EE1007
Introduction to Computer Science Laboratory

Final project: SAT

Andy, Yu-Guang Chen
Assistant Professor, Department of EE
National Central University
andyygchen@ee.ncu.edu.tw
Slides Credit: TA Meng-Syuan LI

-Version_1.0_20231224
-Version_1.1_20240106
-Version_1.2_20240108
-Version_1.3_20240109

SAT

- Boolean satisfiability problem(SAT)
  - **(A ˅ B) ˄ (¬B ˅ C ˅ ¬D)**
    - ˅, ˄, ¬: logic OR, AND, and NOT
    - Literal: A, B, ¬B
    - Clause: (A ˅ B)
- Goal
  - Finding an assignment of truth values to the variables in the Boolean formula
    - Ex: A = 1 (TRUE), B = 1 (TRUE), C = 1 (TRUE), D = 1(TRUE).

Problem description:

- Using the DPLL algorithm to solve the SAT problem

```
Algorithm DPLL
    Input: A set of clauses Φ.
    Output: A truth value indicating whether Φ is satisfiable.
```

```
function DPLL(Φ)
    // unit propagation:
    while there is a unit clause {l} in Φ do
        Φ ← unit-propagate(l, Φ);
    // pure literal elimination:
    while there is a literal l that occurs pure in Φ do
        Φ ← pure-literal-assign(l, Φ);
    // stopping conditions:
    if Φ is empty then
        return true;
    if Φ contains an empty clause then
        return false;
    // DPLL procedure:
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {¬l});
```


Problem description:

- Using the DPLL algorithm to solve the SAT problem

```
Algorithm DPLL
    Input: A set of clauses Φ.
    Output: A truth value indicating whether Φ is satisfiable.
```

```
function DPLL(Φ)
    // unit propagation:
    while there is a unit clause {l} in Φ do
        Φ ← unit-propagate(l, Φ);
    // pure literal elimination:
    while there is a literal l that occurs pure in Φ do
        Φ ← pure-literal-assign(l, Φ);
    // stopping conditions:
    if Φ is empty then
        return true;
    if Φ contains an empty clause then
        return false;
    // DPLL procedure:
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {¬l});
```

Problem description:

- DPLL algorithm
  - Unit propagation: When a clause contains only one unassigned literal L, that L must be set to TRUE to satisfy the clause
    - Example: (A) ^ (A ˅ B ˅ D) ^ (¬A ˅ B) ➜ (B)
  - Pure Literal Assign: When a variable appears in all clauses in only one form, either as 'x' or 'x', then this variable can be set to either True or False.
    - Example: (A) ^ (A ˅ B) ^ (¬A ˅ B ˅ C) ➜ (A) ^ (A ˅ B)

Problem description:

- Using the DPLL algorithm to solve the SAT problem

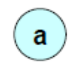
```
Algorithm DPLL
    Input: A set of clauses Φ.
    Output: A truth value indicating whether Φ is satisfiable.
```

```
function DPLL(Φ)
    // unit propagation:
    while there is a unit clause {l} in Φ do
        Φ ← unit-propagate(l, Φ);
    // pure literal elimination:
    while there is a literal l that occurs pure in Φ do
        Φ ← pure-literal-assign(l, Φ);
    // stopping conditions:
    if Φ is empty then
        return true;
    if Φ contains an empty clause then
        return false;
    // DPLL procedure:
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {¬l});
```

Problem description:

◆Flowchart of DPLL algorithm

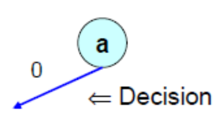
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)



Problem description:

◆Flowchart of DPLL algorithm

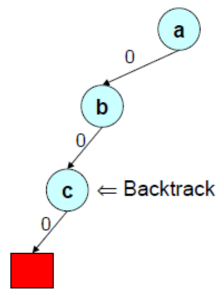
~~(¬a ∨ b ∨ c)~~
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
~~(¬a ∨ b ∨ ¬c)~~
~~(¬a ∨ ¬b ∨ c)~~



Problem description:

◆Flowchart of DPLL algorithm

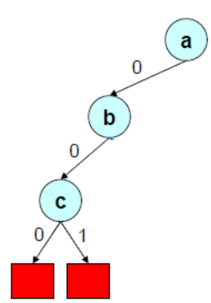
~~(¬a ∨ b ∨ c)~~
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
~~(a ∨ ¬c ∨ d)~~
~~(a ∨ ¬c ∨ ¬d)~~
~~(¬a ∨ b ∨ ¬c)~~
~~(¬a ∨ ¬b ∨ c)~~



Problem description:

◆Flowchart of DPLL algorithm

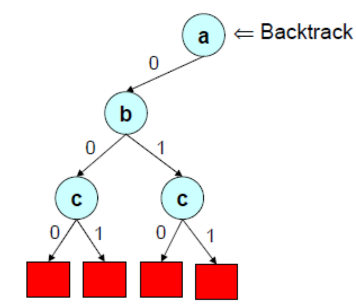
~~(¬a ∨ b ∨ c)~~
~~(a ∨ c ∨ d)~~
~~(a ∨ c ∨ ¬d)~~
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
~~(¬a ∨ b ∨ ¬c)~~
~~(¬a ∨ ¬b ∨ c)~~



Problem description:

◆Flowchart of DPLL algorithm

~~(¬a ∨ b ∨ c)~~
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
~~(¬a ∨ b ∨ ¬c)~~
~~(¬a ∨ ¬b ∨ c)~~

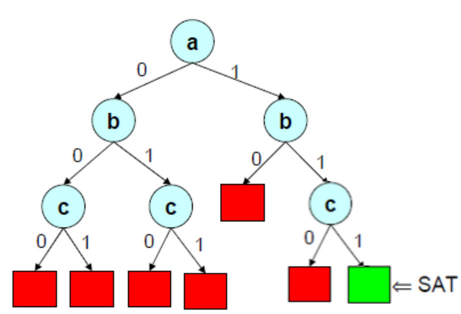


Problem description:

◆Flowchart of DPLL algorithm

➢ a=True, b=True, c=True, d=True or False

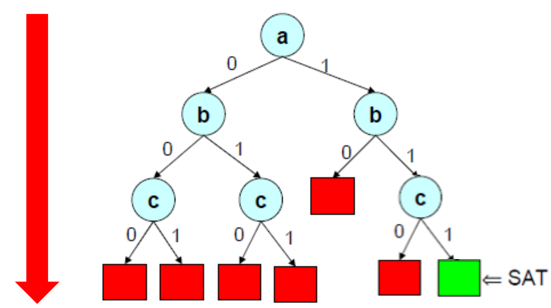
(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)



Problem description:

◆The rule for choosing literals is also referred to as the branch rule.

(¬a ∨ b ∨ c)
(a ∨ c ∨ d)
(a ∨ c ∨ ¬d)
(a ∨ ¬c ∨ d)
(a ∨ ¬c ∨ ¬d)
(¬a ∨ b ∨ ¬c)
(¬a ∨ ¬b ∨ c)



Problem Requirements

◆You need to implement the DPLL algorithm in code and use the Dynamic Largest Individual Sum (DLIS) as your branch rule

◆You must write a function to implement DLIS

◆Otherwise, the Correctness in your score will be 50% off.

$$w(F,u) = \sum_k d_k(F,u)$$

$$\Phi(x,y) = \max\{x,y\}$$

Input file format

- Example: testcase.cnf
  - (1 ˅ -3) ^ (2 ˅ 3 ˅ -1)

```
p cnf 3 2
1 -3 0
2 3 -1 0
%
0
```


Output file format

- Example: testcase.txt
  - The problem has a solution

```
s SATISFIABLE
v " 1 " -> True
v " 2 " -> False
v " 3 " -> True
Done
```

  - There is no solution exists

```
s UNSATISFIABLE
```

Execution program

◆Execute the program with the following command on the workstation

```
[ta112521034@cad ~]$ g++ -std=c++11 986253465_Final.cpp -o Final.out
[ta112521034@cad ~]$ ./Final.out testcase1.cnf testcase1.txt
```


Execution program

◆You can write like this in your program

◆The explanation of argc and argv

➢https://www.ibm.com/docs/en/i/7.1?topic=functions-main-function

```
#include <iostream>
#include <fstream>
using namespace std;
void input_file(char *, ...);
void output_file(char *, ...);

int main(int argc, char *argv[])
{
    //In Final Project you don,t need to cin filename;
    //store your filename by argc, argv when you execute your program
    //compile your .cpp by g++ -std=c++11 986253465.Final.cpp -o Final.out
    //when you execute your .out by: ./Final.out testcase1.cnf testcase1.txt
    //argv[0] represent ./Final.out
    //argv[1] represent testcase1.cnf(you need it!!)
    //argv[2] represent testcase1.txt(you need it!!)
    input_file(argv[1], ...);
    //...your algorithm
    //...
    //...
    output_file(argv[2], ...);

    return 0;
}
void input_file(char *argv, ...)
{
    ifstream infile(argv, ios::in);

    ...
}
void output_file(char *argv, ...)
{
    ofstream outputfile(argv, ios::out);

    ...
}
...
```

Report

- How to compile and execute your program
- The completion of the assignment
- The hardness of this assignment and how you overcome it
- Any suggestions about Final Project

➢ You can also put anything related to the Final project in your report, such as pseudocode, control flow diagram, programming developing thought, etc.

Correctness

- We will judge the correctness from the checker on the workstation

- The output file format has to be the same as Sample. Especially, the newlines, the white space and order of variables

Correctness

- There are 3 public test cases for you to test your program
- We will run all 5 cases within 30 minutes
- (50%) Correctness
  - 5 testcases, 10 points each case

Q&A

- If you have any questions about Final project, please send me an email or attend the TA office hours on Tuesday at 19:00 in room E1-359
- TA: Meng-Syuan Li
- Email: eji.nick302@gmail.com