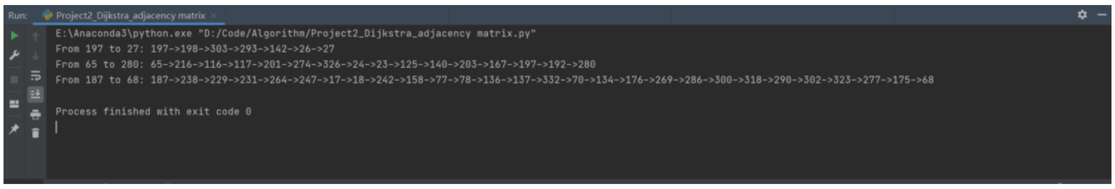
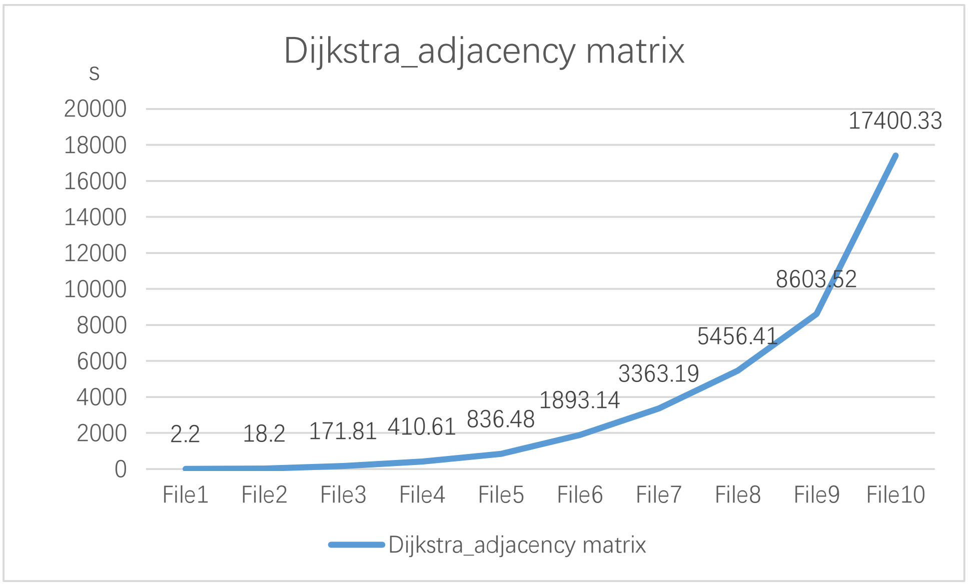
The result of Dijkstra's algorithm using adjacency matrix:



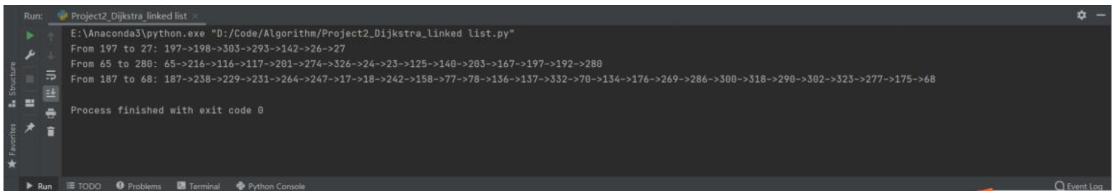
```
E:\Anaconda3\python.exe "D:/Code/Algorithm/Project2_Dijkstra_adjacency matrix.py"
From 197 to 27: 197->198->303->293->142->26->27
From 65 to 280: 65->216->116->117->201->274->326->24->23->125->140->203->167->197->192->280
From 187 to 68: 187->238->229->231->264->247->17->18->242->158->77->78->136->137->332->70->134->176->269->286->300->318->290->302->323->277->175->68

Process finished with exit code 0
```

Time performance plot:



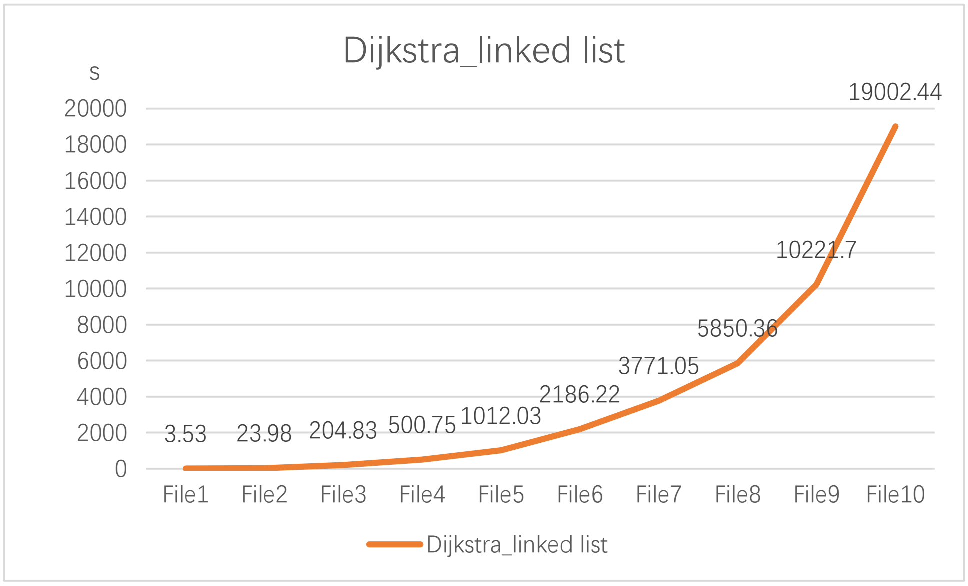
The result of Dijkstra's algorithm using linked list:



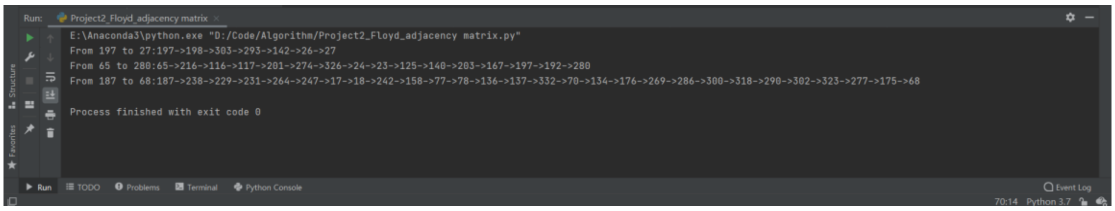
```
E:\Anaconda3\python.exe "D:/Code/Algorithm/Project2_Dijkstra_linked list.py"
From 197 to 27: 197->198->303->293->142->26->27
From 65 to 280: 65->216->116->117->201->274->326->24->23->125->140->203->167->197->192->280
From 187 to 68: 187->238->229->231->264->247->17->18->242->158->77->78->136->137->332->70->134->176->269->286->300->318->290->302->323->277->175->68

Process finished with exit code 0
```

Time performance plot:



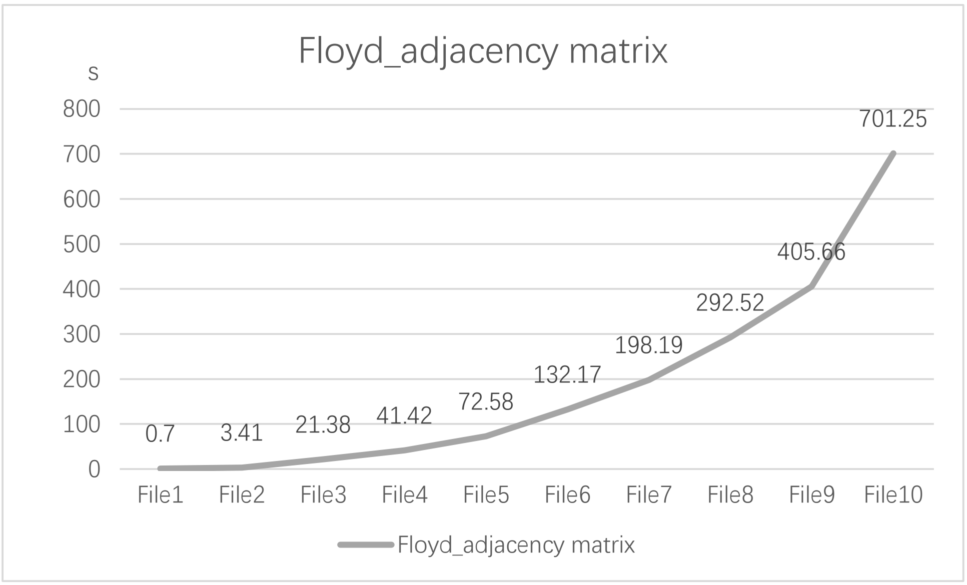
The result of Floyd's algorithm using adjacency matrix:



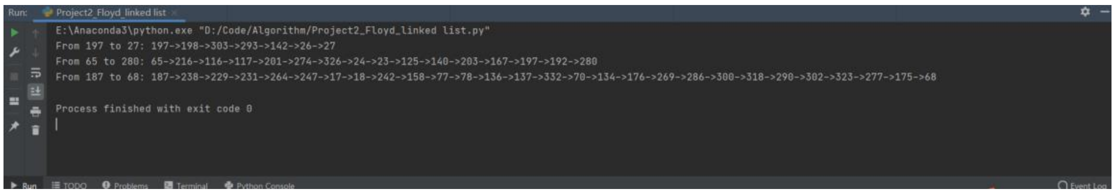
```
E:\Anaconda3\python.exe "D:/Code/Algorithm/Project2_Floyd_adjacency matrix.py"
From 197 to 27:197->198->303->293->142->26->27
From 65 to 280:65->216->116->117->201->274->326->24->23->125->140->203->167->197->192->280
From 187 to 68:187->238->229->231->264->247->17->18->242->158->77->78->136->137->332->70->134->176->269->286->300->318->290->302->323->277->175->68

Process finished with exit code 0
```

Time performance plot:



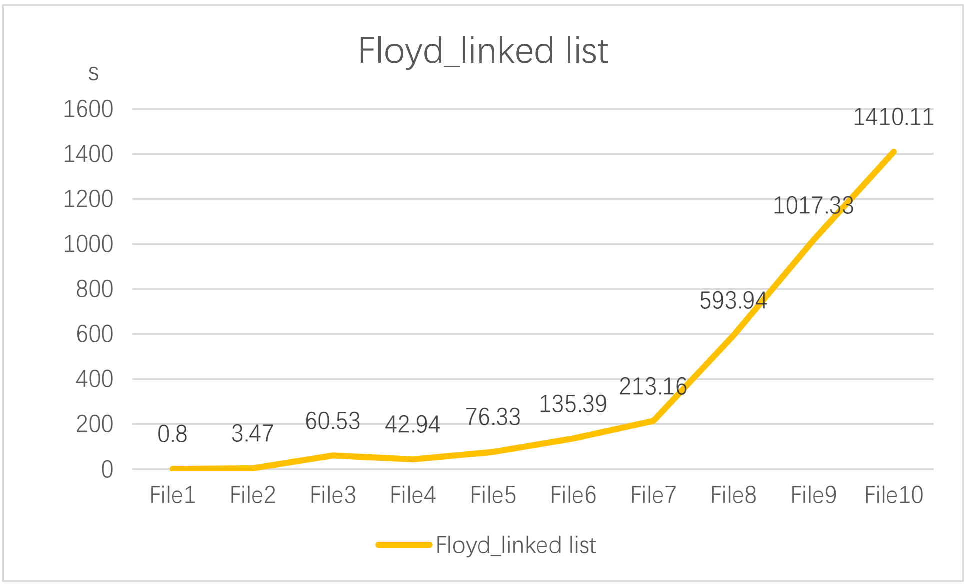
The result of Floyd's algorithm using linked list:



```
E:\Anaconda3\python.exe "D:/Code/Algorithm/Project2_Floyd_linked list.py"
From 197 to 27: 197->198->303->293->142->26->27
From 65 to 280: 65->216->116->117->201->274->326->24->23->125->140->203->167->197->192->280
From 187 to 68: 187->238->229->231->264->247->17->18->242->158->77->78->136->137->332->70->134->176->269->286->300->318->290->302->323->277->175->68

Process finished with exit code 0
```

Time performance plot:



Memory usage discussion:

I use a summary statistics provided by Python. For the Input_File1 and Input_File10, the total memory used for these two algorithms are:

Dijkstra's algorithm using adjacency matrix, file1: 62.67 MB, file10: 78.71 MB

Dijkstra's algorithm using linked list, file1: 62.70 MB, file10: 63.60 MB

Floyd's algorithm using adjacency matrix, file1: 62.81 MB, file10: 154.53 MB

Floyd's algorithm using linked list, file1: 63.04 MB, file10: 80.94 MB

Analysis:

For both Dijkstra and Floyd algorithm using adjacency matrix, the space usage is O(V^2). Because no matter the number of edges in the graph, we create a 2-dimensional array which size is V*V and may be very sparse. The difference is that in Floyd algorithm we can compute all-pair shortest paths, but in Dijkstra algorithm we can only get the shortest path from start node to all other nodes. Thus we have to repeat Dijkstra algorithm for V times to get all-pair shortest paths.

For both Dijkstra's algorithm using linked list, the space usage is O(V). Because we only need store the number of V head pointers in an array, then we can iterate every edge of the graph. However, although linked list used in Floyd's algorithm only needs O(V) space, we still need create a 2-dimential V*V matrix to do dynamic programming.

To conclusion, Floyd's algorithm is much much faster than Dijkstra's algorithm in practice. Because it can compute all-pair shortest paths after running it once while Dijkstra's algorithm have to be repeated for different start nodes. However, less time means more space, both of Floyd's algorithms use more memory than Dijkstra's algorithm. What's more, for two different methods in each algorithm, linked list uses less space than adjacency matrix apparently.