**SAT Project Report**

B07901103 電機三 陳孟宏

1. **Topic:**

* Using SAT engine to solve the “**Eulerian graph (一筆畫圖形)**” problem

1. **Description:**

* An **Eulerian Trail** is a closed walk with no repeated edges but contains all edges of a graph *G*=(*V*(*G*),*E*(*G*)) and return to the start vertex. A graph with an Eulerian trail is considered Eulerian.

1. **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)

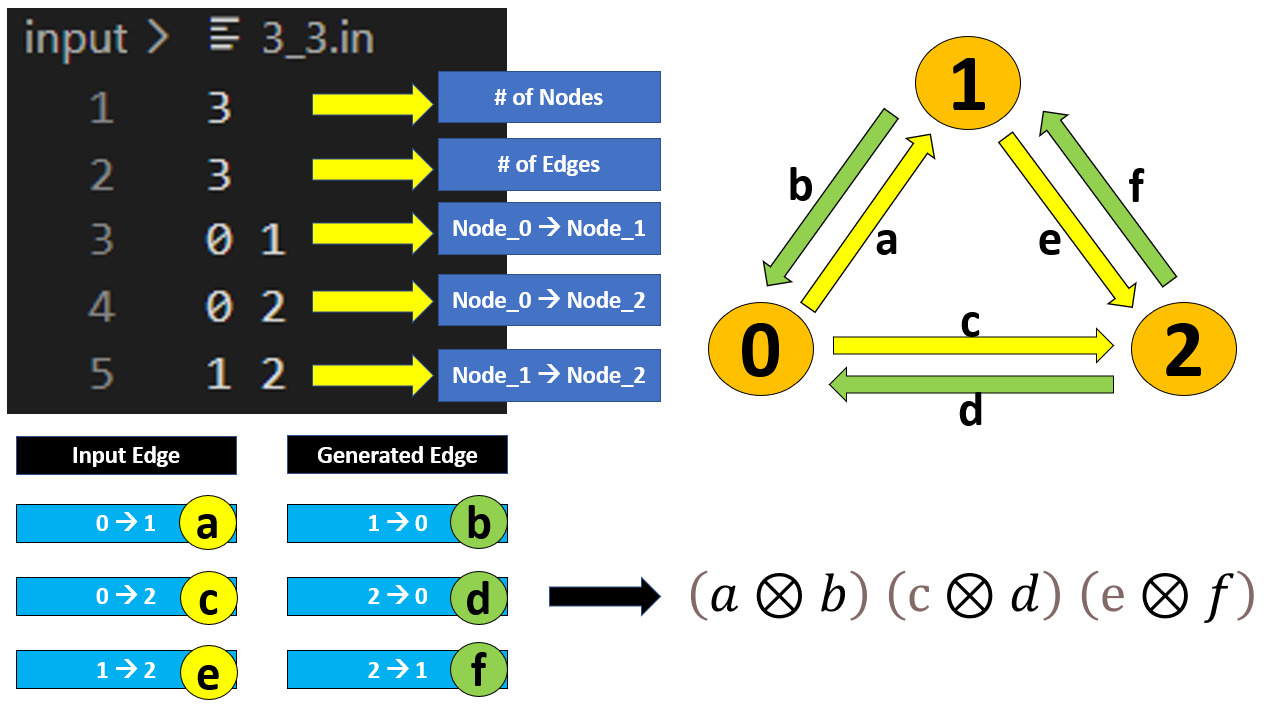
* command line to execute my code (./Hamiltonian-cycle-with-SAT)
  + make
  + ./bin/ham\_cycle\_sat input/<input file> output/<output file>

1. **Constraint:**

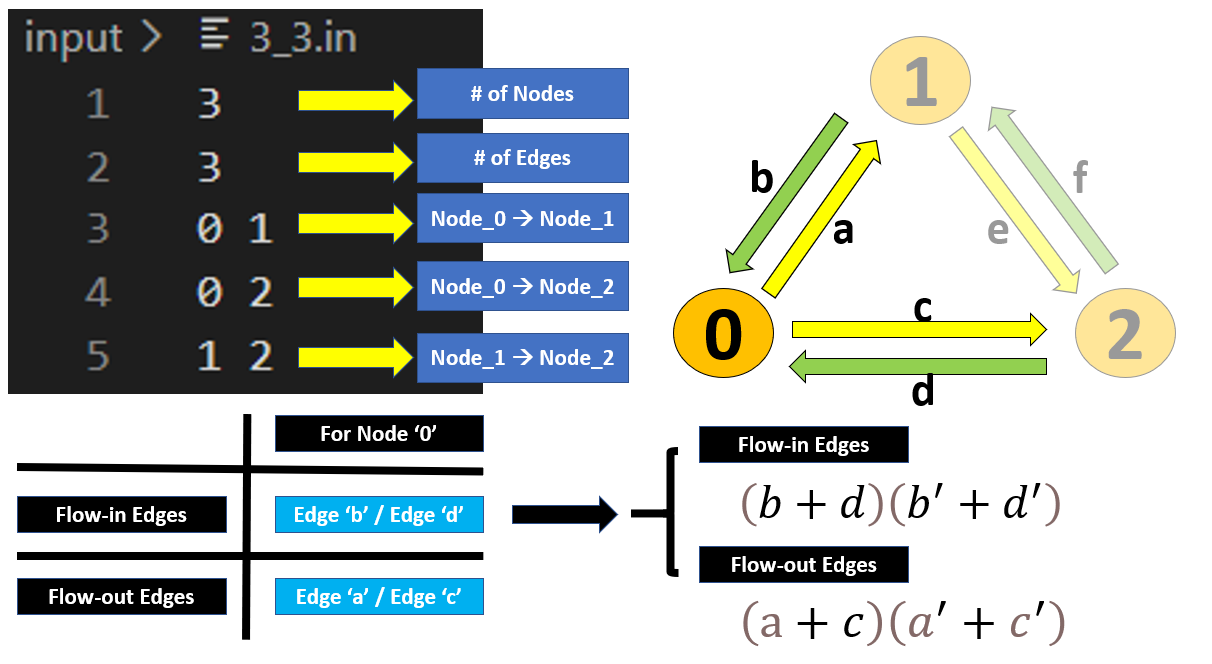
* **For an edge, it can choose only one direction to form the loop.**
  + Read in an edge, and then generate an edge with the other direction.

(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.

****

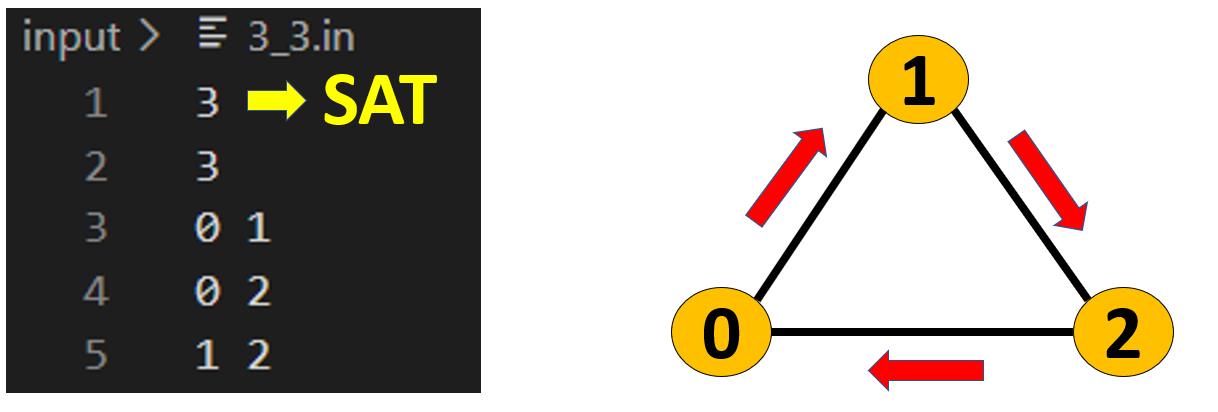
* **(Not a must) Every node should exist at start point and end point with even times totally.**
  + Check the original input edge, for each node, sum the existence of the node at start point and at end point, if there’s a node whose result is “odd”, then the graph cannot form a cycle 🡪 **UNSAT**
* **(Exclude the multi-cycle scenario) Multi-cycle can also satisfy the above constraints, but for the definition of “Eulerian graph”, 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**

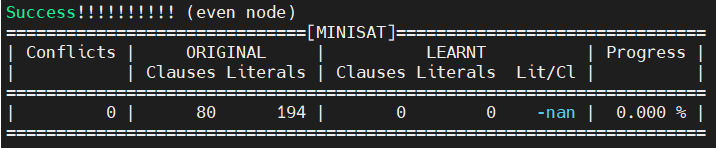
1. **Time Complexity / Space Complexity:**

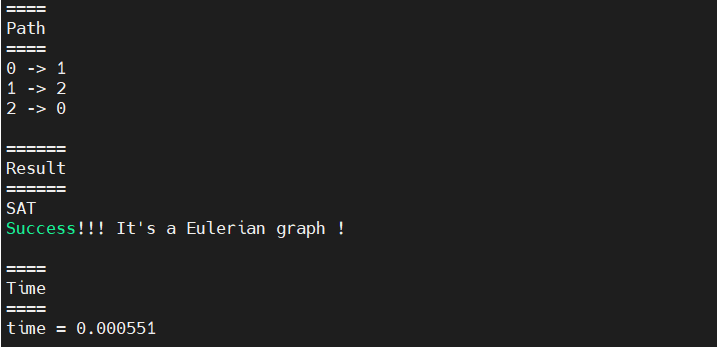
* Time complexity = O(V^3)
  + This part can definitely be optimized, because SAT tool can only eat “2-input”, and I do not modify to “n-input” there.
* Space complexity = O(V^3)
  + In reality, I do not use such huge memory, this is the worst worst case, that is, for each node, there are edges pointing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

1. **Basic Testcase for Correctness:**

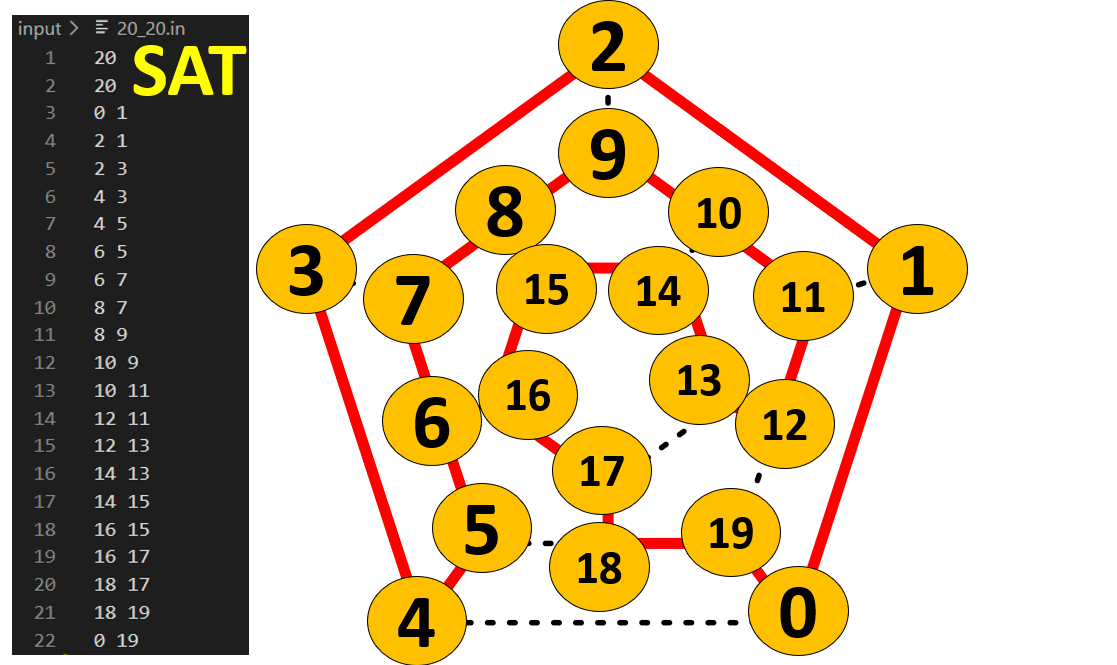
* **SAT**
  + **3 nodes + 3 edges**

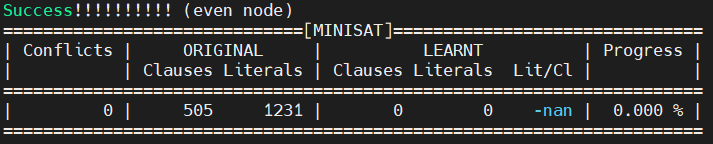


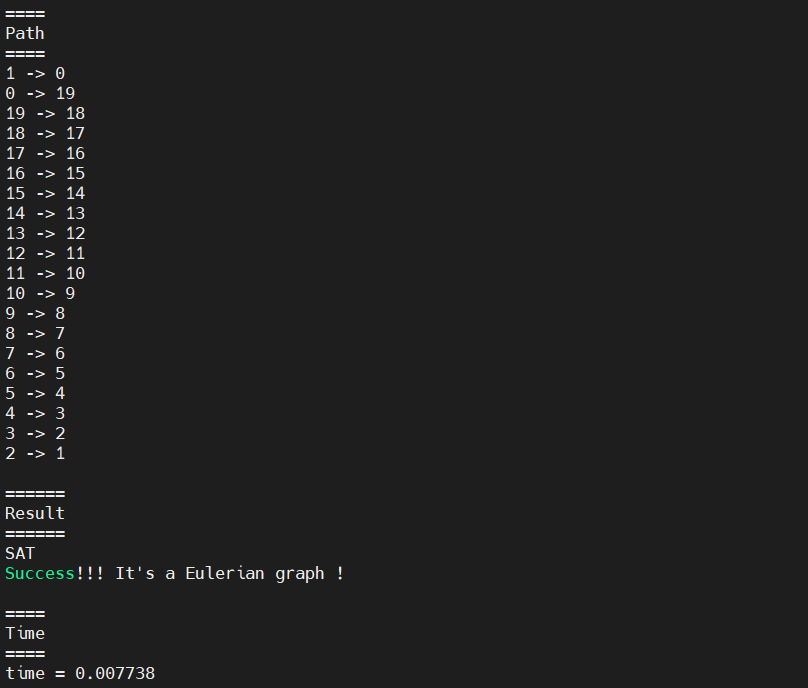


****

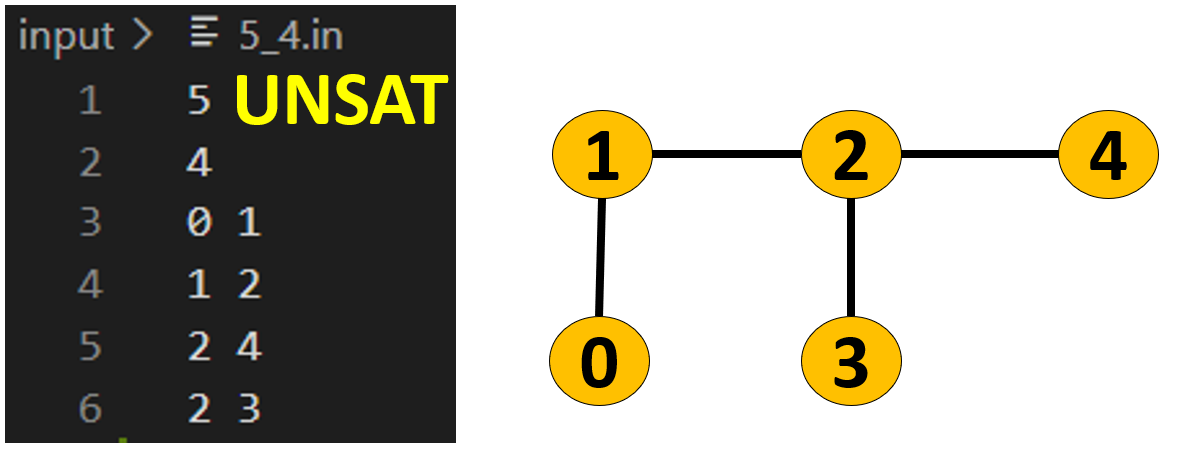
* + **20 nodes + 20 edges**

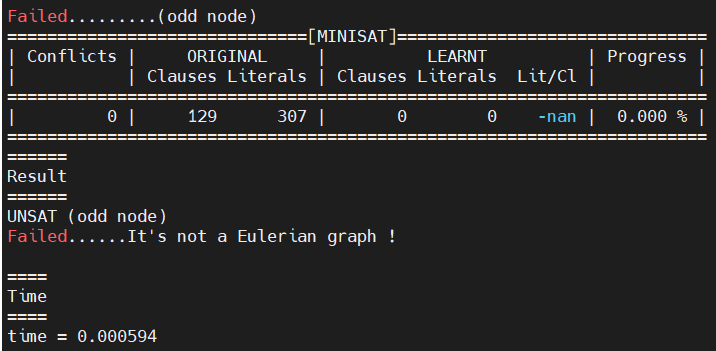
****



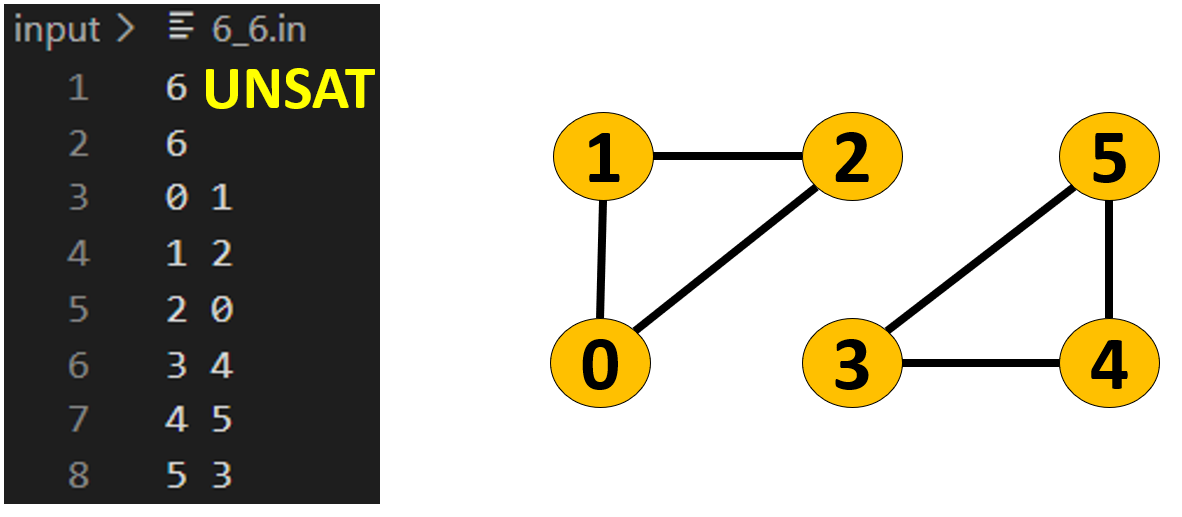
****

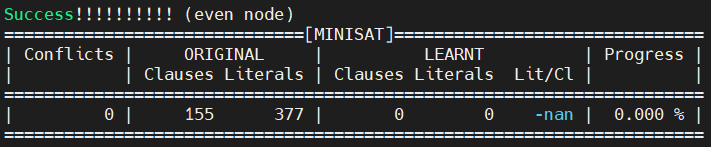
* **UNSAT**
  + **5 nodes + 4 edges (odd node)**

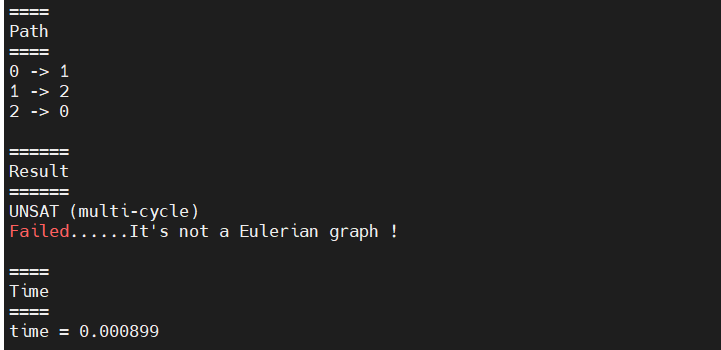


****

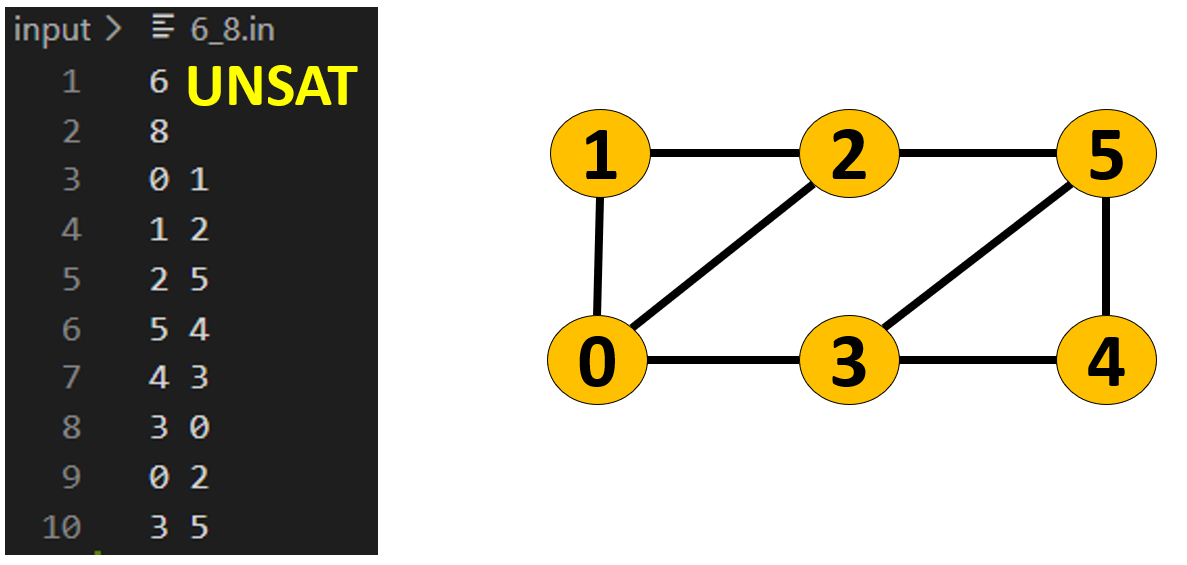
* + **6 nodes + 6 edges (multi-cycle)**

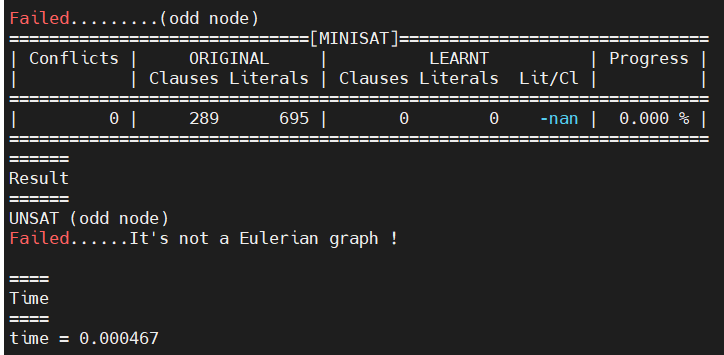




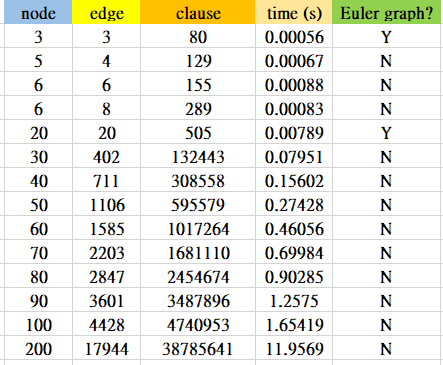
****

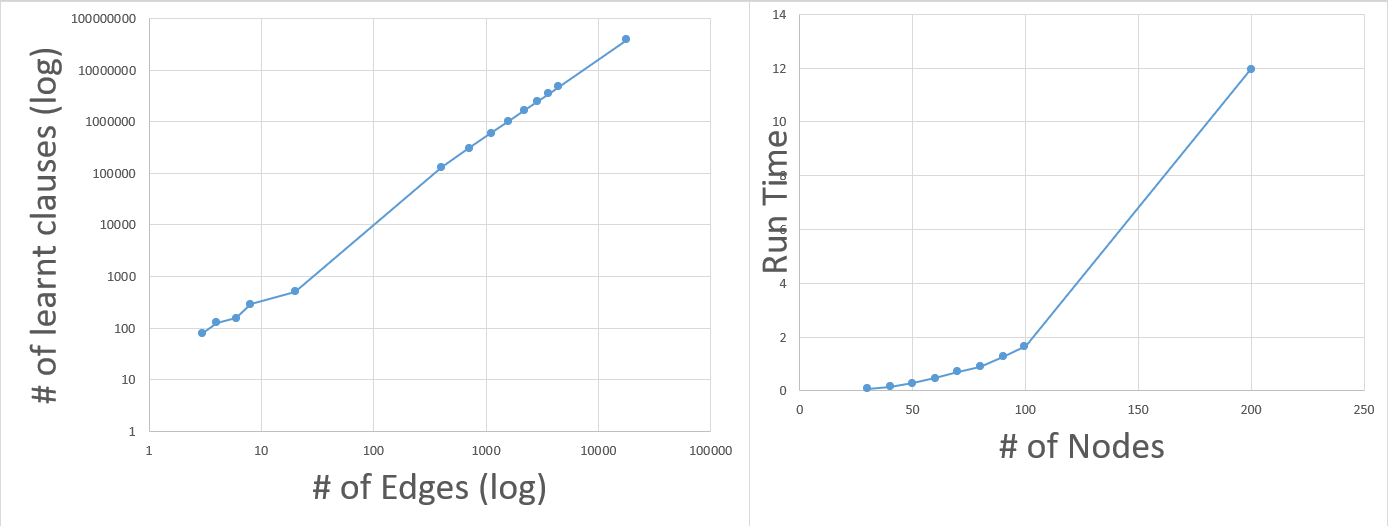
* + **6 nodes + 8 edges (multi-cycle)**



****

1. **Result:**

****

****

1. **Github Link:**

* <https://github.com/HHHUUUGGGOOO/Hamiltonian-cycle-with-SAT>

1. **Feedback:**

這次實作一筆畫圖形, 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, 正常一筆畫圖形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!