SAT Project Report

B07901103 電機三 陳孟宏

1. Topic:

- Using SAT engine to solve the “Hamiltonian cycle” problem

2. Description:

- A Hamiltonian path is a path in an undirected or directed graph (**undirected here**) that visits each vertex exactly once.
- A Hamiltonian cycle is a Hamiltonian path that is a cycle.
- Here, we feed in any undirected graph, it can output:
  - (1) Numbers of learnt clauses and literals.
  - (2) Whether it’s a Hamiltonian cycle.
  - (3) If it is, print the path ; otherwise, print why it is not a Hamiltonian cycle. (Odd node / Multi-cycle).
  - (4) Operation time.

3. Frame / Input Format

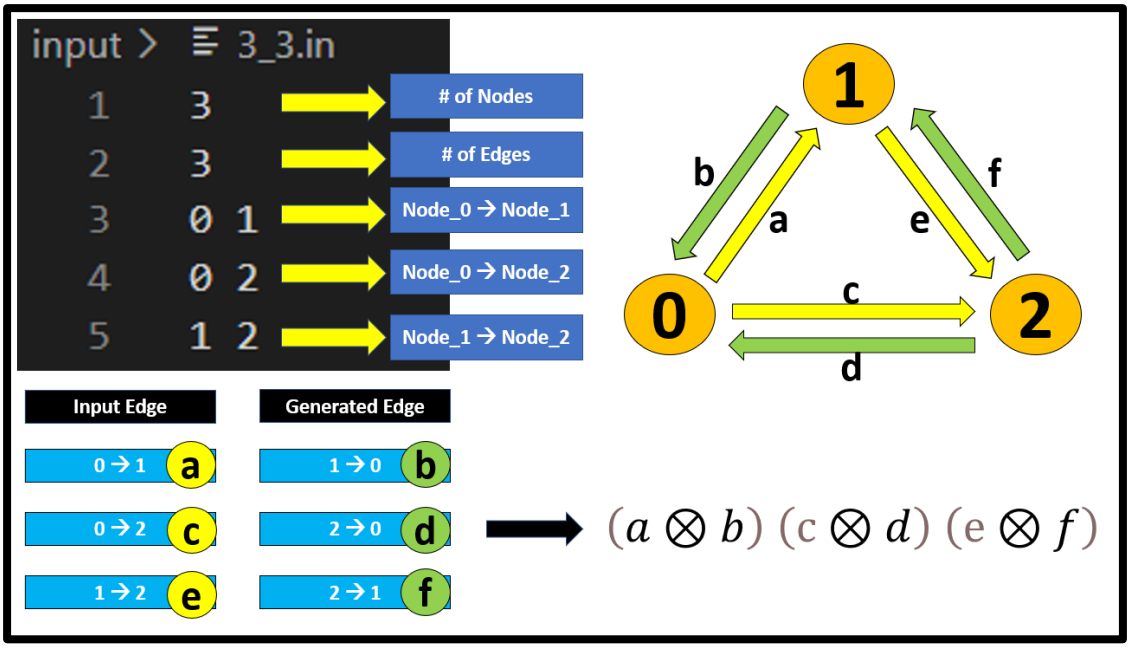
- ./input
  - Testcase (filename = {# of nodes}_{# of edges}.in)
  - Format
    ({# of nodes} \n {# of edges} \n {edges1: node -> node} \n …)
- ./src
  - main.cpp (execute this cpp file)
  - other sat .cpp/.h file
    (convenient to include or write in makefile)
- command line to execute my code (./Hamiltonian-cycle-with-SAT)
  - make
  - ./bin/ham_cycle_sat input/<input file> output/<output file>

4. Constraint:

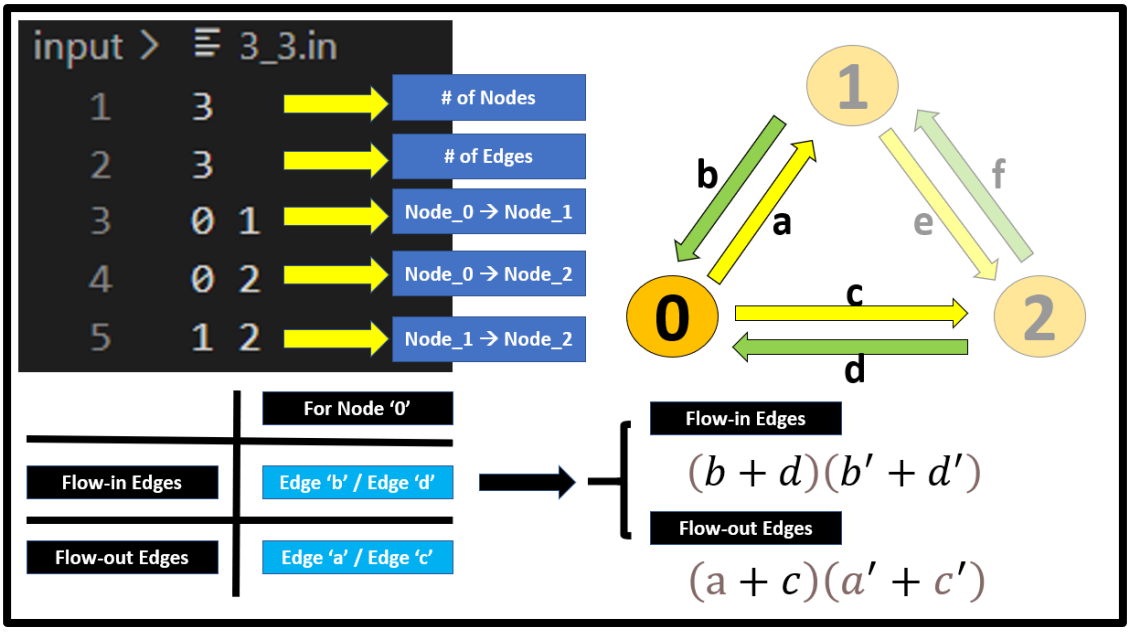
- **For an edge, it can choose only one direction to form the loop.**
  - Read in an edge, and then generate an edge with the other direction.
    (input: a → b, generate: b → a)
  - Assign a variable (integer) to the bi-directional edge, original

edge will be “gates[2*i]”, generated edge will be “gates[2*i+1]”.

- **XOR** them, and **AND all the XOR term**.



- **Consider each node, there are edges flowing into the node and out of the node, but we can only choose one edge to flow in, and one edge to flow out.**
  - Based on the above, we have already encoded every edge with a number.
  - Construct an “array of vector (vector<int> start[V_num], end[V_num])” with size of # of nodes → start[0] = <a,c > means “for node 0, there are edges ‘a’ and ‘c’ which is started from node 0”, and end[0] is the same.
  - For every flow out edge to a node (edge ‘a’ and ‘c’ for node_0), at least we need to choose one to be the Hamiltonian path → **OR every flow-out-edge**, e.g.(a+c).
  - For every flow out edge to a node (edge ‘a’ and ‘c’ for node_0), we can only choose one edge to be the Hamiltonian path, so there’s no any two of the flow-out-edge can exist at the same time → **OR every permutation of two inversed-flow-out-edge**, e.g.(a’+c’).
  - Same way for the flow-in-edge.



- **(Not a must) Every node should exist at start point and end point with even times totally.**
  - Check the original input edge, for each node, sum the existence of the node at start point and at end point, if there's a node whose result is "odd", then the graph cannot form a cycle → **UNSAT**
- **(Exclude the multi-cycle scenario) Multi-cycle can also satisfy the above constraints, but for the definition of "Hamiltonian cycle", it only forms one cycle in the graph. So we need to exclude the multi-cycle case.**
  - After finishing the above constraints, I will print the path if it is SAT, then I traverse the path from any random point, it must pass through every node if the path contains only a cycle. Hence, if there's at least one node that is not gone through, then it means the graph has multi-cycle → **UNSAT**

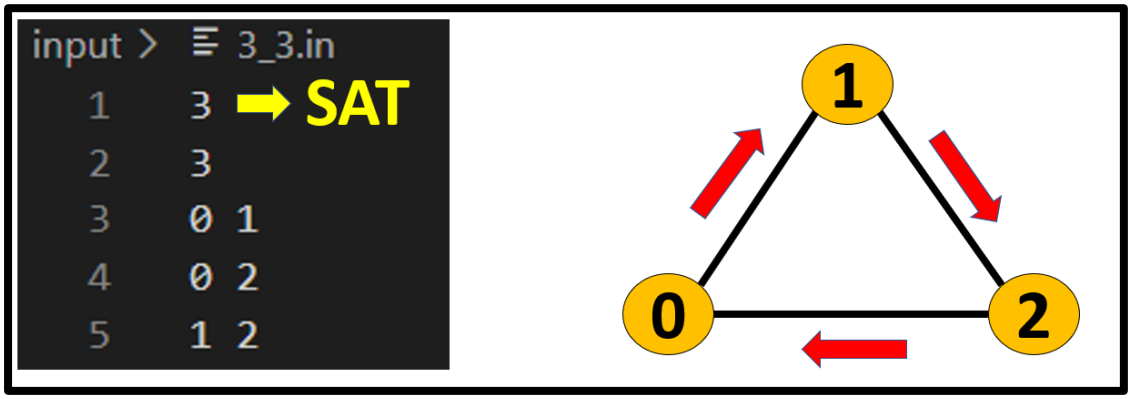
5. Time Complexity / Space Complexity:

- Time complexity = O(V^3)
  - This part can definitely be optimized, because SAT tool can only eat "2-input", and I do not modify to "n-input" there.
- Space complexity = O(V^3)
  - In reality, I do not use such huge memory, this is the worst worst case, that is, for each node, there are edges pointing to the other nodes. One of the constraints will generate "c_2^v" clauses for one node, hence the worst case might use

"$v \times C_2^v = v \times \frac{v(v-1)}{2}$" clauses.

6. Basic Testcase for Correctness:

- SAT
  - **3 nodes + 3 edges**




```
Success!!!!!!!!!! (even node)
================================[MINISAT]================================
| Conflicts |     ORIGINAL     |           LEARNT          | Progress |
|           | Clauses Literals | Clauses Literals   Lit/Cl |          |
=========================================================================
|         0 |       80      194 |       0        0     -nan |  0.000 % |
=========================================================================
```

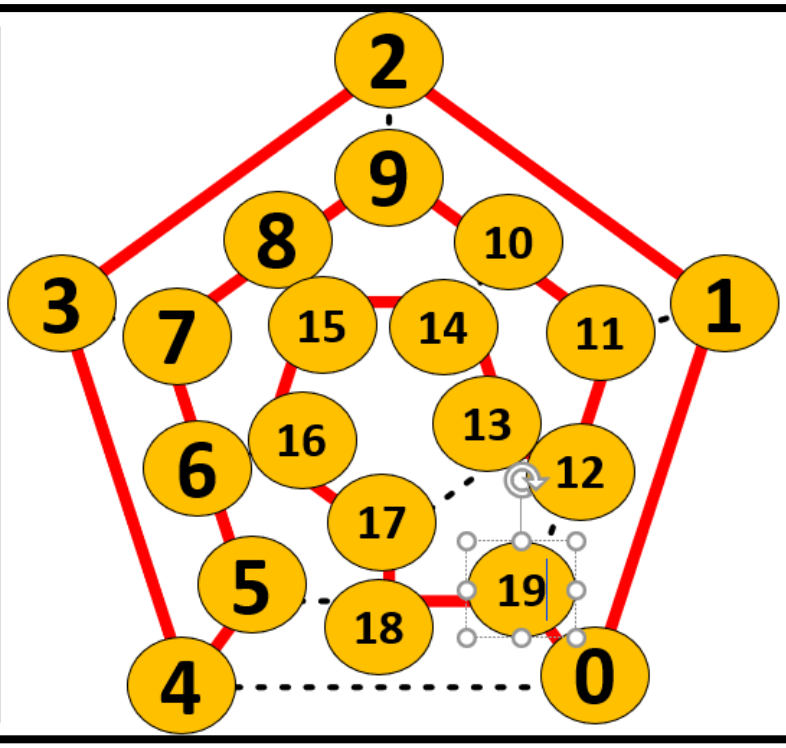
```
====
Path
====
0 -> 1
1 -> 2
2 -> 0

======
Result
======
SAT
Success!!! It's a Hamiltonian cycle !

====
Time
====
time = 0.00056
```

■ **20 nodes + 20 edges**

input > 20_20.in

```
20
20
0 1
2 1
2 3
4 3
4 5
6 5
6 7
8 7
8 9
10 9
10 11
12 11
12 13
14 13
14 15
16 15
16 17
18 17
18 19
0 19
```

SAT

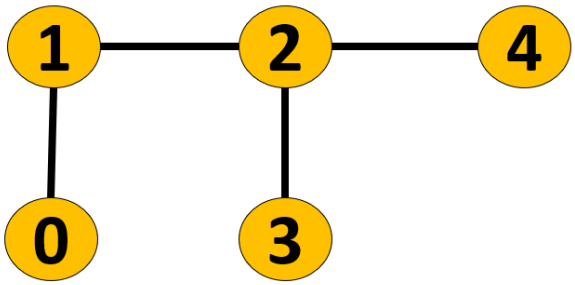



```
Success!!!!!!!!!! (even node)
================================[MINISAT]================================
| Conflicts |     ORIGINAL     |           LEARNT           | Progress |
|           | Clauses Literals | Clauses Literals   Lit/Cl |          |
=========================================================================
|         0 |      505     1231 |       0        0     -nan |  0.000 % |
=========================================================================
```

```
====
Path
====
1 -> 0
0 -> 19
19 -> 18
18 -> 17
17 -> 16
16 -> 15
15 -> 14
14 -> 13
13 -> 12
12 -> 11
11 -> 10
10 -> 9
9 -> 8
8 -> 7
7 -> 6
6 -> 5
5 -> 4
4 -> 3
3 -> 2
2 -> 1

======
Result
======
SAT
Success!!! It's a Hamiltonian cycle !

====
Time
====
time = 0.007894
```

- **UNSAT**
  - **5 nodes + 4 edges (odd node)**

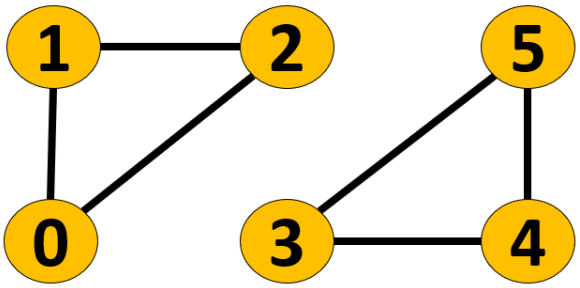
```
input > 5_4.in
1   5 UNSAT
2   4
3   0 1
4   1 2
5   2 4
6   2 3
```




```
Failed.........(odd node)
================================[MINISAT]================================
| Conflicts |     ORIGINAL     |          LEARNT          | Progress |
|           | Clauses Literals | Clauses Literals   Lit/Cl |          |
=========================================================================
|         0 |     129      307 |       0        0     -nan |  0.000 % |
=========================================================================
======
Result
======
UNSAT (odd node)
Failed......It's not a Hamiltonian cycle !

====
Time
====
time = 0.000668
```

  - **6 nodes + 6 edges (multi-cycle)**

```
input > 6_6.in
1   6 UNSAT
2   6
3   0 1
4   1 2
5   2 0
6   3 4
7   4 5
8   5 3
```

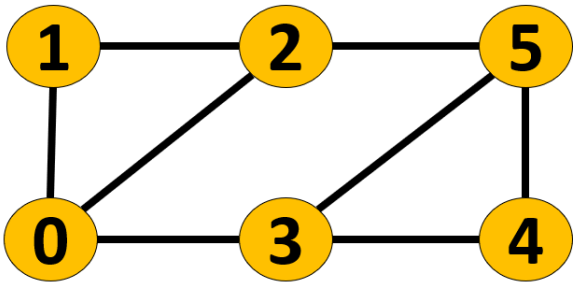


```
Success!!!!!!!!!! (even node)
============================[MINISAT]============================
| Conflicts |     ORIGINAL     |          LEARNT          | Progress |
|           | Clauses Literals | Clauses Literals  Lit/Cl |          |
=================================================================
|         0 |      155      377 |       0        0     -nan |  0.000 % |
=================================================================
```

```
====
Path
====
0 -> 1
1 -> 2
2 -> 0

======
Result
======
UNSAT (multi-cycle)
Failed......It's not a Hamiltonian cycle !

====
Time
====
time = 0.000881
```

- **6 nodes + 8 edges (multi-cycle)**

```
input > 6_8.in
   1    6 UNSAT
   2    8
   3    0 1
   4    1 2
   5    2 5
   6    5 4
   7    4 3
   8    3 0
   9    0 2
  10    3 5
```



```
Failed.........(odd node)
============================[MINISAT]============================
| Conflicts |     ORIGINAL     |           LEARNT          | Progress |
|           | Clauses Literals | Clauses Literals  Lit/Cl |          |
=================================================================
|         0 |     289      695 |       0        0    -nan |  0.000 % |
=================================================================
======
Result
======
UNSAT (odd node)
Failed......It's not a Hamiltonian cycle !

====
Time
====
time = 0.000828
```

7. Feedback:

這次實作 Hamiltonian cycle，並且用 SAT tool 去解，這過程我扎實地把一個 NPC 問題想過一遍，因為網路上沒人用 SAT 工具去解這個問題，正常 Ham cycle 的程式碼就是簡單幾行就可完成，不用大費周章用到 SAT 工具，所以我就針對好幾個憑空想像的極端 case 來找出 constraint 一定要包含哪些，怎麼把問題轉成 CNF 等等，加上時間上的壓力 (前面都沒日沒夜在搞光舞 QAQ)，能找出嚴謹的限制條件其實很有成就感!

更有成就感的是為了實現 SAT 解 CNF，我一直反覆爬 SAT tool 的程式，看懂他並會用真的很不容易，因為那時候 DSnP 我最後還沒寫到 Fraig 時間就到了，現在也算彌補當時不會用 SAT tool 的缺憾了!

每次的 meeting 就是一次的腦力激盪，聽別人不同創意的題目，或是被同學和教授提供一些更好的方向實作，優化我的專題真的都

收穫良多!再來就是好好的拚一下 CAD contest 了!加油!