1.

Assume Pattern is P[1]···P[m]

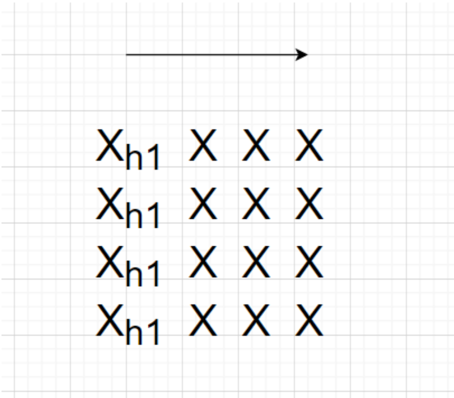
When you have a partial match T[i],T[i+1]···.T[j] and meet a non-matching char, it is impossible that the start of pattern (P[1]) is between T[i] and T[j] again since pattern characters are all **different**. // *(T[i]=P[1])*

So we don't need to backtrack from T[i+1], just iterate through the input and the pattern like common Naïve-String-Matching, when meet non-matching, reset pattern from P[1] ,keep current char pointer of text T.

2.

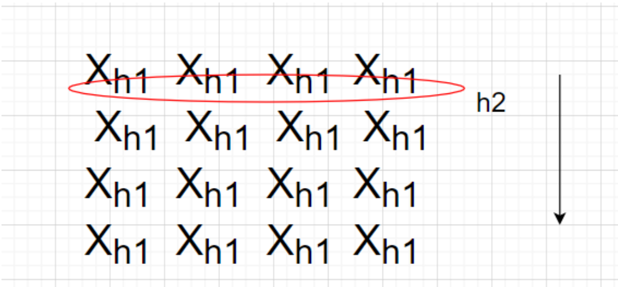
The idea is convert 2D to 1D

Preprocess:

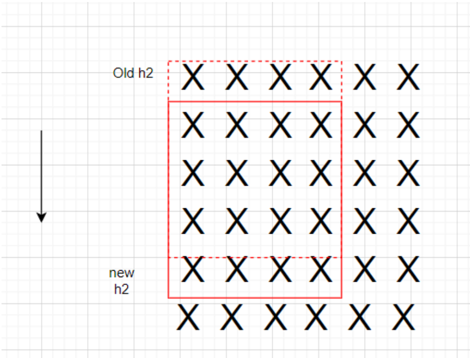


To every column ,calculate the hash of each char

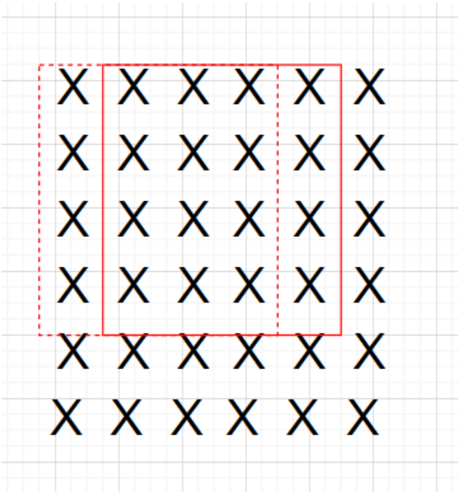
After loop all columns, every char should have a hashed number based on its column, then treat the column-hashed-number in one row as new char, hash it again. (Hash h1 in row instead of hash X in row because h1 has the data of column,)



Then loop the m*m text,

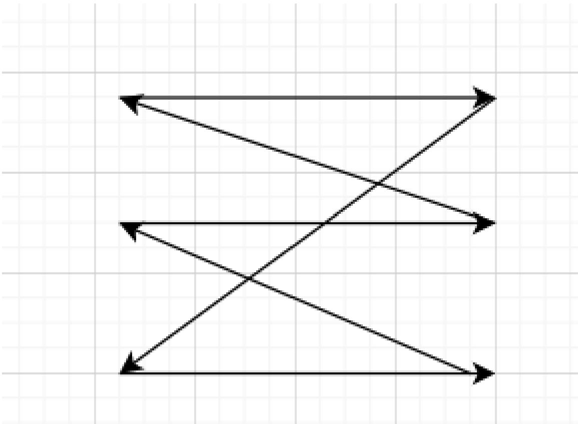


When move vertically, calculate column-hash number h1 of new row ,based on h1, calculate new h2, then total hash can be calculated by (d(total-oldh2*h)+new h2)%q



When move vertically, we need to recalculate every h2, but when calculate h2, it just need to remove old h1 in that row and add new h1 //d(h2-oldh1*h)+new h1)%q

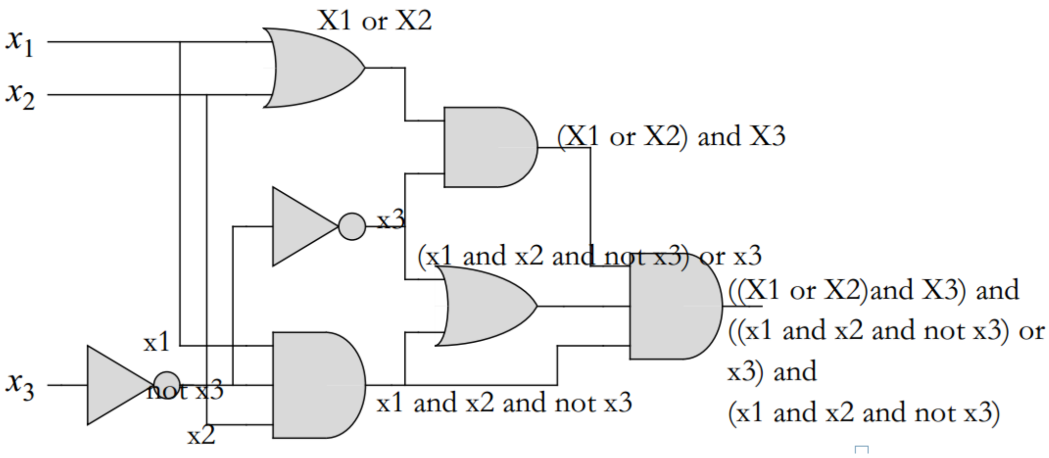
3.
if G is Hamiltonian, then every vertex of G should have indegree and outdegree=1.
Cause it is bipartite number of vertices, one side will have more nodes than other side. Assume one side A has m nodes, one side B has n nodes. Cause it is bipartite, the outdegree of one side = the indegree of the other side. However, A has m outdegree, B has n indegree, unmatched.
So it is non-Hamiltonian.


//should be undirected, the arriow is just used to show the cycle.

Only when the # vertices of both sides are equal, it is possible to have Hamiltonian.

4.
Cause the graph is directed and acyclic, just apply topological sort on the graph. Then if all the adjacent vertices in sorted graph are connected, it means that the Hamilton path exist

5.



The result should be

((X1 or X2) and X3) and ((x1 and x2 and not x3) or x3) and (x1 and x2 and not x3) //combine

=((X1 or X2) and X3) and ((x1 and x2 and not x3) or (x1 and x2 and not x3 and x3))

=((X1 or X2) and X3) and ((x1 and x2 and not x3) or (x1 and x2 and False))

=((X1 or X2) and X3) and ((x1 and x2 and not x3) or False)

= ((X1 or X2) and X3) and (x1 and x2 and not x3)

=(X1 or X2) and X3 and X1 and X2 and not X3

=(X1 or X2) and X1 and X2 and False

=False=0

So it is unsatisfiable.

6.

Step1.This problem is NP since given a graph G and potential path P, we can easily check whether P is a Hamiltonian Path in polynomial time (

Build a hashtable , vertices of G as key, default value is 0,

iterate the path , if the hashTable contains this node,

value++,

if value >1, return false,

else if the hashtable doesn't contain this node, return false).

Step2:

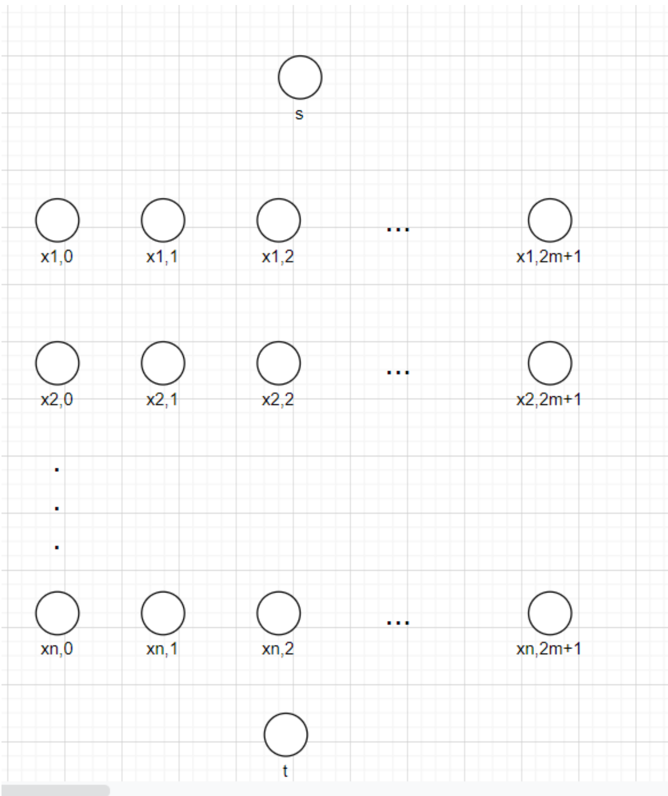
reduct 3SAT to Hamiltonian-Path problem.

need to show that given a Boolean expression E with k clauses, we can construct in polynomial-time a graph G such that E is satisfiable iff G has Hamiltonian-path

Variable: x1,x2…..xn

clause: C1,C2…cm

Step2.1: build n rows, row length =2m+1, //



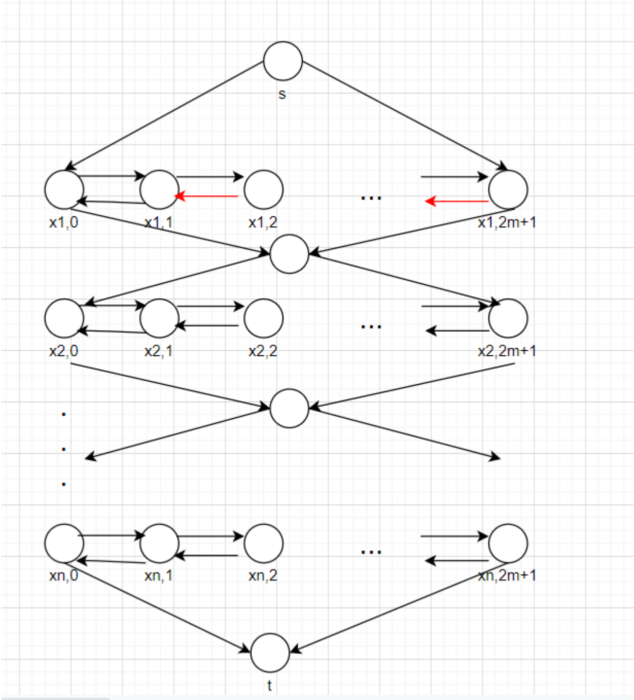
Step2,2, connect vertices

1/ Xi,0 to Xi,2m+1

2/ Xi,2m+1 to X0

3/S->x1,0->X2,2m+1->X3,0->…. // connect indirectly through an intersection point

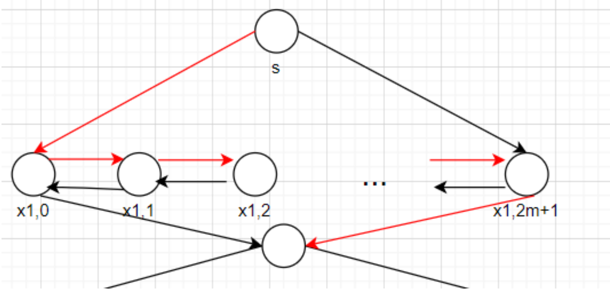
4/S->x1,m+1->X2,0->X3,2m+1…



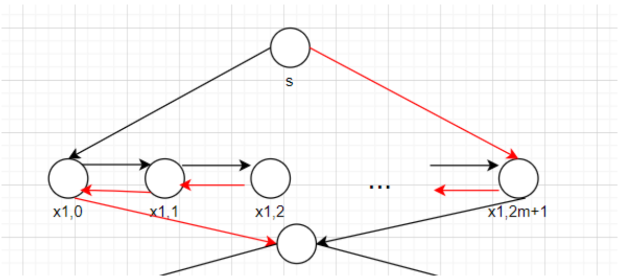
STEP2.3. Create vertices c1 to cm, corresponding to clauses, if Variable Xi is in clause Cj, then connect Xi,2j-1 to Cj, connect Cj to Xi,2j.
If not Xi in clause Cj,, then connect Cj to Xi,2j-1 , Xi,2j to Cj

The construction of graph is complete

If Xi is assigned true, go from top ->left->righ->bot



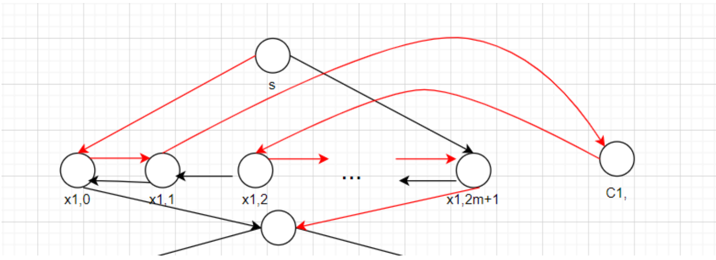
If Xi is assigned false, go from top->right->left->bot



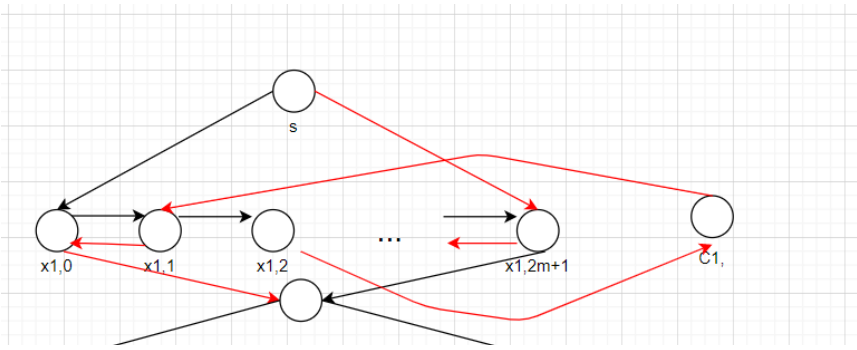
Then all nodes except C1 to Cm are visited exactly once.

Ci means Clause, to make 3SAT true, one variable Xi in every Cj should be true.

If this variable Xi is positive form in clause, then to make the clause true, Xi should be assigned true
we just disconnect Xi 2j-1 to Xi 2j, (the edge is still there, just ignore it), connect Xi,2j-1 to Cj, connect Cj to Xi,2j.



If this variable Xi is negative form in clause, then to make the clause true, Xi should be assigned false then negative Xi will be true,
we just disconnect Xi 2j to Xi 2j-1, (the edge is still there, just ignore it), connect Xi,2j to Cj, connect Cj to Xi,2j-1.



Though in clause maybe several variable may be true at the same time, we just use one of them, because we need to make sure Cj are also visited once.

Then all noded from s to t, X1,0 to Xn,2m+1, C1 to Cj are visited and only visited once, Then we build a Hamilton path in G , 3SAT is reduced into a Ham-path problem

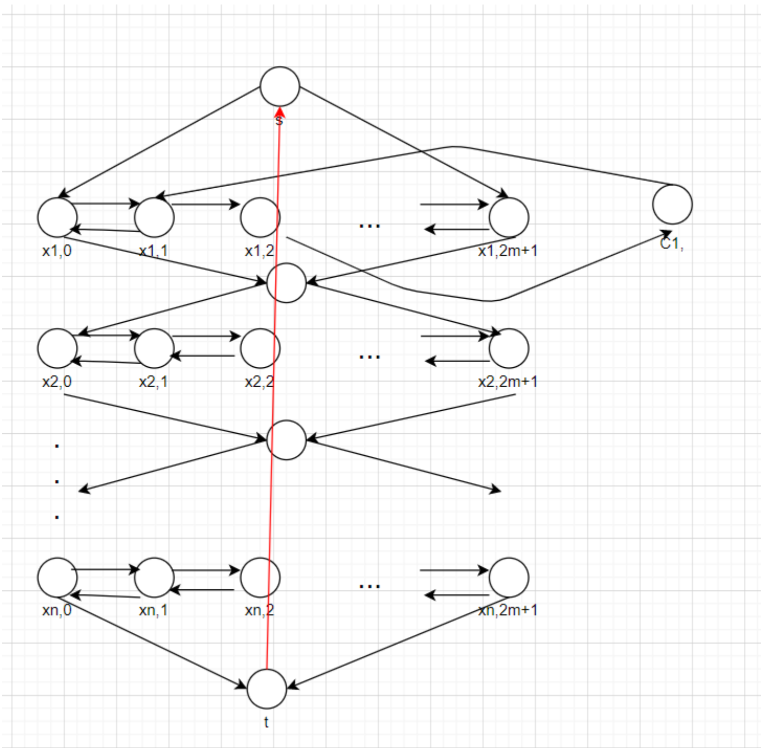
7.

Step1,

This problem is NP since given a graph G and the cycle C, we can check the repeated vertices like the Ham-Path way(question 6) in polynomial time. Then run DFS to find all connected parts in polynomial time, then we can check whether cycle C is biggest cycle.

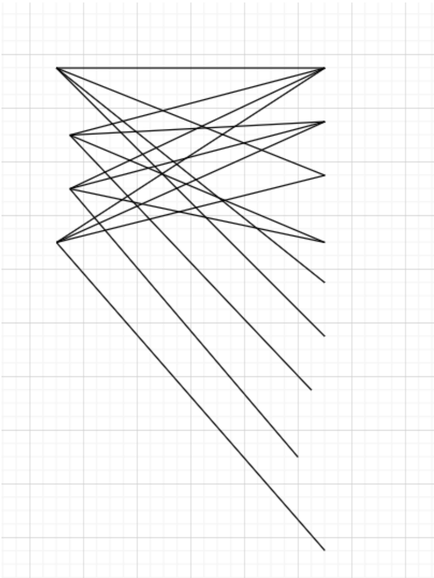
Step2:

The Hamiltonian path problem is a special case of longest-simple-cycle problem, we just need to connect the t to s back, then it is longest-simple-cycle in that graph, cause every node is visited, which shows that Hamiltonian path problem can be reduced to a longest-simple-cycle problem

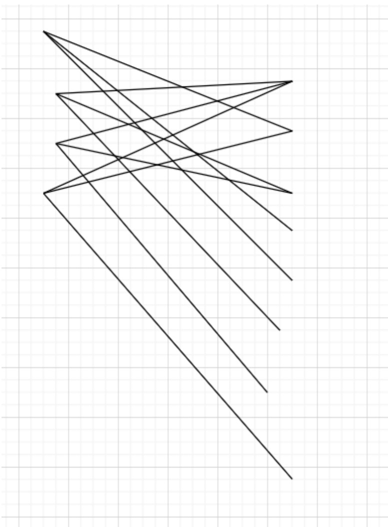


8.

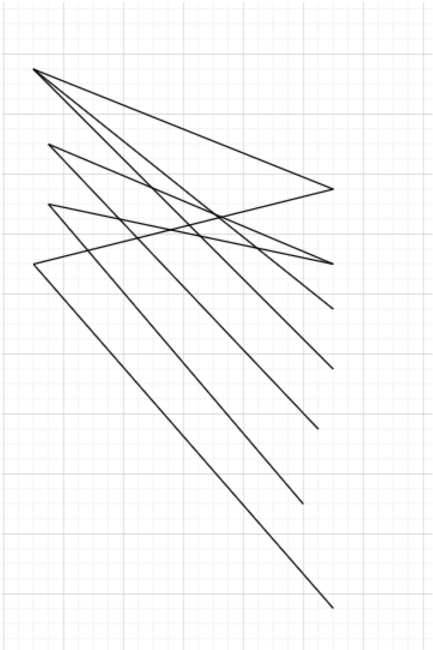
assume the vertices of left side has 4 vertices of degree(4,4,4,,4), right side has 9 vertices of degree(4,3,2,2,1,1,1,1) , the vertex-cover is size 4 //left side, approximation ratio =9/4>2



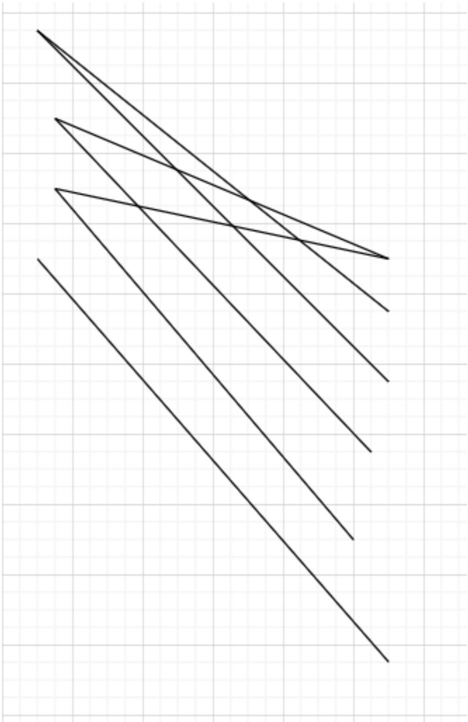
After choose first, left side degree 3 3 3 3



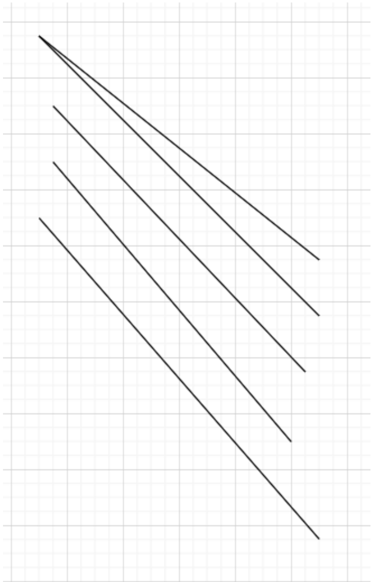
After choose second, left side degree 3 2 2 2



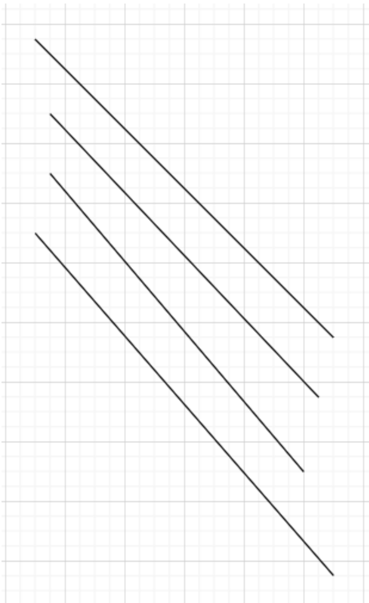
After choose third, left side degree 2 2 2 1



After choose fourth, left side degree 2 1 1 1



After choose fifth, left side degree 1 1 1 1



Still need to choose another 4 vertices