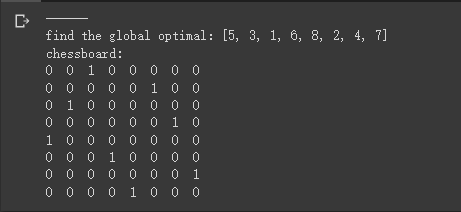
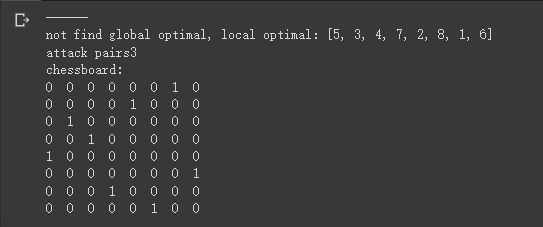
Project 1 report

1. The first search algorithm I choose is the hill climbing. The specific logic and algorithm of hill climbing will not be introduced here. Mainly talk about the composition of the code. Since there are too many sequences to be searched, we first perform an optimization on the searched objects before applying the search algorithm. So, I wrote a small program here to filter out the sequences of “only one queen in each row and column”. There are 8\*7\*6\*5\*4\*3\*2=40320 such sequences. It can greatly reduce the running time of subsequent programs, and in this way, when processing each sequence later, it is only necessary to consider whether there are other queens on the two diagonal lines and the line where they are located.

Then two functions are used to calculate the logarithm of the attacking queens of the sequence corresponding to the chessboard and print out the chessboard corresponding to the sequence. When calculating the logarithm n of the queens attacking each other on the chessboard corresponding to the sequence, 0<=n<=28, and the optimal solution should satisfy n=0. Therefore, it is only necessary to check whether the eight queens of the current chessboard have other queens in their respective rows and two diagonals, and there is no need to judge whether there are other queens in the same column.

Finally, enter the main program and load the previously deleted sequence. Two functions are called to complete the hill-climbing search. The only thing to note here is that only when the number of pairs of queens attacking each other is 0 is the global optimal solution, which is also the biggest problem of mountain climbing.



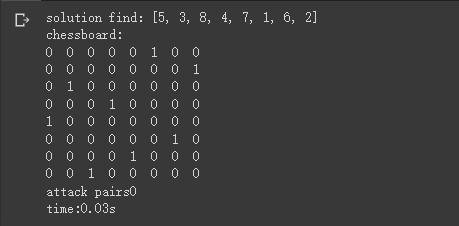


As I said earlier, two results are also shown here, the first one finds a global optimum, and the second one finds a local optimum. Usually, hill climbing can find the solution to the problem very quickly, because it is generally easy to start from a poor state and expand. However, the hill-climbing method often falls into a local optimum and it is difficult to extricate itself, that is to say, it is possible to reach such a state during the execution of the algorithm - in this state, no better improvement can be made. For example, in solving the eight-queen problem, start with a randomly generated chessboard with eight queens on it. Using the steepest climbing method has a high probability of falling into a local optimum, and it can only be solved in 10% of cases. question. The process of mountain climbing is relatively fast. In solving the Eight Queens Problem, it takes only four steps to successfully obtain the solution on average, but in the same way, it may fall into a local optimum in the third step.

1. The second search method I will choose the DFS search method here.

Similar to the previous method, we also set two functions to calculate the logarithm of the attacking queens of the sequence corresponding to the chessboard and print out the chessboard corresponding to the sequence.

Then go to the main function part, here we need to always expand the deepest node. DFS uses the LIFO queue, that is, the stack is used. And the goal test is done when the node is generated. We're going to use the stack to store the unextend leaf nodes; the initial state is 8 zeros, meaning there are no queens on the board. Then enter the loop, first place the queen randomly. After the queen is placed in the temp row of the column, if the sequence corresponding to the chessboard has no queens attacking each other, the sequence is stored in the frontier set. Finally, do the goal test when generating the node. If there is no 0 element in the sequence, that is, the eight queens have been placed, the sequence is the solution sequence.

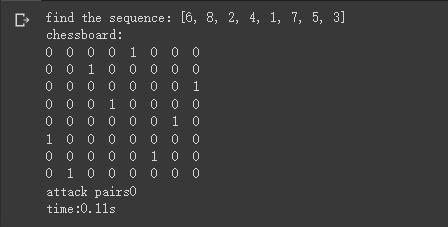


The above picture shows the running result. In order to have a parameter to reflect the performance of the algorithm, I specially added the running time as a reference. DFS I think is really a great algorithm because DFS expands the node that is newest to the frontier set first, so it quickly puts eight queens in place. The running speed is also very fast.

1. The third algorithm I will use the A\* search algorithm introduced in class.

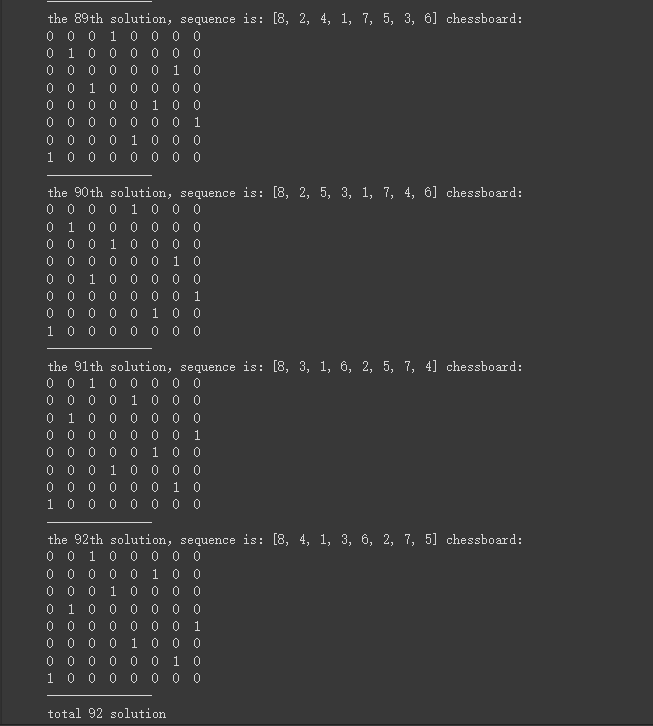
Still use the calculation sequence in the previous algorithm to correspond to the chessboard's attacking queen logarithm and the chessboard corresponding to the printout sequence, and then adjust the search logic in the main program.

In the A\* algorithm, we need to use the definition of the algorithm f(n)=g(n)+h(n), and do a goal test before expanding the node. where g(n) = the number of queens that are not placed, and h(n) = the number of pairs of queens that attack each other. It is still necessary to set the loop function. If the frontier is not empty, the loop will continue. If the solution is successfully found, it will jump out of the loop and output the solution. If the frontier is empty and the solution has not been found, it will fail. Then do the goal test before expanding the node. If the sequence h(n)=g(n)=0, the sequence is the solution sequence.



The results of the A\* algorithm is also shown here, along with the running time. In fact, I think that these algorithms with goal tests in search are all excellent, and it is difficult to say which algorithm is better. But I stumbled across a very long test result in several attempts. I think it is because f(n)=g(n)+h(n) in the A\* algorithm, the frontier set is sorted from small to large by f(n), so simple addition will sometimes result in a large number of "8 queens have been put well, that is, g(n)=0, but the number of pairs of queens attacking each other is close to 0, but not 0, that is, the node with h(n)!=0” such situation is placed in the first place or in front, and such nodes are definitely not meets the requirements. However, such a node cannot be expanded, because the 8 queens have been placed, and the node can only be skipped to judge the next node. This situation results in sometimes requiring a large amount of run time or unstable run time. So, A\* will become an unstable algorithm in some specific cases.

1. The last method is the most classic enumeration method. This 8 queen question is a very classic question in the interview questions of the brushing website and many companies. The enumeration method has also been improved to the current recursive backtracking method. The basic idea is: the first row occupies a queen first. The second row occupies one more and cannot attack the first queen. The third row occupies one more, and so on until the nth row occupies one. When the nth row cannot stand, cancel the queen of the n-1 row, and re-occupy a queen position in the next position of the n-1th queen, until it occupies the last position of the n-1th row. When it still doesn't work, cancel the n-2 line, when the queen of the n-2 line is in the last position of the n-2 line, cancel n-3, n-2 is in the last position, then n- 3 lines must no longer be the last position. Then re-find the position of the queen in row n-2. Repeat this cycle. I attach the result below.



This method has the largest time and algorithm complexity, but the only advantage is that it can enumerate all the possibilities that meet the conditions.