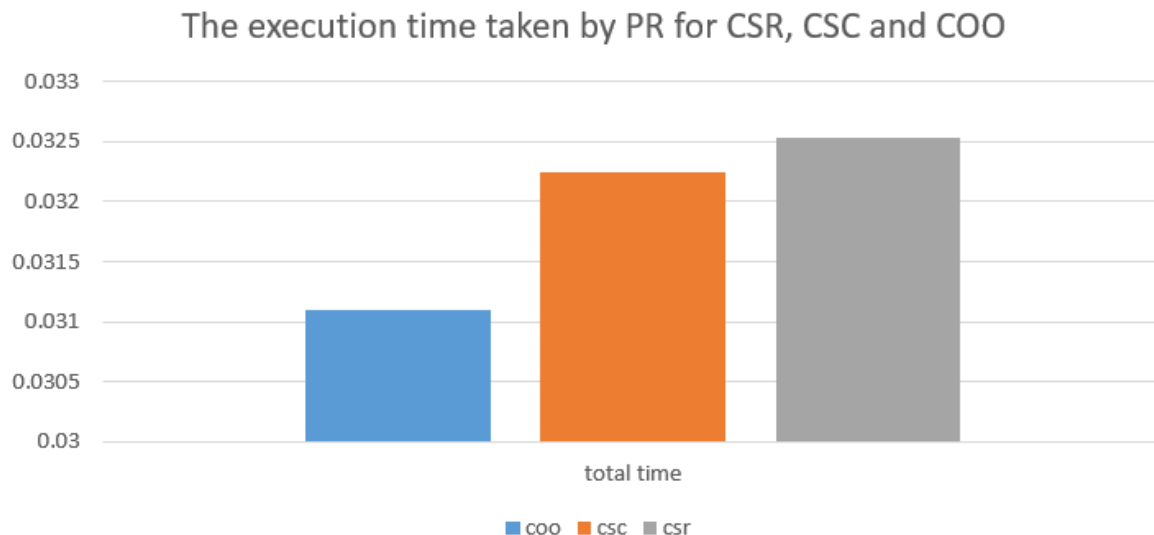
- The COO pattern is more likely to exhibit a random access pattern. COO stores each edge as a coordinate (source, destination), and accessing edges in a sequential order might not be as straightforward as in CSR or CSC.

Importance of Access Pattern Distinction for Concurrent Programs:

- **Sequential Access:** concurrent programs that exhibit sequential access patterns tend to have reduced contention for memory resources and perform better in a multi-threaded environment.
- **Random Access:** This contention can cause performance bottlenecks and reduce the scalability of concurrent programs. Therefore, understanding and optimizing access patterns is essential to avoid data access conflicts and maximize the benefits of parallel processing.

CSR	Pattern	CSC	Pattern	COO	Pattern
index	Sequential	index	Sequential	source	random
destination	random	source	random	destination	random
x	Sequential	x	Sequential	x	Sequential
y	Sequential	y	Sequential	y	Sequential

- Record the execution time taken by the PageRank algorithm for each of the three graph data structures (CSR, CSC and COO). Plot these in a chart and describe your observations. Can you explain why some formats result in lower execution time than others?



These observations suggest that COO is the most efficient data structure for the PageRank algorithm in terms of execution time, followed by CSC and CSR.

The reasons for these differences in execution time can be attributed to how each data structure stores and accesses the graph's edges:

- COO represents the graph as a list of edges, making it efficient for the PageRank algorithm to access and traverse the edges in a straightforward manner. This leads to faster execution.
- CSC stores the graph with a focus on incoming edges, which may require additional processing during the PageRank algorithm, resulting in slightly longer execution times compared to COO.
- CSR, similar to CSC, focuses on incoming edges, and this may introduce some overhead in accessing and processing edges during the PageRank algorithm, contributing to its slightly longer execution time.

- Modify your code to use single-precision floating-point numbers instead of double-precision floating-point numbers for one of the three sparse matrix formats (e.g., perform a search-replace operation in the PageRank.java and Driver.java files). Record the residual error at the end of each power iteration step and the average execution time when using single-precision vs. double-precision floating-point numbers. Report these numbers using charts. Can you explain this result?

```
Format: CSR
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCSR2.txt
Reading input: 0.0016072 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 2.52E-5 seconds
iteration 1: residual error=0.39273486 xnorm=1.0
time=9.765E-4 seconds
iteration 2: residual error=0.1508225 xnorm=1.0 time=6.77E-
5 seconds
iteration 3: residual error=0.060983114 xnorm=1.0
time=1.226E-4 seconds
iteration 4: residual error=0.0336311 xnorm=1.0 time=5.78E-
5 seconds
iteration 5: residual error=0.016629592 xnorm=1.0
time=5.14E-5 seconds
iteration 6: residual error=0.009559519 xnorm=1.0
time=5.21E-5 seconds
iteration 7: residual error=0.0054202834 xnorm=1.0
time=5.19E-5 seconds
iteration 8: residual error=0.0030897646 xnorm=1.0
time=5.21E-5 seconds
```

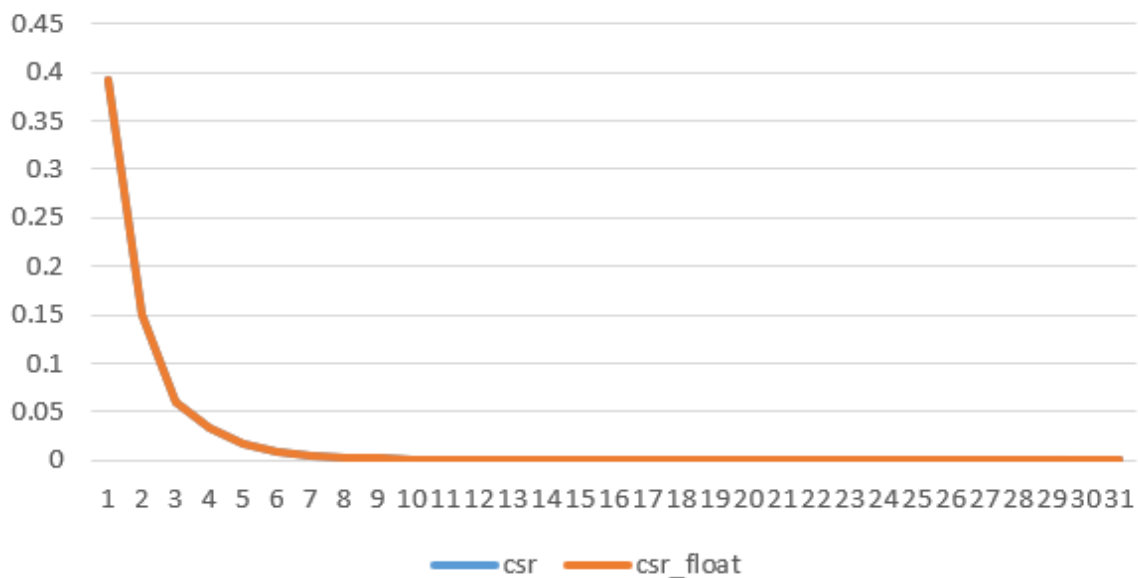
iteration 9: residual error=0.001881806 xnorm=1.0
time=5.37E-5 seconds
iteration 10: residual error=0.001094755 xnorm=1.0
time=6.66E-5 seconds
iteration 11: residual error=6.901766E-4 xnorm=1.0
time=6.75E-5 seconds
iteration 12: residual error=4.0733465E-4 xnorm=1.0
time=6.92E-5 seconds
iteration 13: residual error=2.6221178E-4 xnorm=1.0
time=6.64E-5 seconds
iteration 14: residual error=1.5763543E-4 xnorm=1.0
time=6.88E-5 seconds
iteration 15: residual error=1.0281196E-4 xnorm=1.0
time=6.6E-5 seconds
iteration 16: residual error=6.20943E-5 xnorm=1.0
time=6.89E-5 seconds
iteration 17: residual error=4.1075284E-5 xnorm=1.0
time=6.7E-5 seconds
iteration 18: residual error=2.4936162E-5 xnorm=1.0
time=6.8E-5 seconds
iteration 19: residual error=1.6654143E-5 xnorm=1.0
time=6.92E-5 seconds
iteration 20: residual error=1.0190066E-5 xnorm=1.0
time=6.95E-5 seconds
iteration 21: residual error=6.8158843E-6 xnorm=1.0
time=6.78E-5 seconds
iteration 22: residual error=4.213769E-6 xnorm=1.0
time=8.04E-5 seconds
iteration 23: residual error=2.8260984E-6 xnorm=1.0
time=6.71E-5 seconds
iteration 24: residual error=1.7439015E-6 xnorm=1.0
time=6.93E-5 seconds
iteration 25: residual error=1.1734664E-6 xnorm=1.0
time=6.64E-5 seconds
iteration 26: residual error=7.3597766E-7 xnorm=1.0
time=6.77E-5 seconds
iteration 27: residual error=4.863832E-7 xnorm=1.0
time=6.78E-5 seconds
iteration 28: residual error=3.0733645E-7 xnorm=1.0
time=6.78E-5 seconds
iteration 29: residual error=2.0861626E-7 xnorm=1.0

```
time=6.81E-5 seconds
iteration 30: residual error=1.2759119E-7 xnorm=1.0
time=6.97E-5 seconds
iteration 31: residual error=9.9185854E-8 xnorm=1.0
time=6.61E-5 seconds
PageRank: total time: 0.0272176 seconds
Writing file: 0.009733700000000001 seconds
All done
```

```
Format: CSR
Input file CSR: rMatGraph_J_5_100.csr
Input file CSC: rMatGraph_J_5_100.csc
Input file COO: rMatGraph_J_5_100.coo
Algorithm: pr
Number of threads: 1
Output file: outputfileCSR2.txt
Reading input: 0.0016072 seconds
Number of vertices: 128
Number of edges: 708
Initialisation: 2.52E-5 seconds
iteration 1: residual error=0.39273486 xnorm=1.0 time=9.765E-4 seconds
iteration 2: residual error=0.1508225 xnorm=1.0 time=6.77E-5 seconds
iteration 3: residual error=0.060983114 xnorm=1.0 time=1.226E-4 seconds
iteration 4: residual error=0.0336311 xnorm=1.0 time=5.78E-5 seconds
iteration 5: residual error=0.016629592 xnorm=1.0 time=5.14E-5 seconds
iteration 6: residual error=0.009559519 xnorm=1.0 time=5.21E-5 seconds
iteration 7: residual error=0.0054202834 xnorm=1.0 time=5.19E-5 seconds
iteration 8: residual error=0.0030897646 xnorm=1.0 time=5.21E-5 seconds
iteration 9: residual error=0.001881806 xnorm=1.0 time=5.37E-5 seconds
iteration 10: residual error=0.001094755 xnorm=1.0 time=6.66E-5 seconds
iteration 11: residual error=6.901766E-4 xnorm=1.0 time=6.75E-5 seconds
iteration 12: residual error=4.0733465E-4 xnorm=1.0 time=6.92E-5 seconds
```

```
iteration 13: residual error=2.6221178E-4 xnorm=1.0 time=6.64E-5 seconds
iteration 14: residual error=1.5763543E-4 xnorm=1.0 time=6.88E-5 seconds
iteration 15: residual error=1.0281196E-4 xnorm=1.0 time=6.6E-5 seconds
iteration 16: residual error=6.20943E-5 xnorm=1.0 time=6.89E-5 seconds
iteration 17: residual error=4.1075284E-5 xnorm=1.0 time=6.7E-5 seconds
iteration 18: residual error=2.4936162E-5 xnorm=1.0 time=6.8E-5 seconds
iteration 19: residual error=1.6654143E-5 xnorm=1.0 time=6.92E-5 seconds
iteration 20: residual error=1.0190066E-5 xnorm=1.0 time=6.95E-5 seconds
iteration 21: residual error=6.8158843E-6 xnorm=1.0 time=6.78E-5 seconds
iteration 22: residual error=4.213769E-6 xnorm=1.0 time=8.04E-5 seconds
iteration 23: residual error=2.8260984E-6 xnorm=1.0 time=6.71E-5 seconds
iteration 24: residual error=1.7439015E-6 xnorm=1.0 time=6.93E-5 seconds
iteration 25: residual error=1.1734664E-6 xnorm=1.0 time=6.64E-5 seconds
iteration 26: residual error=7.3597766E-7 xnorm=1.0 time=6.77E-5 seconds
iteration 27: residual error=4.863832E-7 xnorm=1.0 time=6.78E-5 seconds
iteration 28: residual error=3.0733645E-7 xnorm=1.0 time=6.78E-5 seconds
iteration 29: residual error=2.0861626E-7 xnorm=1.0 time=6.81E-5 seconds
iteration 30: residual error=1.2759119E-7 xnorm=1.0 time=6.97E-5 seconds
iteration 31: residual error=9.9185854E-8 xnorm=1.0 time=6.61E-5 seconds
PageRank: total time: 0.0272176 seconds
Writing file: 0.009733700000000001 seconds
All done
```

The residual error between float and double



In both DOUBLE and FLOAT data types, the values of the residual errors are very close, indicating that single-precision floating-point numbers have sufficient precision in the context of this specific problem. This allows them to operate with lower memory overhead and faster execution speed.

FLOAT uses single-precision floating-point numbers, which means each floating-point number occupies 4 bytes, while DOUBLE uses double-precision floating-point numbers, with each floating-point number occupying 8 bytes. This results in lower memory usage and faster memory access speed.

FLOAT is more susceptible to precision loss due to its smaller representation range. Nevertheless, in this particular case, the precision of single-precision floating-point numbers is sufficiently high for PageRank calculations, and it does not significantly affect the accuracy of the results.

Therefore, the results suggest that selecting the appropriate data type can significantly reduce memory usage and improve execution speed while maintaining reasonable result accuracy.