

The result of minimum-spanning tree using adjacency matrix:

```
Run: Project3_Kruskal_adjacency matrix x
864 1004 642
1555 1972 642
1691 1983 643
1908 1992 650
1855 1990 652
1463 1464 666
541 1923 669
87 516 693
20 513 733
656 1937 733
1696 1824 740
1749 1987 742
1001 1890 774
706 1004 779
865 866 780
1637 1908 780
83 90 842
565 1927 999
1674 1982 1013
798 1943 1031
1347 1967 1032
1593 1927 1178
1091 1957 1205
22 96 1208
1395 1929 1252
83 487 1395
1890 1991 3703
The total distance of the minimum-spanning tree is: 371466
Memory use: 95.76 MB
Process finished with exit code 0
```

The result of minimum-spanning tree using linked list:

```
Run: Project3_Kruskal_linked list x
87 516 693
20 513 733
656 1937 733
1696 1824 740
1749 1987 742
1001 1890 774
706 1004 779
865 866 780
1637 1908 780
83 90 842
565 1927 999
1674 1982 1013
798 1943 1031
1347 1967 1032
1593 1927 1178
1091 1957 1205
22 96 1208
1395 1929 1252
83 487 1395
1890 1991 3703
The total distance of the minimum-spanning tree is: 371466
Memory use: 65.45 MB
Process finished with exit code 0
```

The analysis of the memory usage:

As the result shows, the memory usage of adjacency matrix is 95.76MB and the usage of linked list is 65.45MB. Because for a Graph (V, E), the space complexity using two-dimensional matrix is $O(V*V)$, the space complexity using linked list is $O(V)$. So we can see clearly using linked list use much less memory than using adjacency matrix, especially when the number of nodes is huge and graph is sparse. The linked list can only store existing edges, but adjacency list has to store $V*V$ edges even if there is no edge between two nodes.