Problem: **Graph coloring**
Given a graph, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using same color.

The algorithms used in this evaluation:
1. Tabu Search
2. Simulated annealing
3. Hybrid TS-SA: keep the strategies of SA; moreover, using 1 tabu list to store the solution which has already used (same as close list) to be able to make the algorithm not come to the same solution again, and have a better fitness function value in the end

The fitness function F here will be calculate by the number of edges N (having 2 vertices with different colors) + number of edges(if N = total number of edges in the graph)(*) - number of color using to color the graph(**).
This fitness function having (*) part is to avoid the “bad” case. If not,
For example:
The graph has 40 edges, and to color it requires at least 5 colors.
+ Case 1: one solution has 39 edges satisfies the requirement, needing 4 different colors, but the last edge is not, it has the same color for both sides. The fitness function value of this solution is:
  F = 39 - 4 = 35
+ Case 2: one solution has all edges satisfies the requirement, needing 5 different colors. The fitness function value of this solution is:
  F = 40 - 5 = 35

Two cases has the same fitness function but we know that the second case is better one
The (**) part is to choose the solution has the better performance, which needs less color to color all graph satisfying the requirement.

Therefore, the higher the value of the fitness function is, the better the algorithm is.
The function to calculate the memory usage here is to calculate for all the stack, list and processes used in the algorithm

Running the code in C++ with the graph having 23 vertices (from 0 to 22) and 71 egdes.
The adjacent list of the graph is as below:

```
[0]-> 1 3 6 8 12 14 17 19
[1]-> 2 5 7 11 13 16 18
[2]-> 4 6 9 12 15 17 20
[3]-> 4 5 9 11 15 16 20
[4]-> 7 8 13 14 18 19
[5]-> 10 12 14 21
[6]-> 10 11 13 21
[7]-> 10 12 15 21
[8]-> 10 11 15 21
[9]-> 10 13 14 21
[10]-> 16 17 18 19 20
[11]-> 22
[12]-> 22
[13]-> 22
[14]-> 22
[15]-> 22
[16]-> 22
[17]-> 22
[18]-> 22
[19]-> 22
[20]-> 22
[21]-> 22
[22]->
```

+) Memory usage (KB)

	TS	SA
1	790528	744144
2	789784	737280
3	798788	737280
4	790034	737280
5	787729	744144
6	793124	74414
7	790528	737280
8	790528	737280
9	798788	737280
10	789844	737280

+) Fitness function (the number of colors used)

	TS	SA	Hybrid TS-SA
1	87(7)	82(12)	84(10)
2	88(6)	80(14)	82(12)
3	89(5)	84(10)	87(7)
4	89(5)	87(7)	87(7)
5	87(7)	83(11)	84(10)
6	88(6)	84(10)	83(11)
7	88(6)	80(14)	86(8)
8	87(7)	87(7)	86(8)
9	89(5)	82(12)	84(10)
10	89(5)	86(8)	86(8)

+) Time consuming (milisecond)

	TS	SA	Hybrid TS-SA
1	801.84	294.56	353.23

2	791.56	300.18	310.15
3	791.33	303.15	350.18
4	800.14	291.59	354.6
5	783.29	297.8	353.79
6	803.12	301.12	277.28
7	790.18	303.54	359.13
8	795.16	297.13	331.97
9	790.1	288.9	351.66
10	785.12	303.15	324.9

From the results of the algorithms, we can see that, the SA runs with less memory usage than the TS, however, since the last result mainly depends on the probability p when choosing the next solution, so it does not guarantee the best at the end; meanwhile, TS chooses the nest solution as the better one in the neighborhood, so the fitness value of the last solution is better than searching by SA. My proposed hybrid TS-SA gives better results in the end than the SA, but not as good as TS.
In the future, I will find other way to get the better algorithms, which balances the memory usage and the quality of the result.