You have a car.
A car is made of an engine, a transmission, a tire, and a roof.
Engine choice: *engines.txt* file
Tire choices: *tires.txt* file
Transmission choices: *transmissions.txt* file
Roof choices: Sunroof, Moonroof, No roof

You have a car like **START STATE** (EFI,Danlop,AT,Noroof)
You want to have a car like **GOAL STATE** (V12,Pirelli,CVT,Sunroof)
Each year you can change only 1 component of the car which results in a way that makes a valid car model i.e **You can't change your component in a way which makes your car invalid at any point of time**. The valid car models are given in the valid_book.csv file.
You want to have your dream car as soon as possible i.e Minimize the years after which you get your desired car..

How to calculate the cost between two states: Number of mismatched components.
Example: **Δ**E = 3 for (EFI,Danlop,AT,Noroof) and (EFI,Danlop,AT,Sunroof) as only the roof state mismatches.

If at any point, you arrive at an invalid state, you can't go any further from that one, just discard that state.

Probability function for simulated annealing: e^(ΔE/t) **[Note: this function is required only when ΔE is negative]**
Here, ΔE = next_node.val - current_node.val
Y = year passed aka BFS level
t = 1 / Y

Probability modeling:

```
random.uniform(0, 1) <=  e
```

Simulated annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

- Memory requirement O(1)
- If you use heuristic cost instead of heuristic value, then ΔE should be current.cost - next.cost.