Documentation

Problem 1. *How do we parallelize the serial implementation?*

Answer.
After reading the document and the pseudocode of the Dijkstra Algorithm, we have find that there are two parts that we can parallelize.
Firstly, it is the process of calculating the cheapest edges for the vertices which have been added to the known sets.
Secondly, we can update the distance and precedes values of the vertices which haven't been added the the known sets parallelly.
Here comes our implementation:

(1) We got the array $loc\_mat, loc\_dist, loc\_pred, loc\_known$ to store the local information for the vertices that are assigned to the process n, and the array $glo\_known, gol\_dist, glo\_pred$ to store the total information and these information is always the same.

(2) Initialization:
initialize the local information parallelly, and use the function "Allgather" to update every process's global information.

(3) the main iteration:
Firstly, find the cheapest edges parallelly, and use the function "Allreduce" to get the [vertex, edge value] pair and broadcast to all processes. And update the global_known and the loc_known arrays.
Second, parallelly update the vertices' distance and precedes information, the implementation is similar with the serial one.
Thirdly, use "Allgather" to update every process's global distance and precedes information.

(4) For the process 0, output our results.

Problem 2. *Does our program work perfect?*

Answer. At the moment, it works well.
We have change the I/O program to make it read data from file, and write data in file, which is easy for us to check whether it is right.
We have wrote a python script to check whether our program works correctly.
We have wrote a simple script to run the program in bulk.
File run_result.txt is our results.

Problem 3. *What's the speed-up of it comparing to the serial one?*

Answer. Because the file reading and writing time is too long, so we don't count the file accessing time.
For 2 process:

size	Parallel time(s)	Serial time(s)	S/P
10	0.000032	0.000002	0.0625
100	0.000245	0.000109	0.4449
300	0.000746	0.000944	1.2654
400	0.001101	0.001902	1.7275
500	0.001579	0.002012	1.2742
800	0.002927	0.005354	1.8292
1000	0.004588	0.008938	1.9482
2000	0.018461	0.037647	2.0393
3000	0.041107	0.087909	2.1385
4000	0.074495	0.142097	1.9075
5000	0.100481	0.219371	2.1832

Table 1: process number: 2

We can find that with the data amounts' increasing, the speed-up is nearly 2.

size	Parallel time(s)	Serial time(s)	S/P
10	0.000028	0.000003	0.1071
100	0.000190	0.000089	0.4684
300	0.000830	0.000692	0.8337
400	0.001013	0.001646	1.6249
500	0.001610	0.001862	1.1565
800	0.002912	0.005094	1.7493
1000	0.004359	0.007980	1.8307
2000	0.016590	0.033604	2.0256
3000	0.034609	0.074535	2.1536
4000	0.065540	0.136445	2.0819
5000	0.101422	0.223791	2.2065

Table 2: process number: 5

We can find that with the data amounts' increasing, the speed-up is nearly 2.
Similarly, for 10 process, we can find that the speed-up is nearly 2.

So we have the conclusion:
1. With the problem size's incresing, the parallel program performs much better than the

serial one.
2. What limits the speed-up is not only the problem size, also the system resources. Although we have 5 process running the program, we didn't get higher speed-up, and the reason is probably that we don't have so much processes in fact, so the context for the process may limit the speed-up.