Assignment description: There are 3 jars: 12 liters, 8 liters and 5 liters. The first jar is full and the rest are empty. The goal is to get 6 liters using only the given jars.

Solution:

Before trying to use different algorithms and applying them to the task, let's rewrite the statements in Prolog terms:

jar(Id, MaximumCapacity, CurrentCapacity) – term for describing a jar;

state of the system is described as a list of jars:

E.g., [jar(1, 12, 12), jar(2, 8, 0), jar(3, 5, 0)].

Now let's describe a query:

%% solve (InitState, Goal, Steps). :- solve ([jar(1, 12, 12), jar(2, 8, 0), jar(3, 5, 0)], jar(_, _, 6), Steps).

Goal = jar(_, _, 6) means that we don't care about which jar will contain 6 liters of water.

So, the code below solves the problem:

```
write (Steps).
replace (S, [S|A], Z, [Z|A]).
replace (S, [B|A], Z, [B|As]) :- replace(S, A, Z, As).
%% the strategy is simply the final states after every step
%% having initial and final states it is not difficult to find the intermediate steps
solve (State, Goal, _, [State]) :- member (Goal, State).
Solve (State, Goal, History, [State|Steps]) :-
member (jar, State), member (Jar2, State), not(jar = Jar2),
move (jar, Jar2, ResJar, ResJar2),
replace (jar, State, ResJar, State2),
replace (Jar2, State2, ResJar2, StateX),
```

```
%%% final state checking
not (member(StateX, [State|History])),
solve (StateX, Goal, [State|History], Steps).
%% move(jar(_Id, Max, Cur), jar(_Id2, Max2, Cur2), ...,...).
move (jar(Id1, Max1, Cur), jar(Id2, Max2, Cur2),
jar (Id1, Max1, 0), jar (Id2, Max2, Cur3)) :-
Cur > 0, Cur3 is Cur2 + Cur, Cur3 =< Max2.
move (jar(Id1, Max1, Cur), jar (Id2, Max2, Cur2),
jar (Id1, Max1, Cur3), jar (Id2, Max2, Max2)) :-
Cur > 0, Cur3 is Cur2 + Cur - Max2, Cur3 >= 0.
```