1. 8-queen

1(a). Average num of attacking pairs

IDS: 0

Hill-Climbing: 1/30 = 0.03333333333333333

Genetic Algorithm: 0

1(b). Average running time

IDS: 1474.52 sec

Hill-Climbing: 0.050899569193522134 sec/run

Genetic Algorithm: 54.434397260348 sec/run

1(c). Success rate

Hill-Climbing: 29/30 = 0.9667 = 96.67%

Genetic Algorithm: 30/30 = 1 = 100%

2. 50-queen

2(a). Average num of attacking pairs

IDS: timeout

Hill-Climbing: 0

Genetic Algorithm: 16.566666666666666

2(b). Average running time

IDS: timeout

Hill-Climbing: 101.93729825814565 sec/run

Genetic Algorithm: 1176.4266528685887 sec/run

2(c). Success rate

IDS: timeout

Hill-Climbing: 30/30 = 1 = 100%

Genetic Algorithm: 0/30 = 0 = 0%

實作細節:

(1) IDS

```
int attacking_pairs(vector<int>& state){
    int cnt=0;
    for(int i=0;i<N;i++){
        for(int j=i+1;j<N;j++){
            if(state[i]==state[j] || abs(state[i]-state[j])==abs(i-j)){
                cnt++;
            }
        }
    }
    return cnt;
}
```

上圖的 attacking_pairs()函式計算盤面上有幾組皇后會互相攻擊。

DFS fringe 使用 stack，stack 裡儲存的資料結構為 pair<vector<int>, int>，分別代表棋盤狀態與該狀態在 search tree 裡的深度。其中，棋盤狀態為一長度為 N(皇后數)的陣列，例如棋盤狀態 state=[3,4,2,5,6,1,7,0]，則 state[0]表示位於 0-th row 的那隻皇后，它所在的 column index = state[0] = 3rd 上；棋盤上的 (row, col) = (1, 4) & (2, 2)位置上分別有皇后…依此類推。

同時，需維護一個 visited(型別為 set<vector<int>>)來記錄哪些 state 已經被走訪過，避免重複走訪。

在 generate neighbor 的時候須注意該 child 的深度有沒有超過 depth limit，如果沒有就可以將該 child 放入 stack 跟 visited。

(2) Hill Climbing

```
#count num of attacking pairs given state[]
def heuristic(board, state):
    cnt=0
    for i in range(N):
        for j in range(i+1,N):
            if i!=j:
                rowi=state[i];coli=i
                rowj=state[j];colj=j
                if rowi==rowj or coli==colj or abs(rowi-rowj)==abs(coli-colj):
                    cnt+=1
    return cnt
```

Evaluation function 同 IDS，計算盤面上有幾對皇后會互相攻擊。但棋盤狀態的 representation 跟 IDS 有稍稍不同，同樣都是一長度為 N(皇后數)的陣列，例如棋盤狀態 state=[3,4,2,5,6,1,7,0]，但 state[0]表示的是位於 0-th col 的那隻

皇后，它所在的 row index = state[0] = 3rd 上；棋盤上的(row, col) = (4, 1) & (2, 2)位置上分別有皇后…依此類推。

在 generate neighbor 時，會去走訪所有可能的 neighbors 並挑出"min # of attacks"的 state 當作下一輪的出發點。
在 Hill Climbing 時，起點(初始盤面)都是隨機產生，並不斷地往最好的 neighbor 前進。1)如果目前的 state 就是 best neighbor，代表我們已經走到了 optimum，就可以結束；2)如果目前的 state 不是 best neighbor 而且這兩個 state 的 evaluation value 一樣，代表我們現在卡在 plateau, ridge, or shoulder，此時會挑一個隨機的 column 並將該 column 上的皇后移到隨機的 row 上，跳離 local optimum。

(3) Genetic Algorithm

Fitness function 定義為"# of non-attacking pairs"，計算方法=
最大可能的 attacking pairs(= C(N, 2)) - (水平 attacking + 斜線 attacking)。GA 會一直 iterate 直到它找到一組最佳解或是 generation 大於等於 10000 為止。
在 crossover 的時候會產生兩個 child，並比較他們的 fitness，把較佳 fitness 的 child 放進新的 population 中，以作為下一次的 generation。

GA type: generational
Representation: Permutation
Population size: 100
Parent selection: fitness-proportionate (Roulette wheel)
Crossover: 1-point
Crossover rate: 1
Mutation: apply once to the whole string(randomly choose an index and assign a valid random number to the value at that index)
Mutation rate: 0.1
Survivor selection: age-based
Termination: finds an optimal answer or generation >= 10000
Runs: 30