Name: Nguyen Huu Khang

ID: 18125086

REPORT LAB 02:

Task	Completed
Manipulate the input and output	**YES**
Implement the PL-Resolution	**YES**
Implement the David-Putnam algorithm	**YES**
Provide valid results for the PL-Resolution	**YES**
Provide valid results for the David-Putnam algorithm	**YES**
Report sufficient information in the document	**YES**

I. Preprocess input data:

The input KB contains strings which hard for me to handle with.

Therefore, I take the idea from pysat that convert KB into numeric arrays in which the numeric value is the value in ascii of each character ('A' = 65,…)

For examle:

Input:

```
['-A OR B\n', 'B OR -C\n', 'A OR -B OR C\n', '-B', '-A\n']
```

After converted:

```
[[-65, 66], [66, -67], [65, -66, 67], [-66], [65]]
```

When we need to print KB out, we just need to convert it back to char.

II. PL_resolution:

1. Algorithm:

I use the pseudo code represented in lecture 7 as a reference to implement this algorithm.

```
function PL-RESOLUTION(KB,α) returns true or false
  inputs: KB, the knowledge base, a sentence in propositional logic
          α, the query, a sentence in propositional logic
  clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
  new ← { }
  loop do
    for each pair of clauses C_i, C_j in clauses do
      resolvents ← PL-RESOLVE(C_i, C_j)
      if resolvents contains the empty clause then return true
      new ← new ∪ resolvents
    if new ⊆ clauses then return false
    clauses ← clauses ∪ new
```
53

(Source: p53 - Slide 7)

2. Implementation:

The implementation of this algorithm consists of 3 files:

+ main.py: main program to run the algorithm and test

+ pl_resolution.py: implementation of the algorithm and some helper functions (check_is_existed, pl_resolve)

+ utils.py: helper functions to convert input into numeric format to handle easier and print output.

**Note:

- When the number of clauses in KB is large, the algorithm produces many new clauses but most of them are useless, so I tried to improve it by adding condition:

len(clause_1) + len(clause_2) – Len of new clause >= min(len(clause_1), len(clause_2))

- After each pair resolved, I sort the new_clause by abs.

3. **Test cases:**

a, Test 1:

Input:

```
-A
4
-A OR B
B OR -C
A OR -B OR C
-B
```

Output:

```
4
C
-A
B
-C
4
-B OR C
A OR C
A OR -B
{}
YES
Time runs:  0.005214214324951172  (second)
```

Time runs: 0,005(sec)

b, Test 2:

Input:

```
A
4
-A OR B
B OR -C
A OR B OR C
B
```

Output:

```
2
B OR C
A OR B
0
NO
Time runs:  0.002770662307739258  (second)
```

Time runs: 0.0028(sec)

c, Test 3:

Input:

```
-A
11
-A OR B OR C
D OR E OR F
-B OR -D
-E OR G
G OR H
A OR -H
B OR -G
G OR K
-C OR H
-K
-B
```

Output:

```
15
-A OR C OR -D
B OR C OR -H
-A OR B OR H
-A OR C
B OR C
-B OR E OR F
D OR F OR G
-D OR -G
B OR -E
A OR G
B OR H
A OR -C
B OR K
-G
G
25
B OR C OR G
B
E OR F OR -G
B OR D OR F
C OR -D OR -H
-A OR -D OR H
C OR -D
-B OR F OR G
-D OR -E
-D OR H
-D OR K
-E
H
C OR -H
A OR B
K
-A OR H
C
C OR -D OR G
-D
```

```
A OR B OR -H
B OR G OR H
B OR -C OR H
C OR G
{}
YES
Time runs:  0.06944704055786133  (second)
```

Time runs: 0.07(sec)

d, Test4:

Input:

```
-A
20
-A OR B OR C
D OR E OR F
-B OR -D
-E OR G
G OR H
A OR -H
B OR -G
G OR K
-C OR H
-H OR I OR -J
J OR L
-F OR L
M OR N
N OR P
-G OR -N
N OR Q
B OR Q
-Q
-K
-B
```

Output: (because it’s too long so I only present get the final result here)

```
{}
YES
Time runs:  0.1808927059173584  (second)
```

Time runs: 0.18(sec)

e, Test 5:

Input:

```
A
20
-A OR B OR C
D OR E OR F
B OR -D
-E OR G
G OR H
A OR H
B OR -G
G OR K
-C OR H
H OR I OR -J
J OR L
-F OR L
M OR N
N OR P
-G OR N
N OR Q
B OR Q
Q
K
B
```

Output:

```
13
B OR C OR H
-A OR B OR H
B OR E OR F
D OR F OR G
D OR E OR L
B OR -E
-E OR N
B OR H
H OR N
H
B OR K
K OR N
H OR I OR L
8
B OR D OR F
D OR F OR N
B OR F OR G
B OR E OR L
D OR G OR L
B OR F OR N
B OR D OR L
D OR L OR N
2
B OR G OR L
B OR L OR N
0
NO
Time runs:  0.025098800659179688
```

Time runs: 0.025 (sec)

III, Davis-Putnam algorithm:

1. Algorithm:

I use the pseudo code represented in lecture 7 as a reference to implement this algorithm.

function DP(Δ)

  for φ **in** *vocabulary* (Δ) **do**

    var $\Delta' \leftarrow \{\,\}$;

    for Φ_1 **in** Δ **for** Φ_2 **in** Δ such that $\varphi \in \Phi_1$ $\neg\varphi \in \Phi_2$ **do**

      var $\Phi' \leftarrow \Phi_1 - \{\varphi\} \cup \Phi_2 - \{\neg\varphi\}$;

      if not *tautology*(Φ') **then** $\Delta' \leftarrow \Delta' \cup (\Phi')$;

    $\Delta \leftarrow \Delta - \{\Phi \in \Delta \mid \varphi \in \Phi \text{ or } \neg\varphi \in \Phi\} \cup \Delta'$;

  return {**if** $\{\,\} \in \Delta$ **then** *unsatisfiable* **else** *satisfiable;*

function *tautology*(Φ)

  $\varphi \in \Phi$ and $\neg\varphi \in \Phi$

92

(Source: p92- Slide 7)

2. Implementation:

The implementation of this algorithm consists of 3 files:

+ main.py: main program to run the algorithm and test

+ davis_putnam.py: implementation of the algorithm and some helper functions (join_2_clauses, tautology, update_clauses, compare, extend_clauses)

+ utils.py: helper functions to convert input into numeric format to handle easier.

3. Test cases:

a, Test 1:

Input:

```
-A
4
-A OR B
B OR -C
A OR -B OR C
-B
```

Ouput:

```
4
B
B OR -C
-B OR C
-B
4
{}
-C
C
{}
YES
Time runs:  0.04086470603942871  (second)
```

Time runs: 0.0409(sec)

b, Test 2:

Input:

```
A
4
-A OR B
B OR -C
A OR B OR C
B
```

Output:

```
3
B OR -C
B OR C
B
NO
Time runs:  0.017992258071899414  (second)
```

Time rus: 0.018(sec)

c, Test 3:

Input:

```
-A
11
-A OR B OR C
D OR E OR F
-B OR -D
-E OR G
G OR H
A OR -H
B OR -G
G OR K
-C OR H
-K
-B
```

Output:

```
12
B OR C
D OR E OR F
-B OR -D
-E OR G
G OR H
-H
B OR -G
G OR K
-C OR H
-K
-B
-H OR B OR C
12
C
D OR E OR F
-D
-E OR G
G OR H
-H
-G
G OR K
-C OR H
-K
{}
C OR -D
YES
Time runs:  0.031978845596313348  (second)
```

Time runs: 0.032(sec)

d, Test4:

Input:

```
-A
20
-A OR B OR C
D OR E OR F
-B OR -D
-E OR G
G OR H
A OR -H
B OR -G
G OR K
-C OR H
-H OR I OR -J
J OR L
-F OR L
M OR N
N OR P
-G OR -N
N OR Q
B OR Q
-Q
-K
-B
```

Output:

```
21
B OR C
D OR E OR F
-B OR -D
-E OR G
G OR H
-H
B OR -G
G OR K
-C OR H
-H OR I OR -J
J OR L
-F OR L
M OR N
N OR P
-G OR -N
N OR Q
B OR Q
-Q
-K
-B
-H OR B OR C
20
C
D OR E OR F
-D
-E OR G
G OR H
-H
-G
G OR K
-C OR H
-H OR I OR -J
J OR L
-F OR L
M OR N
```

```
N OR P
-G OR -N
N OR Q
-Q
-K
{}
C OR -D
YES
Time runs:  0.07096290588378906  (second)
```

Time runs: 0.07(sec)

e, Test 5:

Input:

```
A
20
-A OR B OR C
D OR E OR F
B OR -D
-E OR G
G OR H
A OR H
B OR -G
G OR K
-C OR H
H OR I OR -J
J OR L
-F OR L
M OR N
N OR P
-G OR N
N OR Q
B OR Q
Q
K
B
```

Ouput: (The output is very long so I only present the final result)

```
resolved by symbol  81
NO
Time runs:  0.12198209762573242  (second)
```

Time runs: 0.122(sec)

IV. Compare between two algorithms:

Test case	KB length	PL_resolution (sec)	Davis_Putnam (sec)	Answer
1	4	0,005	0.0409	Yes
2	4	0.0028	0.018	No
3	11	0.07	0.032	Yes
4	20	0.18	0.07	Yes
5	20	0.025	0.122	No

PL_resolution works better if KB length is small and in some cases when the answer is NO.

Davis_Putnam runtime may vary a little bit due to which symbol to resolve but after all it works quite good in case KB length is big.

Complexity:

- The deduction of PL_resolution require exponential times and for large KB it's take so long to find the answer.
- The complexity of Davis-Putnam algorithm in worst case is O(n*m^2) with n is the number of symbols, m is number of clauses.

In small KB, PL_resolution is better.

In large KB, Davis-Putnam is much better.